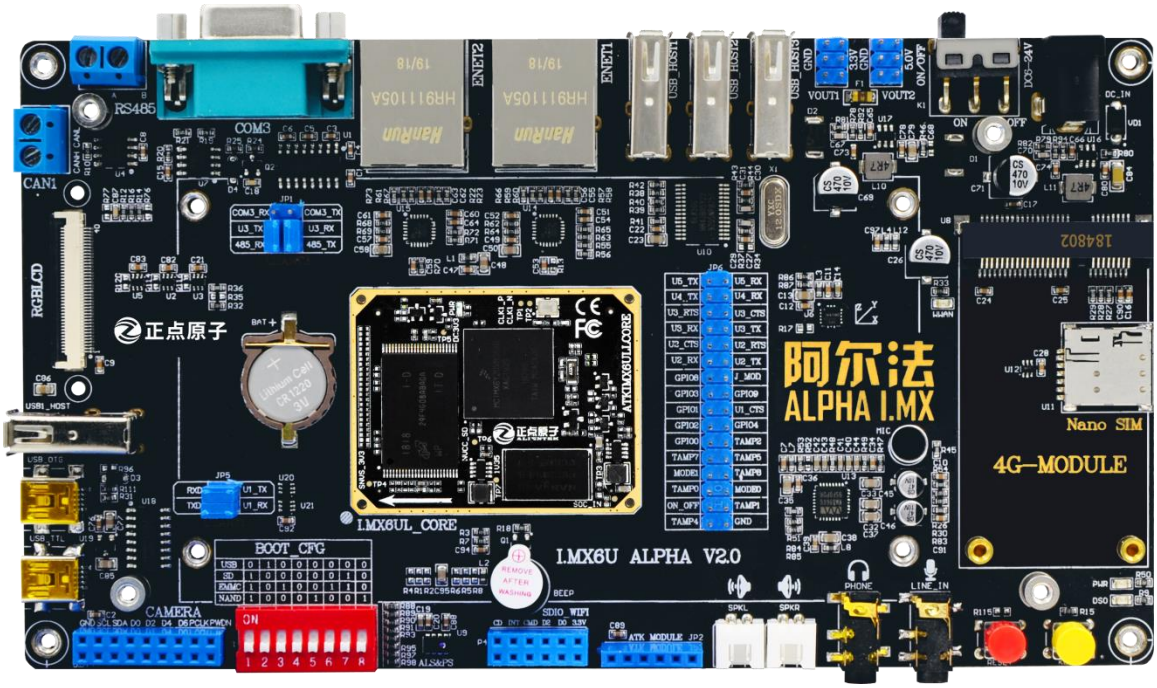
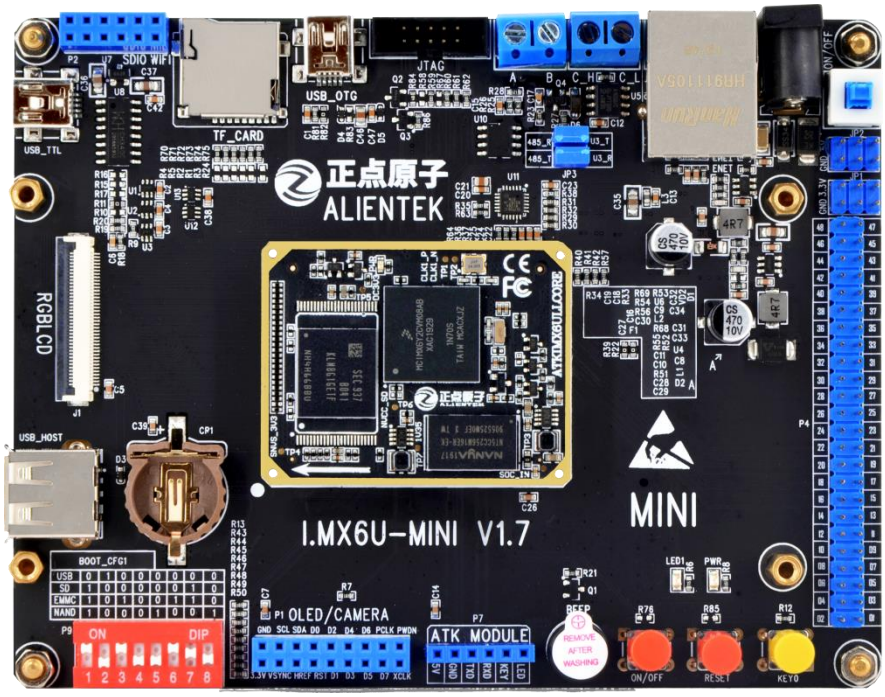
I.MX6U 嵌入式 Qt 开发指南

V1.1





正点原子

正点原子公司名称 ：广州市星翼电子科技有限公司
原子哥在线教学平台：www.yuanzige.com
开源电子网 / 论坛 ：http://www.openedv.com/forum.php
正点原子淘宝店铺 ：https://openedv.taobao.com
正点原子官方网站 ：www.alientek.com
正点原子 B 站视频 ：https://space.bilibili.com/394620890

电话：020-38271790 传真：020-36773971

请关注正点原子公众号，资料发布更新我们会通知。
请下载原子哥 APP，数千讲视频免费学习，更快更流畅。

扫码关注正点原子公众号

扫码下载“原子哥”APP

文档更新说明

版本	版本更新说明	负责人	校审	发布日期
V1.0	初稿：	邓志茂	左忠凯 陈梓雄 曾健拍	2021.6.18
V1.0.1	修正： 1.文档中的小错误。包括文字错误，病句，段落格式，代码小修正。代码请看 gitee 记录。	邓志茂	左忠凯 陈梓雄 曾健拍	2021.7.5
V1.0.2	修改： 1.第 7.1.5 和 7.2.1 小小节头文件源码不对应的问题。 2.第 13.2.1 小节，闹钟例程数据出现负数和删除闹钟不成功的小 bug。 3.第 24.1 小节，使用 esp8266 可能出现连接不上的问题。	邓志茂	左忠凯 陈梓雄 曾健拍	2021.7.28
V1.1	修改： 1. 第 7.8 小节和某些多余的文字 7.8.4 小节成员变量声明未使用的问题，7.9 小节控件描述错误，感谢 QQ 网友《九月阁》的反馈。 添加： 1. 第二十七章 车牌识别 2. 第二十八章 视频监控	邓志茂	左忠凯 陈梓雄 曾健拍	2021.11.22

目录

前言

本教程是面向初学者的基础教程（主讲偏嵌入式方向），于 2020.11.05 开始编写。从最简单的工程 helloworld 开始说起，一步一步引领大家从最基础的搭建，到简单的实例，以及使用常用的控件，可以说是手把手教学。内容比较精简，图文并茂。相当于引领读者上机练习。网上也有一些很好的 Qt 书籍，他们的例子比较复杂（不过讲解的比较好），往往一个例子有很多内容，也许对新手来说学起来比较吃力。特别是面向 Qt 初学者，力求把 Qt 开发中最常用的部件/组件加以例子说明展现在 Qt 初学者面前。使初学者快速掌握如何使用 Qt 最基本的操作，以及使用常用部件/控件。**如果您是要深入学习 Qt，这本教程并不适合您**。通过本教程的学习，一般能够在最短的时间内比较快速地去了解与运用Qt这门应用技术，及在嵌入式Qt上快速应用。**本教程重点不是讲解 Qt 提供的方法（函数）或者语法细节。重点是 Linux 的 Qt 环境部署，应用例子，使用控件/类，及如何在开发板上开发一些项目。**其中 ARM 板子的移植部署，交叉编译等正点原子已经有额外的文档说明，文档位于正点原子 I.MX6U 光盘资料的顶层目录下。顺便说一句，想要开发好看的界面，PS 基本技能需要懂一些，至少会一点点美工，因为等到熟练 Qt 后，开发过程会重点在界面上，而不是功能。

本教程主要以 Ubuntu 下开发为例，实际上在 Windows 下开发也一样的。这本教程的目的就是让读者快速实际上我们写的项目有可能还要放到开发板上运行，所以我们还是**统一使用 Ubuntu 来开发**。在 Ubuntu 下开发也并没有那么难，大家做好心理准备。

在初学 Qt 期间，我们也可以参考一些优秀的 Qt 学习网站或者博客，比如一去丶二三里的博客，或者多浏览 QTCN 开发网等网站，看看别人是怎么设计或者编写 Qt 应用程序，多借鉴或者参考。甚至去学习研究整个 Qt 源码，这样才有所提高。本教程旨在引领大家入门 Qt，内容精简，希望对大家有所帮助，水平有限，如有错误，请联系作者 QQ1252699831 指正，或者到正点原子论坛 www.openedv.com 发帖讨论，或发邮件到 dengzhimao@alientek.com，附上截图和文字说明，笔者会及时更新，谢谢大家。

本教程所使用的环境：

- Windows 7 64bits，也适用于 Windows 8-10。
- Ubuntu18.04。也支持 Ubuntu16，不过 Ubuntu16 会有些许不同。此文档基于 Ubuntu18.04 进行开发 Qt 讲解，尽量与笔者 Ubuntu 版本一样，少走弯路。本教程就是基于 Ubuntu18 来编写的。强烈建议与笔者一样使用 Ubuntu18 来开发，本教程开发的界面或者图标效果图都是在 Ubuntu18 上截图的，如果在其他版本的 Ubuntu 上有差异或者显示不出来，请自行查找解决方法。已知 7.4 小节及相关网络 http 请求的例子不能在 Ubuntu16 上正常运行。7.4 小节是显示不出菜单栏，网络相关是因为 Ubuntu16 版本的 Openssl 版本问题。
- 建议读者会使用 FileZilla、WinSCP 及 Windows Git 进行 Ubuntu 与 Windows 间互传文件的方法。
- Qt 版本为 Qt5.12.9。尽量与作者的版本一样，此 Qt 版本为长期支持版本。

- 本教程的源码已经上传到 coding（代码管理平台）上。找到第 12 个目录（Qt 开发指南例程源码）教程代码目录简洁，按篇即可找到例程所在的目录下。下载地址为 https://alientek-linux.coding.net/public/imx6ull/01_Soure_Code/git/files。

Ubuntu 18 上开发 Qt，本教程需要安装的工具软件总结，或者需要用到时再根据教程提示一步步安装。

基本工具，包括网络设置，文本编辑，版本控制工具。

```
sudo apt-get install net-tools openssh-server vim git
```

教程里实验需要安装的工具汇总。

```
sudo apt-get install gcc g++ lsb-core lib32stdc++6 libglu1-mesa-dev \
gstreamer1.0-plugins-base gstreamer1.0-plugins-bad gstreamer1.0-libav \
gstreamer1.0-plugins-good gstreamer1.0-plugins-ugly gstreamer1.0-pulseaudio \
cmake
```

第一篇 入门篇

很高兴与大家一起学习本教程的入门篇。入门篇的内容可大体分两部分。第一部分是 C++入门；第二部分是 Qt 基础入门。

第一部分 C++入门主要是针对没有学习过 C++的朋友们而写。当然 C++不可能就那几十页就写完的啦，重要是我们理解 C++的编程概念。C++的概念比较多。如果您对 C++十分了解，或者学习过，请直接跳过 C++学习部分。了解 C++对学习 Qt 来就比较简单啦，如果这部分 C++基础刚踏足的同志们，记得多看看！因为 C++基础部分都是 C++语言写的啊！

第二部分 Qt 基础入门主要是与大家一起学习 Qt 的基础部分。主要是环境的搭建和一些设置，快速熟悉项目的创建和编译。最后到学习 Qt 控件部分。同理有 Qt 基础的朋友们也可以跳过这部分的内容啦！不过某些环境配置的可以建议看一看。再次强调**教程是基于 Ubuntu 环境开发**，后面有些内容用 Ubuntu 开发比较方便，别习惯了 Windows，到时用 Ubuntu 就难了啦！和作者的环境一样遇到问题也好解决！最后祝大家学习愉快！

第一章 在 Ubuntu 下编写 C++

在 Ubuntu 上面编写 C++，本章节内容主要介绍在 Ubuntu 在终端窗口下使用 vi/vim 编辑一个 C++源文件。通过编写最简单的示例“Hello,World!”。带领大家学习如何在 Ubuntu 终端下编辑和编译 C++。这里要求大家会在 Ubuntu 上使用 vi/vim，也就是要求大家有一点 Ubuntu 入门的基础。如果没有这些基础也是可以拷贝 C++的代码到 Windows 上使用像 Dev-C++这种轻量级 C/C++ 集成开发环境（IDE）进行编写和编译。

但是笔者还是希望大家和笔者一起学习在 Ubuntu 下编写 C++，因为后面第二章的内容都是在 Ubuntu 下编写和讲解 C++的基础。同时也可以锻炼在 Linux 开发 C++的能力！

1.1 C++简介

C++ （c plus plus）是一种静态类型的、编译式的、通用的、大小写敏感的、不规则的编程语言，支持过程化编程、面向对象编程和泛型编程。C++ 被认为是一种中级语言，它综合了高级语言和低级语言的特点。C++ 是由 Bjarne Stroustrup 于 1979 年在新泽西州美利山贝尔实验室开始设计开发的。C++ 进一步扩充和完善了 C 语言，最初命名为带类的 C，后来在 1983 年更名为 C++。C++ 是 C 的一个超集，事实上，任何合法的 C 程序都是合法的 C++ 程序。截止 2020 年，在 2017 年发布 C++17，已经是第五个 C++标准了。我们也见过或者听过 C++98，这样的 C++标准，也就是 1998 年发布的 C++，所以叫 C++98，是 C++的第一个标准。

学习 C++我们要理解概念，而不是深究语言技术细节。我们只要带着第二章的 C++基础概念，学习 Qt 或者写 C++会有一定的帮助。

1.2 C++环境设置

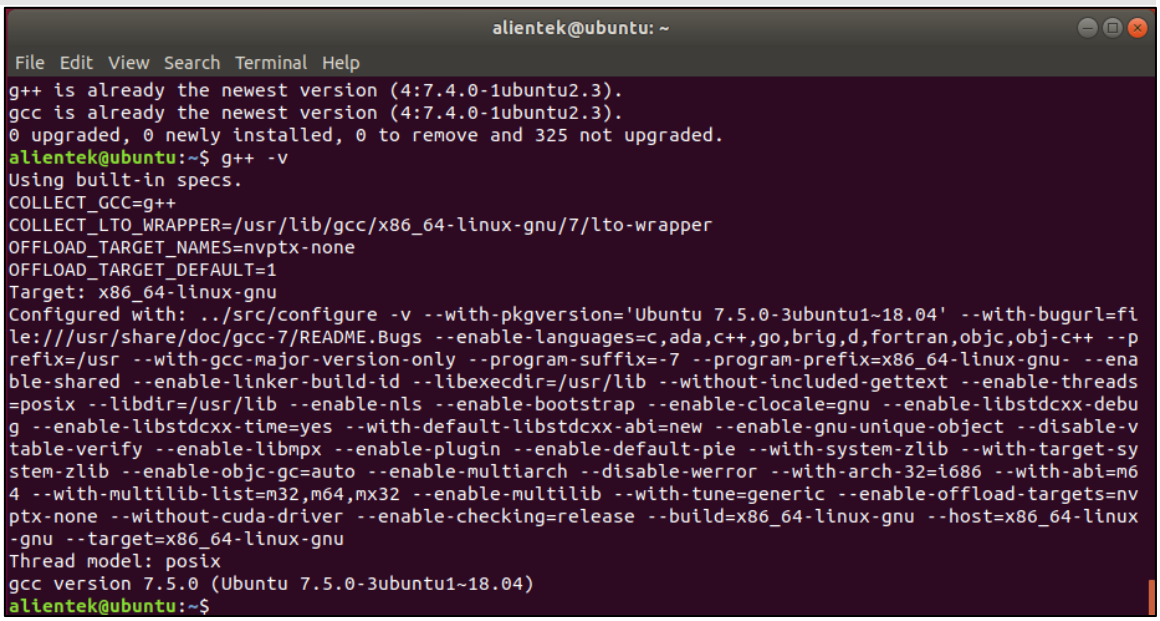
为了写这份教程，作者也是从新装了一个 Ubuntu18.04。从头搭建环境。我们先配置软件源的服务器地址为阿里云的地址。这样我们可以从国内去获取软件源，下载速度会更快。

我们要在 Ubuntu 编写 C++程序，那么需要有能编写代码的文本编辑和 C++编译器。在新装的 Ubuntu 环境里，编译 C 语言的 GCC 没有安装，编译 C++的 G++也没有安装。执行下面的执指令安装编译 C 语言和 C++的环境。

```
sudo apt-get install gcc g++
sudo apt-get install lsb-core lib32stdc++6 // 安装其他库
```

安装完成后，可以使用下面的指令来查看安装的 gcc 和 g++的版本。

```
g++ -v
gcc -v
```



1.3 编写一个简单的 C++程序

在终端输入下面的指令，首先我们创建一个 C++目录，然后使用 cd 指令进入 C++目录。再创建 01_hello_world 目录，进入 01_hello_world 目录，然后使用 vi 指令编辑 01_hello_world.cpp。

```
mkdir C++              // 创建一个 C++目录。
cd C++                 // 进入创建的 C++目录。
mkdir 01_hello_world // 创建一个 01_hello_world 目录
cd 01_hello_world  // 进入 01_hello_world 目录下。
vi 01_hello_world.cpp   // 编辑 cpp 文件，拷贝下文的内容
```

```
alientek@ubuntu:~$ mkdir C++
alientek@ubuntu:~$ cd C++
alientek@ubuntu:~/C++$ mkdir 01_hello_world
alientek@ubuntu:~/C++$ cd 01_hello_world
alientek@ubuntu:~/C++/01_hello_world$ vi 01_hello_world.cpp
```

拷贝下面的内容到 01_hello_world.cpp。

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      cout << "Hello, World!" << endl;
6      return 0;
7  }
```

第 1 行，C++ 语言定义了一些头文件，这些头文件包含了程序中必需的或有用的信息。上面这段程序中，包含了头文件 <iostream>。

第 2 行，using namespace std; 告诉编译器使用 std 命名空间。命名空间是 C++ 中一个相对新的概念。其中 std 就是 C++里的标准命名空间，也就是标准库里写好的了，我们可以直接调用。

第 3 行，int main() 是主函数，程序从这里开始执行。

第 5 行，cout << "Hello World"<<endl; 会在屏幕上显示消息 "Hello World"并换行。“<<”是运算符，endl 是换行语句。

第 6 行，return 0; 终止 main( )函数，并向调用进程返回

执行下面的语句进行编译和运行这个简单的 C++程序。

```
g++ 01_hello_world.cpp -o 01_hello_world // 使用 g++编译。-o 后面加的是输出的目标文件。
./01_hello_world                         // 在终端下执行，打印"Hello, World!"并换行。
```

```
alientek@ubuntu:~/C++/01_hello_world$ g++ 01_hello_world.cpp -o 01_hello_world
alientek@ubuntu:~/C++/01_hello_world$ ./01_hello_world
Hello, World!
alientek@ubuntu:~/C++/01_hello_world$
```

我们可以拓展一下，如何输出多行。可以像下面一样无限加下去。其中我们发现打印了第一个 Hello,world!后也换行了，因为使用了“\n”。C++中可以使用 C 语言的语句，C++是 C 语言的超集。

```
1  #include <iostream>
2  using namespace std;
3  int main()
```

```
4   {
5       cout << "Hello, world!\n" << "Hello, world!"  << endl;
6       return 0;
7   }
```

第 5 行，我们在里面再加用“<<”插入运算符（重载运算符）再插入一句“Hello, world!”打印，这样终端上就打印了两行“Hello, world!”。

第二章 C++基础

在第二章 C++基础里，这里主要介绍概念为主，主要介绍 C++与 C 语言中常用的不同点，和一些新的变化。其中不会去说指针、数据类型、变量类型、判断和循环等这些知识，这些和 C 语言基本是一样使用的。我们主要学习 C++的面向对象编程，对学习 Qt 有很大的帮助，理解第 2.2 章节的概念很重要。Qt 里就能体现到 C++编程带来的优势和便处。就算没学过 C++，学习 Qt 也不会很难。写 C++基础这章，笔者已经把重要的概念写出来，但是实际上 C++的内容不止这么多，第二章是快餐式 C++入门，**主要是为了更好的理解 Qt 中的 C++语法，学习 Qt 时也方便理解其中的内容。**

2.1 C++语言新特性

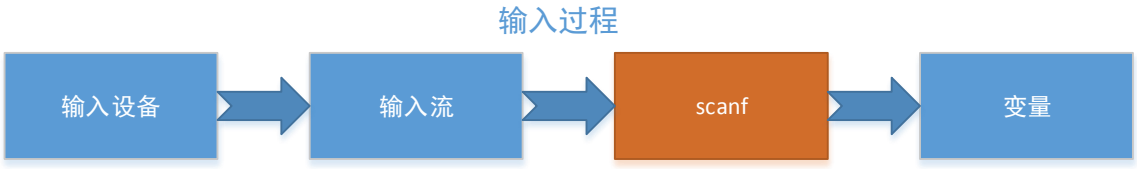
2.1 C++的新特性

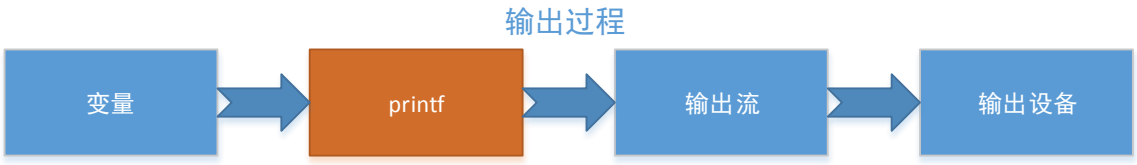
C++比 C 语言新增的数据类型是布尔类型（bool）。但是在新的 C 语言标准里已经有布尔类型了，但是在旧的 C 语言标准里是没有布尔类型的，编译器也无法解释布尔类型。

在传统的 C 语言里，变量初始化时必须在程序的前面定义在前面，而 C++则是可以随用随定义。C++也可以直接初始化，比如 int x(100);这样就直接赋值 x=100，这些都是 C++特性的好处。这里只说这些常用的新特性，其他特性不做描述或者解释了。

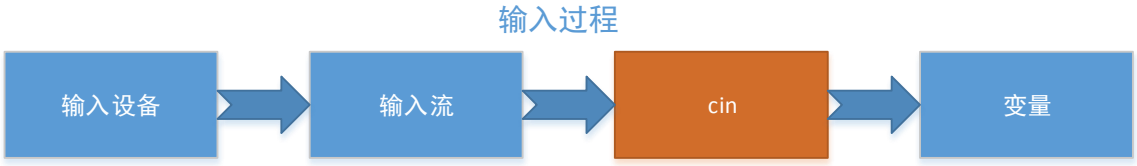
2.2 C++的输入输出方式

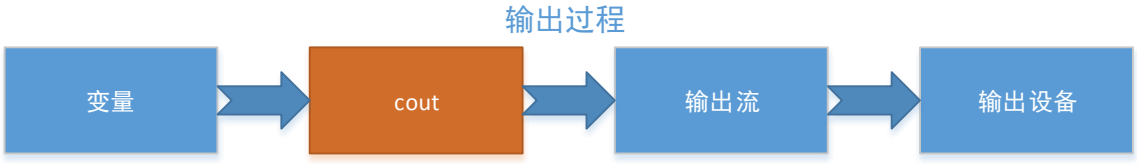
在 C 语言里，我们是这样输入或者输出的。





在 C++里，我们使用以 cin 和 cout 代替了 scanf 和 printf。在输入和输出的流程上是不变的，只是关键字变了，用法也变了。





要说效率上，肯定是 C 语言的 scanf 和 printf 的效率高，但是没有 C++中的 cin 和 cout 使用方便。

C++的 I/O 语法方式如下。

cout 语法形式：

```
cout<<x<<endl;
```

x 可以是任意数据类型，甚至可以写成一个表达式，这比 C 语言需要指定数据类型方便多了，endl 指的是换行符，与 C 语言的“\n”效果一样。

错误示例：

```
cout<<x,y<<endl;        // 在变量间不能用“,”。
```

正确写法：

```
cout<<x<<y;             // endl 可流省略，只是一个换行的效果。
```

cin 语法形式：

```
cin>>x;
```

x 可以是任意数据类型。

拓展，如何输入两个不同的变量。

```
cin>>x>>y;
```

2.3 C++之命名空间 namespace

在第 1.3 小节里我们已经使用过命名空间，如下代码第 2 行。using namespace std;同时我们要注意第 1 行，不能写成 iostream.h，有.h 的是非标准的输入输出流，c 的标准库。无.h 的是标准输入输出流就要用命名空间。

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      cout << "Hello, World!" << endl;
6      return 0;
7  }
```

using 是编译指令，声明当前命名空间的关键词。可以从字面上理解它的意思，using 翻译成使用。这样可以理解成使用命名空间 std。因为 cin 和 cout 都是属于 std 命名空间下的东西，所以使用时必须加上 using namespace std;这句话。cin 和 cout 可以写 std::cin 和 std::cout，“::”表示作用域，cin 和 cout 是属于 std 命名空间下的东西，这里可以理解成 std 的 cin 和 std 的 cout。

为什么要使用命名空间？

有些名字容易冲突，所以会使用命名空间的方式进行区分，具体来说就是加个前缀。比如 C++ 标准库里面定义了 vector 容器，您自己也写了个 vector 类，这样名字就冲突了。于是标准库里的名字都加上 std:: 的前缀，您必须用 std::vector 来引用。同理，您自己的类也可以加个自定义的前缀。但是经常写全名会很繁琐，所以在没有冲突的情况下您可以偷懒，写一句 using namespace std;，接下去的代码就可以不用写前缀直接写 vector 了。

从命名空间开始我们就隐隐约约可以看到 C++面向对象的影子了。命名空间在很多 C++库里使用到。有些公司也会自定义自己的 C++库，里面使用了大量的命名空间。从这里我们也可以看出 C++是非常之有条理的，容易管理的，不含糊，易使用的。

在初学 Qt 时我们是比较少使用命名空间，或者比较少看到命名空间。当然也是可以在 Qt 里自定义命名空间，然后与 C++一样正常使用。

下面通过一个简单的例子来介绍自定义的命名空间和使用自定义的命名空间。在 Ubuntu 上我们新建一个目录 02_namespace_example，然后在 02_namespace_example 里新建一个 02_namespace_example.cpp 文件，内容如下。

```
1  #include <iostream>
2  using namespace std;
3
4  namespace A
5  {
6      int x = 1;
7      void fun() {
8          cout<<"A namespace"<<endl;
9      }
10 }
11 using namespace A;
12 int main()
13 {
14     fun();
15     A::x = 3;
16     cout<<A::x<<endl;
17     A::fun();
18     return 0;
19 }
```

第 4 行，自定义了命名空间 A，里面定义了一个变量 x，并将 x 赋值为 1；定义了一个函数 fun()，并在 fun()加了输出打印语句 `cout<<"A namespace"<<endl;`。

第 11 行，声明使用命名空间 A。

第 14 行，在第 11 行声明了命名空间 A 后，才能直接使用 fun();否则要写成 A::fun();

第 15 行，将 A 命名空间下的 x 重新赋值为 3。

第 16 行，打印出 A 命名空间下的 x 的值。

第 17 行，调用 A 命名空间下的 fun()。

执行下面的指令开始编译。

```
g++ 02_namespace_example.cpp -o 02_namespace_example
```

编译完成执行的结果如下。

```
alientek@ubuntu:~/C++/02_namespace_example$ ./02_namespace_example
A namespace
3
A namespace
alientek@ubuntu:~/C++/02_namespace_example$
```

2.2 C++面向对象

面向对象的三大特征是继承，多态和封装，C++重面向对象重要的就是这些，我们下面通过一些简单的实例加以理解，从这小节开始，我们将开启新的编程旅途。与 C 语言编程的思想完全不同了，这就是 C++!理解概念和掌握这些编程方法对学习 C++有很大的好处。

2.2.1 类和对象

C++ 在 C 语言的基础上增加了面向对象编程，C++ 支持面向对象程序设计。类是 C++ 的核心特性，通常被称为用户定义的类型。类用于指定对象的形式，它包含了数据表示法和用于处理数据的方法。类中的数据和方法称为类的成员。函数在一个类中被称为类的成员。

打个比方说明一下什么是类，比如有一条小狗，小狗有名字叫旺财，旺财的年龄是 2 岁，同时旺财会汪汪的叫，也能跑。我们统称狗这个为类，类是我们抽象出来的，因为狗不只有上面的属性，还有体重，毛发的颜色等等，我们只抽象出几种属性成一个类。具体到哪条狗就叫对象。

从类中实例化对象分两种方法，一种是从栈中实例化对象，一种是从堆中实例化对象。

下面以自定义狗类介绍如何自定义类和如何使用对象。

在 Ubuntu 上编辑一个 03_class_dog_example 目录，在 03_class_dog_example 目录下新建一个 03_class_dog_example.cpp 文件，内容如下。

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class Dog
6  {
7  public:
8      string name;
9      int age;
10
11     void run() {
12         cout<<"小狗的名字是:"<<name<<","<<"年龄是"<<age<<endl;
13     }
14 };
15
16 int main()
17 {
18     Dog dog1;
19
20     dog1.name = "旺财";
21     dog1.age = 2;
22     dog1.run();
23
24     Dog *dog2 = new Dog();
25
26     if (NULL == dog2) {
27         return 0;
28     }
29     dog2->name = "富贵";
```

```
30      dog2->age = 1;
31      dog2->run();
32
33
34      delete dog2;
35      dog2 = NULL;
36      return 0;
37  }
```

第 5 行，定义了一个 Dog 狗，定义类时，起的类名要尽量贴近这个类，让人一看就明白，您这个类是做什么的。

第 7 行，访问限定符 public（公有的），此外还有 private（私有的）和 protected（受保护的）。写这个的目的是为了下面我们要调用这些成员，不写访问限定符默认是 private。关于访问限定符，如果是初学者可能会难理解。简单的来说，访问限定符就是设置一个成员变量和成员函数的访问权限而已，初学者暂时不必要深究什么时候应该用 public 和什么时候应该用 private。

第 8 至 11 行，定义了一个字符串变量 name，整形变量 age。和一个方法 run()。我们在这个 run()里打印相应的狗名和狗的年龄。PS：string 是 C++的数据类型，方便好用，使用频率相当高。

第 18 行，从栈中实例化一个对象 dog1(可以随意起名字)。

第 20 至 22 行，为 dog1 的成员变量赋值，dog1 的 name 赋值叫“旺财”，年龄为 2 岁。然后调用 run()方法，打印 dog1 的相关变量的信息。

第 24 行，从堆中实例化对象，使用关键字 new 的都是从堆中实例化对象。

第 26 行，从堆中实例化对象需要开辟内存，指针会指向那个内存，如果 new 没有申请内存成功，p 即指向 NULL，程序就自动退出，下面的就不执行了，写这个是为了严谨。

第 29 至 31 行，和 dog1 一样，为 dog2 的成员赋值。

第 34 和 35 行，释放内存，将 dog2 重新指向 NULL。

如果没有语法错误，我们完全可以预测到打印的结果。我们学习 C 语言的结构体，类其实和结构类似，可以说类是结构体的升级版本。

执行下面的指令开始编译。

```
g++ 03_class_dog_example.cpp -o 03_class_dog_example
```

编译完成后执行的结果如下。

```
alientek@ubuntu:~/C++/03_class_dog_example$ ./03_class_dog_example
小狗的名字是:旺财,年龄是2
小狗的名字是:富贵,年龄是1
alientek@ubuntu:~/C++/03_class_dog_example$
```

通过上面的例子我们已经学习了什么是类，和什么是对象。以描述 Dog 为一类（抽象出来的），从 Dog 类中实例出来就是对象（实际事物）。对象拥有 Dog 类里的属性，可以从栈中实例化对象，亦可从堆中实例化对象。类的编写过程和对象的使用过程大致如上了。我们只需要理解这个步骤，明白类的定义和使用即可。

2.2.1.1 构造函数与析构函数

什么是构造函数？构造函数在对象实例化时被系统自动调用，仅且调用一次。构造函数出现在哪里？前面我们学过类，实际上定义类时，如果没有定义构造函数和析构函数，编译器就会生成一个构造函数和析构函数，只是这个构造和析构函数什么事情也不做，所以我们不会注意到一点。

构造函数的特点如下：

（1） 构造函数必须与类名同名；

（2） 可以重载，（重载？新概念，后面学到什么是重载。）；

（3） 没有返回类型，即使是 void 也不行。

什么是析构函数？与构造函数相反，在对象结束其生命周期时系统自动执行析构函数。实际上定义类时，编译器会生成一个析构函数。

析构函数的特点如下：

（1） 析构函数的格式为~类名()；

（2） 调用时释放内存（资源）；

（3） ~类名()不能加参数；

（4） 没有返回值，即使是 void 也不行。

下面我们通过简单的例子来说明构造函数和析构函数的使用。新建一个目录 04_structor_example，编辑一个 04_structor_example.cpp 内容如下。

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class Dog
6  {
7  public:
8      Dog();
9      ~Dog();
10 };
11
12 int main()
13 {
14     Dog dog;
15     cout<<"构造与析构函数示例"<<endl;
16     return 0;
17 }
18
19 Dog::Dog()
20 {
21     cout<<"构造函数执行！"<<endl;
22 }
23
24 Dog::~Dog()
```

```
25  {
26      cout<<"析构函数执行！"<<endl;
27  }
```

我们还是以简单的狗类作为示例，定义一个狗类，把构造函数和析构函数写上。前面不是说会自动生成构造函数和析构函数的吗？注意是编译时，编译器生成的。当我们要使用构造函数和析构函数时需要我们自己在类里添加。

第 5 至第 10 行，定义了一个狗类，并在里面写了构造函数和析构函数。

第 14 行，使用 Dog 类实例化一个 dog 对象。

第 15 行，打印一句"构造与析构函数示例"。

第 19 至 22 行，类的函数可以在类里实现，也可以在类外实现，不过在类外实现时需要使用“::”，此时我们把类的构造函数定义在类的外面，打印一句"构造函数执行！"。

第 14 至 27 行，类的析造函数定义在类的外面，打印一句"析造函数执行！"。

执行下面的指令开始编译。

```
g++ 04_structor_example.cpp -o 04_structor_example
```

编译完成后执行的结果如下。

```
alientek@ubuntu:~/C++/04_structor_example$ ./04_structor_example
构造函数执行!
构造与析构函数示例
析构函数执行!
alientek@ubuntu:~/C++/04_structor_example$
```

其实执行的结果也是可以预测的，在对象实例化时会调用构造函数，所以 Dog()先执行，然后再在 main()函数里继续执行 cout<<"构造与析构函数示例"<<endl;。最后对象生命周期结束时才会执行析构函数。

2.2.1.2 this 指针

一个类中的不同对象在调用自己的成员函数时，其实它们调用的是同一段函数代码，那么成员函数如何知道要访问哪个对象的数据成员呢？

没错，就是通过 this 指针。每个对象都拥有一个 this 指针，this 指针记录对象的内存地址。在 C++中，this 指针是指向类自身数据的指针，简单的来说就是指向当前类的当前实例对象。

关于类的 this 指针有以下特点：

（1） this 只能在成员函数中使用，全局函数、静态函数都不能使用 this。实际上，成员函数默认第一个参数为 T * const this。也就是一个类里面的成员了函数 int func(int p)，func 的原型在编译器看来应该是 int func(T * const this,int p)。

（2） this 在成员函数的开始前构造，在成员函数的结束后清除。

（3） this 指针会因编译器不同而有不同的放置位置。可能是栈，也可能是寄存器，甚至全局变量。

下面以简单的例子来说明 this 的用法。我们还是以狗类为例，按上面的 this 解释，this 只能够在成员函数使用，并可以指向自身数据。我们就可以写这样简单的例子来说明 this 的用法。我们在 Qt 里也会遇到 this 这个东西，下面这个例子就很容易解释 Qt 里的 this 指针的用法。

新建一个目录 05_this_pointer_example，编辑一个 05_this_pointer_example.cpp 内容如下。

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class Dog
6  {
7  public:
8      string name;
9      void func();
10 };
11
12 int main()
13 {
14     Dog dog;
15     dog.func();
16     return 0;
17 }
18
19 void Dog::func()
20 {
21     this->name = "旺财";
22     cout<<"小狗的名字叫："<<this->name<<endl;
23 }
```

第 21 和 22 行，在类的成员函数里使用了 this 指针，并指向了类里的成员 name。先将 name 赋值叫“旺财”，然后我们打印 name 的值。

当程序没有语法错误里我们可以预测打印的结果，就是“小狗的名字叫：旺财”。

执行下面的指令开始编译。

```
g++ 05_this_pointer_example.cpp -o 05_this_pointer_example
```

程序执行的结果如下。

```
alientek@ubuntu:~/C++/05_this_pointer_example$ ./05_this_pointer_example
小狗的名字叫：旺财
alientek@ubuntu:~/C++/05_this_pointer_example$
```

2.2.2 继承

面向对象程序设计中最重要的一个概念是继承。继承允许我们依据另一个类来定义一个类，这使得创建和维护一个应用程序变得更容易。这样做，也达到了重用代码功能和提高执行效率的效果。

当创建一个类时，您不需要重新编写新的数据成员和成员函数，只需指定新建的类继承了一个已有的类的成员即可。这个已有的类称为基类，新建的类称为派生类。在 Qt 里大量的使用了这种特性，当 Qt 里的类不满足自己的要求时，我们可以重写这个类，就是通过继承需要重写的类，来实现自己的类的功能。

一个类可以派生自多个类，这意味着，它可以从多个基类继承数据和函数。定义一个派生类，我们使用一个类派生列表来指定基类。类派生列表以一个或多个基类命名，形式如下：

```
class derived-class: access-specifier base-class
```

与类的访问修饰限定符一样，继承的方式也有几种。其中，访问修饰符 access-specifier 是 public、protected 或 private 其中的一个，base-class 是之前定义过的某个类的名称。如果未使用访问修饰符 access-specifier，则默认为 private。

下面来捋一捋继承的方式，例子都是以公有成员和公有继承来说明，其他访问修饰符和其他继承方式，大家可以在教程外自己捋一捋。这个公有成员和继承方式也没有什么特别的，无非就是不同的访问权限而已，可以这样简单的理解。

1. 公有继承（public）：当一个类派生继承公有基类时，基类的公有成员也是派生类的公有成员，基类的保护成员也是派生类的保护成员，基类的私有成员不能直接被派生类访问，但是可以通过调用基类的公有和保护成员来访问。
2. 保护继承（protected）： 当一个类派生继承保护基类时，基类的公有和保护成员将成为派生类的保护成员。
3. 私有继承（private）：当一个类派生继承私有基类时，基类的公有和保护成员将成为派生类的私有成员。

下面我们还是以狗类为例，在 2.2.1 小节里我们定义的狗类，已经定义了 name，age 和 run() 方法。假设我们不想重写这个狗类，而是新建一个 Animal 类，让狗类去继承这个 Animal 类。假设是公有继承，那么我们是不是可以在狗类实例的对象里去使用继承 Animal 类里的成员呢？带着这个疑问，我们使用下面的例子来说明。

新建一个目录 06_inherit_example，编辑一个 06_inherit_example.cpp 内容如下。

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  /*动物类，抽象出下面两种属性，
6   *颜色和体重，是每种动物都具有的属性
7   */
8  class Animal
9  {
10 public:
11     /* 颜色成员变量 */
12     string color;
13     /* 体重成员变量 */
14     int weight;
```

```
15  };
16
17  /*让狗类继承这个动物类，并在狗类里写自己的属性。
18   *狗类拥有自己的属性 name，age，run()方法，同时也继承了
19   *动物类的 color 和 weight 的属性
20   */
21  class Dog : public Animal
22  {
23  public:
24      string name;
25      int age;
26      void run();
27  };
28
29  int main()
30  {
31      Dog dog;
32      dog.name = "旺财";
33      dog.age = 2;
34      dog.color = "黑色";
35      dog.weight = 120;
36      cout<<"狗的名字叫："<<dog.name<<endl;
37      cout<<"狗的年龄是："<<dog.age<<endl;
38      cout<<"狗的毛发颜色是："<<dog.color<<endl;
39      cout<<"狗的体重是："<<dog.weight<<endl;
40      return 0;
41  }
```

第 21 行，Animal 作为基类，Dog 作为派生类。Dog 继承了 Animal 类。访问修饰符为 public（公有继承）。

执行下面的指令开始编译。

```
g++ 06_inherit_example.cpp -o 06_inherit_example
```

编译完成执行的结果为如下。

```
alientek@ubuntu:~/C++/06_inherit_example$ ./06_inherit_example
狗的名字叫：旺财
狗的年龄是：2
狗的毛发颜色是：黑色
狗的体重是：120
alientek@ubuntu:~/C++/06_inherit_example$
```

2.2.3 重载

C++ 允许在同一作用域中的某个函数和运算符指定多个定义，分别称为**函数重载**和**运算符重载**。

重载声明是指一个与之前已经在该作用域内声明过的函数或方法具有相同名称的声明，但是它们的参数列表和定义（实现）不相同。

当您调用一个重载函数或重载运算符时，编译器通过把您所使用的参数类型与定义中的参数类型进行比较，决定选用最合适的定义。选择最合适的重载函数或重载运算符的过程，称为**重载决策**。

2.2.3.1 函数重载

在同一个作用域内，可以声明几个功能类似的同名函数，但是这些同名函数的形式参数（指参数的个数、类型或者顺序）必须不同。我们不能仅通过返回类型的不同来重载函数。在 Qt 源码里，运用了大量的函数重载，所以我们是有必要学习一下什么是函数重载。不仅在 C++，在其他语言的里，都能看见函数重载。因为需要不同，所以有重载各种各样的函数。

下面通过一个小实例来简单说明一下函数重载的用法。我们还是以狗类为说明，现在假设有个需求。我们需要打印狗的体重，分别以整数记录旺财的体重和小数记录旺财的体重，同时以整数打印和小数打印旺财的体重。那么我们可以通过函数重载的方法实现这个简单的功能。

新建一个目录 07_func_overloading，编辑一个 07_func_overloading.cpp 内容如下。

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class Dog
6  {
7  public:
8      string name;
9      void getWeight(int weight) {
10         cout<<name<<"的体重是: "<<weight<<"kG"<<endl;
11     }
12
13     void getWeight(double weight) {
14         cout<<name<<"的体重是: "<<weight<<"kG"<<endl;
15     }
16 };
17
18 int main()
19 {
20     Dog dog;
21     dog.name = "旺财";
22     dog.getWeight(10);
23     dog.getWeight(10.5);
24     return 0;
25 }
```

第 9 行，写了一个方法 getWeight(int weight)，以 int 类型作为参数。

第 13 行，以相同的函数名 getWeight，不同的参数类型 double weight，这样就构成了函数重载。

第 22 行与第 23 行，分别传进参数不同的参数，程序就会匹配不同的重载函数。

执行下面的指令编译。

```
g++ 07_func_overloading.cpp -o 07_func_overloading
```

程序执后的结果如下。

```
alientek@ubuntu:~/C++/07_func_overloading$ ./07_func_overloading
旺财的体重是: 10kG
旺财的体重是: 10.5kG
alientek@ubuntu:~/C++/07_func_overloading$
```

通过上面的例子我们可以知道重载函数的使用方法，避免用户传入的参数类型，有可能用户传入的参数类型不在我们写的重载函数里，假若用户传入了一个字符串类型，这样编译器就会匹配不到相应的重载函数，编译时就会报错。其实我们还可以多写几个重载函数，设置多几种类型，如 string 类型，char 类型，float 类型等。

2.2.3.2 运算符重载

运算符重载的实质就是函数重载或函数多态。运算符重载是一种形式的 C++多态。目的在于让人能够用同名的函数来完成不同的基本操作。要重载运算符，需要使用被称为运算符函数的特殊函数形式，运算符函数形式：operatorp（argument-list），operator 后面的'p'为要重载的运算符符号。重载运算符的格式如下：

```
<返回类型说明符> operator <运算符符号>(<参数表>)
{

    <函数体>

}
```

下面是可重载的运算符列表：

双目算术运算符	+ (加)，-(减)，*(乘)，/(除)，% (取模)
关系运算符	==(等于)，!= (不等于)，< (小于)，> (大于)，<=(小于等于)，>=(大于等于)
逻辑运算符	||(逻辑或)，&&(逻辑与)，!(逻辑非)
单目运算符	+ (正)，-(负)，*(指针)，&(取地址)
自增自减运算符	++(自增)，--(自减)
位运算符		(按位或)，& (按位与)，~(按位取反)，^(按位异或),，<< (左移)，>>(右移)

赋值运算符	=, +=, -=, *=, /= , % = , &=, |=, ^=, <<=, >>=
空间申请与释放	new, delete, new[ ] , delete[]
其他运算符	()(函数调用)，->(成员访问)，,(逗号)，[](下标)

下面是不可重载的运算符列表：

成员访问运算符	.
成员指针访问运算符	.*, ->*
域运算符	::
长度运算符	sizeof
条件运算符	?:
预处理符号	#

根据上表我们知道可以重载的运算符有很多，我们以重载“+”运算符为例，实际上用重载运算符我们在实际应用上用的比较少，我们只需要了解和学习这种思想即可。

下面的实例使用成员函数演示了运算符重载的概念。在这里，对象作为参数进行传递，对象的属性使用 this 运算符进行访问。下面还是以我们熟悉的狗类为例。声明加法运算符用于把两个 Dog 对象相加的体重相加，返回最终的 Dog 对象然后得到第三个 Dog 对象的体重。

新建一个目录 08_operator_example，编辑一个 08_operator_example.cpp 内容如下。

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class Dog
6  {
7  public:
8      int weight;
9      Dog operator+(const Dog &d) {
10         Dog dog;
11         dog.weight = this->weight + d.weight;
12         return dog;
13     }
14
15 };
16
17 int main()
18 {
19     Dog dog1;
20     Dog dog2;
21     Dog dog3;
```

```
22
23      dog1.weight = 10;
24      dog2.weight = 20;
25      dog3 = dog1 + dog2;
26      cout<<"第三只狗的体重是: "<<dog3.weight<<endl;
27      return 0;
28  }
```

第 9 至 13 行，重载“+”运算符，注意函数必须与类名同名，把 Dog 对象作为传递，使用 this 运算符进行访问。然后返回一个 dog 对象。

执行下面指令进行编译。

```
g++ 08_operator_example.cpp -o 08_operator_example
```

编译完成后运行的结果如下。

```
alientek@ubuntu:~/C++/08_operator_example$ ./08_operator_example
第三只狗的体重是: 30
alientek@ubuntu:~/C++/08_operator_example$
```

结果可以预知的，重载运算符“+”，可以把两个对象进行相加。在普通的算术运算符“+”是不能将两个对象进行相加的，所以我们重载运算符的意义可以体现在这里。

2.2.4 多态

C++多态意味着调用成员函数时，会根据调用函数的对象的类型来执行不同的函数；

形成多态必须具备三个条件：

1. 必须存在继承关系；
2. 继承关系必须有同名虚函数（其中虚函数是在基类中使用关键字 virtual 声明的函数，在派生类中重新定义基类中定义的虚函数时，会告诉编译器不要静态链接到该函数）；
3. 存在基类类型的指针或者引用，通过该指针或引用调用虚函数。

这里我们还需要理解两个概念：

虚函数：

是在基类中使用关键字 virtual 声明的函数。在派生类中重新定义基类中定义的虚函数时，会告诉编译器不要静态链接到该函数。我们想要的是在程序中任意点可以根据所调用的对象类型来选择调用的函数，这种操作被称为动态链接，或后期绑定。虚函数声明如下：virtual ReturnType FunctionName(Parameter) 虚函数必须实现，如果不实现，编译器将报错

纯虚函数：

若在基类中定义虚函数，以便在派生类中重新定义该函数更好地适用于对象，但是您在基类中又不能对虚函数给出有意义的实现，这个时候就会用到纯虚函数。纯虚函数声明如下：virtual void funtion1()=0; 纯虚函数一定没有定义，纯虚函数用来规范派生类的行为，即接口。包含纯虚函数的类是抽象类，抽象类不能定义实例，但可以声明指向实现该抽象类的具体类的指针或引用。

上面那些概念大家可以捋一捋，毕竟 C++概念还是挺多的。为什么说到多态要与虚函数和纯虚函数扯上关系？光说概念没有实例确实难理解。下面我们还是以我们熟悉的狗类和动物类，另外加一个猫类进行多态的讲解。

新建一个目录 09_polymorphism_example，编辑一个 09_polymorphism_example.cpp 内容如下。（PS: polymorphism 翻译多态的意思，笔者就以这种方式命名例程了）。

```
#include <iostream>
#include <string>
using namespace std;

/* 定义一个动物类 */
class Animal
{
public:
    virtual void run() {
        cout<<"Animal 的 run()方法"<<endl;
    }
};

/* 定义一个狗类，并继承动物类 */
class Dog : public Animal
{
public:
    void run() {
        cout<<"Dog 的 run()方法"<<endl;
    }

};

/* 定义一个猫类，并继承动物类 */
class Cat : public Animal
{
public:
    void run() {
        cout<<"Cat 的 run()方法"<<endl;
    }

};

int main()
{
    /* 声明一个 Animal 的指针对象，注：并没有实例化 */
    Animal *animal;
    /* 实例化 dog 对象 */
    Dog dog;
    /* 实例化 cat 对象 */
```

```
41      Cat cat;
42
43      /* 存储 dog 对象的地址 */
44      animal = &dog;
45      /* 调用 run()方法 */
46      animal->run();
47
48      /* 存储 cat 对象的地址 */
49      animal = &cat;
50      /* 调用 run()方法 */
51      animal->run();
52      return 0;
53  }
```

第 9 行、第 18 行和第 28 行，都有一个 run()方法。其中我们可以看到基类 Animal 类的 run()方法前面加了关键字 virtual。这样让基类 Animal 类的 run()方法变成了虚函数。在这个例子里我们可以知道**虚函数是 C++中用于实现多态(polymorphism)的机制。核心理念就是通过基类访问派生类定义的函数**。简单的来说，上面的实例是基类 Animal 声明了一个指针 animal。然后通过基类的指针来访问 Dog 类对象与 Cat 类的对象的 run()方法，前提是基类的 run()方法必须声明为虚函数，如果不声明为虚函数，基类的指针将访问到基类自己的 run()方法。我们可以尝试把 virtual 关键字去掉再重新编译测试，如果不加关键字 virtual 会是什么情况。

第 44 行和第 49 行，可以理解是 animal 指针实例化的过程。当基类的 run()方法定义成虚函数，编译器不静态链接到该函数，它将链接到派生类的 run()方法，进行实例化。

执行下面的指令编译。

```
g++ 09_polymorphism_example.cpp -o 09_polymorphism_example
```

编译完成执行的结果如下。

```
alientek@ubuntu:~/C++/09_polymorphism_example$ ./09_polymorphism_example
Dog的run()方法
Cat的run()方法
alientek@ubuntu:~/C++/09_polymorphism_example$
```

2.2.5 数据封装

封装是面向对象编程中的把数据和操作数据的函数绑定在一起的一个概念，这样能避免受到外界的干扰和误用，从而确保了安全。数据封装引申出了另一个重要的 OOP 概念，即**数据隐藏**。

数据封装是一种把数据和操作数据的函数捆绑在一起的机制，**数据抽象**是一种仅向用户暴露接口而把具体的实现细节隐藏起来的机制，C++ 通过创建类来支持封装和数据隐藏（public、protected、private）。

其实我们在第 2.2 小节开始就已经接触了数据封装。在 C++程序中，任何带有公有和私有成员的类都可以作为数据封装和数据抽象的实例。通常情况下，我们都会设置类成员状态为私有（private），除非我们真的需要将其暴露，这样才能保证良好的封装性。这通常应用于数据成员，但它同样适用于所有成员，包括虚函数。

下面我们还是以狗类为例，增加一个食物的方法 addFood(int number)。将获得食物的方法设定在 public 下，这样 addFood(int number)方法就暴露出来了，也就是对外的接口。然后我们设置狗类的私有成员(private)食物的份数 total。我们在这个教程里第一次使用 private，在这章节里我们也可以学到什么时候该使用 private 什么时候使用 public。total 为获得的食物总数，然后我们还写一个公开的方法 getFood()在 public 下，通过 getFood()来打印出小狗总共获得了几份食物。

新建一个目录 10_encapsulation_example，编辑一个 10_encapsulation_example.cpp 内容如下。（PS: encapsulation 翻译封装的意思，笔者就以这种方式命名例程了）。

```
#include <iostream>
#include <string>
using namespace std;

class Dog
{
public:
    string name;

    Dog(int i = 0)
    {
        total = i;
    }

    void addFood(int number) {
        total = total + number;
    }

    int getFood() {
        return total;
    }
private:
    int total;
};


int main()
{
    Dog dog;

    dog.name = "旺财";

    dog.addFood(3);
    dog.addFood(2);
```

```
35
36      cout<<dog.name<<"总共获得了"<<dog.getFood()<<"份食物"<<endl;
37
38      return 0;
39  }
```

第 10 至第 13 行，在构造函数里初始化 total 的数量，不初始化 total 的数量默认是随 int 类型的数。所以我们需要在构造函数里初始化，也体现了构造函数的功能，一般是在构造函数里初始化。不要在类内直接赋值初始化，有可能有些编译器不支持。

第 15 至 17 行，addFood(int number)，在这个方法里，将获得的食物份数赋值给 total。

第 19 至 21，getFood()，在这个方法里，将返回食物的总份数。通过调用这个方法，即可访问私有成员的 total 总数。

第 33 和 34 行，添加食物的份数。

第 36 行，打印食物的总份数。

执行下面的指令编译。

```
g++ 10_encapsulation_example.cpp -o 10_encapsulation_example
```

编译完成执行的结果如下。

```
alientek@ubuntu:~/C++/10_encapsulation_example$ ./10_encapsulation_example
旺财总共获得了5份食物
alientek@ubuntu:~/C++/10_encapsulation_example$
```

2.2.6 数据抽象

数据抽象是指，只向外界提供关键信息，并隐藏其后台的实现细节，即只表现必要的信息而不呈现细节。数据抽象是一种依赖于接口和实现分离的编程（设计）技术。

数据抽象的好处：

1. 类的内部受到保护，不会因无意的用户级错误导致对象状态受损。
2. 类实现可能随着时间的推移而发生变化，以便应对不断变化的需求，或者应对那些要求不改变用户级代码的错误报告。

举个简单的例子，比如我们生活中的手机。手机可以拍照、听音乐、收音等等。这些都是手机上的功能，用户可以直接使用。但是拍照的功能是如何实现的，是怎么通过摄像头取像然后怎么在屏幕上显示的过程，作为用户是不需要知道的。也就是暴露的不用太彻底，用户也不必须知道这种功能是如何实现的，只需要知道如何拍照即可。

就 C++ 编程而言，C++ 类为数据抽象提供了可能。它们向外界提供了大量用于操作对象数据的公共方法，也就是说，外界实际上并不清楚类的内部实现。

其实像 cout 这个对象就是一个公共的接口，我们不必要知道 cout 是如何在屏幕上显示内容的。cout 已经在底层实现好了。

在上一节我们已经学习过数据封装，**数据封装**是一种把数据和操作数据的函数捆绑在一起的机制，而**数据抽象**是一种仅向用户暴露接口而把具体的实现细节隐藏起来的机制。

C++ 程序中，任何带有公有和私有成员的类都可以作为数据抽象的实例。例子略，例子可参考上 2.2.5 小节的例子。

2.2.7 接口（抽象类）

接口描述了类的行为和功能，而不需要完成类的特定实现。C++ 接口是使用抽象类来实现的，抽象类与数据抽象互不混淆，数据抽象是一个把实现细节与相关的数据分离开的概念。**如果类中至少有一个函数被声明为纯虚函数，则这个类就是抽象类**。纯虚函数是通过在声明中使用 "= 0" 来指定的。

设计抽象类（通常称为 ABC）的目的，是为了给其他类提供一个可以继承的适当的基类。抽象类**不能被用于实例化对象**，它只能**作为接口使用**。如果试图实例化一个抽象类的对象，会导致编译错误。

因此，如果一个 ABC 的子类需要被实例化，则必须实现每个虚函数，这也意味着 C++ 支持使用 ABC 声明接口。如果没有在派生类中重写纯虚函数，就尝试实例化该类的对象，会导致编译错误。可用于实例化对象的类被称为具体类。

根据概念我们来写个实例来说明抽象类。

还是以狗类为说明，例程与 2.2.4 小节类似，只是 Aninmal 类的 run()方法定义为纯虚函数，纯虚函数不用实现，由派生类 Dog 和 Cat 类实现重写即可。

新建一个目录 11_abstract_class，编辑一个 11_abstract_class.cpp 内容如下。

```
1  #include <iostream>
2
3  using namespace std;
4
5  /* 定义一个动物类 */
6  class Animal
7  {
8  public:
9      virtual void run() = 0;
10 };
11
12 /* 定义一个狗类，并继承动物类 */
13 class Dog : public Animal
14 {
15 public:
16     void run() {
17         cout<<"Dog 的 run()方法"<<endl;
18     }
19
20 };
21
22 /* 定义一个猫类，并继承动物类 */
```

```
23  class Cat : public Animal
24  {
25  public:
26      void run() {
27          cout<<"Cat 的 run()方法"<<endl;
28      }
29
30  };
31
32  int main()
33  {
34      /* 实例化 dog 对象 */
35      Dog dog;
36
37      /* 实例化 cat 对象 */
38      Cat cat;
39
40      /*  dog 调用 run()方法 */
41      dog.run();
42
43      /* cat 调用 run()方法 */
44      cat.run();
45
46      return 0;
47  }
```

执行下面指令进行程序编译。

```
g++ 11_abstract_class.cpp -o 11_abstract_class
```

程序运行的结果如下。

```
alientek@ubuntu:~/C++/11_abstract_class$ ./11_abstract_class
Dog的run()方法
Cat的run()方法
alientek@ubuntu:~/C++/11_abstract_class$
```

虽然结果和例程与 2.2.4 小节一样，但是却表现了两种不同的思想。学 C++重要的是思想，当我们对这种思想有一种的了解后，不管是 Qt 或者其他 C++程序，我们都能快速学习和了解。C++的内容就到此结束了。在这个 C++基础中，我们的例子非常简单，也十分之易懂，重要的是理解概念，许多 C++的课程都是以 C++的功能甚至是很复杂的算法作讲解，内容复杂且多。只要我们理解好上面的 C++的基础，对学习 C++有很大的帮助，不要求对 C++有很深的理解，至少在我们后面学习 Qt 时已经大概了解 Qt 中的 C++语法。

第三章 初识 Qt

本章将介绍什么是 Qt，同时与大家一起安装 Qt，根据不同用户的编程习惯，这里我们介绍在 Windows 安装和在 Ubuntu 下安装。教程重点是以在 Ubuntu 环境下编写 Qt 作讲解。配置 Ubuntu 下的 Qt Creator 的中文输入法，讲解 Qt Creator 的界面组成与设置。新建一个简单的“Hello world”例程带大家一起学习 Qt 应用程序的建立、编译及调试步骤。

我们重点关注 3.4 小节，搭建 Linux 下 Qt 的开发环境，因为搭建环境是很关键的一步。有些朋友可能在搭建环境这一步就卡住，走不下去了，后面的开发还有很长的路要走。所以我们需要有足够的耐心，不要浮躁，细心的按教程的步骤走，遇到错误，学会随机应变，找解决方法。如果对 Ubuntu 的使用不熟悉，建议多多练习。

3.1 Qt 是什么

Qt 是一个跨平台的 C++开发库。主要用来开发图形用户界面（Graphical User Interface，简称 GUI）程序。

Qt 虽然经常被当做一个 GUI 库，用来开发图形界面应用程序，但这并不是 Qt 的全部；Qt 除了可以绘制漂亮的界面（包括控件、布局、交互），还包含很多其它功能，比如多线程、访问数据库、图像处理、音频视频处理、网络通信、文件操作等，这些 Qt 都已经内置了。

Qt 还存在 Python、Ruby、Perl 等脚本语言的绑定， 也就是说可以使用脚本语言开发基于 Qt 的程序。开源社区就是这样，好东西就会被派生扩展，到处使用，越来越壮大。

Qt 支持的操作系统有很多，例如通用操作系统 Windows、Linux、Unix，智能手机系统 Android、iOS、WinPhone， 嵌入式系统 QNX、VxWorks 等等。

简单的来说，Qt 可以做很多东西，好比如 Windows 下的软件也有很多是 Qt 开发的，这里我很喜欢它的一个特性就是跨平台使用，“跨平台”代表一份代码可以无需任何修改或者小修改就可以在其他平台上运行。

笔者不再介绍 Qt 的发展史了，这些网上可以查到。写太多感觉也没有什么用，反正 Qt 就是您做界面需要选择的一个好工具！

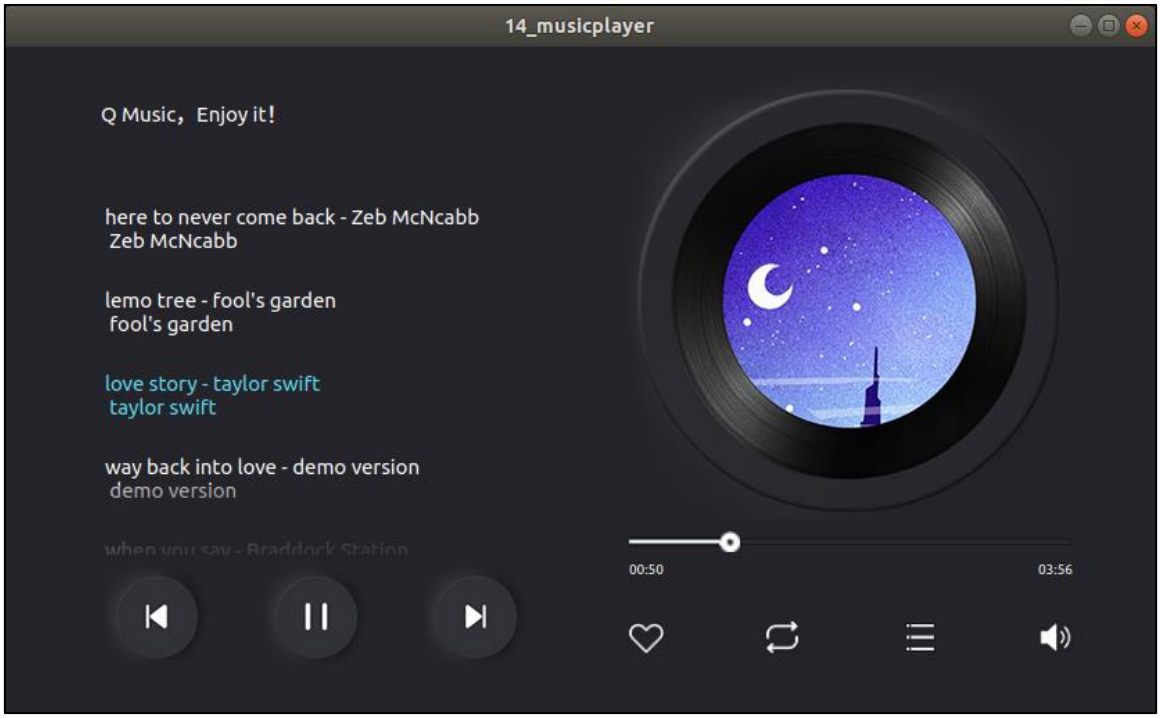
3.1.1 Qt 与 Qt Creator 的关系

3.1.2 Qt 能做什么

Qt 能做什么？理论上来说，您能想到的 App，Qt 它都基本能做。像 WPS 这种大型的桌面应用，Maya(3D 建模)，一些游戏的内核都可以使用 Qt 来实现。甚至您能快速使用 Qt 来开发一款像酷狗，网易云音乐这样的音乐播放器。在嵌入式里，使用 Qt 来开发界面已经是无可替代的一种趋势。工控界面最常用，一些移动端的界面也开始使用 Qt。像点菜机，温度采集数据显示，汽车仪表，速度显示界面等。Qt 能做优美的界面，所以推出了 QML（一种描述性的语言）。每次 Qt 更新都有在 Quick（QML 使用的模块）上优化或者添加功能，看来 Qt 打算在移动设备端的开发市场端进军！呵呵，其实 Qt 早已经在这块发展了！

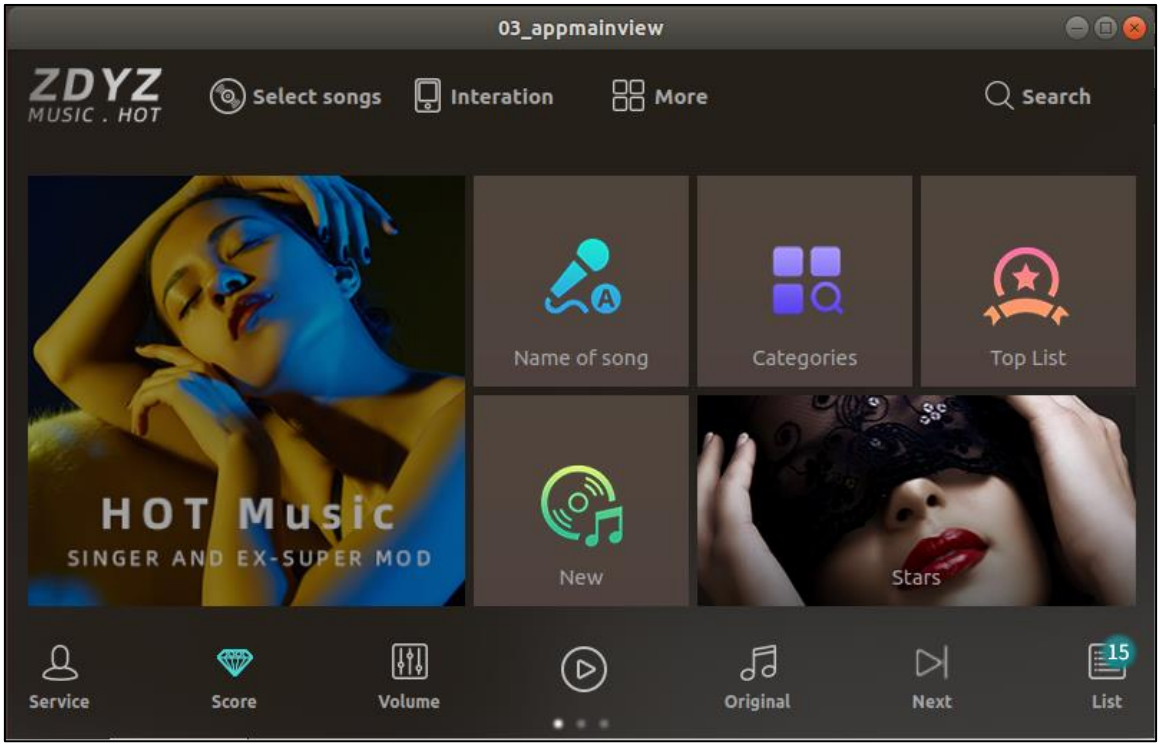
下面是本书开发的一些界面小例子，请欣赏。要想开发好的界面关键是要有想法。可以说如果您是做界面其实大多数时间花在美化界面上，功能都是比较容易实现的。最好读者需要有一定的美工基础，例如会基本的 PS，在开发过程中就不用去麻烦公司的美工了，开发界面和程序一起做。

暗黑主题音乐播放器界面 UI 设计。



简约的视频界面设计。

炫酷车载音乐 APP 主界面开发。



这些设计都尽在第二篇提高篇里的第十二章多媒体章节和项目实战篇里。笔者水平有限，但是爱设计 UI，尽量把 UI 设计得好看与实用。至少比其他 Qt 书籍设计的好看。Qt 设计本来就应该在界面上多下功夫，功能都是不难实现的，日后在大家会发现，设计界面占用的时间可能是 70%，写程序可能占 30%。大家应该在设计界面上多多下功夫!!!

3.1.3 Qt/C++与 QML

QML 是 Qt C++基础上拓展的。也就是说 Qt C++上所使用界面的类，基本都是可以拓展到 QML 上。如果不能拓展的，也可以直接 QML 与 C++一起使用。实际上 Qt C++效率更高。所以本教程都是 Qt C++为基础做教程入门教程，QML 可以说是一个加分项。Qt QML 模块提供 QML 语言开发应用程序和库提供了一个框架。QML 本身不涉及显示部分的内容，所以 Qt Quick 模块充当了这部分的作用。至于他们三者的关系，我们就这样简单的了解一下就行了。

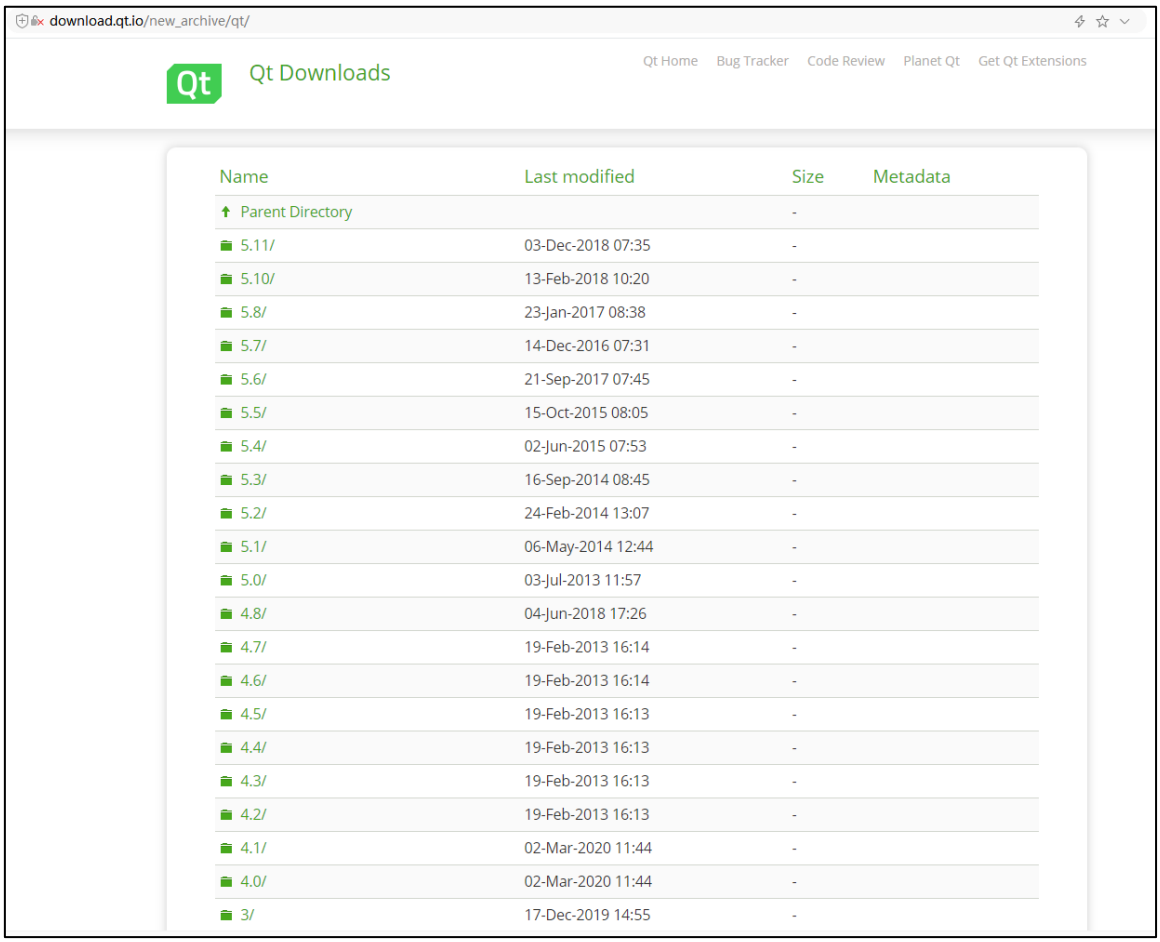
Qt C++在底层编写功能更突出，若要做好看的界面且功能强大，Qt C++与 QML 语言结合是必不可少的。QML 语言负责界面层，而底层调用 Qt C++。比如文件的读写、图像处理、多线程通信等这些都是需要 C++来编写完成的。

在目前来说，Qt C++更成熟，QML 目前也比较成熟了，因为每次更新版本 QML 也有更新。不过在低版本的 Qt（如 Qt5.6 以下），QML 还不够成熟，QML 在高版本的 Qt（如 Qt5.7 以上）更成熟。C++依旧是 Qt 的主要编程语言，Qt 5 也并没有忽略它，Qt 5 添加了很多新的 C++ API，而且会持续更新。引入 QML 应该是 Qt 向移动端市场的一个突破，还有 python 和 javascript 都可以在 Qt 里用。

说了 C++又说 QML，但是 Qt 基础教程还是主打 Qt C++，有了基础学 Qt 里其他的东西就不难！

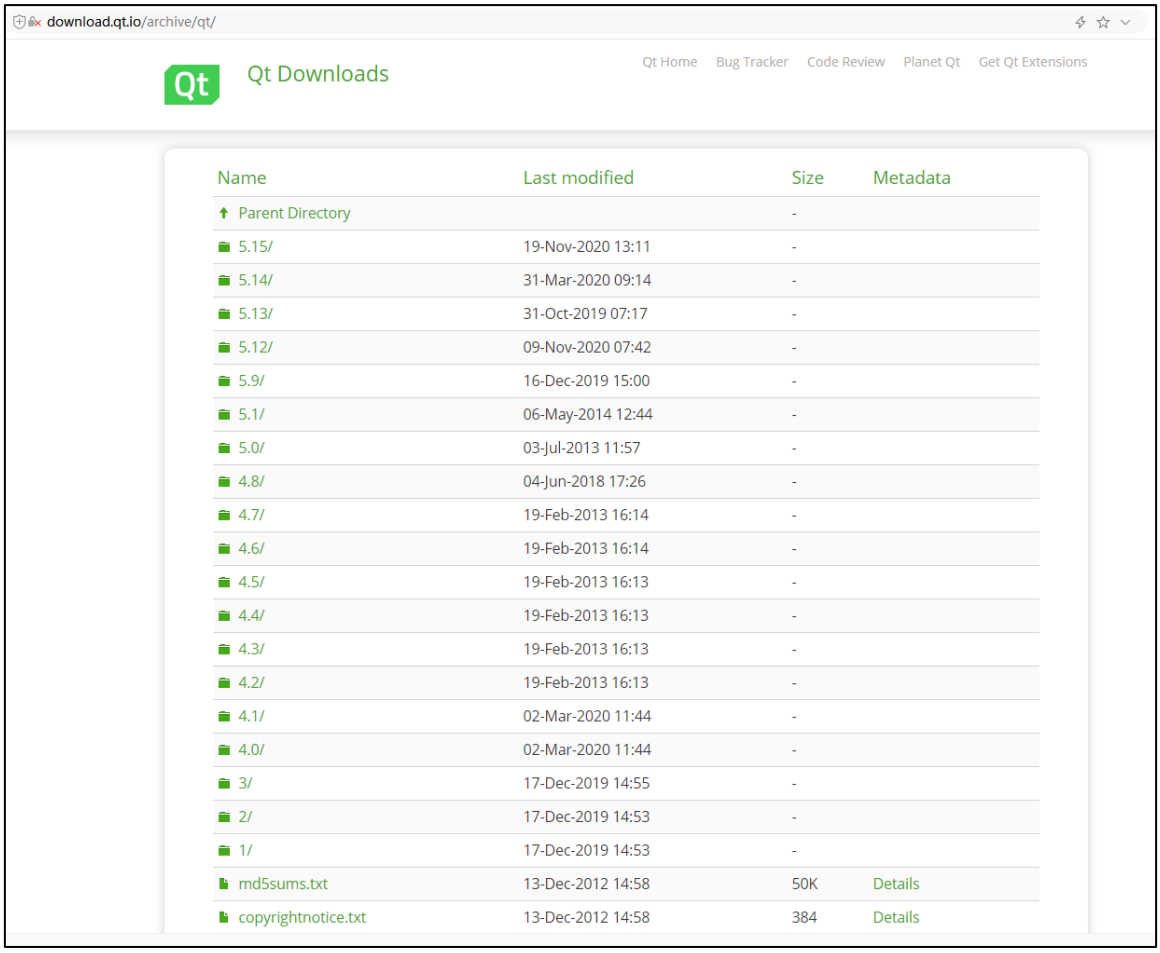
3.2 如何选择 Qt 版本

初学者应该要如何选择 Qt 的版本？截至 2020 年最新的 Qt5 版本是 Qt5.15。可以参考这个网址 https://doc.qt.io/qt-5/qt5-intro.html 来看看各个 Qt 版本的更新说明。Qt6 已经出了，我们有必要直接上最新版本的 Qt6 吗？不！Qt6 理论上与 Qt5 能兼容，但是许多公司在用 Qt5 甚至 Qt 4。学习了 Qt5，Qt6 等稳定版本出来，等市场上用的多，我们用 Qt6 才是最明智的选择，新出的事物往往要有一定的时间去适应！我们主要是简单对整个 Qt 版本说明一下。首先，Qt 版本的升级肯定是对上一版本进行优化。比如 Qt4 与 Qt5 的的语法区别，Qt5 的信号槽写法与 Qt4 的信号槽写法有区别。除了 Qt5.10 以上的版本，Qt5.6 与 Qt5.9 是 Qt 长期支持的版本 LTS（Long Term Support，简称 LTS）。但是 2020 年 3 月最近 Qt 官方将 Q5.2~Qt5.8、Qt5.11 归档到 http://download.qt.io/new_archive/qt/。所以很多小伙伴们想用老版本的 Qt 要到上面这个链接去下载了。毕竟有些时候我们开发板程序处于某个 Qt 版本下开发的，如果要迁移到新版本的 Qt 可能还要考虑到兼容问题！不过变化不会很大的，请放心！



下面贴出 Qt 官方下载链接 http://download.qt.io/archive/qt/。我们直接进入这个链接可以看到下图。下图和上图加起来才是 Qt 的完整版本。那么多版本任君选择。还可以看到 Qt 版本在不断的更新，更多的功能将会注入到 Qt 这个 GUI 库里，使我们开发变得更简单和有趣！Qt6 的时代即将到来！

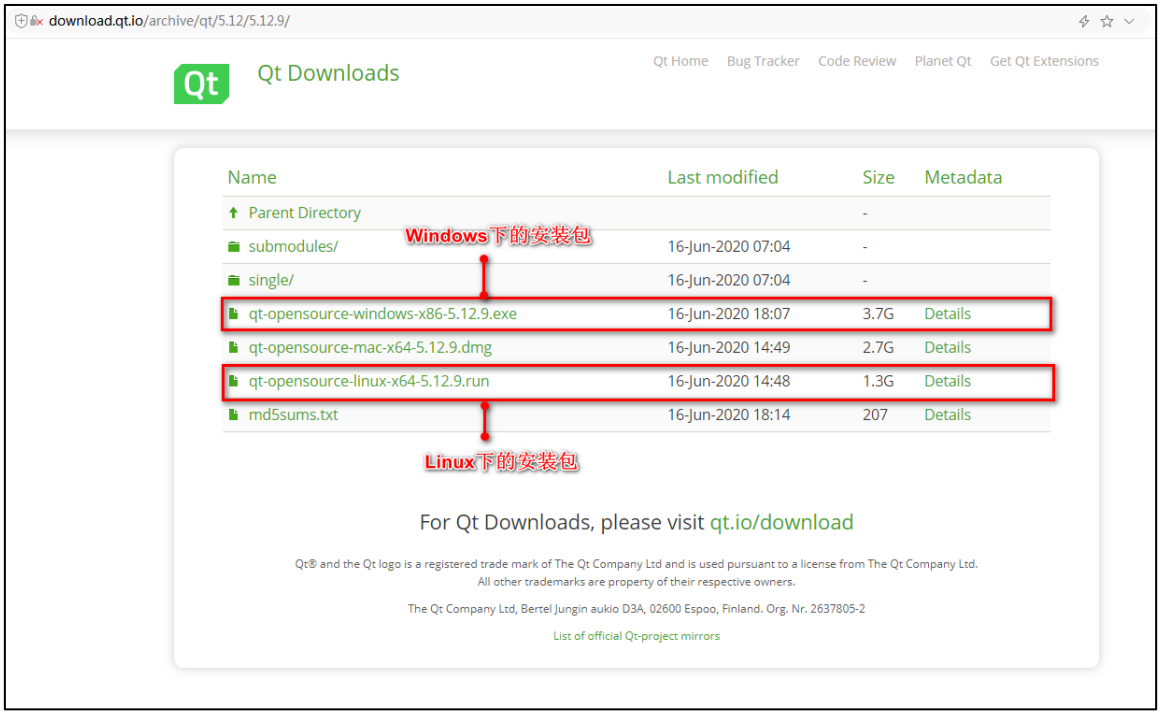
理论上我们选择 Qt 的版本越新越好，这是当然的，不过我们还是要确定一个版本是必须的，因为日后写好的程序要长期运行在一个确定的版本里，避免随意升级带来其他兼容性问题，或者重复重复移植等工作。Qt 还会不断的更新，Qt5.9 及 Qt5.12 是两个 LTS 长期支持的版本，本教程以的 Qt5.12.9 版本进行开发及实验说明。



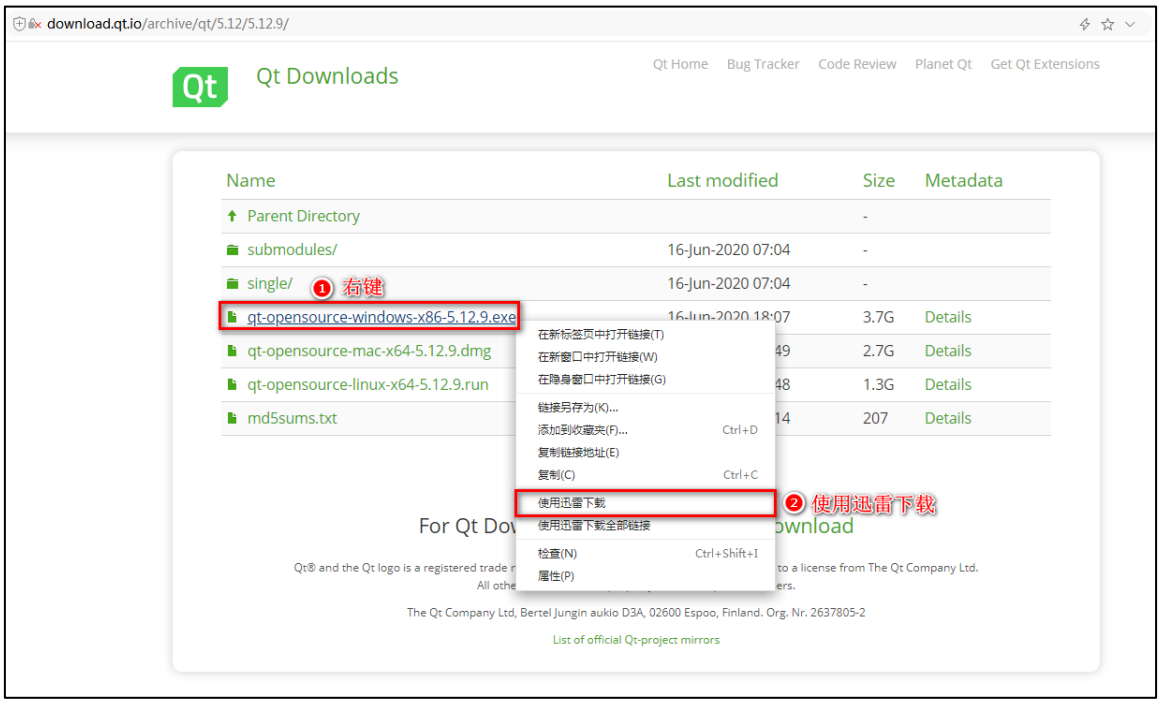
3.3 Windows 下安装 Qt

在上一小节里，我们已经知道 Qt 的下载地址。在 Windows 按如下方法下载。进入地址，此节只介绍 Windows 下的 Qt 安装。

进入 http://download.qt.io/archive/qt/5.12/5.12.9/，（注意如果找不到下载链接，我们就进行 http://download.qt.io 这个顶层目录一个个目录找，因为 Qt 下载链接会变动）。 从 Qt5.12 版本里选择 Qt5.12 的第九个版本。 “.dmg”是 Mac 系统版本的安装包，此处不作安装讲解。下图为 Windows 和 Linux 下的安装包格式。

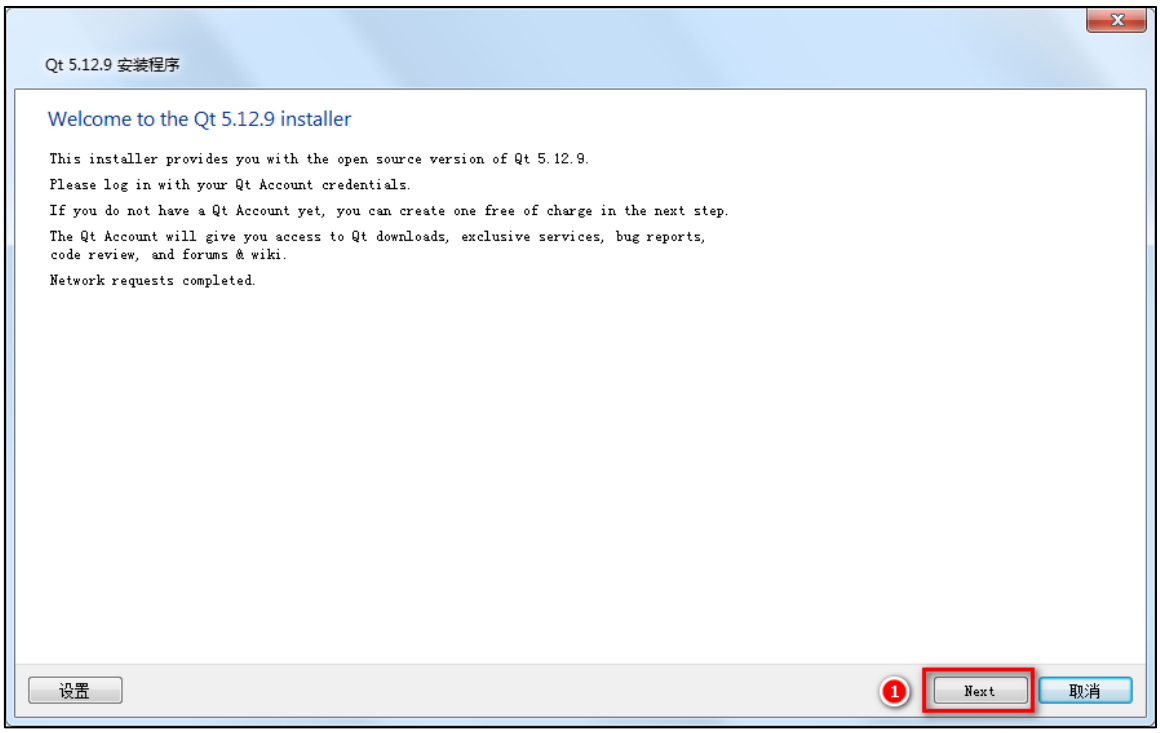


鼠标右键选中 Winodws 下载安装包，选择使用迅雷下载，如果没有安装迅雷就直接点击链接下载吧，下载速度可能会比较慢。使用迅雷下载，速度会比较快。

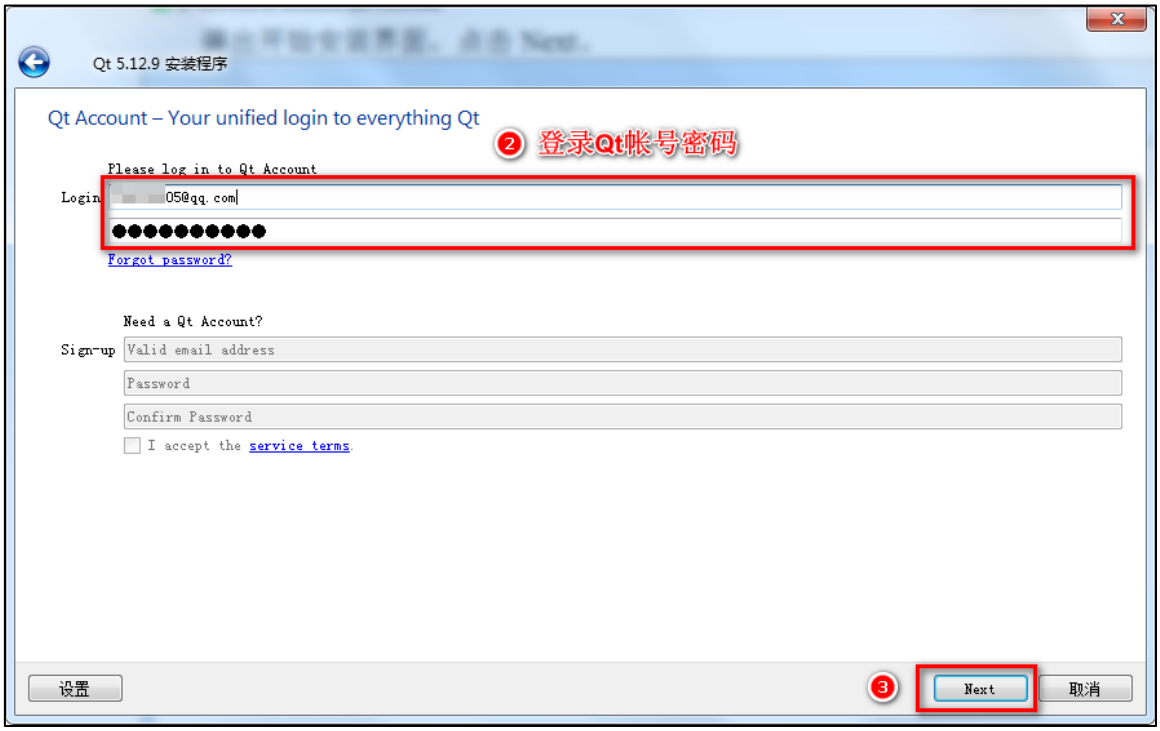


下载完成后，找到安装包文件，双击该安装包文件。开始安装。

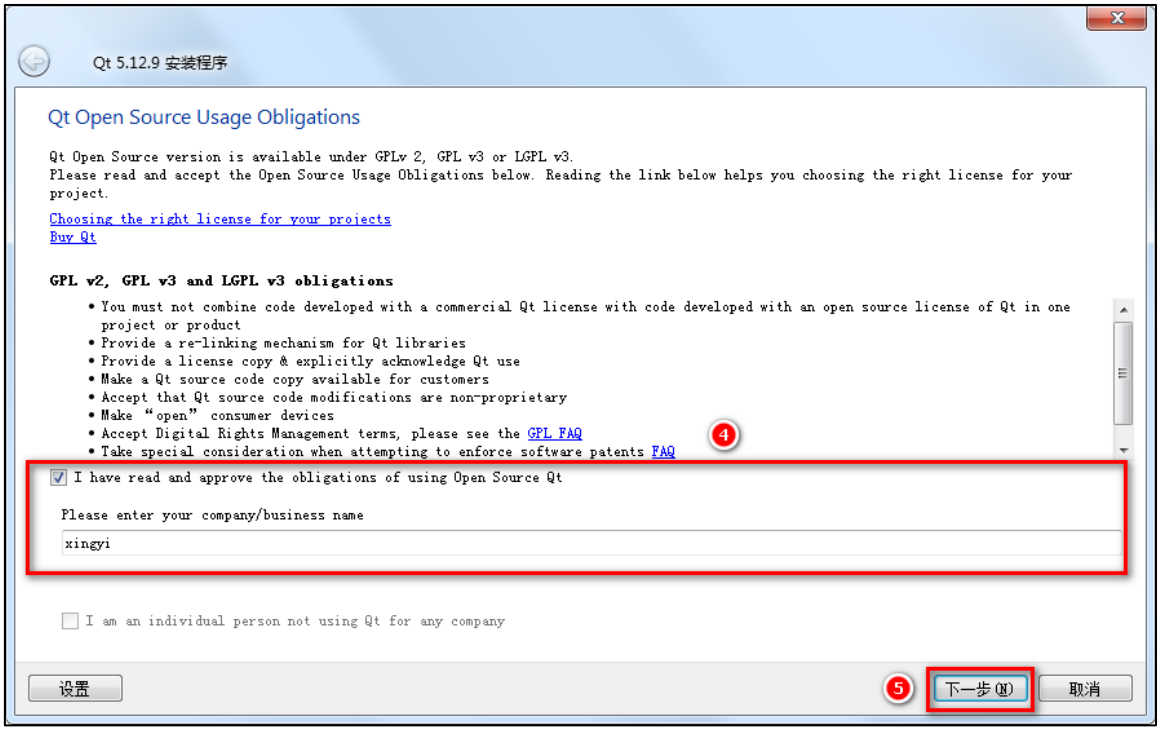
弹出开始安装界面，点击 Next。



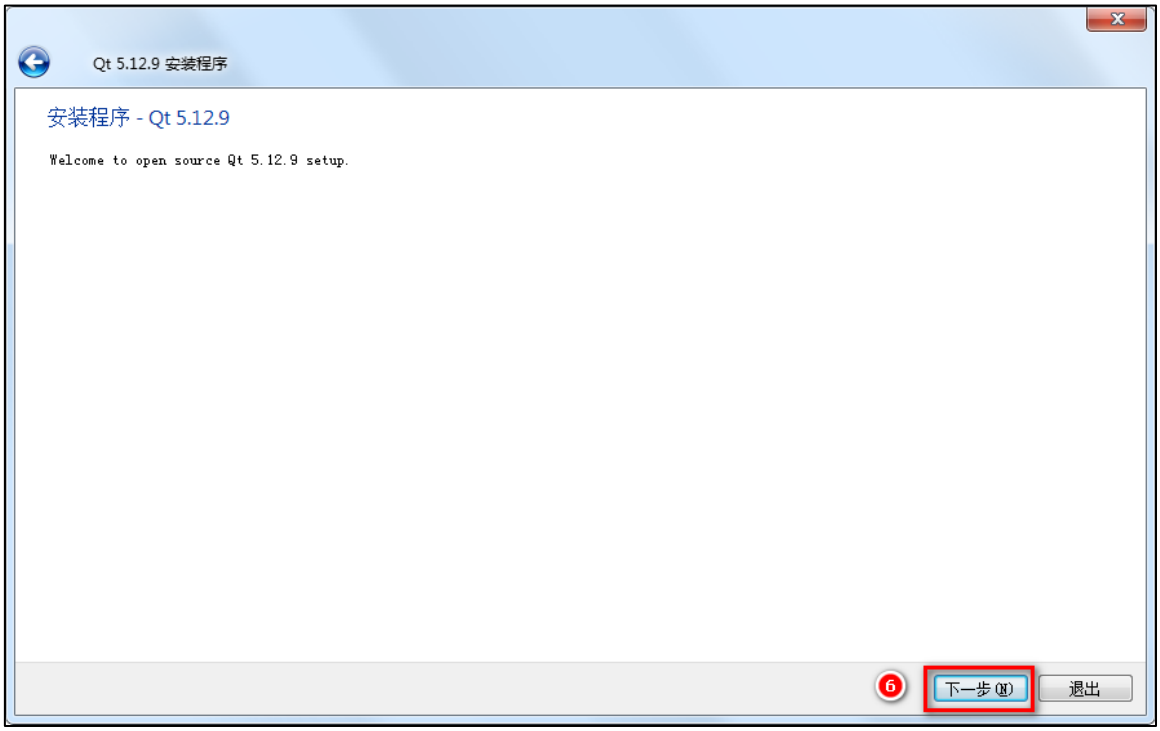
登录 Qt 帐号，如果您还没有帐号及密码，请到 https://www.qt.io/自行注册一个。



Qt 安装程序，开源版本的 Qt 遵循 GPLv 2，GPL v3 或者 LGPL v3 协议。勾选同意使用开源版本 Qt，填写公司/个人的名字。

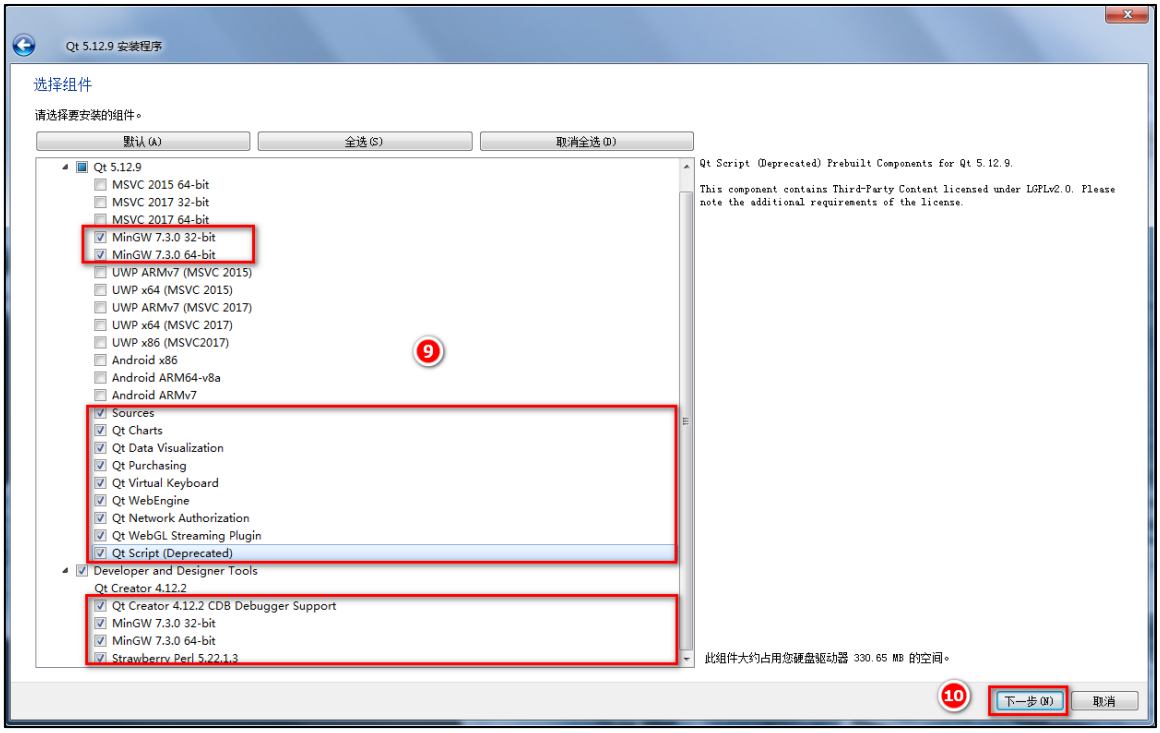


Qt 的欢迎安装界面

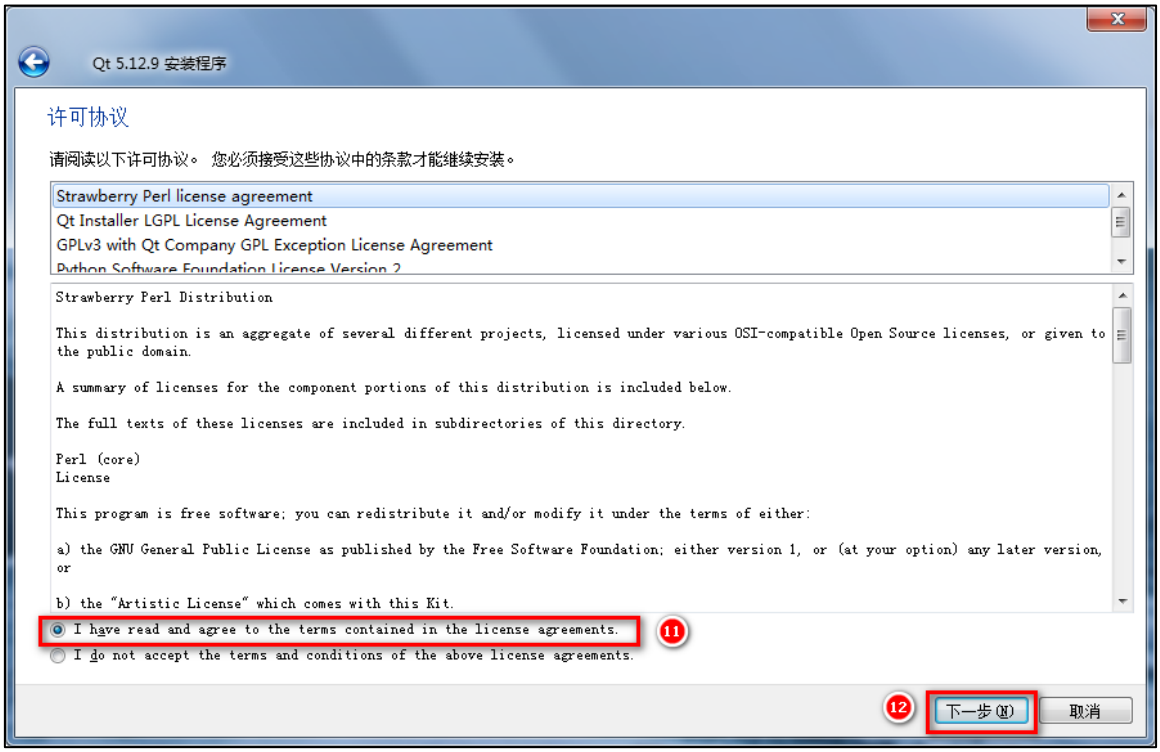


修改安装目录

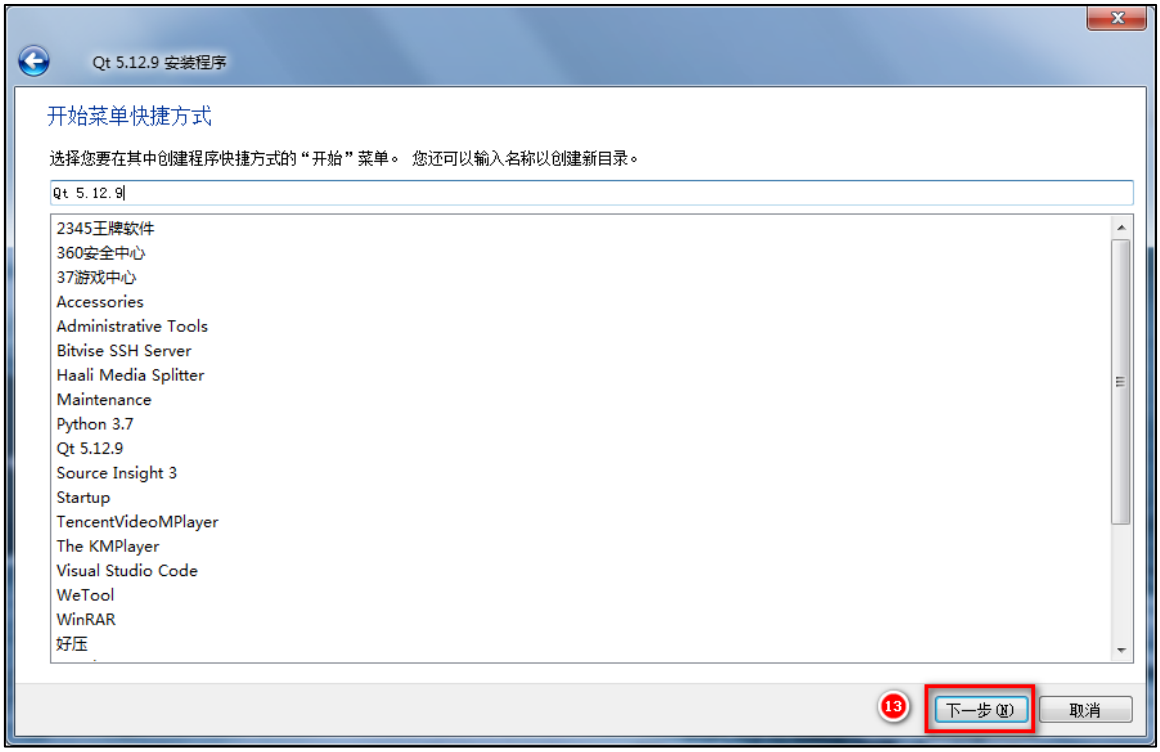
选择需要安装的组件，Windows 选择安装的组件如下。MSVC 是微软的 VC 编译器，需要安装 VC 相关库才能使用。MinGW 是指是 Minimalist GNU on Windows 的缩写，允许您在 GNU/Linux 和 Windows 平台生成本地的 Windows 程序而不需要第三方 C 运行时库。除了安卓选项 arm 等选项，我们只需要勾选 MinGW 的即可。Sources 是 Qt 源码，Source 以下的是 Qt 的一些模块，可选择安装或者不装。这里我们选择全装。



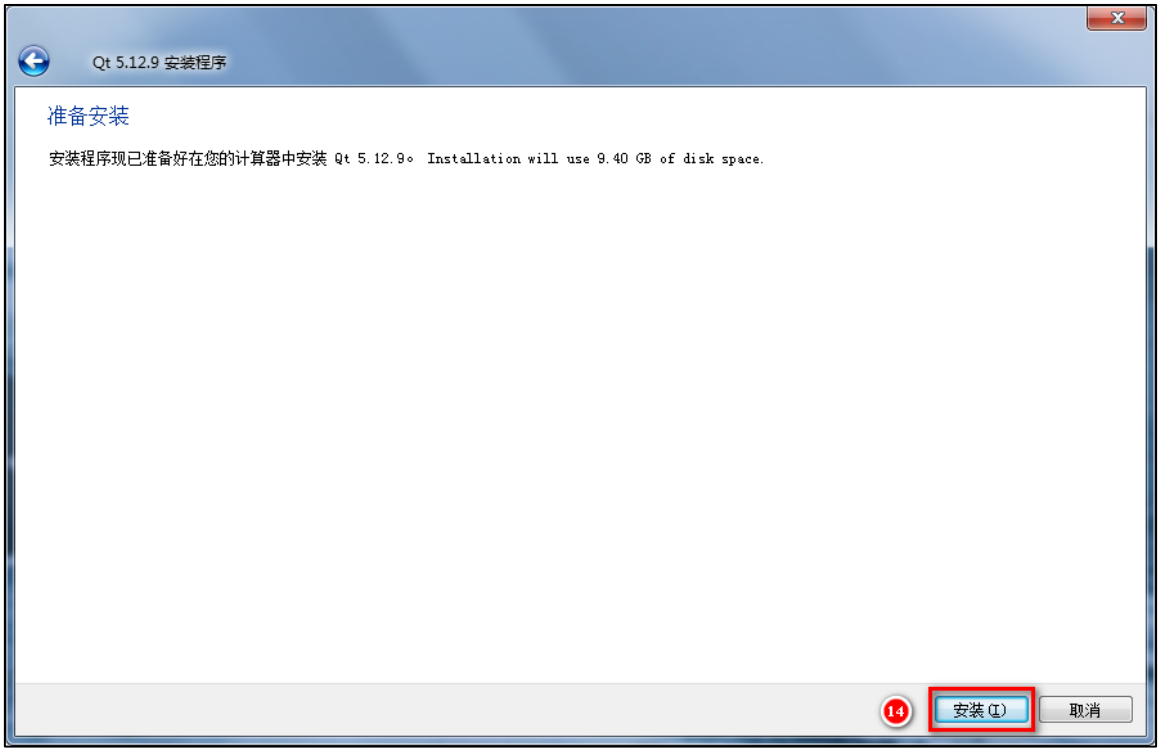
选择同意许可协议，再点击下一步。



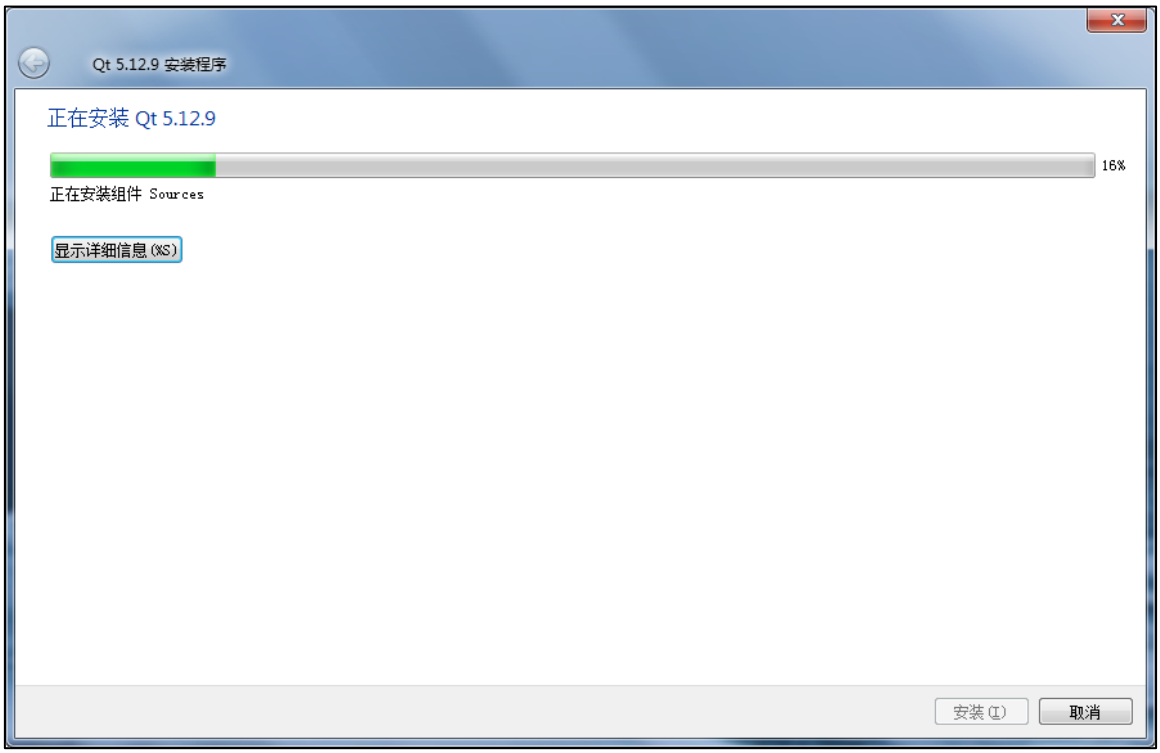
点击下一步。



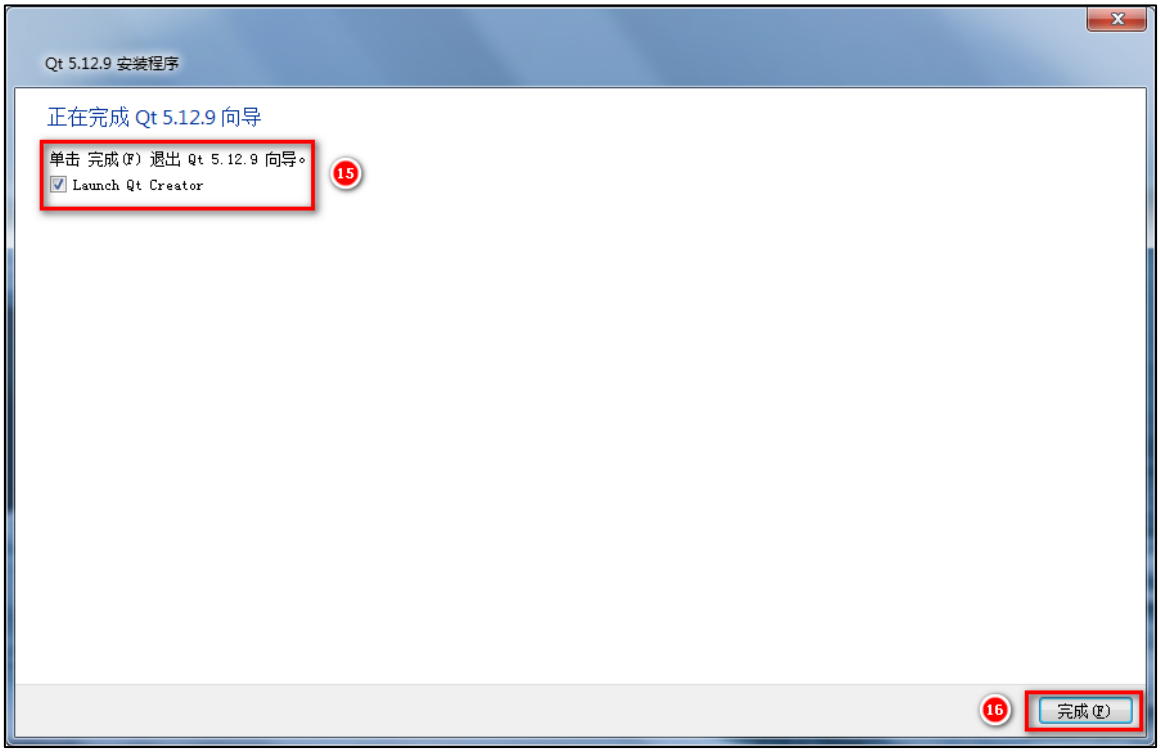
准备安装 Qt，可以看到我们安装的组件越多占用的安装空间就会越大。接近 10G。


Qt 5.12.9 安装程序
准备安装
安装程序现已准备好在您的计算器中安装 Qt 5.12.9。 Installation will use 9.40 GB of disk space.
14
安装(I)
取消

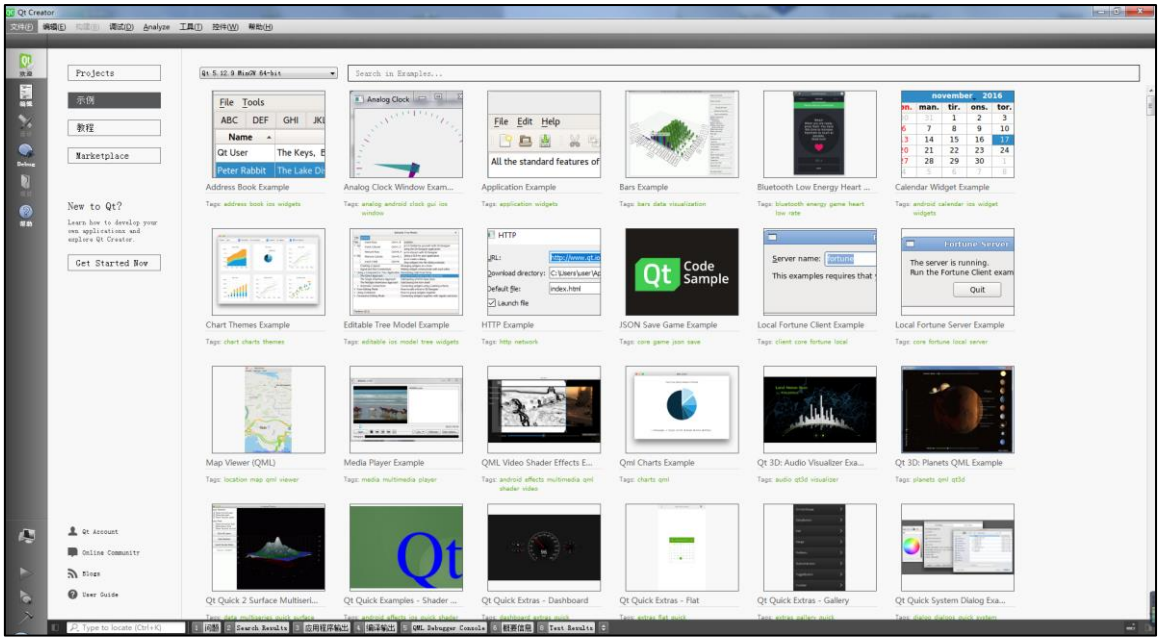
正在安装 Qt。


Qt 5.12.9 安装程序
正在安装 Qt 5.12.9
16%
正在安装组件 Sources
显示详细信息(S)
安装(I)
取消

安装完成，勾选启动 Qt Creator，点击完成。



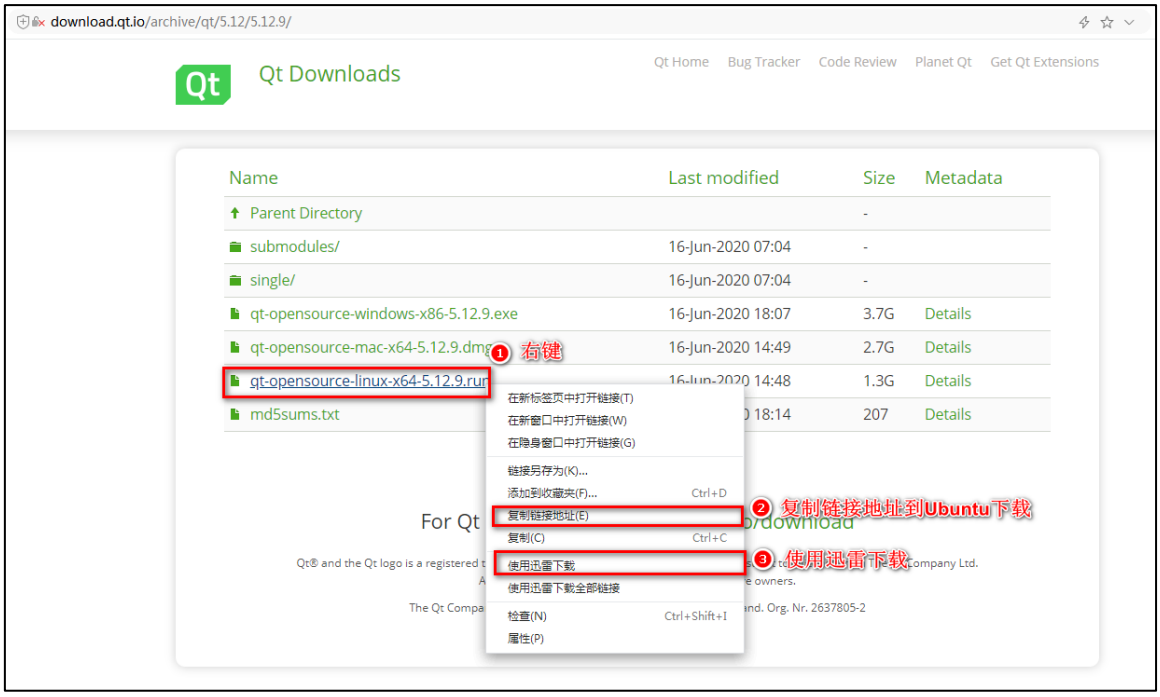
安装完成打开 Qt Creator 的界面如下。



3.4 Linux 下安装 Qt

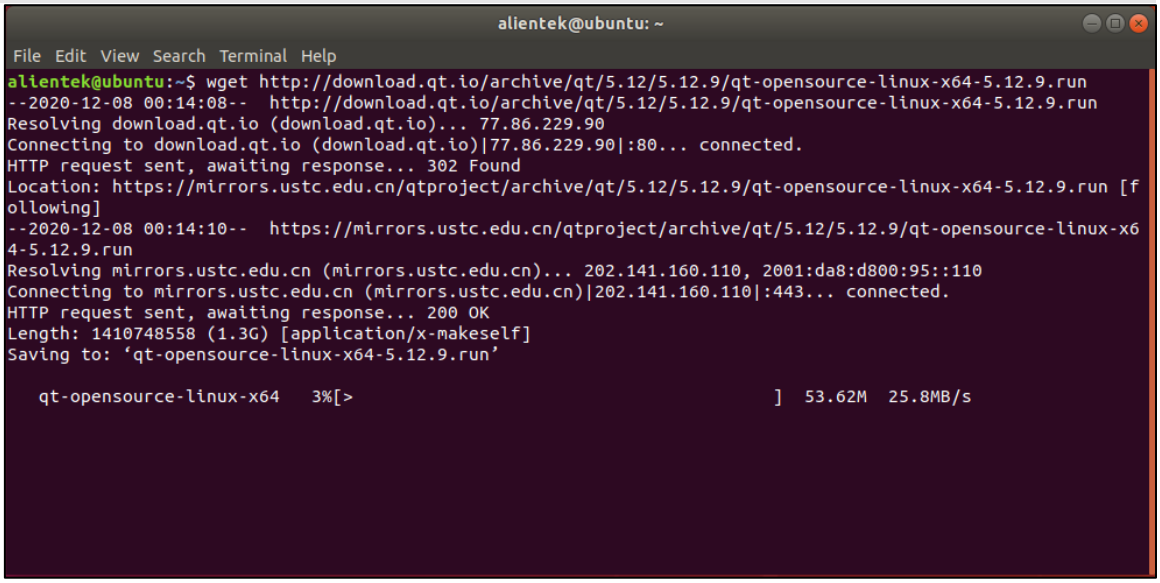
3.4.1 安装 Qt

同样地，进入 http://download.qt.io/archive/qt/5.12/5.12.9/下载页面（注意如果找不到下载链接，我们就进行 http://download.qt.io 这个顶层目录一个个目录找，因为 Qt 下载链接会变动），选择 Linux 的安装包下载。使用迅雷下载再拷贝过去 Ubuntu 虚拟机或者直接复制链接地址到 Ubuntu 虚拟机下载。



如下图，复制链接下载地址到 Ubuntu 虚拟机终端下使用指令 wget 下载，http://download.qt.io/archive/qt/5.12/5.12.9/qt-opensource-linux-x64-5.12.9.run 。

```
wget http://download.qt.io/archive/qt/5.12/5.12.9/qt-opensource-linux-x64-5.12.9.run
```

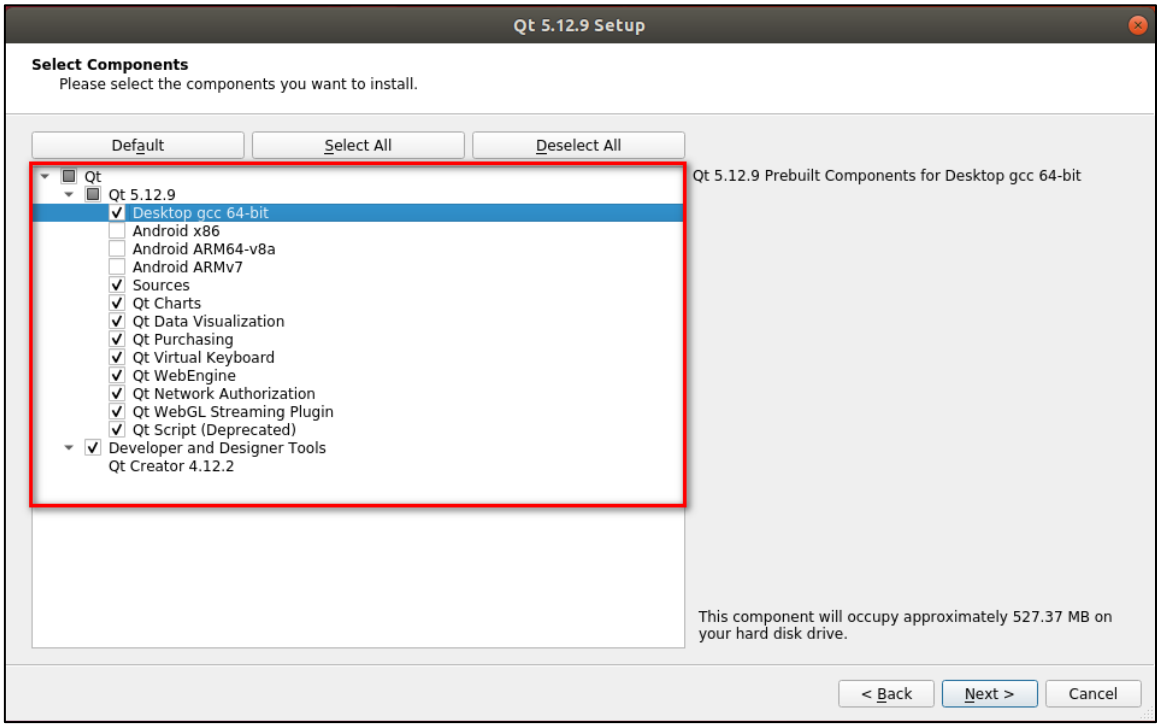


赋予可执行权限，加上 sudo 权限进入安装，这样会安装在/opt 目录下。

```
chmod +x qt-opensource-linux-x64-5.12.9.run
sudo ./qt-opensource-linux-x64-5.12.9.run
```

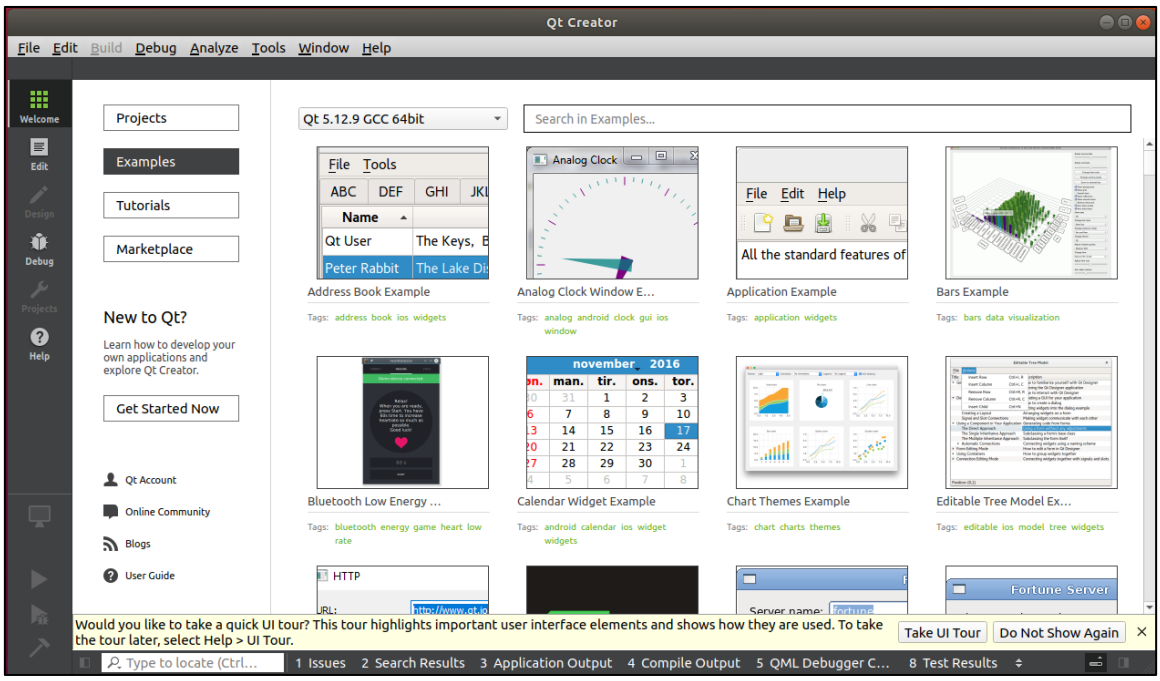
```
alientek@ubuntu:~$ chmod +x qt-opensource-linux-x64-5.12.9.run
alientek@ubuntu:~$ sudo ./qt-opensource-linux-x64-5.12.9.run
```

执行安装指令后，将会弹出 Qt 的安装界面，这与 Windows 下的 Qt 安装步骤一样，请参阅 3.3 小节，安装选择目录时，默认安装目录即可。安装组件选择如下。



安装完成，在左下角应用程序中心找到 Qt Creator，点击打开 Qt Creator。

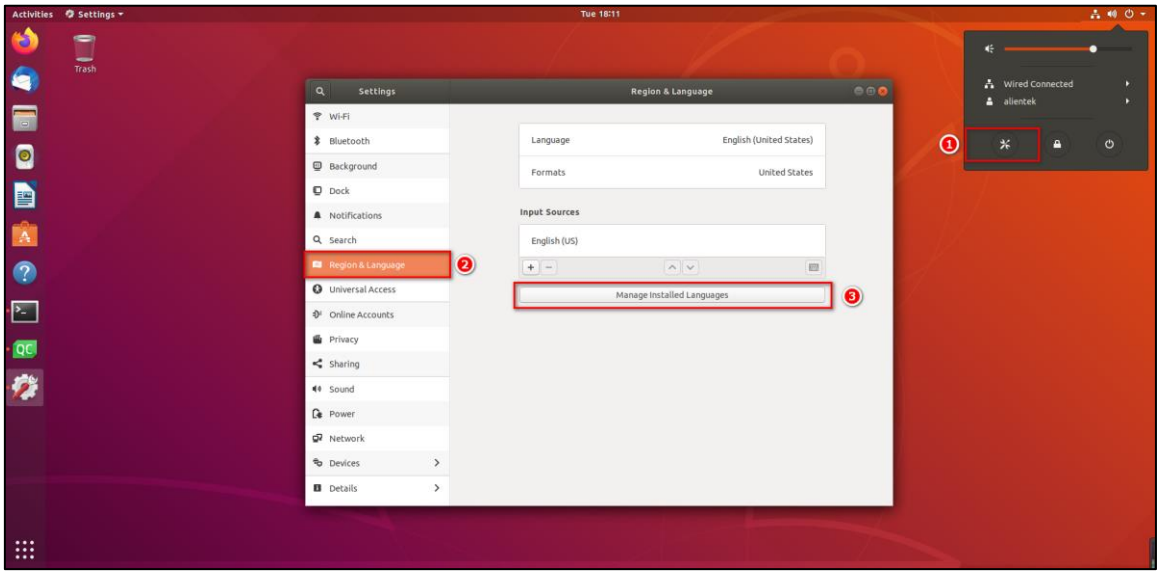
打开 Qt Creator 的界面如下，安装完成。



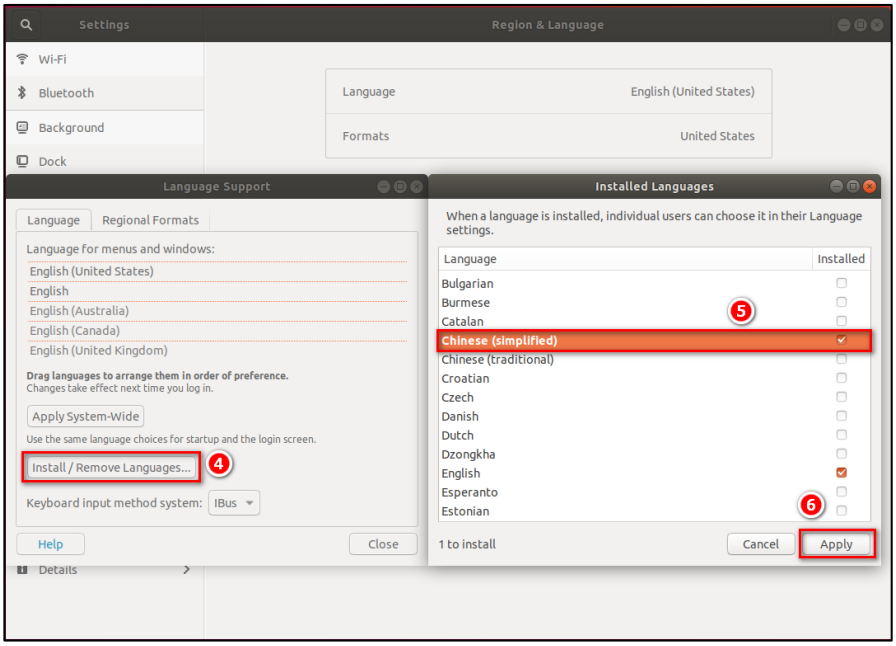
3.4.2 配置 Qt Creator 输入中文

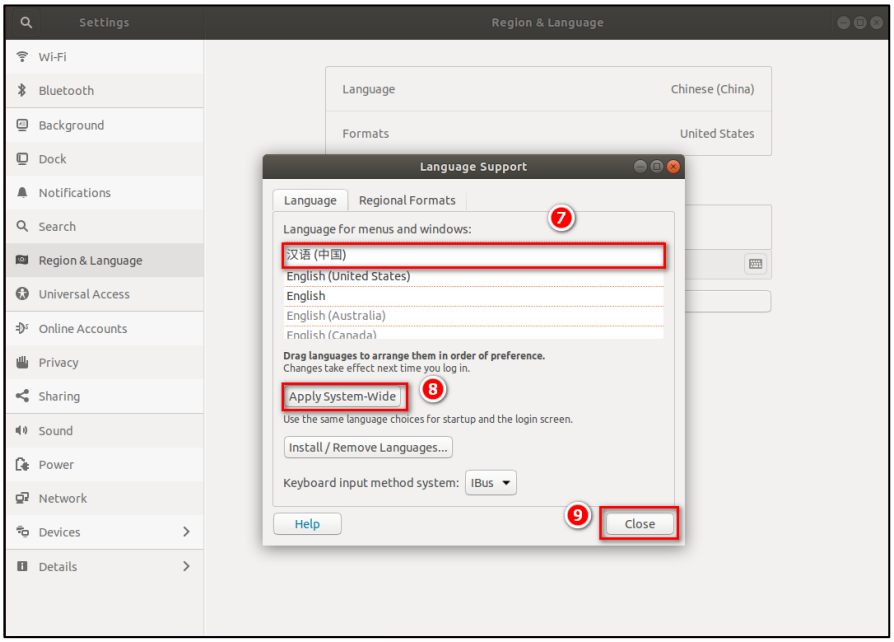
3.4.2.1 配置 Ubuntu 中文环境

Ubuntu 的默认安装环境是使用的语言是英文，很多同学不习惯。现在我们将 Ubuntu 的系统语言设置成中文。在 Ubuntu 右上角，点击设置图标如下图第①步。

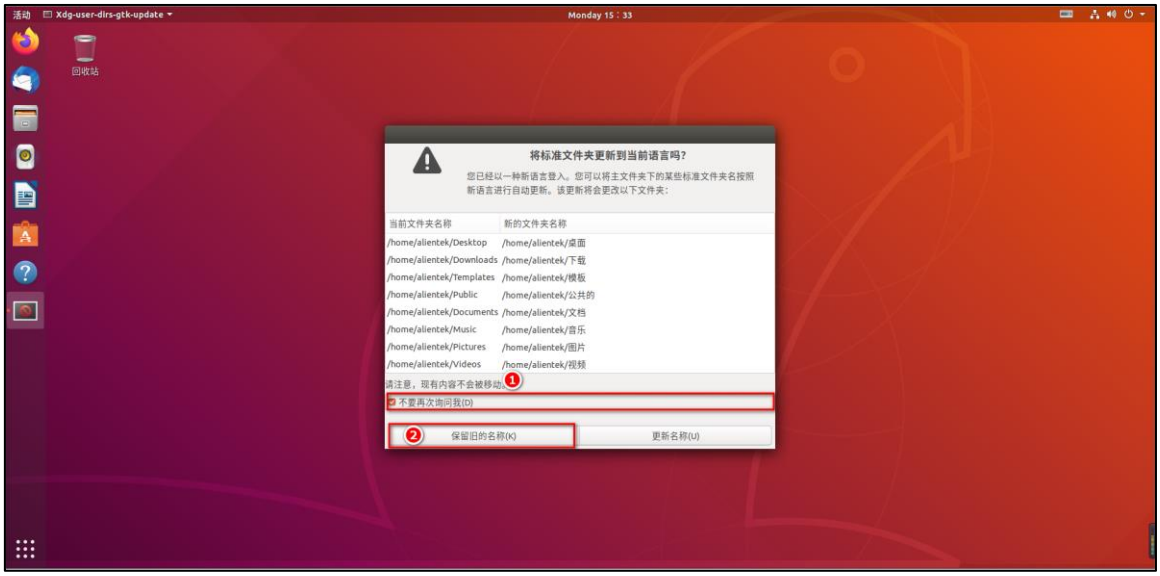


按如下图设置，点击（install/Remove Languages …）安装或者移除语言，在安装语言处选择简体中文，点击 Apply 应用即可。





配置完成后，点击重启（或者注销 Ubuntu），重启后，因为我们已经更新了系统的语言，Ubuntu 询问我们需不需要将系统文件夹的名称也改成中文。这里作者选择否，保留旧的名称。保留旧的名称有一定的好处，就是我们进入这些访目录时，直接使用英文，不用切换到中文输入法。严格来说，最好是统一用英文环境开发了。这里为了初学者或者有强迫中文者，所以我们这里需要配置中文的环境，及后期开发 Qt 需要写中文注释，方便理解与给后人看。



3.4.2.2 配置中文输入法

在上面我们已经配置好中文环境，并有拼音输入法 ibus，但 ibus 并不好用，Qt Creator 不支持 ibus 输入中文。好的生产工具决定好的生产力，下面我们介绍一下 Fcitx 输入法。

Fcitx (Flexible Input Method Framework) ——即小企鹅输入法，它是一个以 GPL 方式发布的输入法平台，可以通过安装引擎支持多种输入法，支持简入繁出，是在 Linux 操作系统中常用的中文输入法。它的优点是：短小精悍、跟程序的兼容性比较好。

Fcitx 内置的输入法支持中文 拼音 和基于字符表的输入(例如五笔)，根据语言的不同，有不同的输入法引擎可以选择。

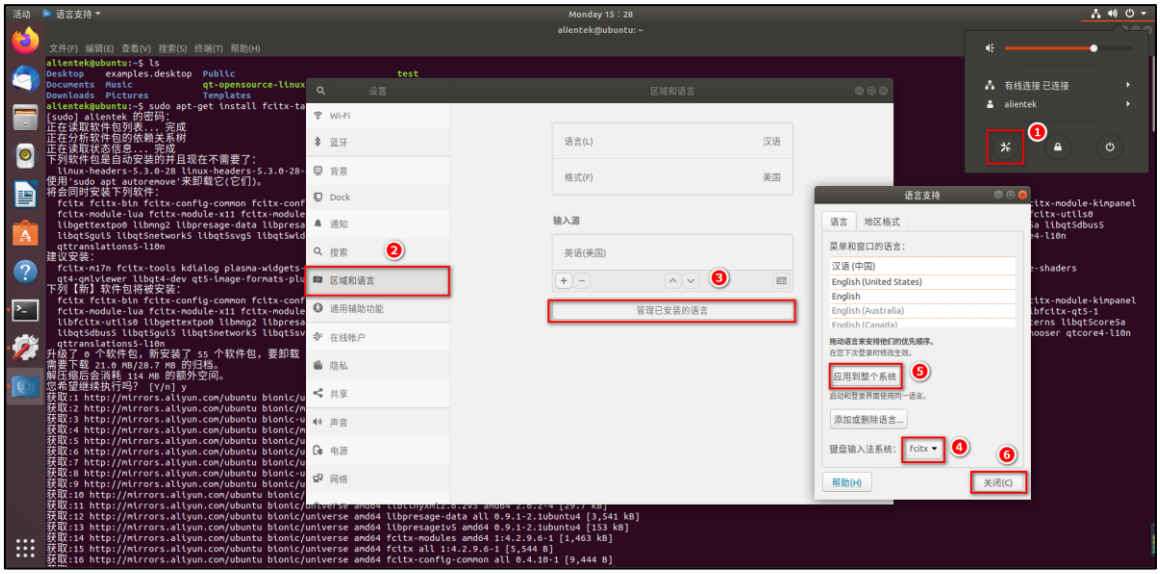
在 Fcitx 支持的拼音输入法中，内置拼音响应速度最快。Fcitx 同样支持流行的第三方拼音输入法以提供更好的整句输入效果.

- fcitx-sunpinyin 在输入速度和输入精度之间有较好的平衡。
- fcitx-libpinyin 算法比 sunpinyin 先进。
- fcitx-rime, 即著名中文输入法 Rime IME 的 Fcitx 版本。但它不支持 Fcitx 本身的 #特殊符号和 #快速输入功能，自定义设置请参见官方（http://rime.im/）。
- fcitx-googlepinyin, Google 拼音输入法 for Android.
- fcitx-sogoupinyin（AUR,） 搜狗输入法 for linux—支持全拼、简拼、模糊音、云输入、皮肤、中英混输入。官方网址（http://pinyin.sogou.com/linux/）。
- fcitx-cloudpinyin 可以提供云拼音输入的支持，支持 Fcitx 下的所有拼音输入法，Fcitx-rime 除外。
- fcitx-chewing 为 Fcitx 添加 chewing (繁体中文注音) 输入引擎支持。依赖 libchewing。
- fcitx-table-extra adds Cangjie，Zhengma，Boshiamy support。

安装 Fcitx 输入法，下面主要介绍两种输入法（五笔输入法与拼音输入），本次以安装五笔输入法为例。

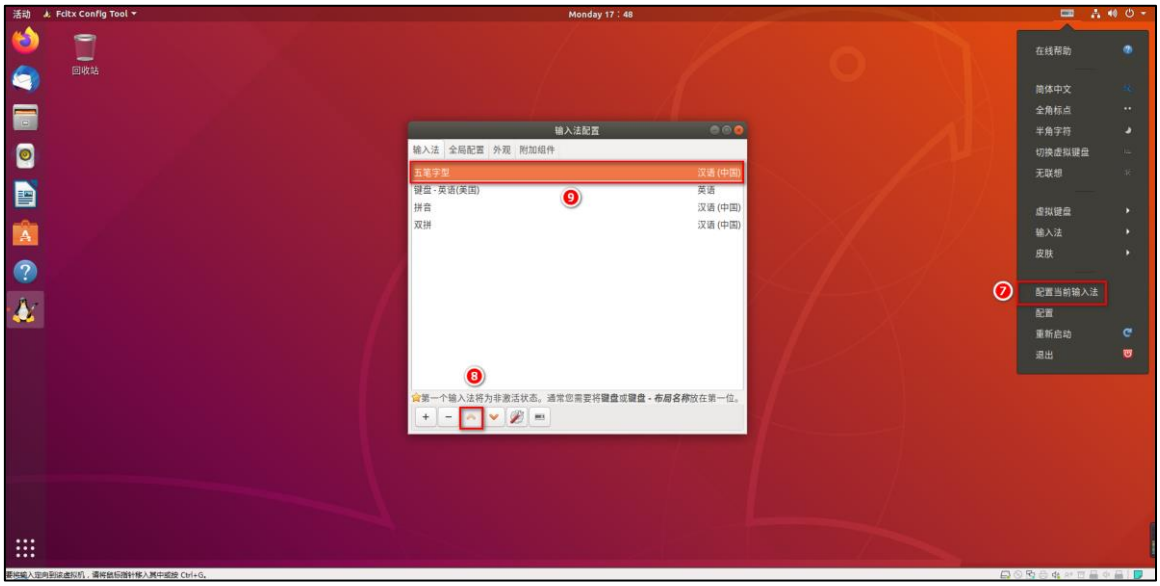
```
sudo apt-get install fcitx-table-wubi          // 五笔输入法
sudo apt-get install fcitx-sunpinyin       // 拼音输入法请装 sunpinyin 或者 fcitx-googlepinyin
```

再到右上角点击系统设置，找到语言支持，将键盘输入方式系统点击下拉复选框选择为 fcitx，然后点击应用到整个系统，再关闭。如下图步骤。



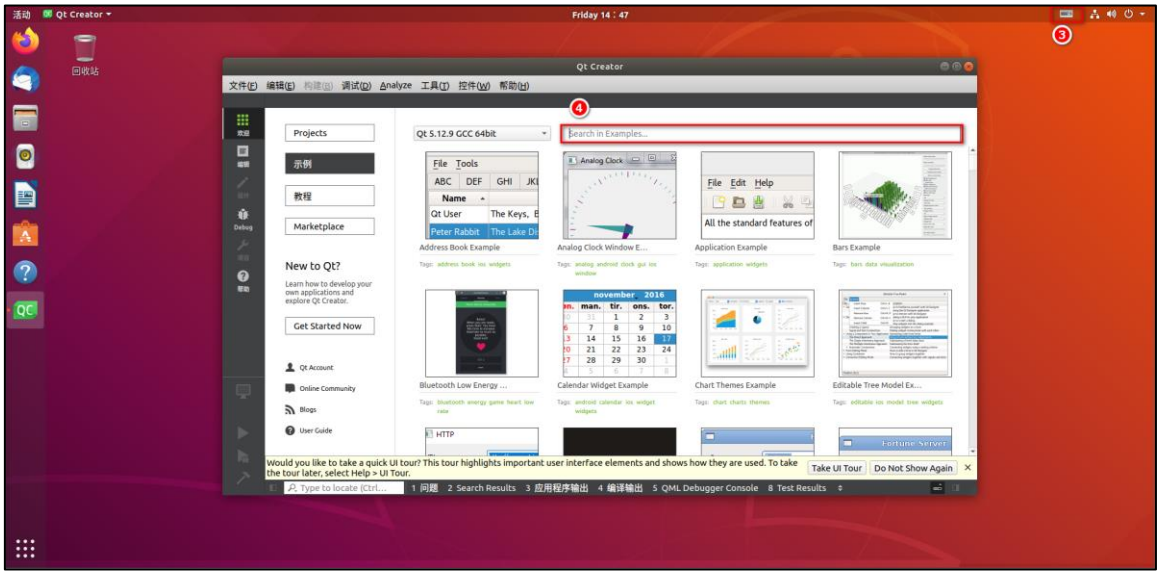
为了确保刚配置的环境生效，完成以上步骤后重启 Ubuntu 系统。

按如下步骤，在弹出的“输入法配置”对话框中，如果没有出现五笔字形输入法，点击左下角的“+”号按钮将五笔字形添加到输入法列表中，再点击第 8 步的“^”将五笔字型添加至最上方。



打开 Qt Creator，在左下角“显示应用程序”，找到 Qt Creator 图标单击打开 Qt Creator。

此时右上角的输入法标志还是英文输入法，无法在④处输入中文，按 Ctrl+Shift 也切换不到中文输入法状态下。



作者为解决 Linux Qt Creator 不能输入中文的问题，已经编译过 fcitx 插件，已经上传到 Gitee 仓库里，大家可以下载直接使用。

如果您没有安装 Git，请先使用下面的指令安装 Git。

```
sudo apt-get install git
```

使用下面的指令下载 Qt Creator 插件，本插件支持 Qt5，Qt4 没有测试过。本插件曾经在 Qt5.5.1 上测试过，后来升级了 Qt 版本到 Qt5.12.9。Qt5.5.1 与 Qt5.12.9 这两个 Qt 版本的安装目录插件路径有变动（指版本相差大，安装路径有变动）。从 Gitee 下载的插件适用于 Qt5.12 版本左右版本。如果使用其他版本请自行修改 fcitx-install.sh 脚本里的安装路径！

```
git clone https://gitee.com/QQ1252699831/fcitx-qt5-1.1.1.git
cd fcitx-qt5-1.1.1
```

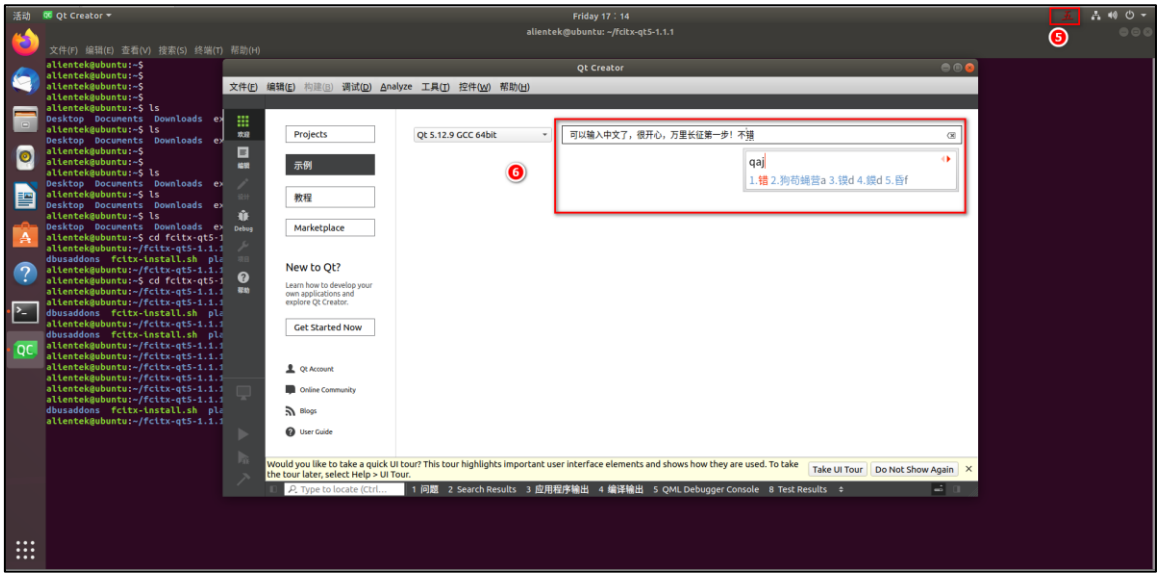
```
alientek@ubuntu:~$ git clone https://gitee.com/QQ1252699831/fcitx-qt5-1.1.1.git
正克隆到 'fcitx-qt5-1.1.1'...
remote: Enumerating objects: 82, done.
remote: Counting objects: 100% (82/82), done.
remote: Compressing objects: 100% (77/77), done.
remote: Total 82 (delta 25), reused 0 (delta 0), pack-reused 0
展开对象中: 100% (82/82), 完成.
alientek@ubuntu:~$ cd fcitx-qt5-1.1.1/
alientek@ubuntu:~/fcitx-qt5-1.1.1$ ls
dbusaddons  fcitx-install.sh  platforminputcontext  README.en.md  README.md  widgetsaddons
alientek@ubuntu:~/fcitx-qt5-1.1.1$ _
```

直接执行脚本安装，原理是拷贝 fcitx-qt5-1.1.1 文件夹下里面的插件到 Qt Creator 的安装目录下。请注意个人安装 的 Qt Creator 路径，一定要与作者安装的路径一样，否则需要自己修改 fcitx-install.sh 脚本里的路径！

```
./fcitx-install.sh
```

```
alientek@ubuntu:~/fcitx-qt5-1.1.1$
alientek@ubuntu:~/fcitx-qt5-1.1.1$ ./fcitx-install.sh
[sudo] alientek 的密码:
alientek@ubuntu:~/fcitx-qt5-1.1.1$
```

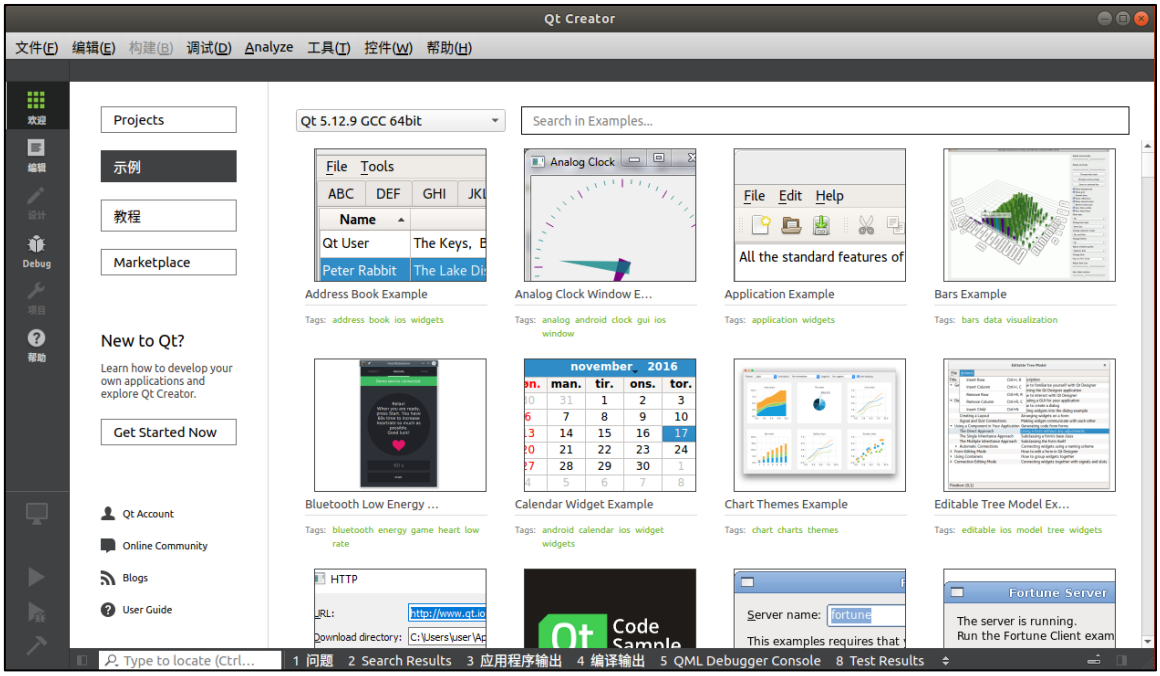
此时再重新打开 Qt Creator 方可输入中文！如下图。需要按 Ctrl+Space(空格键)激活输入法，再按 Ctrl+Shfit 切换输入法到五笔输入法。



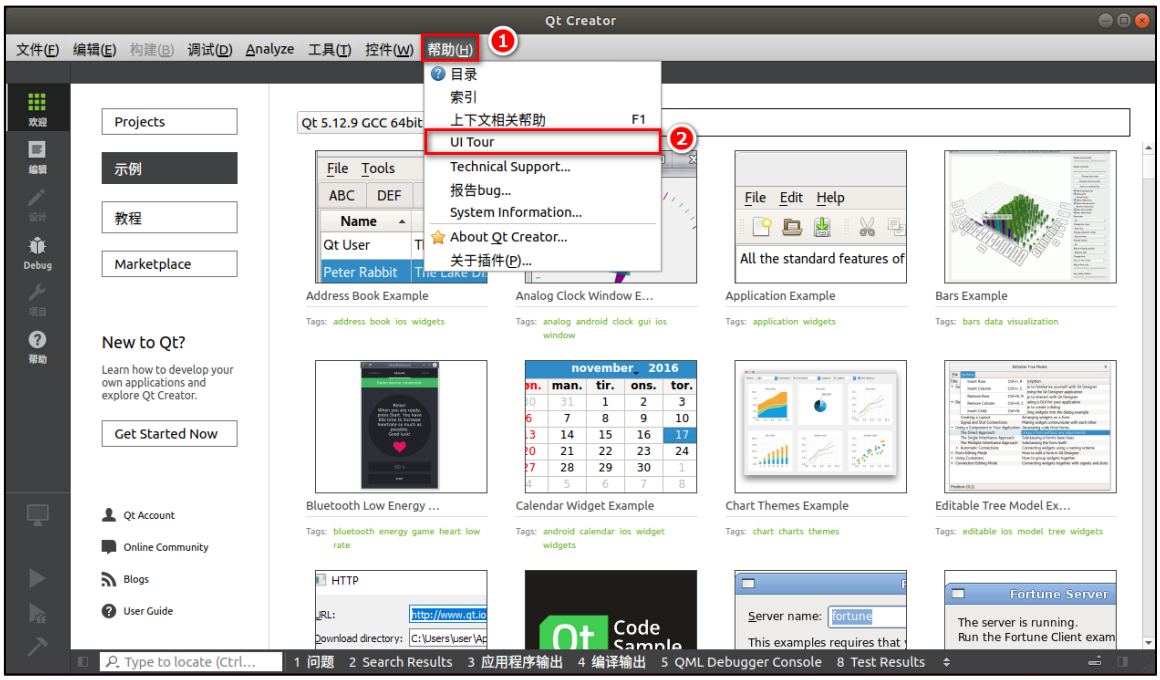
3.5 Qt Creator 简单使用

3.5.1 Qt Creator 界面组成

启动 Qt Creator 后，Qt Creator 的主界面如下图，默认打开的是欢迎页面。可以看到 Qt Creator 里自带很多示例。在 Ubuntu 里，由于 Qt Creator 安装在/opt 目录下，这个目录普通用户是没有权限写的，只能够读。如果要打开示例先点击后选择“复制项目并打开”。作者比较喜欢 Qt Creator 界面设计的。十分简洁，还自带示例，不会写也会参考！最重要的是 Qt Creator 里**左侧栏**的“帮助”按钮，有很多使用说明，总结的非常好，对学习 Qt 的类有很大帮助，如果学习到某个方法或者类不会用，可以打开这个“帮助”来搜索这个类或者方法的用法。没有任何一本教程能有 Qt 帮助文档这么详细了，可惜是英文的，初学者学起来还是会一定难度。多多参考这个 Qt 帮助文档，因为那里能学到很多，例子说明也特别多，每个方法、类、信号与槽等都有详细的解释。



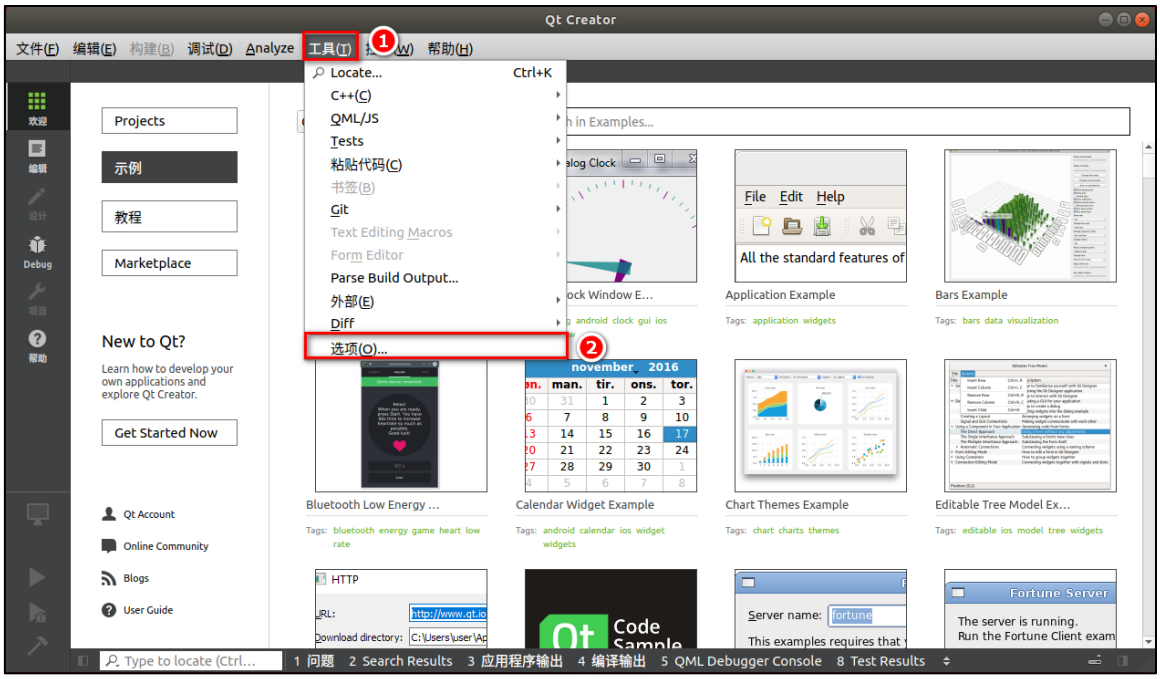
单击**顶部**工具栏的“帮助”菜单，点击“UI Tour”会出现介绍 Qt Creator 各组件的说明。这个大家一定要看看，虽然是英文的说明，但是大体有个了解。这里就不再截图说明了。后面编写第一个 Qt 程序也会说明整个项目编写及编译运行的流程。界面组成我们只需要了解这么多。



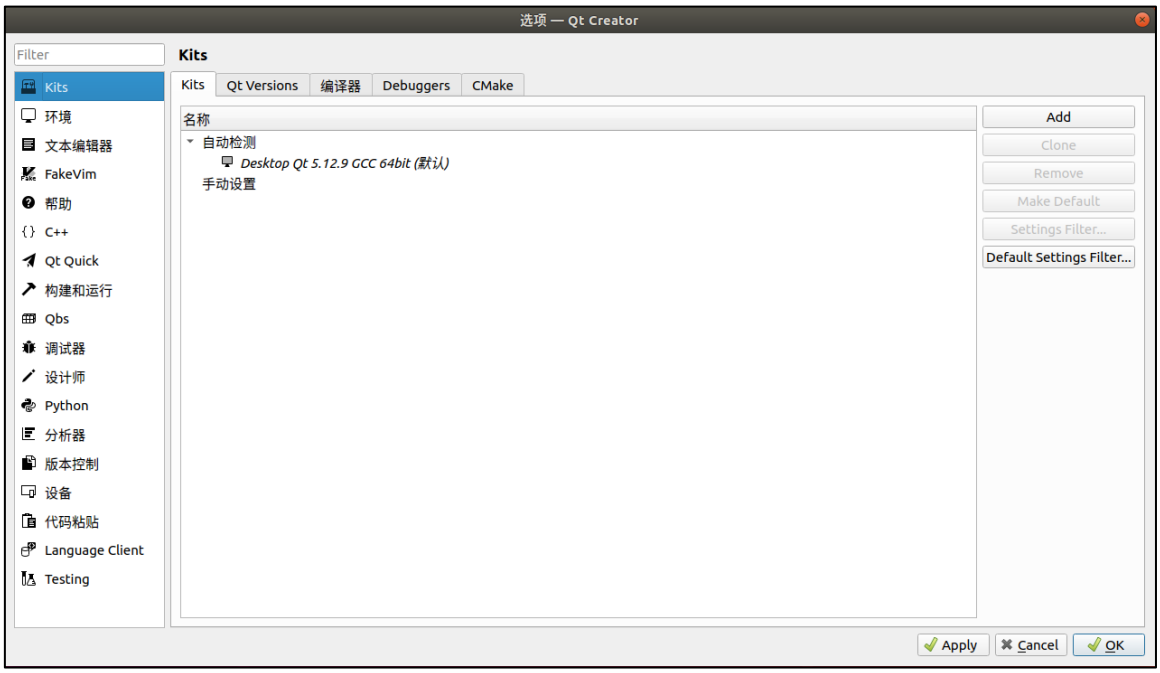
3.5.2 Qt Creator 设置

Qt Creator 里的设置比较重要。在 Qt Creator 的设置里我们可以做些什么呢？我们一般会设置字体的大小、颜色和背景颜色等这样看起来比较炫，或者比较符合自己的风格。Qt Creator

里也有很多主题，拥有好多种搭配，相信有一种是您喜欢的，如果不喜欢可以自定义里面的字体大小、颜色和背景颜色等。点击顶部的菜单栏的“工具”》“选项”。



打开选项后可以看到如下图的界面。



下面我们主要介绍常用的几项：

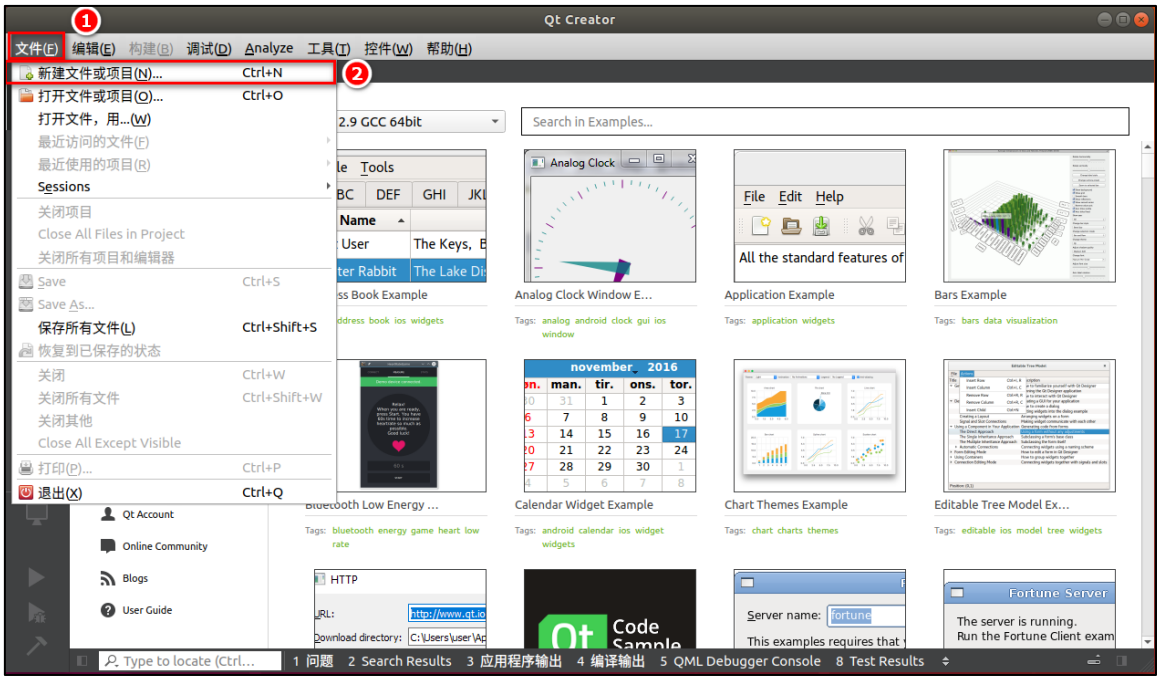
1. Kits：主要显示的是编译工具。我们在 Ubuntu 安装 Qt Creator 时，在安装选项里已经勾选了 Desktop Qt5.12.9 GCC 64bit 这个选项。所以在 Kits 这个页面就能检测到安装的编译工具。还有 Qt Versions 等项都可以自由查看。重要的是我们可以在这个 Kits 里配置 ARM 平台的编译工具（后面以某个 ARM 板卡平台为教程时有讲到如何配置），之所以 Qt 能够跨平台，是因为 Qt 有不同平台的编译工具。
2. 环境：在这个项里可以设置不同的主题和语言等。Qt Creator 默认的主题和系统语言即可。
3. 文本编辑器：可以设置文本编辑器的字体大小、颜色等。还可以设置某些类型的字体颜色，如关键字、字符串和注释等。
4. 构建和运行：常用的是设置项目的目录。其他一般不用修改，默认即可。

3.6 第一个 Qt 程序

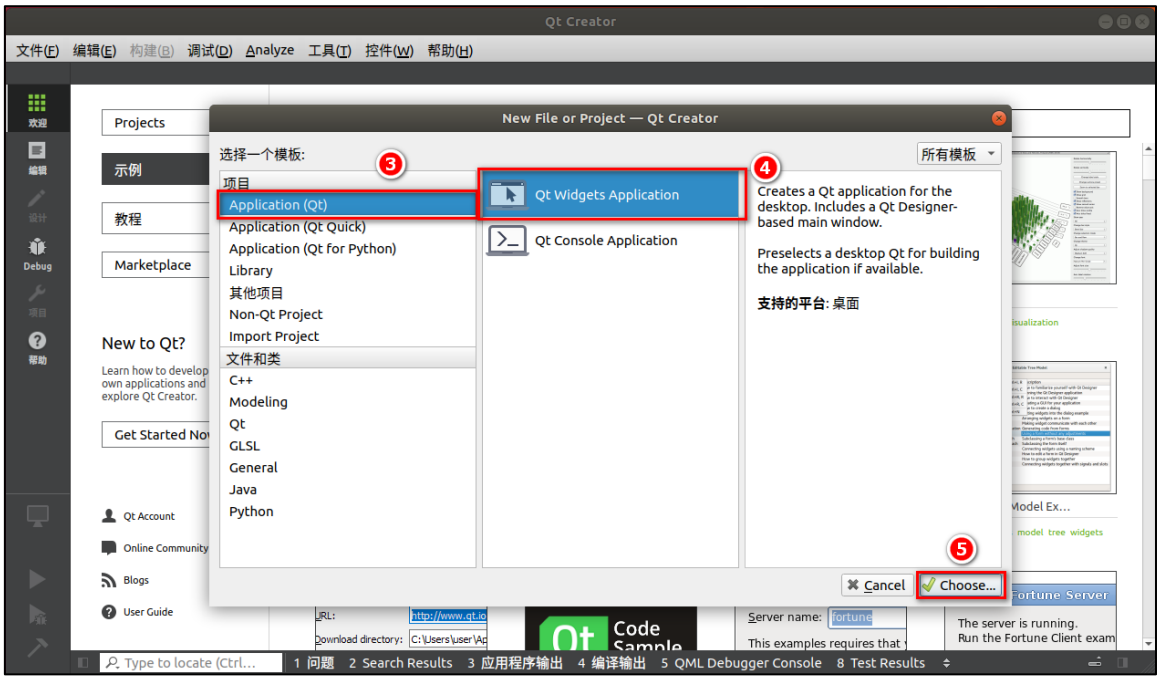
“hello world ”的起源要追溯到 1972 年，贝尔实验室著名研究员 Brian Kernighan 在撰写“B 语言教程与指导(Tutorial Introduction to the Language B)”时初次使用（程序），这是目前已知最早的在计算机著作中将 hello 和 world 一起使用的记录。第一个程序都是以“hello,world”作为默认的第一个程序，我们延续这种经典的做法吧，感受程序的伟大。同时可以让我们初步了解 Qt 项目搭建的基础流程。

3.6.1 新建一个项目

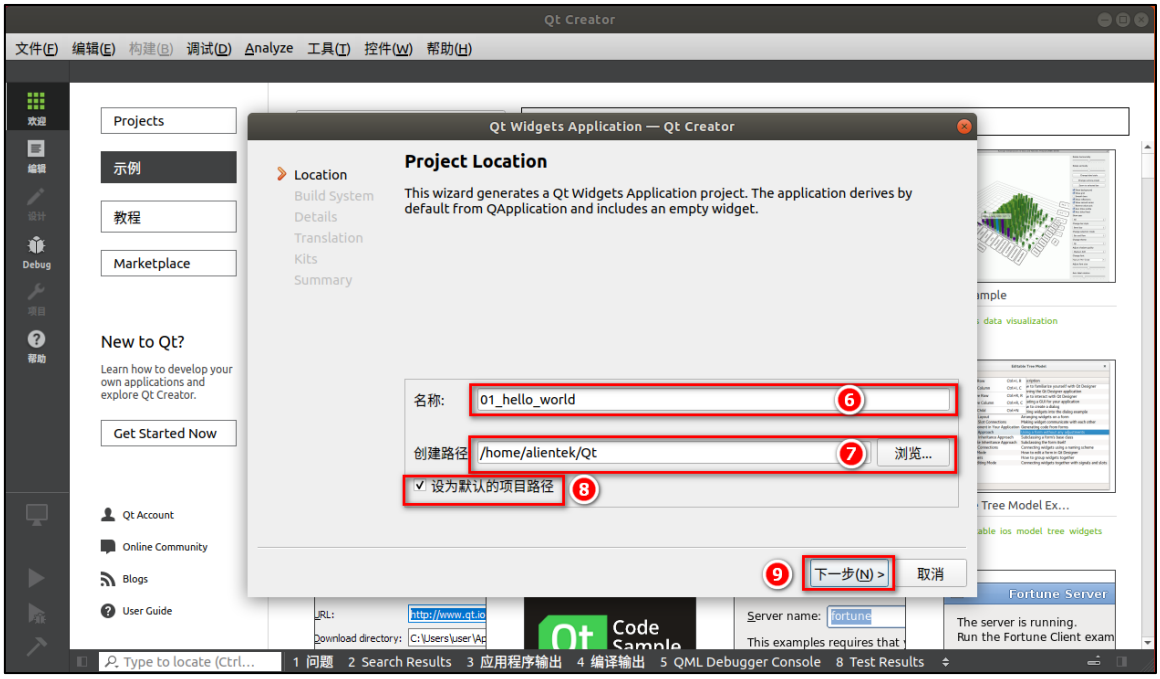
在 Ubuntu18 里打开 Qt Creator，也就是左下角软件中心处点击后，找到 Qt Creator 的图标后点击打开。单击文件 Qt Creator 的文件，选择新建文件或者项目。注意有快捷键 Ctrl + N。直接在 Qt Creator 激活状态和英文状态的输入法下使用“Ctrl + N”也可以快速打开新建项目。



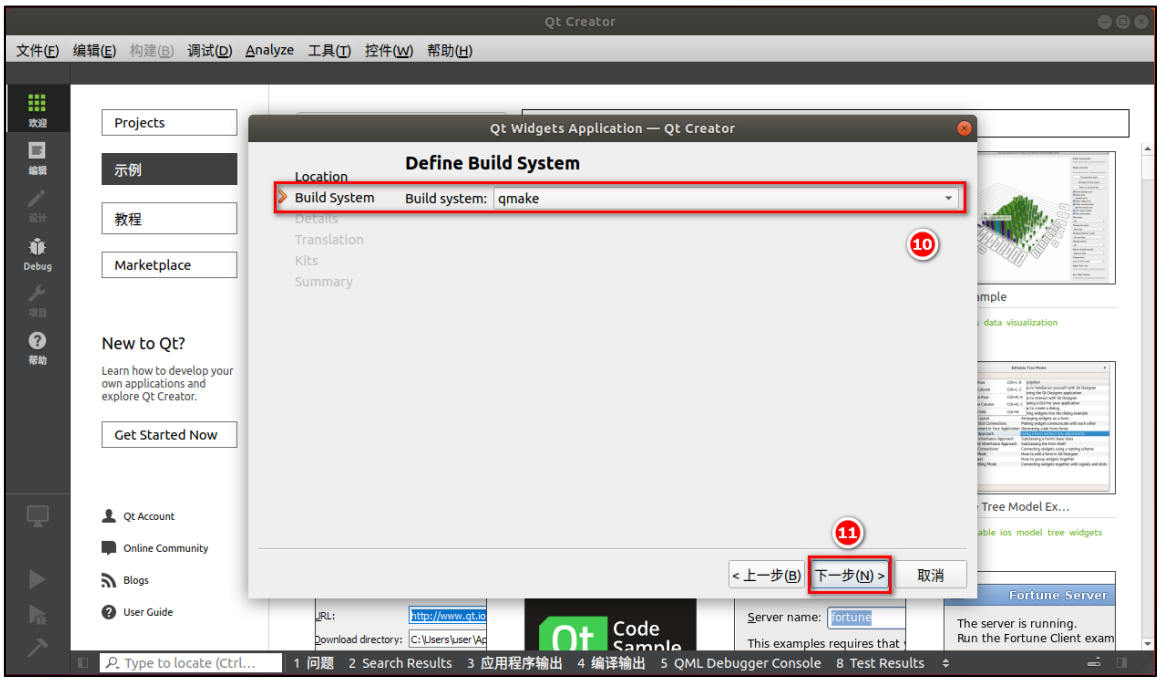
弹出的新建项目如下图，这里我们可以看到有很多模板（包括项目模板和文件和类模板）可以使用，包括 Qt，Qt Quick，Qt for Python，…，C++等等。作为初学者我们选择第一个 Application(Qt)和 Qt Widgets Application，所谓的模板就是 Qt 为了方便开发程序，在新建工程时可以让用户基于一种模板来编写程序，包括 cpp 文件，ui 文件都已经快速的创建，而不用用户手动创建这些文件。这样对用户的开发带来极大的便捷。当然用户可以可以自己手动创建项目，一个一个往里面加也是可以的。但是初学者不建议这么做，我们的目的是先体验一遍项目搭建的流程，等日后有空再自己回头尝试这么做吧！



如果您的输入法现在处于中文输入法，先按 Ctrl 再按 Space（空格）切换成英文输入法，在名称处输入项目为“01_hello_world”，这里的项目路径为笔者个人的家目录路径下的 Qt 目录下，勾选设为默认的项目路径，这样以后做项目实验时都是默认选择这个目录作为项目路径而不用自己手动选择路径了。选择下一步。



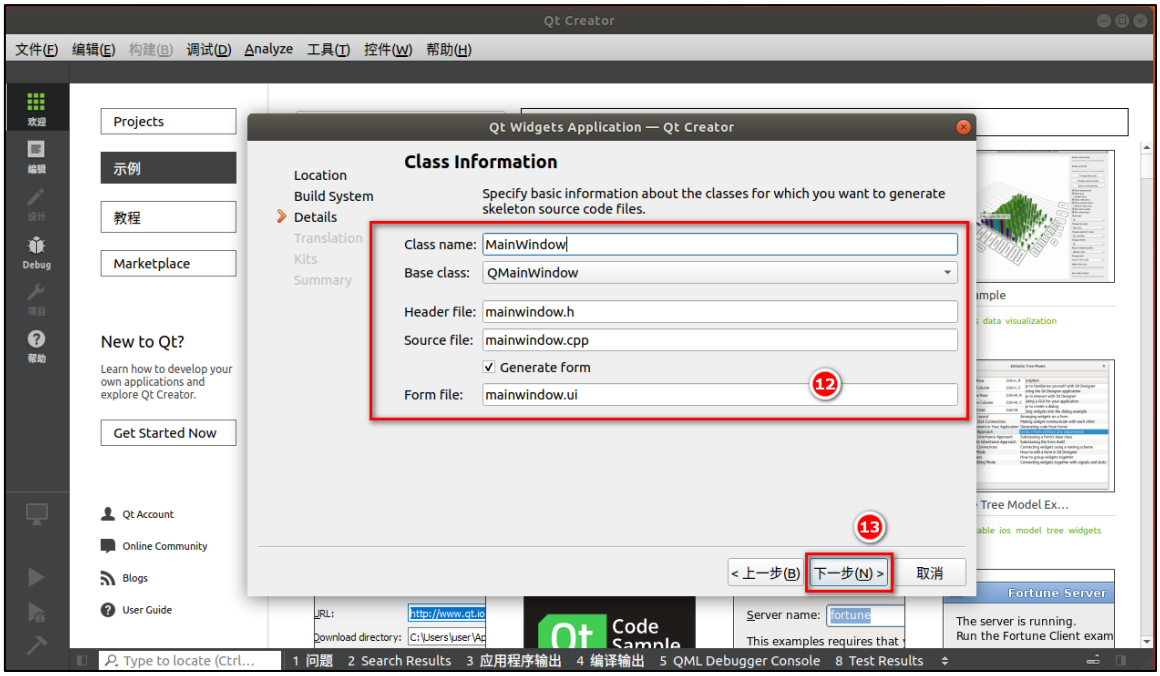
默认已经是选择 qmake 编译，主要用 qmake 生成 Makefile 用于项目的编译。点击下一步即可。



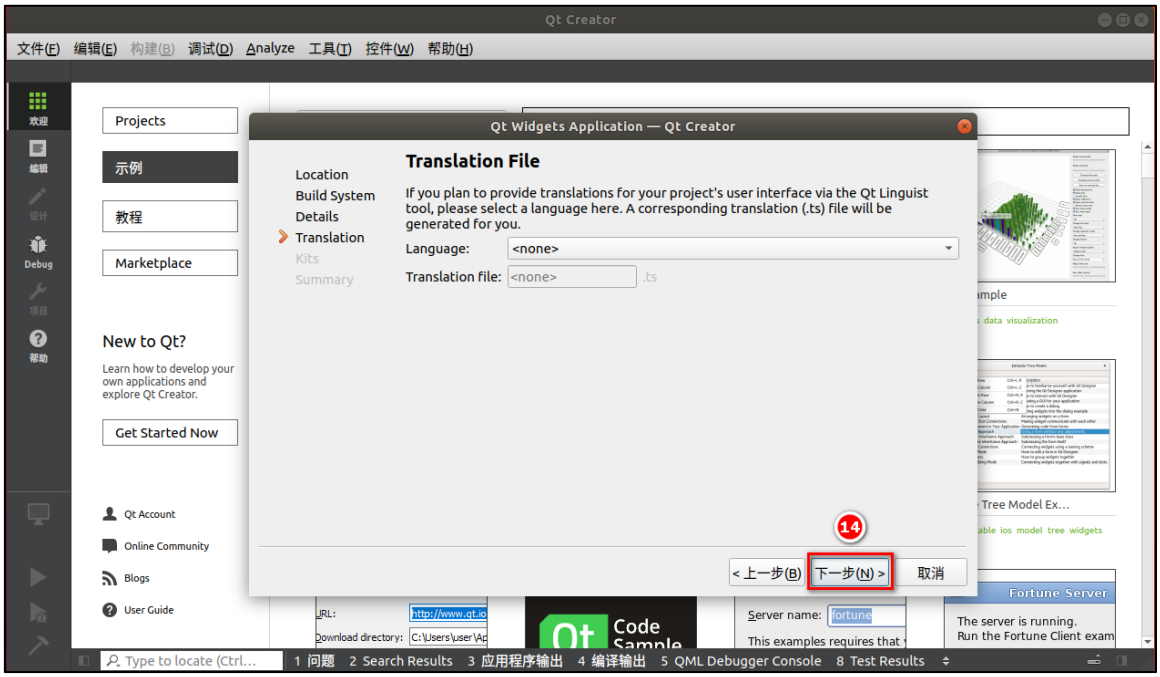
这里默认选择的基类为 QMainWindow。在 Base class 一项中我们还可以看到还有 QWidget 和 QWialog 这样的基类可以选择。在 C++篇我们已经学习什么叫基类，简单的来说，我们创建的这个项目是基于 QMainWindow 类去开发的。默认勾选“Generate form”，意思是生成 ui 窗体文件 mainwindow.ui。为了学习方便，我们统一默认基类为 QMainWindow，但是注意，在嵌入式里一般不需要标题栏，状态栏等，所以常用的是 QWidget 基类。

- QMainWindow：主窗口类，主窗口具有主菜单栏、工具栏和状态栏。类似于一般的应用程序的主窗口。如果您想做个嵌套的窗口程序开发的软件，不妨选择这个 QMainWindow。

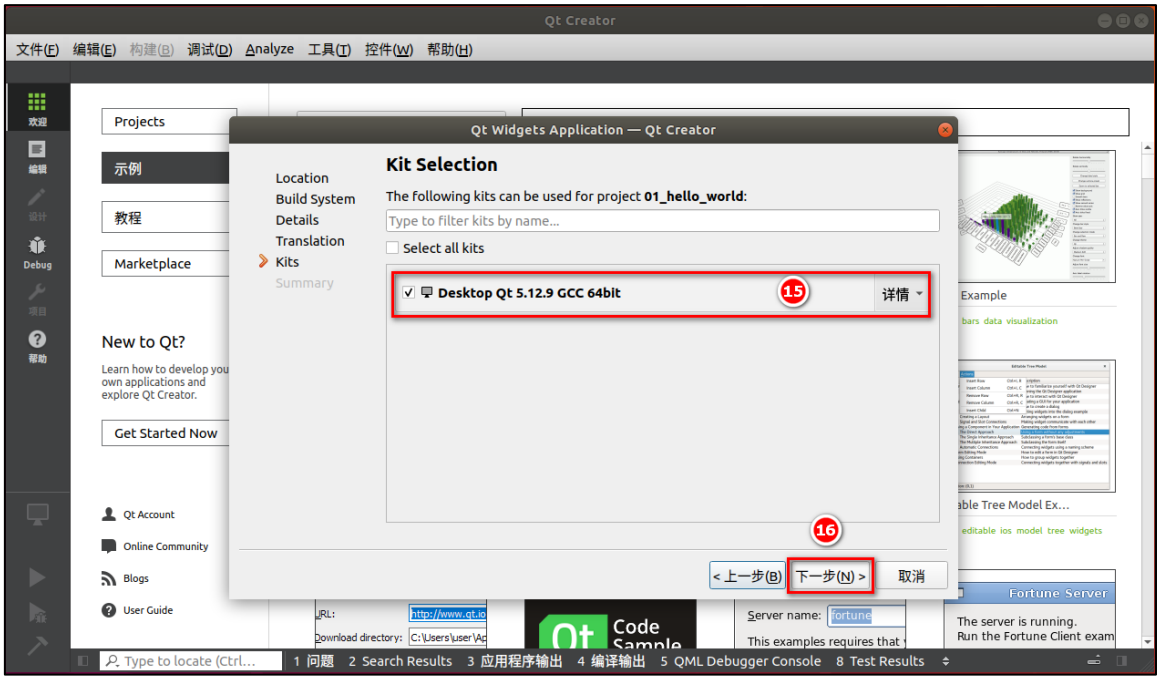
- QWidget：是可视界面类的基类，也就是说 QMainWindow 类也是由 QWidget 继承封装而来。所以 QWidget 要比 QMainWindow 功能少一些。
- QDialog：对话框类，建立一个对话框界面。比较少使用此项作为基类。一般以 QMainWindow 和 QWidget 作为基类的居多。注因为 QWidget 不带窗口标题栏等，嵌入式里最好 QWidget。



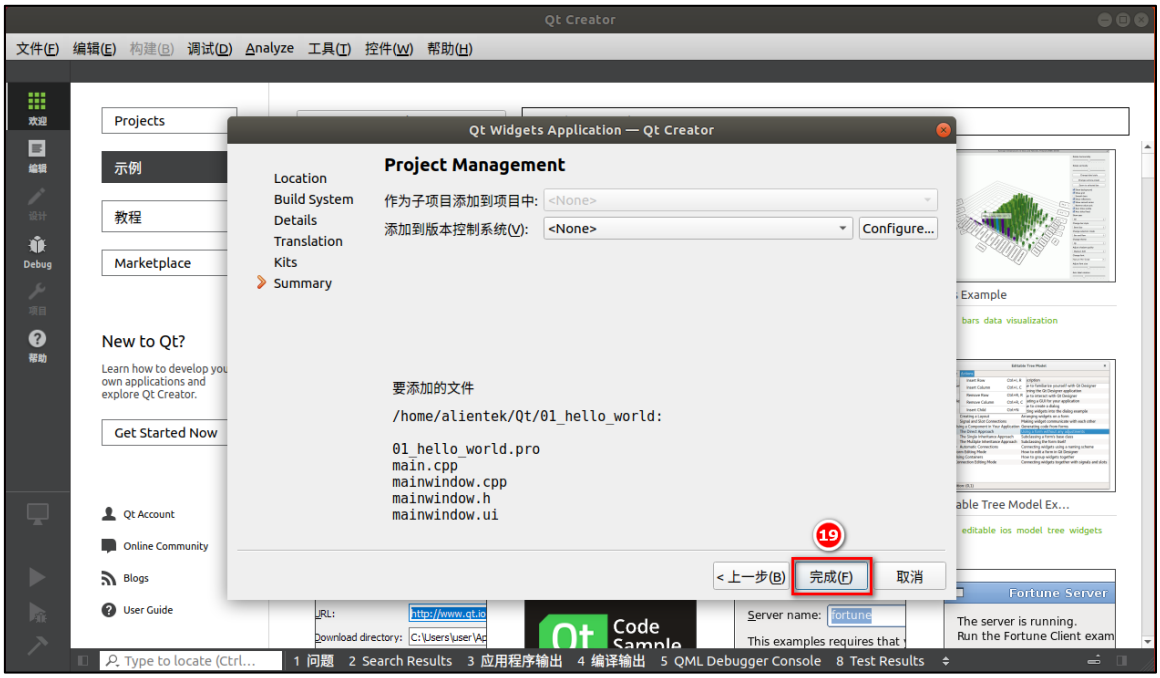
在高版本的 Qt Creator 里，有 Translation 这一项，Qt 提供了一个（.ts）的文件给您，.ts 是可读的翻译文件，使用简单的 XML 格式。我们暂时只需要知道这个东西是个可读的翻译文件即可。极少需要使用到。点击下一步。



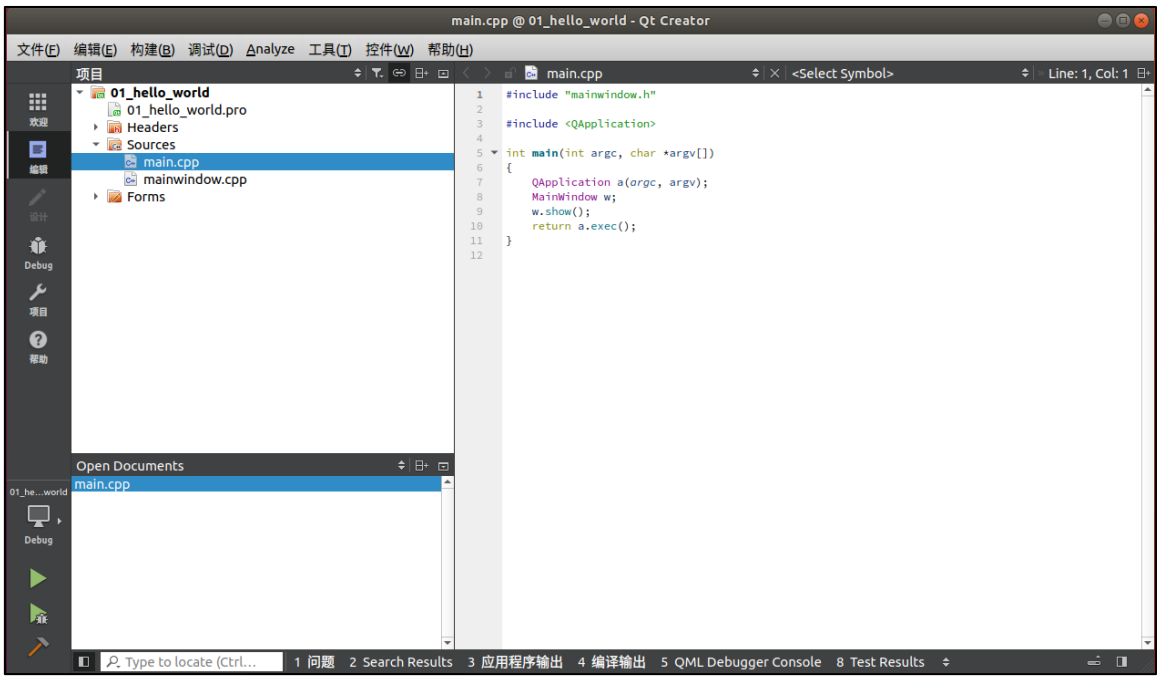
勾选编译器，这个编译器是我们在安装组件时选择的，使用这个编译器可以编译出 Ubuntu 版本上跑的可执行程序。这么一说是不是觉得还能编译出其他平台的可执行程序？没错，假若我们现在有 ARM 平台的 Qt 编译器，那么选择 ARM 平台的 Qt 编译器即可编译出 Qt 在 ARM 平台上的可执行文件(这里说的可执行文件类似 window 的 exe 程序文件一样，直接能够运行)。点击下一步。



到这里，Qt Creator 询问是否使用版本控制，如果您学习过像 Git 这样的版本控制工具，可以选择 Git，将您的项目使用版本控制。默认选择是无版本控制，直接点击完成即可。



新建的“01_hello_world”项目如下。



3.6.2 项目文件介绍

其中，左侧有上下两个子窗口，上面的窗口显示了项目的文件结构，显示当前的项目为“01_hello_world”，细心的还会发现“01_hello_world”是用粗体黑色标明。说明此项目是活动项目，活动项目的项目根节点都是用粗体字体表示的。如果打开了多个项目，那么我们只需要观察哪个是加粗的项目名就表示当前活动项目。

Qt Creator 和其他 IDE 开发软件一样。都是分组管理项目内的各种源文件，下面是项目内的文件简介。

- 01_hello_world.pro 是项目管理文件，这个项目管理文件十分重要，当您加入了文件或者删除了文件，Qt Creator 会自动修改这个*.pro 文件。有时候需要打开这个*.pro 文件添加我们的设置项。
- Header 分组，这个节点下存放的是项目内所有的头文件*.h。
- Source 分组，这个节点下存放的是项目内的所有 C++源码文件*.cpp。
- Forms 分组，这个节点下是存放项目内所有界面文件*.ui。*.ui 文件由 XML 语言描述组成，编译时会生成相应的 cpp 文件，这样交叉编译器就可以编译它了。

3.6.2.1 项目文件*.pro

01_hello_world.pro 文件如下所示。此 pro 文件在高版本的 Qt 与低版本的 Qt 文件有些区别，且在 Windows 与 Linux 系统下生成的 pro 也有些区别，但是变化不大。

01_hello_world.pro 文件内容

```
1  QT       += core gui
2
```

```
3   greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5   CONFIG += c++11
6
7   # The following define makes your compiler emit warnings if you use
8   # any Qt feature that has been marked deprecated (the exact warnings
9   # depend on your compiler). Please consult the documentation of the
10  # deprecated API in order to know how to port your code away from it.
11  DEFINES += QT_DEPRECATED_WARNINGS
12
13  # You can also make your code fail to compile if it uses deprecated APIs.
14  # In order to do so, uncomment the following line.
15  # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16  #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18  SOURCES += \
19      main.cpp \
20      mainwindow.cpp
21
22  HEADERS += \
23      mainwindow.h
24
25  FORMS += \
26      mainwindow.ui
27
28  # Default rules for deployment.
29  qnx: target.path = /tmp/$${TARGET}/bin
30  else: unix:!android: target.path = /opt/$${TARGET}/bin
31  !isEmpty(target.path): INSTALLS += target
```

第 1 行，添加了 Qt 的支持的模块，core 与 gui 库是 Qt 的默认设置。

第 3 行，比较 Qt5 版本，如果是 Qt5 版本，在 main.cpp 中 application 是在 QtWidgets 中的，因此要包含这个库。

第 5 行和第 11 行，分别配置的是使用 c++11 和添加 QT_DEPRECATED_WARNINGS 定义。

第 18 行，SOURCES 下的是源文件。

第 22 行，HEADERS 下是头文件。

第 25 行，FORMS 下是 ui 界面文件。

第 28 行，部署默认的规则。

第 29 行，qnx:判断是不是 qnx 操作系统，赋值 target.path = /temp/$${TARGET}/bin。

第 30 行，如果是 unix 系统但不是安卓，赋值 target.path = /opt/$${TARGET}/bin。

第 31 行，如果 target.path 为空目录，赋值 INSTALLS += target。

如果需要修改生成目标的可执行程序名字，可赋值 TARGET = xxx。否则 TARGET 将默认取值为项目的名字。

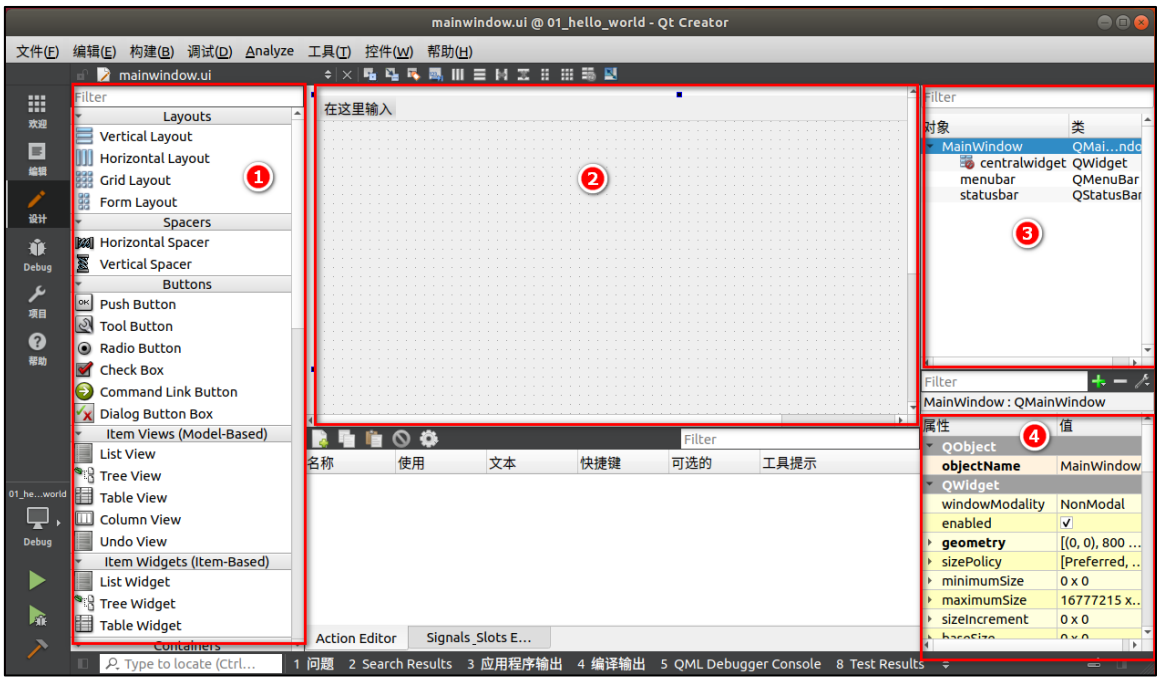
3.6.2.2 样式文件*.ui

mainwindow.ui 是一个 xml 类型的文件，它的 xml 内容如下。这个文件是生成的不能手动编辑。只能够通过图形界面修改其属性。

mainwindow.ui 文件内容

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ui version="4.0">
3   <class>MainWindow</class>
4   <widget class="QMainWindow" name="MainWindow">
5    <property name="geometry">
6     <rect>
7      <x>0</x>
8      <y>0</y>
9      <width>800</width>
10     <height>600</height>
11    </rect>
12   </property>
13   <property name="windowTitle">
14    <string>MainWindow</string>
15   </property>
16   <widget class="QWidget" name="centralwidget"/>
17   <widget class="QMenuBar" name="menubar">
18    <property name="geometry">
19     <rect>
20      <x>0</x>
21      <y>0</y>
22      <width>800</width>
23      <height>28</height>
24     </rect>
25    </property>
26   </widget>
27   <widget class="QStatusBar" name="statusbar"/>
28  </widget>
29  <resources/>
30  <connections/>
31 </ui
```

双击 mainwindow.ui 后可以跳转到设计界面，如下图。下面主要介绍主体部分。



1. ①是控件栏，有各种各样的控件，上方的 Filter 是过滤器，输入首写字母就可以快速定到我们想要找的控件。
2. ②显示的是我们的窗口程序了，上面已经带有 MainWindow 对象及其几个子对象，默认 MainWindow 就带有菜单栏和状态栏。
3. ③是对象栏，②处用到的对象都在③处显示。
4. ④是属性栏，点击③处对象栏的某个对象，就可以在④属性栏里编辑它的属性了。属性项有很多，包括位置，大小，文字，颜色，字体等等。

3.6.2.3 头文件*.h

点击左边栏的编辑，回到项目的编辑工作窗口。点击项目下的 Headers 下的 mainwindow.h，mainwindow.h 一般有与之对应的一个 cpp 文件叫 mianwindow.cpp。其中 mainwindow.h 包含类的声明，mianwindow.cpp 包含类的实现。这与我们前面学习的 c++是一样的。

mainwindow.h 的内容如下。

mianwindow.h 定义如下

```
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5
6   QT_BEGIN_NAMESPACE
7   namespace Ui { class MainWindow; }
8   QT_END_NAMESPACE
9
10  class MainWindow : public QMainWindow
```

```
11  {
12      Q_OBJECT
13
14  public:
15      MainWindow(QWidget *parent = nullptr);
16      ~MainWindow();
17
18  private:
19      Ui::MainWindow *ui;
20  };
21  #endif // MAINWINDOW_
```

第 7 行，定义名称空间 Ui ，里面有一个类 MainWindow ，这个 MainWindow 和第 10 行里的 MainWindow 不是同一个对象。实际上，这个文件里的 MainWindow 类有一个成员*ui 就是指向第 19 行的 Ui::MainWindow 的指针。这都是为了使用.ui 文件设计界面的。

第 12 行，MainWindow 的声明中第一行是 Q_OBJECT，这是一个宏，由 Qt 进行处理，这也是 Qt 针对 C++扩展的地方，所有用到信号的类都要加这个宏。

3.6.2.4 源文件*.cpp

点击项目下的 Sources 下的 mainwindow.cpp。可以看到它的内容如下。

mianwindow.cpp 实现如下

```
1   #include "mainwindow.h"
2   #include "ui_mainwindow.h"
3
4   MainWindow::MainWindow(QWidget *parent)
5       : QMainWindow(parent)
6       , ui(new Ui::MainWindow)
7   {
8       ui->setupUi(this);
9   }
10
11  MainWindow::~MainWindow()
12  {
13      delete ui;
14  }
```

第 2 行，MainWindow 的实现类中，第 2 行 include 了一个文件 ui_mainwindow.h 这个文件是 Qt 根据.ui 文件自动生成的，也就是说 ui_mainwindow.h 要点击编构建后才生成，我们才能看到这个文件。构建/编译后可以在 debug/release 的目录找到这个文件。

第 6 行，在 MainWindow 构造函数中用“,”隔开，new 一个 Ui 中的 MainWindow。这里是一种初始化成员的方法。

第 8 行，ui->setupUi(this);这句话是进行界面初始化，将 this（指的是 MainWindow 类的本身），作为参数传到 setupUi 里，ui 界面文件的对象会以 this 为父对象，所有子对象都将显示在 MainWindow 里。我们要想使用 ui 里的对象，必须将代码写在 ui->setupUi(this)这句话之后，因为 ui->setupUi(this)会先初始化里面的对象，只有初始化里面的对象我们才能使用这个对象。

第 13 行，析构函数里 delete 掉 ui。在 Qt 里我们需要在析构函数里 delete 的对象一般是 new 创建的并且没有父对象的对象。

小提示：this 在成员函数的开始执行前构造的，在成员的执行结束后清除。

点击项目下的 Sources 下的 main.cpp。可以看到它的内容如下。

mian.cpp 实现如下

```
1  #include "mainwindow.h"
2
3  #include <QApplication>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8      MainWindow w;
9      w.show();
10     return a.exec();
11 }
```

第 3 行，包含 QApplication 类的定义。在每一个使用 Qt 的应用程序中都必须使用一个 QApplication 对象。QApplication 管理了各种各样的应用程序的广泛资源，比如默认的字体和光标。

第 5 行，main()函数是程序的入口。几乎在使用 Qt 的所有情况下，main()只需要在把控制转交给 Qt 库之前执行一些初始化，然后 Qt 库通过事件来向程序告知用户的行为。argc 是命令行变量的数量，argv 是命令行变量的数组。

第 7 行，a 是这个程序的 QApplication。它在这里被创建并且处理这些命令行变量。

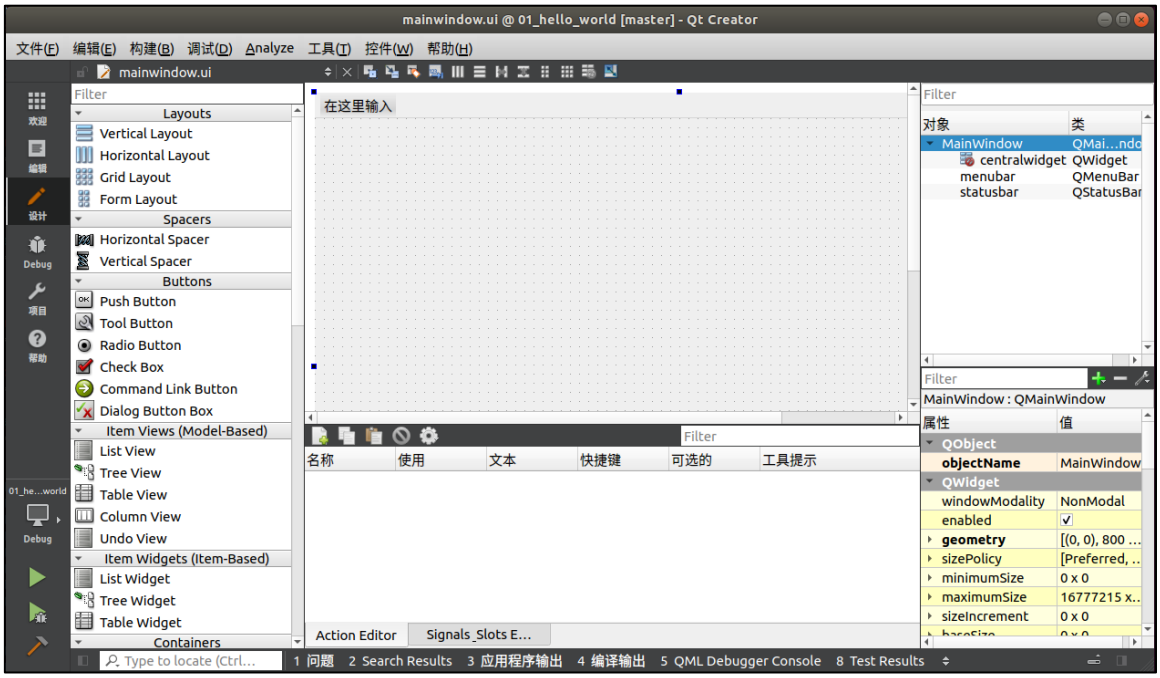
第 8 行，创建一个对象 w，这个对象就是 MainWindow。

第 9 行，调用方法 show()。这样程序界面才能显示。

第 10 行，这里就是 main()把控制转交给 Qt，并且当应用程序退出的时候 exec()就会返回。在 exec()中，Qt 接受并处理用户和系统的事件并且把它们传递给适当的窗口部件。

3.6.3 修改 ui 文件显示 hello world

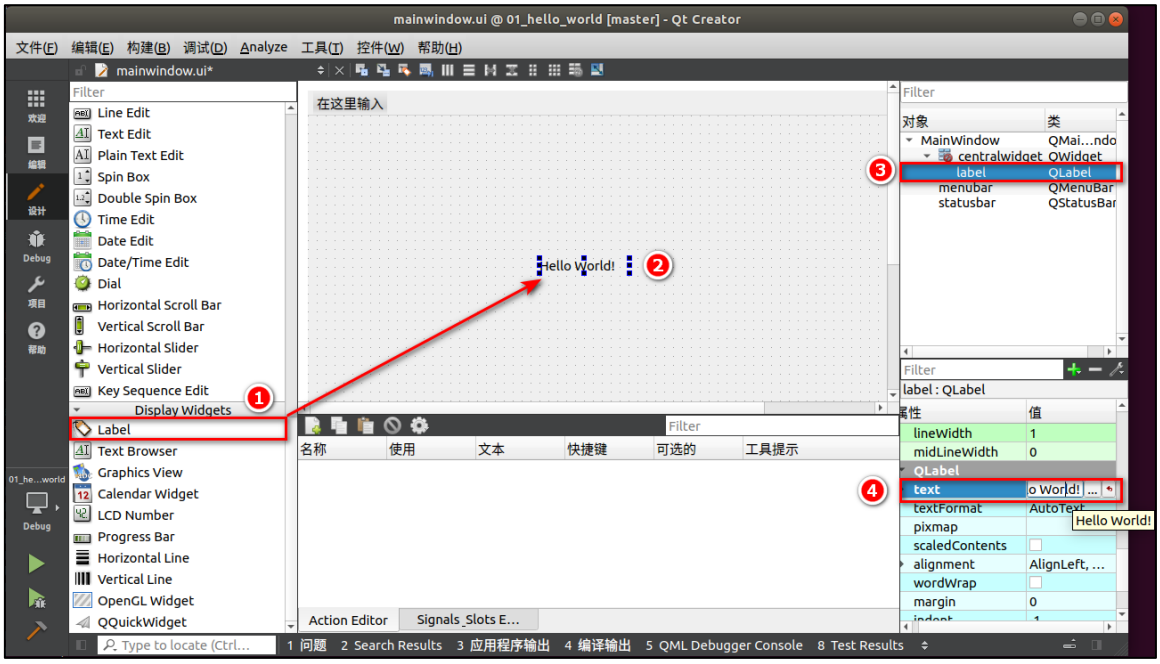
双击 mainwindow.ui，进入“Ui 设计器”页面如下。



可以看到中间的要设计的主窗体目前是空的，但是窗体上已经有好几个成员了，可以看到右边的对象栏，已经有 3 个对象成员在里面了，只是我们看不出来而已，效果不太明显。

我们要在窗体里显示“Hello World!”，那么需要使用左边的控件栏，目前左边的控件栏的控件有很多，也分很多类别。这些对于我们初学者来说，暂时不用知道它们是怎么用的。后面教程其中的类别进行说明它们是如何使用的。现在我们主要是想要显示“Hello World!”。要明白我们的主要目的，切勿看到左边的控件栏就想一个个的去试，又不知道怎么去用。这样的学习方式是不对的。要显示“Hello World!”，那么我们需要用常用的文本显示控件，常用的就是 Label 文本控件了，当然 PushButon 按钮类也是可以显示文本的。但是我们不需要按钮的功能，只需要显示文本。所以用 Label 文本控件就可以了。

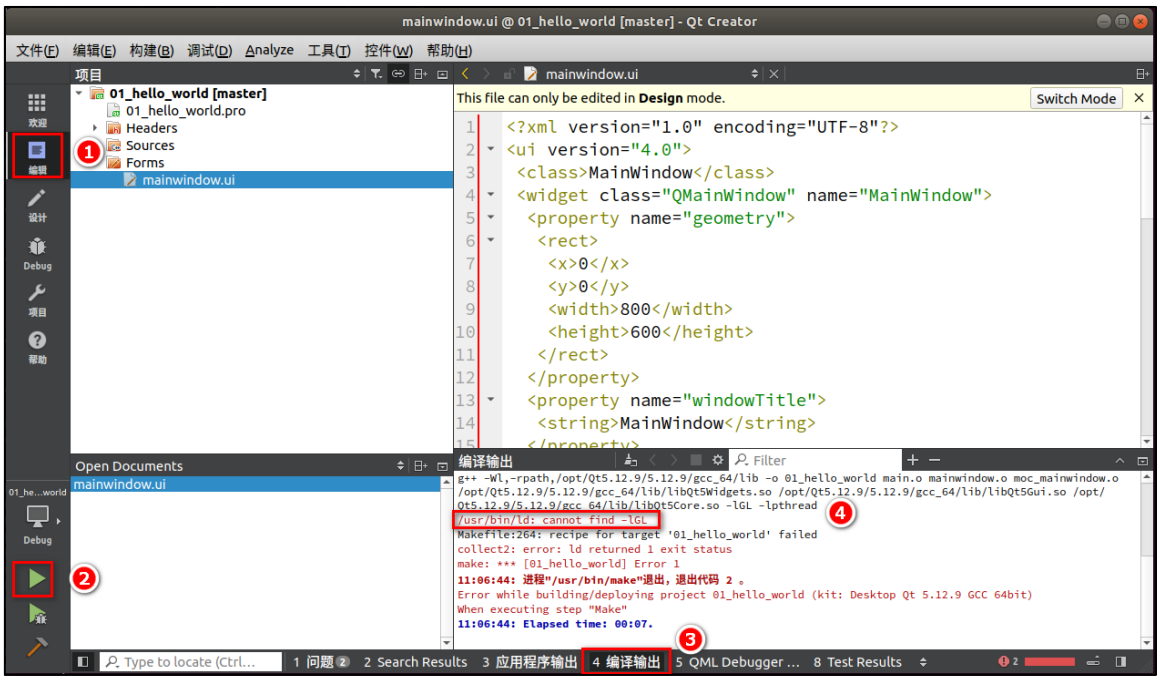
如下图，在左边的控件图，找到 Label 控件，Label 控件在“Display Widgets”（翻译为显示小部件）下，选中 Label 控件，将其拖至②处中间的设计窗体里。然后在③处选中 label 对象，再在④处 label 对象的属性栏找到 text 文件属性，修改里面的文本为“Hello World!”。这样我们就已经显示“Hello World”在窗口里了。当然双击中间的 Label 控件也是可以修改其文本属性的，对初学者来说还是按步骤修改，这样学习“Ui 设计器”是如何使用的。这里我们已经完成显示“Hello World!”的设计了。



3.6.4 项目编译&调试&运行

完成上面的“Hello World!”后，我们肯定下一步是想编译运行我们的程序了，看看效果如何！

按如下步骤，先返回编辑页面，再点击②处左下角的绿色三角符号（或者按 Ctrl + R 快捷键编译且运行），编辑时会输出编译信息，点击③处的编译输出窗口里，可以看到编译时的编译信息，能看到编译警告、编译错误、编译链接、相关编译器编译过程等。

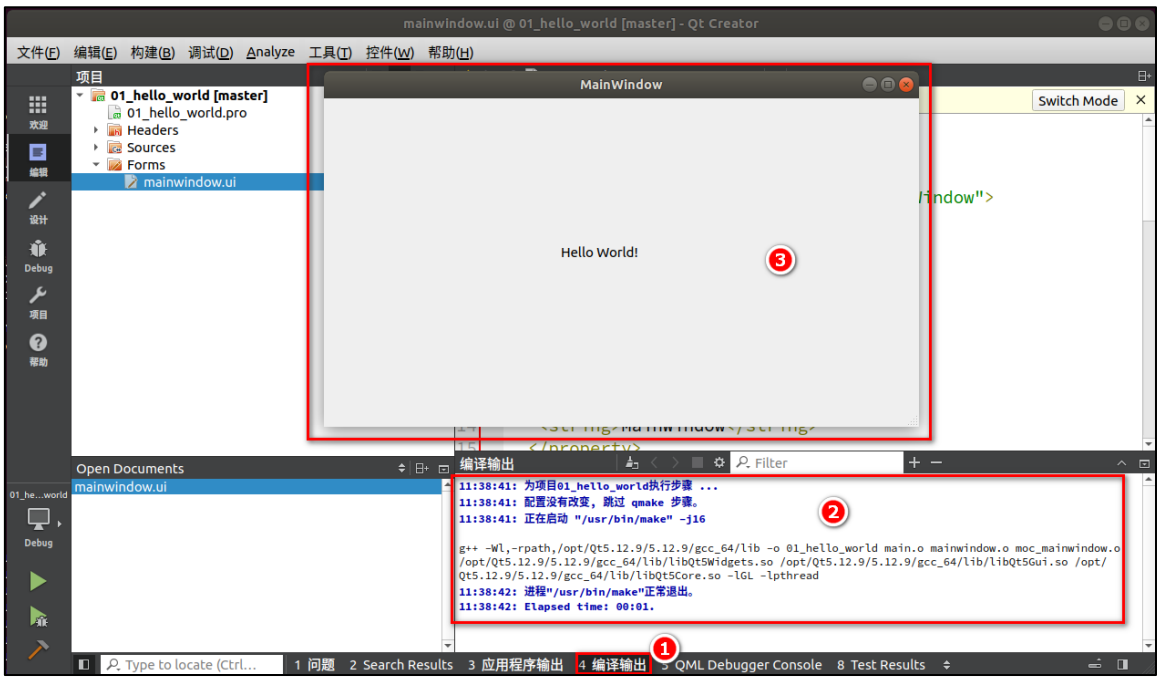


在 Ubuntu 下刚装的 Qt Creator，毫无意外，第一次编译时会报“Cannot find -lgL”的错误。由于 Qt5.0 的库默认会链接到 OpenGL，但是在 Ubuntu 机器上没有安装 OpenGL，所以们需要在 Ubuntu 下安装 OpenGL Library。在 Ubuntu 终端下输入如下指令。

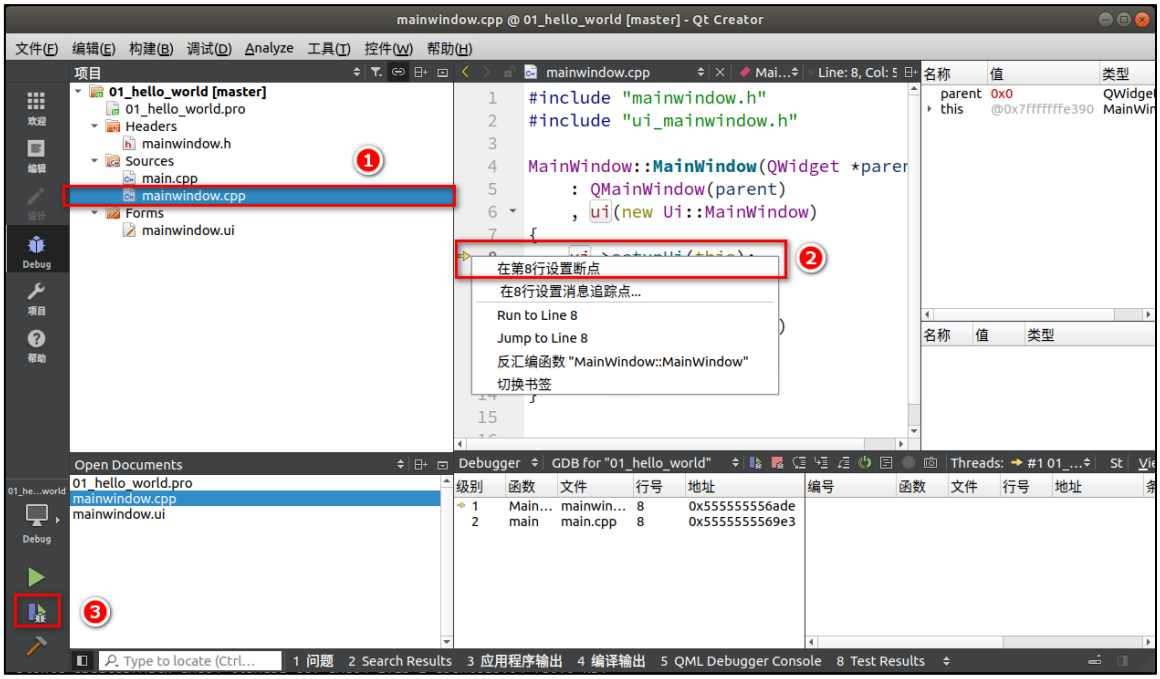
```
sudo apt-get install libglu1-mesa-dev
```

安装完成后，我们再点击左下角的绿色三角形试试，可以看到如下结果！在下图②处已经看到没有报错信息了。如果还有其他报错，建议回到第一章 C++环境设置里寻找 C++环境配置的相关指令安装好相应的库。可能初学者跳过了 C++就直接到 Qt 这部分了，Qt 也依赖 C++的环境的！

运行的结果如③处，弹出一个窗口，中间就是我们设计的 Hello World!



如果需要调试，请先关闭上面的“Hello World!”窗口，点击右上角的“x”关闭即可！再按如下步骤点击，在程序里右键设置断点，再点击左下角③处的 Debug 绿色三角符号按钮。不过我们很少使用调试，这种简单的程序完全是可以预知它的运行过程的，如果我们想要知道它的运行过程，是可以使用这个调试的功能去了解 Qt 程序的运行流程的！



第四章 使用 Qt Designer 开发

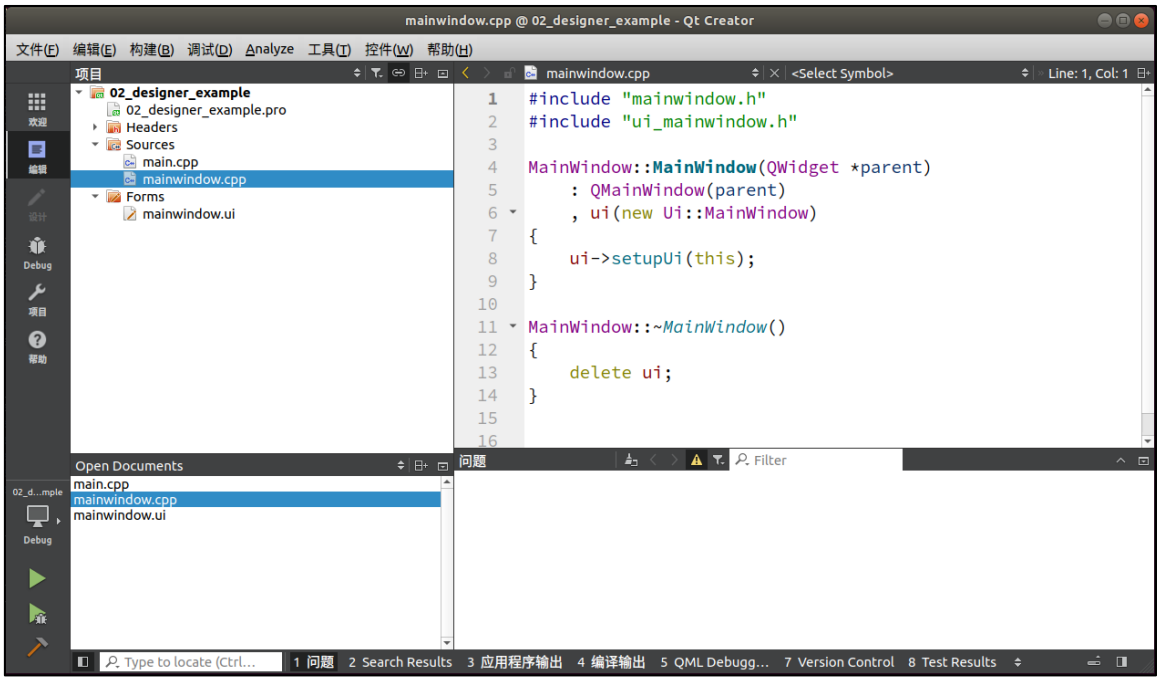
本章将简介使用 Qt Creator 里自带的 Qt Designer，使用 Qt Designer 比较方便的构造 UI 界面。特点是方便布局，比较形象。

4.1 使用 UI 设计器开发程序

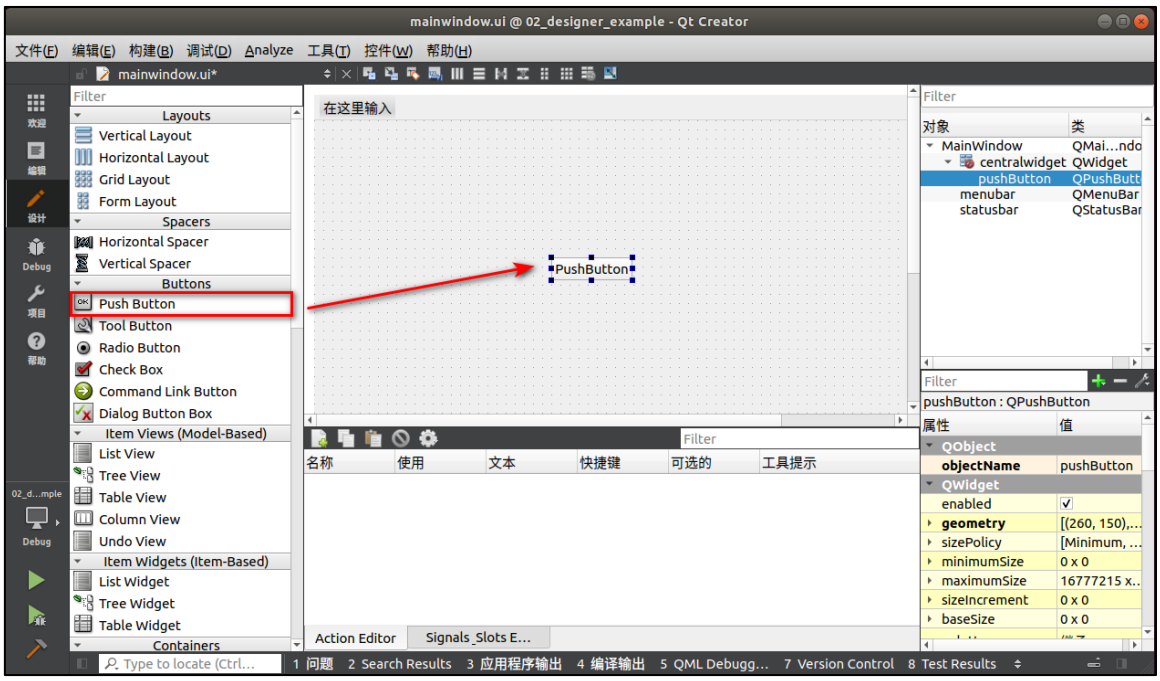
在这小节里我们继续学习如何使用 Qt Designer 开发程序，Qt Designer 是属于 Qt Creator 的一个功能而已，大家不要搞混了。Qt Designer 也叫 UI 设计师或者 UI 设计器，这都是指的同一个东西而已。下面就简单介绍使用 UI 设计器开发程序，以连接信号与槽为例，简单的介绍这个开发流程。最后我们思考一下这种开发方式的好处以及不便之处。

4.1.1 在 UI 文件添加一个按钮

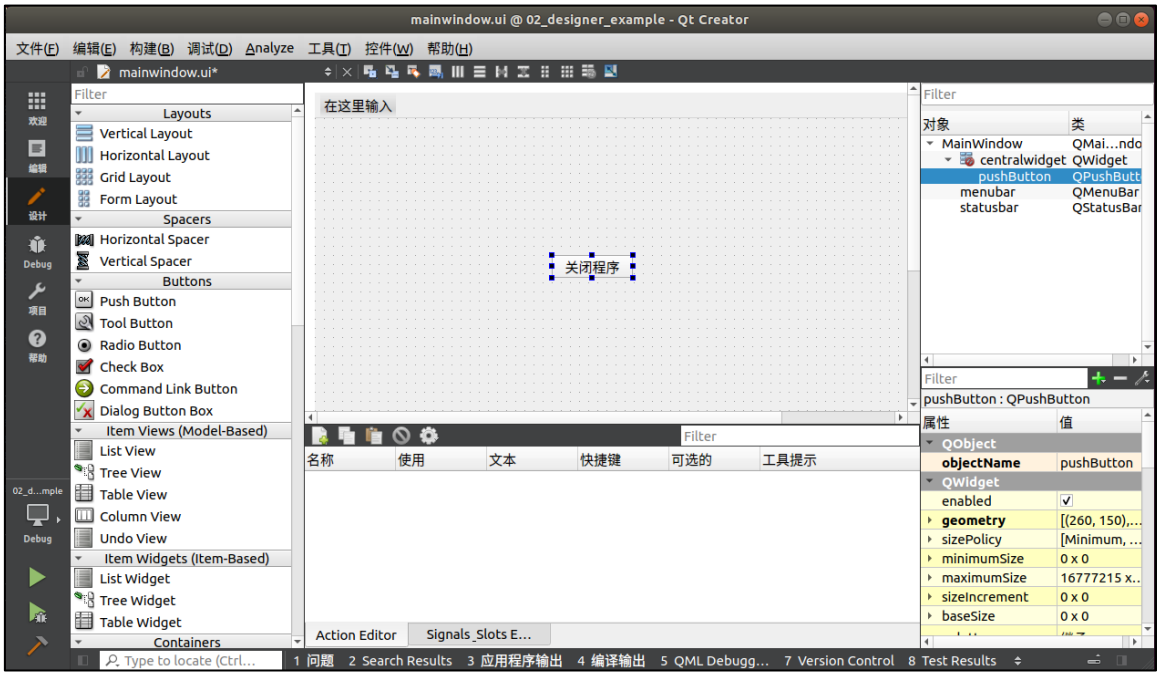
新建一个项目为 02_designer_example。如果还不会新建工程，请回到 3.6 小节，步骤一样，这里不再详细说项目的建立过程了。新建的项目如下，为了方便截图，笔者已经把 3.6 小节的项目“01_hello_world”关闭了。剩下的就是我们这个新建的项目 02_designer_example。



添加按钮的方法与 3.6.3 小节添加 Label 的方法一样，在左边找到 Push Button，然后拖拽到中间的显示窗体里，如下图。



并将这个 PushButton 的 text 属性（文本属性）改为“关闭程序”。我们在 4.1.2 小节要设计点击这个按钮将关闭这个窗口，关闭这个程序。

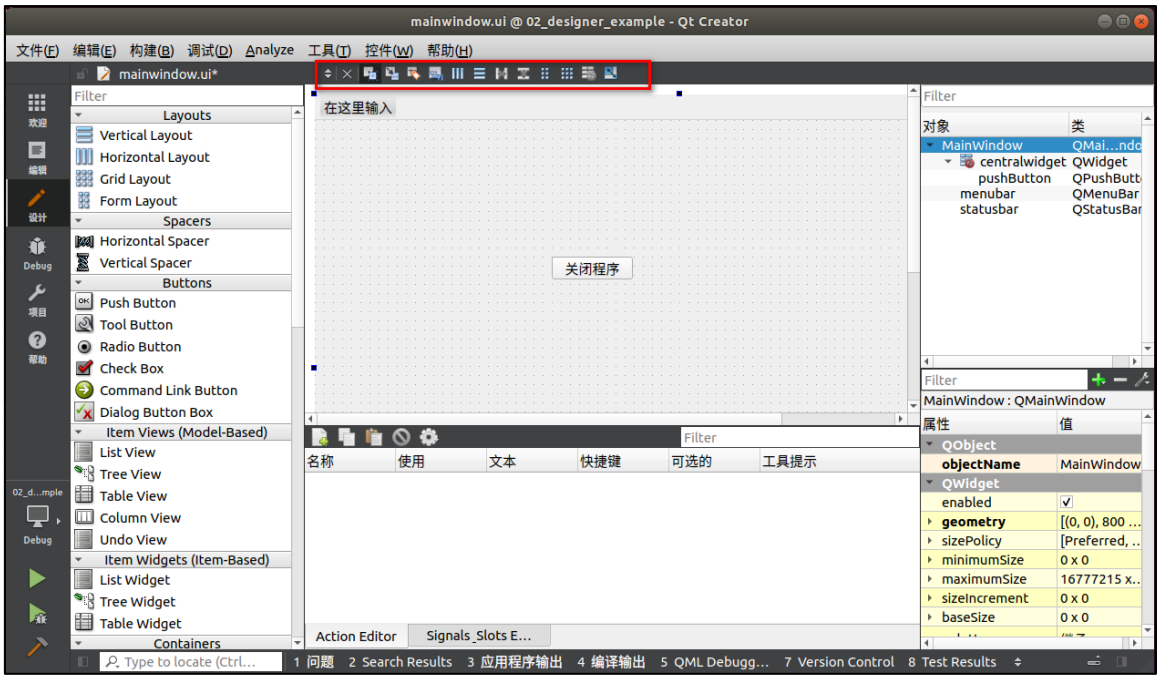


4.1.2 在 UI 文件里连接信号与槽

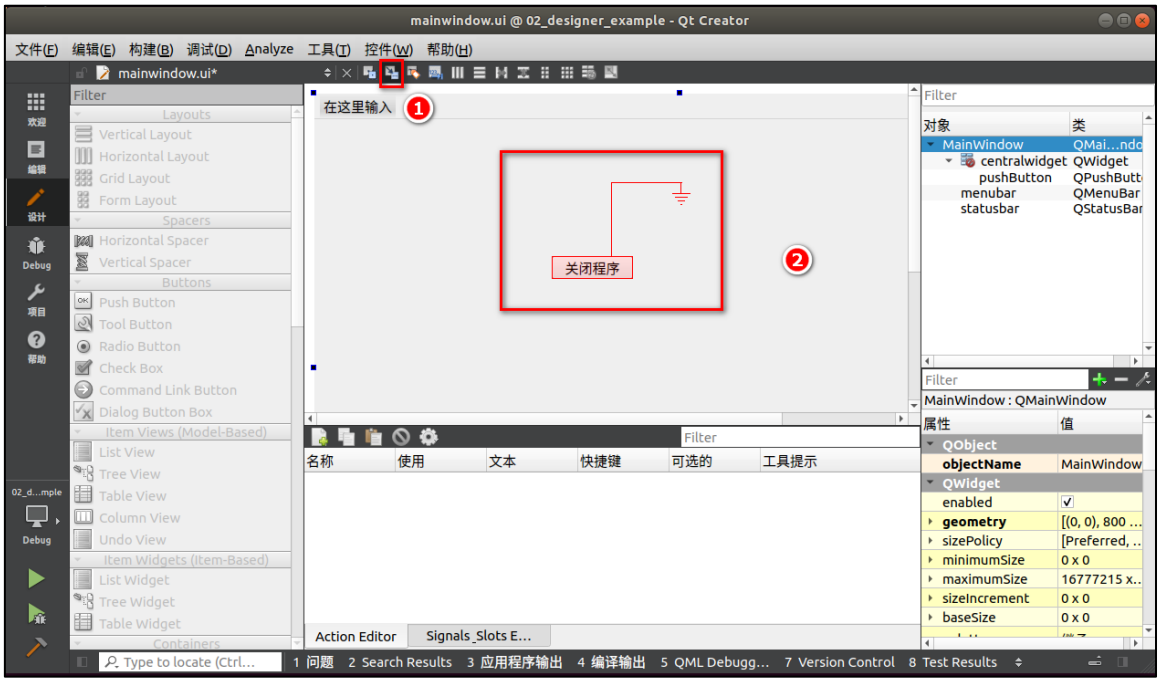
在 UI 设计器里有两种方法可以连接信号与槽。初学者可能不了解什么是 Qt 的信号与槽。这里先简单的介绍一下信号与槽的功能。所谓信号即是一个对象发出的信号，槽即是当这个对象发出这个信号时，对应连接的槽就发被执行或者触发。以上为非专业术语解释，在下一章节可以学到 Qt 信号与槽机制。现在我们只要了基本的去了解一下这个信号与槽即可。可以说信号与槽在 Qt 里是必不可少的。要想事件做出对应的动作就必须用到信号与槽。

UI 设计器里信号与槽的连接方法一：

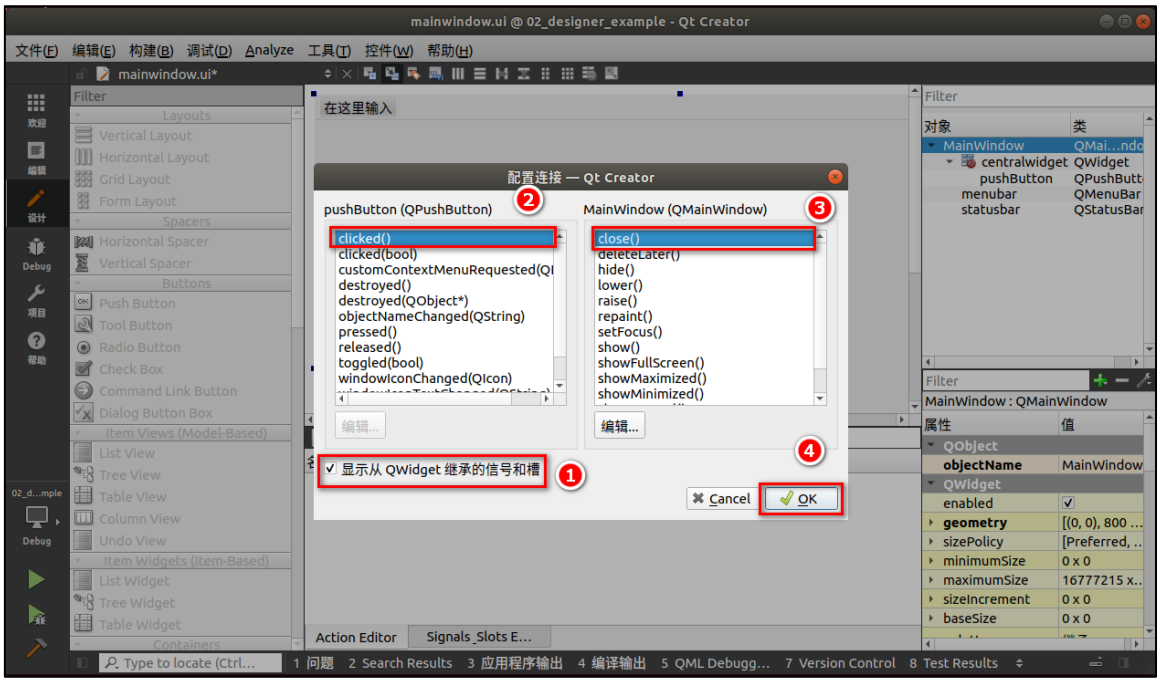
在主窗体的上面部分，我们可以看到一些小小的按钮，如下图框框部分。用鼠标放在这些按钮上面可以查看这个按钮是什么作用。信号槽连接的按钮也在上面。



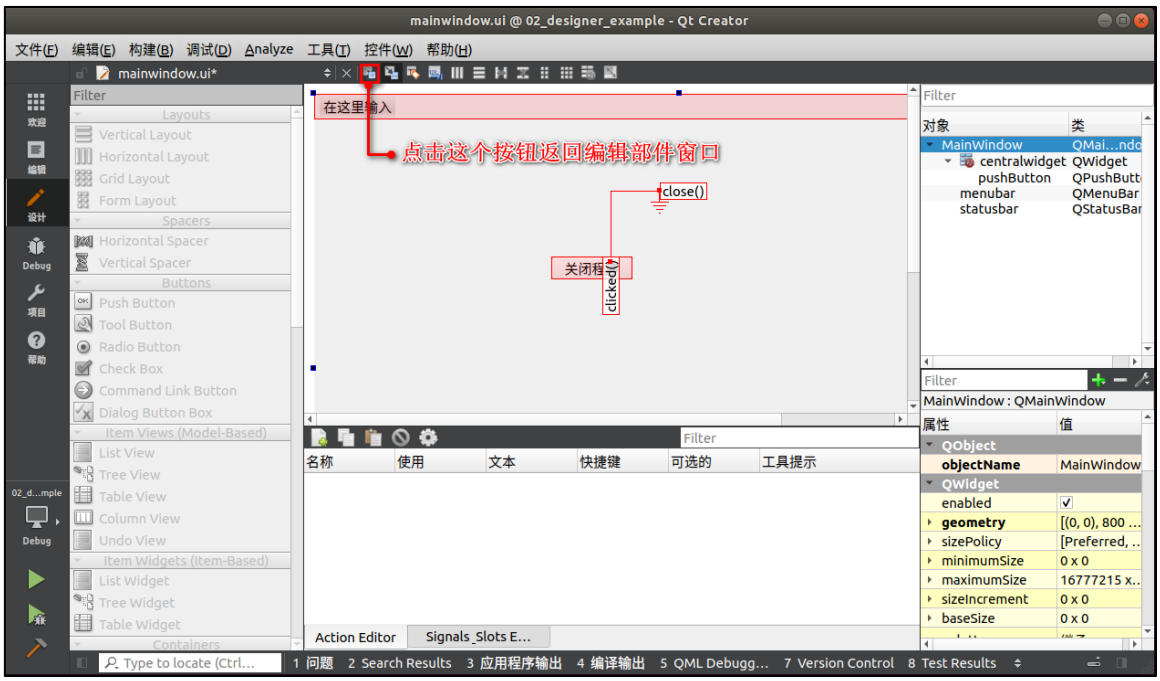
点击信号槽连接的按钮如下，如下图①处，点击进入信号槽连接模式（若想退出信号槽连接模式，则点击①处左边的按钮），进入信号与槽的连接模式后，将鼠标选中我们的“关闭程序”按钮，按住按钮，然后用鼠标向外拖动，如②处。此时就会出现信号槽连接的符号。



之后按如下图步骤选择，左边的“关闭程序”pushButton 按钮的信号，可以看到一个对象的信号可以有多种。右边的 QMainWindow 的槽函数，如果有其他对象，右边不一定只有 MainWidnow 的槽函数（槽），也有可能是其他对象的槽。我们选择按钮的 clicked()信号，将其连接 MainWindow 对象的 close()槽。这样就完成了信号与槽的连接，非常简单。我们也可以预知这个信号与槽的功能，当“关闭程序”pushButton 发出了 clicked()信号（也就是单击信号）。这个信号由“关闭程序”pushButton 被单击时发出。它就会触发 MainWindow 的 close()。进而使整个程序关闭。MainWindow 的 close()就是退出关闭程序，退出程序的意思。

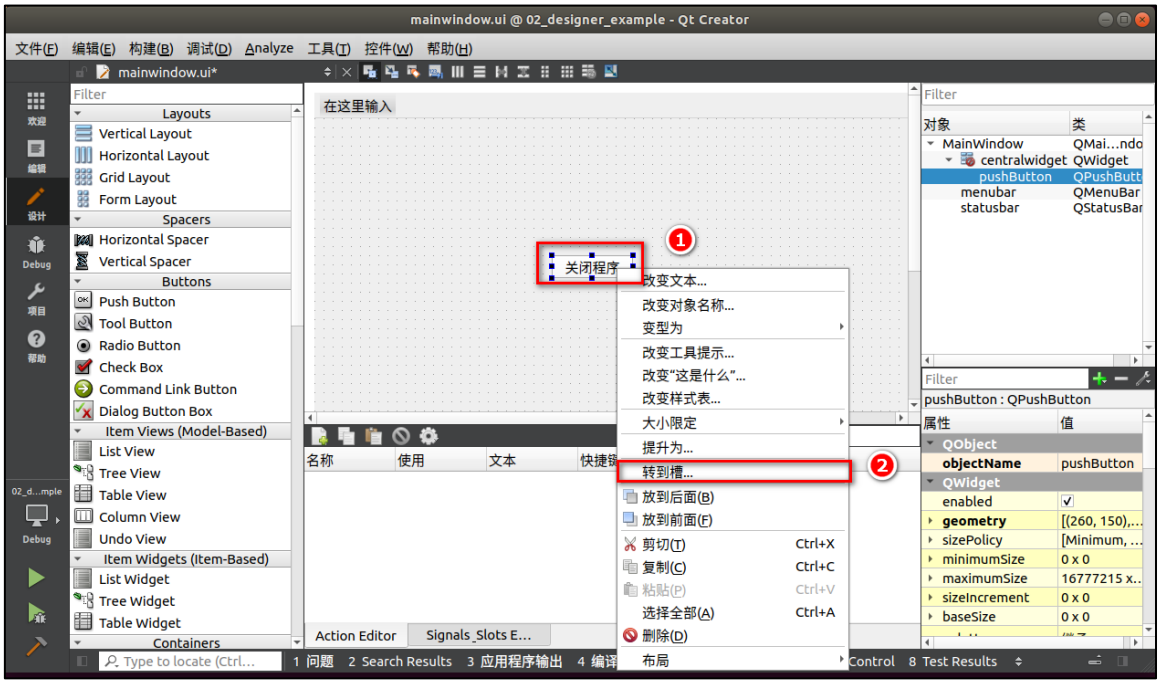


完成信号槽连接，如下图。要想返回编辑部件模式点击如下图标注位置的按钮。下图就是信号与槽连接的图示了。在编辑部件模式下我们是看不见的，只有信号槽模式才能看见这样的图示。

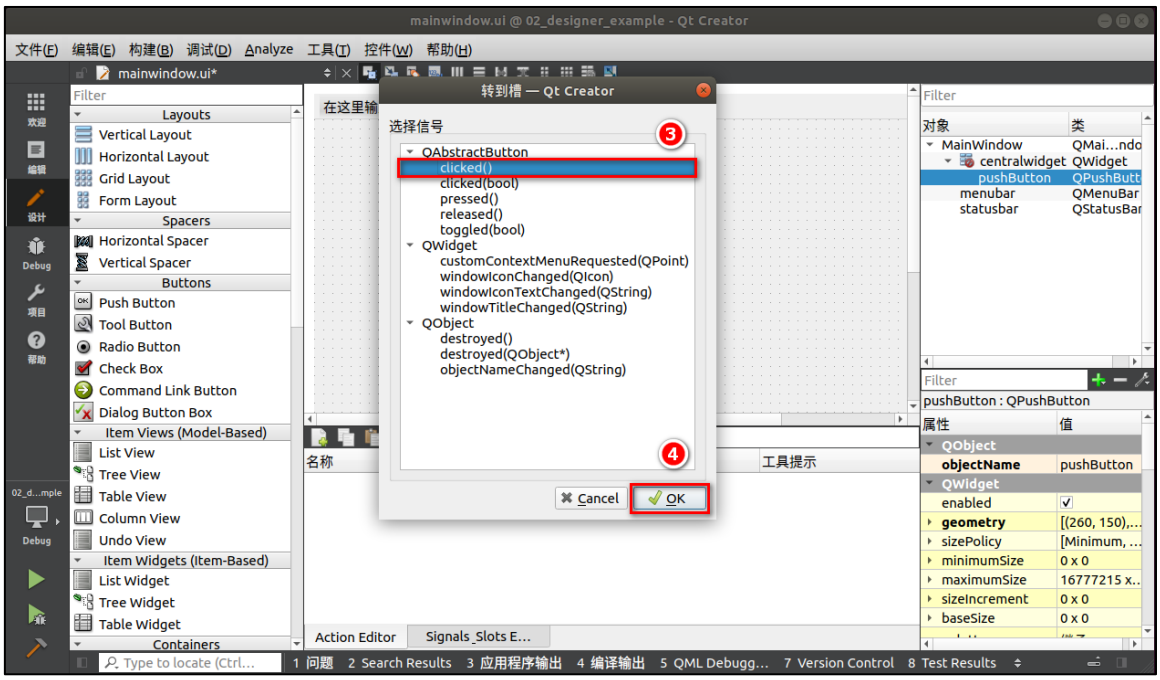


UI 设计器里信号与槽的连接方法二：

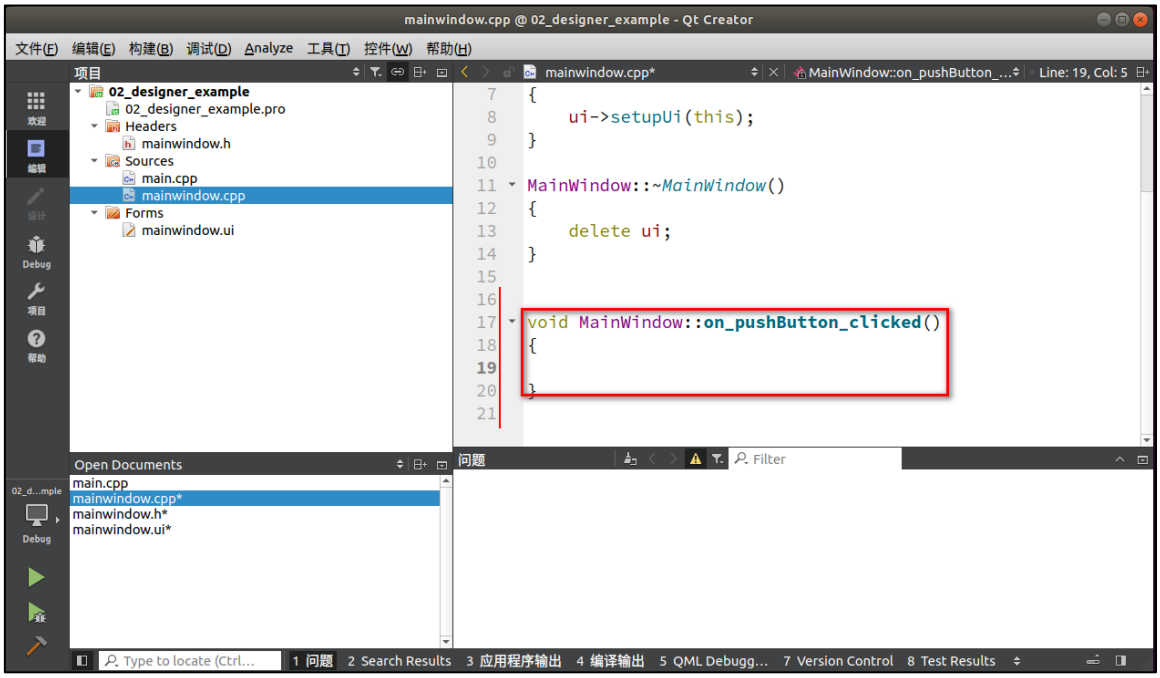
选中“关闭程序”pushButton 按钮，然后右键，如下图。选择“转到槽”。



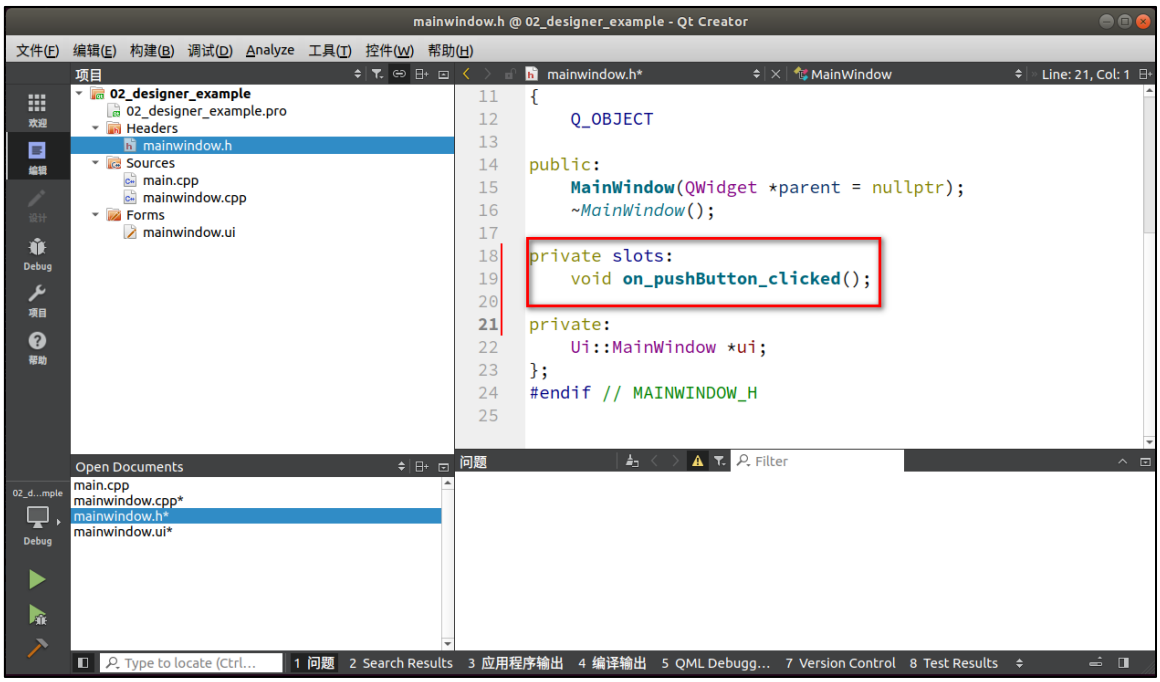
点击“转到槽”后，弹出下面的窗口，这一步是先让我们选择信号。按如下图选择。如果细心的同学，我们还发现这个 clicked()信号并不是 pushButton 的，而是 QAbstactButton 的。只是 pusbButton 继承了 QAbstracButton，同时把这个信号也继承了下来。除此之外我们还看到其他信号也是不是属于 pushButton 的，也是被继承下来了。所以我们在 C++基础部分学过的继承。在 Qt 里的作用表现的淋漓尽致！根本不用重写 pushButton 的 clicked()事件。pushButton 只需要继承父类的 clicked()事件即可！



点击 OK 后，就会跳转到槽函数里，这个代码由 Qt Creator 自动生成。

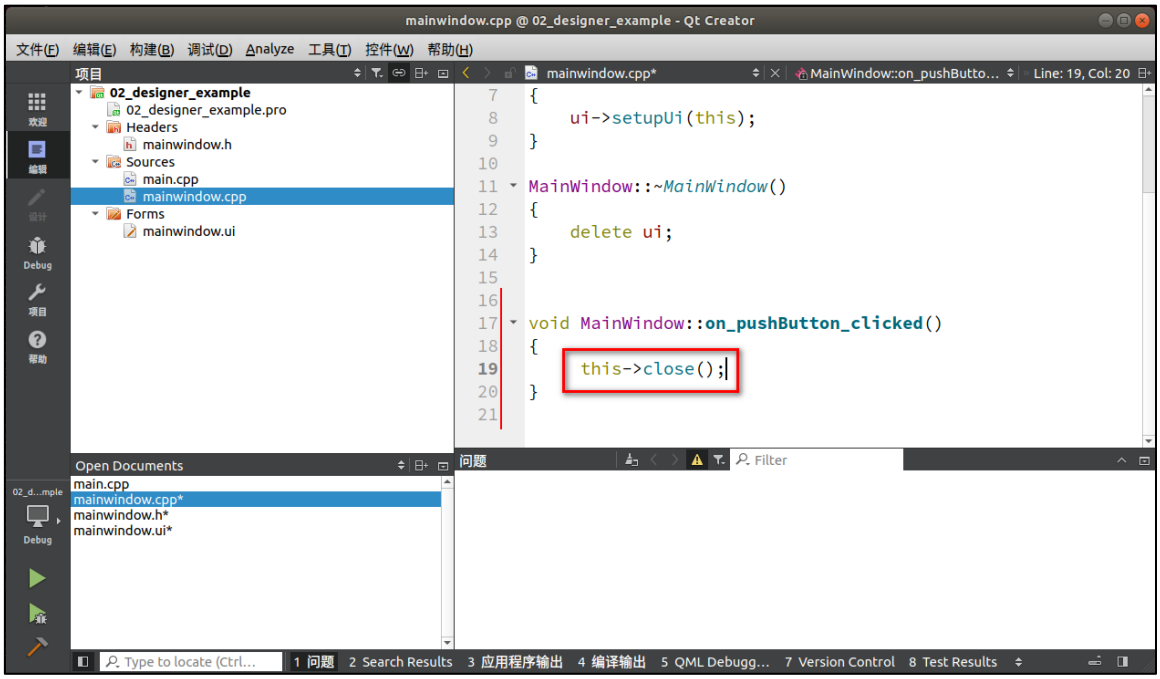


同时在 mainwindow.h 里声明了这个槽函数。



如果我们学过 C#，这就好像 C#里的跳转到事件一样。其实这种便捷的编程方式很多编程的 IDE 都非常类似。只要我们对这种 IDE 有一定的了解，学习起来就不会觉得难。

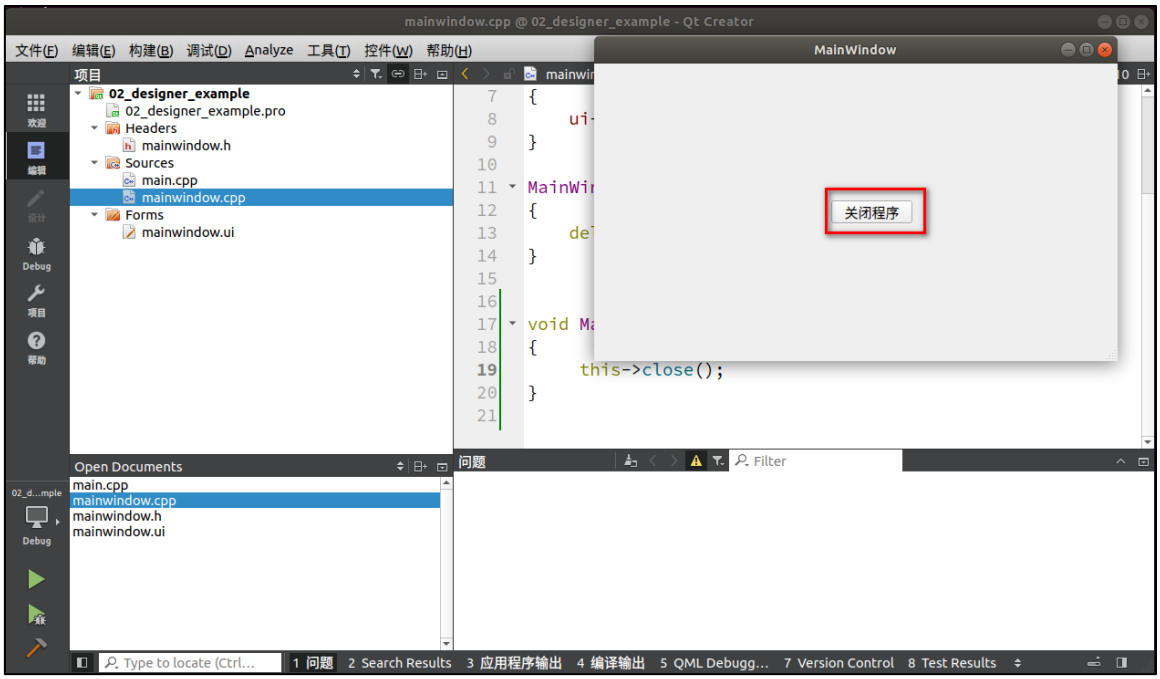
返回到 mainwindow.cpp 找到 on_pushButton_clicked 这个槽函数里。在这个槽数里写上 this->close();调用 close()方法关闭整个程序。



4.1.3 编译及运行创建的 UI 项目

直接按 Ctrl + r 或者点击左下角的第一个绿色三角符号，编译且运行这个项目。运行的结果如下。点击“关闭程序”按钮，看看是不是整个程序关闭退出了呢？如果细心的同学还会思考，我们在上面用了两种方法连接信号与槽。那他们是如何连接的呢？他们两者同时连接会有影响

吗？在哪里连接的呢？这些都是我们需要探讨的问题？难道第二种方法写上槽函数就会自动被触发？下一节我们继续学习 Qt 信号与槽了解信号与槽的写法就能解开这个谜了！



第五章 Qt 信号与槽

在这章节里，我们学习 Qt 的信号与槽，这里分一个章节来学习这个 Qt 的信号与槽，可见这个信号与槽有多么重要。在学习 Qt 的过程中，信号与槽是必不可少的部分，也是 Qt 编程的基础，是 Qt 编程的一大创新（其实与 C#的事件很相似，编程都是类似的），Qt 的信号与槽在 Qt4 时或者更早前已经出现，并不是属于哪个版本的。只是 Qt4 与 Qt5 的信号槽连接的写法有些区别。本教程对 Qt4 的信号与槽连接写法不做讲解。

同时，由此章节开始，我们将不使用 Qt Designer 的方式进行开发，也可以在新建项目时把*ui 文件不勾选。为什么不用 Qt Designer 方式开发程序了呢？Qt Designer 方式开发程序简单，优点就是方便快捷，ui 文件的优点就是能比较直观快捷的看到整体的布局。

但是缺点也很明显！简单的一两个部件还是可以的，但是控件多了就不好管理，每次都要打开 ui 文件添加新控件，不好用代码管理！读者也无从知道笔者是如进行界面布局的，且信号槽的连接方式是生成代码之后才能在 setupUi 函数里才能看到，或者需要进入 Ui 设计器里的信号槽模式里才能看到信号槽的连接，这样十分之不方便！没有代码那么直观！所以统一用代码绘界面，可以锻炼我们的布局能力，和代码逻辑能力！

本章的内容如下：

5.1 小节里我们学习 Qt 信号与槽机制的定义。

5.2 小节开始我们就学习如何在项目里定义我们的信号与槽的创建和使用方法。

5.1 Qt 信号与槽机制

信号与槽（Signal & Slot）是 Qt 编程的基础，也是 Qt 的一大创新。因为有了信号与槽的编程机制，在 Qt 中处理界面各个组件的交互操作时变得更加直观和简单。

信号（Signal）就是在特定情况下被发射的事件，例如 PushButton 最常见的信号就是鼠标单击时发射的 clicked() 信号，一个 ComboBox 最常见的信号是选择的列表项变化时发射的 CurrentIndexChanged() 信号。

GUI 程序设计的主要内容就是对界面上各组件的信号的响应，只需要知道什么情况下发射哪些信号，合理地去响应和处理这些信号就可以了。

槽（Slot）就是对信号响应的函数。槽就是一个函数，与一般的 C++函数是一样的，可以定义在类的任何部分（public、private 或 protected），可以具有任何参数，也可以被直接调用。槽函数与一般的函数不同的是：槽函数可以与一个信号关联，当信号被发射时，关联的槽函数被自动执行。

信号与槽关联是用 QObject::connect() 函数实现的，其基本格式是：

```
QObject::connect(sender, SIGNAL(signal()), receiver, SLOT(slot()));
```

connect() 是 QObject 类的一个静态函数，而 QObject 是所有 Qt 类的基类，在实际调用时可以忽略前面的限定符，所以可以直接写为：

```
connect(sender, SIGNAL(signal()), receiver, SLOT(slot()));
```

其中，sender 是发射信号的对象的名称，signal() 是信号名称。信号可以看做是特殊的函数，需要带括号，有参数时还需要指明参数。receiver 是接收信号的对象名称，slot() 是槽函数的名称，需要带括号，有参数时还需要指明参数。

SIGNAL 和 SLOT 是 Qt 的宏，用于指明信号和槽，并将它们的参数转换为相应的字符串。例如，在 4.1 小节的项目 02_designer_example 的 ui_mainwindow.h 文件（ui_mainwindow.h 文件是编译后产生的，位于 build-02_designer_example-Desktop_Qt_5_12_9_GCC_64bit-Debug 文件夹目录下）中，在 setupUi() 函数中有如下的语句：

```
QObject::connect(pushButton, SIGNAL(clicked()), MainWindow, SLOT(close()));
```

其作用就是将 pushButton 按钮的 clicked() 信号与窗体（MainWindow）的槽函数 close() 相关联，这样，当单击 pushButton 按钮（就是界面上的“X”按钮）时，就会执行 MainWindow 的 close() 槽函数。在这里我们就可以解决 4.1.3 小节的中部分疑问了。我们使用第一种方法没看到信号与槽是怎么连接的，实际上它在 UI 文件编译后在生成的代码 ui_mainwindow.h 里！

关于信号与槽的使用，有以下一些规则需要注意：

一个信号可以连接多个槽，例如：

```
connect(pushButton, SIGNAL(clicked()), this, SLOT(hide());
connect(pushButton, SIGNAL(clicked()), this, SLOT(close());
```

这是当一个对象 pushButton 的被单击时，所在窗体有两个槽进行响应，一个 hide()用于隐藏主窗体，一个 close 用于关闭主窗体。这里只是举个例子，实际上当执行 close()后，hide()这个槽已经没有什么意义了，因为已经程序退出了，但是实际它有执行，因为它连接在 close()的前面。当一个信号与多个槽函数关联时，槽函数按照建立连接时的顺序依次执行。

当信号和槽函数带有参数时，在 connect()函数里，要写明参数的类型，但可以不写参数名称。

多个信号可以连接同一个槽，例如在 02_designer_example 里，我们再拖 2 个 pushButton 到设计的窗体里。同时也让它按 4.1 小节项目里的第一种方法连接信号与槽，把三个按钮的 clicked()信号都连接到 close()槽函数里。编译后可以在 ui_mainwindow.h 这个文件里的 setupUi()函数里，有如下的信号槽连接语句。

```
connect(pushButton,SIGNAL(clicked()),this,SLOT(close()));
connect(pushButton_2,SIGNAL(clicked()),this,SLOT(close()));
connect(pushButton_3,SIGNAL(clicked()),this,SLOT(close()));
```

这样，当任何一个 pushButton 被单击时，都会执行 close()函数，进而关闭或者退出程序。

一个信号可以连接另外一个信号（说明了 connect 万物皆可连，非常好用！），例如：

```
connect(pushButton, SIGNAL(objectNameChanged(QString)),this, SIGNAL(windowTitelChanged(QString)));
```

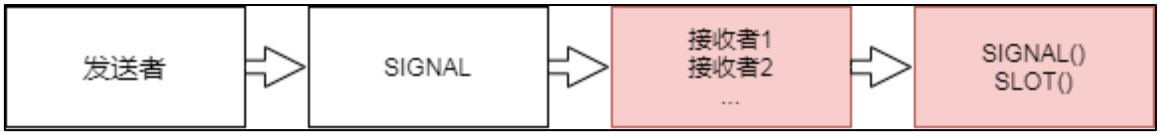
这样，当一个信号发射时，也会发射另外一个信号，实现某些特殊的功能。

严格的情况下，信号与槽的参数个数和类型需要一致，至少信号的参数不能少于槽的参数。如果不匹配，会出现编译错误或运行错误。

在使用信号与槽的类中，必须在类的定义中加入宏 Q_OBJECT（特别重要）。

当一个信号被发射时，与其关联的槽函数通常被立即执行，就像正常调用一个函数一样。只有当信号关联的所有槽函数执行完毕后，才会执行发射信号处后面的代码。

总结如下图，可以看到发送者与发送的信号是在一起的，接收者与接收的信号/槽是在一起的。它们不能在 connect()方法里写乱顺序！由发送者发送出信号到接收者用信号/槽接收。



信号槽连接的方法已经讲解了。这里只是稍稍提一下，断开连接的方法，初学者基本用不到断开连接的操作。使用 disconnect()。disconnect()，这个方法重载了好几个函数，解开格式如下。

```
bool QObject::disconnect(const QObject *sender, const char *signal, const QObject *receiver, const char *method)
```

- 断开一切与 myObject 连接的信号或槽。

```
disconnect(myObject, 0, 0, 0);
```

相当于非静态重载函数：

```
myObject->disconnect();
```

- 断开所有连接到特定信号的东西。

```
disconnect(myObject, SIGNAL(mySignal()), 0, 0);
```

相当于非静态重载函数：

```
myObject->disconnect(SIGNAL(mySignal()));
```

- 与指定的接收者断开连接。

```
disconnect(myObject, 0, myReceiver, 0);
```

相当于非静态重载函数：

```
myObject->disconnect(myReceiver);
```

信号与槽机制是 Qt GUI 编程的基础，使用信号与槽机制可以比较容易地将信号与响应代码关联起来。

5.2 如何在项目里创建信号

我们先新建一个项目，并把项目命名为 03_signal_slot_example，如果还不会新建项目，请回到 3.6 小节查看项目如何建立的。只是新建项目命名为 03_signal_slot_example，同时取消勾选*ui 文件（为什么取消？可以在第五章章节开头处找理由），其他步骤不变。

由于信号只需声明，无需定义。所以我们只需要在 mianwindow.h 里声明信号即可。代码如下，如下图黑色加粗部分代码就是创建的信号。这里创建一个 **`void pushButtonTextChanged();`**信号，定义信号最好是贴合信号本身的含义。笔者定义这个信号的意思是**按钮的文本发生改变后**的 signal。

mainwindow.h 添加信号后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  /* 引入 QPushButton */
6  #include <QPushButton>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15
16 signals:
17     /* 声明一个信号，只需声明，无需定义 */
18     void pushButtonTextChanged();
19
20 };
21 #endif // MAINWINDOW_H
```

第 16 至 18 行，声明一个信号 `void pushButtonTextChanged();`，可以看到信号无需 public 等关键字修饰。

5.3 如何在项目中创建槽

创建槽的方法也很简单，也是直接在 mianwindow.h 里直接声明槽，在 mianwindow.cpp 里实现槽的定义，**声明槽必须写槽的定义(定义指函数体的实现)，否则编译器编译时将会报错**。

槽有以下特点：

1. 槽可以是任何成员函数、普通全局函数、静态函数
2. 槽函数和信号的参数和返回值要一致

根据上面的槽特点，由于我们在 5.2 小节里声明了信号 `void pushButtonTextChanged();` 所以我们声明的槽函数必须是无返回值类型 void，和无需参数。所以声明槽的代码如下。此外我们还声明一个 QPushButton 对象 pushButton。对象 pushButton 可以写成简写 btn。这个根据个人习惯即可！简写的名称建议不要让人看不懂即可！同时还声明一个按钮点击的槽。

mainwindow.h 添加槽函数后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  /* 引入 QPushButton */
6  #include <QPushButton>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15
16
17 signals:
18     /* 声明一个信号，只需声明，无需定义 */
19     void pushButtonTextChanged();
20
21 public slots:
22     /* 声明一个槽函数 */
23     void changeButtonText();
24
25     /* 声明按钮点击的槽函数 */
26     void pushButtonClicked();
27
28 private:
28     /* 声明一个对象 pushButton */
30     QPushButton *pushButton;
31 };
32 #endif // MAINWINDOW_H
```

第 23 行，声明一个槽函数 `void changeButtonText();`。

第 26 行，声明了一个槽函数 `void pushButtonClicked();`。

第 31 行，声明了一个 QPushButton 对象 pushButton。

在 mainwindow.cpp 里实现声明的槽函数 `void changeButtonText();`和 `void pushButtonClicked();`。同时还实例化了 pushButton 对象。代码如下。

mainwindow.cpp 添加槽的实现代码

```
1  #include "mainwindow.h"
2
3  MainWindow::MainWindow(QWidget *parent)
4      : QMainWindow(parent)
5  {
6      /* 设置窗体的宽为 800,高为 480 */
7      this->resize(800,480);
8
9      /* 实例化 pushButton 对象 */
10     pushButton = new QPushButton(this);
11
12     /* 调用 setText()方法设定按钮的文本 */
13     pushButton->setText("我是一个按钮");
14 }
15
16 MainWindow::~MainWindow()
17 {
18
19 }
20
21 /* 实现按钮点击槽函数 */
22 void MainWindow::pushButtonClicked()
23 {
24     /* 使用 emit 发送信号 */
25     emit pushButtonTextChanged();
26 }
27
28 /* 实现按钮文本改变的槽函数 */
29 void MainWindow::changeButtonText()
30 {
31     /* 在槽函数里改变按钮的文本 */
32    pushButton->setText("被点击了！");
33 }
```

第 7 行，设置程序窗体的大小，假若不设置，窗体为最小分辨率。这里设置宽为 800，高为 480。刚好是正点原子 RGB LCD 屏里其中一个分辨率。实际开发需要以硬件的实际分辨率

开发即可。本教程都是以 800*480 分辨率开发。不建议使用低于这个分辨率开发，分辨率太低，显示的内容过少。

第 10 行，实例化 pushButton 对象，在堆中实例化，并指定父对象为 this。因为我们在 mainwindow.h 里声明的是指针类型的对象。实际上 `QPushButton(this)` 的原型是 `QPushButton(QWidget *parent = nullptr)`。在 C++基础里第 2.2.3 小节里就已经学过重载，实际上 QPushButton 类中还有下面两种初始化的方法。这也是 C++在 Qt 里重载的体现。

```
QPushButton(const QString &text, QWidget *parent = nullptr)
QPushButton(const QIcon &icon, const QString &text, QWidget *parent = nullptr)
```

在 22 行和 29 行里，实现了槽方法。

5.4 如何在项目中连接信号与槽

5.2 和 5.3 小节只是声明了信号和定义槽，要在项目中连接信号与槽。完成连接的代码如下。

信号槽连接的代码如下。

```
connect(pushButton, SIGNAL(clicked()), this, SLOT(pushButtonClicked()));
connect(this, SIGNAL(pushButtonTextChanged()), this, SLOT(changeButtonText()));
```

注意，发送信号的对象，和接收的信号的对象。因为我们 pushButtonClicked()是本类里定义的槽，所以用 this 来接收。同理，pushButtonTextChanged()也是本类定义的信号。所以发送者写成 this。changeButtonText()也是本类的槽函数，所以接收槽的对象也是 this。很多初学者不了解这里的信号的发送者和槽的接收者，究竟该写谁。我们要细读上面的代码和 5.1 小节的信号槽机制的定义慢慢理解。

在 mainwindow.cpp 中信号槽连接的代码如下。

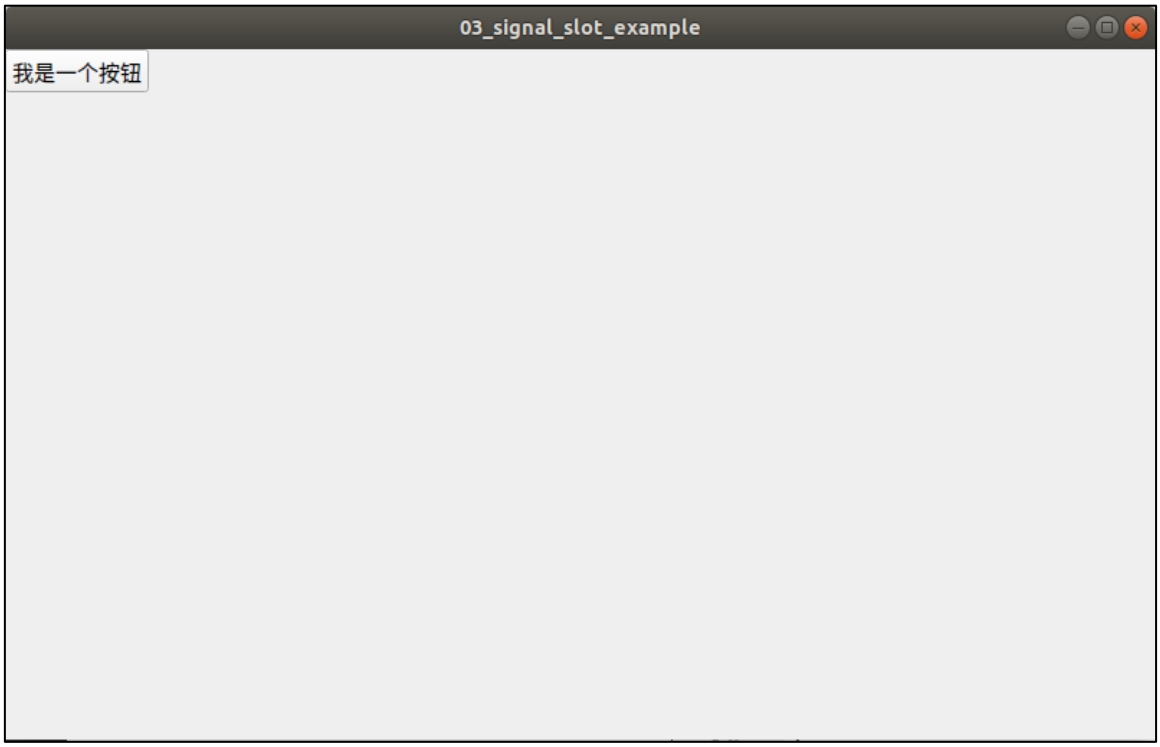
mainwindow.cpp 实现连接信号槽

```
1  #include "mainwindow.h"
2
3  MainWindow::MainWindow(QWidget *parent)
4      : QMainWindow(parent)
5  {
6      /* 设置窗体的宽为 800,高为 480 */
7      this->resize(800,480);
8
9      /* 实例化 pushButton 对象 */
10     pushButton = new QPushButton(this);
11
12     /* 调用 setText()方法设定按钮的文本 */
13     pushButton->setText("我是一个按钮");
14
15     /* 信号与槽连接 */
16     connect(pushButton, SIGNAL(clicked()), this,
                        SLOT(pushButtonClicked()));
```

```
17      connect(this, SIGNAL(pushButtonTextChanged()), this,
                              SLOT(changeButtonText()));
18  }
19
20  MainWindow::~MainWindow()
21  {
22
23  }
24
25  /* 实现按钮点击槽函数 */
26  void MainWindow::pushButtonClicked()
27  {
28      /* 使用 emit 发送信号 */
29      emit pushButtonTextChanged();
30  }
31
32  /* 实现按钮文本改变的槽函数 */
33  void MainWindow::changeButtonText()
34  {
35      /* 在槽函数里改变按钮的文本 */
36     pushButton->setText("被点击了！");
37  }
```

第 16、17 行，连接信号与槽，整个流程就是当点击了按钮，然后触发了 pushButtonClicked()，pushButtonClicked()槽里发送 pushButtonTextChanged()信号，changeButtonText()槽响应 pushButtonTextChanged()信号，我们在 changeButtonText()槽实现响应的动作（事件）。最终的实现效果是按钮的文本由“我是一个按钮”被点击时变成“被点击了!”。

编译程序及运行后，未点击按钮前如下。



点击按钮后。

上面的程序虽然简单，但是我们也最终实现了自定义的信号与槽。

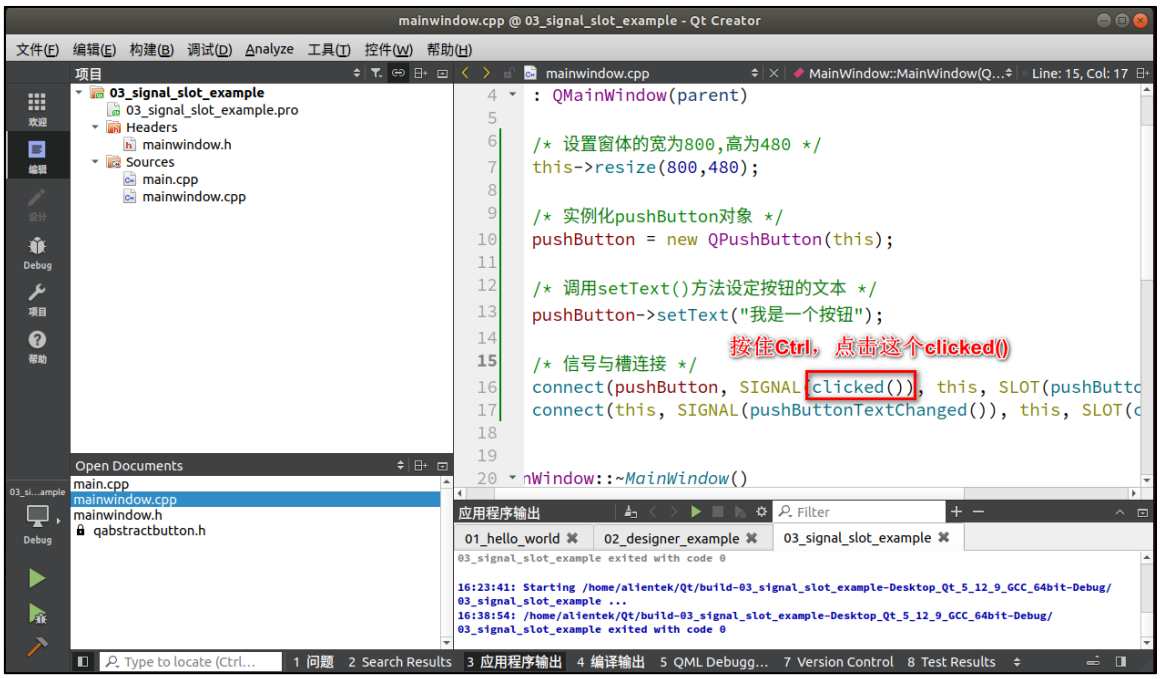
5.5 学会使用 Qt 类的信号与槽

Qt 里有大量的信号与槽，都是 Qt 自定义好的。基本够我们使用，如果没有找到想要的信号与槽，我们就可以按 5.2~5.4 小节自定义信号与槽的方法去定义自己的信号和槽了。那么我们该如何使用 Qt 信号与槽呢？

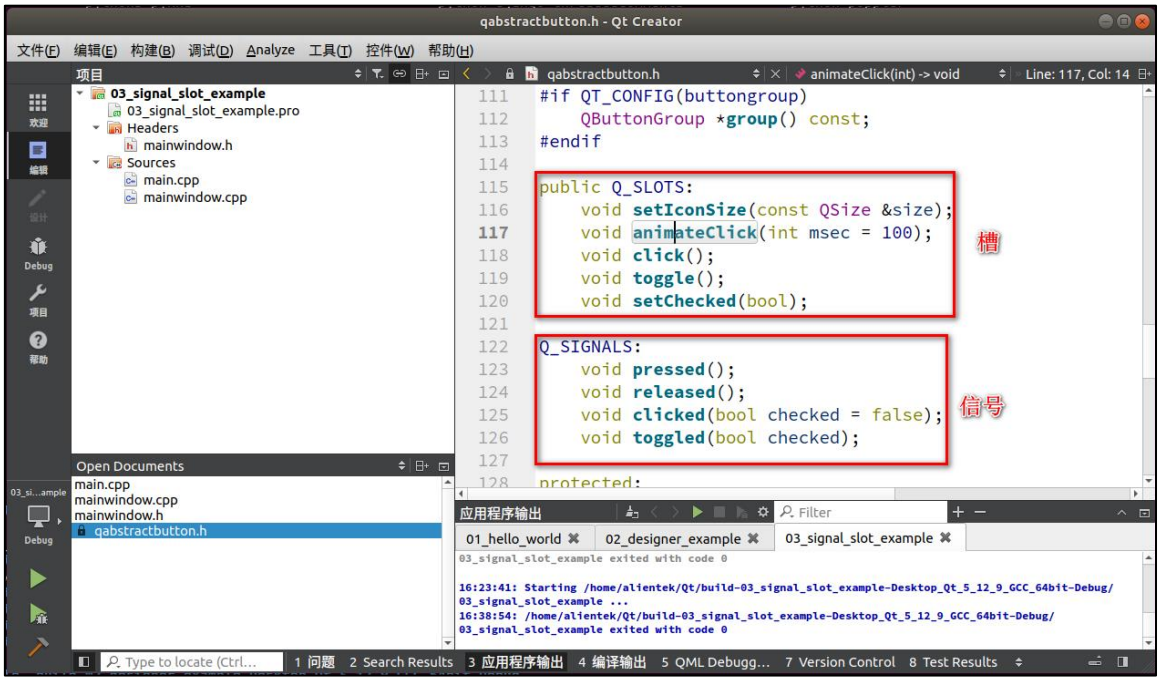
要想使用 Qt 的信号与槽，那么我们必须知道有哪些信号与槽。在 5.4 小节的代码里。

```
connect(pushButton, SIGNAL(clicked()), this, SLOT(pushButtonClicked()));
```

如下图示，按住 Ctrl 键，再点击 clicked()，进入 clicked()这个信号的定义处。



进入 QPushButton 的定义处，我们看到 QPushButton 不止 clicked 信号，还有其他信号，也有 QPushButton 的槽函数（返回上一步按 Alt + 方向左键）。在这里我们只是简单的看了如何在已知信号和槽里查找其他信号与槽。实际上在开发中我们经常需要使用 Qt 帮助文档来查看 Qt 定义的信号与槽。我们将在下一章里讲解 Qt 帮助文档的使用，帮助文档里面就有如何查看 Qt 的信号与槽等等。



第六章 Qt Creator 的使用技巧

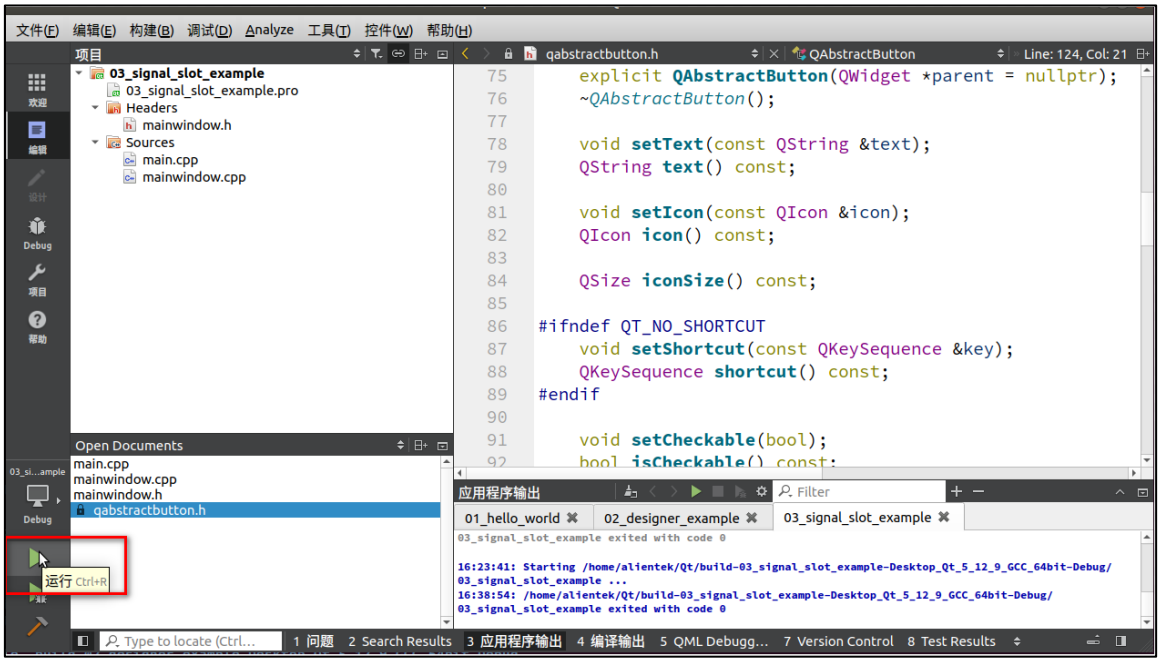
在任何一款编程的 IDE 软件里，都有相应的编程技巧。在这章里我们主要这两种最常用的技巧——Qt Creator 的快捷键的使用和 Qt 的帮助文档的使用。其中最重要的就是 Qt 的帮助文档里。可以说 Qt 帮助文档其实就是一本教程，任何其他的 Qt 教程都没有 Qt 文档写的详细，只不过是英文版本的而已。这样的对我们初学者来说入门还是有些吃力的。下面由笔者和大家的学习下 Qt Creator 的快捷键和 Qt 的帮助文档的使用方法。

6.1 Qt Creator 的快捷键

6.2 Qt 帮助文档的使用

6.1 Qt Creator 的快捷键

在 Qt Creator 里，假若自己不知道某些功能按钮的快捷键是什么，可以将鼠标移至该按钮上面就可以知道它的快捷键了。如下图，想知道运行的快捷键是什么，那么我们将鼠标移至 Qt Creator 的左下角的运行的按钮上，此时会提示运行快捷键是“Ctrl + R”。当然需要在英文输入法的前提下才能生效。

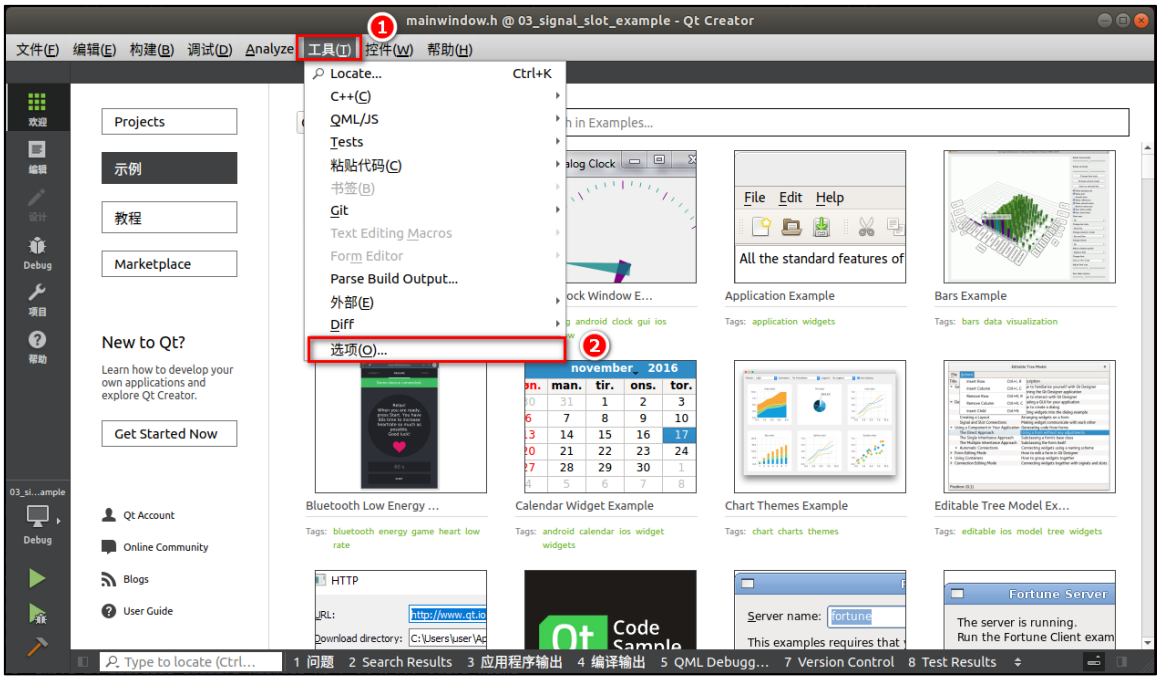


Qt 快捷键有很多，我们不可能能记住那么多快捷键。我们只需要使用常用的快捷键即可。像简单的 Ctrl + C 和 Ctrl + V 复制粘贴等这些我们就不说了，这些都与 Windows 下的操作一样。常用的快捷键有如下表格。

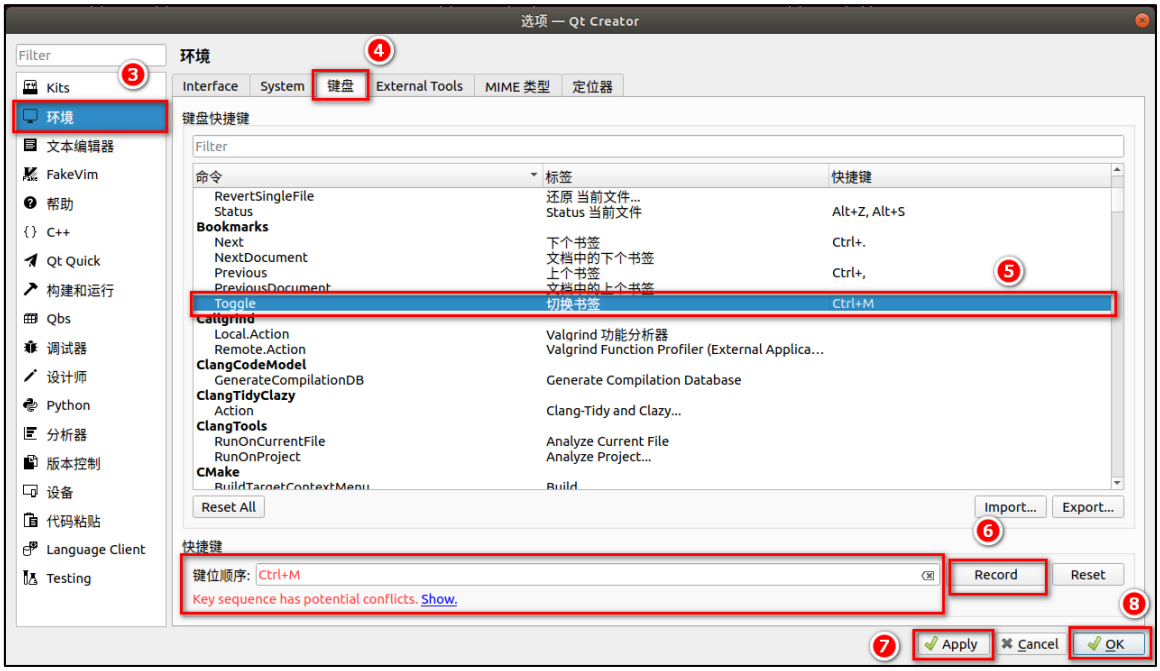
一般快捷键	新建文件或项目(N)	Ctrl + N
	关闭当前窗口/关闭当前文件	Ctrl + W
	关闭所有文件	Ctrl + Shfit + W
	关闭当前文件（windows）	Ctrl + F4
	运行	Ctrl + R
	返回上一级（返回），常用于跳转代码	Alt + ←（方向左键）
	进入下一级（前进），常用于跳转代码	Alt + →（方向右键）
	Qt 会自动排版对齐代码	Ctrl + I
	减小字体大小	Ctrl+- (Ctrl+鼠标滚轮向下)
	增加字体大小	Ctrl++ (Ctrl+鼠标滚轮向上)
	重置字体大小	Ctrl+0
	折叠	Ctrl+<

常用编辑快捷键	展开	Ctrl+>
	复制行	Ctrl+Ins
	复制到行下	Ctrl+Alt+Down
	复制到行上	Ctrl+Alt+Up
	在当前行上方插入新行	Ctrl+Shift+Enter
	在当前行下方插入新行	Ctrl+Enter
	查看剪切板历史	Ctrl+Shift+V
	剪切行	Shift+Del
	追加行	Ctrl+J
	向下移动当前行	Ctrl+Shift+Down
	向上移动当前行	Ctrl+Shift+Up
	切换函数声明/定义	Ctrl + 鼠标左键/Shift + F2
	编辑信号和槽	F4
	跳转至以}结尾的块	Ctrl+}
	跳转至以{开始的块	Ctrl+{
	打开类型层次窗口	Ctrl+Shift+T

上面的快捷键可能会与 Ubuntu 或者 Windows 系统的快捷键冲突，或者是有些功能没有定义快捷键，我们也可以在 Qt Creator 里自定义快捷键或者修改原有的快捷键，步骤如下。



在“环境”项下找到键盘，如图，⑤处，切换书签的快捷键，“Ctrl + M”显示红色，说明是与系统的快捷键冲突了。我们可以选中这项，按⑥处“Record”重新记录切换书签的快捷键。再点击“Apply”和“OK”即可。有些命令还没有定义快捷键，如果我们希望使用这个命令使用快捷键，那么我们也可以按如下步骤记录自定义我们的快捷键即可，注意不要与系统的快捷键重复。

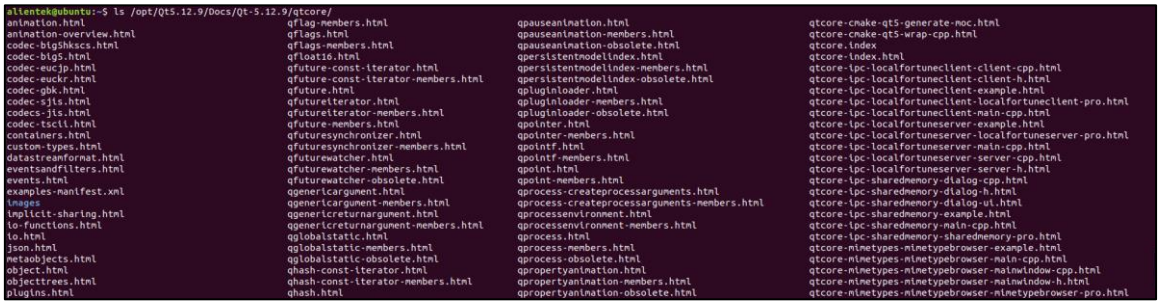


以上 Qt Creator 的快捷键仅供参考，在编程里实际上常用的也不会很多，只需要使用常用的即可！

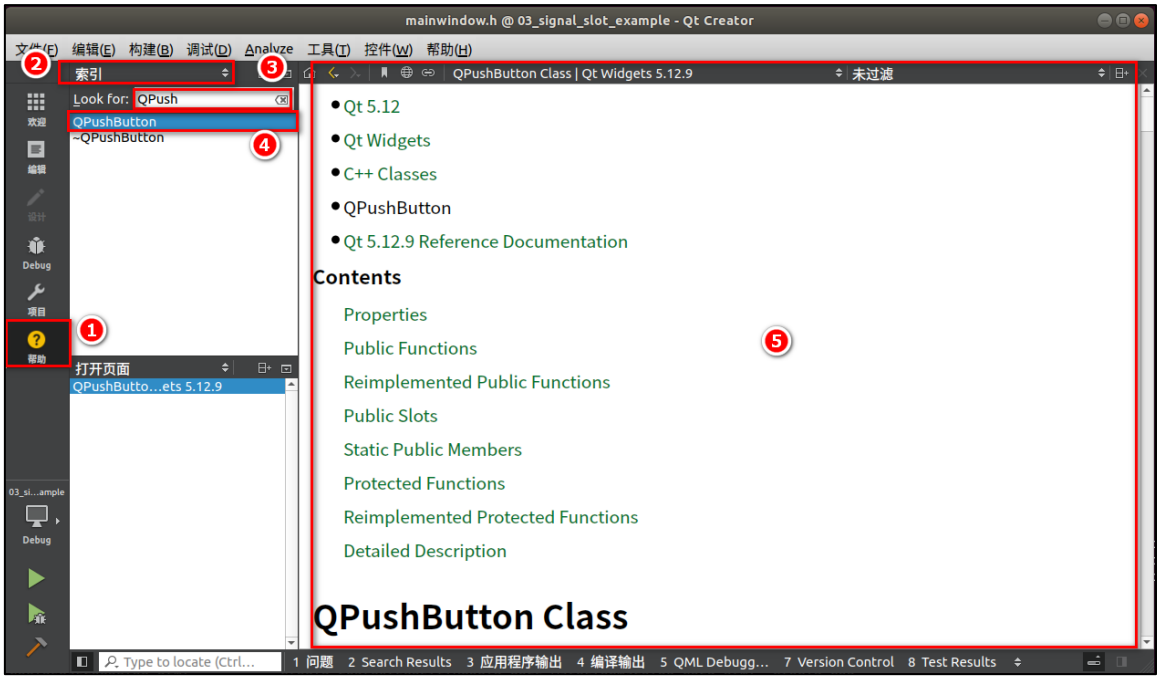
6.2 Qt 帮助文档的使用

在任何一款 IDE 编程软件里，都离不开参考帮助文档。有些 IDE 编程软件需要手动下载这些帮助文档。Qt Creator 则不需要，在我们安装 Qt 时，帮助文档已经安装在我们的“安装目录/Qt5.12.9/Docs/”下，使用的是 html 文本的方式，我们可以使用浏览器打开这种 html 文本。

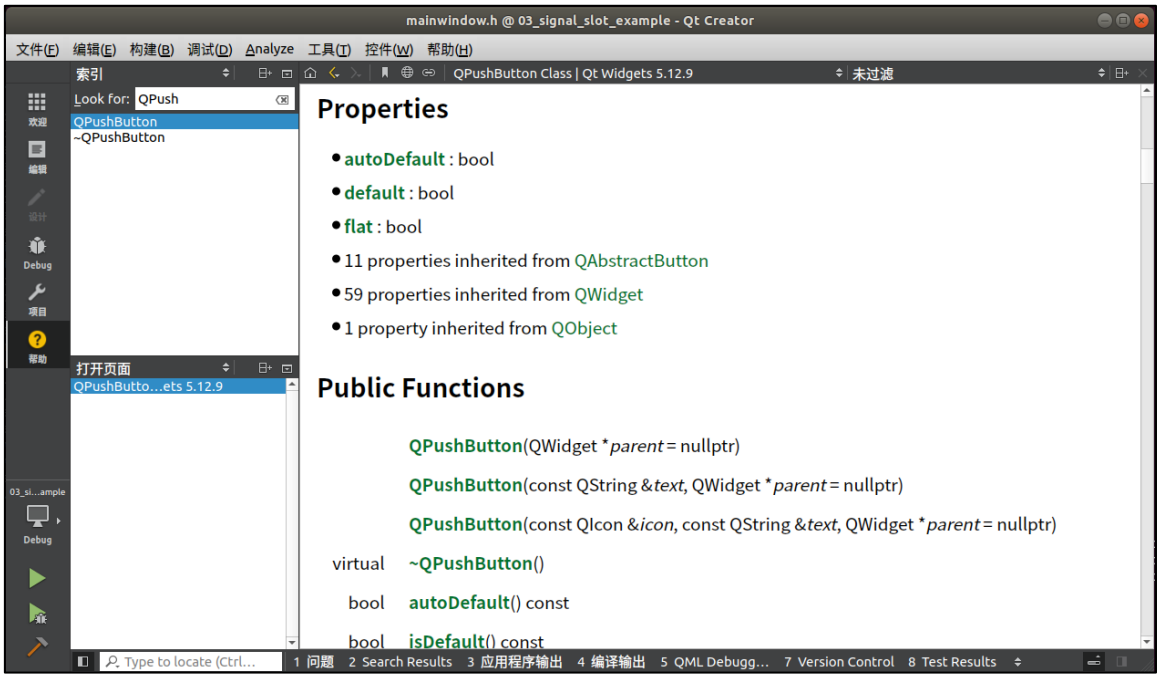
在 Ubuntu 下的安装目录下的帮助文档如下图，下图是 QtCore（Qt 的核心模块）下的帮助文档。



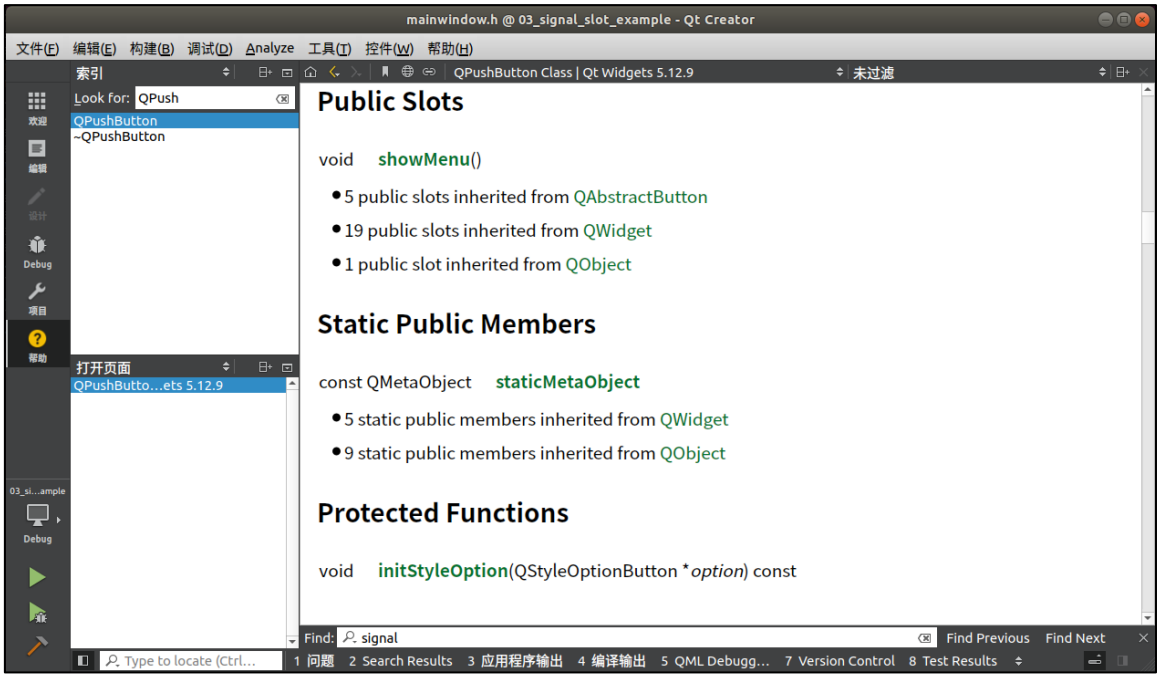
我们也可以在 Qt Creator 下直接搜索相应的关键字。如下图步骤，比如我们要查看 QPushButton 类，先点击①处的帮助再点击②处的下拉选为索引，在③处输入“QPushButton”，也可以只需要输入“QPush”匹配索引值即可。点击匹配的选项，如下图⑤处，就是我们探索出来的帮助文档。



如何使用搜索出来的帮助文档呢？比如要查看 QPushButton 类有哪些属性和函数可以用可以查看“Properties”和“Public Functions”这个标题下的属性和函数，如下图。可以看到 QPushButton 类有如下属性和函数可以使用。



再比如，假若我们不知道这个 QPushButton 类有哪些可用的槽，我们可以继续往下查看，找到 Public Slots 即可！我们查看某个方法用法时，直接单击进入查看某个方法的用法即可！别忘记使用上一小节的快捷键（Alt + ←（方向左键））返回到上一级，或者进入下一级（Alt + →（方向右键））了。



Qt 的帮助文档使用方法流程基本如上了。初学者可以阅读这些的英文文档会感到吃力，可能会问，有没有中文的帮助文档？我们要知道这种都是国外软件，因为帮助文档可能会更新变动，且翻译起来要人力和时间，况且翻译的原意有可能是不贴近原来的 Qt 帮助文档。我们还是老老实实看 Qt 最权威最全面的帮助文档即可！要想学好 Qt，帮助文档是少看不了的！其实 Qt 的帮助文档就是一本十分庞大且非常好的教程了，我们这个教程只是引领大家入门 Qt，引领大家在 Qt 这个非常庞大的文档里学习常用的控件和方法。Qt 自带有非常多的例子可以参考，这点 Qt 是做的很好的。看帮助文档，我们还需要多多练习，并不是看帮助文档就会了，实践起来才知道，基础都是靠多练的。

第七章 Qt 控件

老子曾说：“天下难事，必做于易；天下大事，必做于细”。再复杂的项目，都是由一个个小小的部分组成，只有掌握了 Qt 的基本，遇到大项目也不用担心了！从这章开始我们开始学习 Qt 的窗口部件，其中每种类型的窗口部件都选取一个作为例子讲解它们的用法，通过例子大家能举一反三。本章也是纯代码编程，不使用 Qt Designer 图形界面开发程序。笔者认为纯代码编程的代码容易让人看懂，在 Qt Designer 里设计就显得没有过多逻辑可言。在这章里我们可以学习**常用的控件初始化方法**，设置窗体的大小位置，颜色，文本，设计创意的例子快速学习各类控件的使用方法，力求把例子写的实用，代码注释详细。因为控件非常多，如果觉得学习控件枯燥，**可以等用到这个控件（部件）时再参考，不必要一次性全部掌握，这章节的目的是了解 Qt 控件类型及使用方法**。

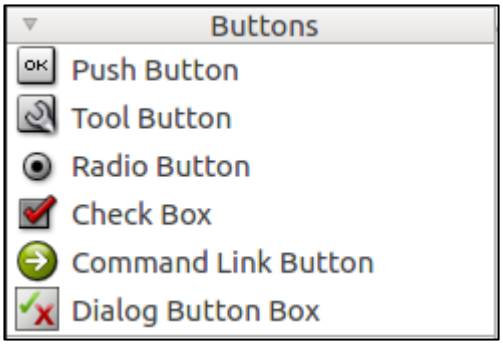
值得留意的小节是 7.1.3 小节，该小节讲解如何添加资源图片和 qss 文件。后面的例程都可以参考 7.1.3 小节添加资源图片，不再重复写这种添加步骤。

第 7.8 和 7.9 小节，我们在嵌入式里经常用于处理数据，建立模型，应该花多点时间学习这两个小节。

7.1 按钮

在 Qt 里，最常用使用的控件就是按钮了，有了按钮，我们就可以点击，从而响应事件，达到人机交互的效果。不管是嵌入式或者 PC 端，界面交互，少不了按钮。

Qt 按钮部件是一种常用的部件之一，Qt 内置了六种按钮部件如下：



（1） QPushButton:下压按钮

（2） QToolButton:工具按钮

（3） QRadioButton:选择按钮

（4） QCheckBox:检查框

（5） QCommandLinkButton:命令链接按钮

（6） QDialogButtonBox:对话框按钮

这六种按钮部件作用简介如下：

QPushButton 继承 QAbstractButton 类，被 QCommandLinkButton 继承。通常用于执行命令或触发事件。

QToolButton 继承 QAbstractButton 类。是一种用于命令或者选项的可以快速访问的按钮，通常在 ToolBar 里面。工具按钮通常显示的是图标，而不是文本标签。ToolButton 支持自动浮起。在自动浮起模式中，按钮只有在鼠标指向它的时候才绘制三维的框架。

QRadioButton 继承 QAbstractButton 类。RadioButton 单选按钮（单选框）通常成组出现，用于提供两个或多个互斥选项。

QCheckBox 继承 QAbstractButton。复选按钮（复选框）与 RadioButton 的区别是选择模式，单选按钮提供多选一，复选按钮提供多选多。

QCommandLinkButton 控件中文名是“命令链接按钮”。QCommandLinkButton 继承 QPush Button。QCommandLinkButton 控件和 RadioButton 相似，都是用于在互斥选项中选择一项。表面上同平面按钮一样，但是 CommandLinkButton 除带有正常的按钮上的文字描述文本外，默认情况下，它也将携带一个箭头图标，表明按下按钮将打开另一个窗口或页面。

QDialogButtonBox 按钮盒子（按钮框），是由 QDialogButtonBox 类包装成的。QDialogButtonBox 继承 QWidget。常用于对话框里自定义按钮，比如“确定”和“取消”按钮。

上面说的六种按钮的可用属性，信号和槽，需要用到时可在 Qt 帮助文档里查看。这里我们就略过介绍它们可用属性和信号与槽了。下面我们通过例子讲解每种按钮是如何使用，一起探究它们究竟能实现什么效果。

7.1.1 QPushButton

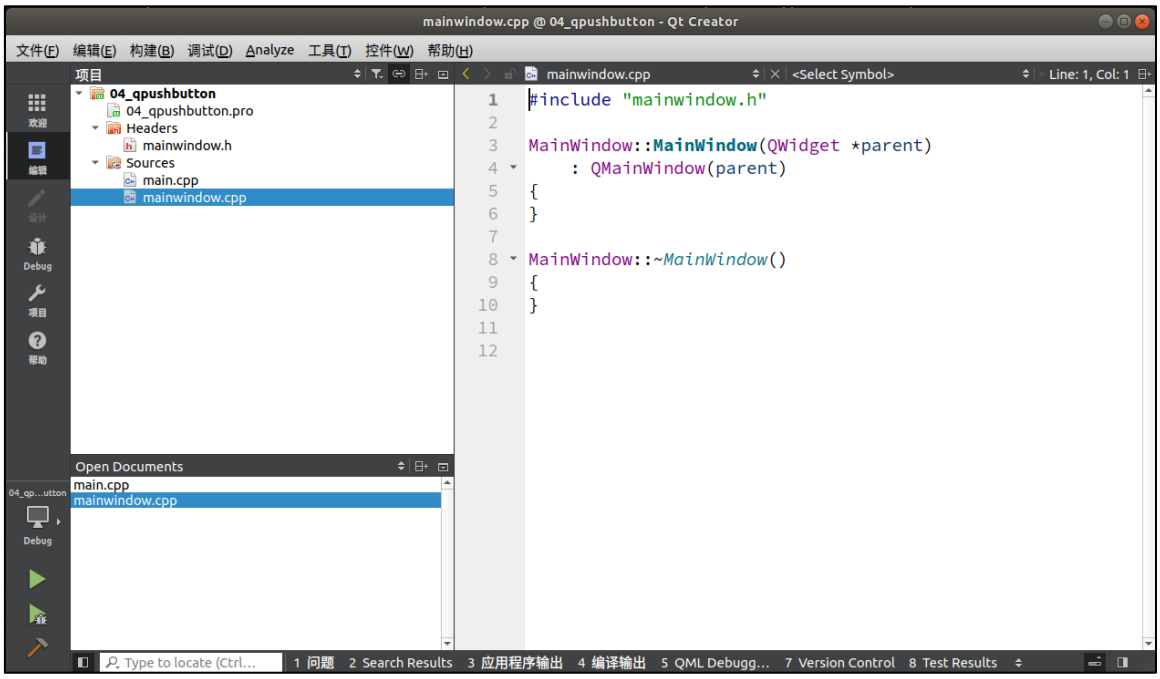
7.1.1.1 控件简介

在第四章里我们就已经接触过 QPushButton 了，在 Qt Designer 里连接信号与槽，从而实现了关闭程序的功能。下面开始重新用编写程序的方法实现使用 QPushButton 连接信号和槽实现一个小例子。

7.1.1.2 用法示例

例 04_qpushbutton 窗口换肤（难度：简单），通过单击不同的按钮，改变窗口的颜色。

新建一个项目名称为为 04_qpushbutton，在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。如果还不会新建一个项目，建议回到 3.6 小节查看如何新建一个项目。完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  /* 引入 QPushButton 类 */
6  #include <QPushButton>
```

```
7
8   class MainWindow : public QMainWindow
9   {
10      Q_OBJECT
11
12  public:
13      MainWindow(QWidget *parent = nullptr);
14      ~MainWindow();
15
16  private:
17      /* 声明一个 QPushButton 对象 pushButton1 */
18      QPushButton *pushButton1;
19      /* 声明一个 QPushButton 对象 pushButton2 */
20      QPushButton *pushButton2;
21
22  private slots:
23      /* 声明对象 pushButton1 的槽函数 */
24      void pushButton1_Clicked();
25      /* 声明对象 pushButton2 的槽函数 */
26      void pushButton2_Clicked();
27  };
28  #endif // MAINWINDOW_H
```

第 6 行，引入 QPushButton 类。

第 18 和 20 行，声明两个 QPushButton 的对象。

第 24 和 26 行，声明两个 QPushButton 对象的槽函数。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置宽高为 800×480,位置在 0, 0。（0, 0）代表原点，Qt 默认最左上角的点为原
点 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 实例化两个按钮对象，并设置其显示文本为窗口皮肤 1 和窗口皮肤 2 */
10      pushButton1 = new QPushButton("窗口皮肤 1", this);
11      pushButton2 = new QPushButton("窗口皮肤 2", this);
12
13      /* 设定两个 QPushButton 对象的位置 */
```

```
14      pushButton1->setGeometry(300,200,80,40);
15      pushButton2->setGeometry(400,200,80,40);
16
17      /* 信号槽连接 */
18      connect(pushButton1, SIGNAL(clicked()), this,
                              SLOT(pushButton1_Clicked()));
19      connect(pushButton2, SIGNAL(clicked()), this,
                              SLOT(pushButton2_Clicked()));
20  }
21
22  MainWindow::~MainWindow()
23  {
24  }
25
26  /* 槽函数的实现 */
27  void MainWindow::pushButton1_Clicked()
28  {
29      /* 设置主窗口的样式 1 */
30      this->setStyleSheet("QMainWindow { background-color: rgba(255, 245,
                             238, 100%); }");
31  }
32
33  /* 槽函数的实现 */
34  void MainWindow::pushButton2_Clicked()
35  {
36      /* 设置主窗口的样式 2 */
37      this->setStyleSheet("QMainWindow { background-color: rgba(238, 122,
                             233, 100%); }");
38  }
```

第 7 行，设置程序窗口的显示位置和显示大小，如果不设置，运行的程序窗口在 Ubuntu 里有可能在某个位置出现，而在 Windows 一般出现在中间。

第 10 和 11 行，实际化 QPushButton 对象。在初始化的时候可以传入 QString 类型串，作为按钮的显示文本。

第 14 行，设置按钮的大小和位置，按钮的大小不能设置过小，否则按钮上的文本可能显示不全。

第 18 行和 19 行，连接两个 QPushButton 对象的信号与槽。

第 27 行至 38 行，两个 QPushButton 的槽函数实现，设置主窗体的样式表，其中设置 background-color 的 rgba 参数即可改变窗体的背景颜色。关于什么是样式表，如何设置样式表，以后会以一个小节专门讲解。

在源文件“main.cpp”具体代码如下，由新建项目时生成，无改动。

mainwindow.cpp 源代码

```
1  #include "mainwindow.h"
2
3  #include <QApplication>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8      MainWindow w;
9      w.show();
10     return a.exec();
11 }
```

7.1.1.3 运行效果

程序编译及运行后，点击窗口皮肤 1 按钮，主窗体显示效果如下。

点击窗口皮肤 2 按钮，主窗体显示效果如下。

7.1.2 QToolButton

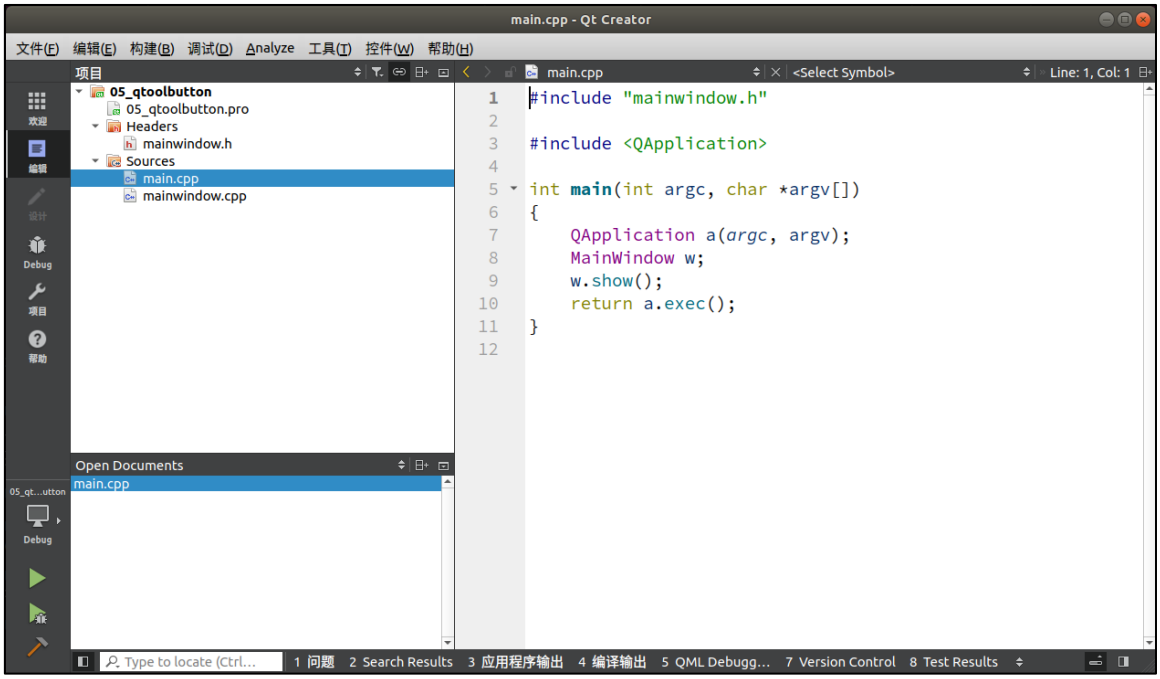
7.1.2.1 控件简介

工具按钮（QToolButton）区别于普通按钮（QPushButton）的一点是，工具按钮（QToolButton）可以带图标。这里区别下图标和按钮的背景图片是不一样的。通常我们在 QToolBar 这种工具条（工具栏）上设置不同的按钮，如果这些按钮还带图标和文本，那么 QToolButton 是个不错的选择。

7.1.2.2 用法示例

例 05_qtoolbutton 自定义工具栏（难度：简单）。

新建一个项目名称为 05_qtoolbutton。在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  /* 引入 QToolButton 类 */
6  #include <QToolButton>
7  /* 引入 QToolBar 类 */
8  #include <QToolBar>
9
10 class MainWindow : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     MainWindow(QWidget *parent = nullptr);
16     ~MainWindow();
17
18 private:
19     /* 声明一个 QToolButton 对象 */
20     QToolButton *toolButton;
21     /* 声明一个 QToolBar 对象 */
22     QToolBar *toolBar;
23 };
24 #endif // MAINWINDOW_H
```

第 20 和 22 行，声明 QToolButton 对象和 QtoolBar 对象。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1  #include "mainwindow.h"
2  #include <QApplication>
3  #include <QStyle>
4
5  MainWindow::MainWindow(QWidget *parent)
6      : QMainWindow(parent)
7  {
8      /* 设置主窗体的位置和大小 */
9      this->setGeometry(0, 0, 800, 480);
10
11     /* 实例化 QToolBar 对象 */
12     toolBar = new QToolBar(this);
13     /* 设置 toolBar 的位置和大小 */
14     toolBar->setGeometry(0, 0, 800, 100);
15
16     /* 实例化 QStyle 类对象，用于设置风格，调用系统类自带的图标 */
17     QStyle *style = QApplication::style();
18
19     /* 使用 Qt 自带的标准图标，可以在帮助文档里搜索 QStyle::StandardPixmap */
20     QIcon icon =
       style->standardIcon(QStyle::SP_TitleBarContextHelpButton);
21
22     /* 实例化 QToolButton 对象 */
23     toolButton = new QToolButton();
24
25     /* 设置图标 */
26     toolButton->setIcon(icon);
27     /* 设置要显示的文本 */
28     toolButton->setText("帮助");
29     /* 调用 setToolButtonStyle()方法，设置 toolButoon 的样式，设置为文本置于
图标下方 */
30     toolButton->setToolButtonStyle(Qt::ToolButtonTextUnderIcon);
31
32     /* 最后将 toolButton 添加到 ToolBar 里 */
33     toolBar->addWidget(toolButton);
34 }
35
36 MainWindow::~MainWindow()
37 {
```

```
38  }
```

这段代码的流程是，初始化 toolBar（工具条/工具栏）对象，然后初始化 toolButton（工具按钮）对象，设置工具按钮的样式。最后将 toolButton（工具按钮）添加到 toolBar（工具条/工具栏）上。这样就完成了自定义工具栏的设计。

在源文件“main.cpp”具体代码如下，由新建项目时生成，无改动。

mainwindow.cpp 源代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.1.2.3 运行效果

最终程序实现的效果如下。成功的将自定义的工具按钮嵌入工具栏中。在许多含有工具栏的软件都可以看到这种设计，都可以使用 Qt 来实现类似的功能。

7.1.3 QRadioButton

7.1.3.1 控件简介

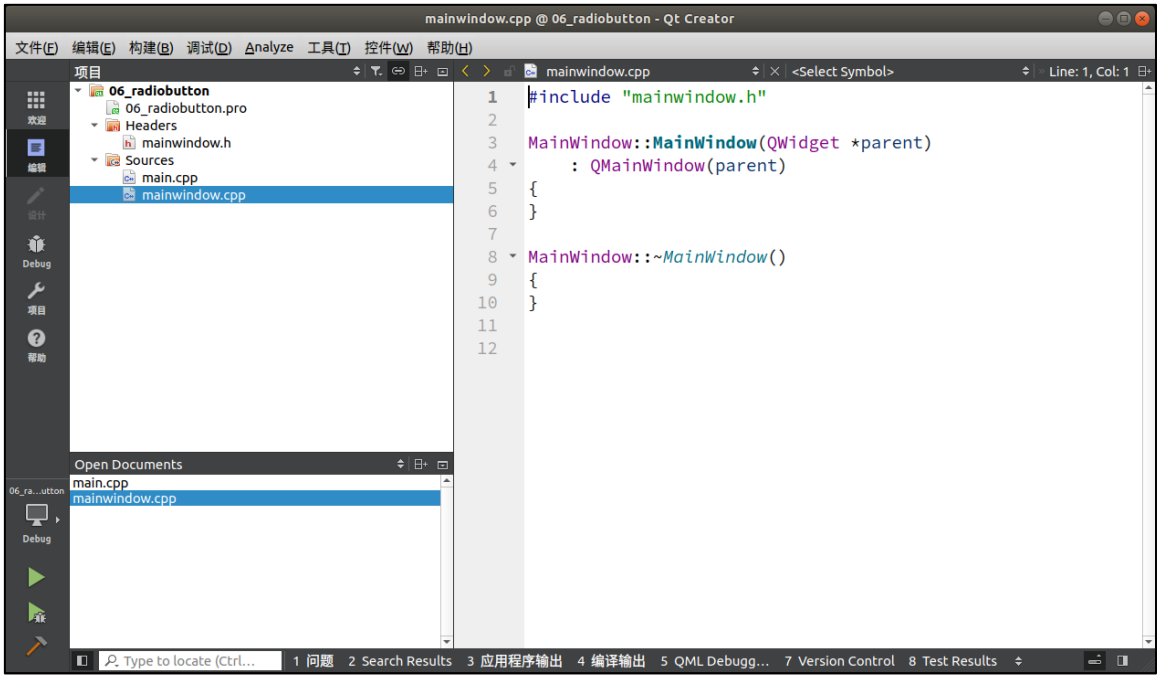
QRadioButton 部件提供了一个带有文本标签的单选框（单选按钮）。

QRadioButton 是一个可以切换选中（checked）或未选中（unchecked）状态的选项按钮。单选框通常呈现给用户一个“多选一”的选择。也就是说，在一组单选框中，一次只能选中一个单选框。默认在同一个父对象下，初始化后点击它们是互斥状态。

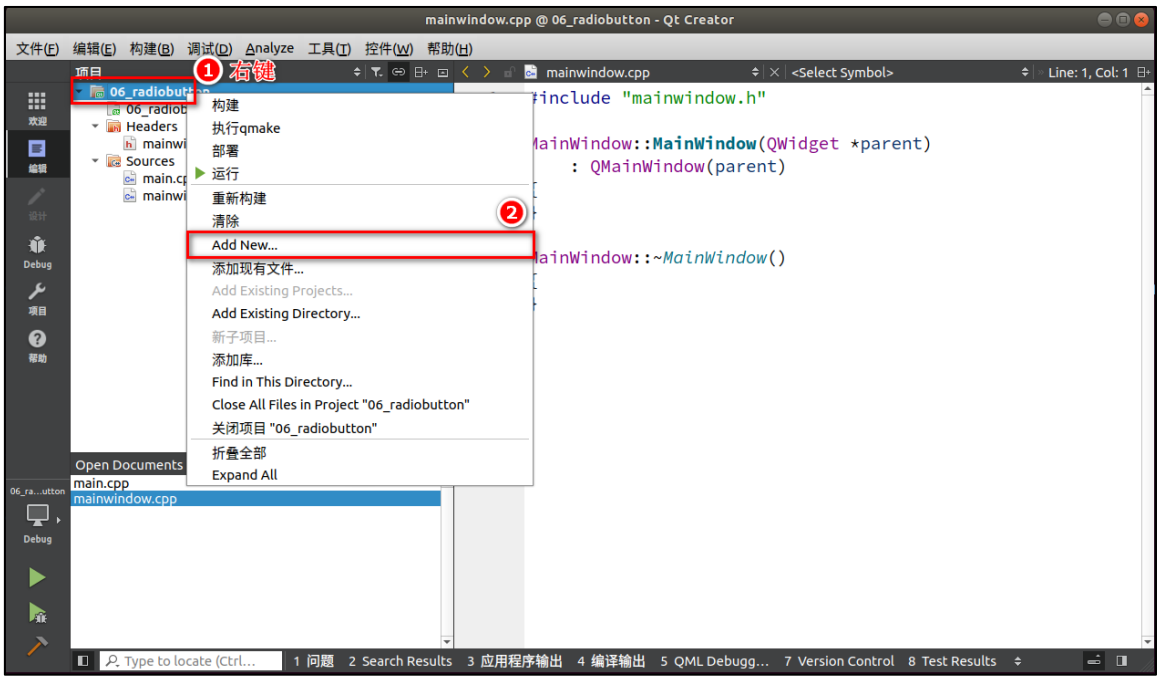
7.1.3.2 用法示例

例 06_radiobutton 仿手机开关效果（难度：中等）。本例将实现手机开关效果，需要使用到 Qt 样式表，加载 qss 样式表文件，与 7.1.1 小节类似，只是把样式表写到 qss 文件里了。这里我们慢慢接触 Qt 的样式表了，正因为有样式表我们才能写一些比较有实际应用的例子和比较炫的例子。

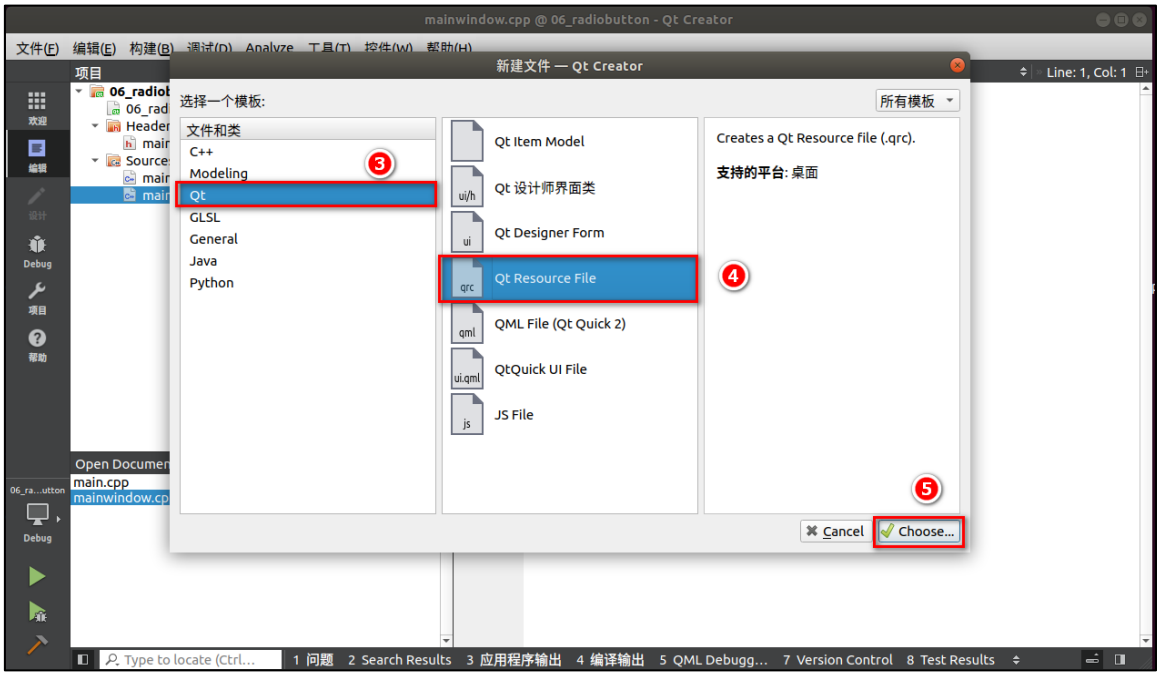
在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



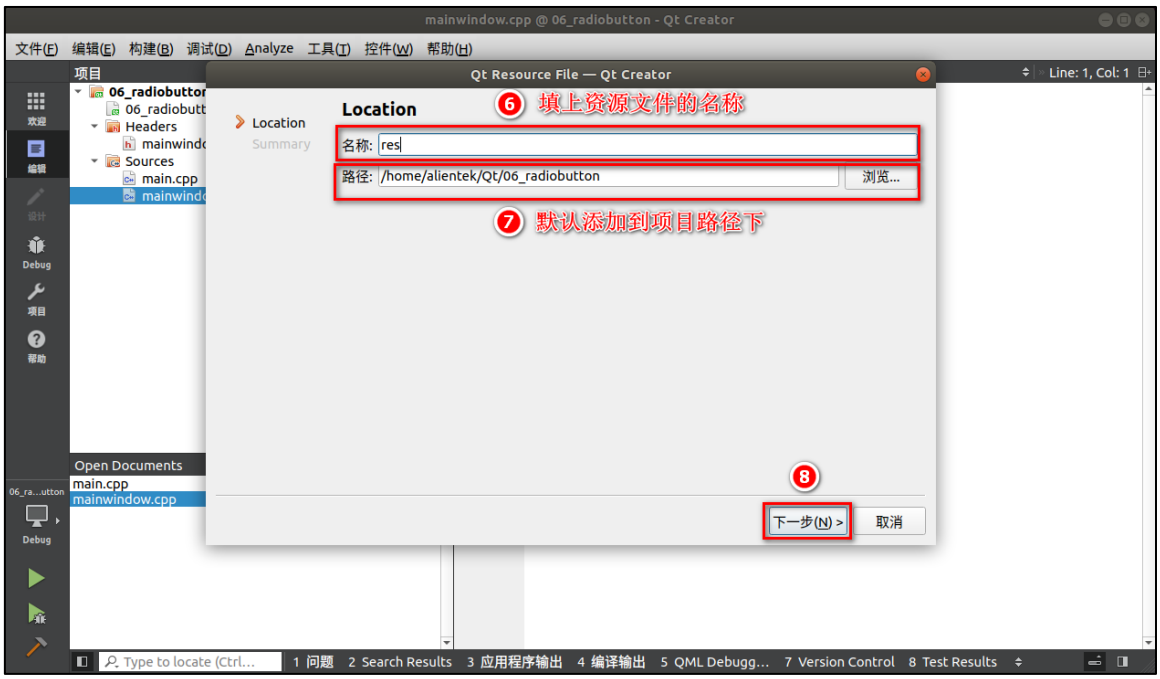
添加资源文件，按如下步骤。右键项目，选择 Add New…。



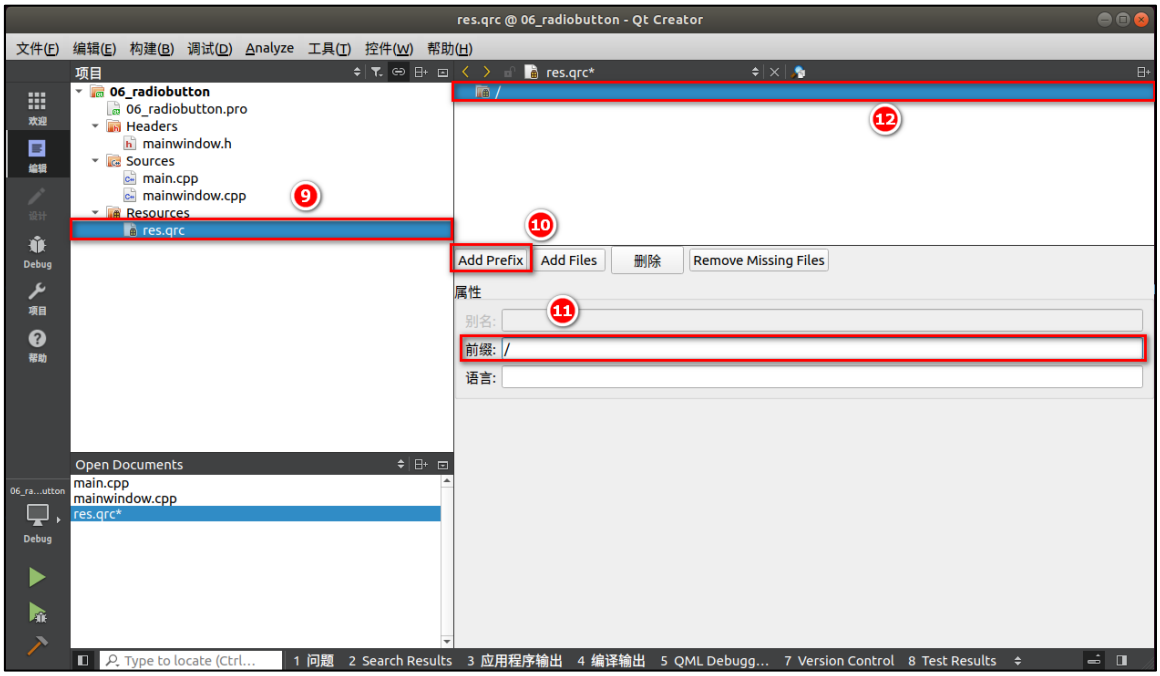
选择一个模板，选择 Qt 模板，再选择 Qt Resource Files，点击 Choose。这里 Qt 模板我们就不详细说了，日后我们需要使用什么模板，用到再了解。现在没必要一下子各个模板的用法，实际上我们常用的不多。



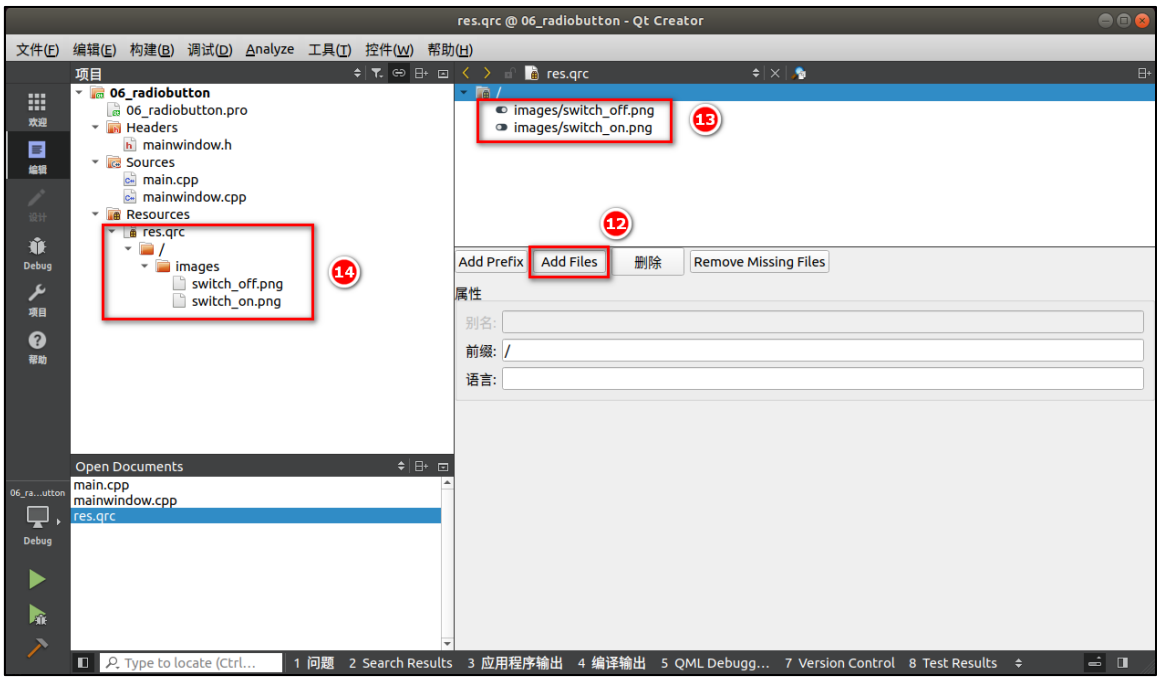
填上资源文件的名称（可随意写一个，笔者简写为 res），默认添加项目路径下。后面的步骤默认即可，点击完成。



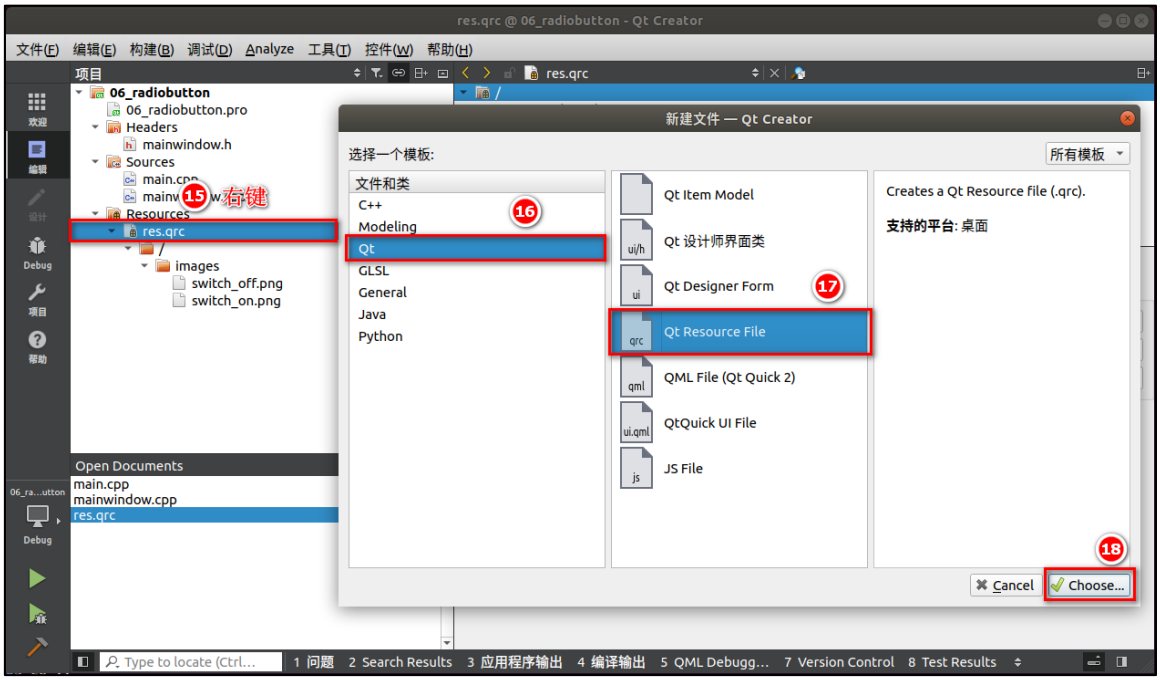
新建完成了资源文件后，默认会进入 res.qrc 文件编辑模式（如果关闭了，可以右键这个文件点击选择“Open in Editor”），点击 Add Prefix 添加前缀，添加前缀的目的是方便分类管理文件，比如我们现在第⑪处添加了前缀/。“/”一定需要写，否则会找不到路径，这有点像 Linux 的根节点一样。



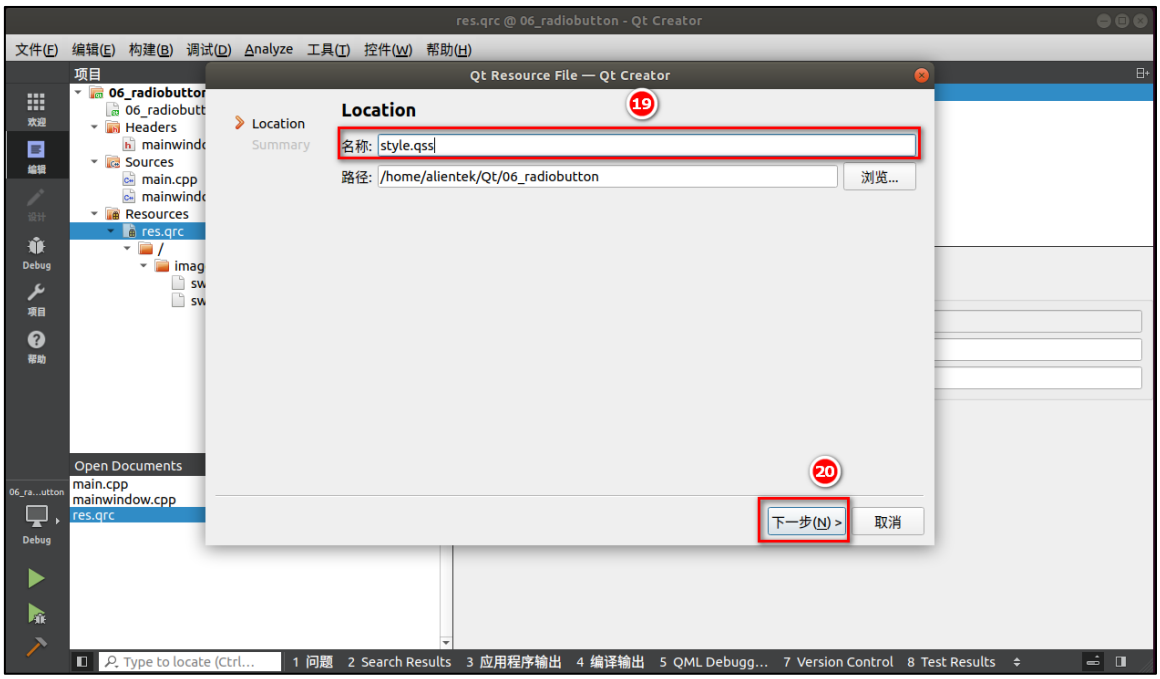
添加了前缀后，我们添加资源图片，放在/images 前缀的下面。这里我们准备了两张图片，在本项目路径 images 文件夹（images 文件夹先手动创建）下。如下图步骤，添加完成需要按“Ctrl + S”保存 res.qrc 才会看到左边的结果。添加完成如下图。



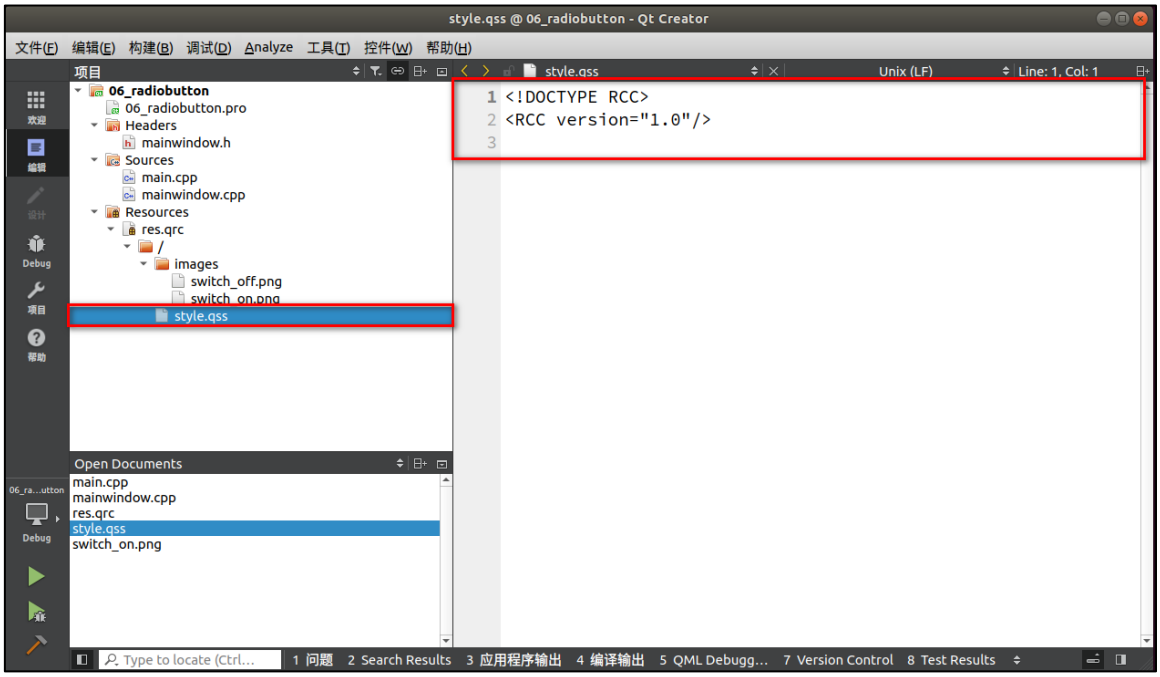
添加 qss 文件。QSS 文件是使用 Qt 程序相关联的样式表文件。它由 GUI 元素的外观和感觉，包括布局，颜色，鼠标的行为，大小和字体。它的风格，一个可以合并到一个 UI（用户界面）。与 HTML 的 CSS 类似，Qt 的样式表是纯文本的格式定义，在应用程序运行时可以载入和解析这些样式定义，从而使应用程序的界面呈现不同的效果。



新建一个 style.qss 文件，如下图，默认添加到项目的路径下，后面步骤默认即可，直至完成。



qss 文件添加后如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  /* 引入 QRadioButton */
6  #include <QRadioButton>
7
```

```
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15
16 private:
17     /* 声明两个 QRadioButton 对象 */
18     QRadioButton *radioButton1;
19     QRadioButton *radioButton2;
20 };
21 #endif // MAINWINDOW_H
```

在第 18 和 19 行声明两个 QRadioButton 对象。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1  #include "mainwindow.h"
2
3  MainWindow::MainWindow(QWidget *parent)
4      : QMainWindow(parent)
5  {
6      /* 主窗体设置位置和显示的大小 */
7      this->setGeometry(0, 0, 800, 480);
8      this->setStyleSheet("QMainWindow {background-color: rgba(200, 50,
100, 100%);}");
9
10     /* 实例化对象 */
11     radioButton1 = new QRadioButton(this);
12     radioButton2 = new QRadioButton(this);
13
14     /* 设置两个 QRadioButton 的位置和显示大小 */
15     radioButton1->setGeometry(300, 200, 100, 50);
16     radioButton2->setGeometry(400, 200, 100, 50);
17
18     /* 设置两个 QRadioButton 的显示文本 */
19     radioButton1->setText("开关一");
20     radioButton2->setText("开关二");
21
22     /* 设置初始状态，radioButton1 的 Checked 为 false,另一个为 true*/
23     radioButton1->setChecked(false);
24     radioButton2->setChecked(true);
```

```
25  }
26
27  MainWindow::~MainWindow()
28  {
29  }
```

第 23 行和 24 行，设置 QRadioButton 对象的初始化状态，让它们互斥。

在源文件“main.cpp”具体代码如下。我们需要在 main.cpp 里加载 qss 文件。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4   /* 引入 QFile */
5   #include <QFile>
6
7   int main(int argc, char *argv[])
8   {
9       QApplication a(argc, argv);
10      /* 指定文件 */
11      QFile file(":/style.qss");
12
13      /* 判断文件是否存在 */
14      if (file.exists() ) {
15          /* 以只读的方式打开 */
16          file.open(QFile::ReadOnly);
17          /* 以字符串的方式保存读出的结果 */
18          QString styleSheet = QLatin1String(file.readAll());
19          /* 设置全局样式 */
20          qApp->setStyleSheet(styleSheet);
21          /* 关闭文件 */
22          file.close();
23      }
24      MainWindow w;
25      w.show();
26      return a.exec();
27  }
```

第 11 行至 23 行，读取 style.qss 的内容。并设置全局样式。

在源文件“style.qss”具体代码如下，与 HTML 里的 css 语法相似。如果不会写 qss 的内容，可以参考 Qt 帮助文档的内容，在里面搜索“qt style”。在里面找相关的例子参考，这里我们只是初步了解下这个 qt style。

style.qss 编程后的代码

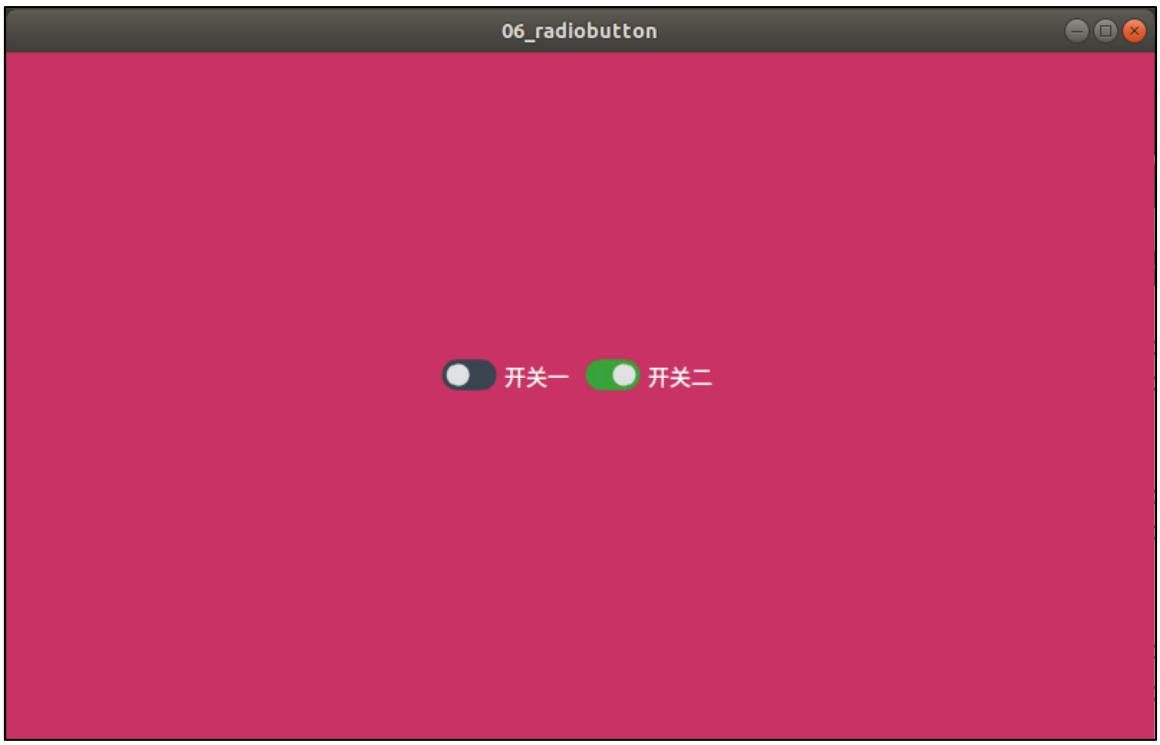
```
1   QRadioButton{
2       spacing: 2px;
```

```
3       color: white;
4   }
5   QRadioButton::indicator {
6       width: 45px;
7       height: 30px;
8   }
9   QRadioButton::indicator:unchecked {
10      image: url(:/images/switch_off.png);
11  }
12  QRadioButton::indicator:checked {
13      image: url(:/images/switch_on.png);
14  }
```

在第 10 行和第 13 行，设置 QRadioButton 的 indicator 的背景图片。这样当它们点击切换时就会看到类似开关切换的效果了。

7.1.3.3 运行效果

编译程序运行的效果如下。点击关闭开关一，开关二即打开；点击开关二，开关一即打开。因为它们默认是互斥的效果。在某种情况下我们需要使用这种效果，比如我们在网上看视频时经常需要切换线路，线路可能有几种，但是只有一种是激活状态的，我们就可以应用到这个方向上。



在这个例子里我们学习到如何添加资源，步骤也详细，后面的例程都可参考这个例程来添加资源文件，不再详细讲解添加过程。我们已经初步了解了 Qt 的样式表文件，如果要做好看的

界面 Qt 的样式表文件是少不了的。可能我们还不懂 Qt 样式表的语法，不知道如何下手。我们可以边学边了解，可以参考 Qt 帮助文档里的用法，qss 的功能不止这么点。现在的重点是学习 QRadioButton 这个控件。

7.1.4 QCheckBox

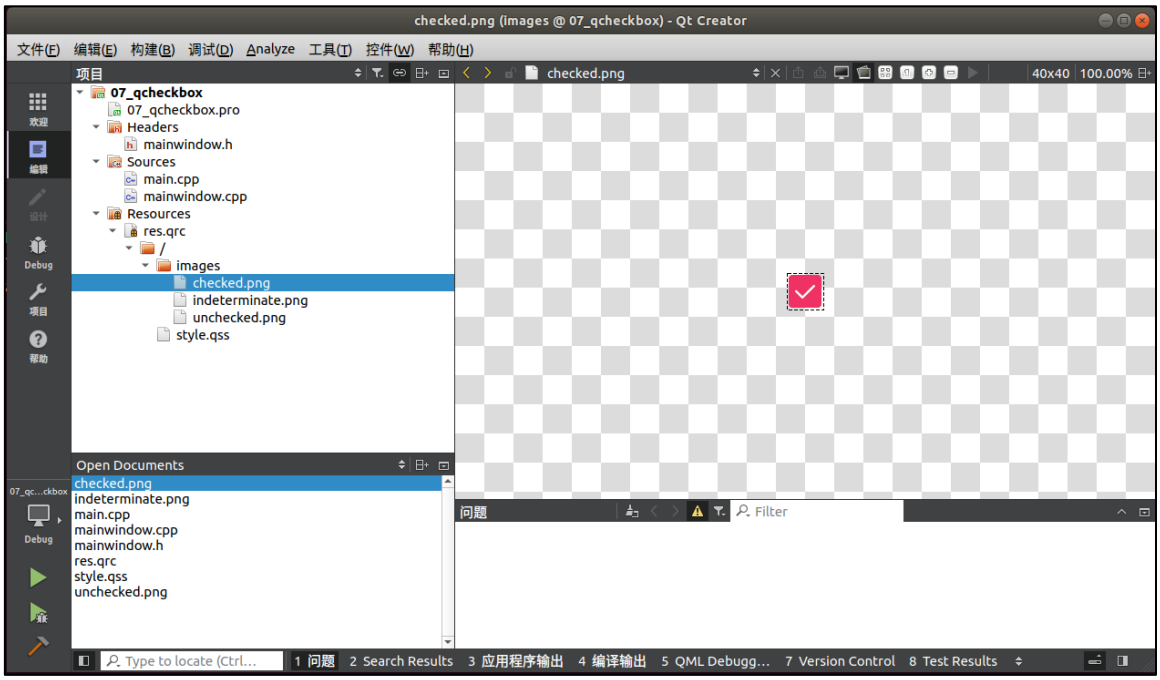
7.1.4.1 控件简介

QCheckBox 继承 QAbstractButton。复选按钮（复选框）与 RadioButton 的区别是选择模式，单选按钮提供多选一，复选按钮提供多选多。

7.1.4.2 用法示例

例 07_qcheckbox，三态选择框（难度：简单）。使用一个 QCheckBox，用户通过点击可改变当选择框的状态。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。笔者已经添加了 qss 文件和三张资源图片。如果还不会添加 qss 文件和资源图片，请参考 7.1.3 小节。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  /* 引入 QCheckBox  */
6  #include <QCheckBox>
```

```
7
8   class MainWindow : public QMainWindow
9   {
10      Q_OBJECT
11
12  public:
13      MainWindow(QWidget *parent = nullptr);
14      ~MainWindow();
15
16  private:
17      /* 声明一个 QCheckBox 对象 */
18      QCheckBox *checkBox;
19  private slots:
20      /* 声明 QCheckBox 的槽函数，并带参数传递，用这个参数接收信号的参数 */
21      void checkBoxStateChanged(int);
22
23  };
24  #endif // MAINWINDOW_H
```

在第 18 和 19 行声明两个 QCheckBox 对象。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 主窗体设置位置和显示的大小及背景颜色 */
7       this->setGeometry(0, 0, 800, 480);
8       this->setStyleSheet("QMainWindow {background-color: rgba(100, 100,
100, 100%);}");
9
10      /* 实例化对象 */
11      checkBox = new QCheckBox(this);
12
13      /* 设置 QCheckBox 位置和显示大小 */
14      checkBox->setGeometry(350, 200, 250, 50);
15
16      /* 初始化三态复选框的状态为 Checked */
17      checkBox->setCheckState(Qt::Checked);
18
19      /* 设置显示的文本 */
20      checkBox->setText("初始化为 Checked 状态");
```

```
21
22      /* 开启三态模式，必须开启，否则只有两种状态，即 Checked 和 Unchecked */
23      checkBox->setTristate();
24
25      /* 连接 checkBox 的信号 stateChanged(int)，与我们定义的槽
checkBoxStateChanged(int)连接 */
26      connect(checkBox, SIGNAL(stateChanged(int)), this,
                        SLOT(checkBoxStateChanged(int)));
27  }
28
29  MainWindow::~MainWindow()
30  {
31  }
32
33  /* 槽函数的实现 */
34  void MainWindow::checkBoxStateChanged(int state)
35  {
36      /* 判断 checkBox 的 state 状态，设置 checkBox 的文本 */
37      switch (state) {
38      case Qt::Checked:
39          /* 选中状态 */
40          checkBox->setText("Checked 状态");
41          break;
42      case Qt::Unchecked:
43          /* 未选中状态 */
44          checkBox->setText("Unchecked 状态");
45          break;
46      case Qt::PartiallyChecked:
47          /* 半选状态 */
48          checkBox->setText("PartiallyChecked 状态");
49          break;
50      default:
51          break;
52      }
53  }
```

第 23 行，需要注意的是设置 QCheckBox 对象 checkBox 需要设置为三态模式。

在源文件“main.cpp”具体代码如下。我们需要在 main.cpp 里加载 qss 文件。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4   /* 引入 QFile */
```

```
5   #include <QFile>
6
7   int main(int argc, char *argv[])
8   {
9       QApplication a(argc, argv);
10      /* 指定文件 */
11      QFile file(":/style.qss");
12
13      /* 判断文件是否存在 */
14      if (file.exists() ) {
15          /* 以只读的方式打开 */
16          file.open(QFile::ReadOnly);
17          /* 以字符串的方式保存读出的结果 */
18          QString styleSheet = QLatin1String(file.readAll());
19          /* 设置全局样式 */
20          qApp->setStyleSheet(styleSheet);
21          /* 关闭文件 */
22          file.close();
23      }
24      MainWindow w;
25      w.show();
26      return a.exec();
27  }
```

第 11 行至 23 行，读取 style.qss 的内容。并设置全局样式。

在源文件“style.qss”具体代码如下。

style.qss 编程后的代码

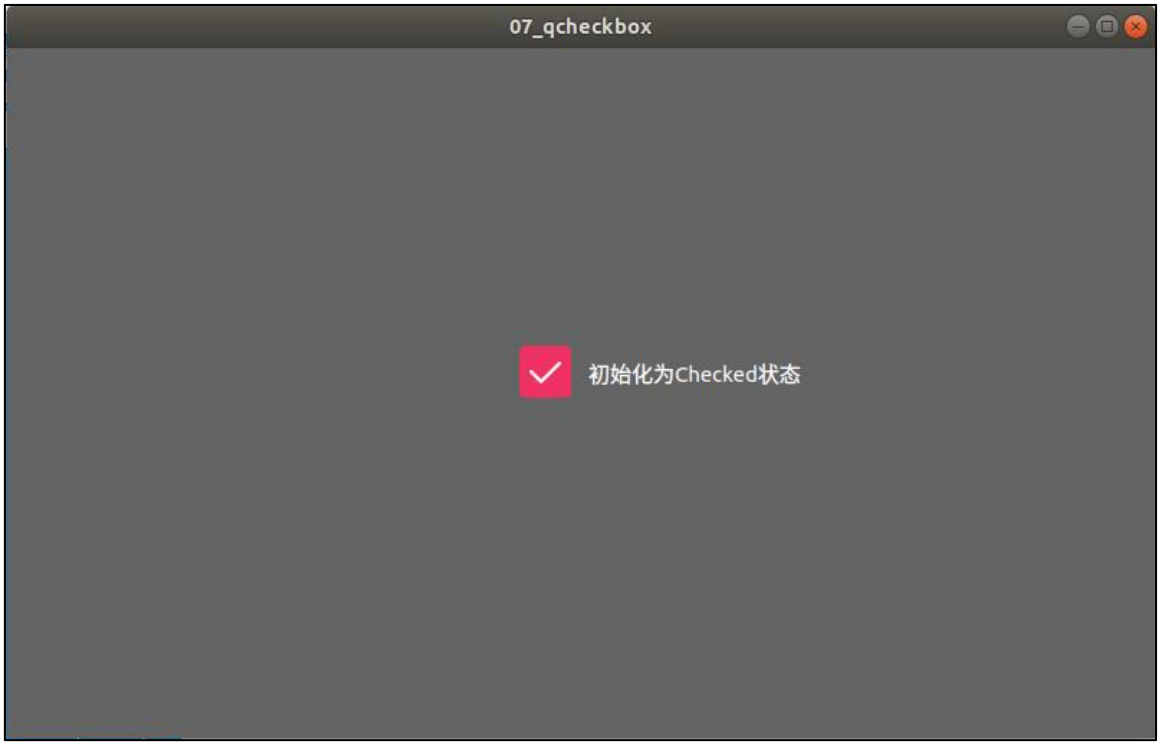
```
1   QCheckBox{
2           spacing: 5px;
3           color: white;
4   }
5   QCheckBox::indicator {
6           width: 50px;
7           height: 50px;
8   }
9   QCheckBox::indicator:enabled:unchecked {
10          image: url(:/images/unchecked.png);
11  }
12  QCheckBox::indicator:enabled:checked {
13          image: url(:/images/checked.png);
14  }
15  QCheckBox::indicator:enabled:indeterminate {
16          image: url(:/images/indeterminate.png);
17  }
```

在第 10 行和第 13 行，设置 QCheckBox 的 indicator 的背景图片。这样当它们点击切换时就会看到 QCheckBox 的三种选择状态了。

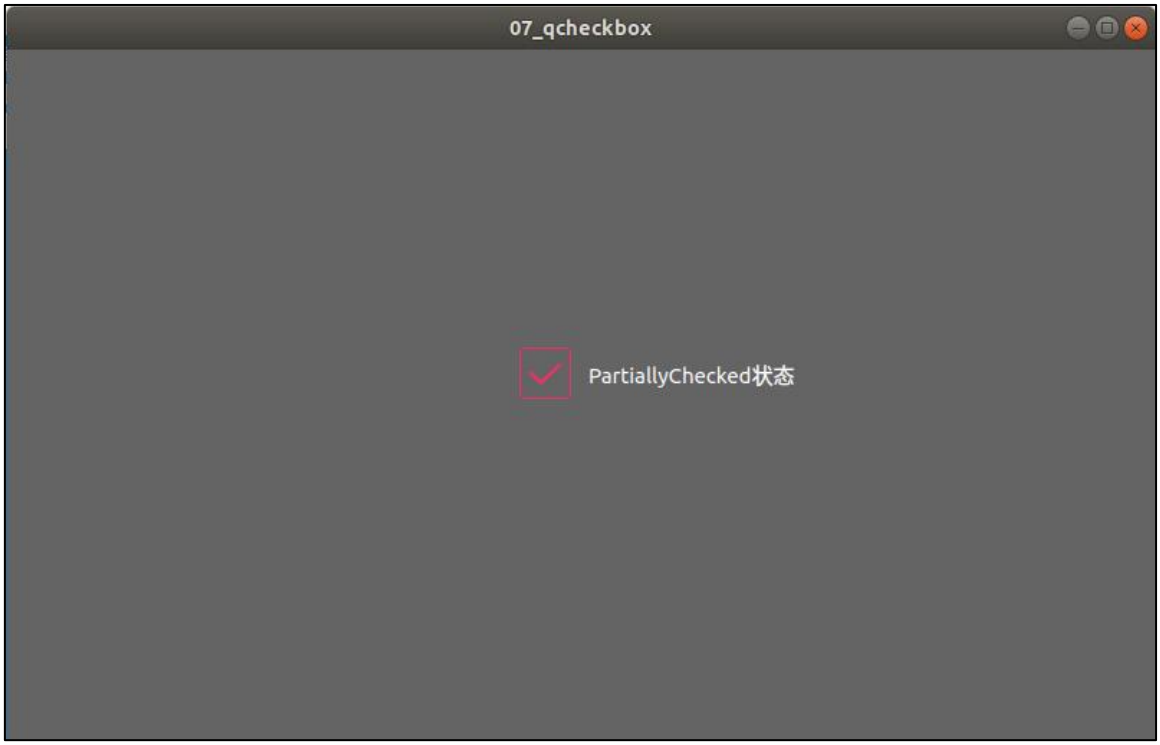
7.1.4.3 运行效果

编译程序运行的效果如下，多次点击 checkBox，即可看到 QCheckBox 的三种状态切换。

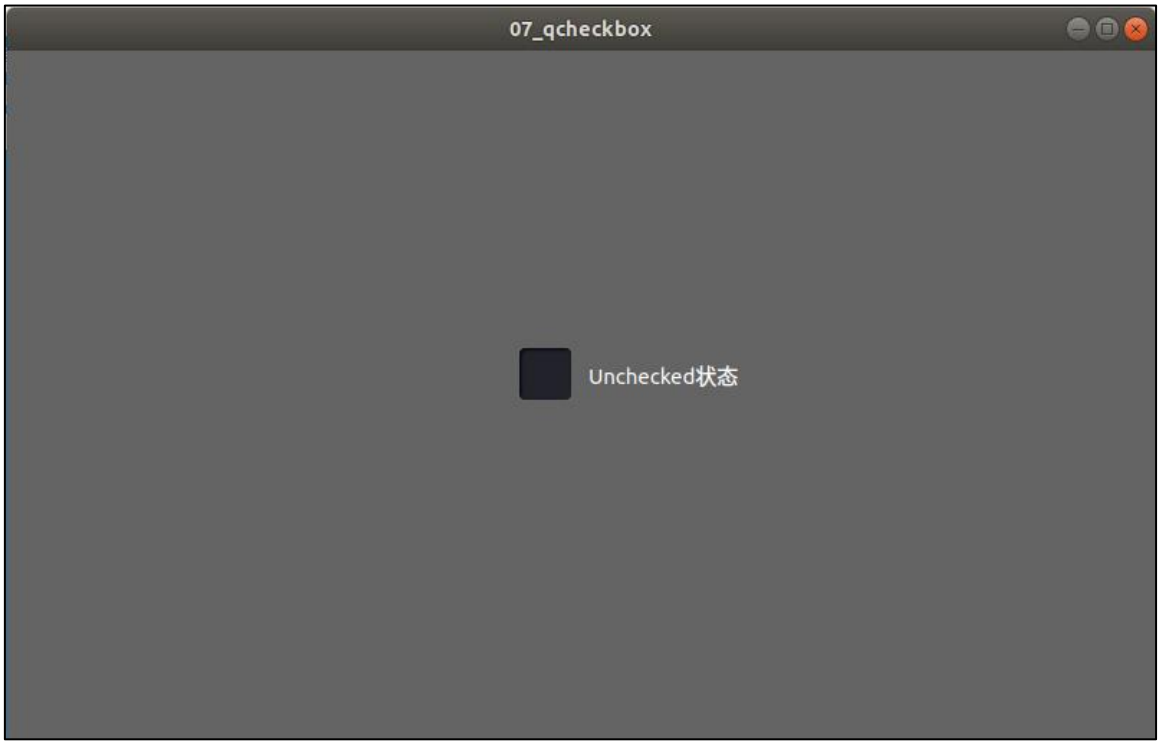
选中状态时。



半选状态。



未选中状态。



我们经常在软件安装时可以看到这种三态选择框，如果我们设计的程序有多种选择的项也可以设计这种选择框。

7.1.5 QCommandLinkButton

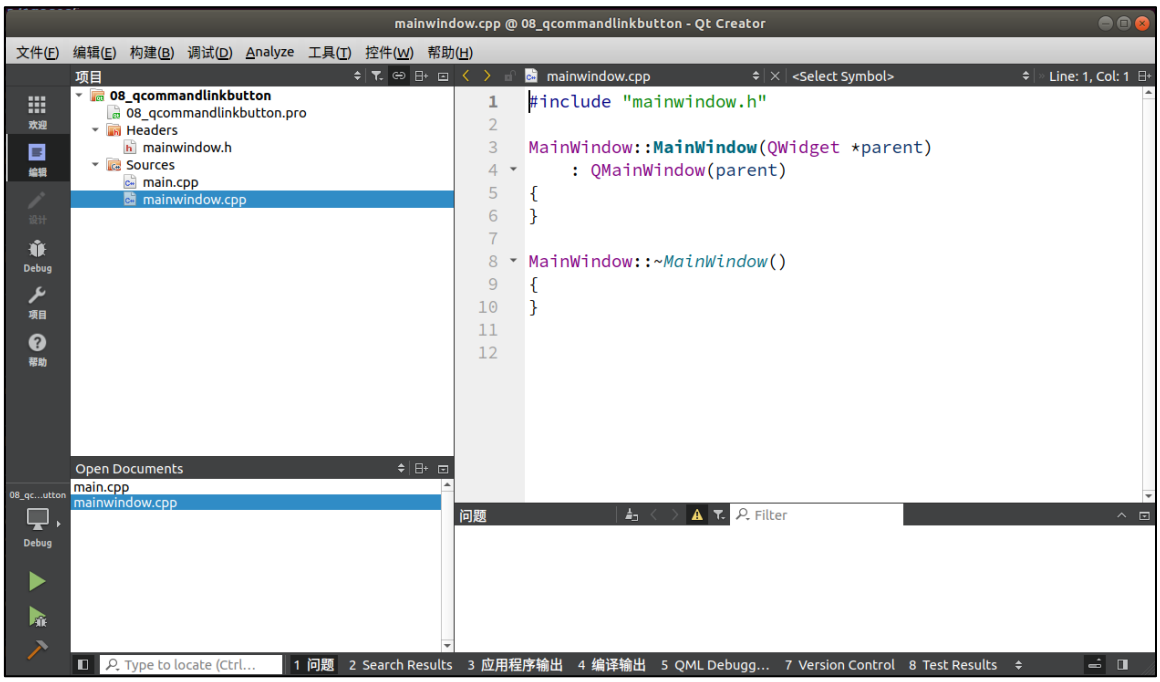
7.1.5.1 控件简介

QCommandLinkButton 控件中文名是“命令链接按钮”。QCommandLinkButton 继承 QPushButton。CommandLinkButton 控件和 RadioButton 相似，都是用于在互斥选项中选择一项。表面上同平面按钮一样，但是 CommandLinkButton 除带有正常的**按钮上的文字描述文本外，默认情况下，它也将携带一个箭头图标**，表明按下按钮将打开另一个窗口或页面。

7.1.5.2 用法示例

例 08_qcommandlinkbutton 链接窗口（难度：简单）。使用一个 QCommandLinkButton，点击打开系统的窗口。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   /* 引入 QCommandLinkButton */
6   #include <QCommandLinkButton>
7
8   class MainWindow : public QMainWindow
9   {
10      Q_OBJECT
11
12  public:
```

```
13      MainWindow(QWidget *parent = nullptr);
14      ~MainWindow();
15
16  private:
17      /* 声明一个 QCommandLinkButton 对象 */
18      QCommandLinkButton *commandLinkButton;
19
20  private slots:
21      /* 声明槽函数，用于点击 commandLinkButton 后触发 */
22      void commandLinkButtonClicked();
23
24
25  };
26  #endif // MAINWINDOW_H
```

在第 18 行，声明一个 QCommandLinkButton 对象。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2   /* 引入桌面服务，用来打开系统文件夹对话框 */
3   #include <QDesktopServices>
4   /* 引入 QUrl */
5   #include <QUrl>
6
7   MainWindow::MainWindow(QWidget *parent)
8       : QMainWindow(parent)
9   {
10      /* 主窗体设置位置和显示的大小 */
11      this->setGeometry(0, 0, 800, 480);
12
13      /* 实例化对象 */
14      commandLinkButton = new QCommandLinkButton(
15                  "打开/home 目录","点击此将调用系统的窗口打开/home 目录",this);
16
17      /* 设置 QCommandLinkButton 位置和显示大小 */
18      commandLinkButton->setGeometry(300, 200, 250, 60);
19
20      /* 信号槽连接 */
21      connect(commandLinkButton, SIGNAL(clicked()), this,
22              SLOT(commandLinkButtonClicked()));
23  }
24
25  MainWindow::~MainWindow()
26  {
27  }
28
29  void MainWindow::commandLinkButtonClicked()
```

```
30  {
31      /* 调用系统服务打开/home 目录 */
32      QDesktopServices::openUrl(QUrl("file:///home/") );
33  }
```

第 14 行，实例化时原型是 QCommandLinkButton::QCommandLinkButton(const QString &text, const QString &description, QWidget *parent = nullptr)。

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

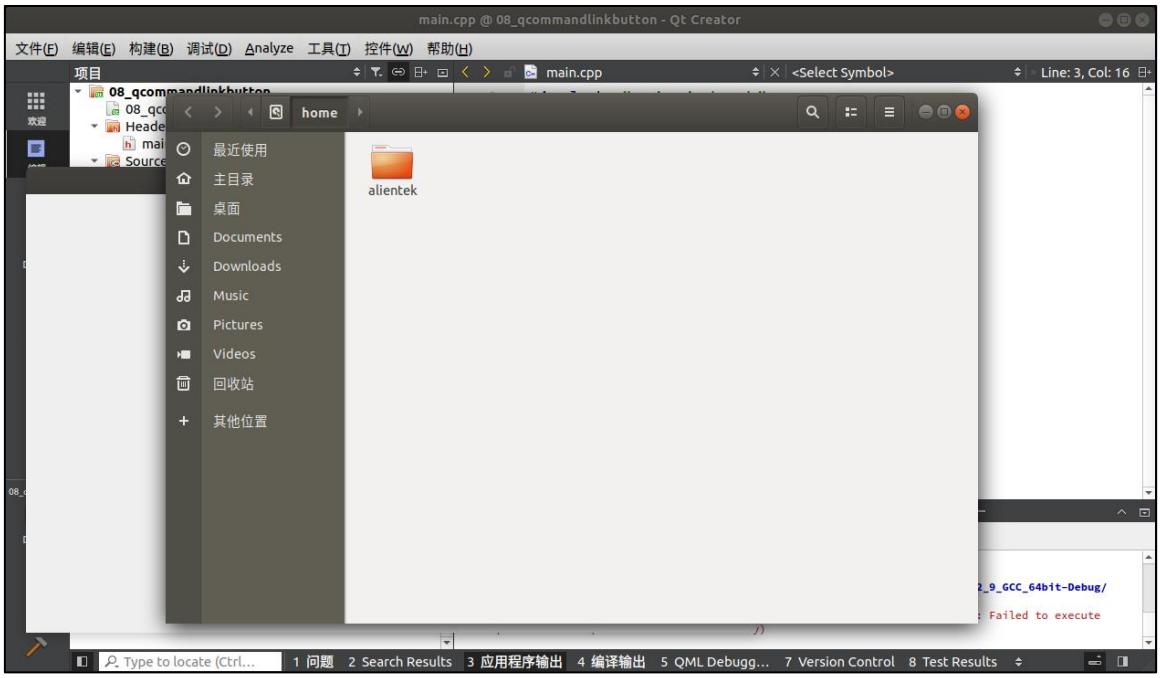
```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.1.5.3 运行效果

程序编译运行的结果如下。

点击中间的打开/home 目录按钮，结果如下。系统弹出一个窗口，直接打开到/home 目录。

点击打开/home 目录后，系统将弹出/home 目录路径窗口。



7.1.6 QDialogButtonBox

7.1.6.1 控件简介

对话框和消息框通常以符合该平台界面指导原则的布局呈现按钮。不同平台的对话框总是有不同的布局。QDialogButtonBox 允许开发人员向其添加按钮，并将自动使用适合用户桌面环境的布局。也就是说我们可以使用系统的自带的对话框按钮，也可以自己定义对话框按钮。

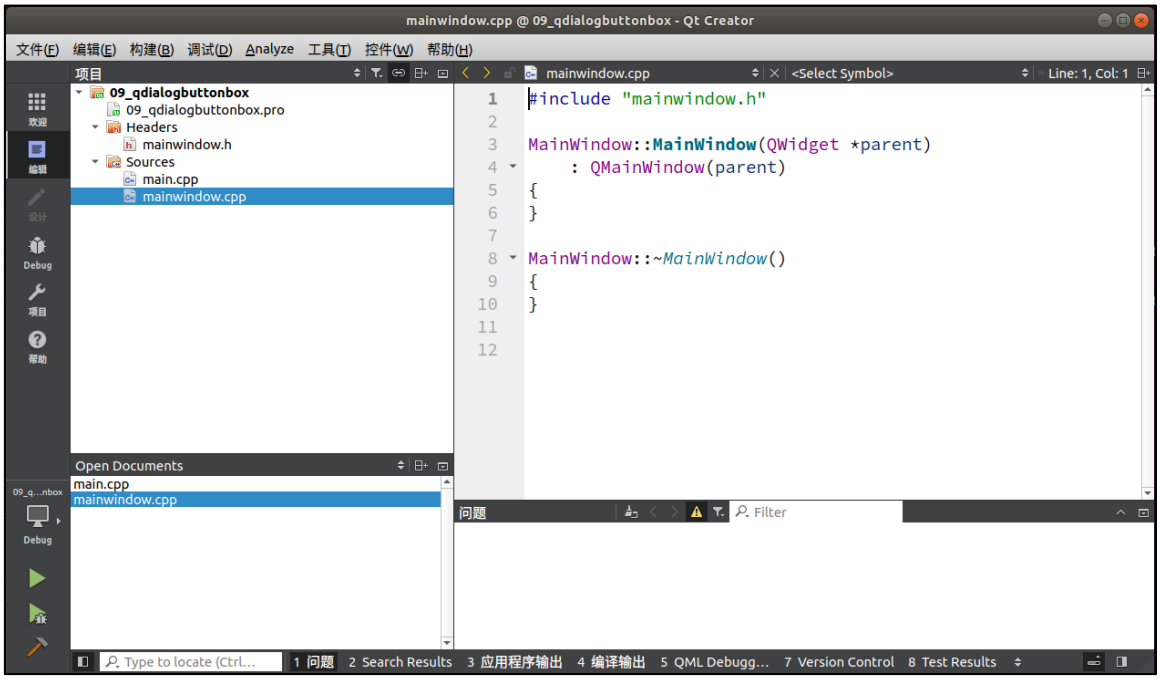
QDialogButtonBox 常用的按钮有如下几种，更多的可以参考 Qt 帮助文档。

```
button_Box = new QDialogButtonBox(QDialogButtonBox::Ok
                                  | QDialogButtonBox::Cancel
                                  | QDialogButtonBox::Open
                                  | QDialogButtonBox::Save
                                  | QDialogButtonBox::Close
                                  | QDialogButtonBox::Discard
                                  | QDialogButtonBox::Apply
                                  | QDialogButtonBox::Reset
                                  | QDialogButtonBox::RestoreDefaults
                                  | QDialogButtonBox::Help
                                  | QDialogButtonBox::SaveAll);
```

7.1.6.2 用法示例

例 09_qdialogbuttonbox， 自定义 QDialogButtonBox 里的按钮（难度：简单）。使用一个 QDialogButtonBox，在 QDialogButtonBox 添加 Qt 提供的按钮，或者自定义按钮。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  /* 引入 QDialogButtonBox */
6  #include <QDialogButtonBox>
7  /* 引入 QPuhsButton */
8  #include <QPushButton>
9
10 class MainWindow : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     MainWindow(QWidget *parent = nullptr);
16     ~MainWindow();
17
18 private:
```

```
19      /* 声明一个 QDialogButtonBox 对象 */
20      QDialogButtonBox *dialogButtonBox;
21
22      /* 声明一个 QPushButton 对象 */
23      QPushButton *pushButton;
24
25  private slots:
26      /* 声明信号槽，带 QAbstractButton *参数，用于判断点击了哪个按钮 */
27      void dialogButtonBoxClicked(QAbstractButton *);
28
29  };
30  #endif // MAINWINDOW_H
```

第 18 行，声明一个 QDialogButtonBox 对象。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2   /* 引入 QDebug */
3   #include <QDebug>
4
5   MainWindow::MainWindow(QWidget *parent)
6       : QMainWindow(parent)
7   {
8       /* 主窗体设置位置和显示的大小 */
9       this->setGeometry(0, 0, 800, 480);
10
11      /* 实例化并设置按钮的盒子的大小和位置 */
12      dialogButtonBox = new  QDialogButtonBox(this);
13      dialogButtonBox->setGeometry(300, 200, 200, 30);
14
15      /*使用 Qt 的 Cancel 按钮*/
16      dialogButtonBox->addButton(QDialogButtonBox::Cancel);
17
18      /*将英文"Cancel"按钮设置为中文"取消" */
19      dialogButtonBox->button(QDialogButtonBox::Cancel)->setText("取消
");
20
21      /* 设定位置与大小 */
22      pushButton = new QPushButton(tr("自定义"));
23
24      /* 将 pushButton 添加到 dialogButtonBox，并设定 ButtonRole 为 ActionRole
*/
```

```
25      dialogButtonBox->addButton(pushButton,
                                   QDialogButtonBox::ActionRole);
26
27      /* 信号槽连接，带参数 QAbstractButton *，用于判断用户点击哪个按键 */
28      connect(dialogButtonBox, SIGNAL(clicked(QAbstractButton * )),
29              this, SLOT(dialogButtonBoxClicked(QAbstractButton *)));
30  }
31
32  MainWindow::~MainWindow()
33  {
34  }
35
36  void MainWindow::dialogButtonBoxClicked(QAbstractButton *button)
37  {
38      /* 判断点击的对象是否为 QDialogButtonBox::Cancel */
39      if(button == dialogButtonBox->button(QDialogButtonBox::Cancel)) {
40          /* 打印"单击了取消键" */
41          qDebug() <<"单击了取消键"<<endl;
42          /* 判断点击的对象是否为 pushButton */
43      }else if(button == pushButton) {
44          /* 打印"单击了自定义键" */
45          qDebug() <<"单击了自定义键"<<endl;
46      }
47  }
```

第 16 行，实例化时原型是 void QDialogButtonBox::addButton(QAbstractButton *button, QDialogButtonBox::ButtonRole role)。

第 41 和 45 行，我们第一次用 qDebug()。Qt 一般调试都是用 qDebug()来打印的。这与 C++的 cout 是功能基本一样。只是 Qt 自定义为 qDebug()而已。

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

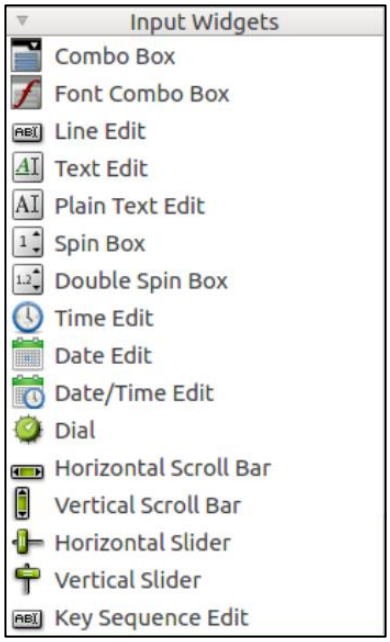
```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.1.6.3 运行效果

程序编译运行的结果如下。点击自定义按钮和取消按钮，在应用程序输出窗口可以看到对应的点击事件。

7.2 输入窗口部件

Qt Designer 窗口部件提供的面板中，提供了 16 种输入部件如下



（1） Comb Box:组合框

（2） Font Comb Box:字体组合框

（3） Line Edit:单行编辑框

（4） Text Edit:文本编辑框

（5） Plain Text Edit:纯文本编辑框

（6） Spin Box:数字旋转框

（7） Double Spin Box:双精度数字旋转框

（8） Time Edit:时间编辑框

（9） Date Edit:日期编辑框

（10） Date/Time Edit:日期时间编辑框

（11） Dial:数字拨盘框

（12） Horizontal Scroll Bar:水平滚动条

（13） Vertical Scroll Bar:垂直滚动条

（14） Horizontal Slider:水平滑动条

（15） Vertical Slider:垂直滑动条

（16） Key sequence Edit:按键序列编辑框

这十六种按钮部件作用简介如下:

QComboBox 继承 QWidget 类，被 QFontComboBox 类继承。通常用于用户显示选项列表的方法，这种方法占用最少的屏幕空间。

QFontComboBox 继承 QComboBox。QFontComboBox 小部件是一个允许用户选择字体系列的组合框。组合框中填充了按字母顺序排列的字体家族名称列表。FontComboBox 常用于工具栏，与 ComboBox 一起用于控制字体大小，并与两个 ToolButtons 一起用于粗体和斜体。

QLineEdit 继承 QWidget。QLineEdit 小部件是一个单行文本编辑器。行编辑允许用户使用一组有用的编辑函数输入和编辑一行纯文本，包括撤消和重做、剪切和粘贴以及拖放。通过更改行编辑的 echoMode()，它还可以用作“只写”字段，用于输入如密码等。

QTextEdit 继承 QAbstractScrollArea，被 QTextBrowser 继承。QTextEdit 是一个高级所见即所得查看器/编辑器，支持使用 html 样式的标记进行 rich text 格式化。它经过优化以处理大型文档并快速响应用户输入。QTextEdit 用于段落和字符。段落是格式化的字符串，它被字包装以适应小部件的宽度。在阅读纯文本时，默认情况下，一个换行表示一个段落。一份文件由零个或多个段落组成。段落中的文字与段落的对齐方式一致。段落之间用硬换行符隔开。段落中的每个字符都有自己的属性，例如字体和颜色。QTextEdit 可以显示图像，列表和表格。如果文本太大而无法在文本编辑的视图中查看，视图中则会出现滚动条。

QPlainTextEdit 是一个支持纯文本的高级查看器/编辑器。它被优化为处理大型文档和快速响应用户输入。

QSpinBox 继承 QAbstractSpinBox。用于处理整数和离散值（例如：月份名称）而 QDoubleSpinBox 则用于处理浮点值。他们之间的区别就是处理数据的类型不同，其他功能都基本相同。QSpinBox 允许用户通过单击上/下按钮或按下键盘上的上/下按钮来选择一个值，以增加/减少当前显示的值。用户还可以手动输入值。

QDoubleSpinBox 继承 QAbstractSpinBox。QDoubleSpinBox 则用于处理浮点值。QDoubleSpinBox 允许用户通过单击“向上”和“向下”按钮或按下键盘上的“向上”或“向下”按钮来选择当前显示的值。用户还可以手动输入值。

QTimeEdit 继承 QDateTimeEdit。QTimeEdit 用于编辑时间，而 QDateEdit 用于编辑日期。

QDateEdit 继承 QDateTimeEdit。QDateEdit 用于编辑日期，而 QTimeEdit 用于编辑时间。

QDateTimeEdit 类提供了一个用于编辑日期和时间的小部件。QDateTimeEdit 允许用户使用键盘或箭头键编辑日期，以增加或减少日期和时间值。箭头键可用于在 QDateTimeEdit 框中从一个区域移动到另一个区域。

QDial 类提供了一个圆形范围控制(如速度计或电位器)。QDial 用于当用户需要在可编程定义的范围内控制一个值，并且该范围要么是环绕的(例如，从 0 到 359 度测量的角度)，要么对话框布局需要一个正方形小部件。由于 QDial 从 QAbstractSlider 继承，因此拨号的行为与滑块类似。 当 wrapping（）为 false（默认设置）时，滑块和刻度盘之间没有真正的区别。 它们共享相同的信号，插槽和成员功能。 您使用哪一个取决于您的用户期望和应用程序类型。

QScrollBar 继承 QAbstractSlider。QScrollBar 小部件提供垂直或水平滚动条，允许用户访问比用于显示文档的小部件大的文档部分。它提供了用户在文档中的当前位置和可见文档数量的可视化指示。滚动条通常配有其他控件，可以实现更精确的导航。

QSlider 继承 QAbstractSlider。QSlider 类提供垂直或水平滑动条小部件，滑动条是用于控制有界值的典型小部件。它允许用户沿着水平或垂直凹槽移动滑块手柄，并将手柄的位置转换为合法范围内的整数值。

QKeySequenceEdit 继承 QWidget。这个小部件允许用户选择 QKeySequence, QKeySequence 通常用作快捷方式。当小部件接收到焦点并在用户释放最后一个键后一秒结束时，将启动记录，通常用作记录快捷键。

7.2.1 QComboBox

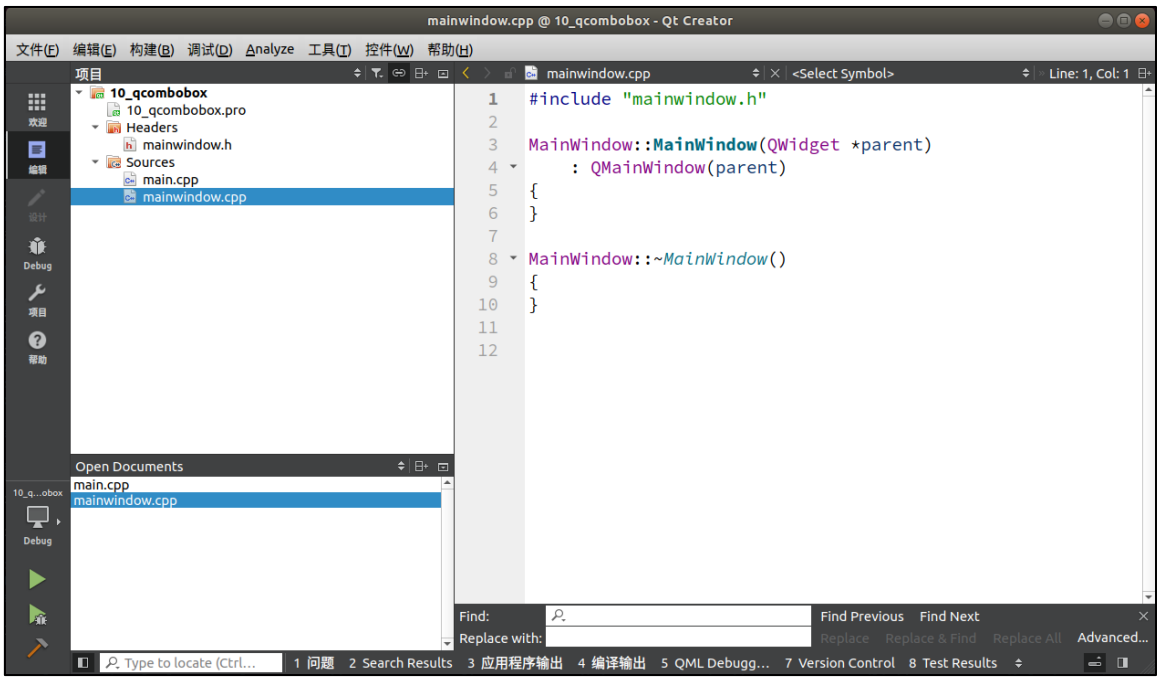
7.2.1.1 控件简介

QComboBox 类提供了 Qt 下拉组合框的组件。

7.2.1.2 用法示例

例 10_qcombobox，选择省份（难度：简单），通过点击下拉按钮的项，选择其中一项，然后打印出当前选择项的内容。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  /* 引入 QComboBox */
6  #include <QComboBox>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15
16 private:
17     /* 声明一个 QComboBox 对象 */
18     QComboBox *comboBox;
19
20 private slots:
21     /* 声明 QComboBox 对象的槽函数 */
22      void comboBoxIndexChanged(int);
23
24 };
25 #endif // MAINWINDOW_H
```

第 20 行，声明一个 QComboBox 对象。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1  #include "mainwindow.h"
2  /* 引入 QDebug */
3  #include <QDebug>
```

```
4
5   MainWindow::MainWindow(QWidget *parent)
6       : QMainWindow(parent)
7   {
8       /* 设置主窗体的显示位置与大小 */
9       this->setGeometry(0, 0, 800, 480);
10
11      /* 实例化对象 */
12      comboBox = new QComboBox(this);
13
14      /* 设置 comboBox 的显示位置与大小 */
15      comboBox->setGeometry(300, 200, 150, 30);
16
17      /* 添加项，我们添加三个省份，作为 comboBox 的三个选项 */
18      comboBox->addItem("广东(默认)");
19      comboBox->addItem("湖南");
20      comboBox->addItem("四川");
21
22      /* 信号槽连接 */
23      connect(comboBox, SIGNAL(currentIndexChanged(int)), this,
24              SLOT(comboBoxIndexChanged(int)));
25  }
26
27  MainWindow::~MainWindow()
28  {
29  }
30
31  void MainWindow::comboBoxIndexChanged(int index)
32  {
33      /* 打印出选择的省份 */
34      qDebug()<<"您选择的省份是"<< comboBox->itemText(index)<<endl;
35  }
```

第 18 至 20 行，添加 Item，也就是项。

第 30 至 34 行，当点击下拉列表改变选择的省份就会触发 currentIndexChanged(int)这个信号，就会相应打印项的省份名称。

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
```

```
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.2.1.3 运行效果

程序编译运行的结果如下，当点击下拉选择框选择省份时，槽函数将打印出您选择的省份。

点击选择“湖南”，则打印出“您选择的省份是湖南”。

QComboBox 我们常会在一些需要下拉列表选择的项目中用到。比如 QQ 登录如果有多个帐号选择就需要这个 QComboBox。

7.2.2 QFontComboBox

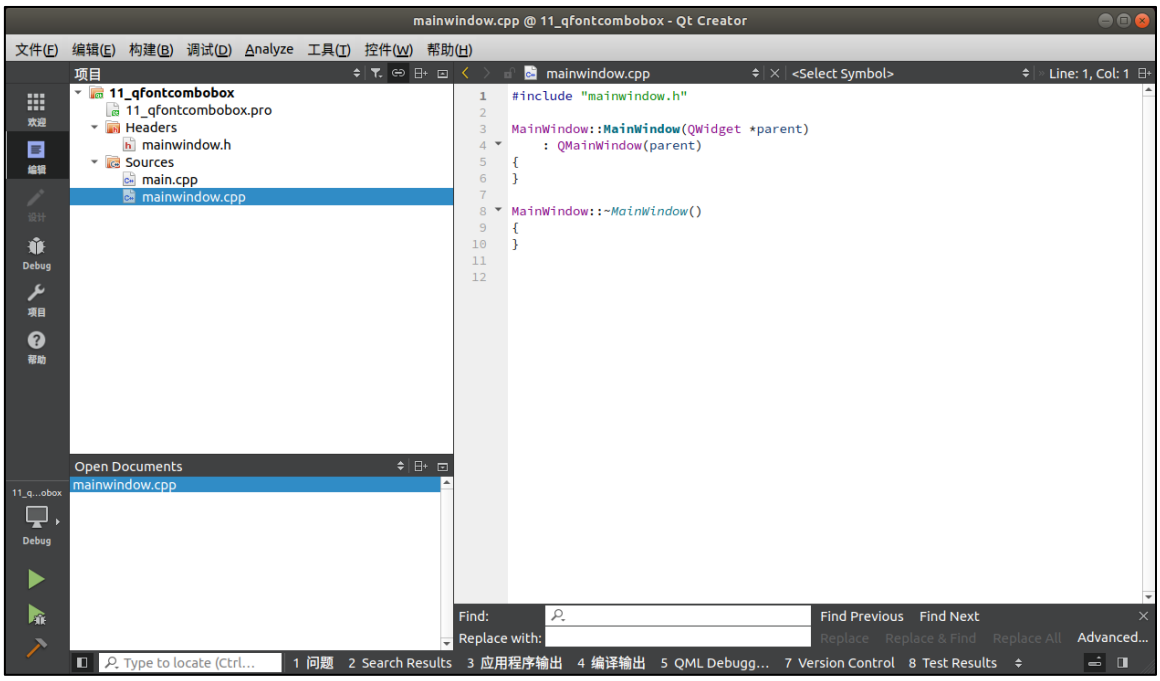
7.2.2.1 控件简介

QFontComboBox 类提供了下拉选择字体系列的组合框小部件。

7.2.2.2 用法示例

例 11_qfontcombobox，字体选择（难度：简单），通过点击下拉按钮的项，选择其中一项，然后打印出当前选择项的内容。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  /* 引入 QFontComboBox */
6  #include <QFontComboBox>
7  /* 引入 QLable */
8  #include <QLabel>
9
10 class MainWindow : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     MainWindow(QWidget *parent = nullptr);
16     ~MainWindow();
17
18 private:
19     /* 声明一个 QFontComboBox 对象 */
20     QFontComboBox *fontComboBox;
21     /* 声明一个 Label 对象，用于显示当前字体变化 */
22     QLabel *label;
23
24 private slots:
```

```
25      /* 声明 QFontComboBox 对象使用的槽函数 */
26      void fontComboBoxFontChanged(QFont);
27
28  };
29  #endif // MAINWINDOW_H
```

第 20 行，声明一个 QFontComboBox 对象。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗体的显示位置和大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 实例化对象 */
10      fontComboBox = new QFontComboBox(this);
11      label = new QLabel(this);
12
13      /* 设置显示的位置与大小 */
14      fontComboBox->setGeometry(280, 200, 200, 30);
15      label->setGeometry(280, 250, 300, 50);
16
17      /* 信号与槽连接 */
18      connect(fontComboBox, SIGNAL(currentFontChanged(QFont)), this,
19              SLOT(fontComboBoxFontChanged(QFont)));
20  }
21
22  MainWindow::~MainWindow()
23  {
24  }
25
26  /* 槽函数实现 */
27  void MainWindow::fontComboBoxFontChanged(QFont font)
28  {
29      /* 将 label 里的文本内容设置为所选择的字体 */
30      label->setFont(font);
31
32      /* 定义一个字符串接收当前项的字体 */
33      QString str = "用此标签显示字体效果\n 设置的字体为: " +
34              fontComboBox->itemText(fontComboBox->currentIndex());
```

```
35
36      /* 将字符串的内容作为 label 的显示内容 */
37      label->setText(str);
38  }
```

第 27 至 37 行，当选择的字体改变时，槽函数就会设置 label 的字体，并打印当前字体的名称。

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.2.2.3 运行效果

程序编译运行的结果如下，当点击 FontCOmboBox 字体组合框选择字体后，Label 标签显示的字体将改变为当前所选择的字体。（注意 Ubuntu 与 Windows 的字体不一样，所以显示的效果有可能不一样，下图为 Ubuntu 的字体显示效果）

在手机，电脑一些软件都有设置字体的功能，由用户自行选择，所以我们这个 QFontComboBox 就可以应用于此种场合。当然也有设置字体的大小，颜色等，这些由我们自由设计。

7.2.3 QLineEdit

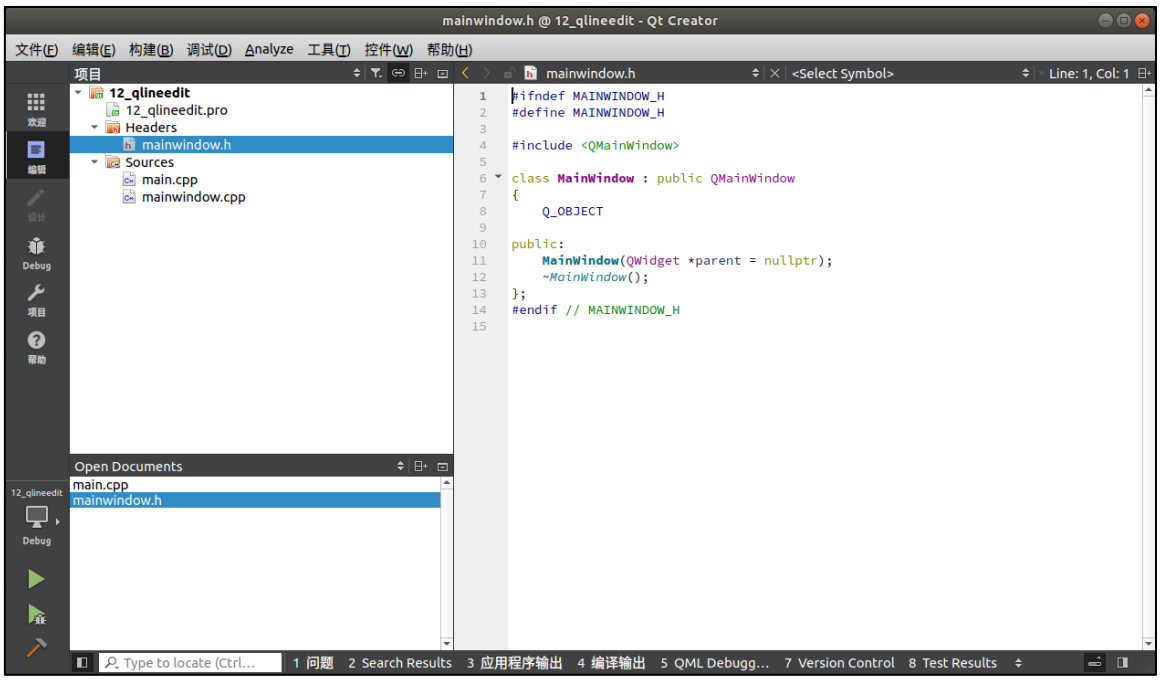
7.2.3.1 控件简介

QLineEdit 小部件是一个单行文本编辑器。行编辑允许用户使用一组有用的编辑函数输入和编辑一行纯文本。包括撤消和重做、剪切和粘贴以及拖放。通过更改行编辑的 echoMode()，它还可以用作“只写”字段，用于输入如密码等。

7.2.3.2 用法示例

例 12_qlineedit，单行输入框（难度：简单），通过点击下拉按钮的项，选择其中一项，然后打印出当前选择项的内容。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。像引入头文件，设置主窗体大小位置和实例化对象这种注释我们慢慢淡化，不再写详细注释了。读者看了前面的对这种设置已经很清楚了。

mainwindow.h 编程后的代码

```
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QLineEdit>
6   #include <QPushButton>
7   #include <QLabel>
8
9   class MainWindow : public QMainWindow
10  {
11      Q_OBJECT
12
13  public:
14      MainWindow(QWidget *parent = nullptr);
15      ~MainWindow();
16  private:
17      /* 声明一个 QLineEdit 对象 */
18      QLineEdit *lineEdit;
19
20      /* 声明一个 QPushButton 对象 */
21      QPushButton *pushButton;
22
23      /* 声明一个 QLabel 对象 */
```

```
24      QLabel *label;
25
26  private slots:
27      /* 声明一个槽函数，响应 pushButton 的 clicked 事件 */
28      void pushButtonClicked();
29  };
30  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       this->setGeometry(0, 0, 800, 480);
7
8       lineEdit = new QLineEdit(this);
9       lineEdit->setGeometry(280, 200, 200, 20);
10
11      pushButton = new QPushButton(this);
12      pushButton->setGeometry(500, 200, 50, 20);
13      pushButton->setText("确认");
14
15      label = new QLabel(this);
16      label->setGeometry(280, 250, 400, 20);
17      label->setText("您输入的内容是：");
18
19      /* 信号槽连接 */
20      connect(pushButton,SIGNAL(clicked()), this,
21              SLOT(pushButtonClicked()));
22  }
23
24  MainWindow::~MainWindow()
25  {
26  }
27
28  void MainWindow::pushButtonClicked()
29  {
30      /* 字符串变量 str */
31      QString str;
32
33      str = "您输入的内容是：";
34      str += lineEdit->text();
```

```
35
36      /* 设置 label 里的文本显示内容 */
37      label->setText(str);
38      /* 在点击了确认键之后清空 lineEdit 单行输入框 */
39      lineEdit->clear();
40  }
```

第 28 至 40 行，当我们在单选输入框里输入完成后，将输入的内容设置为在 label 的文本内容。

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.2.3.3 运行效果

程序编译运行的结果如下，当在 QLineEdit 单行输入框内输入文本内容后，单击 QPushButton 确认按钮后，QLabel 的文本内容将显示您所输入的内容。然后 QLineEdit 将清空，可再次输入。

QLineEdit 的简单使用如上，笔者也是简单的介绍了它的用法。要想写好一点的例子，需要我们主动思考，比如，做个将这个 QLineEdit 应用到密码登录窗口上，输入密码，然后判断这个密码是否与预设的密码一样才解锁等。

7.2.4 QTextEdit

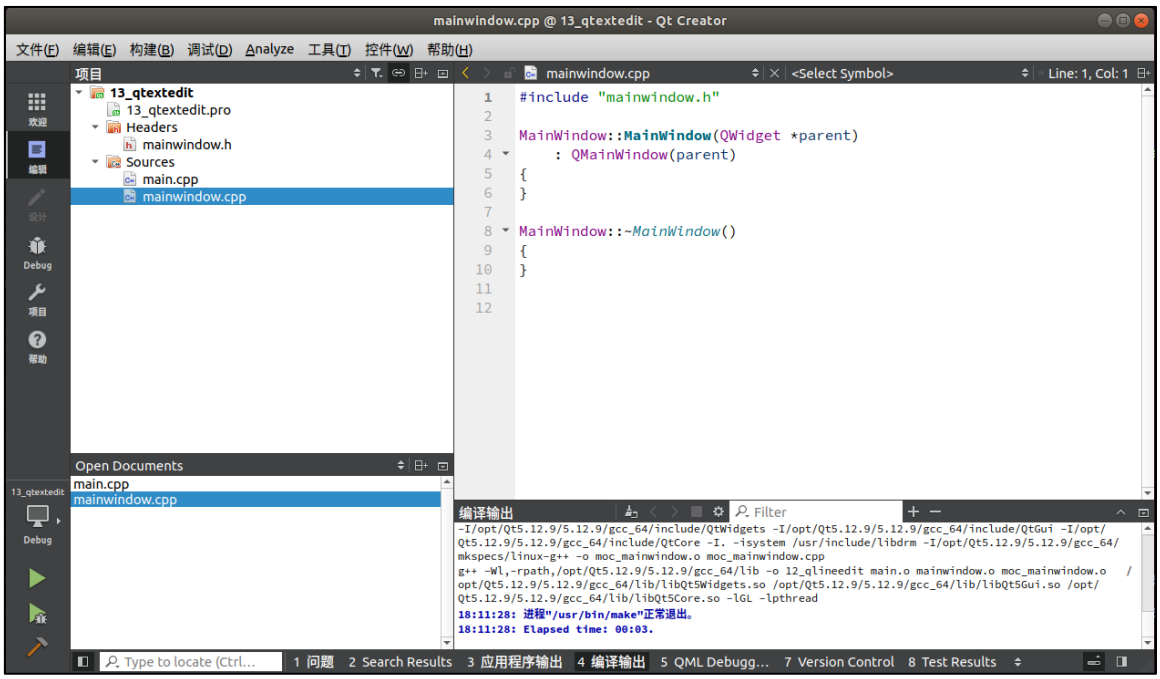
7.2.4.1 控件简介

QTextEdit 类提供了一个查看器/编辑器小部件。

7.2.4.2 用法示例

例 13_qtextedit 文本编辑框（难度：简单），用一个 QTextEdit 来演示文本的输入，用两个 QPushButton 来模拟文本编辑的全选与清除。在 QTextEdit 里也可用键盘的快捷键（如 Ctrl+A）来完成全选，复制，粘贴等操作。Qt 提供了全选，复制粘贴等这一类的函数方便用户操作，下面用简单的实例来演示。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3  #include <QTextEdit>
4  #include <QPushButton>
5
6  #include <QMainWindow>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15
16 private:
17     /* 声明一个 QTextEdit 对象 */
18     QTextEdit *textEdit;
19
20     /* 声明两个 QPushButton 对象 */
21     QPushButton *pushButtonSelectAll;
22     QPushButton *pushButtonClearAll;
23
24 private slots:
25     /* 声明两个槽函数，响应按钮点击响应的事件 */
```

```
26      void pushButtonSelectAllClicked();
27      void pushButtonClearAllClicked();
28
29  };
30  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗体显示的位置和大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 实例和对象，设置位置和显示大小 */
10      textEdit = new QTextEdit(this);
11      textEdit->setGeometry(0, 0, 800, 400);
12
13      /* 实例和对象，设置位置和显示大小，设置文本 */
14      pushButtonSelectAll = new QPushButton(this);
15      pushButtonSelectAll->setGeometry(200, 420, 50, 20);
16      pushButtonSelectAll->setText("全选");
17
18      /* 实例和对象，设置位置和显示大小，设置文本 */
19      pushButtonClearAll = new QPushButton(this);
20      pushButtonClearAll->setGeometry(500, 420, 50, 20);
21      pushButtonClearAll->setText("清除");
22
23      /* 信号槽连接 */
24      connect(pushButtonSelectAll, SIGNAL(clicked()), this,
25              SLOT(pushButtonSelectAllClicked()));
26      connect(pushButtonClearAll, SIGNAL(clicked()), this,
27              SLOT(pushButtonClearAllClicked()));
28
29  }
30
31  MainWindow::~MainWindow()
32  {
33  }
34
35  void MainWindow::pushButtonSelectAllClicked()
36  {
```

```
37      /* 设置焦点为 textEdit */
38      textEdit->setFocus();
39      /* 判断文本编辑框内容是否为空，不为空则全选 */
40      if(!textEdit->toPlainText().isEmpty()){
41          /* 全选 */
42          textEdit->selectAll();
43      }
44  }
45
46  void MainWindow::pushButtonClearAllClicked()
47  {
48      /* 清空 textEdit 里的文本内容 */
49      textEdit->clear();
50  }
51
```

第 35 至 49 行，当我们在文本输入框里输入完成后，当点击全选按钮后，需要设置焦点到 textEdit 上，否则将不能设置全选。

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.2.4.3 运行效果

程序编译运行的结果如下，在编辑框里输入文字后，点击按钮全选，点击清除则清除编辑框内的全部内容。如下图为点击全选的效果。

7.2.5 QPlainTextEdit

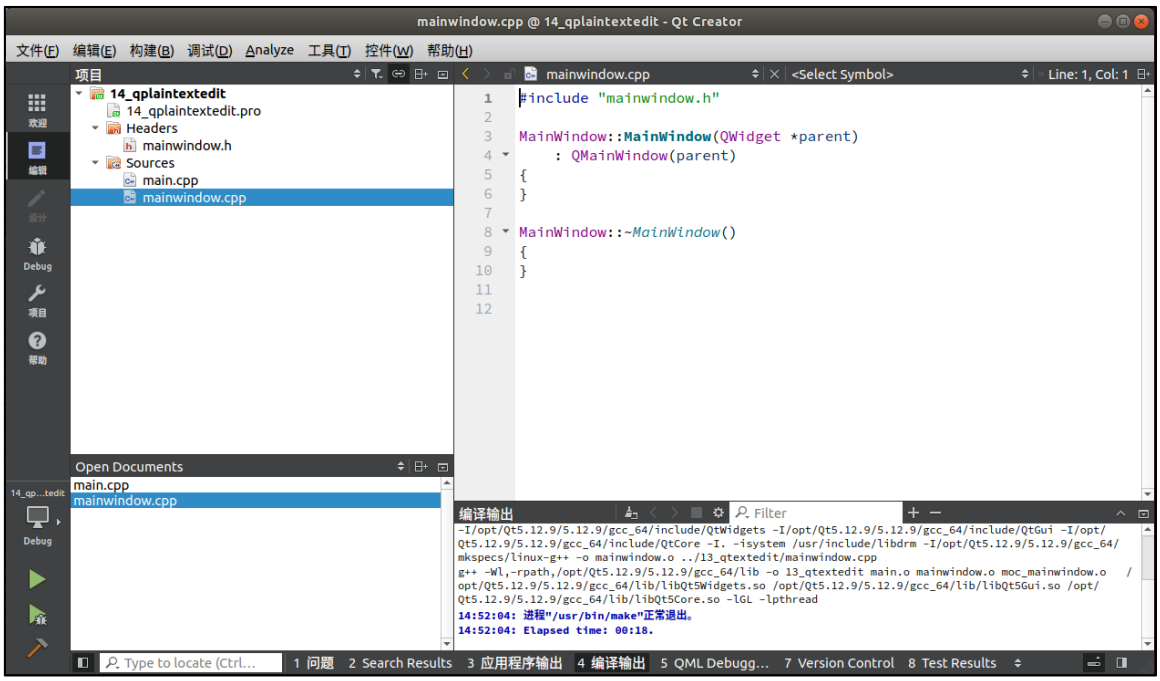
7.2.5.1 控件简介

QPlainTextEdit 类提供了一个用于编辑和显示纯文本的小部件，常用于显示多行文本或简单文本。

7.2.5.2 用法示例

例 14_qplaintextedit 文本浏览编辑器（难度：简单），用一个 QPlainTextEdit 来读取本当前工程里的一个文件，并用一个 RadioButton 里将 QPlainTextEdit 设为只读。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QPlainTextEdit>
5  #include <QRadioButton>
6
7  #include <QMainWindow>
8
9  class MainWindow : public QMainWindow
10 {
11     Q_OBJECT
12
13 public:
14     MainWindow(QWidget *parent = nullptr);
15     ~MainWindow();
16
17 private:
18     /* 声明对象 */
19     QPlainTextEdit *plainTextEdit;
20     QRadioButton *radioButton;
21
22 private slots:
23     /* 槽函数 */
24     void radioButtonClicked();
25
```

```
26  };
27  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2   #include <QDir>
3   #include <QTextStream>
4   #include <QCoreApplication>
5
6   MainWindow::MainWindow(QWidget *parent)
7       : QMainWindow(parent)
8   {
9       /* 设置当前程序的工作目录为可执行程序的工作目录 */
10      QDir::setCurrent(QCoreApplication::applicationDirPath());
11
12      this->setGeometry(0, 0, 800, 480);
13
14      plainTextEdit = new QPlainTextEdit(this);
15      plainTextEdit->setGeometry(0, 50, 800, 430);
16
17      radioButton = new QRadioButton(this);
18      radioButton->setGeometry(650, 20, 100, 20);
19      radioButton->setText("只读模式");
20
21      /* 打开可执行程序目录里的 moc_mainwindow.cpp，注意如果是 Windows 下
22     moc_mainwindow.cpp 并不在当前目录，而在上一级目录"../moc_mainwindow.cpp
"*/
23      QFile file("moc_mainwindow.cpp");
24
25      /* 以只读模式打开，但是可以在 plainTextEdit 里编辑 */
26      file.open((QFile::ReadOnly | QFile::Text));
27
28      /* 加载到文件流 */
29      QTextStream in(&file);
30
31      /* 从文本流中读取全部 */
32      plainTextEdit->insertPlainText(in.readAll());
33
34      /* 信号槽连接 */
35      connect(radioButton, SIGNAL(clicked()), this,
36              SLOT(radioButtonClicked()));
37
38  }
```

```
39
40  MainWindow::~MainWindow()
41  {
42  }
43
44  void MainWindow::radioButtonClicked()
45  {
46      /* 检查 radioButton 是否选中 */
47      if(radioButton->isChecked()) {
48          /* 设置为只读模式 */
49          plainTextEdit->setReadOnly(true);
50      } else {
51          /* 设置为非只读模式 */
52          plainTextEdit->setReadOnly(false);
53      }
54  }
```

第 44 和 54 行，检查 radioButton 是否选中。

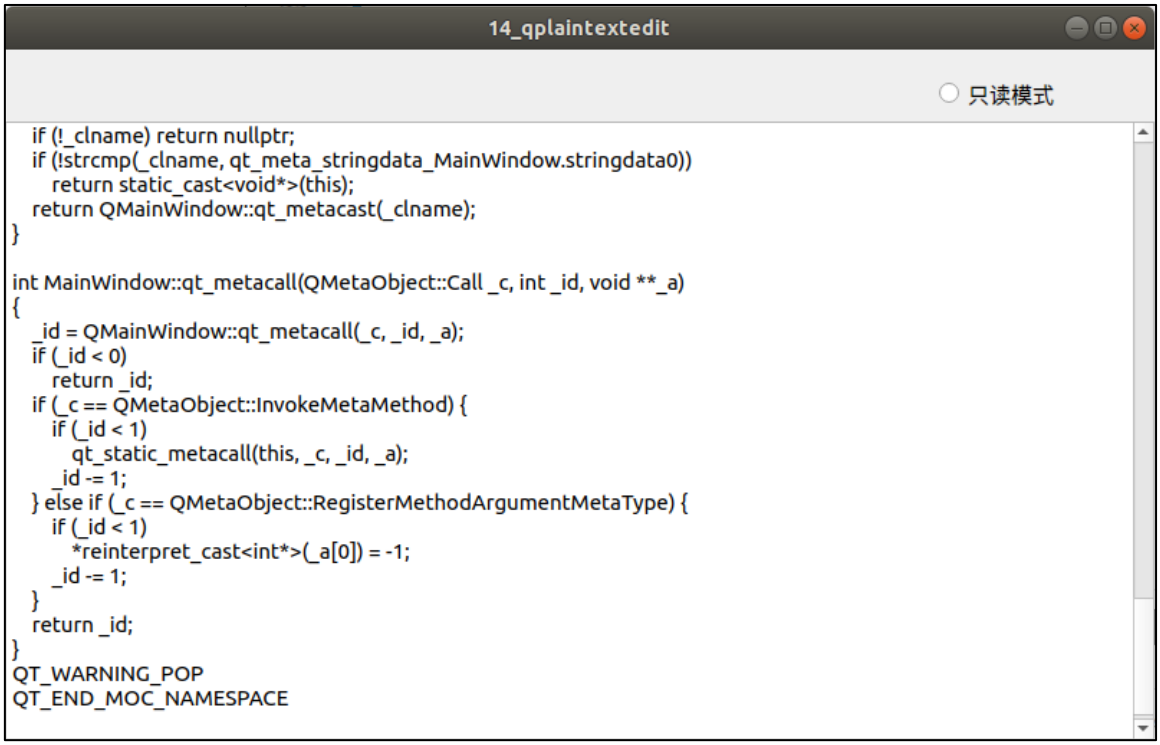
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.2.5.3 运行效果

程序编译运行的结果如下，当程序正常运行后会打开程序当前路径下的”moc_mainwindow.cpp”文件，（注意在 Windows 下 moc_mainwindow.cpp 应该写成”../moc_mainwindow.cpp”），且在 QPlainTextEdit 编辑框下是可编辑的，当选中程序界面上的只读模式时，QPlainTextEdit 编辑框就不可以再编辑。相反可以取消只读模式则可以再编辑。



有了 QTextEdit，为什么还有 QPlainTextEdit？QPlainTextEdit 可以理解为 QTextEdit 的低配版。QPlainTextEdit 支持纯文本显示，QTextEdit 支持富文本（支持多种格式，比如插入图片，链接等）显示。就是多一个样式。QPlainTextEdit 显示的效率比 QTextEdit 高，如果需要显示大量文字，尤其是需要滚动条来回滚动的时候，QPlainTextEdit 要好很多。

7.2.6 QSpinBox

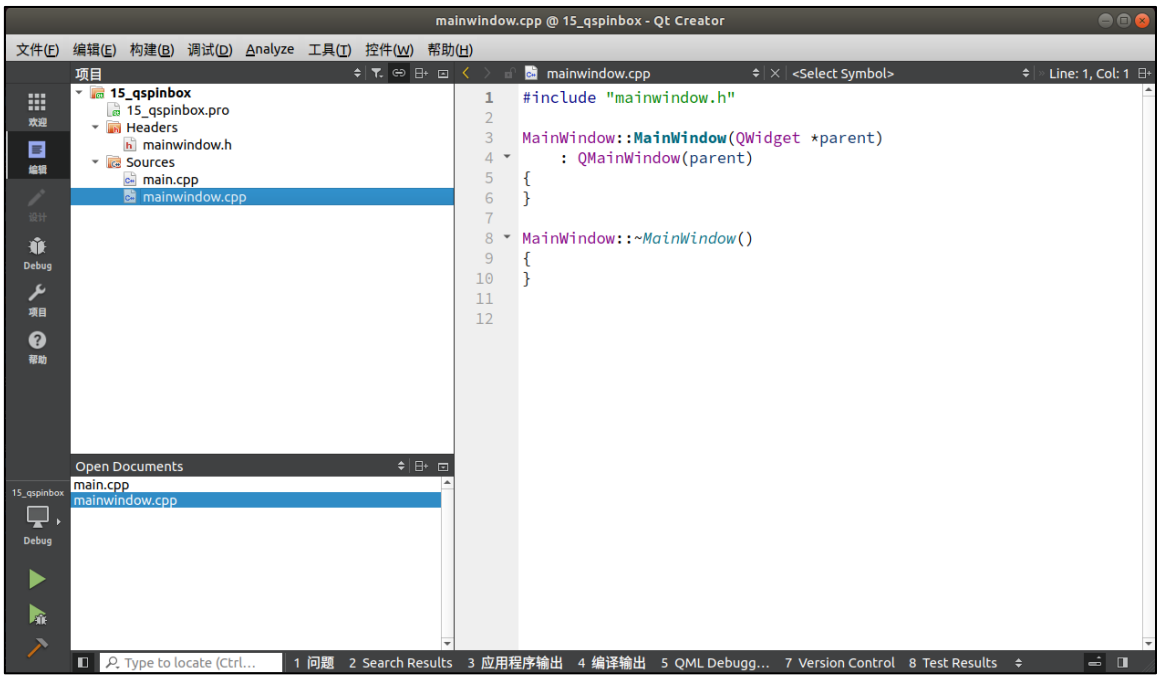
7.2.6.1 控件简介

QSpinBox 类提供了一个微调框小部件。

7.2.6.2 用法示例

例 15_qspinbox 窗口背景不透明调节器（难度：简单），用一个 QSpinBox 来调节程序窗体的整体不透明度。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSpinBox>

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    /* 声明一个 QSpinBox 对象 */
    QSpinBox *spinBox;
private slots:
    /* 槽函数 */
    void spinBoxValueChanged(int);
};
#endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    this->setGeometry(0, 0, 800, 480);

    /* 设置主窗口背景颜色，rgb 颜色标准，a 代表不透明度（0～100）*/
    this->setStyleSheet("QMainWindow{background-color: "
                        "rgba(100, 100, 100, 100%) }");

    spinBox = new QSpinBox(this);
    spinBox->setGeometry(350, 200, 150, 30);

    /* 设置范围 0～100 */
    spinBox->setRange(0, 100);

    /* 设置步长为 10 */
    spinBox->setSingleStep(10);

    /* 设置初始值为 100 */
    spinBox->setValue(100);

    /* 设置后缀 */
    spinBox->setSuffix("%不透明度");

    /* 信号槽连接 */
    connect(spinBox,SIGNAL(valueChanged(int)), this,
            SLOT(spinBoxValueChanged(int)));
}

MainWindow::~MainWindow()
{
}

void MainWindow::spinBoxValueChanged(int opacity)
{
    /* 转换为 double 数据类型 */
    double  dobleopacity = (double)opacity / 100;

    /* 设置窗体不透明度，范围是 0.0～1.0。1 则为不透明，0 为全透明 */
    this->setWindowOpacity(dobleopacity);
}
```

第 42 行，设置主窗体的不透明度。

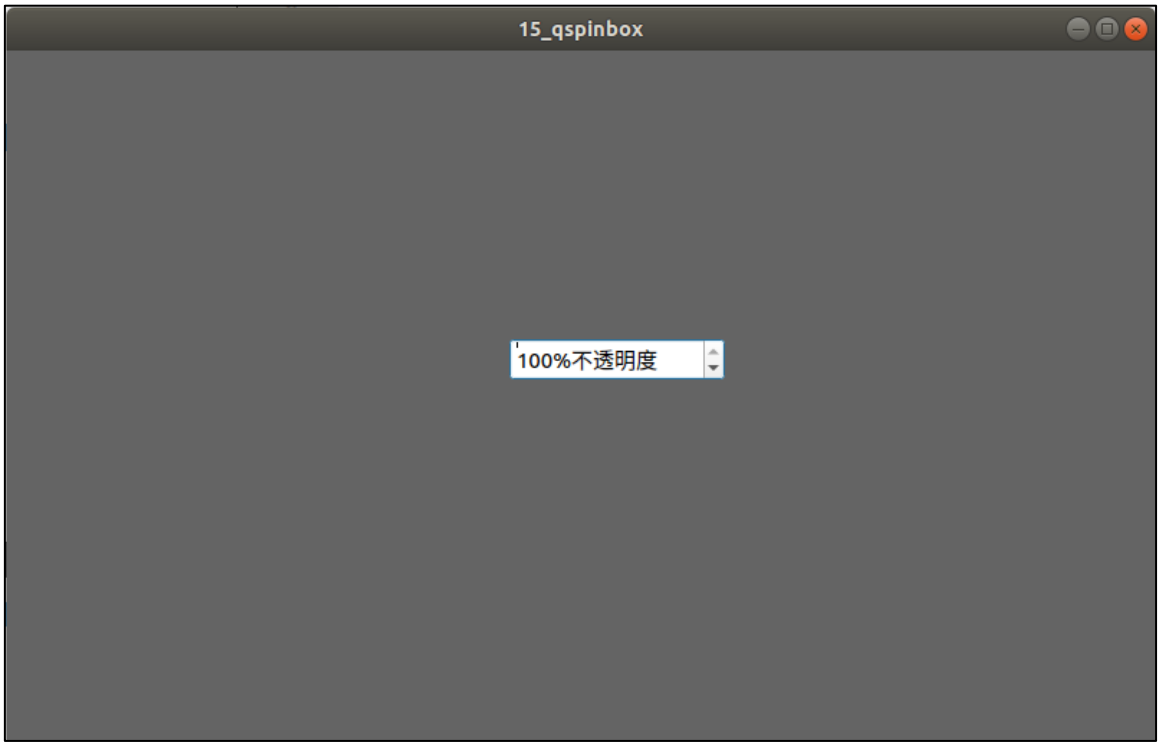
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1  #include "mainwindow.h"
2
3  #include <QApplication>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8      MainWindow w;
9      w.show();
10     return a.exec();
11 }
```

7.2.6.3 运行效果

程序编译运行的结果如下，程序初始化界面时是全不透明，不透明度值为 100%，当点击向下调节 SpinBox 后，整个窗体的不透明将会变小。当不透明度的值变小时，窗口将透明化。



7.2.7 QDoubleSpinBox

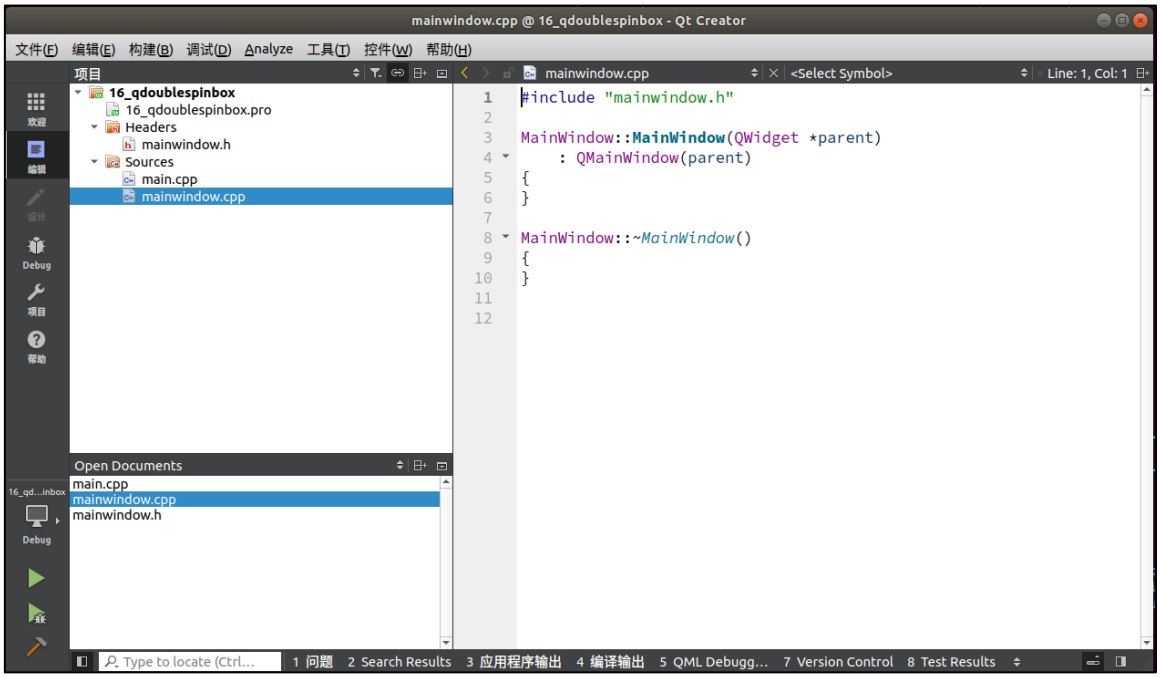
7.2.7.1 控件简介

QDoubleSpinBox 类提供了一个用于处理浮点值微调框小部件。与 QSpinBox 作用基本一样，与 QSpinBox 不同的是，QDoubleSpinBox 类处理的是浮点值数据。

7.2.7.2 用法示例

例 16_qdoublespinbox 窗口大小调节器（难度：简单），用一个 QDoubleSpinBox 来调节程序窗口的整体大小。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QDoubleSpinBox>
6
7  class MainWindow : public QMainWindow
8  {
9      Q_OBJECT
10
11 public:
12     MainWindow(QWidget *parent = nullptr);
13     ~MainWindow();
14
15 private:
16     /* 声明一个 QDoubleSpinBox 对象 */
17     QDoubleSpinBox *doubleSpinBox;
18
```

```
19  private slots:
20      /* 槽函数 */
21      void doubleSpinBoxValueChanged(double);
22
23  };
24  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       this->setGeometry(0, 0, 800, 480);
7
8       /* 实例化和设置显示的位置与大小 */
9       doubleSpinBox = new QDoubleSpinBox(this);
10      doubleSpinBox->setGeometry((this->width() - 200) / 2,
11                                 (this->height() - 30) / 2,
12                                 200, 30);
13      /* 设置前缀 */
14      doubleSpinBox->setPrefix("窗口大小");
15
16      /* 设置后缀 */
17      doubleSpinBox->setSuffix("%");
18
19      /* 设置范围 */
20      doubleSpinBox->setRange(50.00, 100.00);
21
22      /* 设置初始值 */
23      doubleSpinBox->setValue(100.00);
24
25      /* 设置步长 */
26      doubleSpinBox->setSingleStep(0.1);
27
28      /* 信号槽连接 */
29      connect(doubleSpinBox, SIGNAL(valueChanged(double)),
30              SLOT(doubleSpinBoxValueChanged(double)));
31
32  }
33
34  MainWindow::~MainWindow()
35  {
```

```
36  }
37
38  void MainWindow::doubleSpinBoxValueChanged(double value)
39  {
40      /* 重新计算窗口的宽与高 */
41      int w = 800 * value / 100;
42      int h = 480 * value / 100;
43
44      /* 重新设置窗口的宽与高 */
45      this->setGeometry(0, 0, w, h);
46
47      /* 重新设置 doubleSpinBox 的显示位置 */
48      doubleSpinBox->setGeometry((this->width() - 200) / 2,
49                                 (this->height() - 30) / 2,
50                                 200, 30);
51
52  }
```

第 35 至 49 行，重新设置主窗体的宽高和 doubleSpinBox 的显示位置。

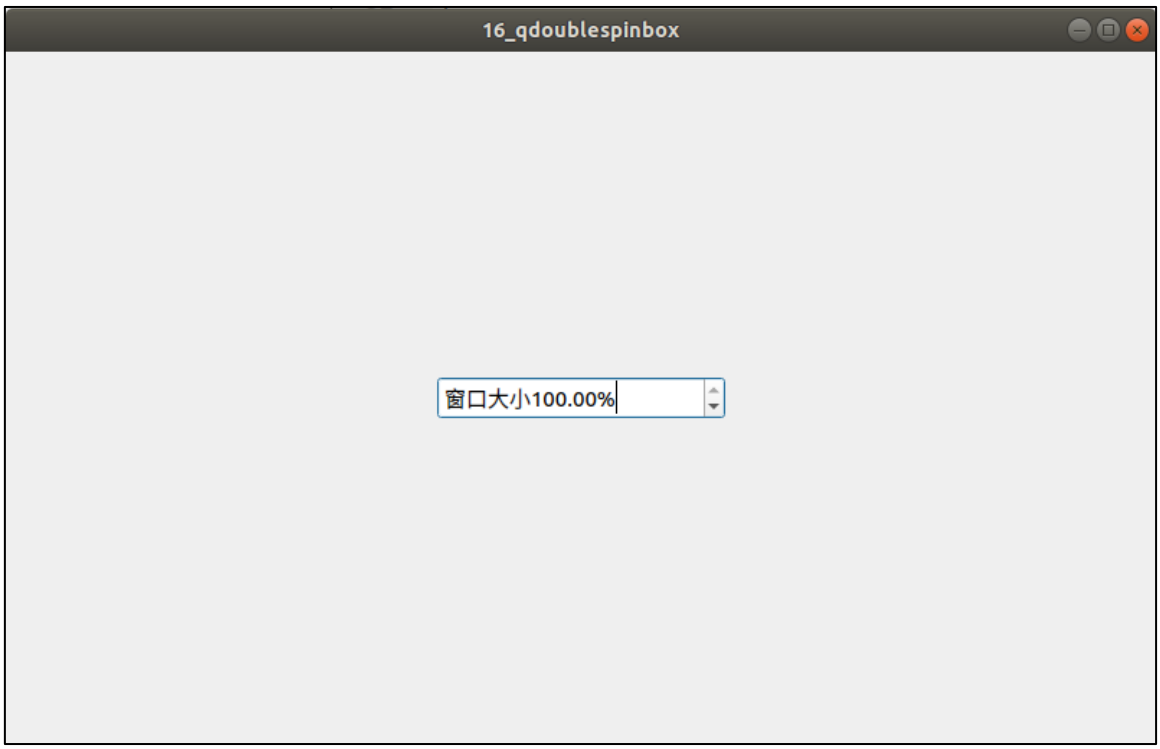
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.2.7.3 运行效果

程序编译运行的结果如下，程序初始化界面窗口大小值为 100%，当点击向下调节 QDoubleSpinBox 时，整个窗体将按 QDoubleSpinBox 里数值的比例缩小，最小为 50.00%，相反当点击向上调节 QDoubleSpinBox 时，窗口大小将整体变大，最大为 100.00%。



7.2.8 QTimeEdit

7.2.8.1 控件简介

QTimeEdit 类提供一个基于 QDateTimeEdit 类编辑时间的小部件。例在 7.2.10 小节。

7.2.9 QDateEdit

7.2.9.1 控件简介

QDateEdit 类提供一个基于 QDateTimeEdit 类编辑时间的小部件。例在 7.2.10 小节。

7.2.10 QDateTimeEdit

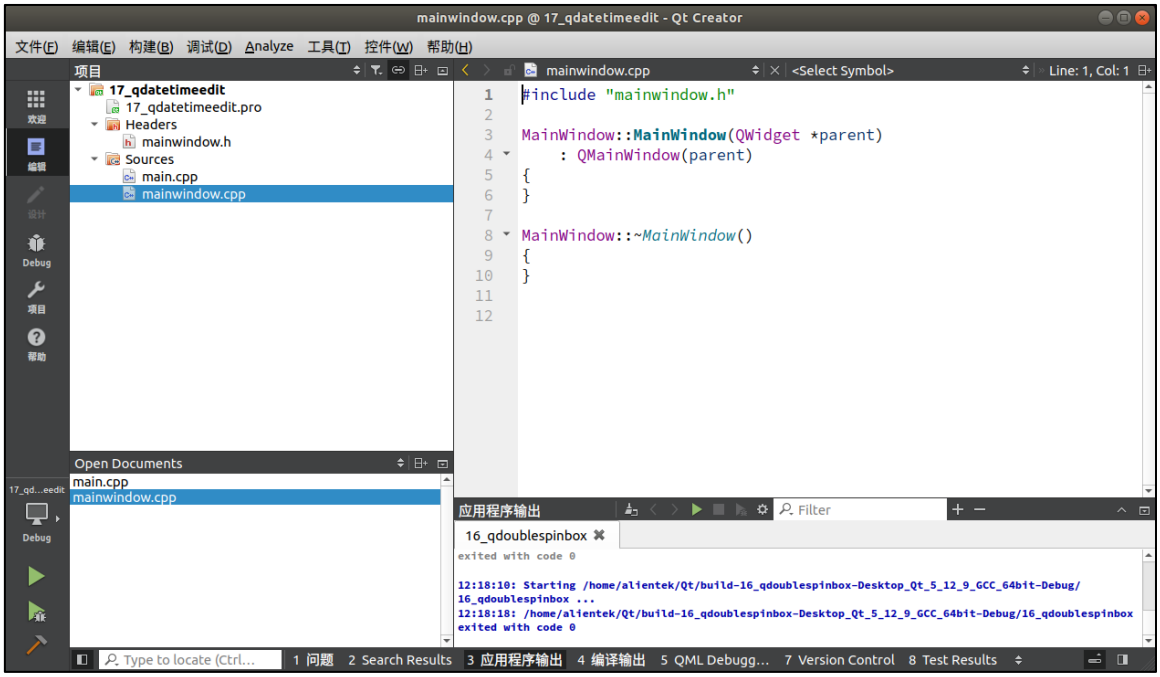
7.2.10.1 控件简介

从名字可知 QDateTimeEdit 类提供了一个用于编辑日期和时间的小部件。QDateTimeEdit 允许用户使用键盘或箭头键编辑日期，以增加或减少日期和时间值。箭头键可用于在 QDateTimeEdit 框中从一个区域移动到另一个区域。实际上是 QDateTimeEdit 和 QDateEdit 的组合。

7.2.10.2 用法示例

例 17_qdatetimeedit 时间日期展示（难度简单），使用一个 QDateTimeEdit，一个 QTimeEdit 以及一个 QDateEdit，传入当前系统时间与日期，展示简单的日期时间控件使用方法。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QDateTimeEdit>
6  #include <QTimeEdit>
7  #include <QDateEdit>
8
9  class MainWindow : public QMainWindow
10 {
11     Q_OBJECT
12
13 public:
14     MainWindow(QWidget *parent = nullptr);
15     ~MainWindow();
16
17 private:
18     /* 声明对象 */
19     QDateTimeEdit *dateTimeEdit;
20     QTimeEdit *timeEdit;
21     QDateEdit *dateEdit;
22 };
23 #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    /* 设置位置与大小 */
    this->setGeometry(0, 0, 800, 480);

    /*实例化对象，传入当前日期与时间*/
    dateTimeEdit = new QDateTimeEdit(
                QDateTime::currentDateTime(),this);
    dateTimeEdit->setGeometry(300, 200, 200, 30);
    /* 弹出日期控件与否 */
    //dateTimeEdit->setCalendarPopup(true);

    /* 实例化对象，传入当前时间 */
    timeEdit = new QTimeEdit(QTime::currentTime(),this);
    timeEdit->setGeometry(300, 240, 200, 30);

    /* 实例化对象，传入当前日期 */
    dateEdit = new QDateEdit(QDate::currentDate(),this);
    dateEdit->setGeometry(300, 280, 200, 30);
}

MainWindow::~MainWindow()
{
}
```

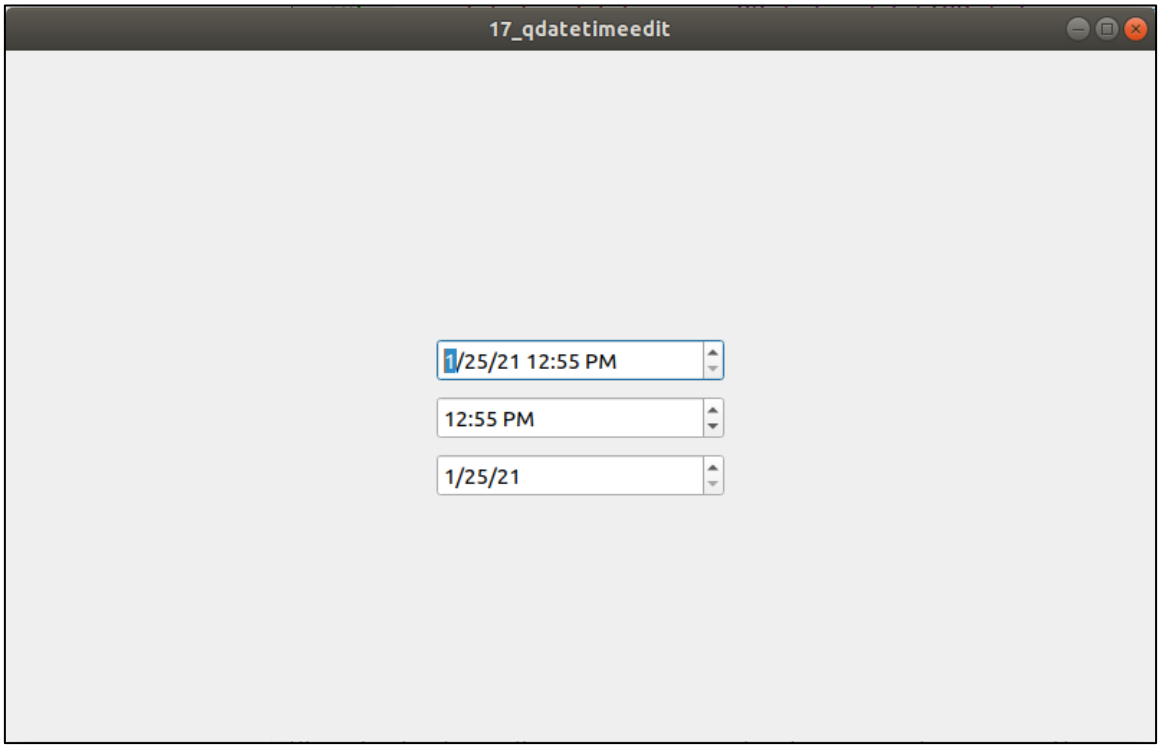
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

7.2.10.3 运行效果

程序编译运行的结果如下，当程序初始化时，分别显示系统当前的时间与日期（注意，windows 下 Qt 程序显示的格式可能不一样，下图为 linux 下的 Qt 程序日期显示格式）。



7.2.11 QDial

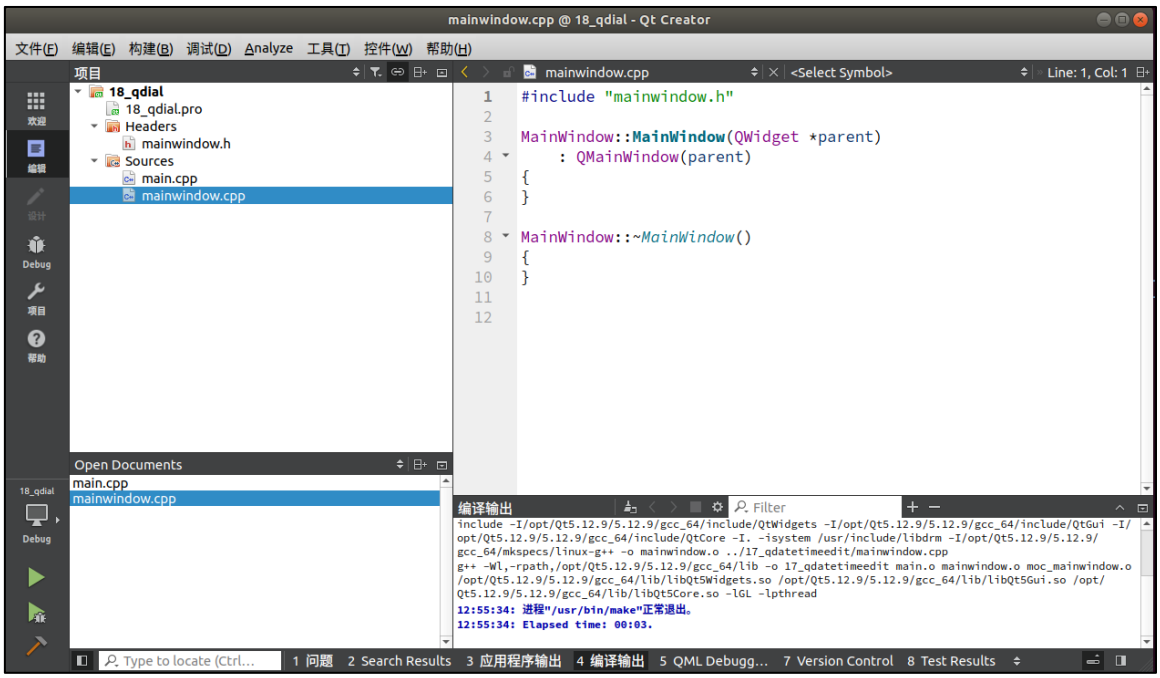
7.2.11.1 控件简介

QDial 类提供了一个圆形范围控制(如速度计或电位器)。QDial 用于当用户需要在可编程定义的范围内控制一个值，并且该范围要么是环绕的(例如，从 0 到 359 度测量的角度)，要么对话框布局需要一个正方形小部件。由于 QDial 从 QAbstractSlider 继承，因此拨号的行为与滑块类似。 当 wrapping（）为 false（默认设置）时，滑块和刻度盘之间没有真正的区别。 它们共享相同的信号，插槽和成员功能。 您使用哪一个取决于您的用户期望和应用程序类型。

7.2.11.2 用法示例

例 18_qdial 车速表（难度：简单），使用一个 QDial，以一个 QLabel，演示 QDial 的用法。当程序初始化界面后，拖动滑块的位置，label 则会显示 dial 的值。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QDial>
5  #include <QLabel>
6  #include <QMainWindow>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15
16 private:
17     /* 声明对象 */
18     QDial *dial;
19     QLabel *label;
20
21 private slots:
22     /* 槽函数 */
23     void dialValueChanged(int);
24
25 };
```

```
26  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗体的位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 实例化对象和设置显示位置与大小 */
10      dial = new QDial(this);
11      dial->setGeometry(300, 100, 200, 200);
12
13      /* 设置页长（两个最大刻度的间距）*/
14      dial->setPageStep(10);
15
16      /* 设置刻度可见 */
17      dial->setNotchesVisible(true);
18
19      /* 设置两个凹槽之间的目标像素数 */
20      dial->setNotchTarget(1.00);
21
22      /* 设置 dial 值的范围 */
23      dial->setRange(0,100);
24
25      /* 开启后可指向圆的任何角度 */
26      //dial->setWrapping(true);
27
28      /* 实例化对象和设置显示位置与大小 */
29      label = new QLabel(this);
30      label->setGeometry(370, 300, 200, 50);
31
32      /* 初始化为 0km/h */
33      label->setText("0km/h");
34
35      /* 信号槽连接 */
36      connect(dial, SIGNAL(valueChanged(int)),
37              this, SLOT(dialValueChanged(int)));
38  }
39
```

```
40  MainWindow::~MainWindow()
41  {
42  }
43
44  void MainWindow::dialValueChanged(int val)
45  {
46      /* QString::number()转换成字符串 */
47      label->setText(QString::number(val) + "km/h");
48  }
```

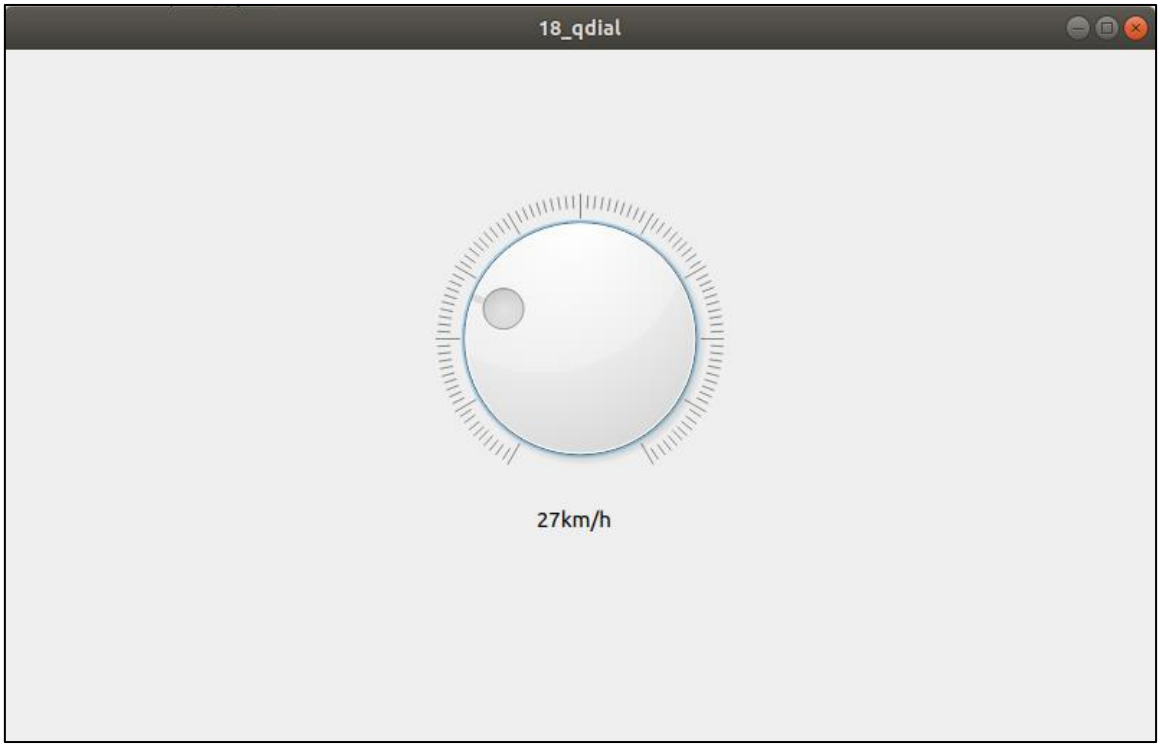
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.2.11.3 运行效果

程序编译运行的结果如下，当程序初始化时，QDial 控件的显示如下（注意，windows 下 QDial 控件显示的格式可能不一样，下图为 linux 下的 QDial 控件的显示样式）。当用鼠标拖动滑块或者按键盘的上下左右方向键时，label 则会显示当前“车速”。



QDial 在很多场合都能使用，比如音量控制，汽车仪表盘，芝麻信用分等场合都可以使用到，只是需要我们有这个创意和想法，还需要个人的美工基础。

7.2.12 QScrollBar

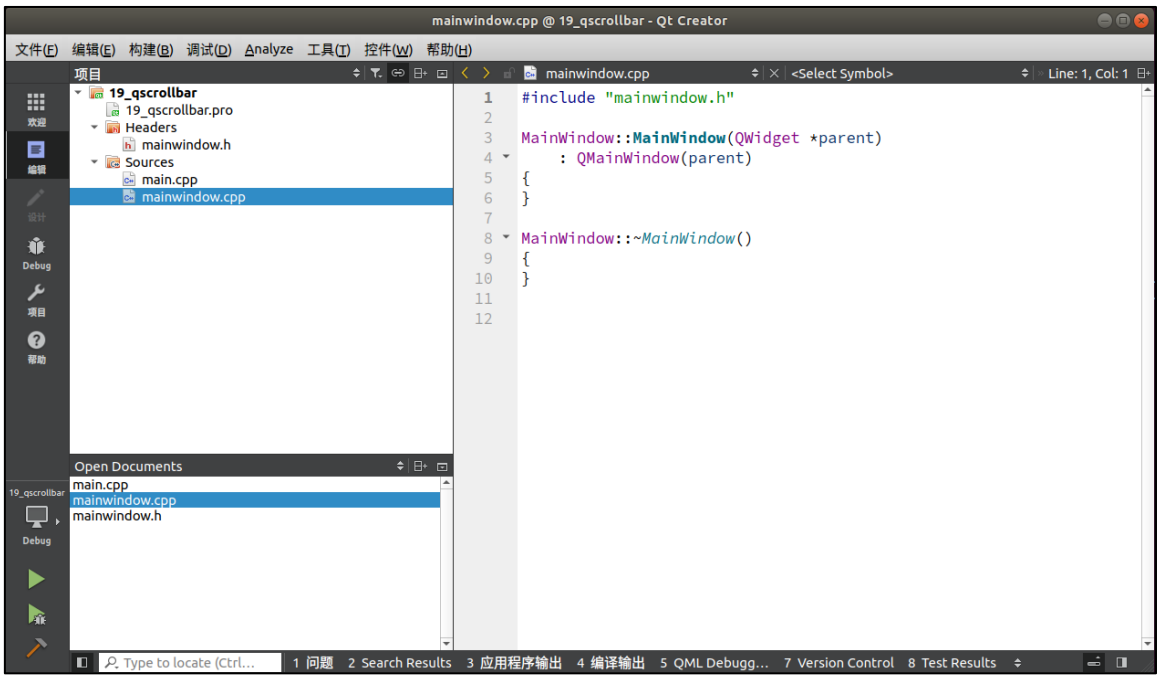
7.2.12.1 控件简介

QScrollBar 继承 QAbstractSlider。QScrollBar 小部件提供垂直或水平滚动条，允许用户访问比用于显示文档的小部件大的文档部分。它提供了用户在文档中的当前位置和可见文档数量的可视化指示。滚动条通常配有其他控件，可以实现更精确的导航(这里指浏览到精确的位置)。

7.2.12.2 用法示例

例 19_qscrollbar 创建水平和垂直滚动条（难度：简单），使用 QScrollBar 类实例化两个控件，一个是水平滚动条，另一个是垂直滚动条，中间用一个标签文本来显示。(本例只创建实例，不对效果进行细化（请注意：一般水平或垂直滚动条都用 QScrollArea 搭配其他控件使用，不单独使用 QScrollBar 去创建滚动条，有些控件“自带”滚动条，例如 QListWidget 等，都可以用 QScrollArea 来设置它的属性)。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QScrollBar>
6  #include <QLabel>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15
16 private:
17     /* 声明水平滚动条对象 */
18     QScrollBar *horizontalScrollBar;
19
20     /* 声明垂直滚动条对象 */
21     QScrollBar *verticalScrollBar;
22
23     /* 声明标签文本 */
24     QLabel *label;
25 };
```

```
26  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗体大小，位置 */
7       this->setGeometry(0,0,800,480);
8
9       /* 实例化水平滚动条及设置位置大小 */
10      horizontalScrollBar = new QScrollBar(Qt::Horizontal, this);
11      horizontalScrollBar->setGeometry(0, 450, 800, 30);
12
13      /* 实例化垂直滚动条及设置位置大小 */
14      verticalScrollBar = new QScrollBar(Qt::Vertical, this);
15      verticalScrollBar->setGeometry(770, 0, 30, 480);
16
17      /* 实例化，标签文本 */
18      label = new QLabel(this);
19      /* 设置文本 */
20      label->setText("这是一个测试");
21      /* 设置位置大小 */
22      label->setGeometry(300, 200, 100, 20);
23  }
24
25  MainWindow::~MainWindow()
26  {
27  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
```

```
11  }
```

7.2.12.3 运行效果

程序编译运行的结果。如下当程序初始化时，滚动条控件的显示如下（注意，windows 下滚动条控件显示的格式可能不一样，下图为 linux 下的滚动条控件的显示样式）。

7.2.13 QSlider

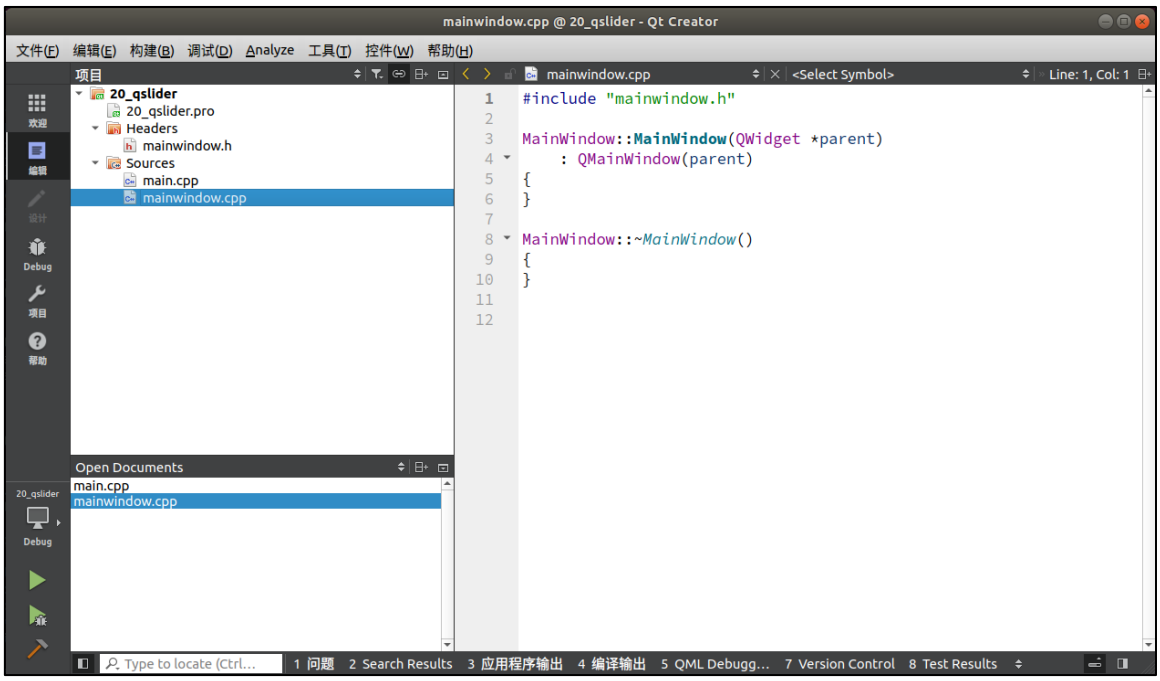
7.2.13.1 控件简介

QSlider 继承 QAbstractSlider。QScrollBar 类提供垂直或水平滑动条小部件，滑动条是用于控制有界值的典型小部件。它允许用户沿着水平或垂直凹槽移动滑块手柄，并将手柄的位置转换为合法范围内的整数值。

7.2.13.2 用法示例

例 20_qslider 创建水平和垂直滑动条（难度：简单）创建两个 QSlider 对象，一个是水平滑动条，另一个是垂直滑动条；用一个 Label 来显示当前水平或垂直滑动条当前的值。设置水平滑动条与水平滑动条相互关联，通过滑动其中一个滑动条的值相应的也会改变另一个滑动条的值。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QSlider>
6  #include <QLabel>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15
16 private:
17     /* 声明对象 */
18     QSlider *horizontalSlider;
19     QSlider *verticalSlider;
20     QLabel *label;
21 private slots:
22     /* 槽函数 */
23     void horizontalSliderValueChanged(int);
24     void verticalSliderValueChanged(int);
25
```

```
26  };
27  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       this->setGeometry(0, 0, 800, 480);
7
8       /* 实例化水平滑动条对象*/
9       horizontalSlider = new QSlider(Qt::Horizontal, this);
10
11      /* 设置显示的位置与大小 */
12      horizontalSlider->setGeometry(250, 100, 200, 20);
13
14      /* 设置值范围 */
15      horizontalSlider->setRange(0, 100);
16
17      /* 实例化垂直滑动条对象 */
18      verticalSlider = new QSlider(Qt::Vertical, this);
19
20      /* 设置显示的位置与大小 */
21      verticalSlider->setGeometry(200, 50, 20, 200);
22
23      /* 设置值范围 */
24      verticalSlider->setRange(0, 100);
25
26      /* 实例化标签文本 */
27      label = new QLabel("滑动条值: 0", this);
28      label->setGeometry(250, 200, 100, 20);
29
30      /* 信号槽连接 */
31      connect(horizontalSlider, SIGNAL(valueChanged(int)),
32              this, SLOT(horizontalSliderValueChanged(int)));
33      connect(verticalSlider, SIGNAL(valueChanged(int)),
34              this, SLOT(verticalSliderValueChanged(int)));
35
36  }
37
38  MainWindow::~MainWindow()
39  {
```

```
40  }
41
42  void MainWindow::horizontalSliderValueChanged(int val)
43  {
44      /* 当水平滑动条的值改变时，改变垂直滑动条的值 */
45      verticalSlider->setSliderPosition(val);
46
47      /* 将 int 类型转变成字符 */
48
49      QString str = "滑动条值: " + QString::number(val);
50
51      /* 显示当前垂直或水平滑动条的值 */
52      label->setText(str);
53
54  }
55
56  void MainWindow::verticalSliderValueChanged(int val)
57  {
58      /* 当垂直滑动条的值改变时，改变水平滑动条的值 */
59      horizontalSlider->setSliderPosition(val);
60  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.2.13.3 运行效果

程序编译运行的结果如下。当程序初始化时，滑动条控件的显示如下（注意，windows 下滑动条控件显示的格式可能不一样，下图为 linux 下的滑动条控件的显示样式）。Label 显示的初始值为 0，当拖动任意一个滑动条来改变当前的值，另一个滑动条的值也在变。

7.2.14 QKeySequenceEdit

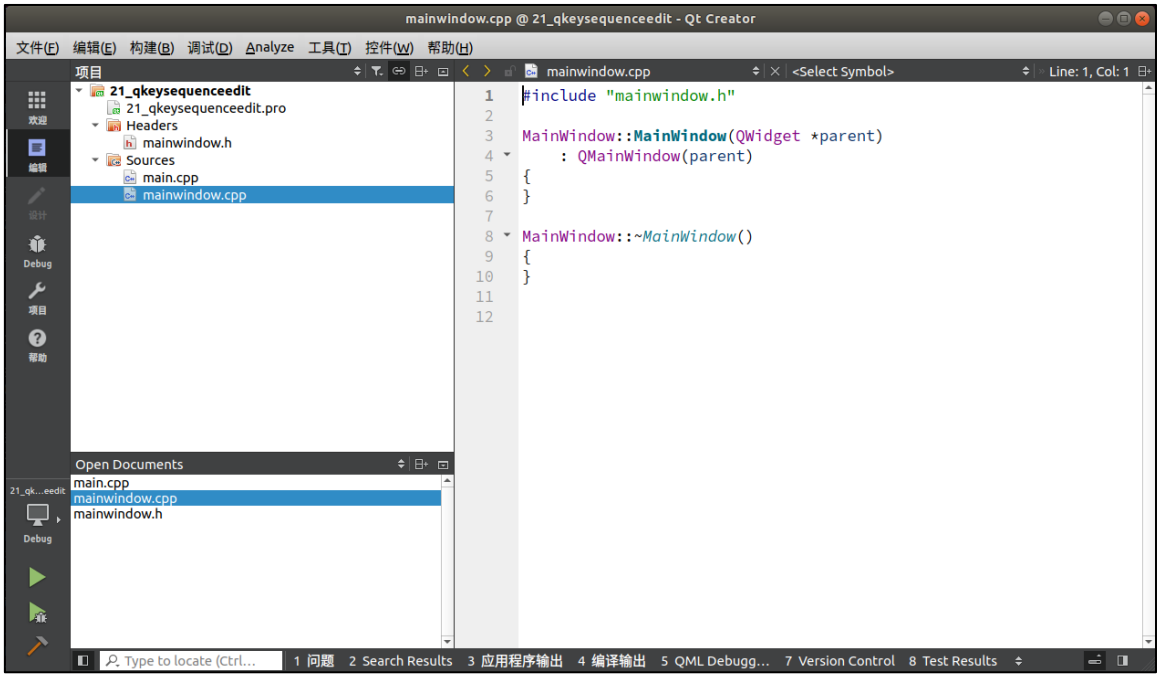
7.2.14.1 控件简介

QKeySequenceEdit 继承 QWidget。这个小部件允许用户选择 QKeySequence, QKeySequence 通常用作快捷方式。当小部件接收到焦点并在用户释放最后一个键后一秒结束时，将启动记录，通常用作记录快捷键。

7.2.14.2 用法示例

例 21_qkeysequenceedit 自定义快捷键（难度：简单），通常 KeySequenceEdit 控件记录快捷键后与 Qt 键盘事件搭配来使用，由于教程后面才说到事件，所以先不与 Qt 键盘事件搭配使用。下面使用一个 QKeySequenceEdit 控件，然后判断输入的组合键是否为 Ctrl + Q，若是，则关闭窗口，退出程序，如果不是，则继续更新记录组合键。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QKeySequenceEdit>
6
7  class MainWindow : public QMainWindow
8  {
9      Q_OBJECT
10
11 public:
12     MainWindow(QWidget *parent = nullptr);
13     ~MainWindow();
14
15 private:
16     /* 声明 QKeySequenceEdit 对象 */
17     QKeySequenceEdit  *keySequenceEdit;
18
19 private slots:
20     /* 声明槽 */
21     void KSEKeySequenceChanged(const QKeySequence &);
22
23 };
24 #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1  #include "mainwindow.h"
2  #include <QDebug>
3
4  MainWindow::MainWindow(QWidget *parent)
5      : QMainWindow(parent)
6  {
7      /* 主窗体设置位置与大小 */
8      this->setGeometry(0, 0, 800, 480);
9
10     /* 实例化 */
11     keySequenceEdit = new QKeySequenceEdit(this);
12
13     /* 设置位置与大小 */
14     keySequenceEdit->setGeometry(350, 200, 150, 30);
15
16     /* 信号槽连接 */
17     connect(keySequenceEdit,
18             SIGNAL(keySequenceChanged(const QKeySequence &)),
19             this,
20             SLOT(KSEKeySequenceChanged(const QKeySequence &))
21             );
22
23 }
24
25 MainWindow::~MainWindow()
26 {
27 }
28
29 void MainWindow::KSEKeySequenceChanged(
30         const QKeySequence &keySequence)
31 {
32     /* 判断输入的组合键是否为 Ctrl + Q，如果是则退出程序 */
33     if(keySequence == QKeySequence(tr("Ctrl+Q"))) {
34         /* 结束程序 */
35         this->close();
36     }else {
37         /* 打印出按下的组合键 */
38         qDebug()<<keySequence.toString()<<endl;
39     }
40 }
```

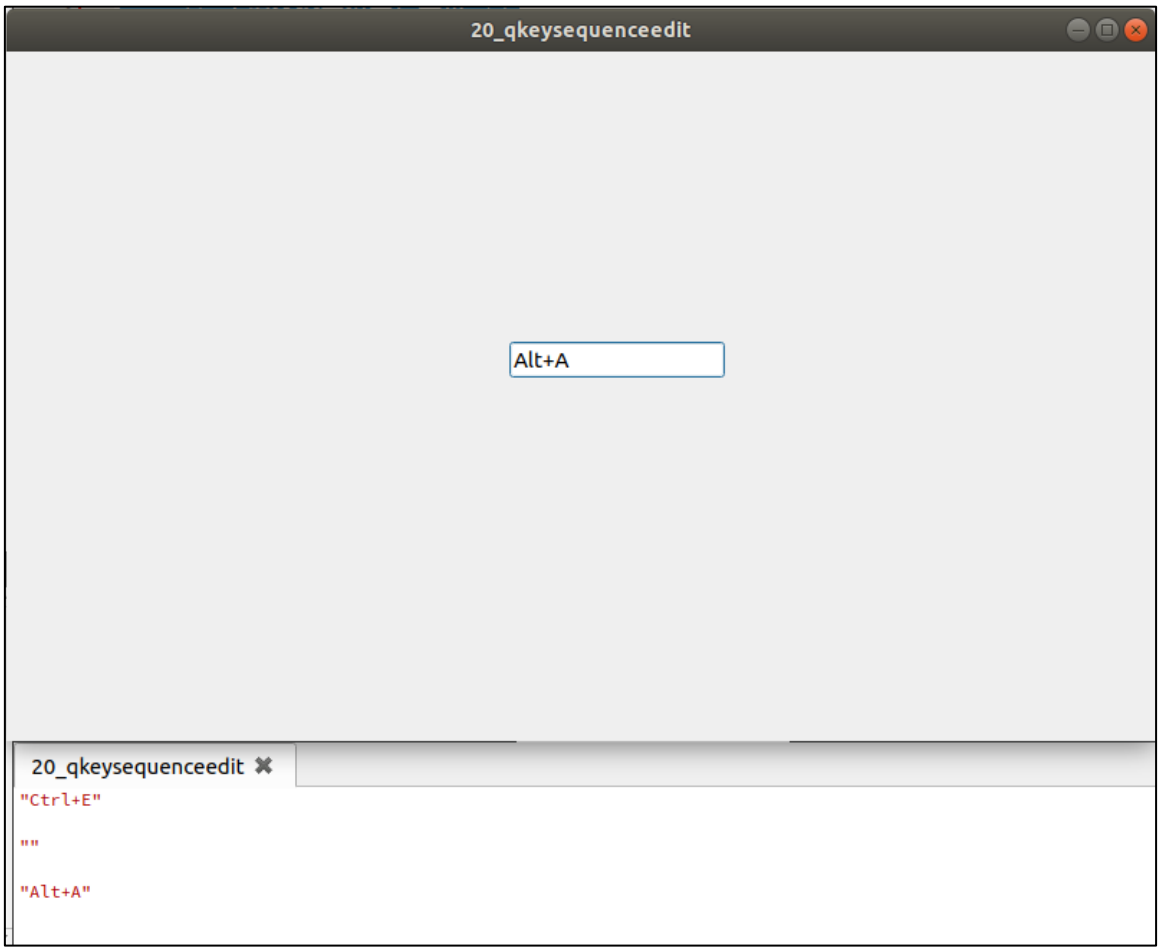
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

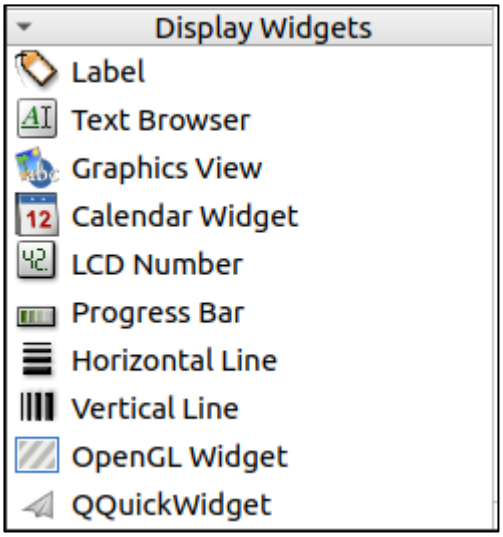
7.2.14.3 运行效果

程序编译运行的结果如下。(注意 Windows 下显示效果可能不同，下图为 linux 下运行的结果)，当焦点在 KeySequenceEdit 里时，按下键盘里的任一键或者组合键，一秒后，KeySequenceEdit 将记录了这个/这组组合键，并打印在输出信息里。直到程序按下 Ctrl + Q 组合键后，程序窗口才关闭，结束。



7.3 显示窗口部件

Qt Designer 显示窗口部件提供的面板中，提供了 10 种显示小部件。在 Qt5.5 以前的 Qt5 版本这个显示窗口部件还有一个 QWebview 部件，在 Qt5.6 开始就被移除了，取而代之的是 QWebengine，但是 QWebengine 不是以小部件的形式在这个显示窗口部件里显示。



（1） Label:标签

（2） Text Browser:文本浏览器

（3） Graphics View:图形视图

（4） Calendar Widget:日历

（5） LCD Number:液晶数字

（6） Progress Bar:进度条

（7） Horizontal Line:水平线

（8） Vertial Line:垂直线

（9） OpenGL Widget:开放式图形库工具

（10） QQuick Widget:嵌入式 QML 工具

这十六种按钮部件作用简介如下:

QLabel 提供了一种用于文本或图像显示的小部件，在前 4.1 与 4.2 某些小节已经出现过 Label 控件，只用了它显示文本，其实它还可以用于显示图像。

QCalendarWidget 继承 QWidget。QCalendarWidget 类提供了一个基于月的日历小部件，允许用户选择日期。CalendarWidget 小部件是用当前月份和年份初始化的，QCalendarWidget 还提供了几个公共插槽来更改显示的年份和月份。

QLCDNumber 继承 QFrame。QLCDNumber 小部件显示一个类似于 lcd 的数字。QLCDNumber 小部件可以显示任意大小的数字。它可以显示十进制、十六进制、八进制或二进制数字。使用 display()插槽很容易连接到数据源，该插槽被重载以接受五种参数类型中的任何一种。

QProgressBar 继承 QWidget。QProgressBar 小部件提供了一个水平或垂直的进度条。进度条用于向用户显示操作的进度，并向他们确认应用程序仍在运行。

QFrame 继承 QWidget。QFrame 类是有框架的窗口部件的基类，它绘制框架并且调用一个虚函数 drawContents()来填充这个框架。这个函数是被子类重新实现的。这里至少还有两个有用的函数：drawFrame()和 frameChanged()。

QTextBrowser 继承 QTextEdit，QTextBrowser 类提供了一个具有超文本导航的文本浏览器。该类扩展了 QTextEdit(在只读模式下)，添加了一些导航功能，以便用户可以跟踪超文本文档中的链接。

QGraphicsView 继承 QAbstractScrollArea。QGraphicsView 是图形视图框架的一部分，它提供了基于图元的模型/视图编程。QGraphicsView 在可滚动视图中可视化 QGraphicsScene 的内容。要创建带有几何项的场景，请参阅 QGraphicsScene 的文档。

要可视化场景，首先构造一个 QGraphicsView 对象，将要可视化的场景的地址传递给 QGraphicsView 的构造函数。或者，可以调用 setScene()在稍后设置场景。

Text Browser（文本浏览器）、Graphics View(图形视图)、OpenGL Widget(开放式图形库工具)、QQuick Widget(嵌入式 QML 工具)将不在本小节介绍，其中 Text Browser（文本浏览器）、Graphics View(图形视图)在 7.4 小节显示窗口之浏览器介绍。

7.3.1 QLabel

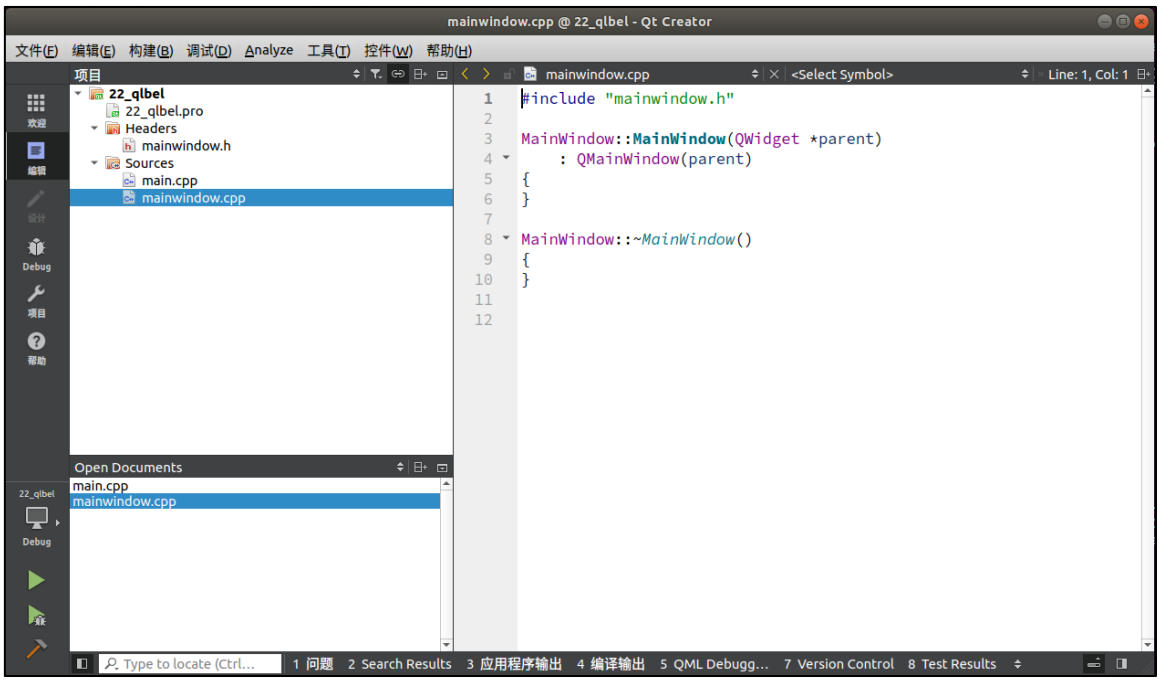
7.3.1.1 控件简介

QLabel 提供了一种用于文本或图像显示的小部件，在前 7.2.11、7.2.12 等小节已经出现过 Label 控件，只用了它显示文本，其实它还可以用于显示图像。

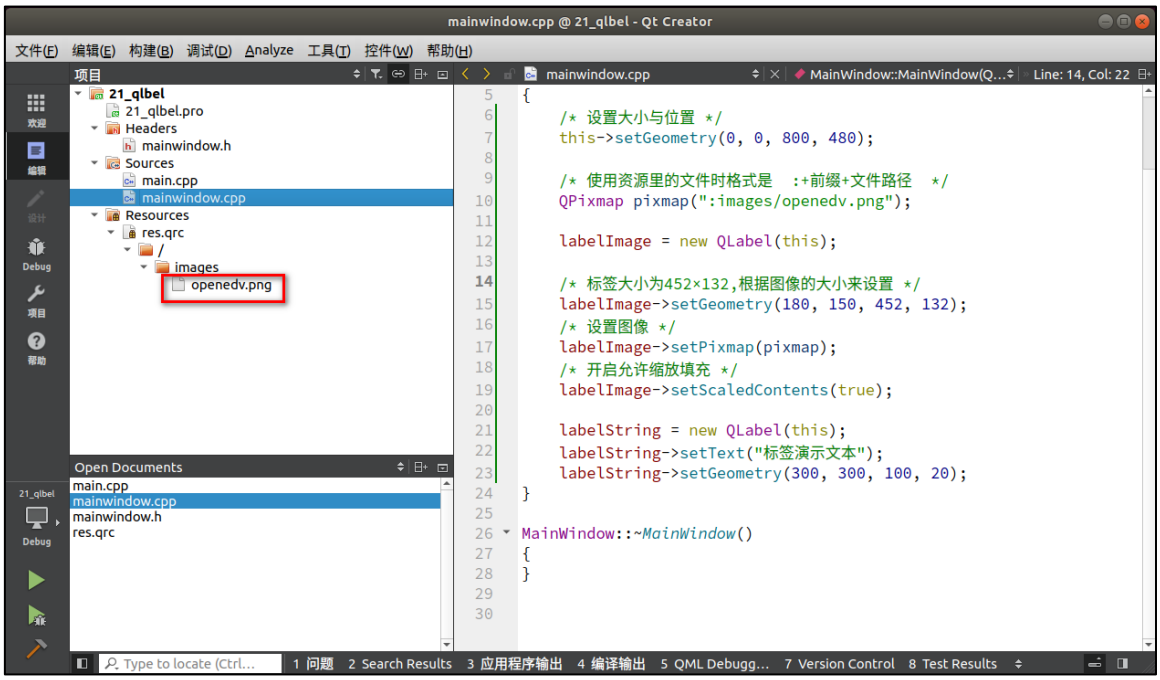
7.3.1.2 用法示例

例 22_qlbel 标签显示图像或文本（难度简单），前面已经在某些小节出现过 label 了，但只是用来显示文本内容，实际上像 button（按钮），widget，label 等都是可以用来显示图像的。只是 button 不常用来显示单张图像，因为 button 是有三种状态，分别是按下，释放，悬停。所以它不常用于单单显示一张图像。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



本例已经添加一张资源图片，如果不清楚如何添加资源图片，请参考 7.1.4 小节。添加完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QLabel>
6
```

```
7  class MainWindow : public QMainWindow
8  {
9      Q_OBJECT
10
11 public:
12     MainWindow(QWidget *parent = nullptr);
13     ~MainWindow();
14
15 private:
16     /* 用一个 QLabel 对象用于显示字符串 */
17     QLabel *labelString;
18
19     /* 用一个 QLabel 对象用于显示图像 */
20     QLabel *labelImage;
21 };
22 #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1  #include "mainwindow.h"
2
3  MainWindow::MainWindow(QWidget *parent)
4      : QMainWindow(parent)
5  {
6      /* 设置大小与位置 */
7      this->setGeometry(0, 0, 800, 480);
8
9      /* 使用资源里的文件时格式是  :+前缀+文件路径  */
10     QPixmap pixmap(":images/openedv.png");
11
12     labelImage = new QLabel(this);
13
14     /* 标签大小为 452×132,根据图像的大小来设置 */
15     labelImage->setGeometry(180, 150, 452, 132);
16     /* 设置图像 */
17     labelImage->setPixmap(pixmap);
18     /* 开启允许缩放填充 */
19     labelImage->setScaledContents(true);
20
21     labelString = new QLabel(this);
22     labelString->setText("标签演示文本");
23     labelString->setGeometry(300, 300, 100, 20);
24 }
25
```

```
26  MainWindow::~MainWindow()
27  {
28  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.3.1.3 运行效果

程序编译运行的结果如下。为了避免引起图像被压缩或者拉伸，本次设置标签的大小为452*132，这是根据图像的实际大小来设置的。

7.3.2 QCalendarWidget

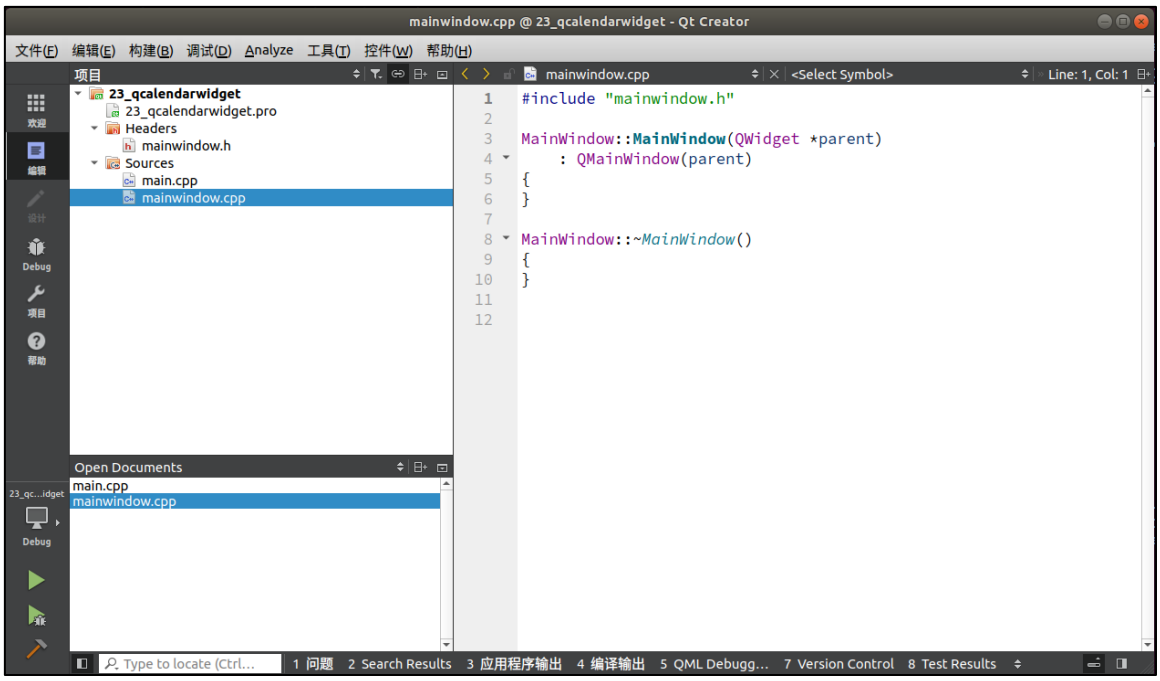
7.3.2.1 控件简介

QCalendarWidget 继承 QWidget。QCalendarWidget 类提供了一个基于月的日历小部件，允许用户选择日期。CalendarWidget 小部件是用当前月份和年份初始化的，QCalendarWidget 还提供了几个公共插槽来更改显示的年份和月份。

7.3.2.2 用法示例

例 23_qcalendarwidget 日历简单的使用（难度：简单），本例只是简单的使用日历控件来达到一定的效果。使用一个 CalendarWidget 控件，一个 Label 来显示当前日历所选择的日期，一个 pushButton 按钮，点击 pushButton 按钮来回到当前系统的日期。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QCalendarWidget>
6  #include <QPushButton>
7  #include <QLabel>
8
9  class MainWindow : public QMainWindow
10 {
```

```
11      Q_OBJECT
12
13  public:
14      MainWindow(QWidget *parent = nullptr);
15      ~MainWindow();
16
17  private:
18      /* 声明 QCalendarWidget，QPushButton，QLabel 对象 */
19      QCalendarWidget *calendarWidget;
20      QPushButton *pushButton;
21      QLabel *label;
22
23  private slots:
24      /* 槽函数 */
25      void calendarWidgetSelectionChanged();
26      void pushButtonClicked();
27
28  };
29  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置位置与大小，下同 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 对象实例化设置显示的位置与大小 */
10      calendarWidget = new QCalendarWidget(this);
11      calendarWidget->setGeometry(200, 20, 400, 300);
12
13      QFont font;
14      /* 设置日历里字体的大小为 10 像素 */
15      font.setPixelSize(10);
16      calendarWidget->setFont(font);
17
18      /* 对象实例化设置显示的位置与大小 */
19      pushButton = new QPushButton("回到当前日期",this);
20      pushButton->setGeometry(200, 350, 100, 30);
21
22      /* 对象实例化设置显示的位置与大小 */
```

```
23      label = new QLabel(this);
24      label->setGeometry(400, 350, 400, 30);
25      QString str = "当前选择的日期:"
26              + calendarWidget->selectedDate().toString();
27      label->setText(str);
28
29      /* 信号槽连接 */
30      connect(calendarWidget, SIGNAL(selectionChanged()),
31              this, SLOT(calendarWidgetSelectionChanged()));
32      connect(pushButton, SIGNAL(clicked()),
33              this, SLOT(pushButtonClicked()));
34  }
35
36  MainWindow::~MainWindow()
37  {
38  }
39
40  void MainWindow::calendarWidgetSelectionChanged()
41  {
42      /* 当日历点击改变当前选择的期时，更新 Label 的显示内容 */
43      QString str = "当前选择的日期:"
44              + calendarWidget->selectedDate().toString();
45      label->setText(str);
46  }
47
48  void MainWindow::pushButtonClicked()
49  {
50      /* 设置当前选定的日期为系统的 QDate */
51      calendarWidget->setSelectedDate(QDate::currentDate());
52  }
53
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
```

```
11  }
```

7.3.2.3 运行效果

程序编译运行的结果如下。当程序运行后，可以通过点击鼠标或键盘来改变当前选定的日期，label 标签则会改变为当前选定的日期，当点击回到当前日期按钮后，日历会改变当前选定的日期并且把当前选定的日期改变为系统当前的日期。拓展：可以用日历来设置生日，日期提醒等，还可以做成一个华丽的系统日历界面等。

7.3.3 QLCDNumber

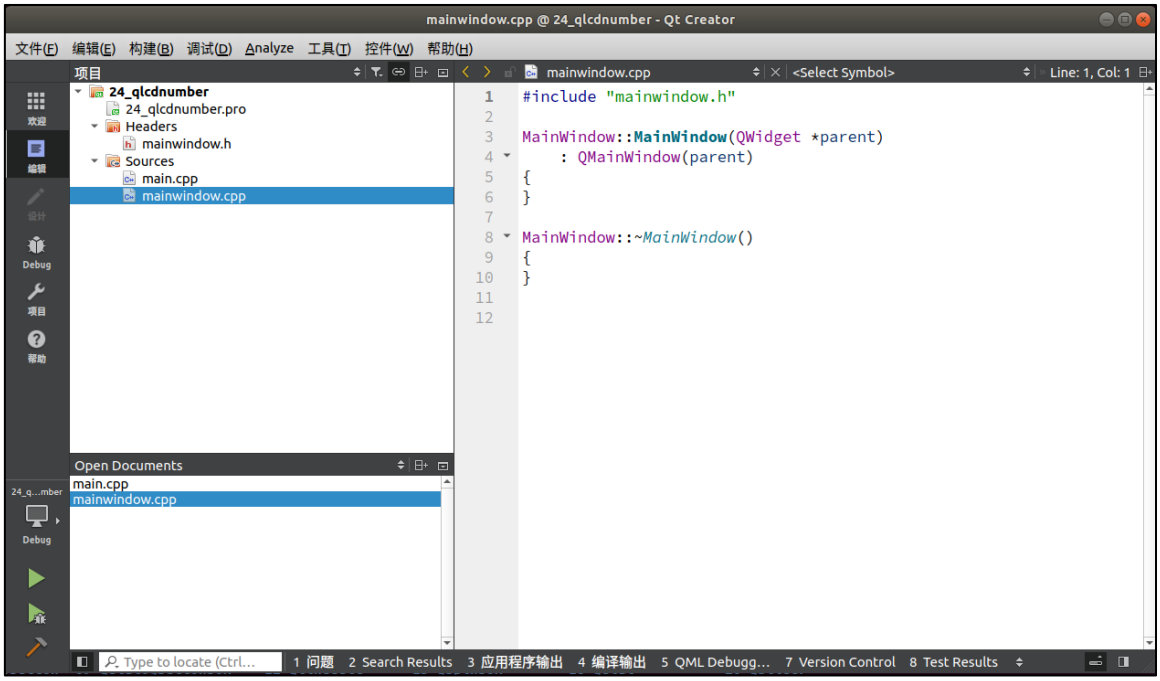
7.3.3.1 控件简介

QLCDNumber 继承 QFrame。QLCDNumber 小部件显示一个类似于 lcd 的数字。QLCDNumber 小部件可以显示任意大小的数字。它可以显示十进制、十六进制、八进制或二进制数字。使用 display()插槽很容易连接到数据源，该插槽被重载以接受五种参数类型中的任何一种。

7.3.3.2 用法示例

例 24_qlcdnumber， LCDNumber 时钟（难度：简单），使用一个 LCDNumber 控件作时钟的显示，一个定时器定时更新 LCDNumber 的时间。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QMainWindow>
6  #include <QLCDNumber>
7  #include <QTimer>
8  #include <QTime>
9
10 class MainWindow : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     MainWindow(QWidget *parent = nullptr);
16     ~MainWindow();
17
18 private:
19     /* 声明 QLCDNumber 对象 */
20     QLCDNumber *lcdNumber;
21
22     /* 声明 QTimer 对象 */
23     QTimer *timer;
24
```

```
25  private slots:
26      /* 槽函数 */
27      void timerTimeOut();
28
29  };
30  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗体的大小与位置 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 实例化与设置显示的位置大小*/
10      lcdNumber = new QLCDNumber(this);
11      lcdNumber->setGeometry(300, 200, 200, 50);
12
13      /* 设置显示的位数 8 位 */
14      lcdNumber->setDigitCount(8);
15      /* 设置样式 */
16      lcdNumber->setSegmentStyle(QLCDNumber::Flat);
17
18      /* 设置 lcd 显示为当前系统时间 */
19      QTime time = QTime::currentTime();
20
21      /* 设置显示的样式 */
22      lcdNumber->display(time.toString("hh:mm:ss"));
23
24      timer = new QTimer(this);
25      /* 设置定时器 1000 毫秒发送一个 timeout()信号 */
26      timer->start(1000);
27
28      /* 信号槽连接 */
29      connect(timer, SIGNAL(timeout()), this,
30              SLOT(timerTimeOut()));
31
32  }
33
34  MainWindow::~MainWindow()
35  {
```

```
36  }
37
38  void MainWindow::timerTimeOut()
39  {
40      /* 当定时器计时 1000 毫秒后，刷新 lcd 显示当前系统时间 */
41      QTime time = QTime::currentTime();
42      /* 设置显示的样式 */
43      lcdNumber->display(time.toString("hh:mm:ss"));
44  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.3.3.3 运行效果

程序编译运行的结果如下。程序运行后，可以看到时间为当前系统的时间， LCDNumber 时钟显示系统时钟的时钟并运行着。

7.3.4 QProgressBar

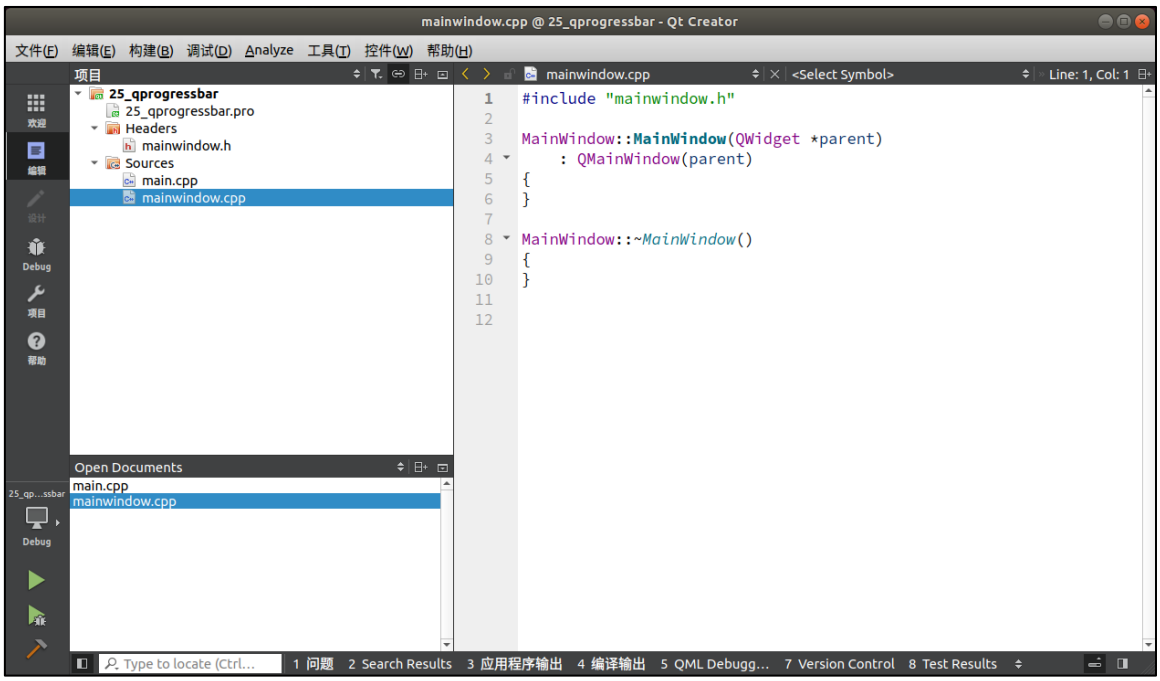
7.3.4.1 控件简介

QProgressBar 继承 QWidget。QProgressBar 小部件提供了一个水平或垂直的进度条。进度条用于向用户显示操作的进度，并向他们确认应用程序仍在运行。

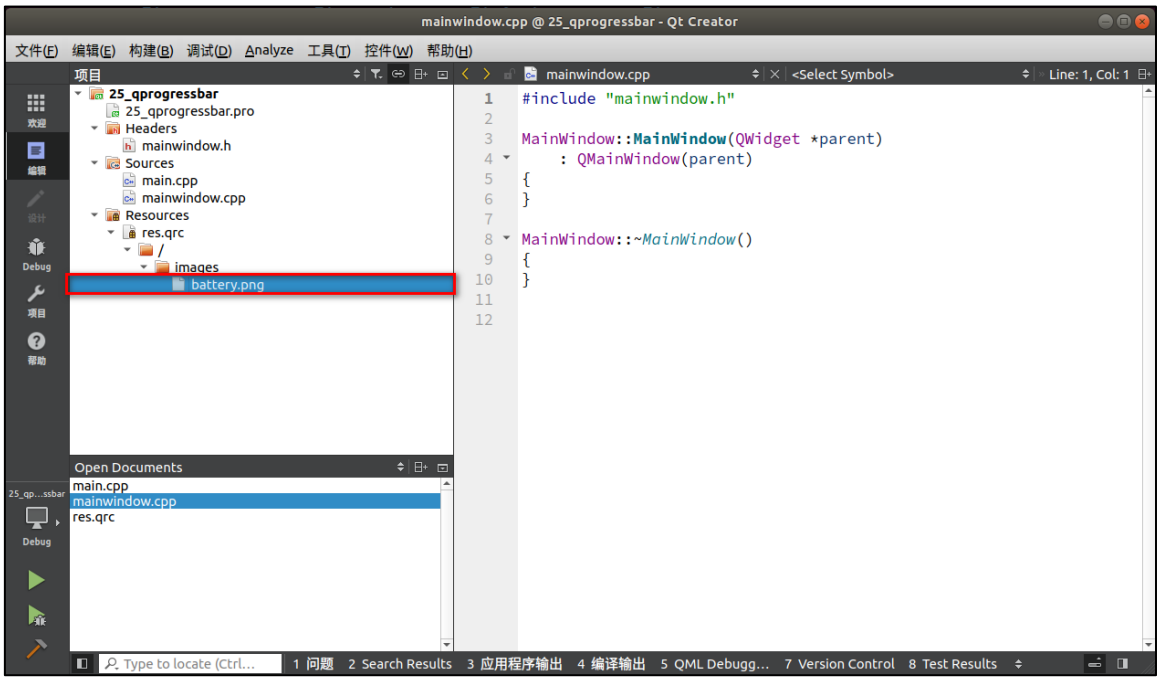
7.3.4.2 用法示例

例 25_qprogressbar，手机电池充电。用一个 QProgressBar 模拟手机电池充电。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



本例已经添加一张电池的背景图资源图片，如果不清楚如何添加资源图片，请参考 7.1.4 小节。添加完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QProgressBar>
```

```
6   #include <QTimer>
7
8   class MainWindow : public QMainWindow
9   {
10      Q_OBJECT
11
12  public:
13      MainWindow(QWidget *parent = nullptr);
14      ~MainWindow();
15
16  private:
17      /* 声明对象 */
18      QProgressBar *progressBar;
19      QTimer *timer;
20
21      /* 用于设置当前 QProgressBar 的值 */
22      int value;
23
24  private slots:
25      /* 槽函数 */
26      void timerTimeOut();
27
28  };
29  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗体位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       progressBar = new QProgressBar(this);
10      progressBar->setGeometry(300, 200, 200, 60);
11
12      /*样式表设置，常用使用 setStyleSheet 来设置样式（实现界面美化的功能），
13       * 具体可参考 styleSheet */
14      progressBar->setStyleSheet (
15                  "QProgressBar{border:8px solid #FFFFFF;"
16                  "height:30;"
17                  "border-image:url(:/images/battery.png);" //背景图片
```

```
18                 "text-align:center;"    // 文字居中
19                 "color:rgb(255,0,255);"
20                 "font:20px;"      // 字体大小为 20px
21                 "border-radius:10px;}"
22                 "QProgressBar::chunk{"
23                 "border-radius:5px;" // 斑马线圆角
24                 "border:1px solid black;" // 黑边，默认无边
25                 "background-color:skyblue;"
26                 "width:10px;margin:1px;}" // 宽度和间距
27                 );
28
29     /* 设置 progressBar 的范围值 */
30     progressBar->setRange(0, 100);
31     /* 初始化 value 为 0 */
32     value = 0;
33     /* 给 progressBar 设置当前值 */
34     progressBar->setValue(value);
35     /* 设置当前文本字符串的显示格式 */
36     progressBar->setFormat("充电中%p%");
37
38     /* 定时器实例化设置每 100ms 发送一个 timeout 信号 */
39     timer = new QTimer(this);
40     timer->start(100);
41
42     /* 信号槽连接 */
43     connect(timer, SIGNAL(timeout()),
44             this, SLOT(timerTimeOut()));
45 }
46
47 MainWindow::~MainWindow()
48 {
49 }
50
51 void MainWindow::timerTimeOut()
52 {
53     /* 定显示器时间到，value 值自加一 */
54     value ++;
55     progressBar->setValue(value);
56     /* 若 value 值大于 100，令 value 再回到 0 */
57     if(value>100)
58         value = 0;
59 }
```

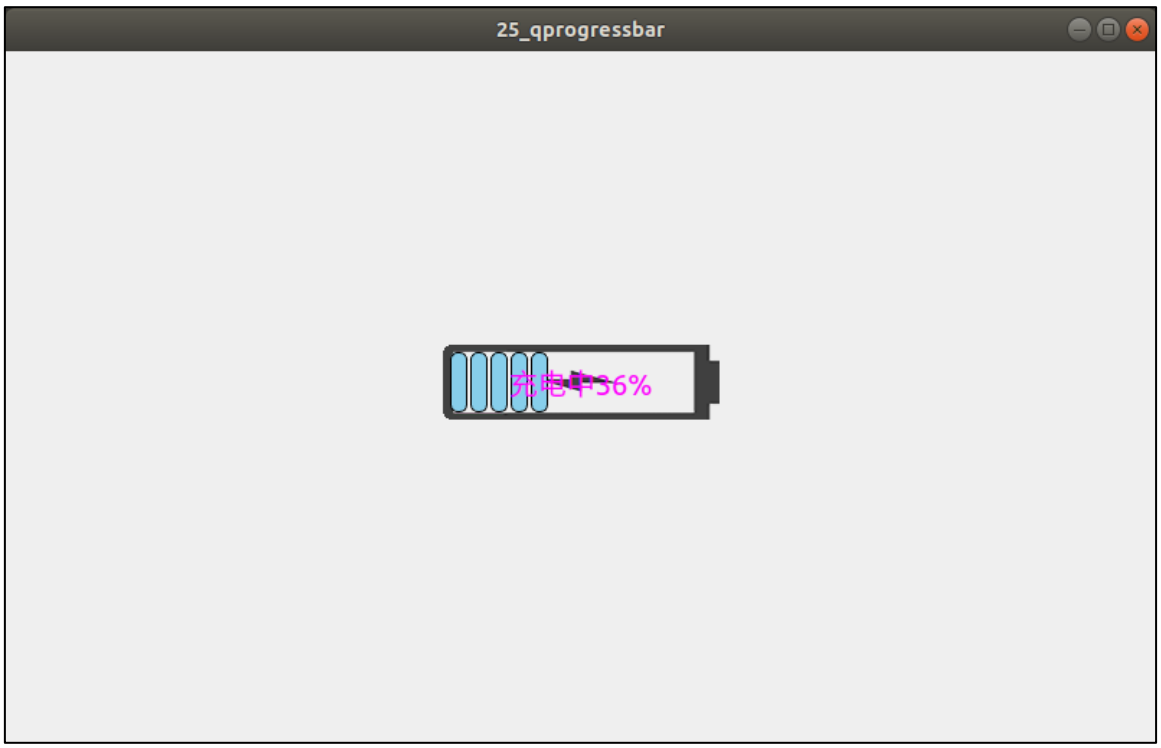
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1  #include "mainwindow.h"
2
3  #include <QApplication>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8      MainWindow w;
9      w.show();
10     return a.exec();
11 }
```

7.3.4.3 运行效果

程序编译运行的结果如下。程序运行后，可以看到在定时器的作用下，电池一直在充电，充到 100%，又重新回到 0%重新充电。QProgressBar 一般用于表示进度，常用于如复制进度，打开、加载进度等。



7.3.5 QFrame

7.3.5.1 控件简介

QFrame 继承 QWidget。QFrame 类是有框架的窗口部件的基类，它绘制框架并且调用一个虚函数 drawContents()来填充这个框架。这个函数是被子类重新实现的。这里至少还有两个有用的函数：drawFrame()和 frameChanged()。

QPopupMenu 使用这个来把菜单“升高”，高于周围屏幕。QProgressBar 有“凹陷”的外观。QLabel 有平坦的外观。这些有框架的窗口部件可以被改变。

QFrame::Shape 这个枚举类型定义了 QFrame 的框架所使用的外形。当前定义的效果有：

- NoFrame - QFrame 不画任何东西
- Box - QFrame 在它的内容周围画一个框
- Panel - QFrame 画一个平板使内容看起来凸起或者凹陷
- WinPanel - 像 Panel，但 QFrame 绘制三维效果的方式和 Microsoft Windows 95（及其它）的一样
- ToolBarPanel - QFrame 调用 QStyle::drawToolBarPanel()
- MenuBarPanel - QFrame 调用 QStyle::drawMenuBarPanel()
- HLine - QFrame 绘制一个水平线，但没有框任何东西（作为分隔是有用的）
- VLine - QFrame 绘制一个竖直线，但没有框任何东西（作为分隔是有用的）
- StyledPanel - QFrame 调用 QStyle::drawPanel()
- PopupPanel - QFrame 调用 QStyle::drawPopupPanel()

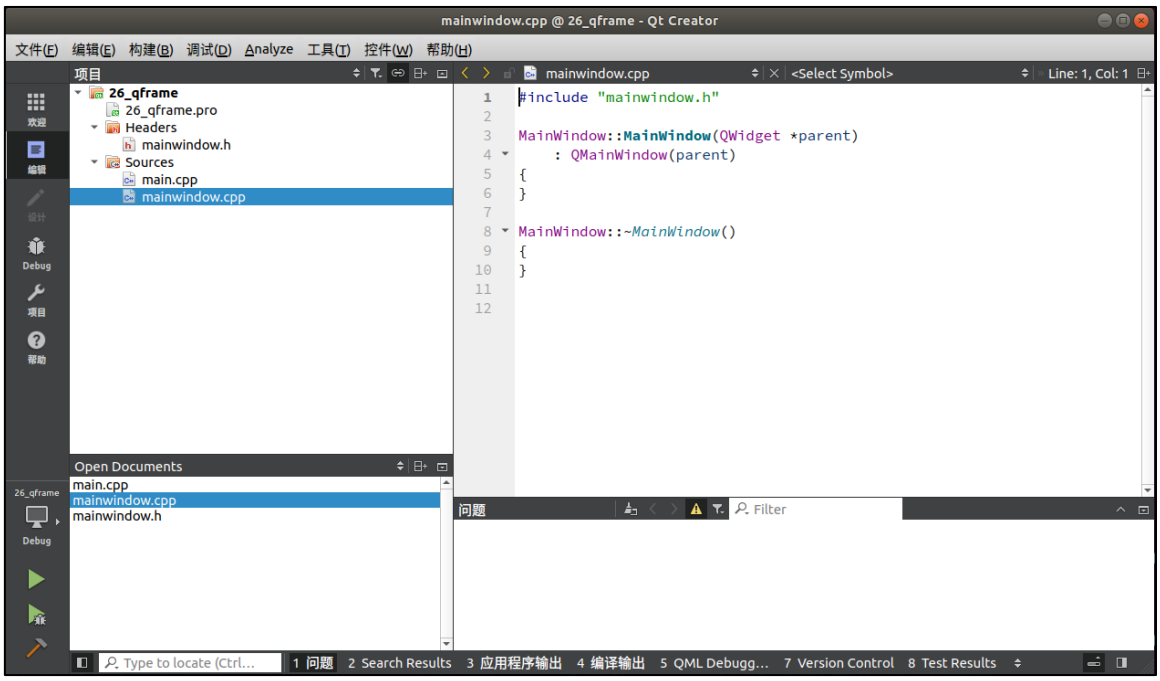
阴影风格有：

- Plain 使用调色板的前景颜色绘制（没有任何三维效果）。
- Raised 使用当前颜色组的亮和暗颜色绘制三维的凸起线。
- Sunken 使用当前颜色组的亮和暗颜色绘制三维的凹陷线。

7.3.5.2 用法示例

例 26_qframe，水平/垂直线（难度：简单），定义两个 QFrame 对象，实例化后设置成一条水平样式，一条垂直样式。实际上 Display Widgets 里的 Horizontal Line/Vertical Line 是 QFrame 已经封装好的控件，也可以通过下面的例子来实现不同的效果。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QFrame>
6
7  class MainWindow : public QMainWindow
8  {
9      Q_OBJECT
10
11 public:
12     MainWindow(QWidget *parent = nullptr);
13     ~MainWindow();
14
15 private:
16     /* 声明对象 */
17     QFrame *hline;
18     QFrame *vline;
19
20 };
21 #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1  #include "mainwindow.h"
2
```

```
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗体的显示位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 实例化 */
10      hline = new QFrame(this);
11      /* 确定起始点，设置长和宽，绘制距形 */
12      hline->setGeometry(QRect(200, 100, 400, 40));
13
14      /* 设置框架样式为 Hline，水平，可设置为其他样式例如 Box，
15       * 由于是样式选择 HLine，所以只显示一条水平直线
16       */
17      hline->setFrameShape(QFrame::HLine);
18      /* 绘制阴影 */
19      hline->setFrameShadow(QFrame::Sunken);
20
21      /* 实例化 */
22      vline = new QFrame(this);
23      /* 确定起始点，设置长和宽，绘制距形 */
24      vline->setGeometry(QRect(300, 100, 2, 200));
25
26      /* 设置框架样式为 Vline，垂直，可设置为其他样式例如 Box，
27       * 由于是样式选择 Vline，所以只显示一条垂直直线
28       */
29      vline->setFrameShape(QFrame::VLine);
30      /* 绘制阴影 */
31      vline->setFrameShadow(QFrame::Sunken);
32  }
33
34  MainWindow::~MainWindow()
35  {
36  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

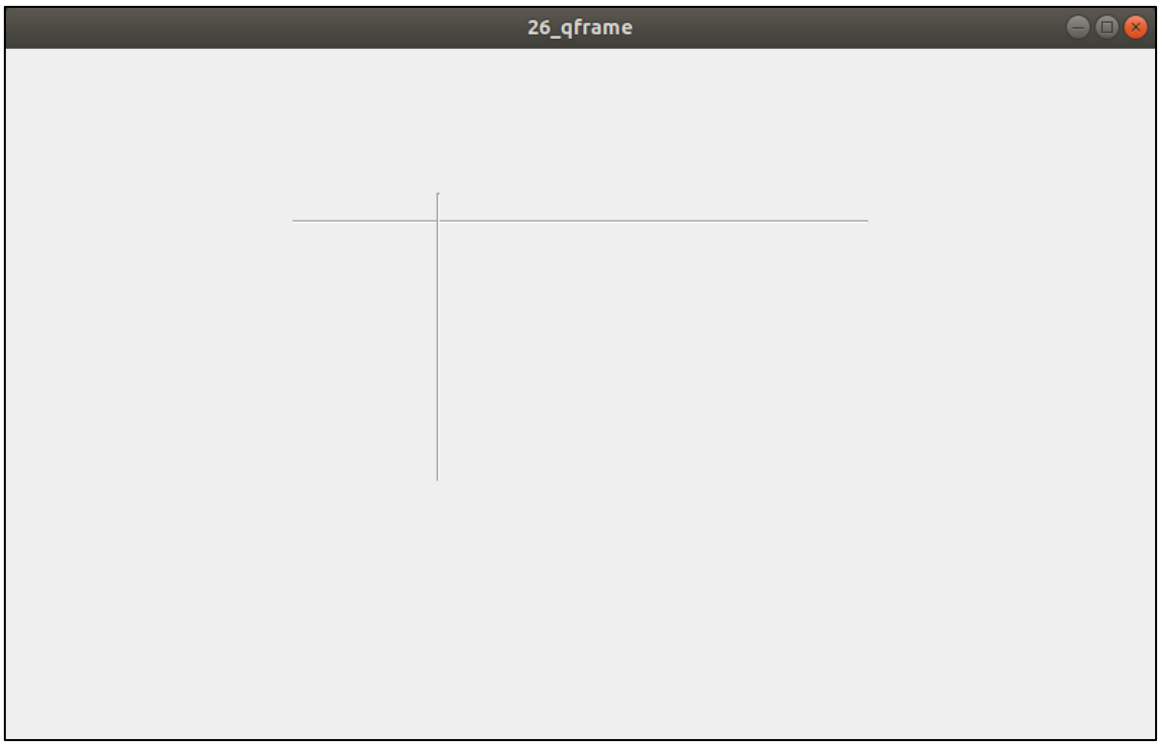
main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
```

```
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.3.5.3 运行效果

程序编译运行的结果如下。在窗口中央出现一条垂直的直线和一条水平的直线。可以用 QFrame 来绘制一些比较好看的边框等，用作美化界面。QFrame 类是所有框架的窗口部件的基类，所以像 QLabel 这种见的控件也可以设置成独特的边框样式。



7.4 显示窗口部件之浏览器

7.4.1 QTextBrowser

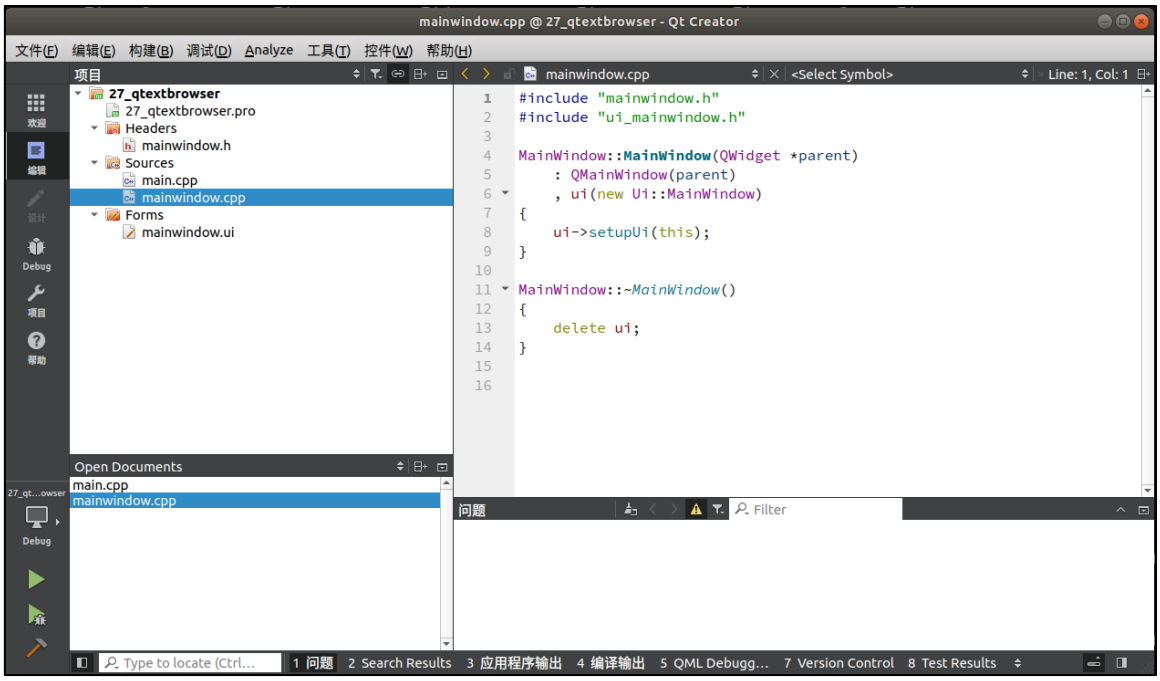
7.4.1.1 控件简介

QTextBrowser 继承 QTextEdit，QTextBrowser 类提供了一个具有超文本导航的文本浏览器。该类扩展了 QTextEdit(在只读模式下)，添加了一些导航功能，以便用户可以跟踪超文本文档中的链接。

7.4.1.2 用法示例

例 27_qtextbrowser，简单的文本浏览器（难度：简单），本例设计一个文本浏览器程序，可以打开并显示 txt、html 等文件。本小节还用到 QAction，菜单栏，学习文件的打开以及处理等。

在新建例程默认继承 QMainWindow 类即可。勾选 mainwindow.ui，因为我们要在工具栏里添加按钮来打开文件等操作。新建项目完成如下。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QTextBrowser>
6
7  QT_BEGIN_NAMESPACE
8  namespace Ui { class MainWindow; }
9  QT_END_NAMESPACE
10
11 class MainWindow : public QMainWindow
12 {
13     Q_OBJECT
14
15 public:
16     MainWindow(QWidget *parent = nullptr);
17     ~MainWindow();
18
19 private:
20     Ui::MainWindow *ui;
21     /* 声明对象 */
22     QTextBrowser *textBrowser;
23     QAction *openAction;
```

```
24
25  private slots:
26      /* 槽函数 */
27      void openActionTriggered();
28  };
29  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2   #include "ui_mainwindow.h"
3   /* 窗口对话框与文本流 */
4   #include <QFileDialog>
5   #include <QTextStream>
6
7   MainWindow::MainWindow(QWidget *parent)
8       : QMainWindow(parent)
9       , ui(new Ui::MainWindow)
10  {
11      ui->setupUi(this);
12      /* 设置主窗体位置与大小 */
13      this->setGeometry(0, 0, 800, 480);
14
15      /* 将窗口标题设置为文本浏览器 */
16      this->setWindowTitle("文本浏览器");
17
18      /* 实例化 */
19      textBrowser = new QTextBrowser(this);
20      /* 将文本浏览器窗口居中 */
21      this->setCentralWidget(textBrowser);
22
23      /* 实例化 */
24      openAction = new QAction("打开",this);
25      /* ui 窗口自带有 menubar(菜单栏)、mainToolbar（工具栏）与
26       * statusbar（状态栏）
27       * menuBar 是 ui 生成工程就有的，所以可以在 menubar 里添加
28       * 我们的 QActiont 等，如果不需要 menubar，可以在 ui 设计
29       * 窗口里，在右则对象里把 menubar 删除，再自己重新定义自己的
30       * 菜单栏
31       */
32      /* 将动作添加到菜单栏 */
33      ui->menubar->addAction(openAction);
```

```
34
35      /* 信号槽连接 */
36      connect(openAction, SIGNAL(triggered()),
37              this, SLOT(openActionTriggered()));
38
39  }
40
41  MainWindow::~MainWindow()
42  {
43      delete ui;
44  }
45
46  void MainWindow::openActionTriggered()
47  {
48      /* 调用系统打开文件窗口，过滤文件名 */
49      QString fileName = QFileDialog::getOpenFileName(
50                  this,tr("打开文件"),"",
51                  tr("Files(*.txt *.cpp *.h *.html *.htm)")
52                  );
53      QFile myFile(fileName);
54      /* 以只读、文本方式打开，若打开失败，则返回 */
55      if(!myFile.open(QIODevice::ReadOnly | QIODevice::Text))
56          return;
57
58      /* 用 QTextStream 对象接收 */
59      QTextStream in (&myFile);
60
61      /* 读取全部数据 */
62      QString myText = in.readAll();
63
64      /* 判断打开文件的后缀，如果是 html 格式的则设置文本浏览器为 html 格式 */
65      if(fileName.endsWith("html") || fileName.endsWith("htm")){
66          textBrowser->setHtml(myText);
67      } else {
68          textBrowser->setPlainText(myText);
69      }
70
71      /* ui 窗口自带有 statusbar（状态栏），设置打开的文件名 */
72      ui->statusbar->showMessage("文件名: " + fileName);
73  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

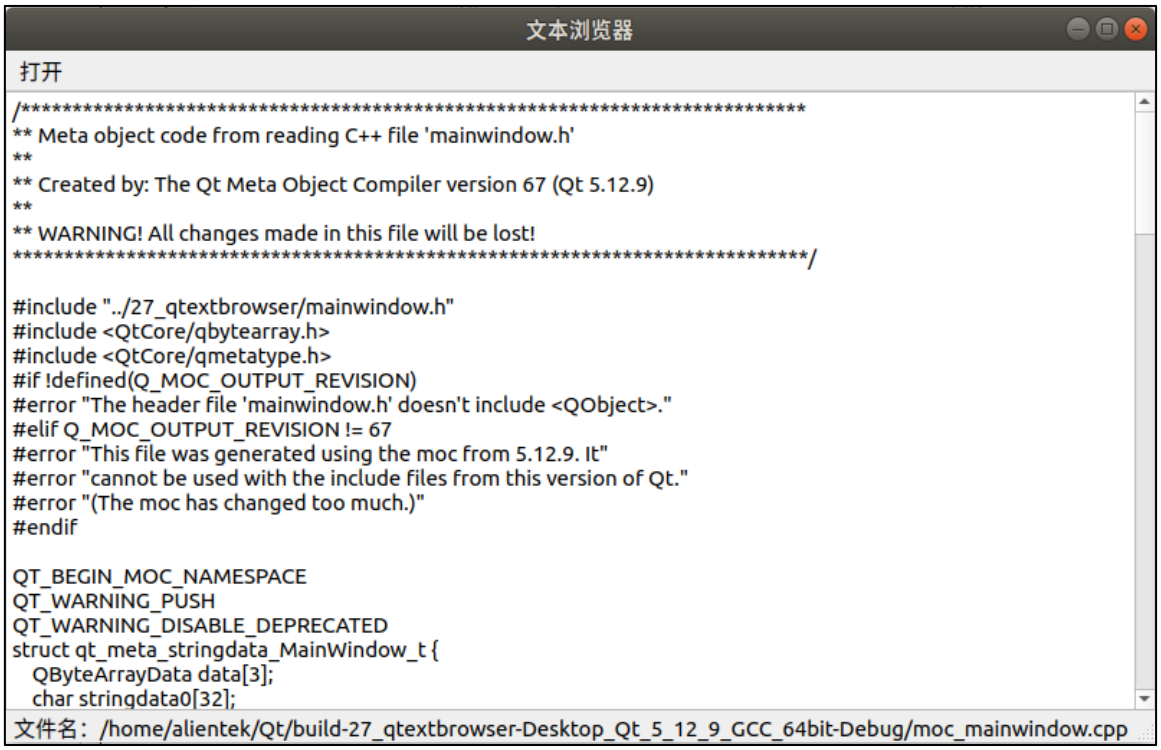
main.cpp 编程后的代码

```
1   #include "mainwindow.h"
```

```
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.4.1.3 运行效果

程序编译运行的结果如下。在菜单栏点击打开后，系统默认是打开的最近打开的位置，选择任意一个可打开的文件，本次打开的是该工程中的 mainwindow.cpp 文件，结果如下图。根据上面的例子可自行拓展打造自己的文本浏览器，例如在菜单栏上加上关闭动作等，在工具栏还可以添加字体颜色，与及背景颜色，以及字体的放大与缩小等，可自行拓展。（备注：经测试 Ubuntu16 的 Qt 程序异常，看不到菜单栏，所以看不到“打开”这个按钮，Ubuntu18 显示正常。）



7.4.2 QGraphicsView

小节前言：

在这里主要是简单介绍 QGraphicsView 这种框架类的介绍与常用接口，实际上这种框架过于复杂，我们也不能在这一小节完全展现，下面就让我们快速去了解 QGraphicsView 图形视图框架类吧！

7.4.2.1 控件简介

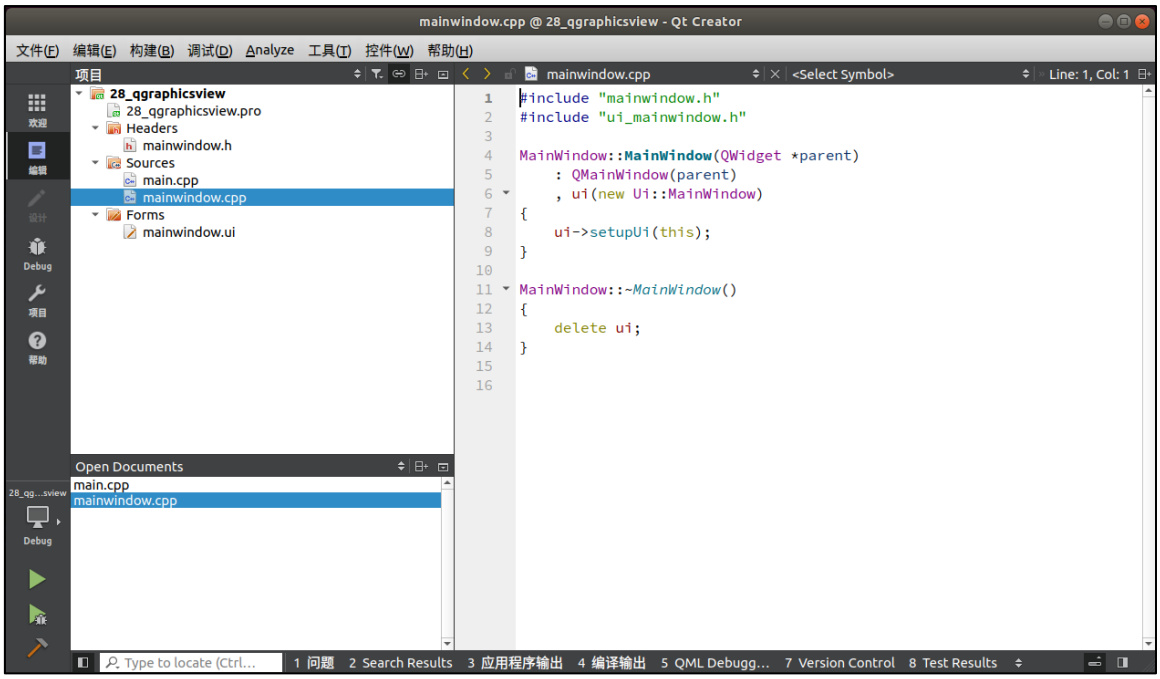
QGraphicsView 继承 QAbstractScrollArea。QGraphicsView 是图形视图框架的一部分，它提供了基于图元的模型/视图编程。QGraphicsView 在可滚动视图中可视化 QGraphicsScene 的内容。要创建带有几何项的场景，请参阅 QGraphicsScene 的文档。

要可视化场景，首先构造一个 QGraphicsView 对象，将要可视化的场景的地址传递给 QGraphicsView 的构造函数。或者，可以调用 setScene()在稍后设置场景。

7.4.2.2 用法示例

例 28_qgraphicsview，简单的图像浏览器（难度：简单），本例设计一个图像浏览器程序，在上一节一的基础上，将它改变为图像浏览器。

在新建例程默认继承 QMainWindow 类即可。勾选 mainwindow.ui，因为我们要在工具栏里添加按钮来打开文件等操作。新建项目完成如下。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QGraphicsView>
6
```

```
7  QT_BEGIN_NAMESPACE
8  namespace Ui { class MainWindow; }
9  QT_END_NAMESPACE
10
11 class MainWindow : public QMainWindow
12 {
13     Q_OBJECT
14
15 public:
16     MainWindow(QWidget *parent = nullptr);
17     ~MainWindow();
18
19 private:
20     Ui::MainWindow *ui;
21     /* 声明对象 */
22     QGraphicsView *graphicsView;
23     QGraphicsScene *graphicsScene;
24     QAction *openAction;
25
26 private slots:
27     /* 槽函数 */
28     void openActionTriggered();
29
30 };
31 #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3  #include <QFileDialog>
4
5  MainWindow::MainWindow(QWidget *parent)
6      : QMainWindow(parent)
7      , ui(new Ui::MainWindow)
8  {
9      ui->setupUi(this);
10     /* 设置主窗体大小 */
11
12     this->setGeometry(0, 0, 800, 480);
13     /* 将窗口标题设置为图像浏览器 */
14     this->setWindowTitle("图像浏览器");
15
16     /* 实例化图形视图对象 */
```

```
17      graphicsView = new QGraphicsView(this);
18      /* 将图像浏览器窗口居中 */
19      this->setCentralWidget(graphicsView);
20
21      /* 实例化场景对象 */
22      graphicsScene = new QGraphicsScene(this);
23
24      /* 在 QGraphicsView 设置场景 */
25      graphicsView->setScene(graphicsScene);
26
27      /* 将动作添加到菜单栏 */
28      openAction = new QAction("打开",this);
29      ui->menubar->addAction(openAction);
30
31      /* 信号槽连接 */
32      connect(openAction, SIGNAL(triggered()),
33              this, SLOT(openActionTriggered()));
34  }
35
36  MainWindow::~MainWindow()
37  {
38      delete ui;
39  }
40
41  void MainWindow::openActionTriggered()
42  {
43      /*调用系统打开文件窗口，设置窗口标题为"打开文件"，过滤文件名*/
44      QString fileName = QFileDialog::getOpenFileName(
45                  this,tr("打开文件"), "",
46                  tr("Files(*.png *.jpg *.bmp)")
47                  );
48      /* 定义 QPixmap 对象，指向 fileName */
49      QPixmap image(fileName);
50      /* 将 image 用 scaled 来重新设置长宽为 graphicsView 的长宽，
51       * 保持纵横比等
52       */
53
54      /* 假若用户没选择文件，则返回 */
55      if(image.isNull())
56          return;
57      image = image.scaled(graphicsView->width(),
58                           graphicsView->height(),
```

```
59                          Qt::KeepAspectRatio,
60                          Qt::FastTransformation
61                          );
62      /* 在添加场景内容前，先清除之前的场景内容 */
63      graphicsScene->clear();
64      /* 添加场景内容 image */
65      graphicsScene->addPixmap(image);
66      /* ui 窗口自带有 statusBar（状态栏），设置打开的文件名 */
67      ui->statusbar->showMessage("文件名: " + fileName);
68  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.4.2.3 运行效果

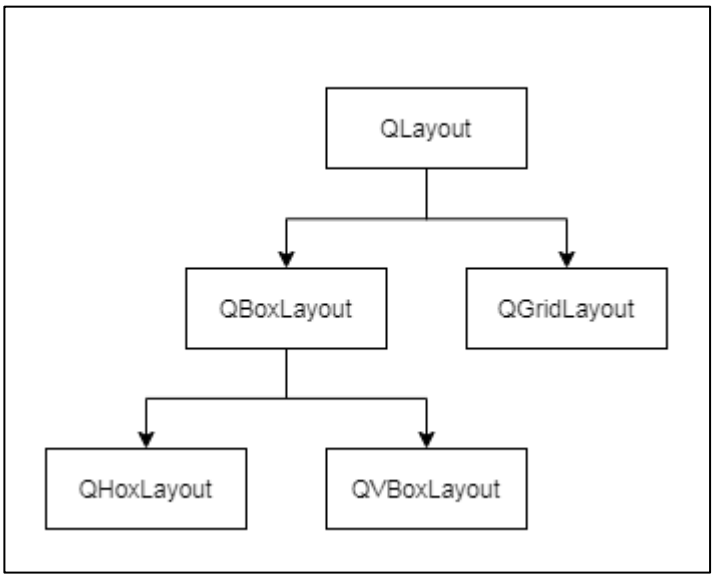
程序编译运行的结果如下。菜单栏点击打开后，系统默认是打开的最近打开的位置，选择任意一个可打开的图片。（备注：本例所展示的图片已经放在工程目录下）。

7.5 布局管理

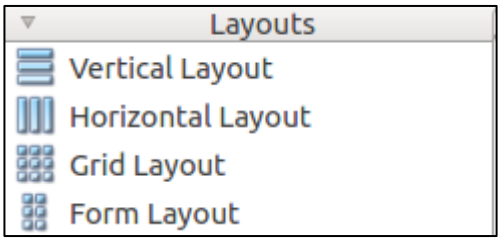
Qt 提供了非常丰富的布局类，基本布局管理类包括：QBoxLayout、QGridLayout、QFormLayout 和 QStackedLayout。这些类都从 QLayout 继承而来，它们都来源于 QObject（而不是 QWidget）。创建更加复杂的布局，可以让它们彼此嵌套完成。

其中 QBoxLayout 提供了水平和垂直的布局管理；QFormLayout 提供了将输入部件和标签成组排列的布局管理；QGridLayout 提供了网格形式的布局管理；QStackedLayout 提供了一组布局后的部件，可以对它们进行分布显示。

它们的继承关系如下图。



下面将学习 Layouts 组里面的 4 种布局，如下图。



各个控件的名称依次解释如下。

（1） Vertiacl Layout:垂直布局

（2） Horizontal Layout:水平布局

（3） Grid Layout:网格布局

（4） Form Layout:表单布局

QBoxLayout 继承 QLayout。QBoxLayout 类提供水平或垂直地排列子部件。QBoxLayout 获取从它的父布局或从 parentWidget()中所获得的空间，将其分成一列框，并使每个托管小部件填充一个框。

QGridLayout 继承 QLayout。QGridLayout 获取可用的空间(通过其父布局或 parentWidget()))，将其分为行和列，并将其管理的每个小部件放入正确的单元格中。由于网格布局管理器中的组件也是会随着窗口拉伸而发生变化的，所以也是需要设置组件之间的比例系数的，与 QBoxLayout 不同的是网格布局管理器还需要分别设置行和列的比例系数。

QFormLayout 继承 QLayout。QFormLayout 类管理输入小部件及其关联标签的表单。QFormLayout 是一个方便的布局类，它以两列的形式布局其子类。左列由标签组成，右列由“字段”小部件(QLineEdit(行编辑器)、QSpinBox(旋转框等))组成。通常使用 setRowWrapPolicy(RowWrapPolicy policy)接口函数设置布局的换行策略进行布局等。

7.5.1 QBoxLayout

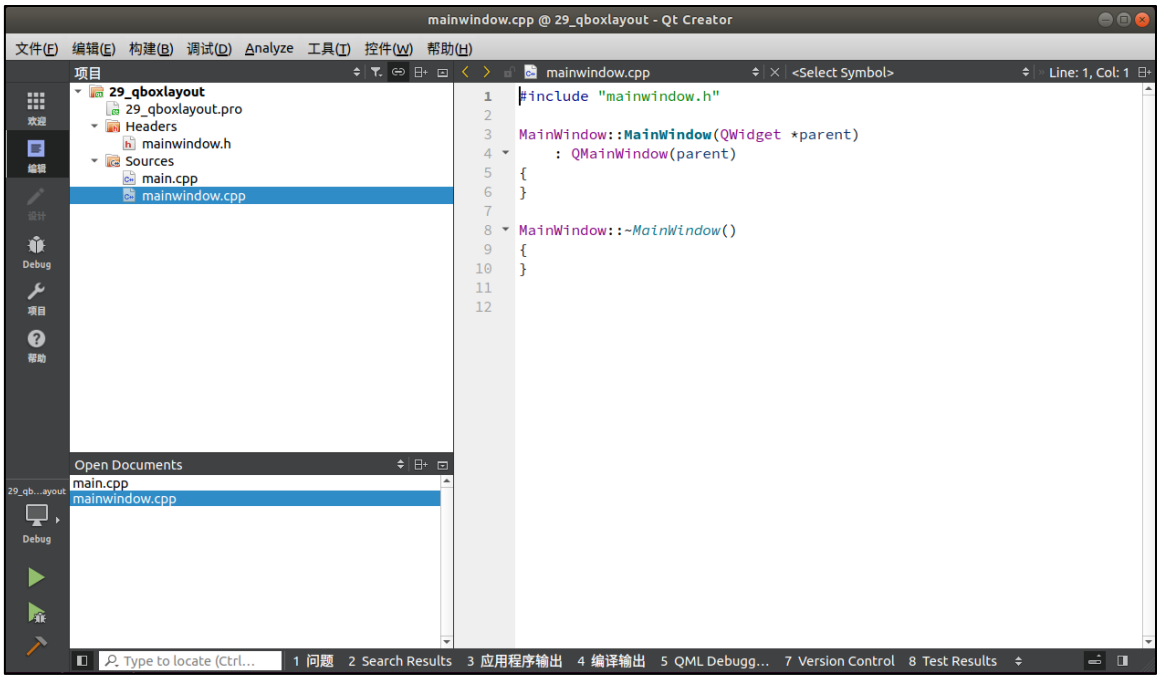
7.5.1.1 控件简介

QBoxLayout 继承 QLayout。QBoxLayout 类提供水平或垂直地排列子部件。QBoxLayout 获取从它的父布局或从 parentWidget()中所获得的空间，将其分成一列框，并使每个托管小部件填充一个框。

7.5.1.2 用法示例

例 29_qboxlayout，垂直或水平布局（难度：简单），使用几个按钮，将他们设置为垂直排布和水平排布，以及设置它们的一些属性。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QHBoxLayout>
6  #include <QVBoxLayout>
7  #include <QPushButton>
8
9  class MainWindow : public QMainWindow
10 {
11     Q_OBJECT
12
13 public:
14     MainWindow(QWidget *parent = nullptr);
15     ~MainWindow();
16
17 private:
18     /* 声明按钮对象数组 */
19     QPushButton *pushButton[6];
20
21     /* 定义两个 widget，用于容纳排布按钮 */
22     QWidget *hWidget;
23     QWidget *vWidget;
```

```
24
25      /* QHBoxLayout 与 QVBoxLayout 对象 */
26      QHBoxLayout *hLayout;
27      QVBoxLayout *vLayout;
28
29  };
30  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2   #include <QList>
3
4   MainWindow::MainWindow(QWidget *parent)
5       : QMainWindow(parent)
6   {
7       /* 设置主窗口的位置与大小 */
8       this->setGeometry(0, 0, 800, 480);
9
10      /* 实例化与设置位置大小 */
11      hWidget = new QWidget(this);
12      hWidget->setGeometry(0, 0, 800, 240);
13
14      vWidget = new QWidget(this);
15      vWidget->setGeometry(0, 240, 800, 240);
16
17      hLayout = new QHBoxLayout();
18      vLayout = new QVBoxLayout();
19
20      /* QList<T>是 Qt 的一种泛型容器类。
21       * 它以链表方式存储一组值，
22       * 并能对这组数据进行快速索引
23       */
24      QList <QString>list;
25      /* 将字符串值插入 list */
26      list<<"one"<<"two"<<"three"<<"four"<<"five"<<"six";
27
28      /* 用一个循环实例化 6 个按钮 */
29      for(int i = 0; i < 6; i++){
30          pushButton[i] = new QPushButton();
31          pushButton[i]->setText(list[i]);
32          if(i < 3) {
33              /* 将按钮添加至 hLayout 中 */
34              hLayout->addWidget(pushButton[i]);
```

```
35          } else {
36              /* 将按钮添加至 vLayout 中 */
37              vLayout->addWidget(pushButton[i]);
38          }
39      }
40      /* 设置间隔为 50 */
41      hLayout->setSpacing(50);
42
43      /* hWidget 与 vWidget 的布局设置为 hLayout/vLayout */
44      hWidget->setLayout(hLayout);
45      vWidget->setLayout(vLayout);
46  }
47
48  MainWindow::~MainWindow()
49  {
50  }
```

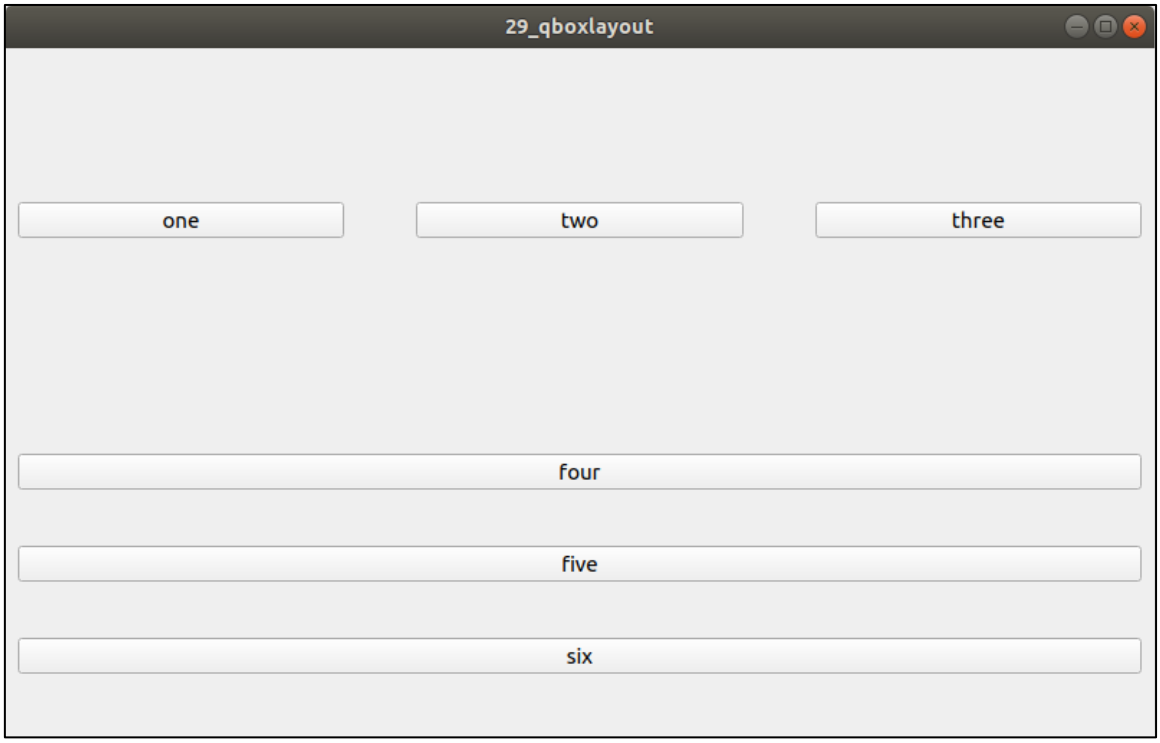
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.5.1.3 运行效果

程序编译运行的结果如下。可以看到在 hWidget 中添加了 3 个水平排布的按钮，在 vWidget 中添加了 3 个垂直排布的按钮。



7.5.2 QGridLayout

7.5.2.1 控件简介

QGridLayout 类提供了布局管理器里的一种以网格（二维）的方式管理界面组件，以按钮组件为例，它们所对应网格的坐标下表，与二维数组类似。

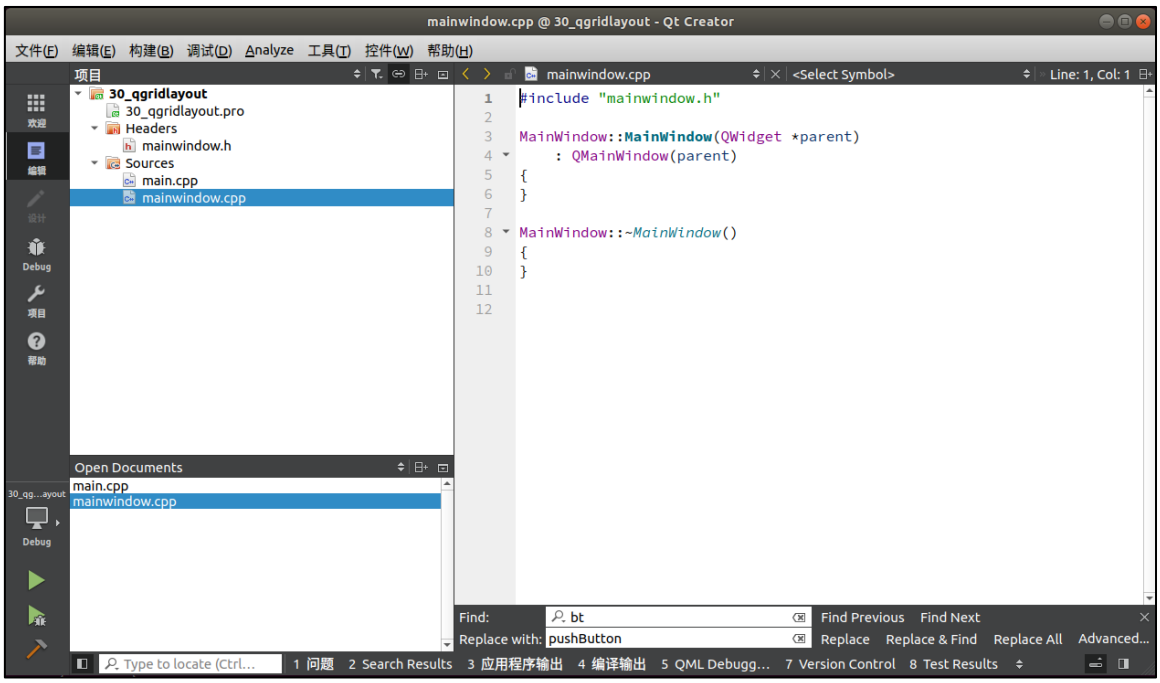
0，0（组件一）	0，1（组件二）
1，0（组件三）	1，1（组件四）

QGridLayout 继承 QLayout。QGridLayout 获取可用的空间(通过其父布局或 parentWidget())，将其分为行和列，并将其管理的每个小部件放入正确的单元格中。由于网格布局管理器中的组件也是会随着窗口拉伸而发生变化的，所以也是需要设置组件之间的比例系数的，与 QBoxLayout 不同的是网格布局管理器还需要分别设置行和列的比例系数。

7.5.2.2 用法示例

例 30_qgridlayout，网格布局（难度：简单），使用几个按钮，将他们设置为网格布局，同时设置它们的行、列比例系数（拉伸因子），以及设置它们的一些属性。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QGridLayout>
6  #include <QPushButton>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15 private:
16
17     /* 声明 widget 窗口部件，用于容纳下面 4 个 pushButton 按钮 */
18     QWidget *gWidget;
19
20     /* 声明 QGridLayout 对象 */
21     QGridLayout *gridLayout;
22
23     /* 声明 pushButton 按钮数组 */
24     QPushButton *pushButton[4];
25
```

```
26  };
27  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 实例化 */
10      gWidget = new QWidget(this);
11      /* 设置 gWidget 居中央 */
12      this->setCentralWidget(gWidget);
13
14      gridLayout = new QGridLayout();
15      /* QList 链表，字符串类型 */
16      QList <QString> list;
17      list<<"按钮 1"<<"按钮 2"<<"按钮 3"<<"按钮 4";
18      for (int i = 0; i < 4; i++){
19          pushButton[i] = new QPushButton();
20          pushButton[i]->setText(list[i]);
21          /* 设置最小宽度与高度 */
22          pushButton[i]->setMinimumSize(100, 30);
23          /* 自动调整按钮的大小 */
24          pushButton[i]->setSizePolicy(
25                      QSizePolicy::Expanding,
26                      QSizePolicy::Expanding
27                      );
28          switch (i) {
29          case 0:
30              /* 将 pushButton[0]添加至网格的坐标(0,0),下同 */
31              gridLayout->addWidget(pushButton[i], 0, 0);
32              break;
33          case 1:
34              gridLayout->addWidget(pushButton[i], 0, 1);
35              break;
36          case 2:
37              gridLayout->addWidget(pushButton[i], 1, 0);
38              break;
39          case 3:
```

```
40              gridLayout->addWidget(pushButton[i], 1, 1);
41              break;
42          default:
43              break;
44          }
45      }
46      /* 设置第 0 行与第 1 行的行比例系数 */
47      gridLayout->setRowStretch(0, 2);
48      gridLayout->setRowStretch(1, 3);
49
50      /* 设置第 0 列与第 1 列的列比例系数 */
51      gridLayout->setColumnStretch(0, 1);
52      gridLayout->setColumnStretch(1, 3);
53
54      /* 将 gridLayout 设置到 gWidget */
55      gWidget->setLayout(gridLayout);
56  }
57
58  MainWindow::~MainWindow()
59  {
60  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.5.2.3 运行效果

程序编译运行的结果如下。可以看到在 gWidget 中添加了 4 个按钮，因为设置了行、列的系数比（拉伸因子），所以看到的按钮是按系数比的比例显示。

7.5.3 QFormLayout

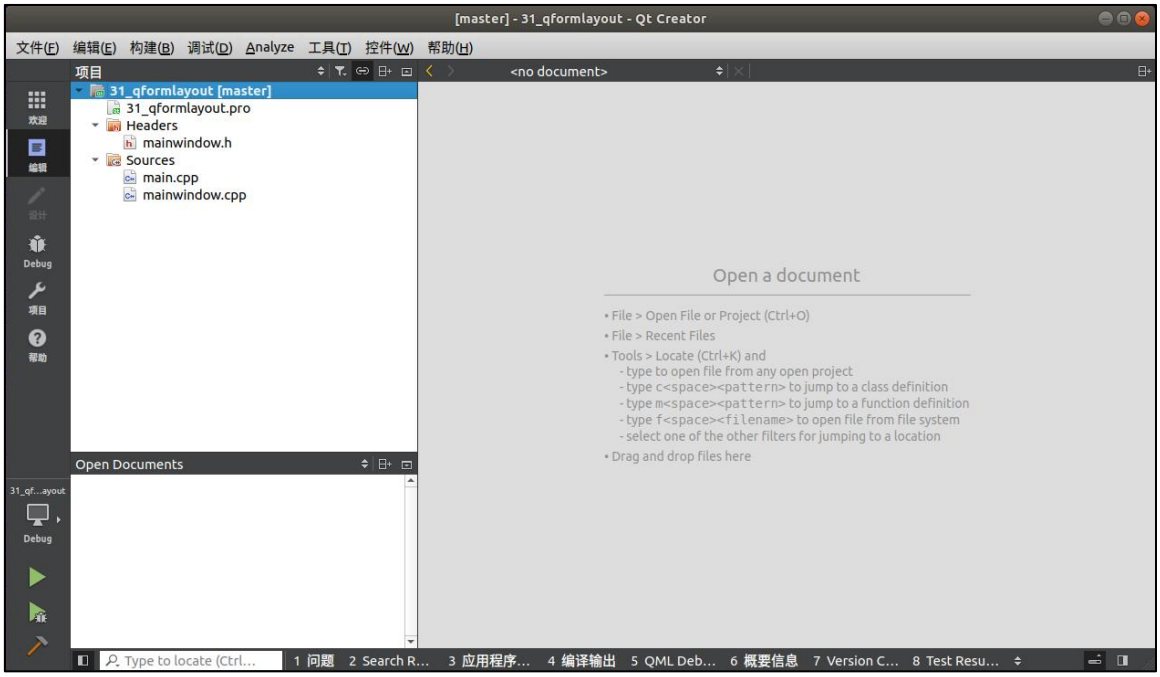
7.5.3.1 控件简介

QFormLayout 继承 QLayout。QFormLayout 类管理输入小部件及其关联标签的表单。QFormLayout 是一个方便的布局类，它以两列的形式布局其子类。左列由标签组成，右列由“字段”小部件(QLineEdit(行编辑器)、QSpinBox(旋转框等))组成。通常使用 setRowWrapPolicy(RowWrapPolicy policy)接口函数设置布局的换行策略进行布局等。

7.5.3.2 用法示例

例 31_qformlayout，表单布局（难度：简单），将使用 addRow(const QString &labelText, QWidget *field)来创建一个带有给定文本的 QLabel 及 QWidget 小部件，并且它们是伙伴关系。简单的展示表单布局的使用。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QFormLayout>
6  #include <QLineEdit>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15 private:
16     /* widget 对象 */
17     QWidget *fWidget;
18
19     /* 用于输入用户名 */
20     QLineEdit *userLineEdit;
21
22     /* 用于输入密码 */
23     QLineEdit *passwordLineEdit;
24
25     /* 声明 QFormLayout 对象 */
```

```
26      QFormLayout *formLayout;
27  };
28  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 实例化及设置位置与大小，下同 */
10      fWidget = new QWidget(this);
11      fWidget->setGeometry(250, 100, 300, 200);
12
13      userLineEdit = new QLineEdit();
14      passwordLineEdit = new QLineEdit();
15
16      formLayout = new QFormLayout();
17
18      /* 添加行 */
19      formLayout->addRow("用户名: ", userLineEdit);
20      formLayout->addRow("密码    : ", passwordLineEdit);
21
22      /* 设置水平垂直间距 */
23      formLayout->setSpacing(10);
24
25      /* 设置布局外框的宽度 */
26      formLayout->setMargin(20);
27
28      /* 将 formLayout 布局到 fWidget */
29      fWidget->setLayout(formLayout);
30  }
31
32  MainWindow::~MainWindow()
33  {
34  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
```

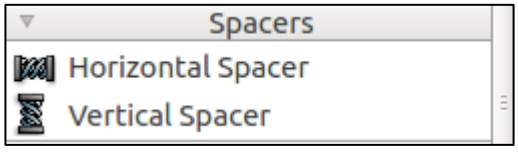
```
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.5.3.3 运行效果

程序编译运行的结果如下。可以看到在 fWidget 中添加了两行，同时设置了它们的间隔，与距边框的宽度。与 QGirdLayout 布局比较，QFomLayout 布局比较适用于行与列比较少的布局格局。如果是多行多列的布局，应该使用 QGirdLayout 布局。

7.6 空间间隔

空间间隔组（Spacers），如下图所示



（1） Horizontal Spacer:水平间隔

（2） Vertical Spacer:垂直间隔

QSpacerItem 继承 QLayoutItem。QSpacerItem 类在布局中提供空白(空间间隔)。所以 QSpacerItem 是在布局中使用的。它包含 Horizontal Spacer（水平间隔）与 Vertical Spacer（垂直间隔）。

7.6.1 QSpacerItem

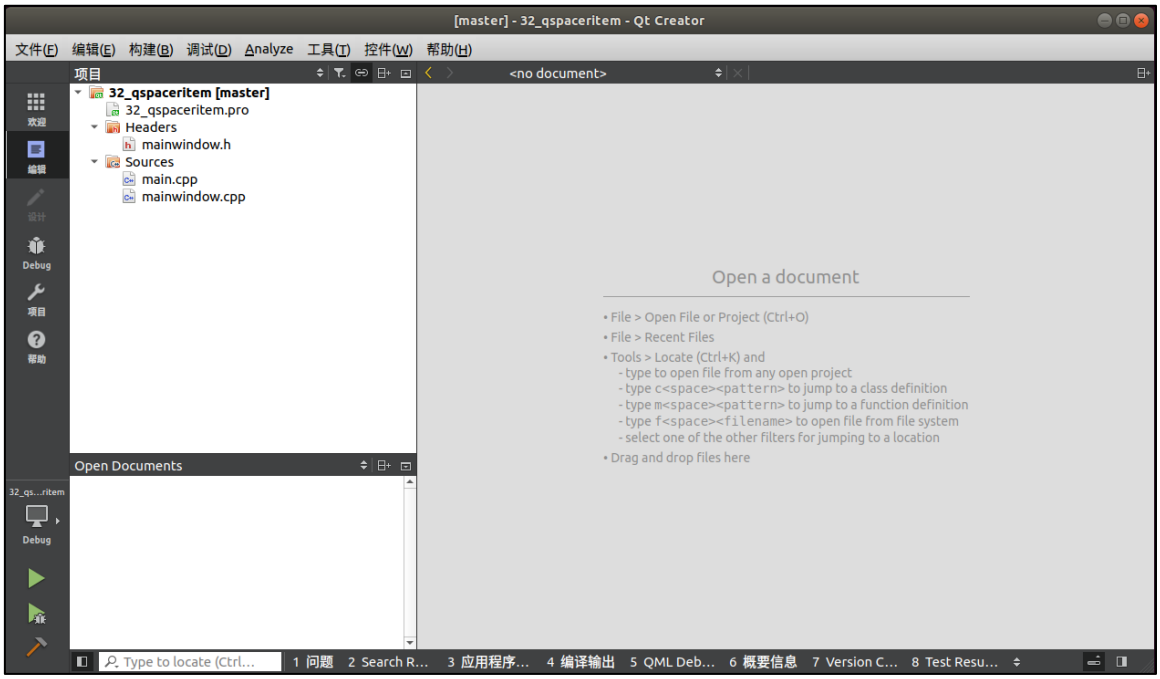
7.6.1.1 控件简介

QSpacerItem 继承 QLayoutItem。QSpacerItem 类在布局中提供空白(空间间隔)。所以 QSpacerItem 是在布局中使用的。

7.6.1.2 用法示例

例 32_qspaceritem，空间间隔（难度：一般），使用 4 个按钮，在垂直布局添加垂直间隔与按钮 1，在水平布局添加按钮 2~4 与水平间隔。简单的展示空间间隔布局的使用方法。在程序运行结果分析了空间间隔部分。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
```

```
4   #include <QMainWindow>
5   #include <QPushButton>
6   #include <QSpacerItem>
7   #include <QBoxLayout>
8
9   class MainWindow : public QMainWindow
10  {
11      Q_OBJECT
12
13  public:
14      MainWindow(QWidget *parent = nullptr);
15      ~MainWindow();
16
17  private:
18      /* 按钮对象数组 */
19      QPushButton *bt[4];
20      /* 垂直间隔 */
21      QSpacerItem *vSpacer;
22      /* 水平间隔 */
23      QSpacerItem *hSpacer;
24      /* 声明一个 widget 用来存放布局的内容 */
25      QWidget *widget;
26      /* 主布局对象 */
27      QHBoxLayout *mainLayout;
28      /* 垂直布局对象 */
29      QVBoxLayout *vBoxLayout;
30      /* 水平布局对象 */
31      QHBoxLayout *hBoxLayout;
32
33  };
34  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗体显示位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       widget = new QWidget(this);
10      /* 居中 widget */
```

```
11      this->setCentralWidget(widget);
12
13      /* 实例化对象 */
14      vSpacer  =  new QSpacerItem(10, 10,
15                              QSizePolicy::Minimum,
16                              QSizePolicy::Expanding
17                              );
18      hSpacer  =  new QSpacerItem(10, 10,
19                              QSizePolicy::Expanding,
20                              QSizePolicy::Minimum
21                              );
22
23      vBoxLayout = new QVBoxLayout();
24      hBoxLayout = new QHBoxLayout();
25      mainLayout = new QHBoxLayout();
26
27      /* 在 vBoxLayout 添加垂直间隔 */
28      vBoxLayout->addSpacerItem(vSpacer);
29
30      QList <QString>list;
31      /* 将字符串值插入 list */
32      list<<"按钮 1"<<"按钮 2"<<"按钮 3"<<"按钮 4";
33      /* 用一个循环实例化 4 个按钮 */
34      for(int i = 0; i < 4 ; i++){
35          bt[i] = new QPushButton();
36          bt[i]->setText(list[i]);
37          if (i == 0){
38              /* 按钮 1,设置为 100*100 */
39              bt[i]->setFixedSize(100, 100);
40              /* 在 vBoxLayout 添加按钮 1 */
41              vBoxLayout->addWidget(bt[i]);
42          } else {
43              /* 按钮 2～4,设置为 60*60 */
44              bt[i]->setFixedSize(60, 60);
45              /* 在 hBoxLayout 添加按钮 2～4 */
46              hBoxLayout->addWidget(bt[i]);
47          }
48      }
49      /* 在 hBoxLayout 添加水平间隔 */
50      hBoxLayout->addSpacerItem(hSpacer);
51
52      /* 在主布局里添加垂直布局 */
53      mainLayout->addLayout(vBoxLayout);
```

```
54      /* 在主布局里添加水平布局 */
55      mainLayout->addLayout(hBoxLayout);
56
57      /* 设置部件间距 */
58      mainLayout->setSpacing(50);
59      /* 将主布局设置为 widget 的布局 */
60      widget->setLayout(mainLayout);
61
62  }
63
64  MainWindow::~MainWindow()
65  {
66  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

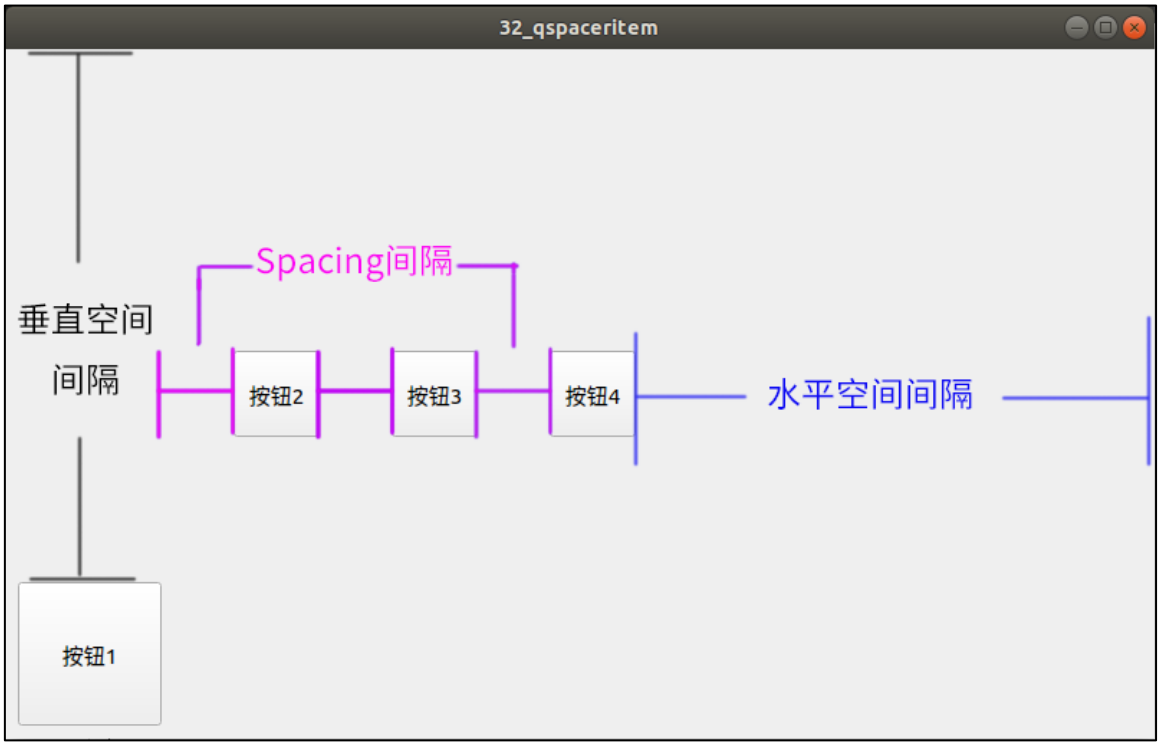
main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.6.1.3 运行效果

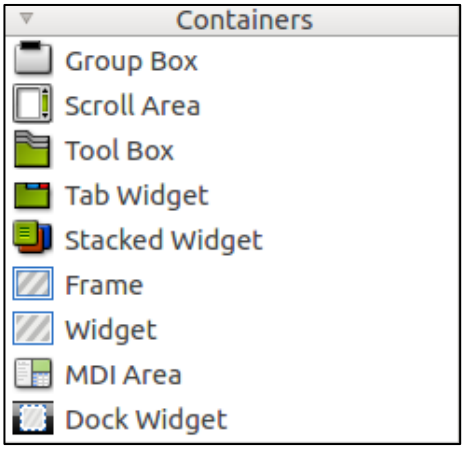
程序编译运行的结果如下，在垂直布局里添加了垂直空间间隔与按钮 1，在水平布局里添加了按钮 2~4 与水平空间间隔。

图解如下：



7.7 容器

容器（Containers）



各个控件的名称依次解释如下。

（1） Group Box:组框

（2） Scroll Area:滚动区域

（3） Tool Box:工具箱

（4） Tab Widget:标签小部件

（5） Stacked WIdget:堆叠小部件

（6） Frame:帧

（7） Widget:小部件

（8） MDI Area:MDI 区域

（9） Dock Widget:停靠窗体部件

各种容器的解释如下：

QGroupBox 继承 QWidget。QGroupBox 为构建分组框提供了支持。分组框通常带有一个边框和一个标题栏，作为容器部件来使用，在其中可以布置各种窗口部件。布局时可用作一组控件的容器，但是需要注意的是，内部通常使用布局控件（如 QBoxLayout）进行布局。组框还提供键盘快捷方式，键盘快捷方式将键盘焦点移动到组框的一个子部件。

QScrollArea 继承 QAbstractScrollArea。滚动区域用于在框架中显示子部件的内容。如果小部件超过框架的大小，视图就会出现滚动条，以便可以查看子小部件的整个区域。

QToolBox 继承 QFrame。QToolBox 类提供了一列选项卡小部件项。工具箱是一个小部件，它显示一列选项卡在另一列的上面，当前项显示在当前选项卡的下面。每个选项卡在选项卡列中都有一个索引位置。选项卡的项是 QWidget。

QTabWidget 继承 QWidget。abWidget 类提供了一组选项卡（多页面）小部件。QTabWidget 主要是用来分页显示的，每一页一个界面，众多界面公用一块区域，节省了界面大小，很方便的为用户显示更多的信息。

QStackedWidget 继承 QFrame。QStackedWidget 类提供了一个小部件堆栈，其中一次只能看到一个小部件，与 QQ 的设置面板类似。QStackedWidget 可用于创建类似于 QTabWidget 提供的用户界面。它是构建在 QStackedLayout 类之上的一个方便的布局小部件。常与 QListWidget 搭配使用，效果如下图，左边的是 QListWidget 列表，右边的是 QStackedWidget。他们一般与

信号槽连接，通过点击左边的 QListWidget 列表，使用信号槽连接后，就可以让右边的 QStackedWidget 显示不同的内容，每次显示一个 widget 小部件。

QWidget 类是所有用户界面对象的基类（如 QLabel 类继承于 QFrame 类，而 QFrame 类又继承于 QWidget 类）。Widget 是用户界面的基本单元：它从窗口系统接收鼠标，键盘和其他事件，并在屏幕上绘制自己。每个 Widget 都是矩形的，它们按照 Z-order 进行排序。 注：Z-order 是重叠二维对象的顺序，例如堆叠窗口管理器中的窗口。典型的 GUI 的特征之一是窗口可能重叠，使得一个窗口隐藏另一个窗口的一部分或全部。当两个窗口重叠时，它们的 Z 顺序确定哪个窗口出现在另一个窗口的顶部。理解：术语"z-order"指沿着 z 轴物体的顺序。三维坐标轴中 x 横轴，y 数轴，z 上下轴。可以将 gui 窗口视为平行与显示平面的一系列平面。因此，窗口沿着 z 轴堆叠。所以 z-order 指定了窗口的前后顺序。就像您桌面上的一叠纸一样，每张纸是一个窗口，桌面是您的屏幕，最上面的窗口 z 值最高。QWidget 不是一个抽象类，它可以用作其他 Widget 的容器，并很容易作为子类来创建定制 Widget。它经常用于创建、放置和容纳其他的 Widget 窗口。由上可知，QWidget 一般用于容纳其他 Widget 窗口，其属性和方法相当的多，对于初学者，我们通常只用它来作可以容纳其他窗口的容器，还会用来接收鼠标，键盘和其他事件等。

QMdiArea 继承 QAbstractScrollArea。QMdiArea 小部件提供一个显示 MDI 窗口的区域。QMdiArea 的功能本质上类似于 MDI 窗口的窗口管理器。大多数复杂的程序，都使用 MDI 框架，在 Qt designer 中可以直接将控件 MDI Area 拖入使用。

QDockWidget 继承 QWidget。QDockWidget 类提供了一个小部件，可以停靠在 QMainWindow 内，也可以作为桌面的顶级窗口浮动。QDockWidget 提供了停靠部件的概念，也称为工具面板或实用程序窗口。停靠窗口是放置在 QMainWindow 中央窗口附近的停靠窗口部件区域中的辅助窗口。停靠窗口可以被移动到当前区域内，移动到新的区域，并由终端用户浮动(例如，不停靠)。QDockWidget API 允许程序员限制 dock widget 的移动、浮动和关闭能力，以及它们可以放置的区域。QDockWidget 的初始停靠区域有 Qt.BottomDockWidgetArea（底部停靠）、Qt.LeftDockWidgetArea（左边停靠、Qt.RightDockWidgetArea（右边停靠）、Qt.TopDockWidgetArea（顶部停靠）和 Qt.NoDockWidgetArea（不显示 Widget）。

在前面某些小节里已经有使用过本小节的控件，例如 QWidget 小部件。下面将上面列出的控件进行进一步的解释与运用。

7.7.1 QGroupBox

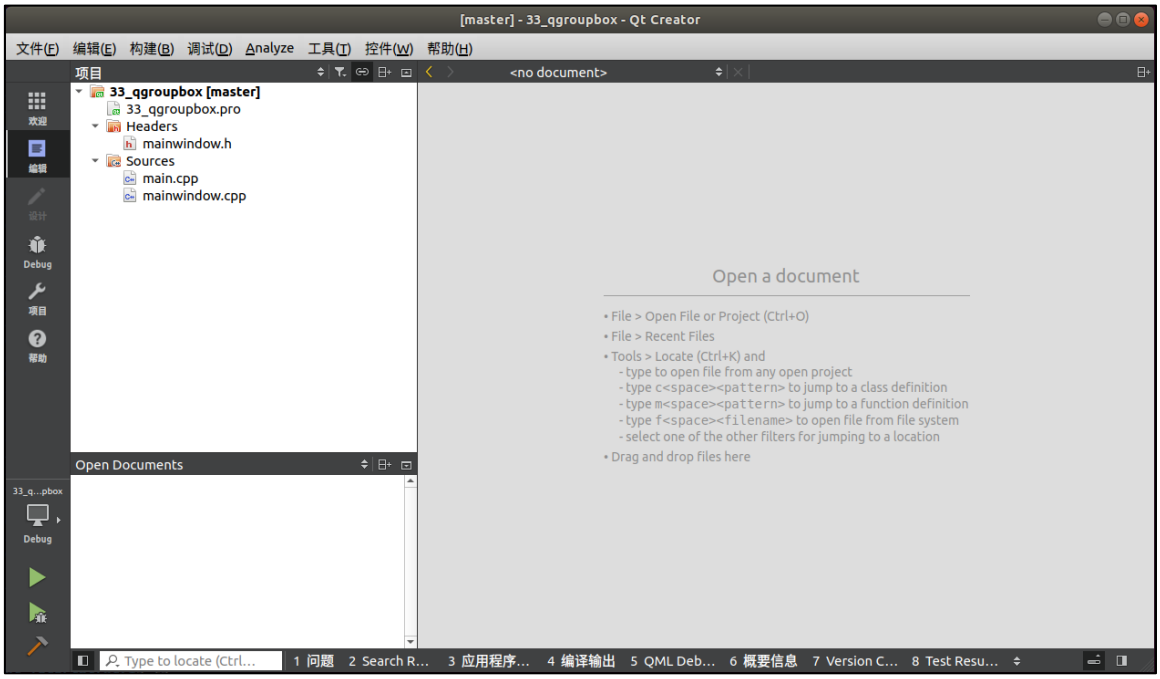
7.7.1.1 控件简介

QGroupBox 小部件提供一个带有标题的组框框架。一般与一组或者是同类型的部件一起使用。

7.7.1.2 用法示例

例 33_qgroupbox，组框示例（难度：简单），使用 3 个 QRadioButton 单选框按钮，与 QVBoxLayout（垂直布局）来展示组框的基本使用。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QRadioButton>
6  #include <QGroupBox>
7  #include <QVBoxLayout>
8
9  class MainWindow : public QMainWindow
10 {
11     Q_OBJECT
12
13 public:
14     MainWindow(QWidget *parent = nullptr);
15     ~MainWindow();
16
17 private:
18     /* 声明对象 */
```

```
19      QGroupBox *groupBox;
20      QVBoxLayout  *vBoxLayout;
21      QRadioButton *radioButton[3];
22  };
23  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2   #include <QList>
3
4   MainWindow::MainWindow(QWidget *parent)
5       : QMainWindow(parent)
6   {
7       /* 设置主窗体位置与大小 */
8       this->setGeometry(0, 0, 800, 480);
9       /* 以标题为“QGroupBox 示例”实例化 groupBox 对象 */
10      groupBox = new QGroupBox(tr("QGroupBox 示例"), this);
11      groupBox->setGeometry(300, 100, 300, 200);
12
13      vBoxLayout = new QVBoxLayout();
14
15      /* 字符串链表 */
16      QList <QString>list;
17      list<<"选项一"<<"选项二"<<"选项三";
18      for(int i = 0; i < 3; i++){
19          radioButton[i] = new QRadioButton();
20          radioButton[i]->setText(list[i]);
21          /* 在 vBoxLayout 添加 radioButton */
22          vBoxLayout->addWidget(radioButton[i]);
23      }
24      /* 添加一个伸缩量 1 */
25      vBoxLayout->addStretch(1);
26      /* vBoxLayout 布局设置为 groupBox 布局 */
27      groupBox->setLayout(vBoxLayout);
28  }
29
30  MainWindow::~MainWindow()
31  {
32  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
```

```
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.7.1.3 运行效果

程序编译运行的结果如下，可以看到 radioButton 有规则的排布在 groupBox 组框里面。

7.7.2 QScrollArea

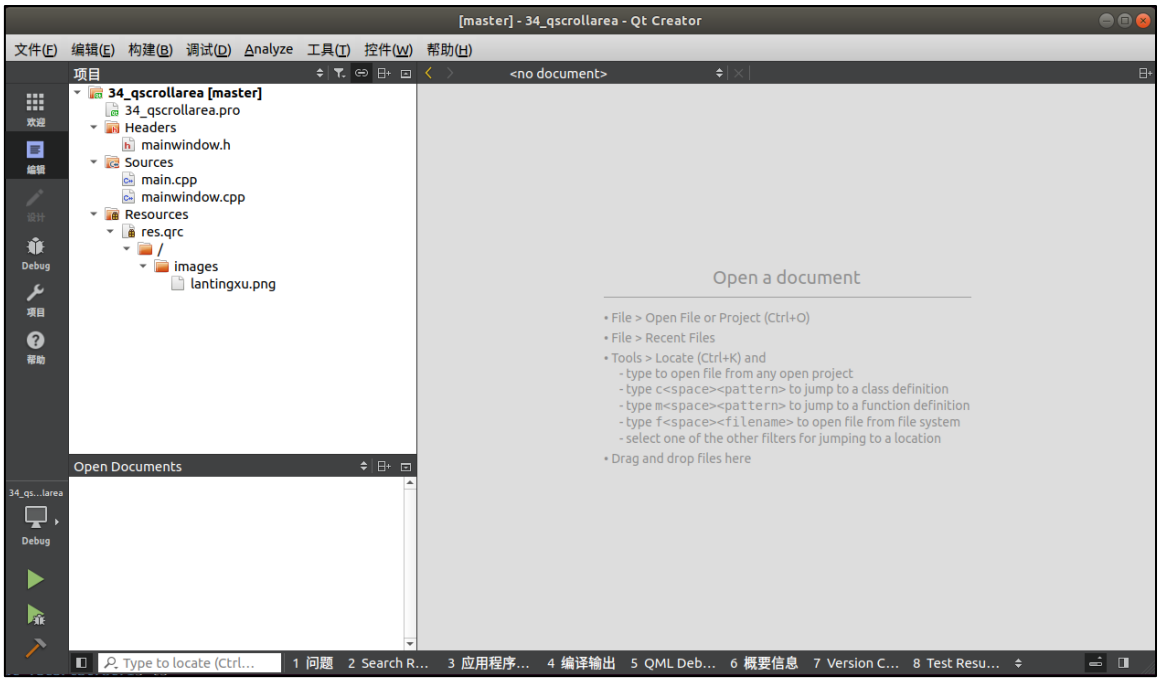
7.7.2.1 控件简介

QScrollArea 类提供到另一个小部件的滚动视图。

7.7.2.2 用法示例

例 34_qscrollarea 滚动视图（难度：简单），使用一个 Label 标签，将 Label 标签设置为一张图片，并把 Label 标签放置于滚动区域内，此时图片应要大于滚动区域才会出现滚动条。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成，并添加了一张资源图片（添加资源图片的步骤请参考 7.1.4 小节），如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QScrollArea>
6  #include <QLabel>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15
16 private:
17     /* 定义 QScrollArea 对象 */
18     QScrollArea *scrollArea;
19     QLabel *label;
20 };
21 #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1  #include "mainwindow.h"
2
3  MainWindow::MainWindow(QWidget *parent)
4      : QMainWindow(parent)
5  {
6      this->setGeometry(0, 0, 800, 480);
7
8      scrollArea = new QScrollArea(this);
9      /* 设置滚动区域为 700*380 */
10     scrollArea->setGeometry(50, 50, 700, 380);
11
12     label = new QLabel();
13     /* label 显示的 lantingxu.png 图片分辨率为 1076*500 */
14     QImage image(":/images/lantingxu.png");
15     label->setPixmap(QPixmap::fromImage(image));
16
17     scrollArea->setWidget(label);
18
19 }
20
21 MainWindow::~MainWindow()
22 {
23 }
```

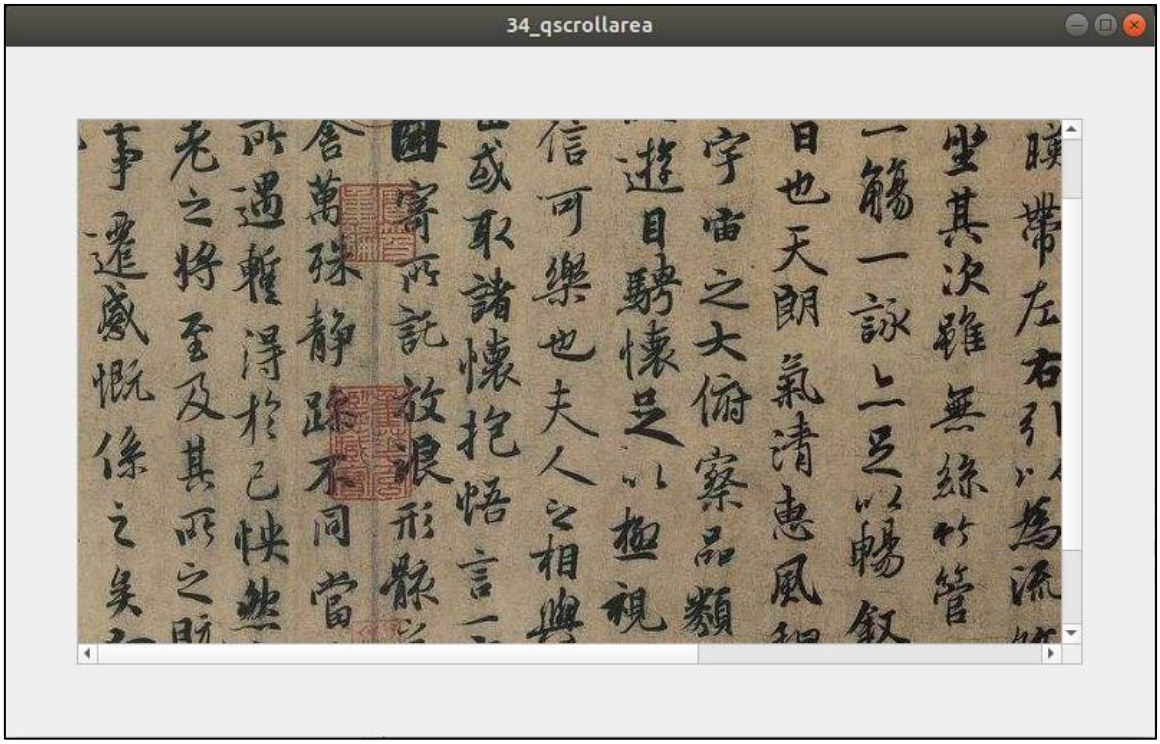
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1  #include "mainwindow.h"
2
3  #include <QApplication>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8      MainWindow w;
9      w.show();
10     return a.exec();
11 }
```

7.7.2.3 运行效果

程序编译运行的结果如下，由于图片的大小大于滚动区域的大小，所以在滚动区域出现了滚动条，可以拖动滚动条来查看这张图片其余部分。



7.7.3 QToolBox

7.7.3.1 控件简介

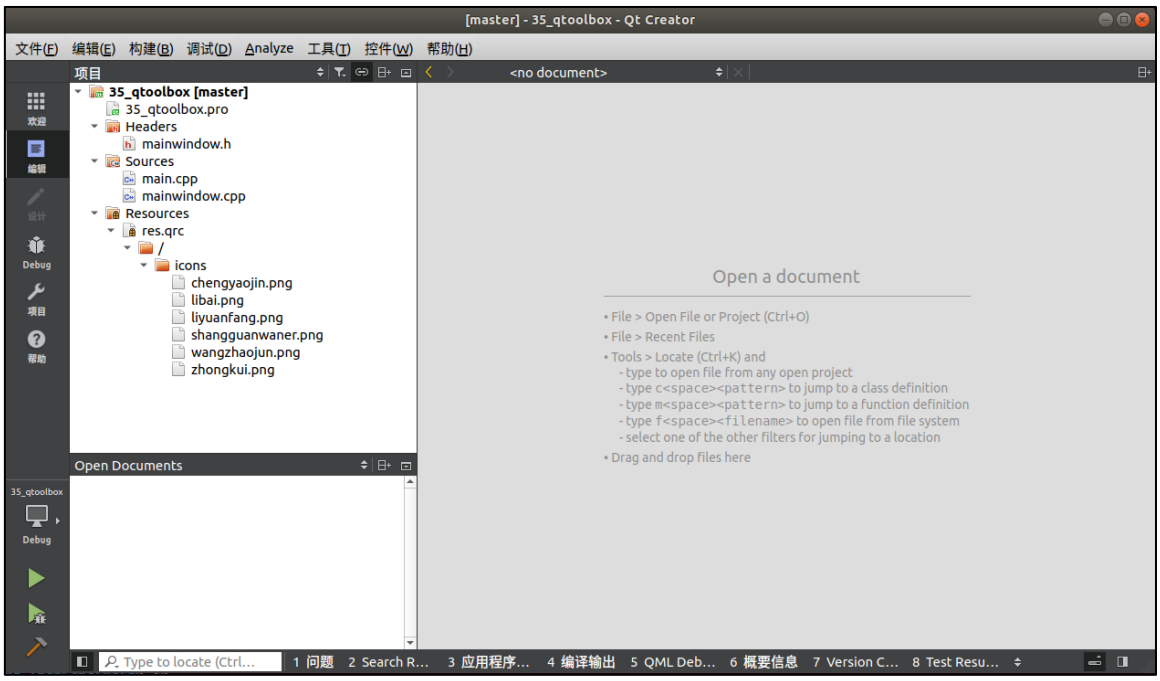
QToolBox（工具盒类）提供了一种列状的层叠窗体，中文译为工具箱，类似抽屉。

7.7.3.2 用法示例

例 35_qtoolbox，QQ 好友面板之 QToolBox（难度：简单），本例将使用到前面的知识 QGroupBox 组框与 QBoxLayout 布局管理。前面我们已经学过 QGroupBox 组框和学过 QBoxLayout。有了前面的基础，那么去理解本例就会快很多。

本例思路:使用 6 个 QToolButton 分成 2 组，使用垂直布局将 QToolButton 的 2 组排布好，然后添加到 2 组 QGroupBox 组框，再把 2 组 QGroupBox 组框作为子项添加到 QToolBox。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成，并添加了几张资源图片（添加资源图片的步骤请参考 7.1.4 小节），如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QToolBox>
6  #include <QGroupBox>
7  #include <QToolButton>
8  #include <QVBoxLayout>
9
10 class MainWindow : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     MainWindow(QWidget *parent = nullptr);
16     ~MainWindow();
17
18 private:
19     /* 声明对象 */
20     QToolBox *toolBox;
21     QGroupBox *groupBox[2];
22     QVBoxLayout  *vBoxLayout[2];
23     QToolButton *toolButton[6];
24
25 };
```

```
26  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       this->setGeometry(0, 0, 800, 480);
7
8       toolBox = new QToolBox(this);
9       toolBox->setGeometry(300, 50, 200, 250);
10      /* 设置 toolBox 的样式，此处设置为 30%不透明度的黑色 */
11      toolBox->setStyleSheet("QToolBox {background-color:rgba(0, 0, 0,
30%);}");
12
13      for(int i = 0; i < 2; i++){
14          vBoxLayout[i] = new QVBoxLayout();
15          groupBox[i] = new QGroupBox(this);
16      }
17
18      /* 字符串链表 */
19      QList <QString>strList;
20      strList<<"李白"<<"王照君"<<"李元芳"<<"程咬金"<<"钟馗"<<"上官婉儿";
21
22      /* 字符串图标链表 */
23      QList <QString>iconsList;
24      iconsList<<":/icons/libai"<<":/icons/wangzhaojun"
25               <<":/icons/liyuanfang"<<":/icons/chengyaojin"
26              <<":/icons/zhongkui"<<":/icons/shangguanwaner";
27
28      for(int i = 0; i < 6; i++){
29          toolButton[i] = new QToolButton();
30          /* 设置 toolButton 图标 */
31          toolButton[i]->setIcon(QIcon(iconsList[i]));
32          /* 设置 toolButton 的文本 */
33          toolButton[i]->setText(strList[i]);
34          /* 设置 toolButton 的大小 */
35          toolButton[i]->setFixedSize(150, 40);
36          /* 设置 toolButton 的 setToolButtonStyle 的样式 */
37          toolButton[i]->setToolButtonStyle(
38                      Qt::ToolButtonTextBesideIcon
39                      );
```

```
40          if( i < 3 ) {
41              /* 将 toolButton 添加到时垂直布局 */
42              vBoxLayout[0]->addWidget(toolButton[i]);
43              /* 添加一个伸缩量 1 */
44              vBoxLayout[0]->addStretch(1);
45          } else {
46              vBoxLayout[1]->addWidget(toolButton[i]);
47              vBoxLayout[1]->addStretch(1);
48          }
49      }
50      /* 将垂直布局的内容添加到组框 groupBox */
51      groupBox[0]->setLayout(vBoxLayout[0]);
52      groupBox[1]->setLayout(vBoxLayout[1]);
53
54      /* 将组框加入 QToolBox 里 */
55      toolBox->addItem(groupBox[0],"我的好友");
56      toolBox->addItem(groupBox[1],"黑名单");
57  }
58
59  MainWindow::~MainWindow()
60  {
61  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.7.3.3 运行效果

程序编译运行的结果如下，本次使用 QToolButool 作为 QGroupBox 的子项，也可以使用其他 QWidget 小部件，如 QWidget 等。(注意本程序在 linux 运行效果如下，若如在 Windows 下，可能 QToolBox 的显示样式不一样)。点击“我的好友”列表，则会出现好友列表，点击“黑名单”则会出现黑名单列表。

7.7.4 QTabWidget

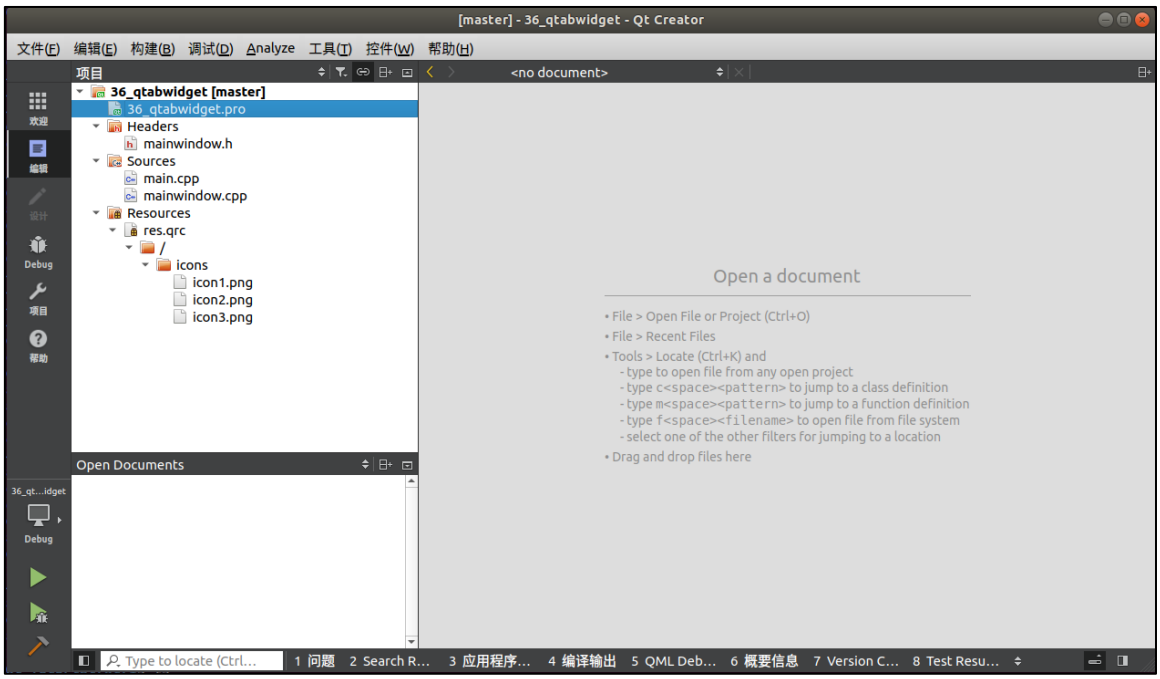
7.7.4.1 控件简介

QTabWidget 继承 QWidget，QTabWidget 类提供了一组选项卡（多页面）小部件。QTabWidget 主要是用来分页显示的，每一页一个界面，众多界面公用一块区域，节省了界面大小，很方便的为用户显示更多的信息。类似浏览器的多标签页面，所以这个控件在实际项目中也会经常用到。

7.7.4.2 用法示例

例 36_qtabwidget，标题栏多页面切换（难度：简单），本例创建 3 个页面，每个页面里有一个 Label 标签部件，点击每个页面的选项卡则会切换到不同的页面上。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成，并添加了几张资源图片（添加资源图片的步骤请参考 7.1.4 小节），如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QTableWidget>
6  #include <QHBoxLayout>
7  #include <QLabel>
8
9  class MainWindow : public QMainWindow
10 {
11     Q_OBJECT
12
13 public:
14     MainWindow(QWidget *parent = nullptr);
15     ~MainWindow();
16
17 private:
18     /* 声明对象 */
19     QWidget *widget;
20     QTabWidget *tabWidget;
21     QHBoxLayout  *hBoxLayout;
22     QLabel *label[3];
23 };
24 #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       this->setGeometry(0, 0, 800, 480);
7
8       widget = new QWidget(this);
9       /* 居中 */
10      this->setCentralWidget(widget);
11
12      /* 多页面小部件 */
13      tabWidget = new QTabWidget();
14
15      /* 水平布局实例化 */
16      hBoxLayout = new QHBoxLayout();
17      QList <QString>strLabelList;
18      strLabelList<<"标签一"<<"标签二"<<"标签三";
19
20      QList <QString>strTabList;
21      strTabList<<"页面一"<<"页面二"<<"页面三";
22
23      QList <QString>iconList;
24      iconList<<":/icons/icon1"
25            <<":/icons/icon2"
26           <<":/icons/icon3";
27
28      for (int i = 0; i < 3; i++) {
29          label[i] = new QLabel();
30          /* 设置标签文本 */
31          label[i]->setText(strLabelList[i]);
32          /* 标签对齐方式（居中） */
33          label[i]->setAlignment(Qt::AlignCenter);
34          /* 添加页面 */
35          tabWidget->addTab(label[i],
36                            QIcon(iconList[i]),
37                            strTabList[i]
38                            );
39      }
40      /* 是否添加关闭按钮 */
41      //tabWidget->setTabsClosable(true);
42      /* 将 tabWidget 水平直排布 */
```

```
43      hBoxLayout->addWidget(tabWidget);
44      /* 将垂直布局设置到 widget */
45      widget->setLayout(hBoxLayout);
46  }
47
48  MainWindow::~MainWindow()
49  {
50  }
```

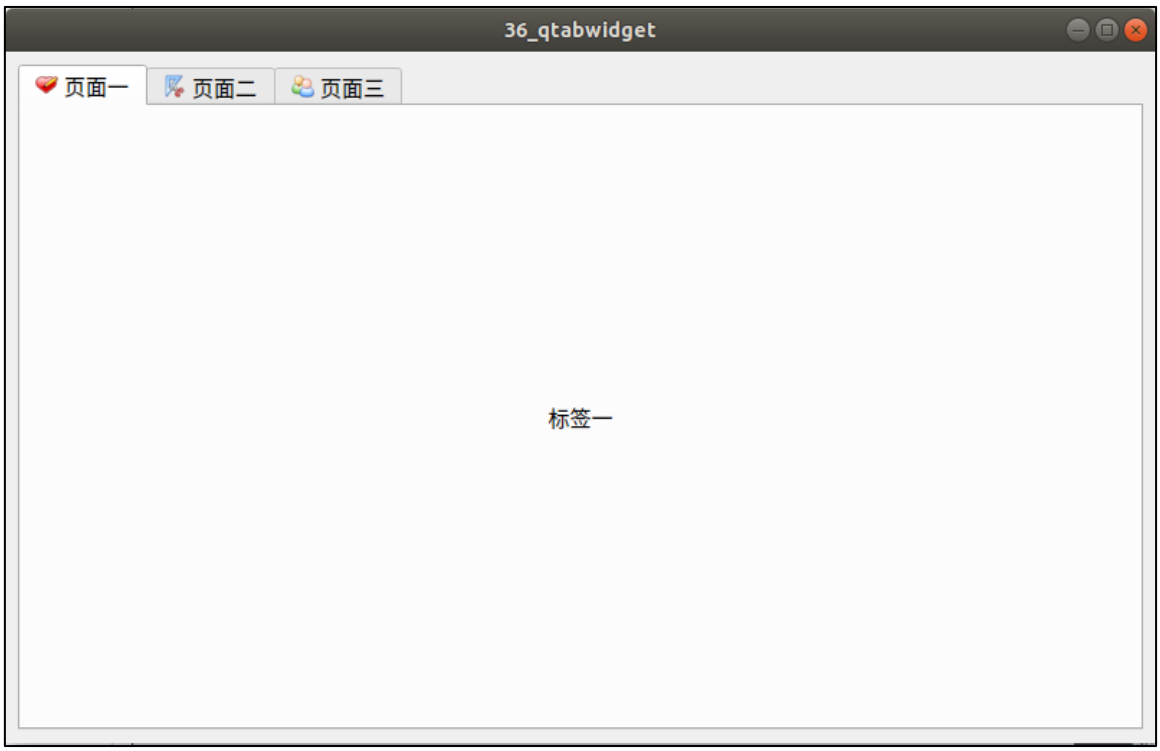
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.7.4.3 运行效果

程序编译运行的结果如下，点击不同页面的选项卡则会切换到不同的页面上。本例还可拓展使用 void setTabsClosable(bool closeable)函数在选项卡后加一个关闭按钮，再连接信号槽实现关闭页面的操作。本例就不再添加代码了，比较简单。

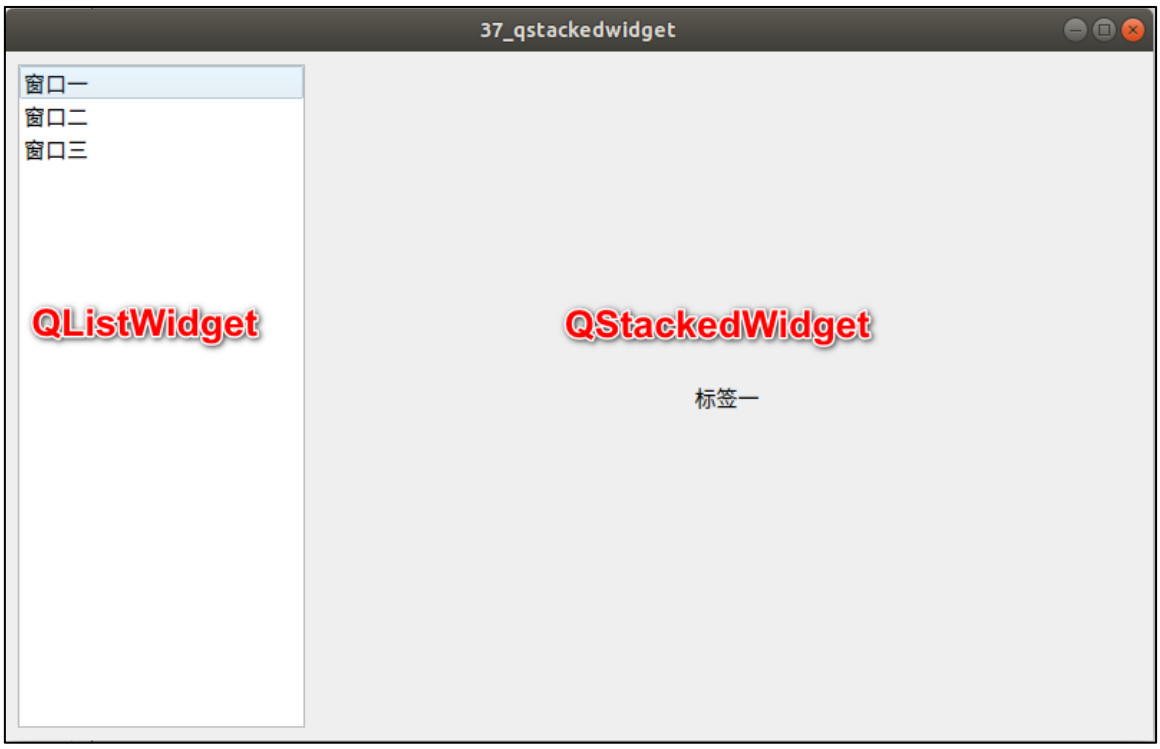


7.7.5 QStackedWidget

7.7.5.1 控件简介

QStackedWidget 继承 QFrame。QStackedWidget 类提供了一个小部件堆栈，其中一次只能看到一个小部件，与 QQ 的设置面板类似。

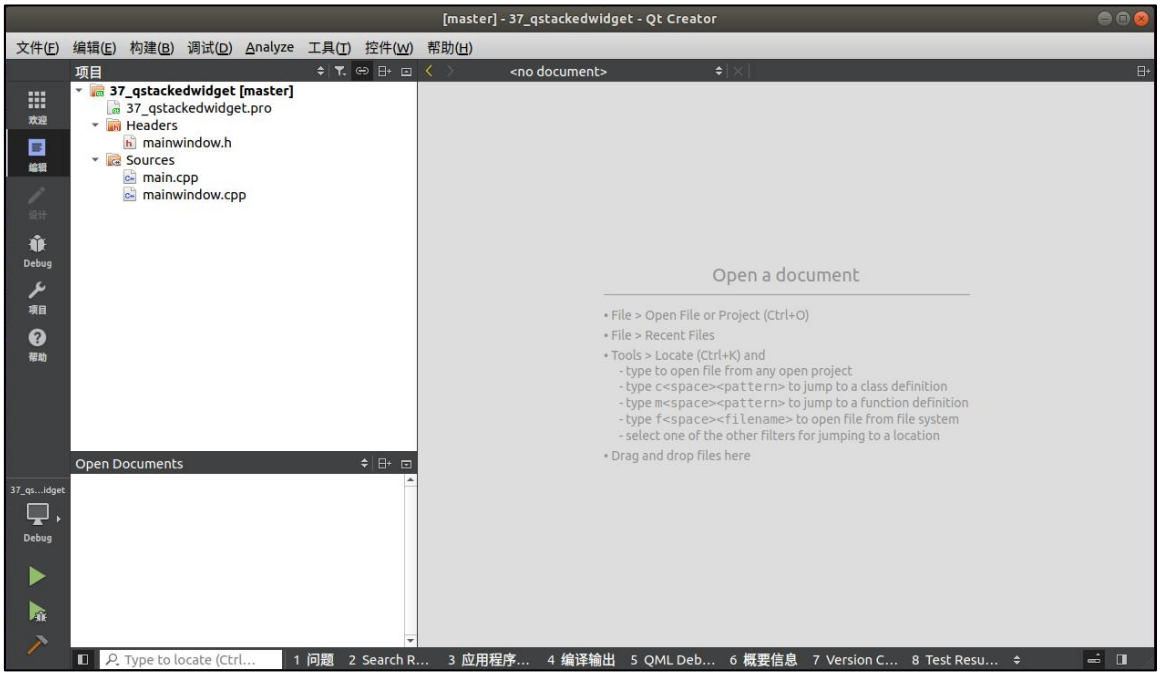
QStackedWidget 可用于创建类似于 QTabWidget 提供的用户界面。它是构建在 QStackedLayout 类之上的一个方便的布局小部件。常与 QListWidget 搭配使用，效果如下图，左边的是 QListWidget 列表，右边的是 QStackedWidget。他们一般与信号槽连接，通过点击左边的 QListWidget 列表，使用信号槽连接后，就可以让右边的 QStackedWidget 显示不同的内容，每次显示一个 widget 小部件。



7.7.5.2 用法示例

例 37_qstackedwidget，列表栏多页面切换（难度：简单），本例创建 3 个堆栈页面，每个页面里有一个 Label 标签部件，点击每个列表的不同项则会切换到不同的页面上。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成，如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QStackedWidget>
6  #include <QHBoxLayout>
7  #include <QListWidget>
8  #include <QLabel>
9
10 class MainWindow : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     MainWindow(QWidget *parent = nullptr);
16     ~MainWindow();
17
18 private:
19     /* widget 小部件 */
20     QWidget *widget;
21     /* 水平布局 */
22     QHBoxLayout *hBoxLayout;
23     /* 列表视图 */
24     QListWidget *listWidget;
25     /* 堆栈窗口部件 */
26     QStackedWidget *stackedWidget;
27     /* 3 个标签 */
28     QLabel *label[3];
29
30 };
31 #endif // MAINWINDOW_H
32
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1  #include "mainwindow.h"
2
3  MainWindow::MainWindow(QWidget *parent)
4      : QMainWindow(parent)
5  {
6      this->setGeometry(0, 0, 800, 480);
7
8      /* widget 小部件实例化 */
```

```
9       widget = new QWidget(this);
10
11      /* 设置居中 */
12      this->setCentralWidget(widget);
13
14      /* 垂直布局实例化 */
15      hBoxLayout = new QHBoxLayout();
16
17      /* 堆栈部件实例化 */
18      stackedWidget = new QStackedWidget();
19
20      /* 列表实例化 */
21      listWidget = new QListWidget();
22
23      QList <QString>strListWidgetList;
24      strListWidgetList<<"窗口一"<<"窗口二"<<"窗口三";
25
26      for (int i = 0; i < 3; i++){
27          /* listWidget 插入项 */
28          listWidget->insertItem(
29                      i,
30                      strListWidgetList[i]
31                      );
32      }
33
34      QList <QString>strLabelList;
35      strLabelList<<"标签一"<<"标签二"<<"标签三";
36
37      for (int i = 0; i < 3; i++){
38          label[i] = new QLabel();
39          /* 设置标签文本 */
40          label[i]->setText(strLabelList[i]);
41          /* 标签对齐方式（居中） */
42          label[i]->setAlignment(Qt::AlignCenter);
43          /* 添加页面 */
44          stackedWidget->addWidget(label[i]);
45      }
46
47      /* 设置列表的最大宽度 */
48      listWidget->setMaximumWidth(200);
49      /* 添加到水平布局 */
50      hBoxLayout->addWidget(listWidget);
```

```
51      hBoxLayout->addWidget(stackedWidget);
52
53      /* 将 widget 的布局设置成 hboxLayout */
54      widget->setLayout(hBoxLayout);
55
56      /* 利用 listWidget 的信号函数 currentRowChanged()与
57       * 槽函数 setCurrentIndex(),进行信号与槽连接
58       */
59      connect(listWidget, SIGNAL(currentRowChanged(int)),
60              stackedWidget, SLOT(setCurrentIndex(int)));
61  }
62
63  MainWindow::~MainWindow()
64  {
65  }
```

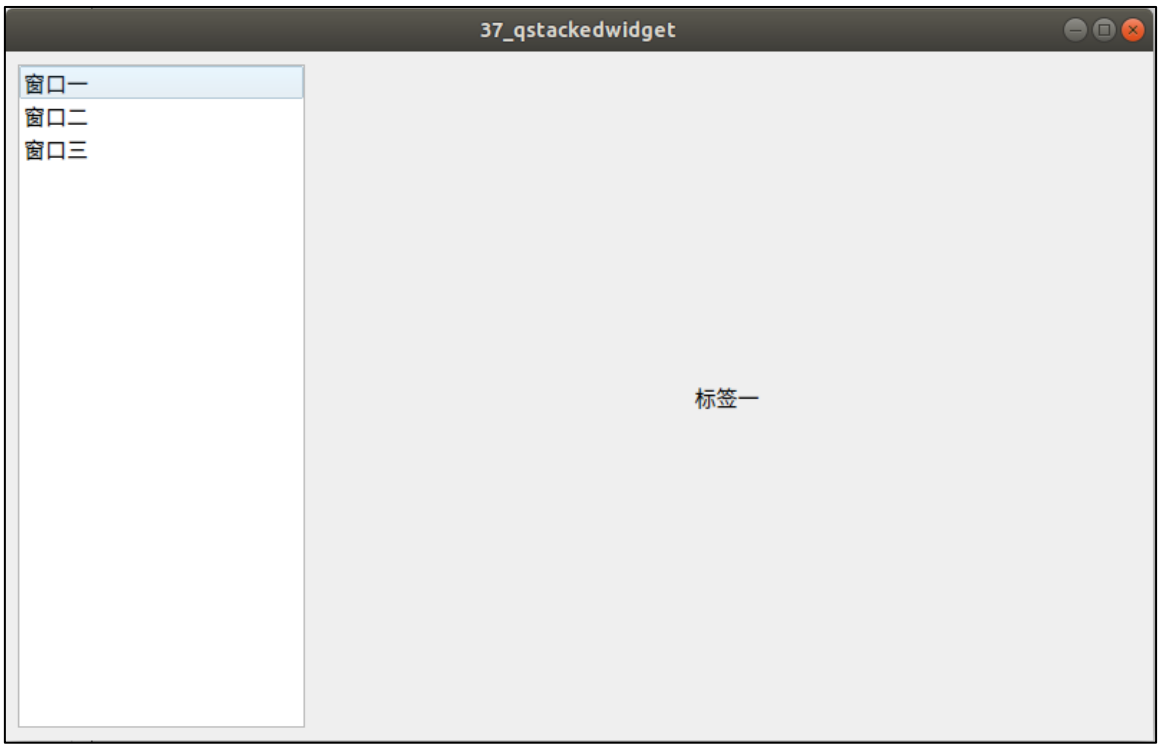
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.7.5.3 运行效果

程序编译运行的结果如下，点击列表视图的不同的项会切换到不同的页面上。



7.7.6 QFrame

在 7.3.5 小节已经举过例子。

7.7.7 QWidget

7.7.7.1 控件简介

QWidget 类是所有用户界面对象的基类（如 QLabel 类继承于 QFrame 类，而 QFrame 类又继承于 QWidget 类）。Widget 是用户界面的基本单元：它从窗口系统接收鼠标，键盘和其他事件，并在屏幕上绘制自己。每个 Widget 都是矩形的，它们按照 Z-order 进行排序。

注：Z-order 是重叠二维对象的顺序，例如堆叠窗口管理器中的窗口。典型的 GUI 的特征之一是窗口可能重叠，使得一个窗口隐藏另一个窗口的一部分或全部。 当两个窗口重叠时，它们的 Z 顺序确定哪个窗口出现在另一个窗口的顶部。

理解：术语"z-order"指沿着 z 轴物体的顺序。三维坐标轴中 x 横轴，y 数轴，z 上下轴。可以将 gui 窗口视为平行与显示平面的一系列平面。因此，窗口沿着 z 轴堆叠。所以 z-order 指定了窗口的前后顺序。就像您桌面上的一叠纸一样，每张纸是一个窗口，桌面是您的屏幕，最上面的窗口 z 值最高。

QWidget 不是一个抽象类，它可以用作其他 Widget 的容器，并很容易作为子类来创建定制 Widget。它经常用于创建、放置和容纳其他的 Widget 窗口。

上面这么多例子都有用到 QWidget，如 7.5.1 小节，请自行参考，没有具体例子可写，比较简单。

7.7.8 QMdiArea

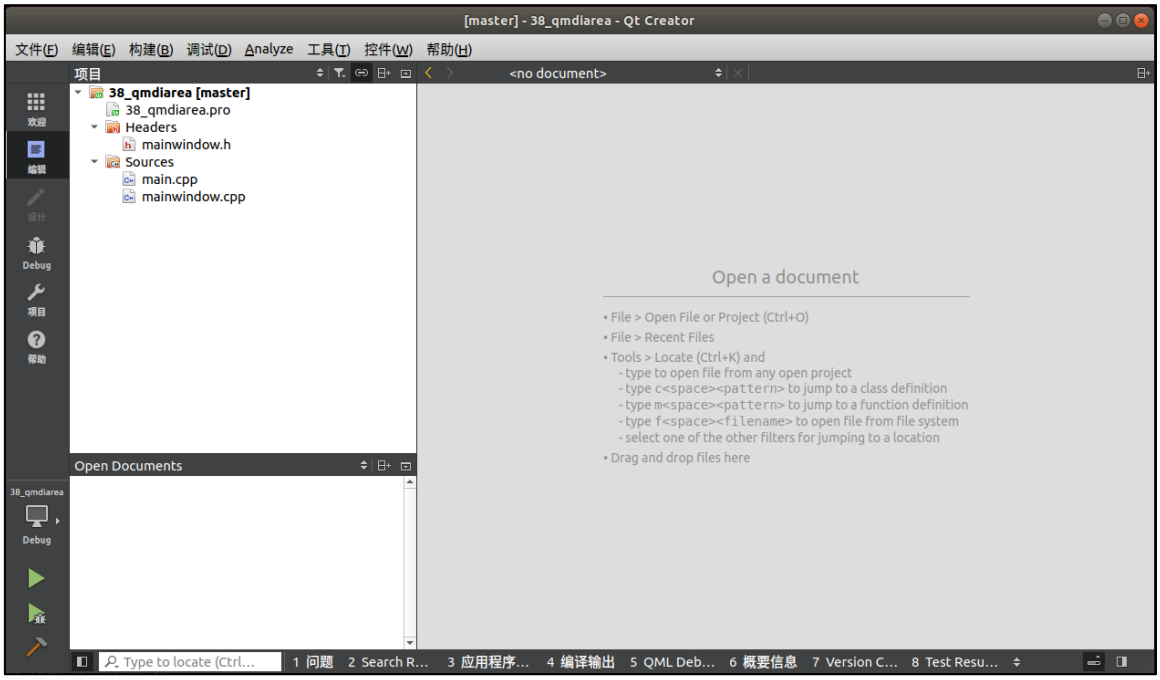
7.7.8.1 控件简介

QMdiArea 继承 QAbstractScrollArea。QMdiArea 小部件提供一个显示 MDI 窗口的区域。QMdiArea 的功能本质上类似于 MDI 窗口的窗口管理器。大多数复杂的程序，都使用 MDI 框架，在 Qt designer 中可以直接将控件 MDI Area 拖入使用。

7.7.8.2 用法示例

例 38_qmdiarea，父子窗口（难度：简单），本例创建一个 MDI Area 区域，使用一个按钮，每单击按钮时，就会在 MDI Area 区域新建一个 MdiSubWindow 窗口。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成，如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QMdiSubWindow>
6   #include <QMdiArea>
7   #include <QPushButton>
8
9   class MainWindow : public QMainWindow
10  {
11      Q_OBJECT
```

```
12
13  public:
14      MainWindow(QWidget *parent = nullptr);
15      ~MainWindow();
16
17  private:
18      /* Mdi Area 区域对象 */
19      QMdiArea *mdiArea;
20      /* MdiSubWindow 子窗口对象 */
21      QMdiSubWindow *newMdiSubWindow;
22      /* 用作点击创建新的窗口 */
23      QPushButton *pushButton;
24
25  private slots:
26      /* 按钮槽函数 */
27      void creat_newMdiSubWindow();
28
29  };
30  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置窗口的显示位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8       pushButton = new QPushButton("新建窗口", this);
9       pushButton->setGeometry(0, 30, 100, 30);
10
11      mdiArea = new QMdiArea(this);
12      /* 设置 MDI Area 区域大小 */
13      mdiArea->setGeometry(100, 30, 700, 430);
14      /* 连接信号槽 */
15      connect(pushButton, SIGNAL(clicked()),
16              this, SLOT(creat_newMdiSubWindow()));
17  }
18
19  void MainWindow::creat_newMdiSubWindow()
20  {
21      newMdiSubWindow = new QMdiSubWindow();
22      newMdiSubWindow->setWindowTitle("新建窗口");
```

```
23
24      /* 如果窗口设置了 Qt::WA_DeleteOnClose 这个属性，
25       * 在窗口接受了关闭事件后，Qt 会释放这个窗口所占用的资源
26       */
27      newMdiSubWindow->setAttribute(Qt::WA_DeleteOnClose);
28
29      /* 添加子窗口 */
30      mdiArea->addSubWindow(newMdiSubWindow);
31      /* 显示窗口，不设置时为不显示 */
32      newMdiSubWindow->show();
33      /* 自适应窗口 */
34      newMdiSubWindow->sizePolicy();
35      /* 以级联的方式排列所有窗口 */
36      // mdiArea->cascadeSubWindows();
37      /* 以平铺方式排列所有窗口 */
38      mdiArea->tileSubWindows();
39  }
40
41  MainWindow::~MainWindow()
42  {
43  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.7.8.3 运行效果

程序编译运行的结果如下，当点击新建窗口按钮时，在 MDI Area 区域里新建了一个窗口标题为“新建窗口”窗口，下图为点击多次新建窗口的效果。本例使用了一个按钮，进行了新建窗口操作，其他功能例如添加删除按钮等，读者可以自行添加。本文在新建窗口里的内容为空，newMdiSubWindow 可以使用 setWidget(QWidget *widget)方法添加内容，如添加前面所学过的 QLineEdit 等。

7.7.9 QDockWidget

7.7.9.1 控件简介

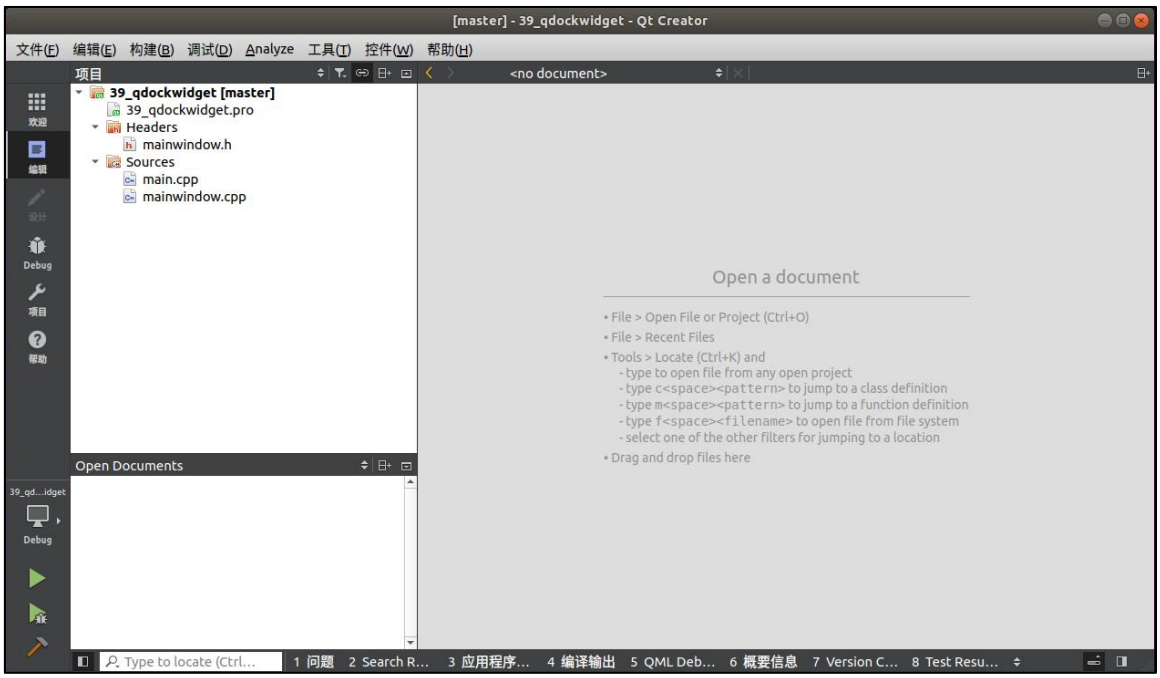
QDockWidget 继承 QWidget。QDockWidget 类提供了一个小部件，可以停靠在 QMainWindow 内，也可以作为桌面的顶级窗口浮动。

QDockWidget 提供了停靠部件的概念，也称为工具面板或实用程序窗口。停靠窗口是放置在 QMainWindow 中央窗口附近的停靠窗口部件区域中的辅助窗口。停靠窗口可以被移动到当前区域内，移动到新的区域，并由终端用户浮动(例如，不停靠)。QDockWidget API 允许程序员限制 dock widget 的移动、浮动和关闭能力，以及它们可以放置的区域。QDockWidget 的初始停靠区域有 Qt.BottomDockWidgetArea（底部停靠）、Qt.LeftDockWidgetArea（左边停靠、；Qt.RightDockWidgetArea（右边停靠）、Qt.TopDockWidgetArea（顶部停靠）和 Qt.NoDockWidgetArea（不显示 Widget）。

7.7.9.2 用法示例

例 39_qdockwidget，停靠窗口（难度：简单），本例创建一个停靠窗口，在停靠窗口里添加文本编辑框，并且把这个停靠窗口放置到 QMainWindow 的顶部。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成，如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QDockWidget>
6  #include <QTextEdit>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15
16 private:
17     /* 停靠窗口部件 */
18     QDockWidget *dockWidget;
19     /* 文本编辑框 */
20     QTextEdit *textEdit;
21
22 };
23 #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    /* 设置主窗体的显示的位置与大小 */
    this->setGeometry(0, 0, 800, 480);

    /* 实例化标题为停靠窗口 */
    dockWidget = new QDockWidget("停靠窗口", this);

    /* 实例化文本编辑框 */
    textEdit = new QTextEdit(dockWidget);

    textEdit->setText("这是一个测试");

    /* 停靠窗口添加文本编辑框 */
    dockWidget->setWidget(textEdit);

    /* 放在主窗体的顶部 */
    this->addDockWidget(Qt::TopDockWidgetArea, dockWidget);
}

MainWindow::~MainWindow()
{
}
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

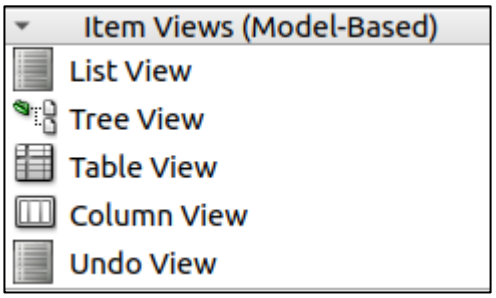
main.cpp 编程后的代码

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

7.7.9.3 运行效果

程序编译运行的结果如下。拖动停靠窗口可以移动到另一个位置，松手则停靠。

7.8 项目视图组(基于模型)



上图需要注意的是，在低版本的 Qt，Column View 与 Undo View 是没有封装成可视控件形式在 Qt Creator 中。

以上各个控件的解释如下：

（1） List View:清单视图

（2） Tree View:树形视图

（3） Table View:表视图

（4） Column View:列表视图

（5） Undo View:撤销列表视图

以下是各个控件的简介：

QListView 继承 QAbstractItemView，被 QListWidget 和 QUndoView 继承。QListView 类提供模型上的列表或图标视图。QListView 以简单的非分层列表或图标集合的形式显示存储在模型中的项。该类用于提供以前由 QListBox 和 QIconView 类提供的列表和图标视图，但是使用了 Qt 的模型/视图体系结构提供的更灵活的方法。QListView 是基于 Model，而 QListWidget 是

基于 Item，这是它们的本质区别。QListView 需要建模，所以减少了冗余的数据，程序效率高;而 QListWidget 是一个升级版本的 QListView（实质是封装好了 model 的 QListView），它已经自己为我们建立了一个数据存储模型（QListWidgetItem），操作方便，直接调用 addItem 即可添加项目（ICON，文字）。

QTreeView 继承 QAbstractItemView，被 QTreeWidget 继承。QTreeView 类提供树视图的默认模型/视图实现。QTreeView 实现了模型项的树表示。该类用于提供以前由 QListView 类提供的标准分层列表，但是使用了 Qt 的模型/视图体系结构提供的更灵活的方法。

QTableView 继承 QAbstractItemView，被 QTableWidget 继承。QTableView 类提供了表视图的默认模型/视图实现。QTableView 实现了一个表视图，用于显示来自模型的项。该类用于提供以前由 QTable 类提供的标准表，但使用 Qt 的模型/视图体系结构提供的更灵活的方法。

QColumnView 继承 QAbstractItemView。QColumnView 在许多 QListViews 中显示一个模型，每个 QListViews 对应树中的每个层次结构。这有时被称为级联列表。QColumnView 类是模型/视图类之一，是 Qt 模型/视图框架的一部分。QColumnView 实现了由 QAbstractItemView 类定义的接口，以允许它显示由派生自 QAbstractItemModel 类的模型提供的数据。

QUndoView 继承 QlistView。QUndoView 类显示 QUndoStack 的内容。QUndoView 是一个 QListView，它显示在撤销堆栈上推送的命令列表。总是选择最近执行的命令。选择不同的命令会导致调用 QUndoStack::setIndex()，将文档的状态向后或向前滚动到新命令。可以使用 setStack() 显式地设置堆栈。或者，可以使用 setGroup()来设置 QUndoGroup 对象。当组的活动堆栈发生变化时，视图将自动更新自身。

7.8.1 QListView

7.8.1.1 控件简介

QListView 继承 QAbstractItemView，被 QListWidget 和 QUndoView 继承。QListView 类提供模型上的列表或图标视图。QListView 以简单的非分层列表或图标集合的形式显示存储在模型中的项。该类用于提供以前由 QListBox 和 QIconView 类提供的列表和图标视图，但是使用了 Qt 的模型/视图体系结构提供的更灵活的方法。QListView 是基于 Model，而 QListWidget 是基于 Item，这是它们的本质区别。QListView 需要建模，所以减少了冗余的数据，程序效率高;而 QListWidget 是一个升级版本的 QListView（实质是封装好了 model 的 QListView），它已经自己为我们建立了一个数据存储模型（QListWidgetItem），操作方便，直接调用 addItem 即可添加项目（ICON，文字）。

QT 提供了一些现成的 models 用于处理数据项（这些是 Qt 处理数据模型的精华，如果用到 Qt 数据模型，下面这些是经常要用到的）：

（1） QStringListModel 用于存储简单的 QString 列表。

（2） QStandardItemModel 管理复杂的树型结构数据项，每项都可以包含任意数据。

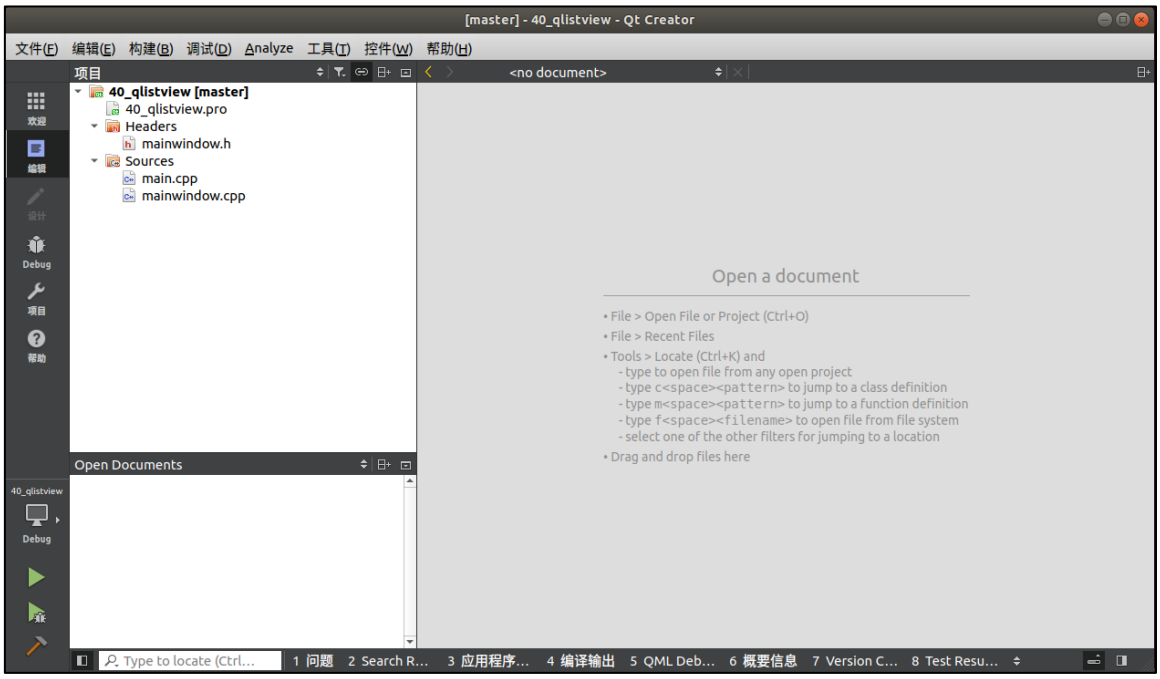
（3） QDirModel 提供本地文件系统中的文件与目录信息。

（4） QSqlQueryModel，QSqlTableModel,QSqlRelationTableModel 用来访问数据库。

7.8.1.2 用法示例

例 40_qlistview，清单图（难度：简单）。使用一个 ListView 控件，展示如何在列表中插入数据，其中表中的内容可编辑，可删除。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成，如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QListView>
6  #include <QStringListModel>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
```

```
15
16  private:
17      /* QListView 对象 */
18      QListView *listView;
19      /* 字符串模型对象 */
20      QStringListModel *stringListModel;
21
22  };
23  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗口位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 实例化 */
10      listView = new QListView(this);
11      /* 将 listView 居中 */
12      setCentralWidget(listView);
13
14      QStringList strList;
15      strList<<"高三（1）班"<<"高三（2）班"<<"高三（3）班";
16
17      /* 实例化，字符串模型 */
18      stringListModel = new QStringListModel(strList);
19
20      /* 向表中插入一段数据 */
21      listView->setModel(stringListModel);
22      /* 设置为视图为图标模式 */
23      listView->setViewMode(QListView::IconMode);
24      /* 设置为不可拖动 */
25      listView->setDragEnabled(false);
26  }
27
28  MainWindow::~MainWindow()
29  {
30  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1  #include "mainwindow.h"
2
3  #include <QApplication>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8      MainWindow w;
9      w.show();
10     return a.exec();
11 }
```

7.8.1.3 运行效果

程序编译运行的结果如下。双击表格中的项，可修改表格的内容，同时也可以删除内容等。

7.8.2 QTreeView

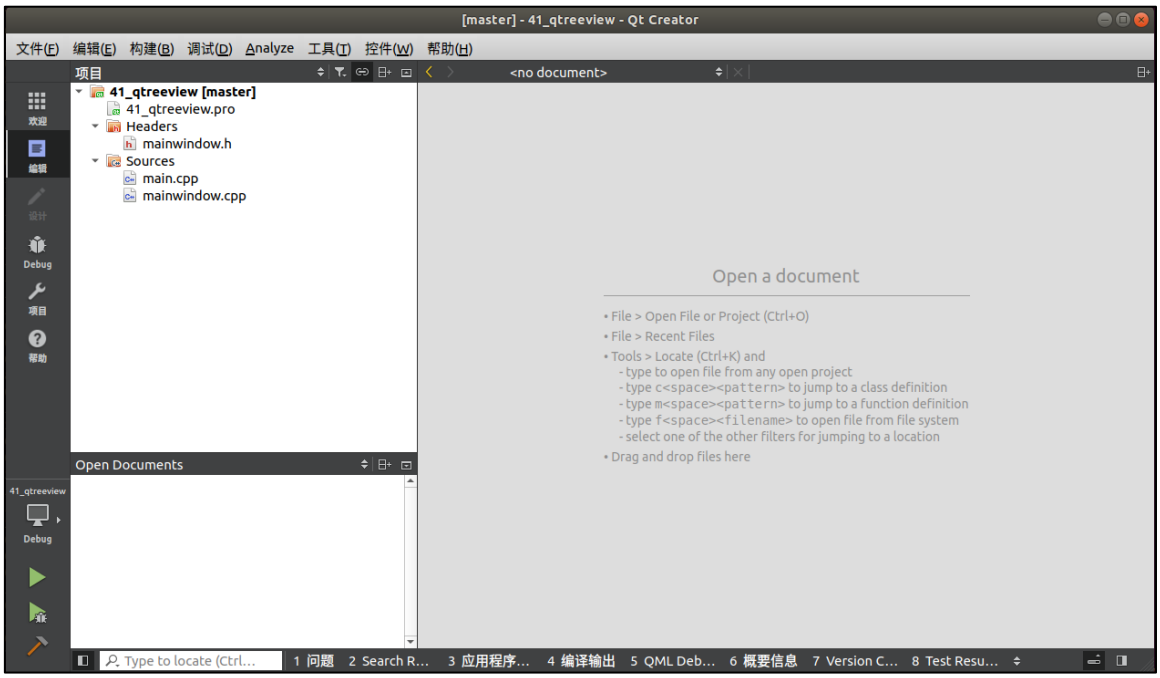
7.8.2.1 控件简介

QTreeView 继承 QAbstractItemView，被 QTreeWidget 继承。QTreeView 类提供树视图的默认模型/视图实现。QTreeView 实现了模型项的树表示。该类用于提供以前由 QListView 类提供的标准分层列表，但是使用了 Qt 的模型/视图体系结构提供的更灵活的方法。

7.8.2.2 用法示例

例 41_qtreeview，仿 word 标题（难度：简单）。要使一个 QTreeView 能够显示数据，需要构造一个 model 并设置 QTreeView。Qt 提供了一些类型的 Model，其中最常用的就是这个 QStandardItemModel 类，一般可以满足大部分需求。另外，表头的内容也由这个 model 管理，setHorizontalHeaderLabels 函数可以设置共有多少列、每列文字。一级标题直接使用 appendRow 方法添加到 model 上，次级标题则是添加到第一个父级标题上，依次构成父子关系树。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成，如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QTreeView>
6
7  class MainWindow : public QMainWindow
8  {
9      Q_OBJECT
10
11 public:
12     MainWindow(QWidget *parent = nullptr);
13     ~MainWindow();
14
15 private:
```

```
16      QTreeView  *treeView;
17
18  };
19  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QStandardItemModel>
4   #include <QStandardItem>
5
6   MainWindow::MainWindow(QWidget *parent)
7       : QMainWindow(parent)
8   {
9       /* 设置窗口的位置与大小 */
10      this->setGeometry(0, 0, 800, 480);
11      /* 实例化 QTreeView 对象 */
12      treeView = new QTreeView(this);
13      /* 居中 */
14      setCentralWidget(treeView);
15
16      /* 构建 Model */
17      QStandardItemModel *sdiModel = new QStandardItemModel(treeView);
18      sdiModel->setHorizontalHeaderLabels(
19                  QStringList()<<QStringLiteral("标题")
20                  << QStringLiteral("名称")
21                  );
22
23      for(int i = 0; i < 5; i++) {
24          /* 一级标题 */
25          QList<QStandardItem*> items1;
26          QStandardItem* item1 =
27                  new QStandardItem(QString::number(i));
28          QStandardItem* item2 =
29                  new QStandardItem(QStringLiteral("一级标题"));
30          /* 添加项一 */
31          items1.append(item1);
32          /* 添加项二 */
33          items1.append(item2);
34          /* appendRow 方法添加到 model 上 */
35          sdiModel->appendRow(items1);
36
```

```
37          for(int j = 0; j < 5; j++) {
38              /* 在一级标题后面插入二级标题 */
39              QList<QStandardItem*> items2;
40              QStandardItem* item3 =
41                      new QStandardItem(QString::number(j));
42              QStandardItem* item4 =
43                      new QStandardItem(QStringLiteral("二级标题"));
44              items2.append(item3);
45              items2.append(item4);
46              /* 使用 appendRow 方法添加到 item1 上 */
47              item1->appendRow(items2);
48          }
49      }
50      /* 设置 Model 给 treeView */
51      treeView->setModel(sdiModel);
52  }
53
54  MainWindow::~MainWindow()
55  {
56  }
```

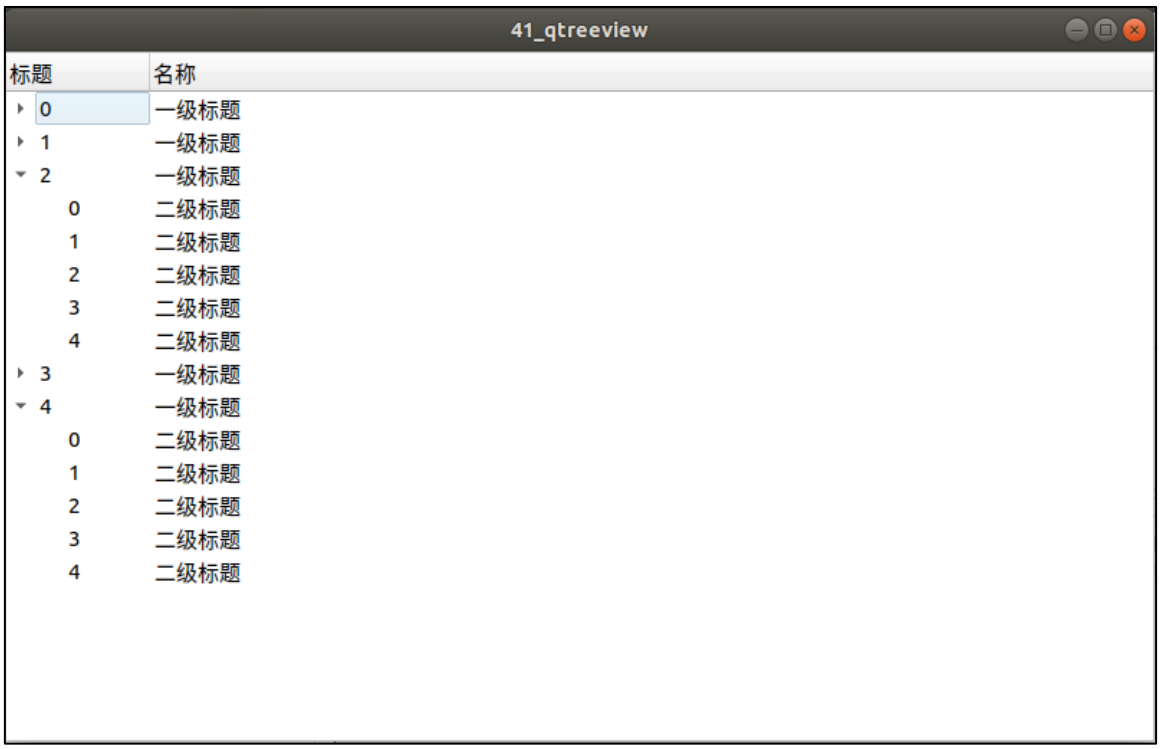
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.8.2.3 运行效果

程序编译运行的结果如下。可以点击数字来展开二级标题。



7.8.3 QTableView

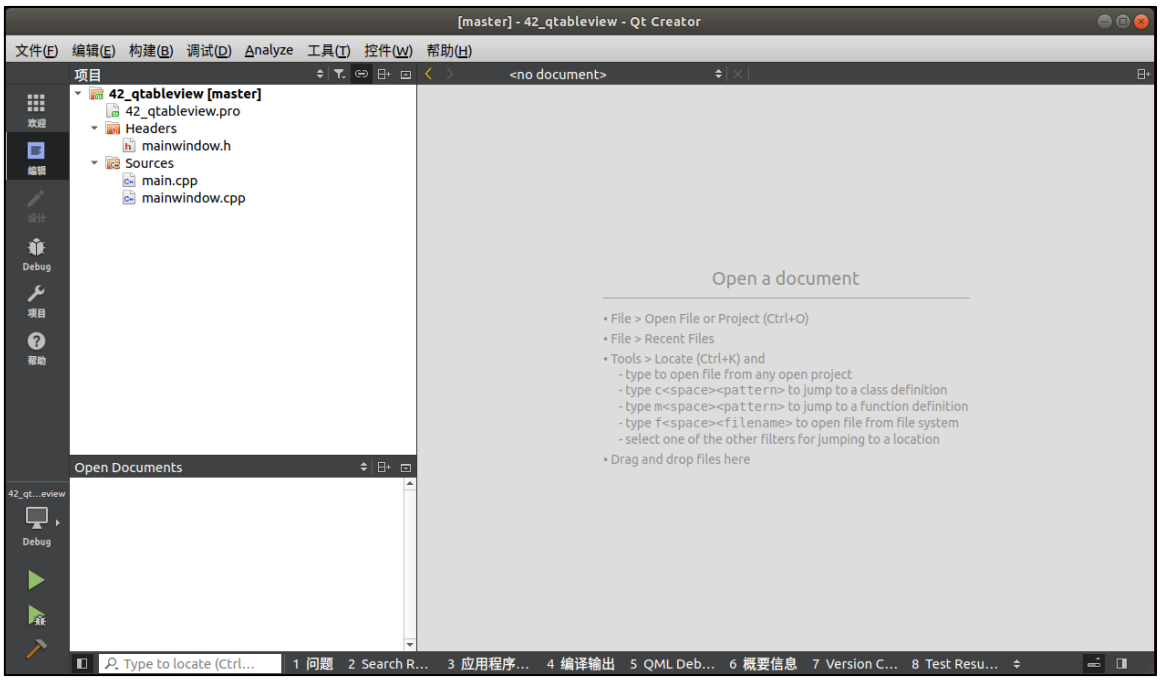
7.8.3.1 控件简介

QTableView 继承 QAbstractItemView，被 QTableWidget 继承。QTableView 类提供了表视图的默认模型/视图实现。QTableView 实现了一个表视图，用于显示来自模型的项。该类用于提供以前由 QTable 类提供的标准表，但使用 Qt 的模型/视图体系结构提供的更灵活的方法。

7.8.3.2 用法示例

例 42_qtableview，表格视图（难度：简单）。要使一个 QTableView 能够显示数据，需要构造一个 model 并设置给 QTableView。Qt 提供了一些类型的 Model，其中最常用的就是这个 QStandardItemModel 类，一般可以满足大部分需求。另外，表头的内容也由这个 model 管理，setHorizontalHeaderLabels 函数可以设置共有多少列、每列文字。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成，如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QTableView>
6
7  class MainWindow : public QMainWindow
8  {
9      Q_OBJECT
10
11 public:
12     MainWindow(QWidget *parent = nullptr);
13     ~MainWindow();
14
15 private:
16     QTableView *tableView;
17 };
18 #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1  #include "mainwindow.h"
2  #include <QStandardItemModel>
3  #include <QHeaderView>
4
5  MainWindow::MainWindow(QWidget *parent)
```

```
6       : QMainWindow(parent)
7   {
8       /* 设置窗口的位置与大小 */
9       this->setGeometry(0, 0, 800, 480);
10      tableView = new QTableView(this);
11      setCentralWidget(tableView);
12      /* 显示网格线 */
13      tableView->setShowGrid(true);
14
15      QStandardItemModel* model = new QStandardItemModel();
16      QStringList labels =
17              QObject::tr("语文,数学,英语").simplified().split(",");
18      /* 设置水平头标签 */
19      model->setHorizontalHeaderLabels(labels);
20
21      /* item */
22      QStandardItem* item = 0;
23      /* model 插入项内容 */
24      for(int i = 0; i < 5; i++){
25          item = new QStandardItem("80");
26          model->setItem(i, 0, item);
27          item = new QStandardItem("99");
28          model->setItem(i, 1, item);
29          item = new QStandardItem("100");
30          model->setItem(i, 2, item);
31      }
32      /* 将 model 设置给 tableView */
33      tableView->setModel(model);
34      /* 平均分列 */
35      tableView->horizontalHeader()
36              ->setSectionResizeMode(QHeaderView::Stretch);
37      /* 平均分行 */
38      tableView->verticalHeader()
39              ->setSectionResizeMode(QHeaderView::Stretch);
40  }
41
42  MainWindow::~MainWindow()
43  {
44  }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

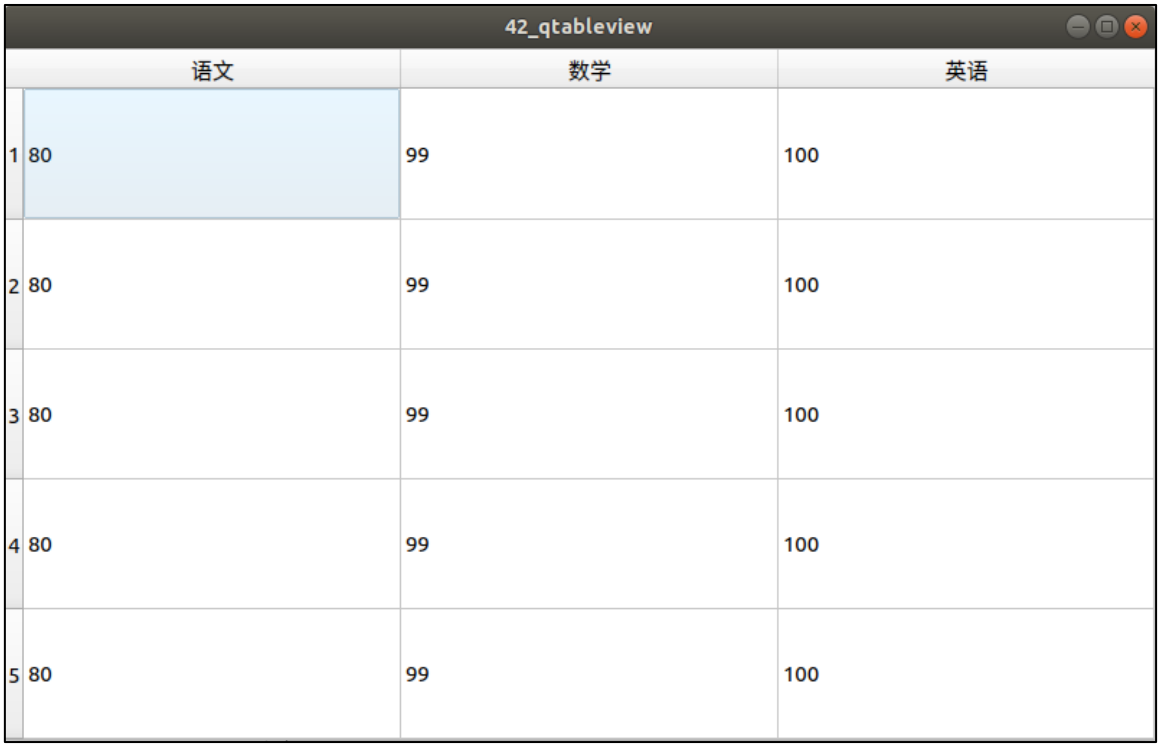
main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
```

```
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.8.3.3 运行效果

程序编译运行的结果如下。建立了一个成绩表格。



7.8.4 QColumnView

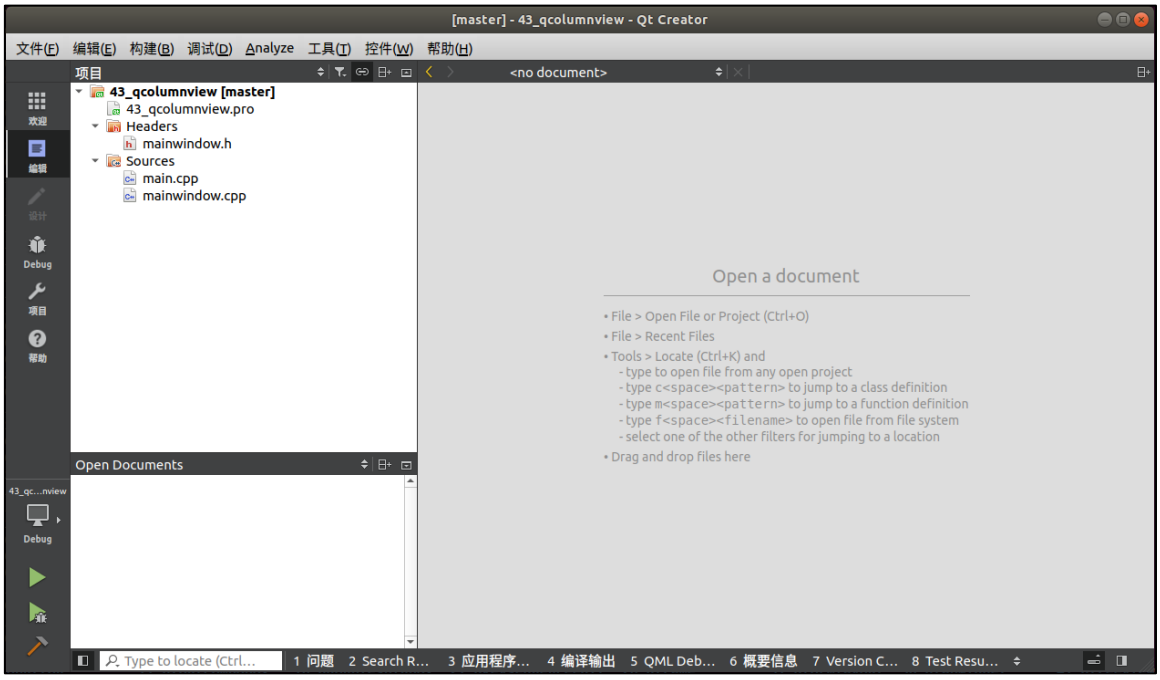
7.8.4.1 控件简介

QColumnView 继承 QAbstractItemView。QColumnView 在许多 QListViews 中显示一个模型，每个 QListViews 对应树中的每个层次结构。这有时被称为级联列表。QColumnView 类是模型/视图类之一，是 Qt 模型/视图框架的一部分。QColumnView 实现了由 QAbstractItemView 类定义的接口，以允许它显示由派生自 QAbstractItemModel 类的模型提供的数据。

7.8.4.2 用法示例

例 43_qcolumnview，收货地址（难度：简单）。使用一个 QColumnView，向其插入多级 QStandardItem。这样就可以模拟成一个多级联的视图。与我们在像某宝，某东里填写的收货地址十分类似。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成，如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QColumnView>
6
7  class MainWindow : public QMainWindow
8  {
9      Q_OBJECT
10
11 public:
12     MainWindow(QWidget *parent = nullptr);
13     ~MainWindow();
14
15 private:
16     QColumnView *columnView;
17 };
```

```
18 #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1  #include "mainwindow.h"
2  #include <QStandardItem>
3
4  MainWindow::MainWindow(QWidget *parent)
5      : QMainWindow(parent)
6  {
7      /* 设置主窗体显示位置与大小 */
8      this->setGeometry(0, 0, 800, 480);
9      QStandardItemModel *model = new QStandardItemModel;
10
11     /* 省份 */
12     QStandardItem *province = new QStandardItem("广东省");
13
14     /* 城市 */
15     QStandardItem *city1 = new QStandardItem("茂名市");
16     QStandardItem *city2 = new QStandardItem("中山市");
17
18     /* 添加城市到省份下 */
19     province->appendRow(city1);
20     province->appendRow(city2);
21
22     QStandardItem *town1 = new QStandardItem("电白镇");
23     QStandardItem *town2 = new QStandardItem("南头镇");
24
25     /* 添加城镇到城市下 */
26     city1->appendRow(town1);
27     city2->appendRow(town2);
28
29     columnView = new QColumnView;
30
31     /* 建立 model */
32     model->appendRow(province);
33
34     /* 设置 model */
35     columnView->setModel(model);
36
37     /* 设置居中 */
38     setCentralWidget(columnView);
39 }
```

```
40
41  MainWindow::~MainWindow()
42  {
43  }
```

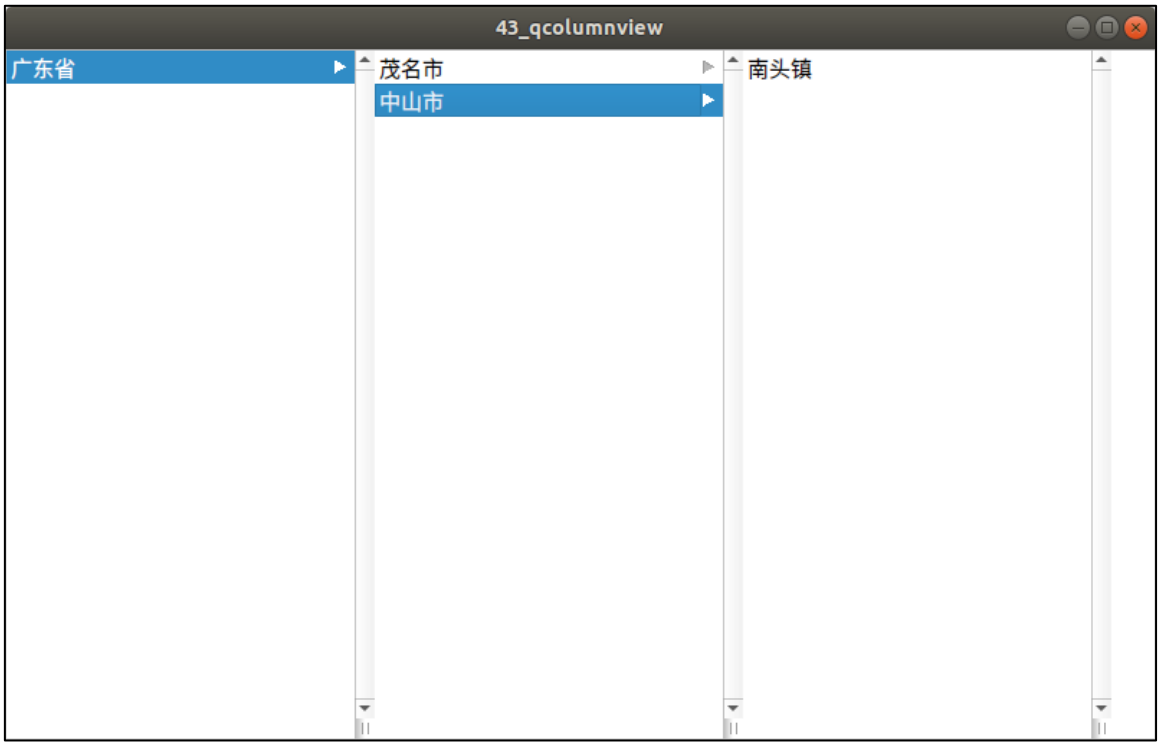
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.8.4.3 运行效果

程序编译运行的结果如下。当点击省份出现城市，点击城市出现城镇。



7.8.5 QUndoView

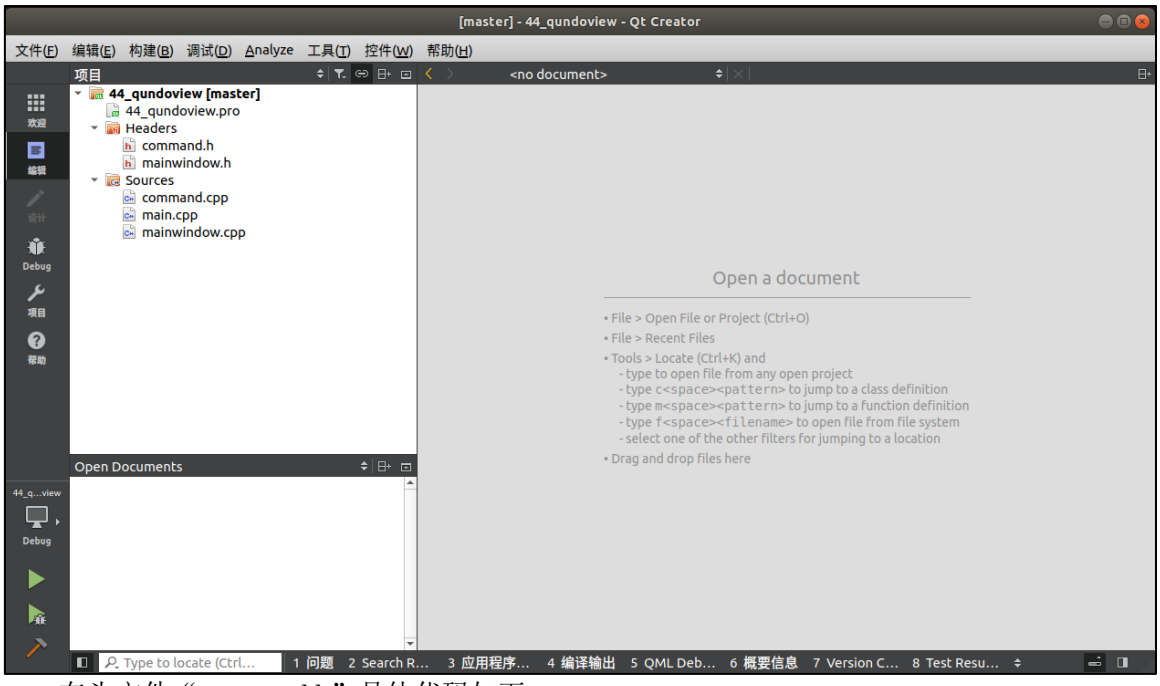
7.8.5.1 控件简介

QUndoView 继承 QlistView。QUndoView 类显示 QUndoStack 的内容。QUndoView 是一个 QListView，它显示在撤销堆栈上推送的命令列表。总是选择最近执行的命令。选择不同的命令会导致调用 QUndoStack::setIndex()，将文档的状态向后或向前滚动到新命令。可以使用 setStack() 显式地设置堆栈。或者，可以使用 setGroup()来设置 QUndoGroup 对象。当组的活动堆栈发生变化时，视图将自动更新自身。

7.8.5.2 用法示例

例 44_qundoview，仿 PS 历史记录（难度：一般）。如果大家学习过 PS，都知道 PS 里有个历史记录面板，点击就会撤回到历史记录的步骤。例子其实也可以参考 Qt 官方提供的例子“Undo Framework Example”。但是官方的例子过于复杂，整个理解下来需要花费大量时间。于是笔者写一个仿 PS 历史记录的例子来加深大家对 QUndoView 的理解。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。同时添加了两个新建文件》点击新建，选择 C++》C++ Source File 命名 command.h 和 command.cpp 文件，用于重写 QUndoCommand。项目新建完成，如下图。



在头文件“command.h”具体代码如下。

command.h 编程后的代码

```
1   #ifndef COMMAND_H
2   #define COMMAND_H
3
4   #include <QUndoCommand>
5   #include <QObject>
6
```

```
7   class addCommand : public QUndoCommand
8   {
9   public:
10      addCommand(int *value, QUndoCommand* parent = 0);
11      ~addCommand();
12
13      /* 重写重做与撤回方法 */
14      void redo() override;
15      void undo() override;
16
17  private:
18      /* 新的 count */
19      int *new_count;
20
21      /* 旧的 count */
22      int old_count;
23  };
24
25  #endif // COMMAND_H
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QUndoView>
6   #include <QUndoStack>
7   #include <QHBoxLayout>
8   #include <QVBoxLayout>
9   #include <QPushButton>
10  #include <QLabel>
11  #include <command.h>
12
13  class MainWindow : public QMainWindow
14  {
15      Q_OBJECT
16
17  public:
18      MainWindow(QWidget *parent = nullptr);
19      ~MainWindow();
20
21  private:
22      /* 水平布局 */
```

```
23      QHBoxLayout *hLayout;
24      /* 水平布局 */
25      QVBoxLayout *vLayout;
26      /* 用于容纳 hLayout 布局 */
27      QWidget *mainWidget;
28      /* 容器作用 QWidget,用于容纳标签与按钮 */
29      QWidget *widget;
30      /* 存放 QUndoCommand 命令的栈 */
31      QUndoStack *undoStack;
32      /* 历史记录面板 */
33      QUndoView *undoView;
34      /* 用于显示计算结果 */
35      QLabel *label;
36      /* 按钮 */
37      QPushButton *pushButton;
38      /* 计算结果 */
39      int count;
40
41  private slots:
42      void pushButtonClieked();
43      void showCountValue(int);
44  };
45  #endif // MAINWINDOW_H
```

在源文件“command.cpp”具体代码如下。

command.cpp 编程后的代码

```
1   #include "command.h"
2   #include <QDebug>
3
4   addCommand::addCommand(int *value, QUndoCommand *parent)
5   {
6       /* 使用 Q_UNUSED,避免未使用的数据类型 */
7       Q_UNUSED(parent);
8
9       /* undoView 显示的操作信息 */
10      setText("进行了加 1 操作");
11
12      /* value 的地址赋值给 new_count */
13      new_count = value;
14
15      /* 让构造函数传过来的*new_count 的值赋值给 old_count */
16      old_count = *new_count;
17  }
```

```
18
19  /* 执行 stack push 时或者重做操作时会自动调用 */
20  void addCommand::redo()
21  {
22      /* 重新赋值给 new_count */
23      *new_count = old_count;
24
25      /* 打印出*new_count 的值 */
26      qDebug()<<"redo:"<<*new_count<<endl;
27  }
28
29  /* 回撤操作时执行 */
30  void addCommand::undo()
31  {
32      /* 回撤操作每次应减一 */
33      (*new_count)--;
34
35      /* 打印出*new_count 的值 */
36      qDebug()<<"undo:"<<*new_count<<endl;
37  }
38
39  addCommand::~addCommand()
40  {
41
42  }
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2   #include <QDebug>
3
4   MainWindow::MainWindow(QWidget *parent)
5       : QMainWindow(parent)
6   {
7       /* 设置主窗体显示的位置与大小 */
8       this->setGeometry(0, 0, 800, 480);
9
10      /* 实例一个水平布局，用于左侧按钮区域与右侧历史记录面板 */
11      hLayout = new QHBoxLayout();
12
13      /* 实例一个水平布局，用于左侧标签与按钮 */
14      vLayout = new QVBoxLayout();
15
```

```
16      /* 主 Widget，因为 MainWindow 自带一个布局，
17       * 我们要新建一个 Widget 容纳新布局
18       */
19      mainWidget = new QWidget();
20
21      /* 用于存放命令行栈 */
22      undoStack = new QUndoStack(this);
23
24      /* 用于容纳左侧标签与按钮布局 */
25      widget = new QWidget();
26
27      /* 历史记录面板实例化 */
28      undoView = new QUndoView(undoStack);
29
30      /* 实例一个按钮，用于加一操作 */
31      pushButton = new QPushButton();
32
33      /* 标签，用于显示计算结果 */
34      label = new QLabel();
35
36      /* 设置 widget 的大小 */
37      widget->setMinimumSize(400, 480);
38
39      /* 将两个 widget 添加到水平布局 */
40      hLayout->addWidget(widget);
41      hLayout->addWidget(undoView);
42
43      /* 初始化 count 的值 */
44      count = 0;
45
46      /* 显示初始化计算结果 */
47      label->setText("计算结果: " + QString::number(count));
48      label->setAlignment(Qt::AlignCenter);
49
50      /* 左侧布局 */
51      vLayout->addWidget(label);
52      vLayout->addWidget(pushButton);
53
54      /* 左侧布局控件的高度设置 */
55      label->setMaximumHeight(this->height() / 5);
56      pushButton->setMaximumHeight(this->height() / 5);
57
```

```
58      /* 按钮文件设置 */
59      pushButton->setText("加 1");
60
61      /* 设置 widget 的布局为 vLayout */
62      widget->setLayout(vLayout);
63
64      /* 将主窗体的布局设置为 hLayout */
65      mainWidget->setLayout(hLayout);
66
67      /* 设置 mainWidget 为主窗体的居中 widget */
68      this->setCentralWidget(mainWidget);
69
70      /* 信号槽连接，按钮点击，执行加一操作 */
71      connect(pushButton, SIGNAL(clicked()), this,
72              SLOT(pushButtonClieked()));
73
74      /* 信号槽连接，历史记录项 index 发生变化，显示 count 大小 */
75      connect(undoStack, SIGNAL(indexChanged(int) ),
76              this, SLOT(showCountValue(int)));
77  }
78
79  /* 入栈操作会自动调用 addCommand 的 redo */
80  void MainWindow::pushButtonClieked()
81  {
82      /* 变量值加一 */
83      count++;
84
85      /* value 指向 count 的地址 */
86      int *value = &count;
87
88      /* 用重写的 addCommand 类实例化 */
89      QUndoCommand *add = new addCommand(value);
90
91      /* 入栈 */
92      undoStack->push(add);
93  }
94
95  void MainWindow::showCountValue(int)
96  {
97      /* 标签用于显示计算结果 */
98      label->setText("计算结果: " + QString::number(count));
99  }
```

```
100
101 MainWindow::~MainWindow()
102 {
103
104 }
```

在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

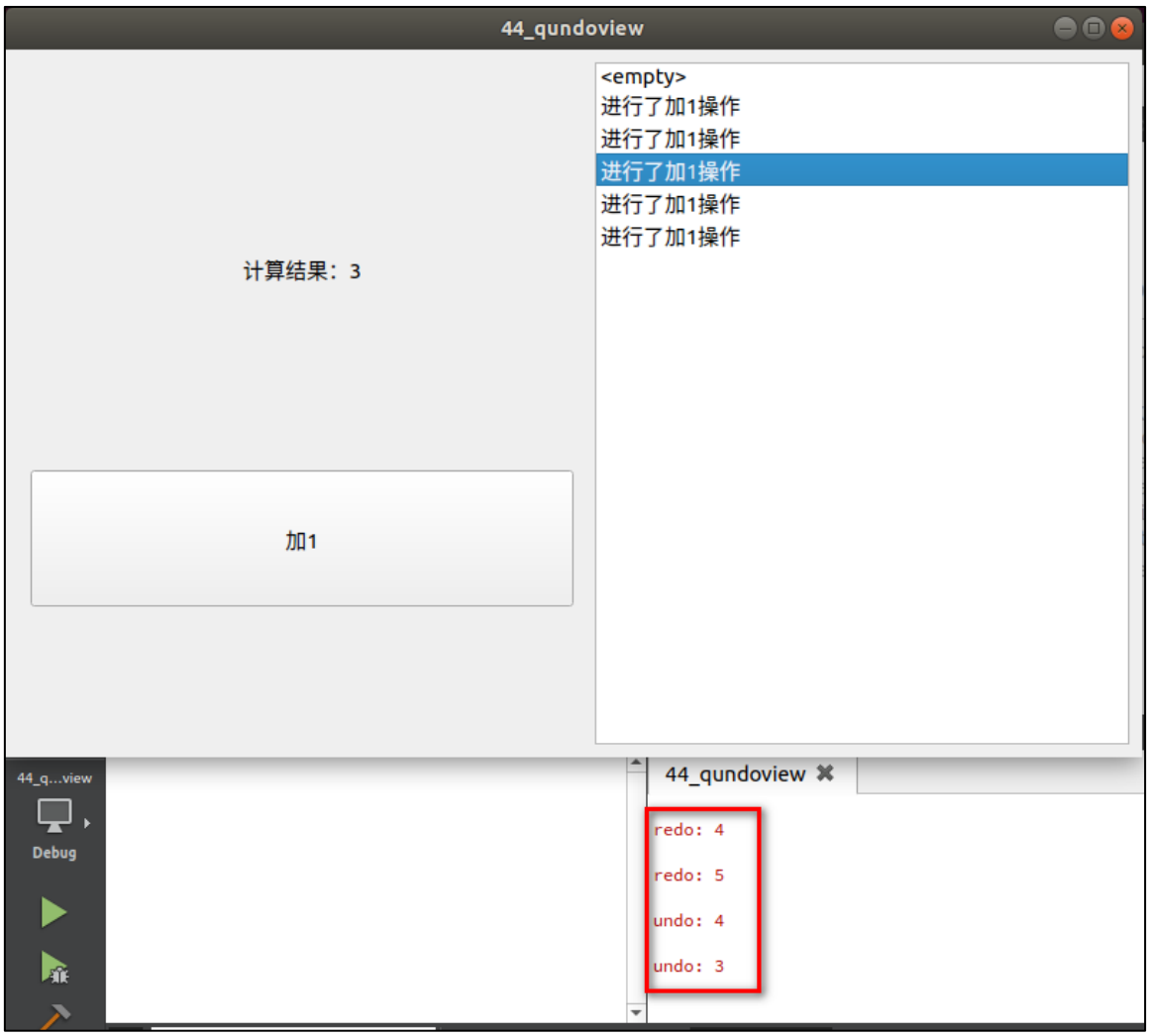
main.cpp 编程后的代码

```
1  #include "mainwindow.h"
2
3  #include <QApplication>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8      MainWindow w;
9      w.show();
10     return a.exec();
11 }
```

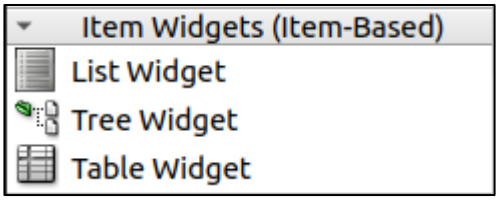
7.8.5.3 运行效果

程序编译运行的结果如下。点击“加 1”按钮，计算结果将加 1，用方向键或者鼠标进行选择右边的历史记录面板将进行重做或者回撤操作，同时应用程序输出窗口打印出 debug 信息，计算结果也将回到该步时的计算结果。

本例总结：使用数学运算加一的方法，简单的说明了 QUndoView 的使用。如果大家想到有好 QUndoView 的使用情景就可以模仿本例再深度优化或者进行知识拓展。



7.9 项目控件组（基于项）



在上一小节学习过视图组，下面学习控件组。仔细观察视图组里的某几个控件与控件组的控件名字相似。以 QListWidget 为例，QListWidget 就是继承 QListView。QListView 是基于模型的，而 QListWidget 是基于项的。两种控件在不同的场合可以酌情选择使用!一般处理大数据使用基于模型的多。视图组与控件组的控件在 Qt 里展示数据时是会经常使用的！大家要掌握它们的使用方法。

以上各个控件的解释如下：

（1） List Widget:清单控件

（2） TreeWidget:树形控件

（3） Table Widget:表控件

以下是各个控件的简介：

QListWidget 继承 QListView。QListWidget 类提供了一个基于项的列表小部件。QListWidget 是一个便捷的类，它提供了一个类似于 QListView 提供的列表视图，但是提供了一个用于添加和删除项目的基于项目的经典接口。QListWidget 使用内部模型来管理列表中的每个 QListWidgetItem。QListView 是基于 model 的，需要自己来建模（例如建立 QStringListModel、QSqlTableModel 等），保存数据，这样就大大降低了数据冗余，提高了程序的效率，但是需要我们对数据建模有一定了解，而 QListWidget 是一个升级版本的 QListView，它已经自己为我们建立了一个数据存储模型（QListWidgetItem），操作方便，直接调用 addItem 即可添加项目（ICON，文字）。

QTreeWidget 继承 QTreeView。QTreeWidget 类提供了一个使用预定义树模型的树视图。QTreeWidget 类是一个便捷的类，它提供了一个标准的树小部件，具有一个类似于 qt3 中的 QListView 类所使用的基于项目的经典接口。该类基于 Qt 的模型/视图体系结构，并使用默认模型来保存项，每个项都是 QTreeWidgetItem。

QTableWidget 继承 QTableView。QTableWidget 类提供了一个带有默认模型的基于项的表视图。表小部件为应用程序提供标准的表显示工具。QTableWidget 中的项由 QTableWidgetItem 提供。

7.9.1 QListWidget

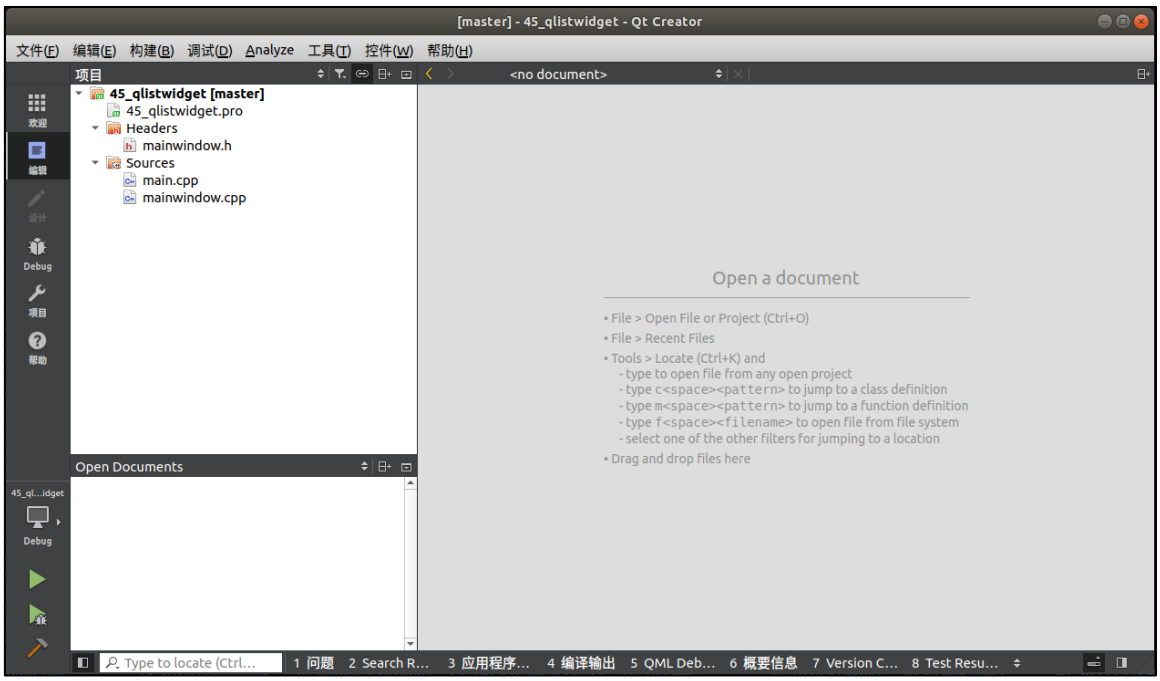
7.9.1.1 控件简介

QListWidget 继承 QListView。QListWidget 类提供了一个基于项的列表小部件。QListWidget 是一个便捷的类，它提供了一个类似于 QListView（下一小节将讲到）提供的列表视图，但是提供了一个用于添加和删除项目的基于项目的经典接口。QListWidget 使用内部模型来管理列表中的每个 QListWidgetItem。

7.9.1.2 用法示例

例 45_qlistwidget，添加“歌曲”（难度：简单）。本例使用一个 QListWidget 以及一个按钮，当单击按钮时，就会调用系统打开文件窗口，过滤 mp3 后缀的文件（本例使用 touch 指令创建 2 个 mp3 后缀的文件，并不是真正的歌曲，在终端输入指令为 touch 0.mp3 1.mp3，本例在项目下已经创建了两个以 mp3 为后缀的文件），当打开系统文件选择框时，就会选择这两个 mp3 文件作为 QListWidget 的项添加到 QListWidget 的窗口中。（PS：我们写音乐播放器就要用到这种操作—打开歌曲。实际本例就是一个打开歌曲的代码部分。）

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成，如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QListWidget>
6  #include <QPushButton>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15
16 private:
17     /* 声明对象 */
18     QListWidget *listWidget;
19     QPushButton *pushButton;
20
21 private slots:
22     void pushButtonClicked();
23
24 };
25 #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1  #include "mainwindow.h"
2  #include "QFileDialog"
3
4  MainWindow::MainWindow(QWidget *parent)
5      : QMainWindow(parent)
6  {
7      /* 设置主窗口的显示位置与大小 */
8      this->setGeometry(0, 0, 800, 480);
9
10     listWidget = new QListWidget(this);
11
12     /* 设置 listWidget 的大小 */
13     listWidget->setGeometry(0, 0, 480, 480);
14
15     listWidget->addItem("请单击右边的添加项添加内容");
16
17     pushButton = new QPushButton(this);
18
19     /* 设置 pushButton 的位置与大小 */
20     pushButton->setGeometry(540, 200, 200, 100);
21     pushButton->setText("添加项");
22
23     /* 信号与槽连接 */
24     connect(pushButton, SIGNAL(clicked()),
25             this, SLOT(pushButtonClicked()));
26 }
27
28 void MainWindow::pushButtonClicked()
29 {
30     /* 调用系统打开文件窗口，设置窗口标题为“打开文件”，过滤文件名 */
31     QString fileName = QFileDialog::getOpenFileName(
32                 this,tr("添加项"),"",
33                 tr("Files(*.mp3)")
34                 );
35
36     /* 判断是否选中打开 mp3 文件 */
37     if (fileName != NULL)
38         /* 添加项到列表中 */
39         listWidget->addItem(fileName);
40 }
```

```
41
42  MainWindow::~MainWindow()
43  {
44  }
```

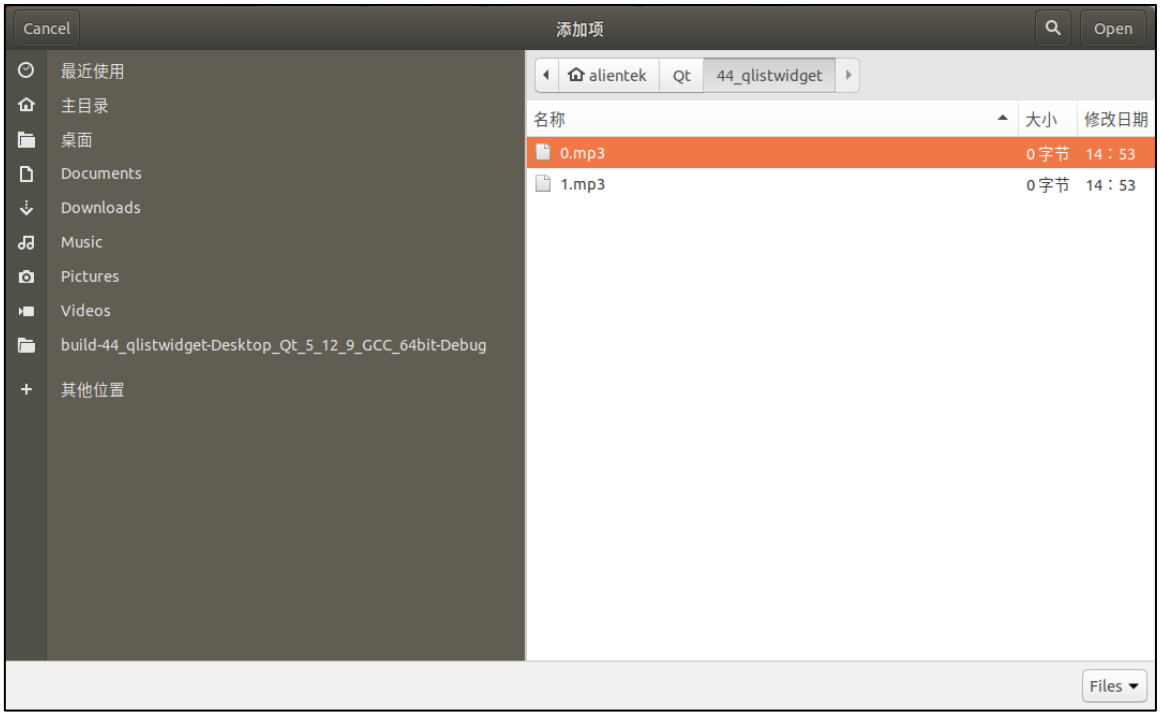
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.9.1.3 运行效果

程序编译运行的结果如下。当点击添加项按钮时出现系统选择文件的对话框，系统打开文件时会过滤 mp3 后缀的文件，点击后缀为 mp3 的文件，双击或者选择后再点击右上角的“Open”的打开来把这个文件添加到左边的 QListWidget 列表中。读者可以模仿这个示例，还可以添加删除项的按钮，或者删除全部按钮等，酷狗音乐播放器的的歌单与此类似。后续继续学习如何播放音乐，与此结合，就可以做出一款音乐播放器了。



7.9.2 QTreeWidget

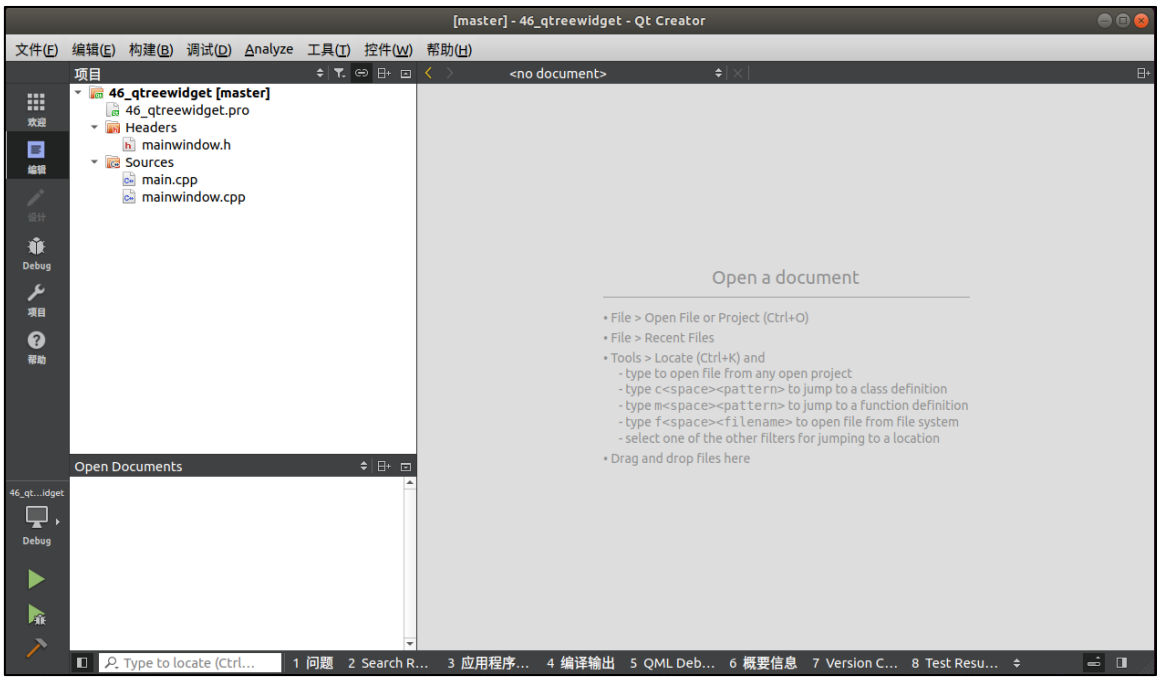
7.9.2.1 控件简介

QTreeWidget 继承 QTreeView。QTreeWidget 类提供了一个使用预定义树模型的树视图。QTreeWidget 类是一个便捷的类，它提供了一个标准的树小部件，具有一个类似于 qt3 中的 QListView 类所使用的基于项目的经典接口。该类基于 Qt 的模型/视图体系结构，并使用默认模型来保存项，每个项都是 QTreeWidgetItem。

7.9.2.2 用法示例

例 46_qtreewidget，群发信息（难度：一般），本例使用一个 TreeWidget，模拟成一个飞信联系人分组，通过选中组内联系人来“群发”信息。实际并不是真正做一个群发信息的飞信，只是模拟飞信群发信息时选择联系人的场景，通过例子来熟悉 QTreeWidget 的使用。本例相对前面的例子稍长，出现了树节点与子节点的概念。本例的思路：当选中顶层的树形节点时，子节点全部被选中；当取消选择顶层树形节点时，子节点原来选中的状态将全部取消；当不完全选中子节点时，树节点显示为半选状态。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成，如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QTreeWidget>
6  #include <QTreeWidgetItem>
7
8  class MainWindow : public QMainWindow
9  {
10     Q_OBJECT
11
12 public:
13     MainWindow(QWidget *parent = nullptr);
14     ~MainWindow();
15
16 private:
17     /* QTreeWidget 对象 */
18     QTreeWidget *treeWidget;
19     /* 顶层树节点 */
20     QTreeWidgetItem *parentItem;
21     /* 声明三个子节点 */
22     QTreeWidgetItem *subItem[3];
23
24     /* 子节点处理函数 */
25     void updateParentItem(QTreeWidgetItem*);
```

```
26
27  private slots:
28      /* 槽函数 */
29      void treeItemChanged(QTreeWidgetItem*, int);
30
31  };
32  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗体大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 实例化 */
10      treeWidget = new QTreeWidget(this);
11
12      /* 居中 */
13      setCentralWidget(treeWidget);
14
15      /* 清空列表 */
16      treeWidget->clear();
17
18      /* 实例化顶层树节点 */
19      parentItem = new QTreeWidgetItem(treeWidget);
20      parentItem->setText(0, "同事");
21
22      parentItem->setFlags(
23                Qt::ItemIsUserCheckable
24                | Qt::ItemIsEnabled
25                | Qt::ItemIsSelectable
26                );
27      /* 树节点设置为未选中 */
28      parentItem->setCheckState(0, Qt::Unchecked);
29
30      /* 字符串链表 */
31      QList <QString> strList;
32      strList<<"关羽"<<"刘备"<<"张飞";
33
```

```
34      for (int i = 0; i < 3; i++){
35          /* 实例化子节点 */
36          subItem[i] = new QTreeWidgetItem(parentItem);
37          /* 设置子节点的文本，参数 0 代表第 0 列 */
38          subItem[i]->setText(0, strList[i]);
39          /* 设置子节点的属性为用户可选、项开启、项可选 */
40          subItem[i]->setFlags(
41                      Qt::ItemIsUserCheckable
42                      | Qt::ItemIsEnabled
43                      | Qt::ItemIsSelectable
44                      );
45          /* 设置子节点的状态为未选中 */
46          subItem[i]->setCheckState(0,Qt::Unchecked);
47      }
48      /* 信号槽连接 */
49      connect(treeWidget,SIGNAL(itemChanged(QTreeWidgetItem* , int)),
50              this, SLOT(treeItemChanged(QTreeWidgetItem* , int)));
51
52  }
53
54  /* 更新树节点函数 */
55  void MainWindow::updateParentItem(QTreeWidgetItem *item)
56  {
57      /* 获取子节点的父节点（树节点） */
58      QTreeWidgetItem* parent = item->parent();
59      if(parent == NULL){
60          return;
61      }
62      /* 初始化选中的数目为 0，下面根据 selectCount 来判断树节点的状态 */
63      int selectCount = 0;
64      /* 获取树节点的子节点总数 */
65      int childCount = parent->childCount();
66      /* 循环判断子节点的状态 */
67      for(int i = 0; i < childCount; i ++){
68          QTreeWidgetItem* childItem =parent->child(i);
69          /* 判断当前子节点的状是否为选中状态，如果是，则加一 */
70          if(childItem->checkState(0) == Qt::Checked) {
71              selectCount ++;
72          }
73      }
74      /* 根据 selectCount 来判断树节点的状态 */
75      /* 当选中的子节点小于或等于 0 时，则为设置树节点为未选中状态 */
```

```
76      if (selectCount <= 0) {
77          /* 设置树节点为未选中状态 */
78          parent->setCheckState(0, Qt::Unchecked);
79          /* 部分选中时，树节点为半选状态 */
80      } else if (selectCount > 0 && selectCount < childCount) {
81          /* 设置为半选状态 */
82          parent->setCheckState(0, Qt::PartiallyChecked);
83          /* 子节点全选时 */
84      } else if (selectCount == childCount){
85          /* 设置为树节点为选中状态 */
86          parent->setCheckState(0, Qt::Checked);
87      }
88  }
89
90  void MainWindow::treeItemChanged(QTreeWidgetItem *item, int)
91  {
92      /* 获取子节点总数 */
93      int count = item->childCount();
94
95      /* 若顶层树节点选中 */
96      if(Qt::Checked == item->checkState(0) ) {
97          /* 若选中的项是树节点，count 会大于 0，否则选中的项是子节点 */
98          if (count > 0) {
99              for (int i = 0; i < count; i++) {
100                 /* 子节点全选 */
101                 item->child(i)->setCheckState(0, Qt::Checked);
102             }
103         } else {
104             /* 子节点处理 */
105             updateParentItem(item);
106         }
107         /* 若顶层树节点取消选中时 */
108     } else if (Qt::Unchecked == item->checkState(0)) {
109         if (count > 0){
110             /* 若选中的项是树节点，count 会大于 0，否则选中的项是子节点 */
111             for (int i = 0; i < count; i++) {
112                 /* 子节点全不选 */
113                 item->child(i)->setCheckState(0, Qt::Unchecked);
114             }
115         } else {
116             /* 子节点处理 */
117             updateParentItem(item);
```

```
118         }
119     }
120 }
121
122 MainWindow::~MainWindow()
123 {
124 }
```

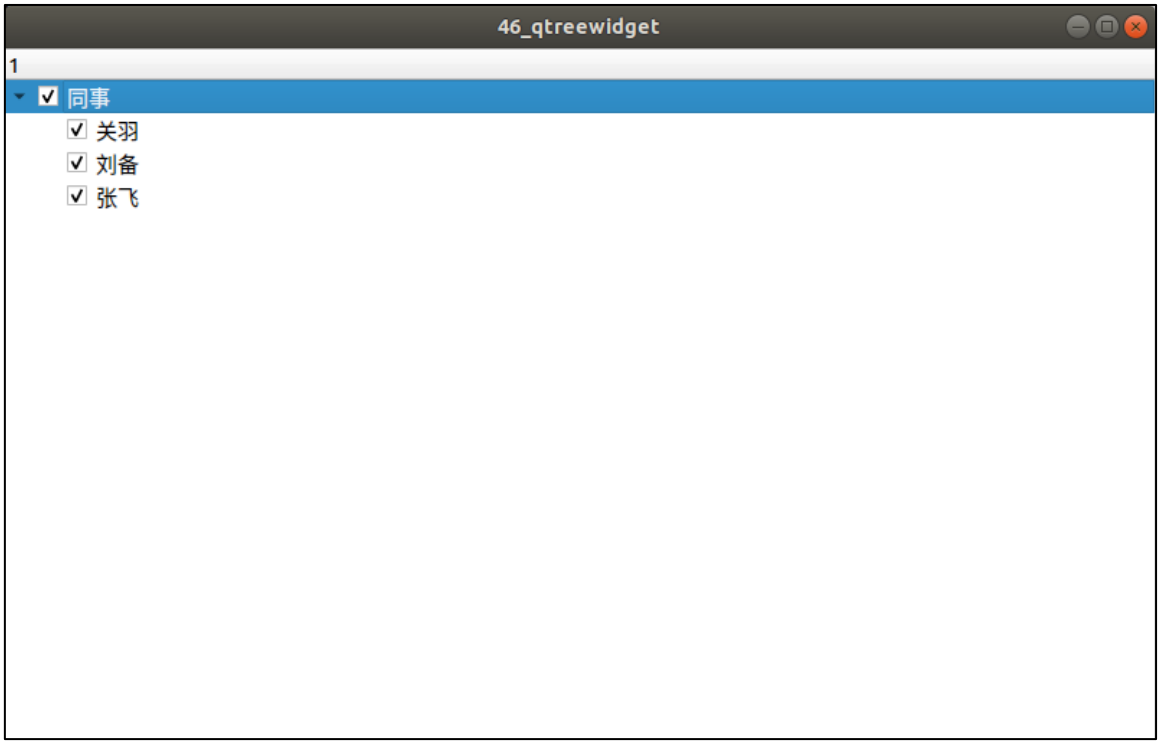
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8       MainWindow w;
9       w.show();
10      return a.exec();
11  }
```

7.9.2.3 运行效果

程序编译运行的结果如下。下图为全选时的状态，好比要群发信息时全选联系人的场景。当选中树节点同事时，子节点（关羽、刘备、张飞）将全选；当树节点同事未选中时，子节点（关羽、刘备、张飞）的状态为未选中；当子节点（关羽、刘备、张飞）选中且不全选时，树节点同事的状态将为半选状态。



7.9.3 QTableWidget

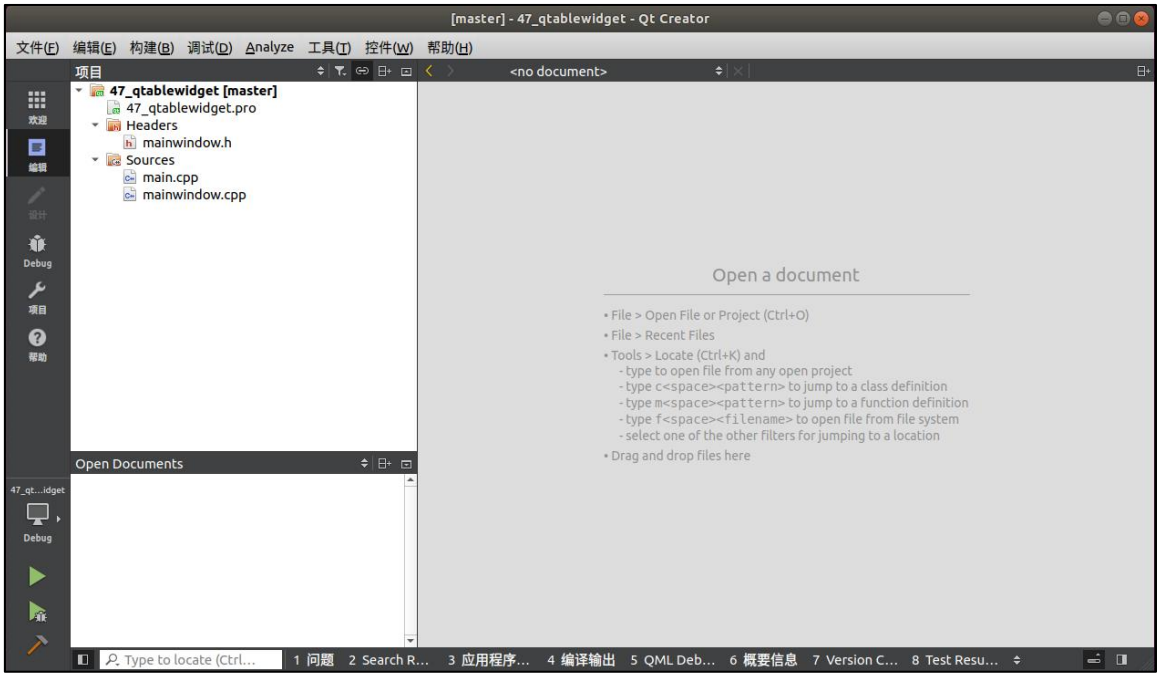
7.9.3.1 控件简介

QTableWidget 继承 QTableView。QTableWidget 类提供了一个带有默认模型的基于项的表视图。表小部件为应用程序提供标准的表显示工具。QTableWidget 中的项由 QTableWidgetItem 提供。

7.9.3.2 用法示例

例 47_qtablewidget，TabelWidget 表格（难度：简单），本例使用一个 TableWidget，绘制一个表格，同时修改项的标题，在表格里可以直接通过双击进行编辑项里的内容，也可以删除项里的内容等。

在新建例程中不要勾选“Generate form”，默认继承 QMainWindow 类即可。项目新建完成，如下图。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QTableWidget>
6
7  class MainWindow : public QMainWindow
8  {
9      Q_OBJECT
10
11 public:
12     MainWindow(QWidget *parent = nullptr);
13     ~MainWindow();
14
15 private:
16     /* QTabelWidget 表格 */
17     QTableWidget *tableWidget;
18
19     /* QTabelWidgetItem 表格数据（项） */
20     QTableWidgetItem *tableWidgetItem[4];
21
22 };
23 #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    /* 设置主窗体的大小与位置 */
    this->setGeometry(0, 0, 800, 480);

    /* 实例化 */
    tableWidget = new QTableWidget(this);
    /* 设置 tableWidget 表居中 */
    setCentralWidget(tableWidget);
    /* 设置列数 */
    tableWidget->setColumnCount(2);
    /* 设置行数 */
    tableWidget->setRowCount(2);
    /* 使用标签设置水平标题标签 */
    tableWidget->setHorizontalHeaderLabels(
                QStringList()<<"姓名"<<"性别"
                );

    /* 字符串类型链表 */
    QList <QString> strList;
    strList<<"小明"<<"小红"<<"男"<<"女";

    for (int i = 0; i < 4; i++) {
        /* 实例化 */
        tableWidgetItem[i] = new QTableWidgetItem(strList[i]);
        /* 设置文本居中对齐 */
        tableWidgetItem[i]->setTextAlignment(Qt::AlignCenter);
    }
    /* 插入数据，表的 index 就是一个二维数组数据 */
    tableWidget->setItem(0, 0, tableWidgetItem[0]);
    tableWidget->setItem(1, 0, tableWidgetItem[1]);
    tableWidget->setItem(0, 1, tableWidgetItem[2]);
    tableWidget->setItem(1, 1, tableWidgetItem[3]);

}

MainWindow::~MainWindow()
{
}
```

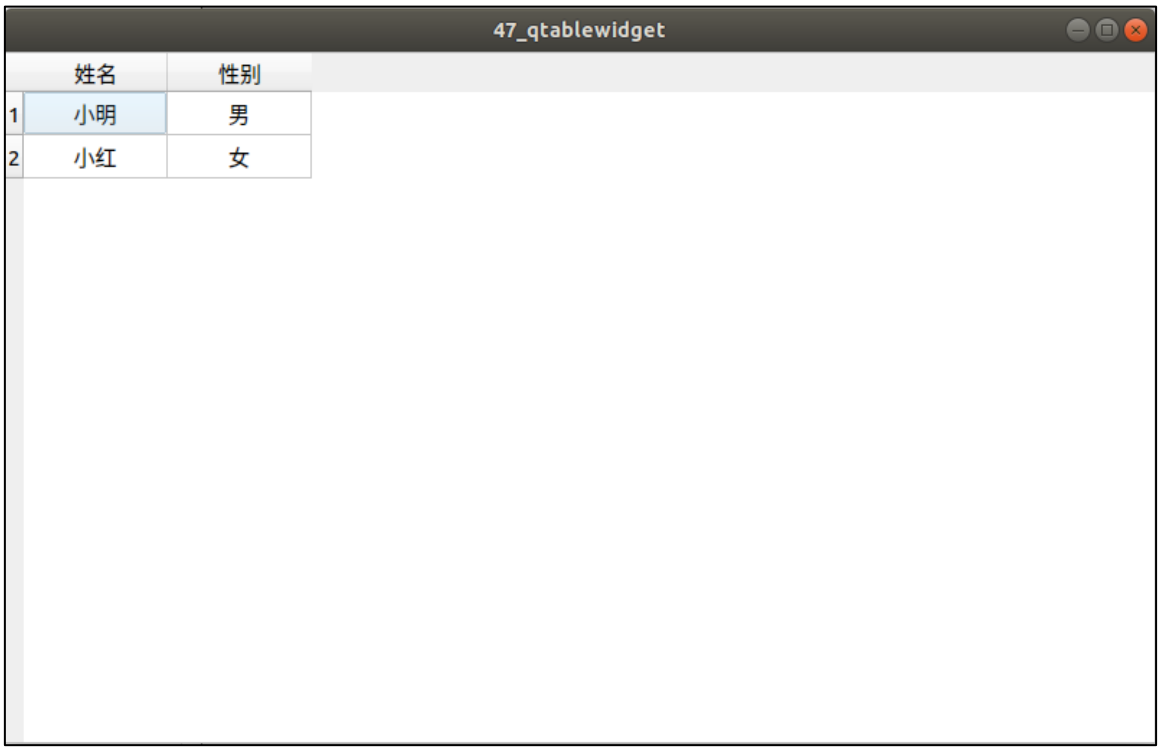
在源文件“main.cpp”具体代码如下。由新建项目时生成，无改动。

main.cpp 编程后的代码

```
1  #include "mainwindow.h"
2
3  #include <QApplication>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8      MainWindow w;
9      w.show();
10     return a.exec();
11 }
```

7.9.3.3 运行效果

程序编译运行的结果如下。双击表格中的项，可修改表格的内容，同时也可以删除内容等。



第二篇 提高篇

在前面我们已经讲解完 Qt Designer 里自带的控件了。本篇开始讲解 Qt 的一些常用类或常用模块。这对我们开发有很大的帮助。包括文件读写操作，绘图，图表的使用，多线程，网络编程，多媒体，及数据库的使用。这些都是嵌入式里常用的类或模块，我们需要掌握它们的使用方法。

第八章 文本读写

在很多时候我们需要读写文本文件进行读写，比如写个 Mp3 音乐播放器需要读 Mp3 歌词里的文本，比如修改了一个 txt 文件后保存，就需要对这个文件进行读写操作。本章介绍简单的文本文件读写，内容精简，让大家了解文本读写的基本操作。

8.1 QFile 读写文本

QFile 类提供了读取和写入文件的接口。在嵌入式里如果需要读写文件，最简单的方法就是用 Qfile，在第 14.2 和 14.3 小节就是利用了 QFile 来读写 Linux 下的字符设备（可把字符设备当作一个文本处理，linux 下一切皆文件），虽然只是写‘0’或‘1’，但也是对文件（文本）的读写了。

QFile 是一个读写文本、二进制文件和资源的 I/O 设备。QFile 可以自己使用，也可以更方便地与 QTextStream 或 QDataStream 一起使用。

文件名通常在构造函数中传递，但它可以在任何时候使用 setFileName()设置。不支持使用其他分隔符(例如'\')。所以在 Windows、 Linux 或者 Mac 里文件的路径都是用'/'。不能看到 Windows 的路径是'\'，我们就可以在写入的文件路径里添加这个'\'。不管操作系统是什么，QFile 的文件分隔符都是'/'。

可以使用 exists()检查文件是否存在，并使用 remove()删除文件。(更高级的文件系统相关操作由 QFileInfo 和 QDir 提供。)用 open()打开文件，用 close()关闭文件，用 flush()刷新文件。通常使用 QDataStream 或 QTextStream 读写数据，但也可以调用 QIODevice 继承的函数 read()、readLine()、readAll()、write()。QFile 还继承 getChar()、putChar()和 ungetChar()，它们一次只处理一个字符。文件的大小由 size()返回。可以使用 pos()获取当前文件位置，也可以使用 seek()移动到新的文件位置。如果已经到达文件的末尾，则 atEnd()返回 true。

QFile::open()函数打开文件时需要传递 QIODevice::OpenModeFlag 枚举类型的参数，决定文件以什么方式打开，QIODevice::OpenModeFlag 类型的主要取值如下：

- QIODevice::ReadOnly：以只读方式打开文件，用于载入文件。
- QIODevice::WriteOnly：以只写方式打开文件，用于保存文件。
- QIODevice::ReadWrite：以读写方式打开。
- QIODevice::Append：以添加模式打开，新写入文件的数据添加到文件尾部。
- QIODevice::Truncate：以截取方式打开文件，文件原有的内容全部被删除。
- QIODevice::Text：以文本方式打开文件，读取时“\n”被自动翻译为换行符，写入时字符串结束符会自动翻译为系统平台的编码，如 Windows 平台下是“\r\n”。

这些取值可以组合，例如 QIODevice::ReadOnly | QIODevice::Text 表示以只读和文本方式打开文件。

使用 QFile 对一个文本文件的操作流程是以下这样的。

8.1.1 应用实例

例 01_qfile，读写文本（难度：简单）。项目路径为 Qt/2/01_qfile。2 是代表第二篇的例程父目录。

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   01_qfile
* @brief        mainwindow.h
* @author       Deng Zhimao
* @email        1252699831@qq.com
* @net          www.openedv.com
* @date         2021-03-27
*****************************************************************/
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTextEdit>
#include <QFile>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPushButton>

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    /* 用于读取文件后显示 */
    QTextEdit *textEdit;

    /* QFile 类型对象 */
    QFile file;

    /* 水平布局 */
    QHBoxLayout *hBoxLayout;

    /* 垂直布局 */
    QVBoxLayout *vBoxLayout;

    /* 水平布局 Widget */
```

```
33      QWidget *hWidget;
34
35      /* 垂直布局 Widget */
36      QWidget *vWidget;
37
38      /* 打开文件按钮 */
39      QPushButton *openPushButton;
40
41      /* 关闭文件按钮 */
42      QPushButton *closePushButton;
43
44  private slots:
45
46      /* 打开文本文件 */
47      bool openFile();
48
49      /* 关闭文本文件 */
50      void closeFile();
51  };
52  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*********************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   01_qfile
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-03-27
*********************************************************************/
#include "mainwindow.h"
#include <QFileDialog>
#include <QDebug>

1   MainWindow::MainWindow(QWidget *parent)
2       : QMainWindow(parent)
3   {
4       /* 设置窗口的位置与大小 */
5       this->setGeometry(0, 0, 800, 480);
6
7       /* 布局设置 */
8       textEdit = new QTextEdit();
```

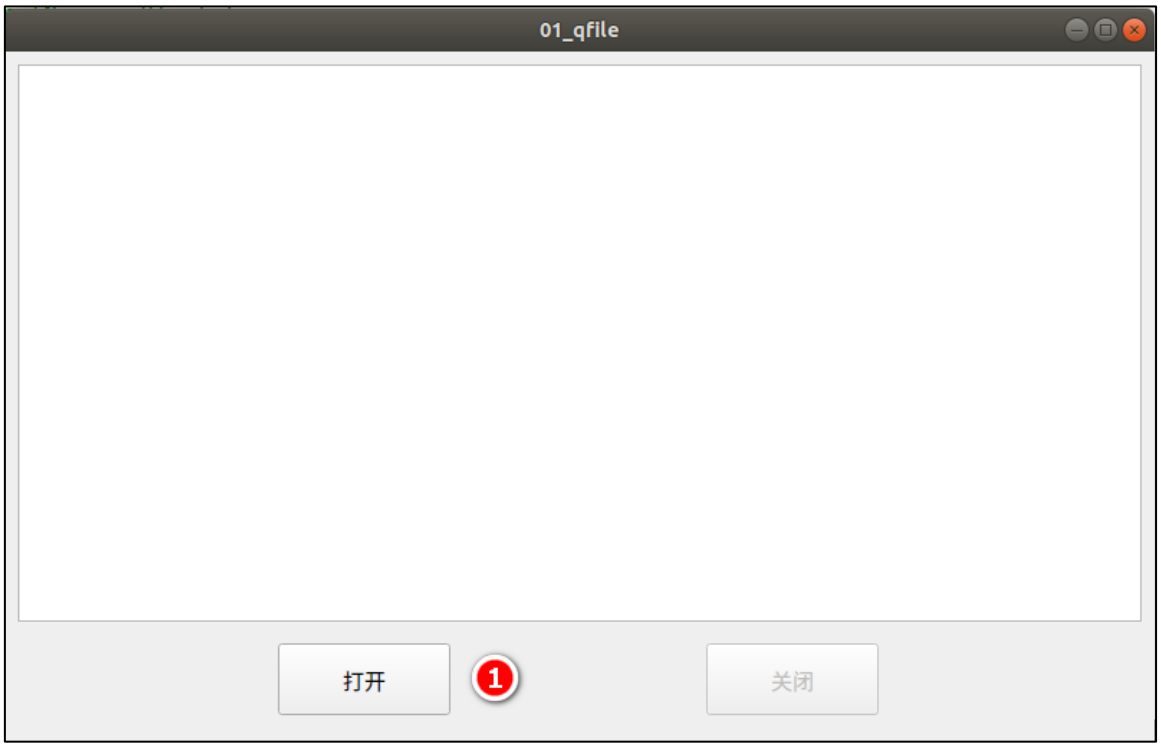
```
9       vBoxLayout = new QVBoxLayout();
10      hBoxLayout = new QHBoxLayout();
11      vWidget = new QWidget();
12      hWidget = new QWidget();
13      openPushButton = new QPushButton();
14      closePushButton = new QPushButton();
15
16      /* 设置两个按钮的大小 */
17      openPushButton->setMinimumHeight(50);
18      openPushButton->setMaximumWidth(120);
19      closePushButton->setMinimumHeight(50);
20      closePushButton->setMaximumWidth(120);
21
22      /* 设置两个按钮的文本 */
23      openPushButton->setText("打开");
24      closePushButton->setText("关闭");
25
26      /* 设置关闭按钮为不可用属性，需要打开文件才设置为可用属性 */
27      closePushButton->setEnabled(false);
28
29      /* 水平布局 */
30      hBoxLayout->addWidget(openPushButton);
31      hBoxLayout->addWidget(closePushButton);
32      hWidget->setLayout(hBoxLayout);
33
34      /* 垂直布局 */
35      vBoxLayout->addWidget(textEdit);
36      vBoxLayout->addWidget(hWidget);
37      vWidget->setLayout(vBoxLayout);
38
39      /* 居中 */
40      setCentralWidget(vWidget);
41
42      /* 信号槽连接 */
43      connect(openPushButton, SIGNAL(clicked()),
44              this, SLOT(openFile()));
45      connect(closePushButton, SIGNAL(clicked()),
46              this, SLOT(closeFile()));
47  }
48
49  MainWindow::~MainWindow()
50  {
51  }
```

```
52
53  bool MainWindow::openFile()
54  {
55      /* 获取文件的路径 */
56      QString fileName = QFileDialog::getOpenFileName(this);
57
58      /* 指向文件 */
59      file.setFileName(fileName);
60
61      /* 判断文件是否存在 */
62      if (!file.exists())
63          return false;
64
65      /* 以读写的方式打开 */
66      if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
67          return false;
68
69      /* 读取文本到 textEdit */
70      textEdit->setPlainText(file.readAll());
71
72      /* 设置打开按钮不可用，需要关闭再打开 */
73      openPushButton->setEnabled(false);
74
75      /* 设置关闭按钮为可用属性 */
76      closePushButton->setEnabled(true);
77
78      /* 关闭文件 */
79      file.close();
80
81      return true;
82  }
83
84  void MainWindow::closeFile()
85  {
86      /* 检测打开按钮是否可用，不可用时，说明已经打开了文件 */
87      if (!openPushButton->isEnabled()) {
88          /* 获取 textEdit 的文本内容 */
89          QString str = textEdit->toPlainText();
90
91          /* 以只读的方式打开 */
92          if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
93              return;
94
```

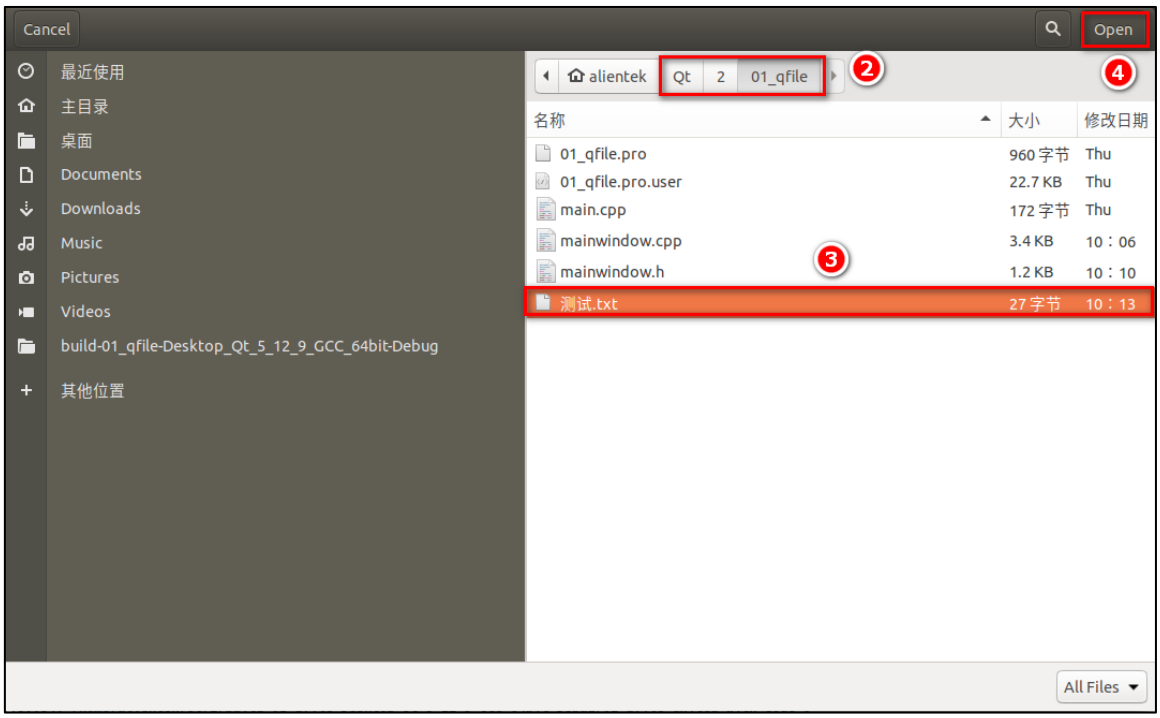
```
95          /* 转换为字节数组 */
96          QByteArray strBytes = str.toUtf8();
97
98          /* 写入文件 */
99          file.write(strBytes, strBytes.length());
100
101         /* 清空 textEdit 的显示内容 */
102         textEdit->clear();
103
104         /* 关闭文件 */
105         file.close();
106
107         /* 重新设置打开和关闭按钮的属性 */
108         openPushButton->setEnabled(true);
109         closePushButton->setEnabled(false);
110     }
111 }
```

8.1.2 程序运行效果

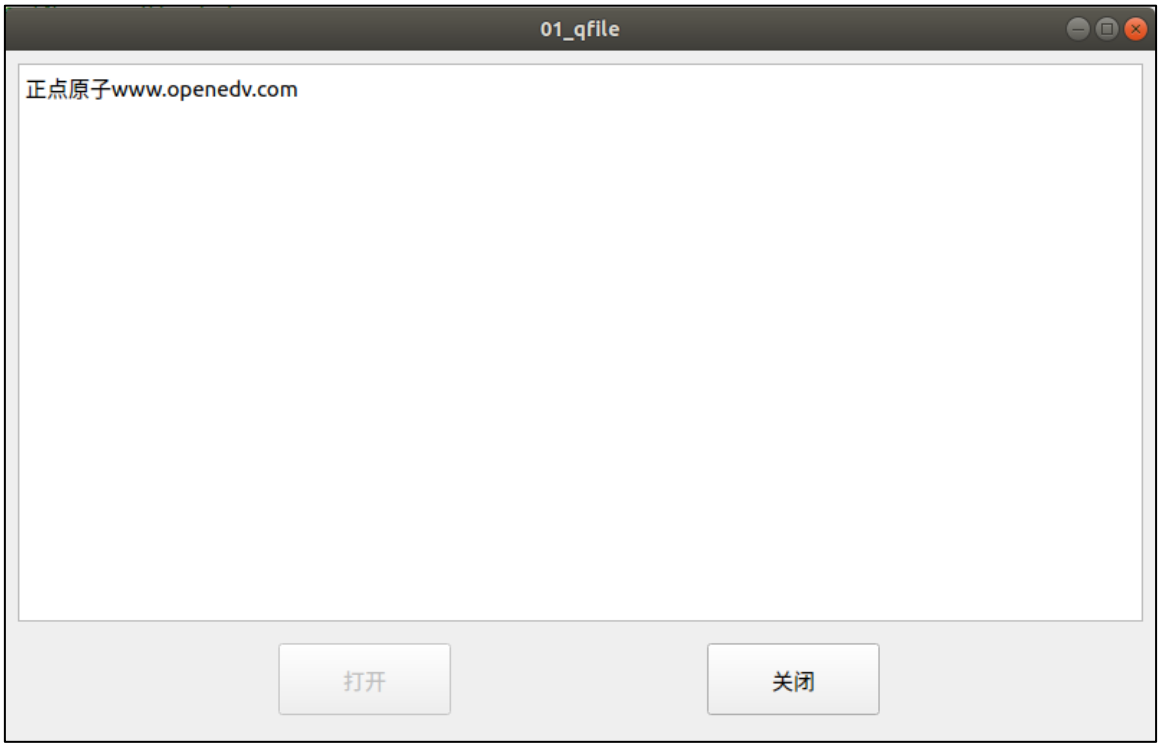
点击打开。



调用系统打开文件的窗口。选择项目路径下的“测试.txt”。



打开后，文本的内容如下，可以进行修改，修改后点击关闭就会写入到此文件里。本例仅仅用两个按钮和一个文本编辑框完成，内容简洁易懂。但是在实际项目里不是用 QPushButton 来做打开文件和关闭文件的，一般设计于在菜单栏里用 QAction 来做。包括添加复制、粘贴、另存为、关闭、等等。可以仿照 Windows 里的记事本，用 Qt 写一个类似的软件完全可以。



8.2 QTextStream 读写文本

QTextStream 类为读写文本提供了一个方便的接口，常与 QFile 结合使用。QTextStream 可以在 QIODevice、QByteArray 或 QString 上操作。使用 QTextStream 的流操作符，您可以方便地读写单词、行和数字。为了生成文本，QTextStream 支持字段填充和对齐的格式化选项，以及数字的格式化。看到 Stream 这个名词就知道，它与流操作有关，那么我们可以使用 C++的操作符“<<”和“>>”(流提取运算符和流插入运算符)进行操作流了。

8.2.1 应用实例

例 02_qtextstream，文本流读写文本（难度：简单）。项目路径为 Qt/2/02_qtextstream。QTextStream 的例子与 QFile 的一样，只是在 QFile 的例子里加入了 QTextStream。下面只写出不同部分的代码。详细直接打开项目查看。

在源文件“mainwindow.cpp”具体代码如下。不同的地方已经用红色字体标出。

mainwindow.cpp 编程后的代码

```
53  bool MainWindow::openFile()
54  {
55      /* 获取文件的路径 */
56      QString fileName = QFileDialog::getOpenFileName(this);
57
58      /* 指向文件 */
59      file.setFileName(fileName);
60
61      /* 判断文件是否存在 */
62      if (!file.exists())
63          return false;
64
65      /* 以读写的方式打开 */
66      if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
67          return false;
68
69      /* 使用文本流读取文件 */
70      QTextStream stream(&file);
71
72      /* 读取文本到 textEdit */
73      textEdit->setPlainText(stream.readAll());
74
75      /* 设置打开按钮不可用，需要关闭再打开 */
76      openPushButton->setEnabled(false);
77
78      /* 设置关闭按钮为可用属性 */
79      closePushButton->setEnabled(true);
```

```
80
81      /* 关闭文件 */
82      file.close();
83
84      return true;
85  }
86
87  void MainWindow::closeFile()
88  {
89      /* 检测打开按钮是否可用，不可用时，说明已经打开了文件 */
90      if (!openPushButton->isEnabled()) {
91
92          /* 以只读的方式打开 */
93          if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
94              return;
95
96          /* 用文本流读取文件 */
97          QTextStream stream(&file);
98
99          /* 获取 textEdit 的文本内容，转为字符串 */
100         QString str = textEdit->toPlainText();
101
102         /* 使用流提取运算符，写入文本流 */
103         stream<<str;
104
105         /* 清空 textEdit 的显示内容 */
106         textEdit->clear();
107
108         /* 关闭文件 */
109         file.close();
110
111         /* 重新设置打开和关闭按钮的属性 */
112         openPushButton->setEnabled(true);
113         closePushButton->setEnabled(false);
114     }
115 }
```

8.2.2 程序运行效果

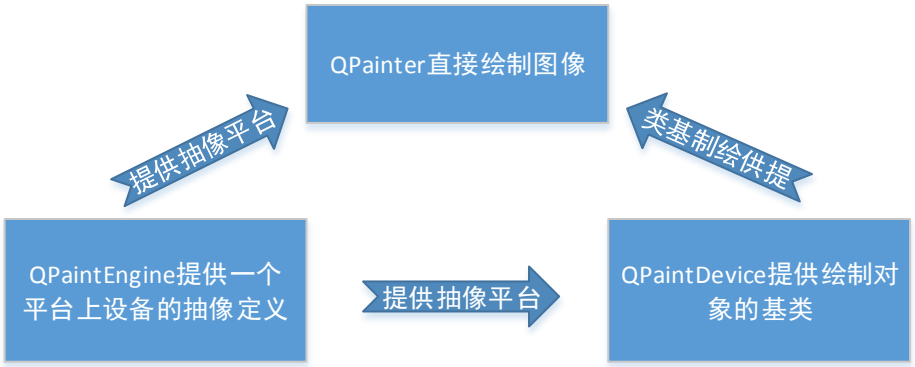
与上一小节（8.1.2 小节）一样。使用 QFile 与 QTextStream 感觉例子看上去没区别。主要是 QTextStream 还支持字段填充和对齐的格式化选项，例子没有体现出来而已，等我们用到一些特性时还是有区别的。

第九章 绘图与图表

绘图与图表在嵌入式里有的比较多，尤其是图表，我们常在股票里看到的“图表折线/曲线图/饼状图等”都可以用 Qt 的图表来实现。绘图和图表的内容本章主要介绍绘图和图表的基本操作，以简单的例子呈现绘图与图表的用法，目的就是快速入门绘图与图表，关于绘图与图表详解最好是看 Qt 官方的帮助文档。

9.1 QPainter 绘图

Qt 里的所有绘图，比如一个按钮和一个 Label 的显示，都有绘图系统来执行。绘图系统基于 QPainter、和 QPaintDevice 和 QPainEngine 类。QPainter 是可以直接用来操作绘图的类，而 QPaintDevice 和 QPainEngine 都比 QPainter 更底层，我们只需要了解一下 QPaintDevice 和 QPainEngine 就行了。可以用下面一张图来表示它们的关系。



一般用于显示的类，如 QWidget、QPixmap、QImage、Qlabel 等可视类控件都可以充当绘图区域的“画布”，从 QWidget 继承的类都有 virtual void paintEvent(QPaintEvent *event);属性。这个 paintEvent()是一个虚函数，它在 qwidget.h 头文件的 protected:修饰符下。

paintEvent()事件可以被重写。（解释：什么是绘图事件？可以这么理解，当界面初始化或者需要刷新时才会执行的事件，也就是说绘图事件在构造对象实例化时会执行，需要刷新界面我们可以使用 update()方法执行 paintEvent()事件）。

paintEvent()事件是父类 QWidget 提供给子类的接口，在父类里定义为空，所以可以说 paintEvent()事件就是专门给子类画图用的。

paintEvent()事件在子类重写的基本结构如下：

```
void Widget::paintEvent(QPaintEvent *)
{
    /* 指定画图的对象，this 代表是本 Widget */
    QPainter painter(this);
    // 使用 painter 在对象上绘图...
}
```

9.1.1 应用实例

本例目的：快速了解 paintEvent()事件的使用。

例 03_qpainter，旋转的 CD（难度：一般）。项目路径为 Qt/2/03_qpainter。本例使用一张 CD 图片，用 QPainter 在 paintEvent()将 CD 画在窗口的中心，并且每 100ms 旋转 1 度角度。所以 CD 看起来是旋转了的效果。

在头文件“mainwindow.h”具体代码如下。

```
                    mainwindow.h 编程后的代码
/*****************************************************************
```

```
   Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
   * @projectName   03_qpainter
   * @brief        mainwindow.h
   * @author       Deng Zhimao
   * @email        1252699831@qq.com
   * @net          www.openedv.com
   * @date         2021-03-29
   ****************************************************************/
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QPainter>
6  #include <QPaintEvent>
7  #include <QTimer>
8
9  class MainWindow : public QMainWindow
10 {
11     Q_OBJECT
12
13 public:
14     MainWindow(QWidget *parent = nullptr);
15     ~MainWindow();
16
17     /* 重写父类下的 protected 方法*/
18 protected:
19     void paintEvent(QPaintEvent *);
20
21 private:
22     /* 定时器，用于定时更新界面 */
23     QTimer *timer;
24     /* 角度 */
25     int angle;
26
27 private slots:
28     /* 槽函数 */
29     void timerTimeOut();
30
31 };
32 #endif // MAINWINDOW_H
```

第 18 行，因为 paintEvent()是父类 QWidget 的 protected 修饰符下虚方法（虚函数），所以建议重写时也写到子类下的 protected 修饰符下。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   03_qpainter
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-03-29
*****************************************************************/
#include "mainwindow.h"
#include "QDebug"
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    /* 设置主窗口位置及颜色 */
    this->setGeometry(0, 0, 800, 480);
    setPalette(QPalette(Qt::gray));
    setAutoFillBackground(true);

    /* 定时器实例化 */
    timer = new QTimer(this);

    /* 默认角度为 0 */
    angle = 0;

    /* 定时 100ms */
    timer->start(100);

    /* 信号槽连接 */
    connect(timer, SIGNAL(timeout()), this, SLOT(timerTimeOut()));
}

MainWindow::~MainWindow()
{
}

void MainWindow::timerTimeOut()
{
    /* 需要更新界面，不设置不更新 */
    this->update();
}

```

```
34  void MainWindow::paintEvent(QPaintEvent *)
35  {
36      /* 指定父对象，this 指本窗口 */
37      QPainter painter(this);
38
39      /* 设置抗锯齿，流畅转换 */
40      painter.setRenderHints(QPainter::Antialiasing
41                             | QPainter::SmoothPixmapTransform);
42      /* 计算旋转角度 */
43      if (angle++ == 360)
44          angle = 0;
45
46      /* QPixmap 类型对象 */
47      QPixmap image;
48
49      /* 加载 */
50      image.load(":/image/cd.png");
51
52      /* QRectF 即，继承 QRect（Qt 的矩形类），F 代表精确到浮点类型 */
53      QRectF rect((this->width() - image.width()) / 2,
54                  (this->height() - image.height()) / 2,
55                  image.width(),
56                  image.height());
57
58      /* 默认参考点为左上角原点（0,0），因为旋转需要以图形的中心为参考点，
59       * 我们使用 translate 把参考点设置为 CD 图形的中心点坐标 */
60      painter.translate(0 + rect.x() + rect.width() / 2,
61                        0 + rect.y() + rect.height() / 2);
62
63      /* 旋转角度 */
64      painter.rotate(angle);
65
66      /* 现在参考点为 CD 图形的中心，我们需要把它设置回原点的位置，
67       * 所以需要减去上面加上的数 */
68      painter.translate(0 - (rect.x() + rect.width() / 2),
69                        0 - (rect.y() + rect.height() / 2));
70
71      /* 画图,QPainter 提供了许多 drawX 的方法 */
72      painter.drawImage(rect, image.toImage(), image.rect());
73
74      /* 再画一个矩形 */
75      painter.drawRect(rect.toRect());
```

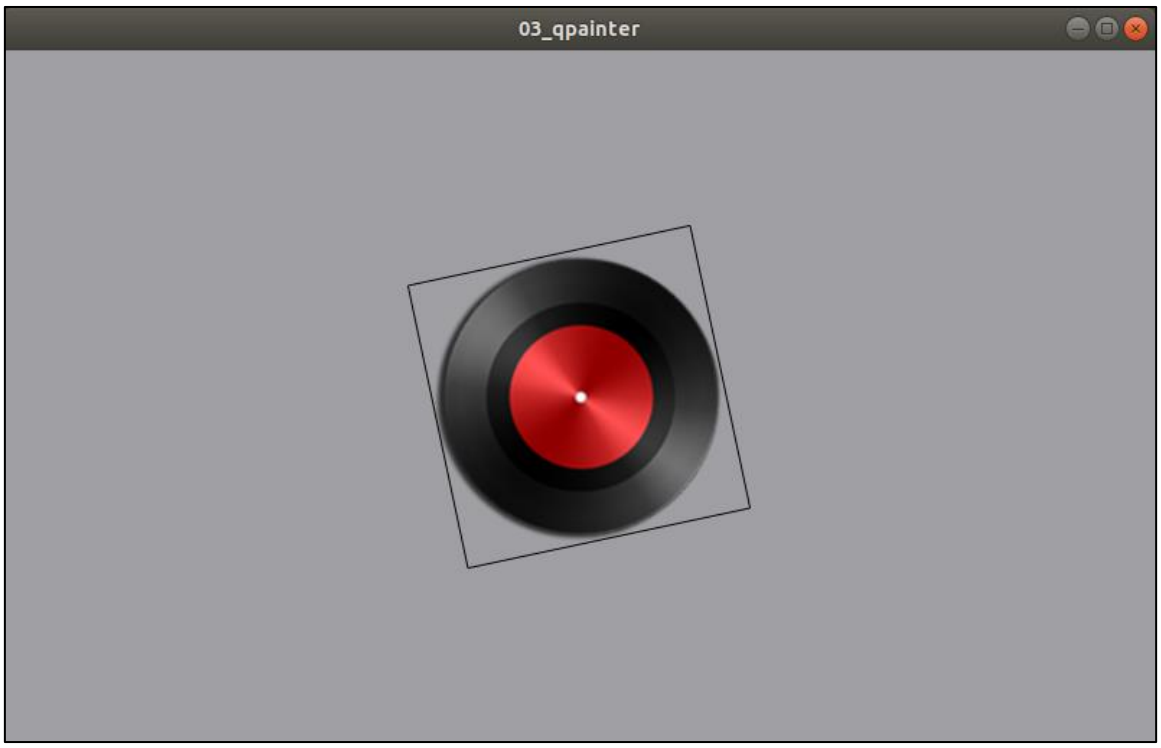
```
76  }
```

第 34~76 行，paintEvent()的实现。首先先指定需要画图的对象，加图片后，使用 translate()设置参考原点，旋转一定的角度后再恢复参考原点。之后就开始画图。在参考原点处可能比较难理解，大家根据上面的注释多多分析。

第 31 行，定时 100ms 更新一次界面。因为 paintEvent 事件在构造函数执行时只会执行一次。我们需要使用 update()方法来更新界面，才能看到 CD 旋转的效果。

9.1.2 程序运行效果

编译运行程序后可以看到如下效果，CD 的外框加画了一个矩形，使旋转更明显。使用 paintEvent 可以实现一些需要绘图的情景，它可能比 Qt 动画类更容易实现。结合 Qt 的画笔，也可以设计一个绘图软件，这多得益于 paintEvent()与 QPainter 使界面开发多了一些可能。在界面设计里，重绘界面使用 paintEvent()也比较多，需要我们掌握这部分内容。

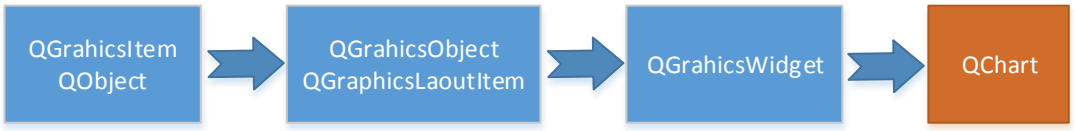


9.2 QChart 图表

自从 Qt 发布以来，给跨平台的用户带来很多便利。在 Qt5.7 之前，Qt 在开源社区版本里没有 Qt Charts（自带的绘图组件库）。这使得像 QWT、QCustomPlot 等第三方库有了巨大的生存空间，作者也在 Qt 5.7 以下版本使用过第三方的 QCustomPlot。要想使用 Qt Charts，我们的 Qt 版本得使用 Qt 5.7 之后的版本。其实 Qt Charts 并不是 Qt 5.7 才有的，是在 Qt 5.7 以前只有商业版本的 Qt 才有 Qt Charts。我们能免费下载的 Qt 版本都是社区（开源）版本。

Qt Charts 很方便的绘制我们常见的曲线图、折线图、柱状图和饼状图等图表。不用自己花精力去了解第三方组件的使用了或者开发第三方组件。Qt 的帮助文档里已经有说明 Qt Charts 主要部件的使用方法。需要用到时我们可以查看 Qt 文档就可以了。

下面我们主要简介一下 Qt Charts 模块，首先先看它的继承关系，（看继承关系可以了解这个类是怎么来的，它不可能是一下子崩出来的）。至于怎么查看 QChart 类的继承关系，在我们第六章里 Qt Creator 的快捷键有讲到，Ctrl + Shift + T ，点击要查询的类的继承关系。



要想在项目里使用 Qt Charts 模块，需要在 pro 文件下添加以下语句。

```
QT += charts
```

如果我们点击查看 Qt Charts 类，我们可以看到要想使用 Qt Charts 类，除了需要包括相应的头文件外，还需要使用命名空间。格式如下。

一般在头文件处加上这个。

```
QT_CHARTS_USE_NAMESPACE
```

或者在头文件类外加上以下语句。

```
using namespace QtCharts;
```

下面我们直接开始例子，了解一下 Qt Charts 的使用。

9.2.1 应用实例

本例目的：快速了解 Qt Charts 的使用。例子非常实用，除了可以绘制静态曲线，也可以绘制动态曲线。例子可以直接应用到实际项目中利用提供接口读取数据绘制动态曲线图。

例 04_qtchart，实时动态曲线（难度：一般）。项目路径为 Qt/2/04_qtchart。本例基本流程如下：使用一个 QSplineSeries 对象（曲线），一个 QChart（图表），一个 QChartView（图表视图）。首先我们创建坐 chart 图表，然后创建两条坐标轴 axisX 与 axisY。将两条坐标轴添加到 chart 图表上，再将 splineSeries 曲线与坐标轴连系起来。最后再将 chart 图表添加到 chartView 图表视图中。曲线上的数据由系统产生随机数，使用定时器更新数据。

项目文件 04_qtchart.pro 文件第一行添加的代码部分如下。

04_qtchart.pro 编程后的代码

```
1  QT       += core gui charts
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
```

```
11  DEFINES += QT_DEPRECATED_WARNINGS
12
13  # You can also make your code fail to compile if it uses deprecated APIs.
14  # In order to do so, uncomment the following line.
15  # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16  #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18  SOURCES += \
19      main.cpp \
20      mainwindow.cpp
21
22  HEADERS += \
23      mainwindow.h
24
25  # Default rules for deployment.
26  qnx: target.path = /tmp/$${TARGET}/bin
27  else: unix:!android: target.path = /opt/$${TARGET}/bin
28  !isEmpty(target.path): INSTALLS += target
```

在头文件“mainwindow.h”具体代码如下。

```
                        mainwindow.h 编程后的代码
    /*****************************************************************
    Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
    * @projectName   04_qtchart
    * @brief         mainwindow.h
    * @author        Deng Zhimao
    * @email         1252699831@qq.com
    * @net           www.openedv.com
    * @date          2021-03-28
    *****************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3   #include <QChartView>
4   #include <QSplineSeries>
5   #include <QScatterSeries>
6   #include <QDebug>
7   #include <QValueAxis>
8   #include <QTimer>
9   #include <QMainWindow>
10
11  /*  必需添加命名空间 */
12  QT_CHARTS_USE_NAMESPACE
```

```
13
14  class MainWindow : public QMainWindow
15  {
16      Q_OBJECT
17
18  public:
19      MainWindow(QWidget *parent = nullptr);
20      ~MainWindow();
21
22  private:
23      /* 接收数据接口 */
24      void receivedData(int);
25
26      /* 数据最大个数 */
27      int maxSize;
28
29      /* x 轴上的最大值 */
30      int maxX;
31
32      /* y 轴上的最大值 */
33      int maxY;
34
35      /* y 轴 */
36      QValueAxis *axisY;
37
38      /* x 轴 */
39      QValueAxis *axisX;
40
41      /* QList int 类型容器 */
42      QList<int> data;
43
44      /* QSplineSeries 对象（曲线）*/
45      QSplineSeries *splineSeries;
46
47      /* QChart 图表 */
48      QChart *chart;
49
50      /* 图表视图 */
51      QChartView *chartView;
52
53      /* 定时器 */
54      QTimer *timer;
55
```

```
56  private slots:
57      void timerTimeOut();
58  };
59  #endif // MAINWINDOW_H
```

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

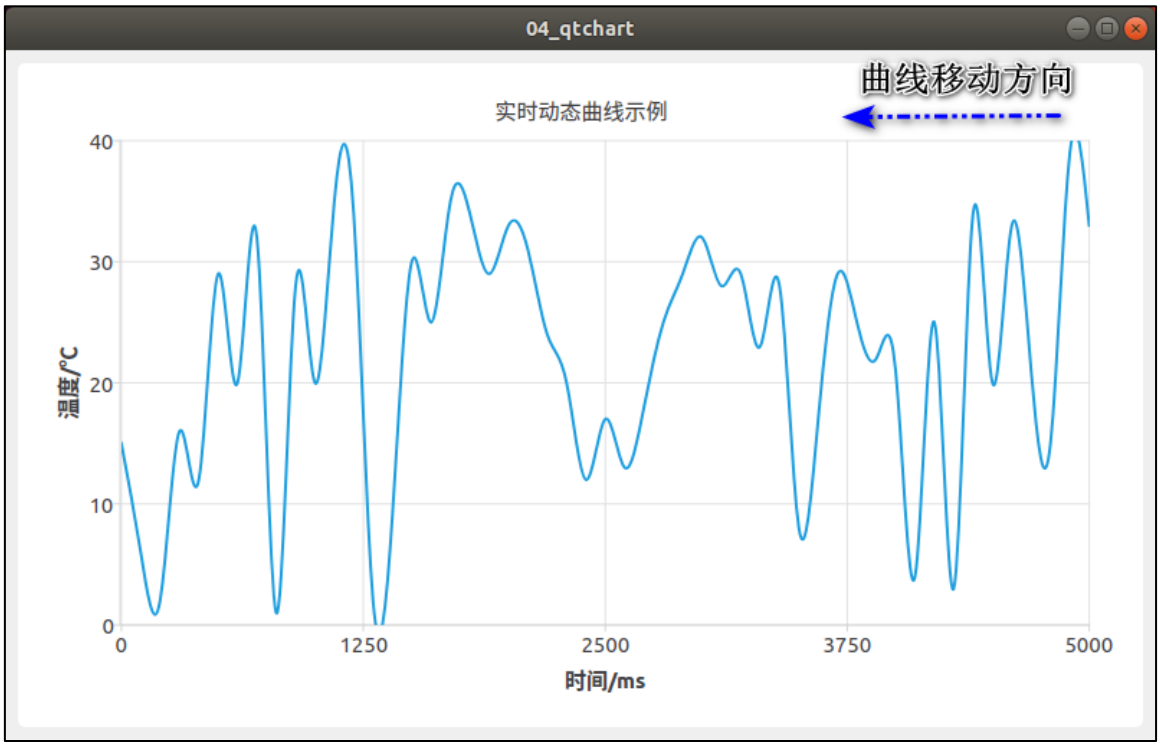
```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   04_qtchart
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-03-28
*****************************************************************/
1   #include "mainwindow.h"
2   #include <QDateTime>
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置最显示位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8       /* 最大储存 maxSize - 1 个数据 */
9       maxSize = 51;
10      /* x 轴上的最大值 */
11      maxX = 5000;
12      /* y 轴最大值 */
13      maxY = 40;
14
15      /* splineSeries 曲线实例化（折线用 QLineSeries） */
16      splineSeries = new QSplineSeries();
17      /* 图表实例化 */
18      chart = new QChart();
19      /* 图表视图实例化 */
20      chartView = new QChartView();
21
22      /* 坐标轴 */
23      axisY = new QValueAxis();
24      axisX = new QValueAxis();
25      /* 定时器 */
26      timer = new QTimer(this);
27
```

```
28      /* legend 译图例类型，以绘图的颜色区分，本例设置为隐藏 */
29      chart->legend()->hide();
30      /* chart 设置标题 */
31      chart->setTitle("实时动态曲线示例");
32      /* 添加一条曲线 splineSeries */
33      chart->addSeries(splineSeries);
34
35      /* 设置显示格式 */
36      axisY->setLabelFormat("%i");
37      /* y 轴标题 */
38      axisY->setTitleText("温度/℃");
39      /* y 轴标题位置（设置坐标轴的方向） */
40      chart->addAxis(axisY, Qt::AlignLeft);
41      /* 设置 y 轴范围 */
42      axisY->setRange(0, maxY);
43      /* 将 splineSeries 附加于 y 轴上 */
44      splineSeries->attachAxis(axisY);
45
46      /* 设置显示格式 */
47      axisX->setLabelFormat("%i");
48      /* x 轴标题 */
49      axisX->setTitleText("时间/ms");
50      /* x 轴标题位置（设置坐标轴的方向） */
51      chart->addAxis(axisX, Qt::AlignBottom);
52      /* 设置 x 轴范围 */
53      axisX->setRange(0, maxX);
54      /* 将 splineSeries 附加于 x 轴上 */
55      splineSeries->attachAxis(axisX);
56
57      /* 将图表的内容设置在图表视图上 */
58      chartView->setChart(chart);
59      /* 设置抗锯齿 */
60      chartView->setRenderHint(QPainter::Antialiasing);
61
62      /* 设置为图表视图为中心部件 */
63      setCentralWidget(chartView);
64
65      /* 定时 200ms */
66      timer->start(200);
67      /* 信号槽连接 */
68      connect(timer, SIGNAL(timeout()), this, SLOT(timerTimeOut()));
```

```
69
70      /* 设置随机种子，随机数初始化 */
71      qsrand(time(NULL));
72  }
73
74  MainWindow::~MainWindow()
75  {
76  }
77
78  void MainWindow::timerTimeOut()
79  {
80      /* 产生随机 0~maxY 之间的数据 */
81      receivedData(qrand() % maxY );
82  }
83
84  void MainWindow::receivedData(int value)
85  {
86      /* 将数据添加到 data 中 */
87      data.append(value);
88
89      /* 当储存数据的个数大于最大值时，把第一个数据删除 */
90      while (data.size() > maxSize) {
91          /* 移除 data 中第一个数据 */
92          data.removeFirst();
93      }
94
95      /* 先清空 */
96      splineSeries->clear();
97
98      /* 计算 x 轴上的点与点之间显示的间距 */
99      int xSpace = maxX / (maxSize - 1);
100
101     /* 添加点，xSpace * i 表示第 i 个点的 x 轴的位置 */
102     for (int i = 0; i < data.size(); ++i) {
103         splineSeries->append(xSpace * i, data.at(i));
104     }
105 }
```

第 84~105 行，是实现曲线移动的重要代码，代码算法是，当数据的个数超过最大值后，我们就删除第一个数据，如此反复，就实现了数据移动的过程，同时图表视图中的曲线因为值的改变实现了“移动”。

9.2.2 程序运行效果


04_qtchart
曲线移动方向
实时动态曲线示例
40
30
20
10
0
温度/℃
0
1250
2500
3750
5000
时间/ms

第十章 多线程

我们写的一个应用程序，应用程序跑起来后一般情况下只有一个线程，但是可能也有特殊情况。比如我们前面章节写的例程都跑起来后只有一个线程，就是程序的主线程。线程内的操作都是顺序执行的。恩，顺序执行？试着想一下，我们的程序顺序执行，假设我们的用户界面点击有某个操作是比较耗时的。您会发现界面点击完了，点击界面对应的操作还没有完成，所以就会冻结界面，不能响应，直到操作完成后，才返回到正常的界面里。如果我们的界面是这么设计的话，估计用户得发毛了。

这种情况我们一般是创建一个单独的线程来执行这个比较耗时的操作。比如我们使用摄像头拍照保存照片。恩，很多朋友问，这个不算耗时吧。对的在电脑上使用 Qt 拍照，处理起来非常快。根本也不需要开启一个线程来做这种事。但是我们是否考虑在嵌入式的 CPU 上做这种事情呢？嵌入式的 CPU 大多数都没有电脑里的 CPU 主频（几 GHz）那么高，处理速度也不快。此时我们就需要考虑开多一个线程来拍照了。拍完照再与主线程（主线程即程序原来的线程）处理好照片的数据，就完成了一个多线程的应用程序了。

官方文档里说，QThread 类提供了一种独立于平台的方法来管理线程。QThread 对象在程序中管理一个控制线程。QThreads 在 run()中开始执行。默认情况下，run()通过调用 exec()来启动事件循环，并在线程中运行 Qt 事件循环。您可以通过使用 QObject::moveToThread()将 worker 对象移动到线程来使用它们。

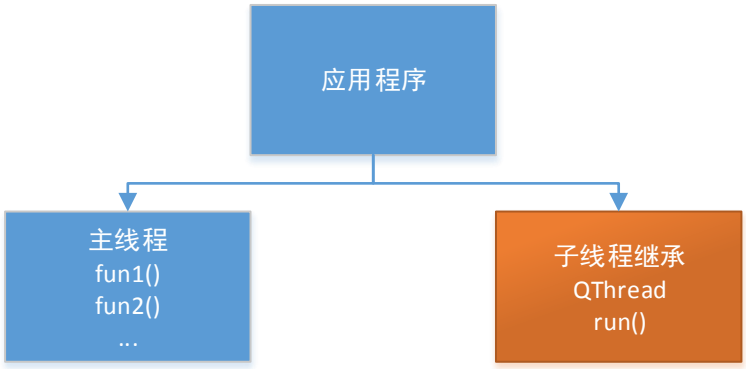
QThread 线程类是实现多线程的核心类。Qt 有两种多线程的方法，其中一种是继承 QThread 的 run()函数，另外一种是把一个继承于 QObject 的类转移到一个 Thread 里。Qt4.8 之前都是使用继承 QThread 的 run()这种方法，但是 Qt4.8 之后，Qt 官方建议使用第二种方法。两种方法区别不大，用起来都比较方便，但继承 QObject 的方法更加灵活。所以 Qt 的帮助文档里给的参考是先给继承 QObject 的类，然后再给继承 QThread 的类。

另外 Qt 提供了 QMutex、QMutexLocker、QReadLocker 和 QWriteLocker 等类用于线程之间的同步，详细可以看 Qt 的帮助文档。

本章介绍主要如何使用 QThread 实现多线程编程，讲解如何通过继承 QThread 和 QObject 的方法来创建线程。还会使用 QMutexLocker 正确的退出一个线程。本章的内容就是这么多，并不深入，所以不难，**目的就是快速掌握 Qt 线程的创建，理解线程**。

10.1 继承 QThread 的线程

在第十章的章节开头说过了，继承 QThread 是创建线程的一个普通方法。其中创建的线程只有 run()方法在线程里的。其他类内定义的方法都在主线程内。恩，这样不理解？我们画个图捋一捋。



通过上面的图我们可以看到，主线程内有很多方法在主线程内，但是子线程，只有 run()方法是在子线程里的。run()方法是继承于 QThread 类的方法，用户需要重写这个方法，一般是把耗时的操作写在这个 run()方法里面。

10.1.1 应用实例

本例目的：快速了解继承 QThread 类线程的使用。

例 05_qthread_example1，继承 QThread 类的线程（难度：一般）。项目路径为 Qt/2/05_qthread_example1。本例通过 QThread 类继承线程，然后在 MainWindow 类里使用。通过点击一个按钮开启线程。当线程执行完成时，会发送 resultReady(const QString &s)信号给主线程。流程就这么简单。

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   05_qthread_example1
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-06
*****************************************************************/
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QThread>
#include <QDebug>
```

```
7   #include <QPushButton>
8
9   /* 使用下面声明的 WorkerThread 线程类 */
10  class WorkerThread;
11
12  class MainWindow : public QMainWindow
13  {
14      Q_OBJECT
15
16  public:
17      MainWindow(QWidget *parent = nullptr);
18      ~MainWindow();
19
20  private:
21      /* 在 MainWindow 类里声明对象 */
22      WorkerThread *workerThread;
23
24      /* 声明一个按钮，使用此按钮点击后开启线程 */
25      QPushButton *pushButton;
26
27  private slots:
28      /* 槽函数，用于接收线程发送的信号 */
29      void handleResults(const QString &result);
30
31      /* 点击按钮开启线程 */
32      void pushButtonClicked();
33  };
34
35  /* 新建一个 WorkerThread 类继承于 QThread */
36  class WorkerThread : public QThread
37  {
38      /* 用到信号槽即需要此宏定义 */
39      Q_OBJECT
40
41  public:
42      WorkerThread(QWidget *parent = nullptr) {
43          Q_UNUSED(parent);
44      }
45
46      /* 重写 run 方法，继承 QThread 的类，只有 run 方法是在新的线程里 */
47      void run() override {
48          QString result = "线程开启成功";
49
```

```
50          /* 这里写上比较耗时的操作 */
51          // ...
52          // 延时 2s，把延时 2s 当作耗时操作
53          sleep(2);
54
55          /* 发送结果准备好的信号 */
56          emit resultReady(result);
57      }
58
59  signals:
60      /* 声明一个信号，译结果准确好的信号 */
61      void resultReady(const QString &s);
62  };
63
64  #endif // MAINWINDOW_H
65
```

第 36 行，声明一个 WorkerThread 的类继承 QThread 类，这里是参考 Qt 的 QThread 类的帮助文档的写法。

第 47 行，重写 run()方法，这里很重要。把耗时操作写于此，本例相当于一个继承 QThread 类线程模板了。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*********************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   05_qthread_example1
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-06
*********************************************************************/
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 对象实例化 */
10      pushButton = new QPushButton(this);
11      workerThread = new WorkerThread(this);
```

```
12
13      /* 按钮设置大小与文本 */
14      pushButton->resize(100, 40);
15      pushButton->setText("开启线程");
16
17      /* 信号槽连接 */
18      connect(workerThread, SIGNAL(resultReady(QString)),
19              this, SLOT(handleResults(QString)));
20      connect(pushButton, SIGNAL(clicked()),
21              this, SLOT(pushButtonClicked()));
22  }
23
24  MainWindow::~MainWindow()
25  {
26      /* 进程退出，注意本例 run()方法没写循环，此方法需要有循环才生效 */
27      workerThread->quit();
28
29      /* 阻塞等待 2000ms 检查一次进程是否已经退出 */
30      if (workerThread->wait(2000)) {
31          qDebug()<<"线程已经结束！"<<endl;
32      }
33  }
34
35  void MainWindow::handleResults(const QString &result)
36  {
37      /* 打印出线程发送过来的结果 */
38      qDebug()<<result<<endl;
39  }
40
41  void MainWindow::pushButtonClicked()
42  {
43      /* 检查线程是否在运行，如果没有则开始运行 */
44      if (!workerThread->isRunning())
45          workerThread->start();
46  }
```

第 11 行，线程对象实例化，Qt 使用 C++基本都是对象编程，Qt 线程也不例外。所以我们也是用对象来管理线程的。

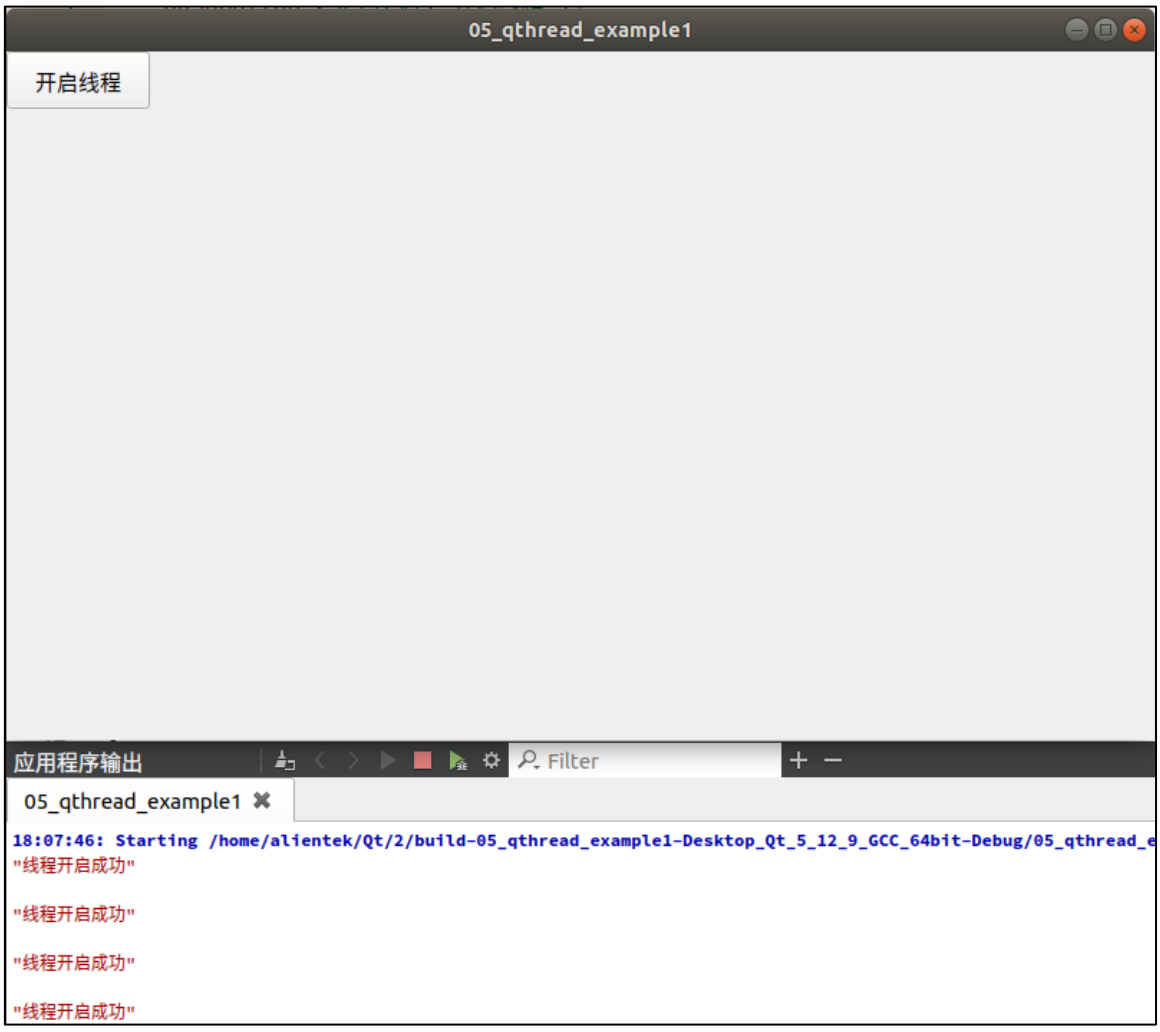
第 24~33 行，在 MainWindow 的析构函数里退出线程，然后判断线程是否退出成功。因为我们这个线程是没有循环操作的，直接点击按钮开启线程后，做了 2s 延时操作后就完成了。所以我们在析构函数里直接退出没有关系。

第 41~46 行，按钮点击后开启线程，首先我们得判断这个线程是否在运行，如果不在运行我们则开始线程，开始线程用 start()方法，它会调用重写的 run()函数的。

10.1.2 程序运行效果

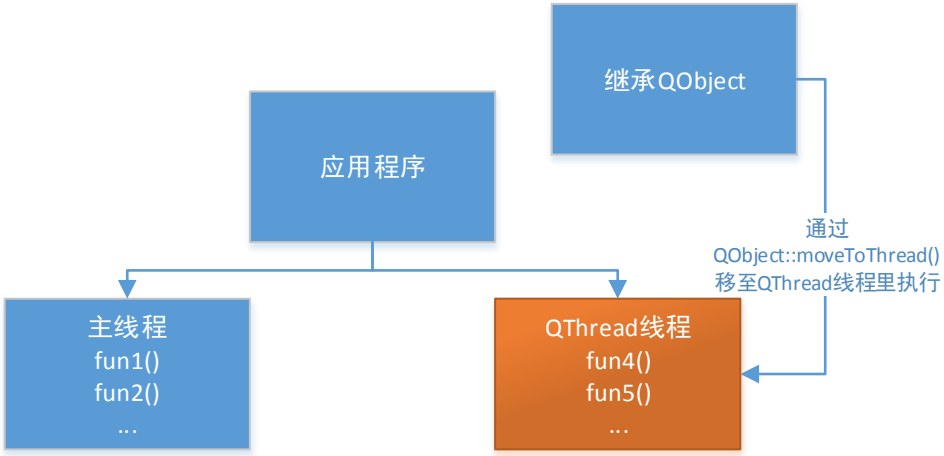
点击开启线程按钮后，延时 2s 后，Qt Creator 的应用程序输出窗口打印出“线程开启成功”。在 2s 内多次点击按钮则不会重复开启线程，因为线程在这 2s 内还在运行。同时我们可以看到点击按钮没卡顿现象。因为这个延时操作是在我们创建的线程里运行的，而 pushButton 是在主线程里的，通过点击按钮控制子线程的运行。

当关闭程序后，子线程将在主线程的析构函数里退出。注意线程使用 wait()方法，这里等待 2s，因为我们开启的线和是延时 2s 就完成了。如果是实际的操作，请根据 CPU 的处理能力，给一个适合的延时，阻塞等待线程完成后，就会自动退出并打印“线程已经结束”。



10.2 继承 QObject 的线程

在第 10 章章节开头已经说过，继承 QThread 类是创建线程的一种方法，另一种就是继承 QObject 类。继承 QObject 类更加灵活。它通过 QObject::moveToThread()方法，将一个 QObeject 的类转移到一个线程里执行。恩，不理解的话，我们下面也画个图捋一下。



通过上面的图不难理解，首先我们写一个类继承 QObject，通过 QObject::moveToThread()方法将它移到一个 QThread 线程里执行。那么可以通过主线程发送信号去调用 QThread 线程的方法如上图的 fun4()，fun5()等等。这些方法都是在 QThread 线程里执行的。

10.2.1 应用实例

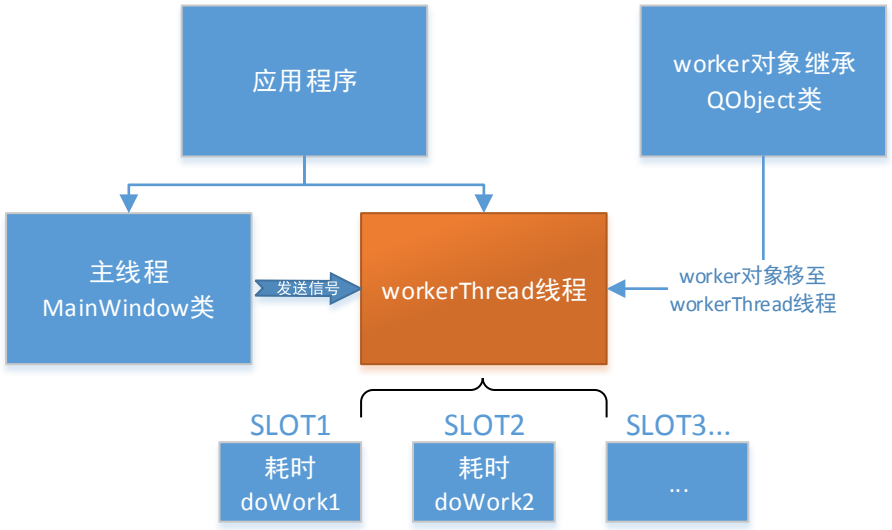
本例目的：快速了解继承 QObject 类线程的使用。

例 06_qthread_example2，继承 QObject 类的线程（难度：一般）。项目路径为 Qt/2/06_qthread_example2。本例通过 QObject 类继承线程，然后在 MainWindow 类里使用。通过点击一个按钮开启线程。另一个按钮点击关闭线程。另外通过加锁的操作来安全的终止一个线程。（我们可以通过 QMutexLocker 可以安全的使用 QMutex 以免忘记解锁。）

在我们谈谈为什么需要加锁来终止一个线程？因为 quit()和 exit()方法都不会中途终止线程。要马上终止一个线程可以用 terminate()方法。但是这个函数存在非常不安全的因素，Qt 官方文档说不推荐使用。

我们可以添加一个 bool 变量，通过主线程修改这个 bool 变量来终止，但是有可能引起访问冲突，所以需要加锁，例程里可能体现不是那么明确，当我们有 doWork1()，doWork2…就能体现到 bool 变量加锁的作用了。但是加锁会消耗一定的性能，增加耗时。

下面的例子是仿照 Qt 官方写的，看似简单，但是流程大家可能不是很明白，所以画个了大体的流程图，给大伙瞧瞧。



在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   06_qthread_example2
* @brief        mainwindow.h
* @author       Deng Zhimao
* @email        1252699831@qq.com
* @net          www.openedv.com
* @date         2021-04-08
*****************************************************************/
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QThread>
#include <QDebug>
#include <QPushButton>
#include <QMutexLocker>
#include <QMutex>

/* 工人类 */
class Worker;

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
```

```
20      ~MainWindow();
21
22  private:
23      /* 开始线程按钮 */
24      QPushButton *pushButton1;
25
26      /* 打断线程按钮 */
27      QPushButton *pushButton2;
28
29      /* 全局线程 */
30      QThread workerThread;
31
32      /* 工人类 */
33      Worker *worker;
34
35  private slots:
36      /* 按钮 1 点击开启线程 */
37      void pushButton1Clicked();
38
39      /* 按钮 2 点击打断线程 */
40      void pushButton2Clicked();
41
42      /* 用于接收工人是否在工作的信号 */
43      void handleResults(const QString &);
44
45  signals:
46      /* 工人开始工作（做些耗时的操作 ） */
47      void startWork(const QString &);
48  };
49
50  /* Worker 类，这个类声明了 doWork1 函数，将整个 Worker 类移至线程 workerThread
*/
51  class Worker : public QObject
52  {
53      Q_OBJECT
54
55  private:
56      /* 互斥锁 */
57      QMutex lock;
58
59      /* 标志位 */
60      bool isCanRun;
61
```

```
62  public slots:
63      /* 耗时的工作都放在槽函数下，工人可以有多份不同的工作，但是每次只能去做一份 */
64      void doWork1(const QString &parameter) {
65
66          /* 标志位为真 */
67          isCanRun = true;
68
69          /* 死循环 */
70          while (1) {
71              /* 此{}作用是 QMutexLocker 与 lock 的作用范围，获取锁后，
72               * 运行完成后即解锁 */
73              {
74                  QMutexLocker locker(&lock);
75                  /* 如果标志位不为真 */
76                  if (!isCanRun) {
77                      /* 跳出循环 */
78                      break;
79                  }
80              }
81              /* 使用 QThread 里的延时函数，当作一个普通延时 */
82              QThread::sleep(2);
83
84              emit resultReady(parameter + "doWork1 函数");
85          }
86          /* doWork1 运行完成，发送信号 */
87          emit resultReady("打断 doWork1 函数");
88      }
89
90      // void doWork2();...
91
92  public:
93      /* 打断线程（注意此方法不能放在槽函数下） */
94      void stopWork() {
95          qDebug()<<"打断线程"<<endl;
96
97          /* 获取锁后，运行完成后即解锁 */
98          QMutexLocker locker(&lock);
99          isCanRun = false;
100     }
101
102 signals:
103     /* 工人工作函数状态的信号 */
```

```
104     void resultReady(const QString &result);
105 };
106 #endif // MAINWINDOW_H
```

第 51~105 行，声明一个 Worker 的类继承 QObject 类，这里是参考 Qt 的 QThread 类的帮助文档的写法。将官方的例子运用到我们的例子里去。

第 62~88 行，我们把耗时的工作都放于槽函数下。工人可以有不同的工作，但是每次只能去做一份。这里不同于继承 QThread 类的线程 run()，继承 QThread 的类只有 run()在新线程里。而继承 QObject 的类，使用 moveToThread()可以把整个继承的 QObject 类移至线程里执行，所以可以有 doWork1(),doWork2…等等耗时的操作，但是这些**耗时的操作都应该作为槽函数**，由主线程去调用。

第 67~80 行，进入循环后使用互拆锁判断 isCanRun 变量的状态，为假即跳出 while 循环，直到 doWork1 结束。注意，虽然 doWork1 结束了，但是线程并没有退出（结束）。因为我们把这个类移到线程里了，直到这个类被销毁。或者使用 quit()和 exit()退出线程才真正的结束！

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*********************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   06_qthread_example2
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-08
*********************************************************************/
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    /* 设置显示位置与大小 */
    this->setGeometry(0, 0, 800, 480);
    pushButton1 =  new QPushButton(this);
    pushButton2 =  new QPushButton(this);


    /* 设置按钮的位置大小 */
    pushButton1->setGeometry(300, 200, 80, 40);
    pushButton2->setGeometry(400, 200, 80, 40);

    /* 设置两个按钮的文本 */
    pushButton1->setText("开启线程");
```

```
18      pushButton2->setText("打断线程");
19
20      /* 工人类实例化 */
21      worker = new Worker;
22
23      /* 将 worker 类移至线程 workerThread */
24      worker->moveToThread(&workerThread);
25
26      /* 信号槽连接 */
27
28      /* 线程完成销毁对象 */
29      connect(&workerThread, SIGNAL(finished()),
30              worker, SLOT(deleteLater()));
31      connect(&workerThread, SIGNAL(finished()),
32              &workerThread, SLOT(deleteLater()));
33
34      /* 发送开始工作的信号，开始工作 */
35      connect(this, SIGNAL(startWork(QString)),
36              worker, SLOT(doWork1(QString)));
37
38      /* 接收到 worker 发送过来的信号 */
39      connect(worker, SIGNAL(resultReady(QString)),
40              this, SLOT(handleResults(QString)));
41
42      /* 点击按钮开始线程 */
43      connect(pushButton1, SIGNAL(clicked()),
44              this, SLOT(pushButton1Clicked()));
45
46      /* 点击按钮打断线程 */
47      connect(pushButton2, SIGNAL(clicked()),
48              this, SLOT(pushButton2Clicked()));
49  }
50
51  MainWindow::~MainWindow()
52  {
53      /* 打断线程再退出 */
54      worker->stopWork();
55      workerThread.quit();
56
57      /* 阻塞线程 2000ms，判断线程是否结束 */
58      if (workerThread.wait(2000)) {
59          qDebug()<<"线程结束"<<endl;
```

```
60      }
61  }
62
63  void MainWindow::pushButton1Clicked()
64  {
65      /* 字符串常量 */
66      const QString str = "正在运行";
67
68      /* 判断线程是否在运行 */
69      if(!workerThread.isRunning()) {
70          /* 开启线程 */
71          workerThread.start();
72      }
73
74      /* 发送正在运行的信号，线程收到信号后执行后返回线程耗时函数 + 此字符串 */
75      emit this->startWork(str);
76  }
77
78  void MainWindow::pushButton2Clicked()
79  {
80      /* 如果线程在运行 */
81      if(workerThread.isRunning()) {
82
83          /* 停止耗时工作，跳出耗时工作的循环 */
84          worker->stopWork();
85      }
86  }
87
88  void MainWindow::handleResults(const QString & results)
89  {
90      /* 打印线程的状态 */
91      qDebug()<<"线程的状态: "<<results<<endl;
92  }
```

第 20 行，工人类实例化。**继承 QObject 的多线程类不能指定父对象**。

第 24 行，工人类实例化后，工人类将自己移至 workerThread 线程里执行。

第 29~32 行，线程结束后，我们需要使用 deleteLater 来销毁 worker 对象和 workerThread 对象分配的内存。deleteLater 会确认消息循环中没有这两个线程的对象后销毁。

10.2.2 程序运行效果

点击开启线程按钮后，应用程序输出窗口每隔 2 秒打印“正在运行 doWork1 函数”，当我们点击打断线程按钮后，窗口打印出“打断 doWork1 函数”。点击打断线程，会打断 doWork1

函数的循环，doWork1 函数就运行结束了。再点击开启线程，可以再次运行 doWork1 函数。本例界面简单，仅用了两个按钮和打印语句作为显示部分，但是对初学线程的朋友们友好，因为程序不长。我们可以结合程序的注释，一步步去理解这种线程的写法。重要的是掌握写法，最后才应用到花里胡哨的界面去吧！

第十一章 网络编程

Qt 网络模块为我们提供了编写 TCP / IP 客户端和服务器的类。它提供了较低级别的类，例如代表低级网络概念的 QTcpSocket，QTcpServer 和 QUdpSocket，以及诸如 QNetworkRequest，QNetworkReply 和 QNetworkAccessManager 之类的高级类来执行使用通用协议的网络操作。它还提供了诸如 QNetworkConfiguration，QNetworkConfigurationManager 和 QNetworkSession 等类，实现承载管理。

想要在程序中使用 Qt 网络模块，我们需要在 pro 项目配置文件里增加下面的一条语句。

```
QT       += network
```

11.1 获取本机的网络信息

为什么先写获取本机网络信息的内容呢？在建立网络通信之前我们至少得获取对方的 IP 地址。在网络应用中，经常需要用到本机的主机名、IP 地址、MAC 地址等网络信息，通常通

在 Windows 通过调出命令行 cmd 窗口输入 ipconfig 或者在 Linux 系统中使用 ifconfig 命令就可以查看相关信息了，在这里我们利用 Qt 做出一个可以查询的界面和功能出来，为了后面的网络编程打下一个简单的基础。

Qt 提供了 QHostInfo 和 QNetworkInterface 类可以用于此类信息查询。更多关于 QHostInfo 和 QNetworkInterface 的相关函数可以在 Qt 的帮助文档中找到。下面我们写代码时会使用到相关的函数，有清楚的注释。

11.1.1 应用实例

本例目的：了解如何通过 QHostInfo 和 QNetworkInterface 类获取本地网络所有接口的信息。

例 07_networkhostinfo，获取本机网络接口信息（难度：一般）。项目路径为 Qt/2/07_networkhostinfo。本例获取本机的网络接口信息，打印在文本浏览框上，点击按钮可直接获取，为了清楚看见是重新获取的过程，本例点击获取本机信息按钮后延时 1s 去刷新获取的信息。点击另一个清空文本信息按钮可以清空文本浏览框上的文本内容。

项目文件 07_networkhostinfo.pro 文件第一行添加的代码部分如下。

07_networkhostinfo.pro 编程后的代码

```
1  QT       += core gui network
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
13 # You can also make your code fail to compile if it uses deprecated APIs.
14 # In order to do so, uncomment the following line.
15 # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18 SOURCES += \
19     main.cpp \
20     mainwindow.cpp
21
22 HEADERS += \
23     mainwindow.h
24
25 # Default rules for deployment.
```

```
26 qnx: target.path = /tmp/$${TARGET}/bin
27 else: unix:!android: target.path = /opt/$${TARGET}/bin
28 !isEmpty(target.path): INSTALLS += target
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   07_networkhostinfo
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-10
*****************************************************************/
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QPushButton>
6  #include <QTextBrowser>
7  #include <QVBoxLayout>
8  #include <QHBoxLayout>
9  #include <QTimer>
10
11 class MainWindow : public QMainWindow
12 {
13     Q_OBJECT
14
15 public:
16     MainWindow(QWidget *parent = nullptr);
17     ~MainWindow();
18
19 private:
20     /* 点击获取和清空文本按钮 */
21     QPushButton *pushButton[2];
22
23     /* 文本浏览框用于显示本机的信息 */
24     QTextBrowser *textBrowser;
25
26     /* 水平 Widget 容器和垂直 Widget 容器*/
27     QWidget *hWidget;
28     QWidget *vWidget;
29
```

```
30      /* 水平布局和垂直布局 */
31      QHBoxLayout *hBoxLayout;
32      QVBoxLayout *vBoxLayout;
33
34      /* 定时器 */
35      QTimer *timer;
36
37      /* 获取本机的网络的信息，返回类型是 QString */
38      QString getHostInfo();
39
40  private slots:
41      /* 定时器槽函数，点击按钮后定时触发 */
42      void timerTimeOut();
43
44      /* 显示本机信息 */
45      void showHostInfo();
46
47      /* 启动定时器 */
48      void timerStart();
49
50      /* 清空 textBrowser 的信息 */
51      void clearHostInfo();
52  };
53  #endif // MAINWINDOW_H
54
```

头文件里主要是声明两个按钮和一个文本浏览框。另外还有一个定时器，声明一些槽函数，比较简单。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   07_networkhostinfo
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-10
*****************************************************************/
1   #include "mainwindow.h"
2   #include <QNetworkInterface>
3   #include <QHostInfo>
4   #include <QThread>
```

```
5   #include <QDebug>
6
7   MainWindow::MainWindow(QWidget *parent)
8       : QMainWindow(parent)
9   {
10      /* 设置位置与大小 */
11      this->setGeometry(0, 0, 800, 480);
12
13      /* 点击获取本地信息按钮和清空文本按钮 */
14      pushButton[0] = new QPushButton();
15      pushButton[1] = new QPushButton();
16
17      pushButton[0]->setText("获取本机信息");
18      pushButton[1]->setText("清空文本信息");
19
20      /* 按钮的大小根据文本自适应,
21       * 注意 setSizePolicy 需要在布局中使用 */
22      pushButton[0]->setSizePolicy(QSizePolicy::Fixed,
23                                   QSizePolicy::Fixed);
24      pushButton[1]->setSizePolicy(QSizePolicy::Fixed,
25                                   QSizePolicy::Fixed);
26
27      /* 水平 Widget 和垂直 Widget 用于添加布局 */
28      hWidget = new QWidget();
29      vWidget = new QWidget();
30
31      /* 水平布局和垂直布局 */
32      hBoxLayout = new QHBoxLayout();
33      vBoxLayout = new QVBoxLayout();
34
35      /* 文本浏览框 */
36      textBrowser = new QTextBrowser();
37
38      /* 添加到水平布局 */
39      hBoxLayout->addWidget(pushButton[0]);
40      hBoxLayout->addWidget(pushButton[1]);
41
42      /* 将水平布局设置为 hWidget 的布局 */
43      hWidget->setLayout(hBoxLayout);
44
45      /* 将文本浏览框和 hWidget 添加到垂直布局 */
46      vBoxLayout->addWidget(textBrowser);
```

```
47      vBoxLayout->addWidget(hWidget);
48
49      /* 将垂直布局设置为 vWidget 的布局 */
50      vWidget->setLayout(vBoxLayout);
51
52      /* 设置 vWidget 为中心部件 */
53      setCentralWidget(vWidget);
54
55      /* 定时器初始化 */
56      timer = new QTimer();
57
58      /* 信号槽连接 */
59      connect(pushButton[0], SIGNAL(clicked()),
60              this, SLOT(timerStart()));
61      connect(pushButton[1], SIGNAL(clicked()),
62              this, SLOT(clearHostInfo()));
63      connect(timer, SIGNAL(timeout()),
64              this, SLOT(timerTimeOut()));
65  }
66
67  MainWindow::~MainWindow()
68  {
69  }
70
71
72  void MainWindow::timerStart()
73  {
74      /* 清空文本 */
75      textBrowser->clear();
76
77      /* 定时 1s */
78      timer->start(1000);
79  }
80
81  void MainWindow::timerTimeOut()
82  {
83      /* 显示本机信息 */
84      showHostInfo();
85
86      /* 停止定时器 */
87      timer->stop();
88  }
89
```

```
90  QString MainWindow::getHostInfo()
91  {
92      /* 通过 QHostInfo 的 localHostName 函数获取主机名称 */
93      QString str = "主机名称: " + QHostInfo::localHostName() + "\n";
94
95      /* 获取所有的网络接口,
96       * QNetworkInterface 类提供主机的 IP 地址和网络接口的列表 */
97      QList<QNetworkInterface> list
98            = QNetworkInterface::allInterfaces();
99
100     /* 遍历 list */
101     foreach (QNetworkInterface interface, list) {
102         str+= "网卡设备:" + interface.name() + "\n";
103         str+= "MAC 地址:" + interface.hardwareAddress() + "\n";
104
105         /* QNetworkAddressEntry 类存储 IP 地址子网掩码和广播地址 */
106         QList<QNetworkAddressEntry> entryList
107               = interface.addressEntries();
108
109         /* 遍历 entryList */
110         foreach (QNetworkAddressEntry entry, entryList) {
111             /* 过滤 IPv6 地址,只留下 IPv4 */
112             if (entry.ip().protocol() ==
113                     QAbstractSocket::IPv4Protocol) {
114                 str+= "IP 地址:" + entry.ip().toString() + "\n";
115                 str+= "子网掩码:" + entry.netmask().toString() + "\n";
116                 str+= "广播地址:" + entry.broadcast().toString() + "\n\n";
117             }
118         }
119     }
120
121     /* 返回网络信息 */
122     return str;
123 }
124
125 void MainWindow::showHostInfo()
126 {
127     /* 获取本机信息后显示到 textBrowser */
128     textBrowser->insertPlainText(getHostInfo());
129 }
130
131 void MainWindow::clearHostInfo()
```

```
132 {
133     /* 判断 textBrowser 是否为空，如果不为空则清空文本 */
134     if (!textBrowser->toPlainText().isEmpty())
135 
136         /* 清空文本 */
137         textBrowser->clear();
138 }
```

第 90~123 行，是本例最重要的代码。

第 93 行，通过 QHostInfo 的 localHostName 函数获取主机名称。

第 97~98 行，通过 QNetworkInterface::allInterfaces()获取网络接口列表 list 类存储 IP 地址子网掩码和广播地址。如果我们用 qDebug()函数打印出 list，可以发现获取了所有的网络信息。而我们要提取网络里面的网络信息使用 QNetworkAddressEntry。

第 106~107 行，使用 QNetworkAddressEntry 从 interface 接口里使用函数 addressEntries()，获取所有的条目。就可以使用 QNetworkAddressEntry 的对象 entry 获取 IP 地址子网掩码和广播地址。

第 110~118 行，因为获取的 entries 在一个 QNetworkInterface 下可能有两个 IP，分别是 ipv4 和 ipv6。这里使用 ip().protocol()来判断协议的类型，只留下 ipv4 类型的信息。筛选信息在我们写程序常常需要的。

11.1.2 程序运行效果

点击获取本机信息，在文本浏览框内就打印出本机的网络信息（包括了主机名，网卡名，ip 地址等）。这里因为过滤掉了 IPv6 的信息。通常一个网卡有两个 ip 地址，一个是 ipv4，另一个是 ipv6 的地址。下面的网卡设备 lo，是本地回环网卡。另一个 ens33 是虚拟机的网卡，由 VMware 虚拟出来的。点击清空文本信息会清空文本浏览框里的网络信息。

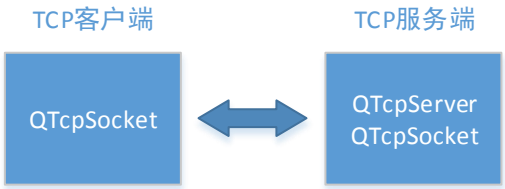
11.2 TCP 通信

11.2.1 TCP 简介

TCP 协议（Transmission Control Protocol）全称是传输控制协议是一种**面向连接的、可靠的、基于字节流**的传输层通信协议。

TCP 通信必**须先建立 TCP 连接**，通信端分为客户端和服务端。服务端通过监听某个端口来监听是否有客户端连接到来，如果有连接到来，则建立新的 socket 连接；客户端通过 ip 和 port 连接服务端，当成功建立连接之后，就可进行数据的收发了。需要注意的是，在 Qt 中，Qt 把 socket 当成输入输出流来对待的，数据的收发是通过 read()和 write()来进行的，需要与我们常见的 send()与 recv()进行区分。

TCP 客户端与服务端通信示意图如下。



11.2.2 TCP 服务端应用实例

本例目的：了解 TCP 服务端的使用。

例 08_tcpserver，TCP 服务端（难度：一般）。项目路径为 Qt/2/08_tcpserver。本例大体流程首先获取本地 IP 地址。创建一个 tcpSocket 套接字，一个 tcpServer 服务端。点击监听即监听本地的主机 IP 地址和端口，同时等待服务端的连接。此程序需要结合客户端一起使用。

项目文件 08_tcpserver.pro 文件第一行添加的代码部分如下。

08_tcpserver.pro 编程后的代码

```
1  QT       += core gui network
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
13 # You can also make your code fail to compile if it uses deprecated APIs.
14 # In order to do so, uncomment the following line.
15 # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18 SOURCES += \
19     main.cpp \
20     mainwindow.cpp
21
22 HEADERS += \
23     mainwindow.h
24
25 # Default rules for deployment.
26 qnx: target.path = /tmp/$${TARGET}/bin
27 else: unix:!android: target.path = /opt/$${TARGET}/bin
28 !isEmpty(target.path): INSTALLS += target
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   08_tcpserver
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
```

```
    * @net           www.openedv.com
    * @date          2021-04-13
    ************************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QTcpServer>
6   #include <QTcpSocket>
7   #include <QVBoxLayout>
8   #include <QHBoxLayout>
9   #include <QPushButton>
10  #include <QTextBrowser>
11  #include <QLabel>
12  #include <QComboBox>
13  #include <QSpinBox>
14  #include <QHostInfo>
15  #include <QLineEdit>
16  #include <QNetworkInterface>
17  #include <QDebug>
18
19  class MainWindow : public QMainWindow
20  {
21      Q_OBJECT
22
23  public:
24      MainWindow(QWidget *parent = nullptr);
25      ~MainWindow();
26
27  private:
28      /* tcp 服务器 */
29      QTcpServer *tcpServer;
30
31      /* 通信套接字 */
32      QTcpSocket *tcpSocket;
33
34      /* 按钮 */
35      QPushButton *pushButton[4];
36
37      /* 标签文本 */
38      QLabel *label[2];
39
40      /* 水平容器 */
```

```
41      QWidget *hWidget[3];
42
43      /* 水平布局 */
44      QHBoxLayout *hBoxLayout[3];
45
46      /* 垂直容器 */
47      QWidget *vWidget;
48
49      /* 垂直布局 */
50      QVBoxLayout *vBoxLayout;
51
52      /* 文本浏览框 */
53      QTextBrowser *textBrowser;
54
55      /* 用于显示本地 ip */
56      QComboBox *comboBox;
57
58      /* 用于选择端口 */
59      QSpinBox  *spinBox;
60
61      /* 文本输入框 */
62      QLineEdit *lineEdit;
63
64      /* 存储本地的 ip 列表地址 */
65      QList<QHostAddress> IPlist;
66
67      /* 获取本地的所有 ip */
68      void getLocalHostIP();
69
70  private slots:
71      /* 客户端连接处理槽函数 */
72      void clientConnected();
73
74      /* 开始监听槽函数 */
75      void startListen();
76
77      /* 停止监听槽函数 */
78      void stopListen();
79
80      /* 清除文本框时的内容 */
81      void clearTextBrowser();
82
```

```
83      /* 接收到消息 */
84      void receiveMessages();
85
86      /* 发送消息 */
87      void sendMessages();
88
89      /* 连接状态改变槽函数 */
90      void socketStateChange(QAbstractSocket::SocketState);
91  };
92  #endif // MAINWINDOW_H
```

头文件里主要是声明界面用的元素，及一些槽函数。重点是声明 tcpServer 和 tcpSocket。在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*********************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   08_tcpserver
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-13
*********************************************************************/
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗体的位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 实例化 tcp 服务器与 tcp 套接字 */
10      tcpServer = new QTcpServer(this);
11      tcpSocket = new QTcpSocket(this);
12
13      /* 开始监听按钮 */
14      pushButton[0] = new QPushButton();
15      /* 停止监听按钮 */
16      pushButton[1] = new QPushButton();
17      /* 清空聊天文本按钮 */
18      pushButton[2] = new QPushButton();
19      /* 发送消息按钮 */
20      pushButton[3] = new QPushButton();
```

```
21
22      /* 水平布局一 */
23      hBoxLayout[0] = new QHBoxLayout();
24      /* 水平布局二 */
25      hBoxLayout[1] = new QHBoxLayout();
26      /* 水平布局三 */
27      hBoxLayout[2] = new QHBoxLayout();
28      /* 水平布局四 */
29      hBoxLayout[3] = new QHBoxLayout();
30
31      /* 水平容器一 */
32      hWidget[0] =  new QWidget();
33      /* 水平容器二 */
34      hWidget[1] =  new QWidget();
35      /* 水平容器三 */
36      hWidget[2] =  new QWidget();
37
38      vWidget = new QWidget();
39      vBoxLayout = new QVBoxLayout();
40
41      /* 标签实例化 */
42      label[0] = new QLabel();
43      label[1] = new QLabel();
44
45      lineEdit = new QLineEdit();
46      comboBox = new QComboBox();
47      spinBox = new QSpinBox();
48      textBrowser = new QTextBrowser();
49
50      label[0]->setText("监听 IP 地址: ");
51      label[1]->setText("监听端口: ");
52
53      /* 设置标签根据文本文字大小自适应大小  */
54      label[0]->setSizePolicy(QSizePolicy::Fixed,
55                              QSizePolicy::Fixed);
56      label[1]->setSizePolicy(QSizePolicy::Fixed,
57                              QSizePolicy::Fixed);
58
59      /* 设置端口号的范围，注意不要与主机的已使用的端口号冲突 */
60      spinBox->setRange(10000, 99999);
61
62      pushButton[0]->setText("开始监听");
```

```
63      pushButton[1]->setText("停止监听");
64      pushButton[2]->setText("清空文本");
65      pushButton[3]->setText("发送消息");
66
67      /* 设置停止监听状态不可用 */
68      pushButton[1]->setEnabled(false);
69
70      /* 设置输入框默认的文本 */
71      lineEdit->setText("www.openedv.com 正点原子论坛");
72
73      /* 水平布局一添加内容 */
74      hBoxLayout[0]->addWidget(pushButton[0]);
75      hBoxLayout[0]->addWidget(pushButton[1]);
76      hBoxLayout[0]->addWidget(pushButton[2]);
77
78      /* 设置水平容器一的布局为水平布局一 */
79      hWidget[0]->setLayout(hBoxLayout[0]);
80
81      /* 水平布局二添加内容 */
82      hBoxLayout[1]->addWidget(label[0]);
83      hBoxLayout[1]->addWidget(comboBox);
84      hBoxLayout[1]->addWidget(label[1]);
85      hBoxLayout[1]->addWidget(spinBox);
86
87      /* 设置水平容器二的布局为水平布局二 */
88      hWidget[1]->setLayout(hBoxLayout[1]);
89
90      /* 水平布局三添加内容 */
91      hBoxLayout[2]->addWidget(lineEdit);
92      hBoxLayout[2]->addWidget(pushButton[3]);
93
94      /* 设置水平容器三的布局为水平布局一 */
95      hWidget[2]->setLayout(hBoxLayout[2]);
96
97      /* 垂直布局添加内容 */
98      vBoxLayout->addWidget(textBrowser);
99      vBoxLayout->addWidget(hWidget[1]);
100     vBoxLayout->addWidget(hWidget[0]);
101     vBoxLayout->addWidget(hWidget[2]);
102
103     /* 设置垂直容器的布局为垂直布局 */
104     vWidget->setLayout(vBoxLayout);
```

```
105
106     /* 居中显示 */
107     setCentralWidget(vWidget);
108
109     /* 获取本地 ip */
110     getLocalHostIP();
111
112     /* 信号槽连接 */
113     connect(pushButton[0], SIGNAL(clicked()),
114             this, SLOT(startListen()));
115     connect(pushButton[1], SIGNAL(clicked()),
116             this, SLOT(stopListen()));
117     connect(pushButton[2], SIGNAL(clicked()),
118             this, SLOT(clearTextBrowser()));
119     connect(pushButton[3], SIGNAL(clicked()),
120             this, SLOT(sendMessages()));
121     connect(tcpServer, SIGNAL(newConnection()),
122             this, SLOT(clientConnected()));
123 }
124
125 MainWindow::~MainWindow()
126 {
127 }
128
129 /* 新的客户端连接 */
130 void MainWindow::clientConnected()
131 {
132     /* 获取客户端的套接字 */
133     tcpSocket = tcpServer->nextPendingConnection();
134     /* 客户端的 ip 信息 */
135     QString ip = tcpSocket->peerAddress().toString();
136     /* 客户端的端口信息 */
137     quint16 port = tcpSocket->peerPort();
138     /* 在文本浏览框里显示出客户端的连接信息 */
139     textBrowser->append("客户端已连接");
140     textBrowser->append("客户端 ip 地址:"
141                         + ip);
142     textBrowser->append("客户端端口:"
143                         + QString::number(port));
144
145     connect(tcpSocket, SIGNAL(readyRead()),
146             this, SLOT(receiveMessages()));
147     connect(tcpSocket,
```

```
148             SIGNAL(stateChanged(QAbstractSocket::SocketState)),
149             this,
150             SLOT(socketStateChange(QAbstractSocket::SocketState)));
151 }
152
153 /* 获取本地 IP */
154 void MainWindow::getLocalHostIP()
155 {
156     // /* 获取主机的名称 */
157     // QString hostName = QHostInfo::localHostName();
158
159     // /* 主机的信息 */
160     // QHostInfo hostInfo = QHostInfo::fromName(hostName);
161
162     // /* ip 列表,addresses 返回 ip 地址列表，注意主机应能从路由器获取到
163     // * IP，否则可能返回空的列表（ubuntu 用此方法只能获取到环回 IP） */
164     // IPlist = hostInfo.addresses();
165     // qDebug()<<IPlist<<endl;
166
167     // /* 遍历 IPlist */
168     // foreach (QHostAddress ip, IPlist) {
169     //     if (ip.protocol() == QAbstractSocket::IPv4Protocol)
170     //         comboBox->addItem(ip.toString());
171     //    }
172
173     /* 获取所有的网络接口,
174      * QNetworkInterface 类提供主机的 IP 地址和网络接口的列表 */
175     QList<QNetworkInterface> list
176             = QNetworkInterface::allInterfaces();
177
178     /* 遍历 list */
179     foreach (QNetworkInterface interface, list) {
180
181         /* QNetworkAddressEntry 类存储 IP 地址子网掩码和广播地址 */
182         QList<QNetworkAddressEntry> entryList
183                 = interface.addressEntries();
184
185         /* 遍历 entryList */
186         foreach (QNetworkAddressEntry entry, entryList) {
187             /* 过滤 IPv6 地址，只留下 IPv4 */
188             if (entry.ip().protocol() ==
189                     QAbstractSocket::IPv4Protocol) {
```

```
190                 comboBox->addItem(entry.ip().toString());
191                 /* 添加到 IP 列表中 */
192                 IPlist<<entry.ip();
193             }
194         }
195     }
196 }
197
198 /* 开始监听 */
199 void MainWindow::startListen()
200 {
201     /* 需要判断当前主机是否有 IP 项 */
202     if (comboBox->currentIndex() != -1) {
203         qDebug()<<"start listen"<<endl;
204         tcpServer->listen(IPlist[comboBox->currentIndex()],
205                 spinBox->value());
206
207         /* 设置按钮与下拉列表框的状态 */
208         pushButton[0]->setEnabled(false);
209         pushButton[1]->setEnabled(true);
210         comboBox->setEnabled(false);
211         spinBox->setEnabled(false);
212
213         /* 在文本浏览框里显示出服务端 */
214         textBrowser->append("服务器 IP 地址: "
215                             + comboBox->currentText());
216         textBrowser->append("正在监听端口: "
217                             + spinBox->text());
218     }
219 }
220
221 /* 停止监听 */
222 void MainWindow::stopListen()
223 {
224     qDebug()<<"stop listen"<<endl;
225     /* 停止监听 */
226     tcpServer->close();
227
228     /* 如果是连接上了也应该断开，如果不断开客户端还能继续发送信息，
229      * 因为 socket 未断开，还在监听上一次端口 */
230     if (tcpSocket->state() == tcpSocket->ConnectedState)
231         tcpSocket->disconnectFromHost();
232
```

```
233     /* 设置按钮与下拉列表框的状态  */
234     pushButton[1]->setEnabled(false);
235     pushButton[0]->setEnabled(true);
236     comboBox->setEnabled(true);
237     spinBox->setEnabled(true);
238 
239     /* 将停止监听的信息添加到文本浏览框中 */
240     textBrowser->append("已停止监听端口："
241                         + spinBox->text());
242 }
243 
244 /* 清除文本浏览框里的内容 */
245 void MainWindow::clearTextBrowser()
246 {
247     /* 清除文本浏览器的内容 */
248     textBrowser->clear();
249 }
250 
251 /* 服务端接收消息 */
252 void MainWindow::receiveMessages()
253 {
254     /* 读取接收到的消息 */
255     QString messages = "客户端：" + tcpSocket->readAll();
256     textBrowser->append(messages);
257 }
258 
259 /* 服务端发送消息 */
260 void MainWindow::sendMessages()
261 {
262     if(NULL == tcpSocket)
263         return;
264 
265     /* 如果已经连接 */
266     if(tcpSocket->state() == tcpSocket->ConnectedState) {
267         /* 发送消息 */
268         tcpSocket->write(lineEdit->text().toUtf8().data());
269 
270         /* 在服务端插入发送的消息 */
271         textBrowser->append("服务端：" + lineEdit->text());
272     }
273 }
274 
```

```
275 /* 服务端状态改变 */
276 void MainWindow::socketStateChange(QAbstractSocket::SocketState
state)
277 {
278     switch (state) {
279     case QAbstractSocket::UnconnectedState:
280         textBrowser->append("scoket 状态: UnconnectedState");
281         break;
282     case QAbstractSocket::ConnectedState:
283         textBrowser->append("scoket 状态: ConnectedState");
284         break;
285     case QAbstractSocket::ConnectingState:
286         textBrowser->append("scoket 状态: ConnectingState");
287         break;
288     case QAbstractSocket::HostLookupState:
289         textBrowser->append("scoket 状态: HostLookupState");
290         break;
291     case QAbstractSocket::ClosingState:
292         textBrowser->append("scoket 状态: ClosingState");
293         break;
294     case QAbstractSocket::ListeningState:
295         textBrowser->append("scoket 状态: ListeningState");
296         break;
297     case QAbstractSocket::BoundState:
298         textBrowser->append("scoket 状态: BoundState");
299         break;
300     default:
301         break;
302     }
303 }
```

上面的代码主要是服务端开启监听，如果有客户端连到服务端，就会发射 newConnection()信号，同时也连接到接收消息的信号与槽函数。点击发送消息按钮就可以使用 tcpSocket 发送消息。注意发送消息和接收消息都是通过 tcpSocket 的 read()和 write()进行。

11.2.3 TCP 客户端应用实例

本例目的：了解 TCP 客户的使用。

例 09_tcpclient，TCP 客户端（难度：一般）。项目路径为 Qt/2/09_ tcpclient。本例大体流程：首先获取本地 IP 地址。创建一个 tcpSocket 套接字，然后用 tcpSocket 套接字使用 connectToHost 函数连接服务端的主机 IP 地址和端口，即可相互通信。

项目文件 08_tcpserver.pro 文件第一行添加的代码部分如下。

09_tcpclient.pro 编程后的代码

```
1   QT       += core gui network
2
3   greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5   CONFIG += c++11
6
7   # The following define makes your compiler emit warnings if you use
8   # any Qt feature that has been marked deprecated (the exact warnings
9   # depend on your compiler). Please consult the documentation of the
10  # deprecated API in order to know how to port your code away from it.
11  DEFINES += QT_DEPRECATED_WARNINGS
12
13  # You can also make your code fail to compile if it uses deprecated APIs.
14  # In order to do so, uncomment the following line.
15  # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16  #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18  SOURCES += \
19      main.cpp \
20      mainwindow.cpp
21
22  HEADERS += \
23      mainwindow.h
24
25  # Default rules for deployment.
26  qnx: target.path = /tmp/${TARGET}/bin
27  else: unix:!android: target.path = /opt/${TARGET}/bin
28  !isEmpty(target.path): INSTALLS += target
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   09_tcpclient
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-13
*****************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
```

```
4   #include <QMainWindow>
5   #include <QTcpServer>
6   #include <QTcpSocket>
7   #include <QVBoxLayout>
8   #include <QHBoxLayout>
9   #include <QPushButton>
10  #include <QTextBrowser>
11  #include <QLabel>
12  #include <QComboBox>
13  #include <QSpinBox>
14  #include <QHostInfo>
15  #include <QLineEdit>
16  #include <QNetworkInterface>
17  #include <QDebug>
18
19  class MainWindow : public QMainWindow
20  {
21      Q_OBJECT
22
23  public:
24      MainWindow(QWidget *parent = nullptr);
25      ~MainWindow();
26
27  private:
28      /* 通信套接字 */
29      QTcpSocket *tcpSocket;
30
31      /* 按钮 */
32      QPushButton *pushButton[4];
33
34      /* 标签文本 */
35      QLabel *label[2];
36
37      /* 水平容器 */
38      QWidget *hWidget[3];
39
40      /* 水平布局 */
41      QHBoxLayout *hBoxLayout[3];
42
43      /* 垂直容器 */
44      QWidget *vWidget;
45
46      /* 垂直布局 */
```

```
47      QVBoxLayout *vBoxLayout;
48
49      /* 文本浏览框 */
50      QTextBrowser *textBrowser;
51
52      /* 用于显示本地 ip */
53      QComboBox *comboBox;
54
55      /* 用于选择端口 */
56      QSpinBox  *spinBox;
57
58      /* 文本输入框 */
59      QLineEdit *lineEdit;
60
61      /* 存储本地的 ip 列表地址 */
62      QList<QHostAddress> IPlist;
63
64      /* 获取本地的所有 ip */
65      void getLocalHostIP();
66
67  private slots:
68      /* 连接 */
69      void toConnect();
70
71      /* 断开连接 */
72      void toDisConnect();
73
74      /* 已连接 */
75      void connected();
76
77      /* 已断开连接 */
78      void disconnected();
79
80      /* 清除文本框时的内容 */
81      void clearTextBrowser();
82
83      /* 接收到消息 */
84      void receiveMessages();
85
86      /* 发送消息 */
87      void sendMessages();
88
```

```
89      /* 连接状态改变槽函数 */
90      void socketStateChange(QAbstractSocket::SocketState);
91  };
92  #endif // MAINWINDOW_H
```

头文件里主要是声明界面用的元素，及一些槽函数。重点是声明 tcpSocket。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*********************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   09_tcpclient
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-13
*********************************************************************/
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    /* 设置主窗体的位置与大小 */
    this->setGeometry(0, 0, 800, 480);

    /* tcp 套接字 */
    tcpSocket = new QTcpSocket(this);

    /* 开始监听按钮 */
    pushButton[0] = new QPushButton();
    /* 停止监听按钮 */
    pushButton[1] = new QPushButton();
    /* 清空聊天文本按钮 */
    pushButton[2] = new QPushButton();
    /* 发送消息按钮 */
    pushButton[3] = new QPushButton();

    /* 水平布局一 */
    hBoxLayout[0] = new QHBoxLayout();
    /* 水平布局二 */
    hBoxLayout[1] = new QHBoxLayout();
    /* 水平布局三 */
    hBoxLayout[2] = new QHBoxLayout();
```

```
27      /* 水平布局四 */
28      hBoxLayout[3] = new QHBoxLayout();
29
30      /* 水平容器一 */
31      hWidget[0] =  new QWidget();
32      /* 水平容器二 */
33      hWidget[1] =  new QWidget();
34      /* 水平容器三 */
35      hWidget[2] =  new QWidget();
36
37
38      vWidget = new QWidget();
39      vBoxLayout = new QVBoxLayout();
40
41      /* 标签实例化 */
42      label[0] = new QLabel();
43      label[1] = new QLabel();
44
45      lineEdit = new QLineEdit();
46      comboBox = new QComboBox();
47      spinBox = new QSpinBox();
48      textBrowser = new QTextBrowser();
49
50      label[0]->setText("服务器地址：");
51      label[1]->setText("服务器端口：");
52
53      /* 设置标签根据文本文字大小自适应大小  */
54      label[0]->setSizePolicy(QSizePolicy::Fixed,
55                              QSizePolicy::Fixed);
56      label[1]->setSizePolicy(QSizePolicy::Fixed,
57                              QSizePolicy::Fixed);
58
59      /* 设置端口号的范围，注意不要与主机的已使用的端口号冲突 */
60      spinBox->setRange(10000, 99999);
61
62      pushButton[0]->setText("连接服务器");
63      pushButton[1]->setText("断开连接");
64      pushButton[2]->setText("清空文本");
65      pushButton[3]->setText("发送消息");
66
67      /* 设置停止监听状态不可用 */
68      pushButton[1]->setEnabled(false);
```

```
69
70      /* 设置输入框默认的文本 */
71      lineEdit->setText("广州星翼电子科技有限公司");
72
73      /* 水平布局一添加内容 */
74      hBoxLayout[0]->addWidget(pushButton[0]);
75      hBoxLayout[0]->addWidget(pushButton[1]);
76      hBoxLayout[0]->addWidget(pushButton[2]);
77
78      /* 设置水平容器的布局为水平布局一 */
79      hWidget[0]->setLayout(hBoxLayout[0]);
80
81      hBoxLayout[1]->addWidget(label[0]);
82      hBoxLayout[1]->addWidget(comboBox);
83      hBoxLayout[1]->addWidget(label[1]);
84      hBoxLayout[1]->addWidget(spinBox);
85
86      /* 设置水平容器的布局为水平布局二 */
87      hWidget[1]->setLayout(hBoxLayout[1]);
88
89      /* 水平布局三添加内容 */
90      hBoxLayout[2]->addWidget(lineEdit);
91      hBoxLayout[2]->addWidget(pushButton[3]);
92
93      /* 设置水平容器三的布局为水平布局一 */
94      hWidget[2]->setLayout(hBoxLayout[2]);
95
96      /* 垂直布局添加内容 */
97      vBoxLayout->addWidget(textBrowser);
98      vBoxLayout->addWidget(hWidget[1]);
99      vBoxLayout->addWidget(hWidget[0]);
100     vBoxLayout->addWidget(hWidget[2]);
101
102     /* 设置垂直容器的布局为垂直布局 */
103     vWidget->setLayout(vBoxLayout);
104
105     /* 居中显示 */
106     setCentralWidget(vWidget);
107
108     /* 获取本地 ip */
109     getLocalHostIP();
110
```

```
111     /* 信号槽连接 */
112     connect(pushButton[0], SIGNAL(clicked()),
113             this, SLOT(toConnect()));
114     connect(pushButton[1], SIGNAL(clicked()),
115             this, SLOT(toDisConnect()));
116     connect(pushButton[2], SIGNAL(clicked()),
117             this, SLOT(clearTextBrowser()));
118     connect(pushButton[3], SIGNAL(clicked()),
119             this, SLOT(sendMessages()));
120     connect(tcpSocket, SIGNAL(connected()),
121             this, SLOT(connected()));
122     connect(tcpSocket, SIGNAL(disconnected()),
123             this, SLOT(disconnected()));
124     connect(tcpSocket, SIGNAL(readyRead()),
125             this, SLOT(receiveMessages()));
126     connect(tcpSocket,
127             SIGNAL(stateChanged(QAbstractSocket::SocketState)),
128             this,
129             SLOT(socketStateChange(QAbstractSocket::SocketState)));
130 }
131
132 MainWindow::~MainWindow()
133 {
134 }
135
136 void MainWindow::toConnect()
137 {
138     /* 如果连接状态还没有连接 */
139     if (tcpSocket->state() != tcpSocket->ConnectedState) {
140         /* 指定 IP 地址和端口连接 */
141         tcpSocket->connectToHost(IPlist[comboBox->currentIndex()],
142                 spinBox->value());
143     }
144 }
145
146 void MainWindow::toDisConnect()
147 {
148     /* 断开连接 */
149     tcpSocket->disconnectFromHost();
150
151     /* 关闭 socket*/
152     tcpSocket->close();
153 }
```

```
154
155 void MainWindow::connected()
156 {
157     /* 显示已经连接 */
158     textBrowser->append("已经连上服务端");
159
160     /* 设置按钮与下拉列表框的状态 */
161     pushButton[0]->setEnabled(false);
162     pushButton[1]->setEnabled(true);
163     comboBox->setEnabled(false);
164     spinBox->setEnabled(false);
165 }
166
167 void MainWindow::disconnected()
168 {
169     /* 显示已经断开连接 */
170     textBrowser->append("已经断开服务端");
171
172     /* 设置按钮与下拉列表框的状态  */
173     pushButton[1]->setEnabled(false);
174     pushButton[0]->setEnabled(true);
175     comboBox->setEnabled(true);
176     spinBox->setEnabled(true);
177 }
178
179 /* 获取本地 IP */
180 void MainWindow::getLocalHostIP()
181 {
182     // /* 获取主机的名称 */
183     // QString hostName = QHostInfo::localHostName();
184
185     // /* 主机的信息 */
186     // QHostInfo hostInfo = QHostInfo::fromName(hostName);
187
188     // /* ip 列表,addresses 返回 ip 地址列表，注意主机应能从路由器获取到
189     // * IP，否则可能返回空的列表（ubuntu 用此方法只能获取到环回 IP） */
190     // IPlist = hostInfo.addresses();
191     // qDebug()<<IPlist<<endl;
192
193     // /* 遍历 IPlist */
194     // foreach (QHostAddress ip, IPlist) {
195     //     if (ip.protocol() == QAbstractSocket::IPv4Protocol)
```

```
196     //          comboBox->addItem(ip.toString());
197     //    }
198
199     /* 获取所有的网络接口,
200      * QNetworkInterface 类提供主机的 IP 地址和网络接口的列表 */
201     QList<QNetworkInterface> list
202           = QNetworkInterface::allInterfaces();
203
204     /* 遍历 list */
205     foreach (QNetworkInterface interface, list) {
206
207         /* QNetworkAddressEntry 类存储 IP 地址子网掩码和广播地址 */
208         QList<QNetworkAddressEntry> entryList
209               = interface.addressEntries();
210
211         /* 遍历 entryList */
212         foreach (QNetworkAddressEntry entry, entryList) {
213             /* 过滤 IPv6 地址，只留下 IPv4 */
214             if (entry.ip().protocol() ==
215                     QAbstractSocket::IPv4Protocol) {
216                 comboBox->addItem(entry.ip().toString());
217                 /* 添加到 IP 列表中 */
218                 IPlist<<entry.ip();
219             }
220         }
221     }
222 }
223
224 /* 清除文本浏览框里的内容 */
225 void MainWindow::clearTextBrowser()
226 {
227     /* 清除文本浏览器的内容 */
228     textBrowser->clear();
229 }
230
231 /* 客户端接收消息 */
232 void MainWindow::receiveMessages()
233 {
234     /* 读取接收到的消息 */
235     QString messages = tcpSocket->readAll();
236     textBrowser->append("服务端: " + messages);
237 }
```

```
238
239 /* 客户端发送消息 */
240 void MainWindow::sendMessages()
241 {
242     if(NULL == tcpSocket)
243         return;
244
245     if(tcpSocket->state() == tcpSocket->ConnectedState) {
246         /* 客户端显示发送的消息 */
247         textBrowser->append("客户端: " + lineEdit->text());
248
249         /* 发送消息 */
250         tcpSocket->write(lineEdit->text().toUtf8().data());
251     }
252 }
253
254 /* 客户端状态改变 */
255 void MainWindow::socketStateChange(QAbstractSocket::SocketState
state)
256 {
257     switch (state) {
258     case QAbstractSocket::UnconnectedState:
259         textBrowser->append("scoket 状态: UnconnectedState");
260         break;
261     case QAbstractSocket::ConnectedState:
262         textBrowser->append("scoket 状态: ConnectedState");
263         break;
264     case QAbstractSocket::ConnectingState:
265         textBrowser->append("scoket 状态: ConnectingState");
266         break;
267     case QAbstractSocket::HostLookupState:
268         textBrowser->append("scoket 状态: HostLookupState");
269         break;
270     case QAbstractSocket::ClosingState:
271         textBrowser->append("scoket 状态: ClosingState");
272         break;
273     case QAbstractSocket::ListeningState:
274         textBrowser->append("scoket 状态: ListeningState");
275         break;
276     case QAbstractSocket::BoundState:
277         textBrowser->append("scoket 状态: BoundState");
278         break;
```

```
279     default:
280         break;
281     }
282 }
```

上面的代码主要是客户端开使用 connectToHost 通过 IP 地址和端口与服务端连接，如果连接成功，就会发射 connected ()信号，同时也连接到接收消息的信号与槽函数。点击发送消息按钮就可以使用 tcpSocket 发送消息。注意发送消息和接收消息都是通过 tcpSocket 的 read()和 write()进行。

11.2.4 程序运行效果

开启服务端后，需要选择本地监听的 IP 地址和监听的端口（特别需要注意，不要选择监听的端口与本地主机的已经使用的端口，所以笔者把端口号设置的特别大，查看本地已经使用的端口号可以使用 netstat 指令。）

启动客户端后，选择需要连接的服务器 IP 地址和服务器监听的端口。点击连接后就可以相互发送消息了。

注意服务端和客户端都本例都是选择了本地环回 IP 127.0.0.1 测试。也可以选择本地的其他 IP 地址进行测试。

TCP 服务端：

TCP 客户端：

11.3 UDP 通信

11.3.1 UDP 简介

UDP（User Datagram Protocol 即用户数据报协议）是一个**轻量级的，不可靠的，面向数据报的无连接协议**。我们日常生活中使用的 QQ，其聊天时的文字内容是使用 UDP 协议进行消息发送的。因为 QQ 有很多用户，发送的大部分都是短消息，要求能及时响应，并且对安全性要求不是很高的情况下使用 UDP 协议。但是 QQ 也并不是完全使用 UDP 协议，比如我们在传输文件时就会选择 TCP 协议，保证文件正确传输。像 QQ 语音和 QQ 视频通话，UDP 的优势就很突出了。在选择使用协议的时候，选择 UDP 必须要谨慎。在网络质量令人十分不满意的环境下，UDP 协议数据包丢失会比较严重。但是由于 UDP 的特性：它**不属于连接型协议**，因而具有资源消耗小，处理速度快的优点，所以通常音频、视频和普通数据在传送时使用 UDP 较多，因为它们即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。

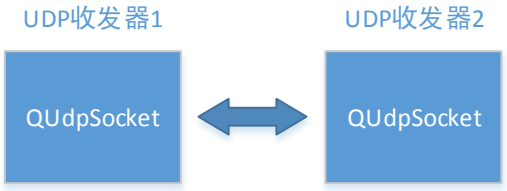
QUdpSocket 类提供了一个 UDP 套接字。QUdpSocket 是 QAbstractSocket 的子类，允许发送和接收 UDP 数据报。使用该类最常见的方法是使用 bind()绑定到一个地址和端口，然后调用 writeDatagram()和 readDatagram() / receiveDatagram()来传输数据。注意发送数据一般少于 512 字节。如果发送多于 512 字节的数据，即使我们发送成功了，也会在 IP 层被分片（分成小片段）。

如果您想使用标准的 QIODevice 函数 read()、readLine()、write()等，您必须首先通过调用 connectToHost()将套接字直接连接到对等体。每次将数据报写入网络时，套接字都会发出 bytesWritten()信号。

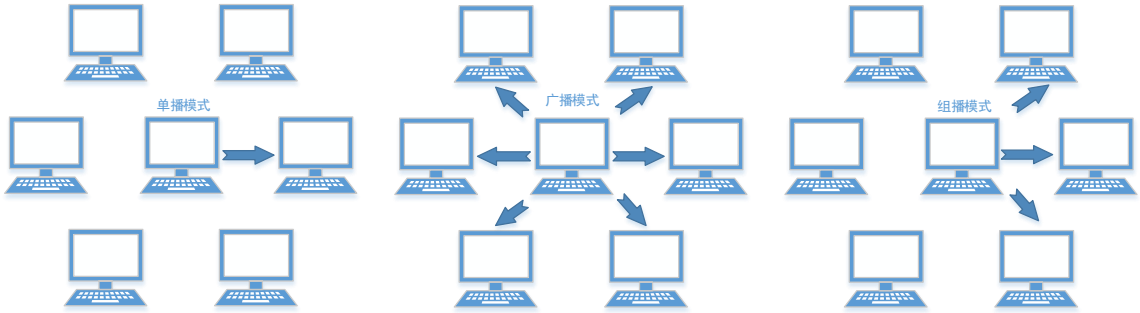
如果您只是想发送数据报，您不需要调用 bind()。readyRead()信号在数据报到达时发出。在这种情况下，hasPendingDatagrams()返回 true。调用 pendingDatagramSize()来获取第一个待处

理数据报的大小，并调用 readDatagram()或 receiveDatagram()来读取它。注意:当您接收到 readyRead()信号时，一个传入的数据报应该被读取，否则这个信号将不会被发送到下一个数据报。

UDP 通信示意图如下。重点是 QUdpSocket 类，已经为我们提供了 UDP 通信的基础。



UDP 消息传送有三种模式，分别是单播、广播和组播三种模式。



- 单播（unicast）：单播用于两个主机之间的端对端通信，需要知道对方的 IP 地址与端口。
- 广播(broadcast)：广播 UDP 与单播 UDP 的区别就是 IP 地址不同，广播一般使用广播地址 255.255.255.255，将消息发送到在同一广播（也就是局域网内同一网段）网络上的每个主机。值得强调的是：本地广播信息是不会被路由器转发。当然这是十分容易理解的，因为如果路由器转发了广播信息，那么势必会引起网络瘫痪。这也是为什么 IP 协议的设计者故意没有定义互联网范围的广播机制。广播地址通常用于在网络游戏中处于同一本地网络的玩家之间交流状态信息等。其实广播顾名思义，就是想局域网内所有的人说话，但是广播还是要指明接收者的端口号的，因为不可能接受者的所有端口都来收听广播。
- 组播（multicast）：组播（多点广播），也称为“多播”，将网络中同一业务类型主机进行了逻辑上的分组，进行数据收发的时候其数据仅仅在同一分组中进行，其他的主机没有加入此分组不能收发对应的数据。在广域网上广播的时候，其中的交换机和路由器只向需要获取数据的主机复制并转发数据。主机可以向路由器请求加入或退出某个组，网络中的路由器和交换机有选择地复制并传输数据，将数据仅仅传输给组内的主机。多播的这种功能，可以一次将数据发送到多个主机，又能保证不影响其他不需要（未加入组）的主机的其他通信。

注意：单播一样和多播是允许在广域网即 Internet 上进行传输的，而广播仅仅在同一局域网上才能进行。

11.3.2 UDP 单播与广播

广播 UDP 与单播 UDP 的区别就是 IP 地址不同，所以我们的实例可以写成一个。我们可以这么理解，单播实际上是通信上对应一对一，广播则是一对多（多，这里指广播地址内的所有主机）。

11.3.2.1 应用实例

本例目的：了解 QUdpSocket 单播和广播使用。

例 10_udp_unicast_broadcast，UDP 单播与广播应用（难度：一般）。项目路径为 Qt/2/10_udp_unicast_broadcast。本例大体流程首先获取本地 IP 地址。创建一个 udpSocket 套接字，然后绑定本地主机的端口（也就是监听端口）。我们可以使用 QUdpSocket 类提供的读写函数 readDatagram 和 writeDatagram，知道目标 IP 地址和端口，即可完成消息的接收与发送。

项目文件 10_udp_unicast_broadcast.pro 文件第一行添加的代码部分如下。

10_udp_unicast_broadcast.pro 编程后的代码

```
1  QT       += core gui network
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
13 # You can also make your code fail to compile if it uses deprecated APIs.
14 # In order to do so, uncomment the following line.
15 # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18 SOURCES += \
19     main.cpp \
20     mainwindow.cpp
21
22 HEADERS += \
23     mainwindow.h
24
25 # Default rules for deployment.
26 qnx: target.path = /tmp/${TARGET}/bin
27 else: unix:!android: target.path = /opt/${TARGET}/bin
```

```
28 !isEmpty(target.path): INSTALLS += target
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   10_udp_unicast_broadcast
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-14
*****************************************************************/
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QUdpSocket>
6  #include <QVBoxLayout>
7  #include <QHBoxLayout>
8  #include <QPushButton>
9  #include <QTextBrowser>
10 #include <QLabel>
11 #include <QComboBox>
12 #include <QSpinBox>
13 #include <QHostInfo>
14 #include <QLineEdit>
15 #include <QNetworkInterface>
16 #include <QDebug>
17
18 class MainWindow : public QMainWindow
19 {
20     Q_OBJECT
21
22 public:
23     MainWindow(QWidget *parent = nullptr);
24     ~MainWindow();
25
26 private:
27     /* Udp 通信套接字 */
28     QUdpSocket *udpSocket;
29
30     /* 按钮 */
31     QPushButton *pushButton[5];
32
```

```
33      /* 标签文本 */
34      QLabel *label[3];
35
36      /* 水平容器 */
37      QWidget *hWidget[3];
38
39      /* 水平布局 */
40      QHBoxLayout *hBoxLayout[3];
41
42      /* 垂直容器 */
43      QWidget *vWidget;
44
45      /* 垂直布局 */
46      QVBoxLayout *vBoxLayout;
47
48      /* 文本浏览框 */
49      QTextBrowser *textBrowser;
50
51      /* 用于显示本地 ip */
52      QComboBox *comboBox;
53
54      /* 用于选择端口 */
55      QSpinBox  *spinBox[2];
56
57      /* 文本输入框 */
58      QLineEdit *lineEdit;
59
60      /* 存储本地的 ip 列表地址 */
61      QList<QHostAddress> IPlist;
62
63      /* 获取本地的所有 ip */
64      void getLocalHostIP();
65
66  private slots:
67      /* 绑定端口 */
68      void bindPort();
69
70      /* 解绑端口 */
71      void unbindPort();
72
73      /* 清除文本框时的内容 */
74      void clearTextBrowser();
```

```
75
76      /* 接收到消息 */
77      void receiveMessages();
78
79      /* 发送消息 */
80      void sendMessages();
81
82      /* 广播消息 */
83      void sendBroadcastMessages();
84
85      /* 连接状态改变槽函数 */
86      void socketStateChange(QAbstractSocket::SocketState);
87  };
88  #endif // MAINWINDOW_H
```

头文件里主要是声明界面用的元素，及一些槽函数。重点是声明 udpSocket。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*********************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   10_udp_unicast_broadcast
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-14
*********************************************************************/
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗体的位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* udp 套接字 */
10      udpSocket = new QUdpSocket(this);
11
12      /* 绑定端口按钮 */
13      pushButton[0] = new QPushButton();
14      /* 解绑端口按钮 */
15      pushButton[1] = new QPushButton();
16      /* 清空聊天文本按钮 */
```

```
17      pushButton[2] = new QPushButton();
18      /* 发送消息按钮 */
19      pushButton[3] = new QPushButton();
20      /* 广播消息按钮 */
21      pushButton[4] = new QPushButton();
22
23      /* 水平布局一 */
24      hBoxLayout[0] = new QHBoxLayout();
25      /* 水平布局二 */
26      hBoxLayout[1] = new QHBoxLayout();
27      /* 水平布局三 */
28      hBoxLayout[2] = new QHBoxLayout();
29      /* 水平布局四 */
30      hBoxLayout[3] = new QHBoxLayout();
31
32      /* 水平容器一 */
33      hWidget[0] =  new QWidget();
34      /* 水平容器二 */
35      hWidget[1] =  new QWidget();
36      /* 水平容器三 */
37      hWidget[2] =  new QWidget();
38
39
40      vWidget = new QWidget();
41      vBoxLayout = new QVBoxLayout();
42
43      /* 标签实例化 */
44      label[0] = new QLabel();
45      label[1] = new QLabel();
46      label[2] = new QLabel();
47
48      lineEdit = new QLineEdit();
49      comboBox = new QComboBox();
50      spinBox[0] = new QSpinBox();
51      spinBox[1] = new QSpinBox();
52      textBrowser = new QTextBrowser();
53
54      label[0]->setText("目标 IP 地址：");
55      label[1]->setText("绑定端口：");
56      label[2]->setText("目标端口：");
57
58      /* 设置标签根据文本文字大小自适应大小  */
```

```
59      label[0]->setSizePolicy(QSizePolicy::Fixed,
60                              QSizePolicy::Fixed);
61      label[1]->setSizePolicy(QSizePolicy::Fixed,
62                              QSizePolicy::Fixed);
63      label[2]->setSizePolicy(QSizePolicy::Fixed,
64                              QSizePolicy::Fixed);
65
66      /* 设置端口号的范围，注意不要与主机的已使用的端口号冲突 */
67      spinBox[0]->setRange(10000, 99999);
68      spinBox[1]->setRange(10000, 99999);
69
70      pushButton[0]->setText("绑定端口");
71      pushButton[1]->setText("解除绑定");
72      pushButton[2]->setText("清空文本");
73      pushButton[3]->setText("发送消息");
74      pushButton[4]->setText("广播消息");
75
76      /* 设置停止监听状态不可用 */
77      pushButton[1]->setEnabled(false);
78
79      /* 设置输入框默认的文本 */
80      lineEdit->setText("您好！");
81
82      /* 水平布局一添加内容 */
83      hBoxLayout[0]->addWidget(pushButton[0]);
84      hBoxLayout[0]->addWidget(pushButton[1]);
85      hBoxLayout[0]->addWidget(pushButton[2]);
86
87      /* 设置水平容器的布局为水平布局一 */
88      hWidget[0]->setLayout(hBoxLayout[0]);
89
90      hBoxLayout[1]->addWidget(label[0]);
91      hBoxLayout[1]->addWidget(comboBox);
92      hBoxLayout[1]->addWidget(label[1]);
93      hBoxLayout[1]->addWidget(spinBox[0]);
94      hBoxLayout[1]->addWidget(label[2]);
95      hBoxLayout[1]->addWidget(spinBox[1]);
96
97      /* 设置水平容器的布局为水平布局二 */
98      hWidget[1]->setLayout(hBoxLayout[1]);
99
100     /* 水平布局三添加内容 */
```

```
101     hBoxLayout[2]->addWidget(lineEdit);
102     hBoxLayout[2]->addWidget(pushButton[3]);
103     hBoxLayout[2]->addWidget(pushButton[4]);
104
105     /* 设置水平容器三的布局为水平布局一 */
106     hWidget[2]->setLayout(hBoxLayout[2]);
107
108     /* 垂直布局添加内容 */
109     vBoxLayout->addWidget(textBrowser);
110     vBoxLayout->addWidget(hWidget[1]);
111     vBoxLayout->addWidget(hWidget[0]);
112     vBoxLayout->addWidget(hWidget[2]);
113
114     /* 设置垂直容器的布局为垂直布局 */
115     vWidget->setLayout(vBoxLayout);
116
117     /* 居中显示 */
118     setCentralWidget(vWidget);
119
120     /* 获取本地 ip */
121     getLocalHostIP();
122
123     /* 信号槽连接 */
124     connect(pushButton[0], SIGNAL(clicked()),
125             this, SLOT(bindPort()));
126     connect(pushButton[1], SIGNAL(clicked()),
127             this, SLOT(unbindPort()));
128     connect(pushButton[2], SIGNAL(clicked()),
129             this, SLOT(clearTextBrowser()));
130     connect(pushButton[3], SIGNAL(clicked()),
131             this, SLOT(sendMessages()));
132     connect(pushButton[4], SIGNAL(clicked()),
133             this, SLOT(sendBroadcastMessages()));
134     connect(udpSocket, SIGNAL(readyRead()),
135             this, SLOT(receiveMessages()));
136     connect(udpSocket,
137             SIGNAL(stateChanged(QAbstractSocket::SocketState)),
138             this,
139             SLOT(socketStateChange(QAbstractSocket::SocketState)));
140 }
141
142 MainWindow::~MainWindow()
143 {
```

```
144 }
145
146 void MainWindow::bindPort()
147 {
148     quint16 port = spinBox[0]->value();
149
150     /* 绑定端口需要在 socket 的状态为 UnconnectedState */
151     if (udpSocket->state() != QAbstractSocket::UnconnectedState)
152         udpSocket->close();
153
154     if (udpSocket->bind(port)) {
155         textBrowser->append("已经成功绑定端口: "
156                             + QString::number(port));
157
158         /* 设置界面中的元素的可用状态 */
159         pushButton[0]->setEnabled(false);
160         pushButton[1]->setEnabled(true);
161         spinBox[1]->setEnabled(false);
162     }
163 }
164
165 void MainWindow::unbindPort()
166 {
167     /* 解绑，不再监听 */
168     udpSocket->abort();
169
170     /* 设置界面中的元素的可用状态 */
171     pushButton[0]->setEnabled(true);
172     pushButton[1]->setEnabled(false);
173     spinBox[1]->setEnabled(true);
174 }
175
176 /* 获取本地 IP */
177 void MainWindow::getLocalHostIP()
178 {
179     // /* 获取主机的名称 */
180     // QString hostName = QHostInfo::localHostName();
181
182     // /* 主机的信息 */
183     // QHostInfo hostInfo = QHostInfo::fromName(hostName);
184
185     // /* ip 列表,addresses 返回 ip 地址列表，注意主机应能从路由器获取到
186     // * IP，否则可能返回空的列表（ubuntu 用此方法只能获取到环回 IP） */
```

```
187     // IPlist = hostInfo.addresses();
188     // qDebug()<<IPlist<<endl;
189
190     // /* 遍历 IPlist */
191     // foreach (QHostAddress ip, IPlist) {
192     //     if (ip.protocol() == QAbstractSocket::IPv4Protocol)
193     //         comboBox->addItem(ip.toString());
194     //    }
195
196     /* 获取所有的网络接口,
197      * QNetworkInterface 类提供主机的 IP 地址和网络接口的列表 */
198     QList<QNetworkInterface> list
199             = QNetworkInterface::allInterfaces();
200
201     /* 遍历 list */
202     foreach (QNetworkInterface interface, list) {
203
204         /* QNetworkAddressEntry 类存储 IP 地址子网掩码和广播地址 */
205         QList<QNetworkAddressEntry> entryList
206                 = interface.addressEntries();
207
208         /* 遍历 entryList */
209         foreach (QNetworkAddressEntry entry, entryList) {
210             /* 过滤 IPv6 地址，只留下 IPv4 */
211             if (entry.ip().protocol() ==
212                     QAbstractSocket::IPv4Protocol) {
213                 comboBox->addItem(entry.ip().toString());
214                 /* 添加到 IP 列表中 */
215                 IPlist<<entry.ip();
216             }
217         }
218     }
219 }
220
221 /* 清除文本浏览框里的内容 */
222 void MainWindow::clearTextBrowser()
223 {
224     /* 清除文本浏览器的内容 */
225     textBrowser->clear();
226 }
227
228 /* 客户端接收消息 */
229 void MainWindow::receiveMessages()
```

```
230 {
231     /* 局部变量，用于获取发送者的 IP 和端口 */
232     QHostAddress peerAddr;
233     quint16 peerPort;
234 
235     /* 如果有数据已经准备好 */
236     while (udpSocket->hasPendingDatagrams()) {
237         /* udpSocket 发送的数据报是 QByteArray 类型的字节数组 */
238         QByteArray datagram;
239 
240         /* 重新定义数组的大小 */
241         datagram.resize(udpSocket->pendingDatagramSize());
242 
243         /* 读取数据，并获取发送方的 IP 地址和端口 */
244         udpSocket->readDatagram(datagram.data(),
245                                 datagram.size(),
246                                 &peerAddr,
247                                 &peerPort);
248         /* 转为字符串 */
249         QString str = datagram.data();
250 
251         /* 显示信息到文本浏览框窗口 */
252         textBrowser->append("接收来自"
253                             + peerAddr.toString()
254                             + ":"
255                             + QString::number(peerPort)
256                             + str);
257     }
258 }
259 
260 /* 客户端发送消息 */
261 void MainWindow::sendMessages()
262 {
263     /* 文本浏览框显示发送的信息 */
264     textBrowser->append("发送：" + lineEdit->text());
265 
266     /* 要发送的信息，转为 QByteArray 类型字节数组，数据一般少于 512 个字节 */
267     QByteArray data = lineEdit->text().toUtf8();
268 
269     /* 要发送的目标 Ip 地址 */
270     QHostAddress peerAddr = IPlist[comboBox->currentIndex()];
271 
```

```
272     /* 要发送的目标端口号 */
273     quint16 peerPort = spinBox[1]->value();
274
275     /* 发送消息 */
276     udpSocket->writeDatagram(data, peerAddr, peerPort);
277 }
278
279 void MainWindow::sendBroadcastMessages()
280 {
281     /* 文本浏览框显示发送的信息 */
282     textBrowser->append("发送: " + lineEdit->text());
283
284     /* 要发送的信息，转为 QByteArray 类型字节数组，数据一般少于 512 个字节 */
285     QByteArray data = lineEdit->text().toUtf8();
286
287     /* 广播地址，一般为 255.255.255.255，
288      * 同一网段内监听目标端口的程序都会接收到消息 */
289     QHostAddress peerAddr = QHostAddress::Broadcast;
290
291     /* 要发送的目标端口号 */
292     quint16 peerPort = spinBox[1]->text().toInt();
293
294     /* 发送消息 */
295     udpSocket->writeDatagram(data, peerAddr, peerPort);
296 }
297 /* socket 状态改变 */
298 void MainWindow::socketStateChange(QAbstractSocket::SocketState
state)
299 {
300     switch (state) {
301     case QAbstractSocket::UnconnectedState:
302         textBrowser->append("scoket 状态: UnconnectedState");
303         break;
304     case QAbstractSocket::ConnectedState:
305         textBrowser->append("scoket 状态: ConnectedState");
306         break;
307     case QAbstractSocket::ConnectingState:
308         textBrowser->append("scoket 状态: ConnectingState");
309         break;
310     case QAbstractSocket::HostLookupState:
311         textBrowser->append("scoket 状态: HostLookupState");
312         break;
```

```
313     case QAbstractSocket::ClosingState:
314         textBrowser->append("scoket 状态：ClosingState");
315         break;
316     case QAbstractSocket::ListeningState:
317         textBrowser->append("scoket 状态：ListeningState");
318         break;
319     case QAbstractSocket::BoundState:
320         textBrowser->append("scoket 状态：BoundState");
321         break;
322     default:
323         break;
324     }
325 }
```

第 146~163 行，绑定端口。使用 bind 方法，即可绑定一个端口。注意我们绑定的端口不能和主机已经使用的端口冲突！

第 165~174 行，解绑端口。使用 abort 方法即可解绑。

第 229~258 行，接收消息，注意接收消息是 QByteArray 字节数组。读数组使用的是 readDatagram 方法，在 readDatagram 方法里可以获取对方的套接字 IP 地址与端口号。

第 261~277 行，单播消息，需要知道目标 IP 与目标端口号。即可用 writeDatagram 方法发送消息。

第 279~296 行，广播消息与单播消息不同的是将目标 IP 地址换成了广播地址，一般广播地址为 255.255.255.255。

11.3.2.2 程序运行效果

本实例可以做即是发送者，也是接收者。如果在同一台主机同一个系统里运行两个本例程序。不能绑定同一个端口！否则会冲突！当您想测试在同一局域网内不同主机上运行此程序，那么绑定的端口号可以相同。

本例设置目标 IP 地址为 127.0.0.1，此 IP 地址是 Ubuntu/Windows 上的环回 IP 地址，可以用于无网络时测试。绑定端口号与目标端口号相同，也就是说，此程序正在监听端口号为 10000 的数据，此程序也向目标 IP 地址 127.0.0.1 的 10000 端口号发送数据，实际上此程序就完成了自发自收。

当我们点击发送消息按钮时，文本消息窗口显示发送的数据“您好!”，同时接收到由本地 IP 127.0.0.1 发出的数据“您好!”。其中 ffff:是通信套接字的标识。呵呵！您可能会问为什么不是本主机的其它地址如（192.168.1.x）发出的呢？因为我们选择了目标的 IP 地址为 127.0.0.1，那么要与此目标地址通信，必须使用相同网段的 IP 设备与之通信。注意不能用本地环回发送消息到其他主机上。因为本地环回 IP 只适用于本地主机上的 IP 通信。

当我们点击广播消息按钮时，广播发送的目标 IP 地址变成了广播地址 255.255.255.255。那么我们将收到从本地 IP 地址 192.168.x.x 的数据。如下图，收到了从 192.168.1.129 发送过来的

数据。因为环回 IP 127.0.0.1 的广播地址为 255.0.0.0，所以要与 255.255.255.255 的网段里的 IP 通信数据必须是由 192.168.x.x 上发出的。如果其他同一网段上的其他主机正在监听目标端口，那么它们将同时收到消息。这也验证了上一小节为什么会从 127.0.0.1 发送数据。

本例不难，可能有点绕，大家多参考资料理解理解，知识点有点多，如果没有些通信基础的话，我们需要慢慢吃透。

11.3.3 UDP 组播

通常，在传统的网络通讯中，有两种方式，一种是源主机和目标主机两台主机之间进行的“一对一”的通讯方式，即单播，第二种是一台源主机与网络中所有其他主机之间进行的通讯，即广播。那么，如果需要将信息从源主机发送到网络中的多个目标主机，要么采用广播方式，这样网络中所有主机都会收到信息，要么，采用单播方式，由源主机分别向各个不同目标主机发送信息。可以看出来，在广播方式下，信息会发送到不需要该信息的主机从而浪费带宽资源，甚至引起广播风暴：而单播方式下，会因为数据包的多次重复而浪费带宽资源，同时，源主机的负荷会因为多次的数据复制而加大，所以，单播与广播对于多点发送问题有缺陷。在此情况下，组播技术就应用而生了。

组播类似于 QQ 群，如果把腾讯向 QQ 每个用户发送推送消息比作广播，那么组播就像是 QQ 群一样，只有群内的用户才能收到消息。想要收到消息，我们得先加群。

一个 D 类 IP 地址的第一个字节必须以“1110”开始，D 类 IP 地址不分网络地址和主机地址，是一个专门保留的地址，其地址范围为 224.0.0.0～239.255.255.255。D 类 IP 地址主要用于**多点广播**（Multicast，也称为**多播（组播）**）之中作为多播组 IP 地址。其中，多播组 IP 地址让源主机能够将分组发送给网络中的一组主机，属于多播组的主机将被分配一个多播组 IP 地址。

由于多播组 lP 地址标识了一组主机（也称为主机组），因此多播组 IP 地址只能作为目标地址，源地址总是为单播地址。

- 224.0.0.0～224.0.0.255 为预留的组播地址（永久组地址），地址 224.0.0.0 保留不做分配，其它地址供路由协议使用。
- 224.0.1.0～238.255.255.255 为用户可用的组播地址（临时组地址），全网范围内有效。
- 239.0.0.0～239.255.255.255 为本地管理组播地址，**仅在特定的本地范围内有效**。

通过以上的信息，我们只需要关注，哪些组播地址可以被我们在本地主机使用即可。在家庭网络和办公网络局域网内使用 UDP 组播功能，那么可用的组播地址范围是 **239.0.0.0～239.255.255.255**。

QUdpSocket 类支持 UDP 组播，提供了 joinMulticastGroup 方法使本地主机加入多播组，leaveMulticastGroup 离开多播组。其他绑定端口，发送接收功能与 UDP 单播和广播完全一样。实际上我们在上一个实例学会使用 joinMulticastGroup 和 leaveMulticastGroup 的应用即可！

11.3.3.1 应用实例

本例目的：了解 QUdpSocket 组播使用。

例 11_udp_multicast，UDP 单播与广播应用（难度：一般）。项目路径为 Qt/2/11_udp_multicast。本例大体流程首先获取本地 IP 地址。创建一个 udpSocket 套接字，加入组播前必须绑定本机主机的端口。加入组播使用 joinMulticastGroup，退出组播使用 leaveMulticastGroup。其他收发消息的功能与上一节单播和广播一样。

项目文件 10_udp_unicast_broadcast.pro 文件第一行添加的代码部分如下。

11_udp_multicast.pro 编程后的代码

```
1   QT       += core gui network
2
3   greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5   CONFIG += c++11
6
7   # The following define makes your compiler emit warnings if you use
8   # any Qt feature that has been marked deprecated (the exact warnings
9   # depend on your compiler). Please consult the documentation of the
10  # deprecated API in order to know how to port your code away from it.
11  DEFINES += QT_DEPRECATED_WARNINGS
12
13  # You can also make your code fail to compile if it uses deprecated APIs.
14  # In order to do so, uncomment the following line.
15  # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16  #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
```

```
18  SOURCES += \
19      main.cpp \
20      mainwindow.cpp
21
22  HEADERS += \
23      mainwindow.h
24
25  # Default rules for deployment.
26  qnx: target.path = /tmp/$${TARGET}/bin
27  else: unix:!android: target.path = /opt/$${TARGET}/bin
28  !isEmpty(target.path): INSTALLS += target
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   10_udp_unicast_broadcast
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-14
*****************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QUdpSocket>
6   #include <QVBoxLayout>
7   #include <QHBoxLayout>
8   #include <QPushButton>
9   #include <QTextBrowser>
10  #include <QLabel>
11  #include <QComboBox>
12  #include <QSpinBox>
13  #include <QHostInfo>
14  #include <QLineEdit>
15  #include <QNetworkInterface>
16  #include <QDebug>
17
18  class MainWindow : public QMainWindow
19  {
20      Q_OBJECT
21
22  public:
```

```
23      MainWindow(QWidget *parent = nullptr);
24      ~MainWindow();
25
26  private:
27      /* Udp 通信套接字 */
28      QUdpSocket *udpSocket;
29
30      /* 按钮 */
31      QPushButton *pushButton[4];
32
33      /* 标签文本 */
34      QLabel *label[3];
35
36      /* 水平容器 */
37      QWidget *hWidget[3];
38
39      /* 水平布局 */
40      QHBoxLayout *hBoxLayout[3];
41
42      /* 垂直容器 */
43      QWidget *vWidget;
44
45      /* 垂直布局 */
46      QVBoxLayout *vBoxLayout;
47
48      /* 文本浏览框 */
49      QTextBrowser *textBrowser;
50
51      /* 用于显示本地 ip */
52      QComboBox *comboBox[2];
53
54      /* 用于选择端口 */
55      QSpinBox  *spinBox;
56
57      /* 文本输入框 */
58      QLineEdit *lineEdit;
59
60      /* 存储本地的 ip 列表地址 */
61      QList<QHostAddress> IPlist;
62
63      /* 获取本地的所有 ip */
64      void getLocalHostIP();
```

```
65
66  private slots:
67      /* 加入组播 */
68      void joinGroup();
69
70      /* 退出组播 */
71      void leaveGroup();
72
73      /* 清除文本框时的内容 */
74      void clearTextBrowser();
75
76      /* 接收到消息 */
77      void receiveMessages();
78
79      /* 组播消息 */
80      void sendMessages();
81
82      /* 连接状态改变槽函数 */
83      void socketStateChange(QAbstractSocket::SocketState);
84  };
85  #endif // MAINWINDOW_H
```

头文件里主要是声明界面用的元素，及一些槽函数。重点是声明 udpSocket。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*********************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   10_udp_unicast_broadcast
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-14
*********************************************************************/
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗体的位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* udp 套接字 */
```

```
10      udpSocket = new QUdpSocket(this);
11
12      /* 参数 1 是设置 IP_MULTICAST_TTL 套接字选项允许应用程序主要限制数据包在
Internet 中的生存时间，
13       * 并防止其无限期地循环，数据报跨一个路由会减一，默认值为 1，表示多播仅适用于
本地子网。*/
14      udpSocket->setSocketOption(QAbstractSocket::MulticastTtlOption,
1);
15
16      /* 加入组播按钮 */
17      pushButton[0] = new QPushButton();
18      /* 退出组播按钮 */
19      pushButton[1] = new QPushButton();
20      /* 清空聊天文本按钮 */
21      pushButton[2] = new QPushButton();
22      /* 组播消息按钮 */
23      pushButton[3] = new QPushButton();
24
25      /* 水平布局一 */
26      hBoxLayout[0] = new QHBoxLayout();
27      /* 水平布局二 */
28      hBoxLayout[1] = new QHBoxLayout();
29      /* 水平布局三 */
30      hBoxLayout[2] = new QHBoxLayout();
31      /* 水平布局四 */
32      hBoxLayout[3] = new QHBoxLayout();
33
34      /* 水平容器一 */
35      hWidget[0] =  new QWidget();
36      /* 水平容器二 */
37      hWidget[1] =  new QWidget();
38      /* 水平容器三 */
39      hWidget[2] =  new QWidget();
40
41
42      vWidget = new QWidget();
43      vBoxLayout = new QVBoxLayout();
44
45      /* 标签实例化 */
46      label[0] = new QLabel();
47      label[1] = new QLabel();
48      label[2] = new QLabel();
```

```
49
50      lineEdit = new QLineEdit();
51      comboBox[0] = new QComboBox();
52      comboBox[1] = new QComboBox();
53      spinBox = new QSpinBox();
54      textBrowser = new QTextBrowser();
55
56      label[0]->setText("本地 IP 地址：");
57      label[1]->setText("组播地址：");
58      label[2]->setText("组播端口：");
59
60      /* 设置标签根据文本文字大小自适应大小  */
61      label[0]->setSizePolicy(QSizePolicy::Fixed,
62                              QSizePolicy::Fixed);
63      label[1]->setSizePolicy(QSizePolicy::Fixed,
64                              QSizePolicy::Fixed);
65      label[2]->setSizePolicy(QSizePolicy::Fixed,
66                              QSizePolicy::Fixed);
67
68      /* 设置端口号的范围，注意不要与主机的已使用的端口号冲突 */
69      spinBox->setRange(10000, 99999);
70
71      pushButton[0]->setText("加入组播");
72      pushButton[1]->setText("退出组播");
73      pushButton[2]->setText("清空文本");
74      pushButton[3]->setText("组播消息");
75
76      /* 设置停止监听状态不可用 */
77      pushButton[1]->setEnabled(false);
78
79      /* 设置输入框默认的文本 */
80      lineEdit->setText("您好！");
81
82      /* 默认添加范围内的一个组播地址 */
83      comboBox[1]->addItem("239.255.255.1");
84
85      /* 设置可编辑，用户可自行修改此地址 */
86      comboBox[1]->setEditable(true);
87
88      /* 水平布局一添加内容 */
89      hBoxLayout[0]->addWidget(pushButton[0]);
90      hBoxLayout[0]->addWidget(pushButton[1]);
```

```
91      hBoxLayout[0]->addWidget(pushButton[2]);
92
93      /* 设置水平容器的布局为水平布局一 */
94      hWidget[0]->setLayout(hBoxLayout[0]);
95
96      hBoxLayout[1]->addWidget(label[0]);
97      hBoxLayout[1]->addWidget(comboBox[0]);
98      hBoxLayout[1]->addWidget(label[1]);
99      hBoxLayout[1]->addWidget(comboBox[1]);
100     hBoxLayout[1]->addWidget(label[2]);
101     hBoxLayout[1]->addWidget(spinBox);
102
103     /* 设置水平容器的布局为水平布局二 */
104     hWidget[1]->setLayout(hBoxLayout[1]);
105
106     /* 水平布局三添加内容 */
107     hBoxLayout[2]->addWidget(lineEdit);
108     hBoxLayout[2]->addWidget(pushButton[3]);
109
110     /* 设置水平容器三的布局为水平布局一 */
111     hWidget[2]->setLayout(hBoxLayout[2]);
112
113     /* 垂直布局添加内容 */
114     vBoxLayout->addWidget(textBrowser);
115     vBoxLayout->addWidget(hWidget[1]);
116     vBoxLayout->addWidget(hWidget[0]);
117     vBoxLayout->addWidget(hWidget[2]);
118
119     /* 设置垂直容器的布局为垂直布局 */
120     vWidget->setLayout(vBoxLayout);
121
122     /* 居中显示 */
123     setCentralWidget(vWidget);
124
125     /* 获取本地 ip */
126     getLocalHostIP();
127
128     /* 信号槽连接 */
129     connect(pushButton[0], SIGNAL(clicked()),
130             this, SLOT(joinGroup()));
131     connect(pushButton[1], SIGNAL(clicked()),
132             this, SLOT(leaveGroup()));
133     connect(pushButton[2], SIGNAL(clicked()),
```

```
134             this, SLOT(clearTextBrowser()));
135     connect(pushButton[3], SIGNAL(clicked()),
136             this, SLOT(sendMessages()));
137     connect(udpSocket, SIGNAL(readyRead()),
138             this, SLOT(receiveMessages()));
139     connect(udpSocket,
140             SIGNAL(stateChanged(QAbstractSocket::SocketState)),
141             this,
142             SLOT(socketStateChange(QAbstractSocket::SocketState)));
143 }
144
145 MainWindow::~MainWindow()
146 {
147 }
148
149 void MainWindow::joinGroup()
150 {
151     /* 获取端口 */
152     quint16 port = spinBox->value();
153     /* 获取组播地址 */
154     QHostAddress groupAddr = QHostAddress(comboBox[1]->currentText());
155
156     /* 绑定端口需要在 socket 的状态为 UnconnectedState */
157     if (udpSocket->state() != QAbstractSocket::UnconnectedState)
158         udpSocket->close();
159
160     /* 加入组播前必须先绑定端口 */
161     if (udpSocket->bind(QHostAddress::AnyIPv4,
162                         port, QUdpSocket::ShareAddress)) {
163
164         /* 加入组播组，返回结果给 ok 变量 */
165         bool ok = udpSocket->joinMulticastGroup(groupAddr);
166
167         textBrowser->append(ok ? "加入组播成功" : "加入组播失败");
168
169         textBrowser->append("组播地址 IP:"
170                             + comboBox[1]->currentText());
171
172         textBrowser->append("绑定端口: "
173                             + QString::number(port));
174
175         /* 设置界面中的元素的可用状态 */
176         pushButton[0]->setEnabled(false);
```

```
177         pushButton[1]->setEnabled(true);
178         comboBox[1]->setEnabled(false);
179         spinBox->setEnabled(false);
180     }
181 }
182
183 void MainWindow::leaveGroup()
184 {
185     /* 获取组播地址 */
186     QHostAddress groupAddr = QHostAddress(comboBox[1]->currentText());
187
188     /* 退出组播 */
189     udpSocket->leaveMulticastGroup(groupAddr);
190
191     /* 解绑，不再监听 */
192     udpSocket->abort();
193
194     /* 设置界面中的元素的可用状态 */
195     pushButton[0]->setEnabled(true);
196     pushButton[1]->setEnabled(false);
197     comboBox[1]->setEnabled(true);
198     spinBox->setEnabled(true);
199 }
200
201 /* 获取本地 IP */
202 void MainWindow::getLocalHostIP()
203 {
204     // /* 获取主机的名称 */
205     // QString hostName = QHostInfo::localHostName();
206
207     // /* 主机的信息 */
208     // QHostInfo hostInfo = QHostInfo::fromName(hostName);
209
210     // /* ip 列表,addresses 返回 ip 地址列表，注意主机应能从路由器获取到
211     // * IP，否则可能返回空的列表（ubuntu 用此方法只能获取到环回 IP） */
212     // IPlist = hostInfo.addresses();
213     // qDebug()<<IPlist<<endl;
214
215     // /* 遍历 IPlist */
216     // foreach (QHostAddress ip, IPlist) {
217     //     if (ip.protocol() == QAbstractSocket::IPv4Protocol)
218     //         comboBox->addItem(ip.toString());
219     //     }
```

```
220
221     /* 获取所有的网络接口,
222      * QNetworkInterface 类提供主机的 IP 地址和网络接口的列表 */
223     QList<QNetworkInterface> list
224           = QNetworkInterface::allInterfaces();
225
226     /* 遍历 list */
227     foreach (QNetworkInterface interface, list) {
228
229         /* QNetworkAddressEntry 类存储 IP 地址子网掩码和广播地址 */
230         QList<QNetworkAddressEntry> entryList
231               = interface.addressEntries();
232
233         /* 遍历 entryList */
234         foreach (QNetworkAddressEntry entry, entryList) {
235             /* 过滤 IPv6 地址，只留下 IPv4，并且不需要环回 IP */
236             if (entry.ip().protocol() ==
237                     QAbstractSocket::IPv4Protocol &&
238                     ! entry.ip().isLoopback()) {
239                 /* 添加本地 IP 地址到 comboBox[0] */
240                 comboBox[0]->addItem(entry.ip().toString());
241                 /* 添加到 IP 列表中 */
242                 IPlist<<entry.ip();
243             }
244         }
245     }
246 }
247
248 /* 清除文本浏览框里的内容 */
249 void MainWindow::clearTextBrowser()
250 {
251     /* 清除文本浏览器的内容 */
252     textBrowser->clear();
253 }
254
255 /* 客户端接收消息 */
256 void MainWindow::receiveMessages()
257 {
258     /* 局部变量，用于获取发送者的 IP 和端口 */
259     QHostAddress peerAddr;
260     quint16 peerPort;
261
```

```
262     /* 如果有数据已经准备好 */
263     while (udpSocket->hasPendingDatagrams()) {
264         /* udpSocket 发送的数据报是 QByteArray 类型的字节数组 */
265         QByteArray datagram;
266 
267         /* 重新定义数组的大小 */
268         datagram.resize(udpSocket->pendingDatagramSize());
269 
270         /* 读取数据，并获取发送方的 IP 地址和端口 */
271         udpSocket->readDatagram(datagram.data(),
272                                 datagram.size(),
273                                 &peerAddr,
274                                 &peerPort);
275         /* 转为字符串 */
276         QString str = datagram.data();
277 
278         /* 显示信息到文本浏览框窗口 */
279         textBrowser->append("接收来自"
280                             + peerAddr.toString()
281                             + ":"
282                             + QString::number(peerPort)
283                             + str);
284     }
285 }
286 
287 /* 客户端发送消息 */
288 void MainWindow::sendMessages()
289 {
290     /* 文本浏览框显示发送的信息 */
291     textBrowser->append("发送: " + lineEdit->text());
292 
293     /* 要发送的信息，转为 QByteArray 类型字节数组，数据一般少于 512 个字节 */
294     QByteArray data = lineEdit->text().toUtf8();
295 
296     /* 要发送的目标 Ip 地址 */
297     QHostAddress groupAddr = QHostAddress(comboBox[1]->currentText());
298 
299     /* 要发送的目标端口号 */
300     quint16 groupPort = spinBox->value();
301 
302     /* 发送消息 */
303     udpSocket->writeDatagram(data, groupAddr, groupPort);
```

```
304 }
305
306 /* socket 状态改变 */
307 void MainWindow::socketStateChange(QAbstractSocket::SocketState
state)
308 {
309     switch (state) {
310     case QAbstractSocket::UnconnectedState:
311         textBrowser->append("scoket 状态: UnconnectedState");
312         break;
313     case QAbstractSocket::ConnectedState:
314         textBrowser->append("scoket 状态: ConnectedState");
315         break;
316     case QAbstractSocket::ConnectingState:
317         textBrowser->append("scoket 状态: ConnectingState");
318         break;
319     case QAbstractSocket::HostLookupState:
320         textBrowser->append("scoket 状态: HostLookupState");
321         break;
322     case QAbstractSocket::ClosingState:
323         textBrowser->append("scoket 状态: ClosingState");
324         break;
325     case QAbstractSocket::ListeningState:
326         textBrowser->append("scoket 状态: ListeningState");
327         break;
328     case QAbstractSocket::BoundState:
329         textBrowser->append("scoket 状态: BoundState");
330         break;
331     default:
332         break;
333     }
334 }
```

第 161~162 行，绑定端口。使用 bind 方法，即可绑定一个端口。注意我们绑定的端口不能和主机已经使用的端口冲突！

第 165 行，使用 joinMulticastGroup 加入组播，QHostAddress::AnyIPv4，是加入 Ipv4 组播的一个接口，所有操作系统都不支持不带接口选择的加入 IPv6 组播组。加入的结果返回给变量 ok。组播地址可由用户点击 comboBox[1]控件输入（默认笔者已经输入一个地址为 239.255.255.1），注意组播地址的范围必须是 239.0.0.0～239.255.255.255 中的一个数。

第 189 行，使用 leaveMulticastGroup 退出组播。

第 192 行，解绑端口。使用 abort 方法即可解绑。

第 256~285 行，接收消息，注意接收消息是 QByteArray 字节数组。读数组使用的是 readDatagram 方法，在 readDatagram 方法里可以获取对方的套接字 IP 地址与端口号。

第 288~304 行，发送消息，组播与广播消息或单播消息不同的是将目标 IP 地址换成了**组播地址** 239.255.255.1。

11.3.3.2 程序运行效果

运行程序后，点击加入组播，然后点击组播消息，本实例可以做即是发送者，也是接收者。如果在同一台主机同一个系统里运行两个本例程序。不能绑定同一个端口！否则会冲突！当您想测试在同一局域网内不同主机上运行此程序，那么绑定的端口号可以相同。

因为是组播消息，所以自己也会收到消息，如果在局域网内其他主机运行此程序，当点击加入组播后，就可以收发消息了。

11.4 网络下载实例

Qt 网络模块还提供了直接访问如 HTTP，FTP 等网络协议的类，这些类是 QNetworkAccessManager、QNetworkRequest 和 QNetworkReply。

通常需要这三个类协作才能完成一个网络操作。可以用于从网络获取时间，天气和图片等等数据。比如本例需要下载一张图片，大概流程如下。

由 QNetworkRequest 类设置一个 URL 地址发起网络协议请求，QNetworkRequest 类保存要用 QNetworkAccessManager 发送的请求。QNetworkRequest 是网络访问 API 的一部分，是一个持有通过网络发送请求所需信息的类。它包含一个 URL 和一些可用于修改请求的辅助信息。

QNetworkAccessManager 类允许应用程序发送网络请求并接收响应。在 QNetworkRequest 发起网络请求后，QNetworkAccessManager 负责发送网络请求，创建网络响应。

QNetworkReply 类就用于 QNetworkAccessManager 创建的网络响应。最终由 QNetworkReply 处理网络响应。它提供了 finished()、readyRead()和 downloadProgress()等信号，可以监测网络响应的执行情况。并且 QNetworkReply 继承于 QIODevice，所以 QNetworkReply 支持流读写，可以直接用 read()和 write 等功能。

11.4.1 应用实例

本例目的：了解 QNetworkAccessManager、QNetworkRequest 和 QNetworkReply 类的使用。

例 12_imagedownload，下载小图片（难度：一般）。项目路径为 Qt/2/12_imagedownload。本例大体流程，设置一个下载图片的 URL，通过 networkReply 处理响应后，从流中读取图片的数据，然后保存到本地。

项目文件 12_imagedownload.pro 文件第一行添加的代码部分如下。

12_imagedownload.pro 编程后的代码

```
1  QT       += core gui network
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
13 # You can also make your code fail to compile if it uses deprecated APIs.
14 # In order to do so, uncomment the following line.
15 # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18 SOURCES += \
19     main.cpp \
20     mainwindow.cpp
21
22 HEADERS += \
23     mainwindow.h
24
25 # Default rules for deployment.
26 qnx: target.path = /tmp/$${TARGET}/bin
```

```
27 else: unix:!android: target.path = /opt/$${TARGET}/bin
28 !isEmpty(target.path): INSTALLS += target
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   12_imagedownload
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-16
*****************************************************************/
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QNetworkAccessManager>
6  #include <QNetworkReply>
7  #include <QFile>
8  #include <QLabel>
9  #include <QPushButton>
10 #include <QProgressBar>
11 #include <QHBoxLayout>
12 #include <QVBoxLayout>
13 #include <QLineEdit>
14
15 class MainWindow : public QMainWindow
16 {
17     Q_OBJECT
18
19 public:
20     MainWindow(QWidget *parent = nullptr);
21     ~MainWindow();
22 private:
23     /* 网络管理 */
24     QNetworkAccessManager *networkAccessManager;
25
26     /* 标签 */
27     QLabel *label[3];
28
29     /* 按钮 */
30     QPushButton *pushButton;
```

```
31
32      /* 下载进度条 */
33      QProgressBar *progressBar;
34
35      /* 水平布局 */
36      QHBoxLayout *hBoxLayout[2];
37
38      /* 垂直布局 */
39      QVBoxLayout *vBoxLayout;
40
41      /* 水平容器 */
42      QWidget *hWidget[2];
43
44      /* 垂直容器 */
45      QWidget *vWidget;
46
47      /* 链接输入框 */
48      QLineEdit *lineEdit;
49
50  private slots:
51      /* 读取数据 */
52      void readyReadData();
53
54      /* 响应完成处理 */
55      void replyFinished();
56
57      /* 下载进度管理 */
58      void imageDownloadProgress(qint64, qint64);
59
60      /* 点击开始下载 */
61      void startDownload();
62
63      /* 响应错误处理函数 */
64      void networkReplyError(QNetworkReply::NetworkError);
65  };
66  #endif // MAINWINDOW_H
```

头文件里主要是声明界面用的元素，及一些槽函数。重点是声明 networkAccessManager。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   12_imagedownload
```

```
    * @brief        mainwindow.cpp
    * @author       Deng Zhimao
    * @email        1252699831@qq.com
    * @net          www.openedv.com
    * @date         2021-04-16
    ****************************************************************/
1   #include "mainwindow.h"
2   #include <QMessageBox>
3   #include <QCoreApplication>
4
5   MainWindow::MainWindow(QWidget *parent)
6       : QMainWindow(parent)
7   {
8       /* 设置主窗体的位置与大小 */
9       this->setGeometry(0, 0, 800, 480);
10
11      /* 标签 0，显示下载的图像 */
12      label[0] = new QLabel();
13      /* 标签 1，显示 URL 标签 */
14      label[1] = new QLabel();
15      /* 下载进度标签 */
16      label[2] = new QLabel();
17
18      /* 下载图片链接输入框 */
19      lineEdit = new QLineEdit();
20
21      /* 下载按钮 */
22      pushButton = new QPushButton();
23
24      /* 下载进度条 */
25      progressBar = new QProgressBar();
26
27      /* 水平布局 */
28      hBoxLayout[0] = new QHBoxLayout();
29      hBoxLayout[1] = new QHBoxLayout();
30
31      /* 垂直布局 */
32      vBoxLayout = new QVBoxLayout();
33
34      /* 水平容器 */
35      hWidget[0] = new QWidget();
36      hWidget[1] = new QWidget();
37
```

```
38      /* 垂直容器 */
39      vWidget = new QWidget();
40
41      label[1]->setText("URL 链接: ");
42      label[2]->setText("文件下载进度: ");
43
44      pushButton->setText("下载");
45
46      /* 设置下载链接地址 */
47      lineEdit->setText("https://ss0.bdstatic.com/70cFuH"
48                        "Sh_Q1YnxGkpoWK1HF6hhy/it/u=42710"
49                        "87328,1384669424&fm=11&gp=0.jpg");
50      /* 设置标签的最小显示大小 */
51      label[0]->setMinimumSize(this->width(),
52                               this->height() * 0.75);
53
54      /* 根据文本文字大小自适应大小 */
55      label[1]->setSizePolicy(QSizePolicy::Fixed,
56                              QSizePolicy::Fixed);
57      label[2]->setSizePolicy(QSizePolicy::Fixed,
58                              QSizePolicy::Fixed);
59      pushButton->setSizePolicy(QSizePolicy::Fixed,
60                                QSizePolicy::Fixed);
61
62      /* 水平布局 0 添加元素 */
63      hBoxLayout[0]->addWidget(label[1]);
64      hBoxLayout[0]->addWidget(lineEdit);
65      hBoxLayout[0]->addWidget(pushButton);
66
67      /* 设置水平布局 0 为水平容器的布局 0 */
68      hWidget[0]->setLayout(hBoxLayout[0]);
69
70      /* 水平布局 1 添加元素 */
71      hBoxLayout[1]->addWidget(label[2]);
72      hBoxLayout[1]->addWidget(progressBar);
73
74      /* 设置水平布局 1 为水平容器的布局 1 */
75      hWidget[1]->setLayout(hBoxLayout[1]);
76
77      /* 垂直布局添加元素 */
78      vBoxLayout->addWidget(label[0]);
79      vBoxLayout->addWidget(hWidget[0]);
```

```
80      vBoxLayout->addWidget(hWidget[1]);
81
82      /* 设置垂直布局为垂直容器的布局 */
83      vWidget->setLayout(vBoxLayout);
84
85      /* 设置居中 */
86      setCentralWidget(vWidget);
87
88      /* 网络管理 */
89      networkAccessManager = new QNetworkAccessManager(this);
90
91      /* 信号槽连接 */
92      connect(pushButton, SIGNAL(clicked()),
93              this, SLOT(startDownload()));
94
95  }
96
97  MainWindow::~MainWindow()
98  {
99  }
100
101 void MainWindow::startDownload()
102 {
103     /* 获取 URL 链接 */
104     QUrl newUrl(QUrl(lineEdit->text()));
105
106     /* 如果下载链接无效，则直接返回 */
107     if (!newUrl.isValid()) {
108         QMessageBox::information(this, "error", "invalid url");
109         return;
110     }
111
112     /* 网络请求 */
113     QNetworkRequest  networkRequest;
114
115     /* 设置下载的地址 */
116     networkRequest.setUrl(newUrl);
117
118     /* 网络响应 */
119     QNetworkReply *newReply =
120             networkAccessManager->get(networkRequest);
121
122     /* 信号槽连接 */
```

```
123     connect(newReply, SIGNAL(finished()),
124             this, SLOT(replyFinished()));
125     connect(newReply, SIGNAL(readyRead()),
126             this, SLOT(readyReadData()));
127     connect(newReply, SIGNAL(downloadProgress(qint64, qint64)),
128             this, SLOT(imageDownloadProgress(qint64, qint64)));
129     connect(newReply,
130             SIGNAL(error(QNetworkReply::NetworkError)),
131             this,
132             SLOT(networkReplyError(QNetworkReply::NetworkError )));
133 }
134
135 void MainWindow::readyReadData()
136 {
137     /* 设置按钮不可用，防止未完成，再次点击 */
138     pushButton->setEnabled(false);
139
140     /* 获取信号发送者 */
141     QNetworkReply *reply = (QNetworkReply *)sender();
142
143     QFile imageFile;
144     /* 保存到当前路径，名称为"下载的.jpg" */
145     imageFile.setFileName(QCoreApplication::applicationDirPath()
146                           + "/下载的.jpg");
147
148     /* 如果此图片已经存在，则删除 */
149     if (imageFile.exists())
150         imageFile.remove();
151
152     /* 读取数据 */
153     QByteArray data =  reply->readAll();
154     /* 如果数据为空，返回 */
155     if (data.isEmpty()) {
156         qDebug()<<"data is null, please try it again!"<<endl;
157         return;
158     }
159
160     /* 判断是不是 JPG 格式的图片，如果不是则返回 */
161     if (! (data[0] == (char)0xff
162           && data[1] == (char)0xd8
163           && data[data.size() - 2] == (char)0xff
164           && data[data.size() - 1] == (char)0xd9)) {
165         qDebug()<<"not JPG data, please try it again!"<<endl;
```

```
166         return;
167     }
168 
169     /* 转为 QPixmap */
170     QPixmap pixmap;
171     pixmap.loadFromData(data);
172     pixmap.save(imageFile.fileName());
173 }
174 
175 void MainWindow::replyFinished()
176 {
177     /* 获取信号发送者 */
178     QNetworkReply *reply = (QNetworkReply *)sender();
179 
180     /* 防止内存泄漏 */
181     reply->deleteLater();
182 
183     /* 判断当前执行程序下的图像是否下载完成 */
184     QFile imageFile(QCoreApplication::applicationDirPath()
185                     + "/下载的.jpg");
186     if (imageFile.exists()) {
187         /* 显示下载的图像 */
188         label[0]->setPixmap(QPixmap(imageFile.fileName()));
189         qDebug() <<"已经成功下载，文件路径为:"
190                  <<imageFile.fileName()<<endl;
191     } else
192         /* 清空显示 */
193         label[0]->clear();
194 
195     /* 设置按钮可用 */
196     pushButton->setEnabled(true);
197 }
198 
199 void MainWindow::imageDownloadProgress(qint64 bytes,
200                                        qint64 totalBytes)
201 {
202     /* 设置进度条的最大值 */
203     progressBar->setMaximum(totalBytes);
204     /* 设置当前值 */
205     progressBar->setValue(bytes);
206 }
207 
```

```
208 /* 网络响应处理函数 */
209 void MainWindow::networkReplyError(QNetworkReply::NetworkError
210                                    error)
211 {
212     switch (error) {
213     case QNetworkReply::ConnectionRefusedError:
214         qDebug()<<"远程服务器拒绝连接"<<endl;
215         break;
216     case QNetworkReply::HostNotFoundError:
217         qDebug()<<"找不到远程主机名"<<endl;
218         break;
219     case QNetworkReply::TimeoutError:
220         qDebug()<<"与远程服务器连接超时"<<endl;
221         break;
222     default:
223         break;
224     }
225 }
```

第 89 行，全局变量 networkAccessManager 实例化。

第 101 行~133 行，首先从单行输入框里获取 URL 链接为 newUrl，判断链接的有效性。然后创建局部变量 networkRequest，设置 networkRequest 请求 URL 为 newUrl。QNetworkReply *newReply = networkAccessManager->get(networkRequest);为最重要的代码，所有响应本次的操作都交给了 newReply。通过信号槽处理对应的操作。

第 135~173 行，这部分代码就是从 newReply 流里读取网络下载的数据。

第 160~167 行，这里笔者做了一些处理，从网络下载的数据可能遇到数据丢失或者下载错误的情况。本例是从百度里下载一张 JPG 格式的图片，因为 JPG 图片的判断依据是第一个字节和第二个字节的数据是 0xff 和 0xd8，倒数第一个和倒数第二个字节数据分别是 0xd9 和 0xff。如果都对，那么判断此数据为 JPG 图片数据。然后进行保存，否则则不保存图片。处理数据往往是需要的，我们经常要对下载的数据进行处理。

第 174~197 行，网络响应完成。记得要删除 reply，防止内存泄漏。如果下载成功图片，则显示图片到 label[0]上。

第 209~225 行，网络响应错误处理函数。

11.4.2 程序运行效果

点击下载按钮后，可以看到本次下载的图片已经保存到当前编译输出的路径下，名字叫“下载的.jpg”,并显示到界面上。由于文件下载的速度非常快，所以下载的进度条一下子就变成了 100%。若想看见下载进度条下载进度缓慢一些，可以修改本例去下载其他文件，注意不要保存为 jpg 图片了。注意：此程序里的下载链接可能失效，请替换自己的图片链接。


12_imagedownload
URL链接:
.com/70cFuHSh_Q1YnxGkpoWK1HF6hhy/it/u=4271087328,1384669424&fm=11&gp=0.jpg
下载
文件下载进度:
100%
11:28:51: Starting /home/alientek/Qt/2/build-12_imagedownload-Desktop_Qt_5_12_9_GCC_64bit-Debug/12_imagedownload
已经成功下载，文件路径为："/home/alientek/Qt/2/build-12_imagedownload-Desktop_Qt_5_12_9_GCC_64bit-Debug/下载的.jpg"

第十二章 多媒体

多媒体（Multimedia）是多种媒体的综合，一般包括文本，声音和图像等多种媒体形式。在计算机系统中，多媒体指组合两种或两种以上媒体的一种人机交互式信息交流和传播媒体。使用的媒体包括文字、图片、照片、声音、动画和影片，以及程式所提供的互动功能。

Qt 的多媒体模块提供了音频、视频、录音、摄像头拍照和录像等功能。本章将介绍 Qt 多媒体的功能和使用。

12.1 Qt 多媒体简介

Qt 从 4.4 版本开始提供的一套多媒体框架，提供多媒体回放的功能。在 Qt 4.6 中实现多媒体播放图形界面主要依赖 phonon 框架。phonon 最初是 一个 源于 KDE 的项目，为使用音频和视频的应用程序开发提供的一个框架。应用程序不用去管多媒体播放是通过什么实现的（如 gstreamer、xine），只需调用相应的接口就行，但这中间需要一个中转，被称为 backend。Qt 也是通过 phonon 来实现跨平台的多媒体播放。

从 Qt5 开始，Qt 就弃用了 phonon，直接使用 Qt Multimedia 模块。我们可以 Qt Multimedia 模块来提供的类实现跨平台的多媒体播放了。使用 Qt Multimedia 就不需要中转了，但是底层还是需要多媒体插件实现的。Qt 只是提供多媒体接口，播放多媒体实际上是通过多媒体插件实现的，我们不需要管这些插件是什么，Qt 在不同平台使用的多媒体插件不同。本章将会介绍如何在 Windows 和 Linux 安装多媒体插件，Mac 系统不考虑，笔者条件有限！

Qt 多媒体模块提供了很多类，主要有 QMediaPlayer，QSound、QSoundEffect、QAudioOutput、QAudioInput、QAudioRecorder、QVideoWidget 等等。类太多了不一一作解释，可以直接复制名字到 Qt 的帮助文档里查看该解释。可以从名称大概了解它们是什么意思，具体类的使用直接看本章的例子。

想要在 Qt 里使用使用 Qt 多媒体模块，需要在 pro 项目文件里添加如下语句。

```
QT          += multimedia
```

注意：Qt 中的音乐播放器与视频播放器需要在 Ubuntu 里安装媒体解码器才能实现播放。

- Ubuntu16 / Ubuntu18，需要安装以下插件。播放音乐需要安装 Gst 解码插件。需要在终端输入如下指令，注意不要复制错误了，下面指令已经在 Ubuntu16/Ubuntu18 测试成功，如果读者 Ubuntu 没有配置网络与源服务器，这些导致安装不成功与本教程无关，确实需要读者好好打下 Ubuntu 操作的基础了！

```
sudo apt-get install gstreamer1.0-plugins-base gstreamer1.0-plugins-bad gstreamer1.0-plugins-good gstreamer1.0-plugins-ugly gstreamer1.0-pulseaudio gstreamer1.0-libav
```

- Windows 需要安装如 LAVFilters 解码器，只需要百度 LAVFilters，找到 LAVFilters 官网下载此软件即可，当然本教程的资料会提供一份 LAVFilters 的安装包。点击页脚下方的程序下载链接跳转到下载本教程所有资料下载地址处，在顶层目录下。

12.2 音效文件播放

播放音效文件，比如简短的提示音（按键音等），可以使用 Qt 的 QSoundEffect 和 QSound 类来播放。

Qt 的 QSoundEffect 和 QSound 类主要区别是 QSound(异步方式播放)只能播放本地的 WAV 音效文件（WAV 音效文件是 PC 机上最为流行的声音文件格式，但其文件尺寸较大，多用于存储简短的声音片段，具有低延时性，不失真的特点），QSoundEffect 不仅可以播放网络文件，也可以播放本地音效文件，播放网络的文件一般使用到 QUrl 链接。

12.2.1 应用实例

本例目的：了解 QSound 类的使用。

例 13_button_sound，按钮音效测试（难度：一般）。项目路径为 Qt/2/13_button_sound。本例大体流程，通过点击一个按钮，然后使用 QSound 来播放音效文件，模仿按键按下的声音。

项目文件 13_button_sound.pro 文件第一行添加的代码部分如下。

13_button_sound.pro 编程后的代码

```
1  QT       += core gui multimedia
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
13 # You can also make your code fail to compile if it uses deprecated APIs.
14 # In order to do so, uncomment the following line.
15 # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18 SOURCES += \
19     main.cpp \
20     mainwindow.cpp
21
22 HEADERS += \
23     mainwindow.h
24
25 # Default rules for deployment.
26 qnx: target.path = /tmp/$${TARGET}/bin
27 else: unix:!android: target.path = /opt/$${TARGET}/bin
28 !isEmpty(target.path): INSTALLS += target
29
30 RESOURCES += \
31     src.qrc
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*************************************************************************
```

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   13_button_sound
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-20
*****************************************************************/
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSound>
#include <QPushButton>

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    /* 按钮 */
    QPushButton *pushButton;

private slots:
    /* 按钮点击槽函数 */
    void pushButtonClicked();

};
#endif // MAINWINDOW_H

```

头文件里主要是声明界面使用的一个按钮，及按钮槽函数。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   13_button_sound
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
```

```
    * @net            www.openedv.com
    * @date           2021-04-20
    ************************************************************************/
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       /* 设置主窗体的位置与大小 */
7       this->setGeometry(0, 0, 800, 480);
8
9       /* 实例化按钮 */
10      pushButton = new QPushButton(this);
11
12      /* 设置按钮文本 */
13      pushButton->setText("按钮音效测试");
14
15      /* 设置按钮的位置与大小 */
16      pushButton->setGeometry(340, 220, 120, 40);
17
18      /* 信号槽连接 */
19      connect(pushButton, SIGNAL(clicked()),
20              this, SLOT(pushButtonClicked()));
21  }
22
23  MainWindow::~MainWindow()
24  {
25  }
26
27  void MainWindow::pushButtonClicked()
28  {
29      /* 异步的方式播放 */
30      QSound::play(":/audio/bell.wav");
31  }
```

第 30 行，直接使用 QSound 的静态函数 play()播放，这种播放方式是异步的，可以多次点击按钮连续听到点击的声音。

12.2.1 程序运行效果

单击按钮后，可以听到播放 1 秒左右的叮咚声，用此方法来模拟单击按钮声音效果。

12.3 音乐播放器

QMediaPlayer 类是一个高级媒体播放类。它可以用来播放歌曲、电影和网络广播等内容。一般用于播放 mp3 和 mp4 等等媒体文件。QMediaPlayer 类常常与 QMediaPlaylist 类一起使用。可以很轻松的设计一个自己喜欢的音乐播放器与视频播放器。

QMediaPlayer 提供了很多信号，我们可以使用这些信号来完成音乐播放器的一系列操作，比如媒体状态改变的信号 stateChanged(QMediaPlayer::State state)，判断这个 state 的状态就可以知道什么时候媒体暂停、播放、停止了。Qt 在媒体播放类已经提供了很多功能函数给我们使用，像直接使用 play()函数就可以实现音乐文件的播放，前提我们需要知道媒体文件的路径。pause()函数可以直接暂停媒体播放等等，这些都可以在 Qt 帮助文档里查看 QMediaPlayer 类的使用方法就可以知道。不再一一列出。

12.3.1 应用实例

本例设计一个比较好看的音乐播放器，界面是笔者模仿网上的一个音乐播放器的界面，并非笔者原创界面，只是笔者用 Qt 实现了网上的一个好看的音乐播放器界面。其中本例有些功能并没有完善，比如播放模式、没有加音量控制等。这些可以由读者自由完善，比较简单。

本例目的：音乐播放器的设计与使用。

例 14_musicplayer，音乐播放器（难度：中等）。项目路径为 Qt/2/14_musicplayer。注意本例有用到 qss 样式文件，关于如何添加资源文件与 qss 文件请参考 7.1.3 小节。音乐播放器的功能这些都为大家所熟知，不用笔者介绍了。

项目文件 14_musicplayer.pro 文件第一行添加的代码部分如下。

14_musicplayer.pro 编程后的代码

```
1  QT       += core gui multimedia
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
13 # You can also make your code fail to compile if it uses deprecated APIs.
14 # In order to do so, uncomment the following line.
15 # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18 SOURCES += \
19     main.cpp \
20     mainwindow.cpp
21
22 HEADERS += \
23     mainwindow.h
24
25 # Default rules for deployment.
26 qnx: target.path = /tmp/${TARGET}/bin
27 else: unix:!android: target.path = /opt/${TARGET}/bin
28 !isEmpty(target.path): INSTALLS += target
29
30 RESOURCES += \
31     res.qrc
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   14_musicplayer
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-20
*****************************************************************/
```

```
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QMediaPlayer>
6   #include <QMediaPlaylist>
7   #include <QPushButton>
8   #include <QSlider>
9   #include <QVBoxLayout>
10  #include <QHBoxLayout>
11  #include <QListWidget>
12  #include <QLabel>
13  #include <QSpacerItem>
14  #include <QDebug>
15
16  /* 媒体信息结构体 */
17  struct MediaObjectInfo {
18      /* 用于保存歌曲文件名 */
19      QString fileName;
20      /* 用于保存歌曲文件路径 */
21      QString filePath;
22  };
23
24  class MainWindow : public QMainWindow
25  {
26      Q_OBJECT
27
28  public:
29      MainWindow(QWidget *parent = nullptr);
30      ~MainWindow();
31
32  private:
33      /* 媒体播放器，用于播放音乐 */
34      QMediaPlayer *musicPlayer;
35
36      /* 媒体列表 */
37      QMediaPlaylist *mediaPlaylist;
38
39      /* 音乐列表 */
40      QListWidget *listWidget;
41
42      /* 播放进度条 */
43      QSlider *durationSlider;
```

```
44
45      /* 音乐播放器按钮 */
46      QPushButton *pushButton[7];
47
48      /* 垂直布局 */
49      QVBoxLayout *vBoxLayout[3];
50
51      /* 水平布局 */
52      QHBoxLayout *hBoxLayout[4];
53
54      /* 垂直容器 */
55      QWidget *vWidget[3];
56
57      /* 水平容器 */
58      QWidget *hWidget[4];
59
60      /* 标签文本 */
61      QLabel *label[4];
62
63      /* 用于遮罩 */
64      QWidget *listMask;
65
66      /* 音乐布局函数 */
67      void musicLayout();
68
69      /* 主窗体大小重设大小函数重写 */
70      void resizeEvent(QResizeEvent *event);
71
72      /* 媒体信息存储 */
73      QVector<MediaObjectInfo> mediaObjectInfo;
74
75      /* 扫描歌曲 */
76      void scanSongs();
77
78      /* 媒体播放器类初始化 */
79      void mediaPlayerInit();
80
81  private slots:
82      /* 播放按钮点击 */
83      void btn_play_clicked();
84
85      /* 下一曲按钮点击*/
```

```
86      void btn_next_clicked();
87
88      /* 上一曲按钮点击 */
89      void btn_previous_clicked();
90
91      /* 媒体状态改变 */
92      void mediaPlayerStateChanged(QMediaPlayer::State);
93
94      /* 列表单击 */
95      void listWidgetCliked(QListWidgetItem*);
96
97      /* 媒体列表项改变 */
98      void mediaPlaylistCurrentIndexChanged(int);
99
100     /* 媒体总长度改变 */
101     void musicPlayerDurationChanged(qint64);
102
103     /* 媒体播放位置改变 */
104     void mediaPlayerPositionChanged(qint64);
105
106     /* 播放进度条松开 */
107     void durationSliderReleased();
108 };
109 #endif // MAINWINDOW_H
```

头文件里主要是声明界面所使用的元素及一些槽函数。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   14_musicplayer
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-20
*****************************************************************/
1   #include "mainwindow.h"
2   #include <QCoreApplication>
3   #include <QFileInfoList>
4   #include <QDir>
5
6   MainWindow::MainWindow(QWidget *parent)
7       : QMainWindow(parent)
```

```
8   {
9       /* 布局初始化 */
10      musicLayout();
11
12      /* 媒体播放器初始化 */
13      mediaPlayerInit();
14
15      /* 扫描歌曲 */
16      scanSongs();
17
18      /* 按钮信号槽连接 */
19      connect(pushButton[0], SIGNAL(clicked()),
20              this, SLOT(btn_previous_clicked()));
21      connect(pushButton[1], SIGNAL(clicked()),
22              this, SLOT(btn_play_clicked()));
23      connect(pushButton[2], SIGNAL(clicked()),
24              this, SLOT(btn_next_clicked()));
25
26      /* 媒体信号槽连接 */
27      connect(musicPlayer,
28              SIGNAL(stateChanged(QMediaPlayer::State)),
29              this,
30              SLOT(mediaPlayerStateChanged(QMediaPlayer::State)));
31      connect(mediaPlaylist,
32              SIGNAL(currentIndexChanged(int)),
33              this,
34              SLOT(mediaPlaylistCurrentIndexChanged(int)));
35      connect(musicPlayer, SIGNAL(durationChanged(qint64)),
36              this,
37              SLOT(musicPlayerDurationChanged(qint64)));
38      connect(musicPlayer,
39              SIGNAL(positionChanged(qint64)),
40              this,
41              SLOT(mediaPlayerPositionChanged(qint64)));
42
43      /* 列表信号槽连接 */
44      connect(listWidget, SIGNAL(itemClicked(QListWidgetItem*)),
45              this, SLOT(listWidgetCliked(QListWidgetItem*)));
46
47      /* slider 信号槽连接 */
48      connect(durationSlider, SIGNAL(sliderReleased()),
49              this, SLOT(durationSliderReleased()));
50
```

```
51      /* 失去焦点 */
52      this->setFocus();
53  }
54
55  void MainWindow::musicLayout()
56  {
57      /* 设置位置与大小,这里固定为 800, 480 */
58      this->setGeometry(0, 0, 800, 480);
59      QPalette pal;
60
61      /* 按钮 */
62      for (int i = 0; i < 7; i++)
63          pushButton[i] = new QPushButton();
64
65      /* 标签 */
66      for (int i = 0; i < 4; i++)
67          label[i] = new QLabel();
68
69      for (int i = 0; i < 3; i++) {
70          /* 垂直容器 */
71          vWidget[i] = new QWidget();
72          vWidget[i]->setAutoFillBackground(true);
73          /* 垂直布局 */
74          vBoxLayout[i] = new QVBoxLayout();
75      }
76
77      for (int i = 0; i < 4; i++) {
78          /* 水平容器 */
79          hWidget[i] = new QWidget();
80          hWidget[i]->setAutoFillBackground(true);
81          /* 水平布局 */
82          hBoxLayout[i] = new QHBoxLayout();
83      }
84
85      /* 播放进度条 */
86      durationSlider = new QSlider(Qt::Horizontal);
87      durationSlider->setMinimumSize(300, 15);
88      durationSlider->setMaximumHeight(15);
89      durationSlider->setObjectName("durationSlider");
90
91      /* 音乐列表 */
92      listWidget = new QListWidget();
93      listWidget->setObjectName("listWidget");
```

```
94      listWidget->resize(310, 265);
95      listWidget->setVerticalScrollBarPolicy(
96                  Qt::ScrollBarAlwaysOff);
97      listWidget->setHorizontalScrollBarPolicy(
98                  Qt::ScrollBarAlwaysOff);
99
100     /* 列表遮罩 */
101     listMask = new QWidget(listWidget);
102     listMask->setMinimumSize(310, 50);
103     listMask->setMinimumHeight(50);
104     listMask->setObjectName("listMask");
105     listMask->setGeometry(0,
106                         listWidget->height() - 50,
107                         310,
108                         50);
109
110     /* 设置对象名称 */
111     pushButton[0]->setObjectName("btn_previous");
112     pushButton[1]->setObjectName("btn_play");
113     pushButton[2]->setObjectName("btn_next");
114     pushButton[3]->setObjectName("btn_favorite");
115     pushButton[4]->setObjectName("btn_mode");
116     pushButton[5]->setObjectName("btn_menu");
117     pushButton[6]->setObjectName("btn_volume");
118
119     /* 设置按钮属性 */
120     pushButton[1]->setCheckable(true);
121     pushButton[3]->setCheckable(true);
122
123     /* H0 布局 */
124     vWidget[0]->setMinimumSize(310, 480);
125     vWidget[0]->setMaximumWidth(310);
126     vWidget[1]->setMinimumSize(320, 480);
127     QSpacerItem *hSpacer0 = new
128           QSpacerItem(70, 480,
129                       QSizePolicy::Minimum,
130                       QSizePolicy::Maximum);
131
132     QSpacerItem *hSpacer1 = new
133           QSpacerItem(65, 480,
134                       QSizePolicy::Minimum,
135                       QSizePolicy::Maximum);
136
137     QSpacerItem *hSpacer2 = new
```

```
138             QSpacerItem(60, 480,
139                         QSizePolicy::Minimum,
140                         QSizePolicy::Maximum);
141
142     hBoxLayout[0]->addSpacerItem(hSpacer0);
143     hBoxLayout[0]->addWidget(vWidget[0]);
144     hBoxLayout[0]->addSpacerItem(hSpacer1);
145     hBoxLayout[0]->addWidget(vWidget[1]);
146     hBoxLayout[0]->addSpacerItem(hSpacer2);
147     hBoxLayout[0]->setContentsMargins(0, 0, 0, 0);
148
149     hWidget[0]->setLayout(hBoxLayout[0]);
150     setCentralWidget(hWidget[0]);
151
152     /* V0 布局 */
153     listWidget->setMinimumSize(310, 265);
154     hWidget[1]->setMinimumSize(310, 80);
155     hWidget[1]->setMaximumHeight(80);
156     label[0]->setMinimumSize(310, 95);
157     label[0]->setMaximumHeight(95);
158     QSpacerItem *vSpacer0 = new
159             QSpacerItem(310, 10,
160                         QSizePolicy::Minimum,
161                         QSizePolicy::Maximum);
162     QSpacerItem *vSpacer1 = new
163             QSpacerItem(310, 30,
164                         QSizePolicy::Minimum,
165                         QSizePolicy::Minimum);
166     vBoxLayout[0]->addWidget(label[0]);
167     vBoxLayout[0]->addWidget(listWidget);
168     vBoxLayout[0]->addSpacerItem(vSpacer0);
169     vBoxLayout[0]->addWidget(hWidget[1]);
170     vBoxLayout[0]->addSpacerItem(vSpacer1);
171     vBoxLayout[0]->setContentsMargins(0, 0, 0, 0);
172
173     vWidget[0]->setLayout(vBoxLayout[0]);
174
175     /* H1 布局 */
176     for (int i = 0; i < 3; i++) {
177         pushButton[i]->setMinimumSize(80, 80);
178     }
179     QSpacerItem *hSpacer3 = new
180             QSpacerItem(40, 80,
181                         QSizePolicy::Expanding,
```

```
182                     QSizePolicy::Expanding);
183     QSpacerItem *hSpacer4 = new
184             QSpacerItem(40, 80,
185                     QSizePolicy::Expanding,
186                     QSizePolicy::Expanding);
187     hBoxLayout[1]->addWidget(pushButton[0]);
188     hBoxLayout[1]->addSpacerItem(hSpacer3);
189     hBoxLayout[1]->addWidget(pushButton[1]);
190     hBoxLayout[1]->addSpacerItem(hSpacer4);
191     hBoxLayout[1]->addWidget(pushButton[2]);
192     hBoxLayout[1]->setContentsMargins(0, 0, 0, 0);
193
194     hWidget[1]->setLayout(hBoxLayout[1]);
195
196     /* V1 布局 */
197     QSpacerItem *vSpacer2 = new
198             QSpacerItem(320, 40,
199                     QSizePolicy::Minimum,
200                     QSizePolicy::Maximum);
201     QSpacerItem *vSpacer3 = new
202             QSpacerItem(320, 20,
203                     QSizePolicy::Minimum,
204                     QSizePolicy::Maximum);
205     QSpacerItem *vSpacer4 = new
206             QSpacerItem(320, 30,
207                     QSizePolicy::Minimum,
208                     QSizePolicy::Minimum);
209     label[1]->setMinimumSize(320, 320);
210     QImage Image;
211     Image.load(":/images/cd.png");
212     QPixmap pixmap = QPixmap::fromImage(Image);
213     int with = 320;
214     int height = 320;
215     QPixmap fitpixmap =
216             pixmap.scaled(with, height,
217                           Qt::IgnoreAspectRatio,
218                           Qt::SmoothTransformation);
219     label[1]->setPixmap(fitpixmap);
220     label[1]->setAlignment(Qt::AlignCenter);
221     vWidget[2]->setMinimumSize(300, 80);
222     vWidget[2]->setMaximumHeight(80);
223     vBoxLayout[1]->addSpacerItem(vSpacer2);
224     vBoxLayout[1]->addWidget(label[1]);
225     vBoxLayout[1]->addSpacerItem(vSpacer3);
```

```
226     vBoxLayout[1]->addWidget(durationSlider);
227     vBoxLayout[1]->addWidget(vWidget[2]);
228     vBoxLayout[1]->addSpacerItem(vSpacer4);
229     vBoxLayout[1]->setContentsMargins(0, 0, 0, 0);
230
231     vWidget[1]->setLayout(vBoxLayout[1]);
232
233     /* V2 布局 */
234     QSpacerItem *vSpacer5 = new
235           QSpacerItem(300, 10,
236                       QSizePolicy::Minimum,
237                       QSizePolicy::Maximum);
238     hWidget[2]->setMinimumSize(320, 20);
239     hWidget[3]->setMinimumSize(320, 60);
240     vBoxLayout[2]->addWidget(hWidget[2]);
241     vBoxLayout[2]->addSpacerItem(vSpacer5);
242     vBoxLayout[2]->addWidget(hWidget[3]);
243     vBoxLayout[2]->setContentsMargins(0, 0, 0, 0);
244
245     vWidget[2]->setLayout(vBoxLayout[2]);
246
247     /* H2 布局 */
248     label[2]->setText("00:00");
249     label[3]->setText("00:00");
250     QFont font;
251
252     font.setPixelSize(10);
253
254     /* 设置标签文本 */
255     label[0]->setText("Q Music, Enjoy it! ");
256     label[2]->setText("00:00");
257     label[3]->setText("00:00");
258     label[2]->setSizePolicy(QSizePolicy::Expanding,
259                             QSizePolicy::Expanding);
260     label[3]->setSizePolicy(QSizePolicy::Expanding,
261                             QSizePolicy::Expanding);
262     label[3]->setAlignment(Qt::AlignRight);
263     label[2]->setAlignment(Qt::AlignLeft);
264     label[2]->setFont(font);
265     label[3]->setFont(font);
266
267     pal.setColor(QPalette::WindowText, Qt::white);
268     label[0]->setPalette(pal);
269     label[2]->setPalette(pal);
```

```
270     label[3]->setPalette(pal);
271
272     hBoxLayout[2]->addWidget(label[2]);
273     hBoxLayout[2]->addWidget(label[3]);
274
275     hBoxLayout[2]->setContentsMargins(0, 0, 0, 0);
276     hWidget[2]->setLayout(hBoxLayout[2]);
277
278     /* H3 布局 */
279     QSpacerItem *hSpacer5 = new
280             QSpacerItem(0, 60,
281                         QSizePolicy::Minimum,
282                         QSizePolicy::Maximum);
283     QSpacerItem *hSpacer6 = new
284             QSpacerItem(80, 60,
285                         QSizePolicy::Maximum,
286                         QSizePolicy::Maximum);
287     QSpacerItem *hSpacer7 = new
288             QSpacerItem(80, 60,
289                         QSizePolicy::Maximum,
290                         QSizePolicy::Maximum);
291     QSpacerItem *hSpacer8 = new
292             QSpacerItem(80, 60,
293                         QSizePolicy::Maximum,
294                         QSizePolicy::Maximum);
295     QSpacerItem *hSpacer9 = new
296             QSpacerItem(0, 60,
297                         QSizePolicy::Minimum,
298                         QSizePolicy::Maximum);
299
300     for (int i = 3; i < 7; i++) {
301         pushButton[i]->setMinimumSize(25, 25);
302         pushButton[i]->setMaximumSize(25, 25);
303     }
304
305     hBoxLayout[3]->addSpacerItem(hSpacer5);
306     hBoxLayout[3]->addWidget(pushButton[3]);
307     hBoxLayout[3]->addSpacerItem(hSpacer6);
308     hBoxLayout[3]->addWidget(pushButton[4]);
309     hBoxLayout[3]->addSpacerItem(hSpacer7);
310     hBoxLayout[3]->addWidget(pushButton[5]);
311     hBoxLayout[3]->addSpacerItem(hSpacer8);
312     hBoxLayout[3]->addWidget(pushButton[6]);
313     hBoxLayout[3]->addSpacerItem(hSpacer9);
```

```
314     hBoxLayout[3]->setContentsMargins(0, 0, 0, 0);
315     hBoxLayout[3]->setAlignment(Qt::AlignHCenter);
316
317     hWidget[3]->setLayout(hBoxLayout[3]);
318
319     //hWidget[0]->setStyleSheet("background-color:red");
320     //hWidget[1]->setStyleSheet("background-color:#ff5599");
321     //hWidget[2]->setStyleSheet("background-color:#ff55ff");
322     //hWidget[3]->setStyleSheet("background-color:black");
323     //vWidget[0]->setStyleSheet("background-color:#555555");
324     //vWidget[1]->setStyleSheet("background-color:green");
325     //vWidget[2]->setStyleSheet("background-color:gray");
326
327 }
328
329 MainWindow::~MainWindow()
330 {
331 }
332
333 void MainWindow::btn_play_clicked()
334 {
335     int state = musicPlayer->state();
336
337     switch (state) {
338     case QMediaPlayer::StoppedState:
339         /* 媒体播放 */
340         musicPlayer->play();
341         break;
342
343     case QMediaPlayer::PlayingState:
344         /* 媒体暂停 */
345         musicPlayer->pause();
346         break;
347
348     case QMediaPlayer::PausedState:
349         musicPlayer->play();
350         break;
351     }
352 }
353
354 void MainWindow::btn_next_clicked()
355 {
356     musicPlayer->stop();
357     int count = mediaPlaylist->mediaCount();
```

```
358     if (0 == count)
359         return;
360 
361     /* 列表下一个 */
362     mediaPlaylist->next();
363     musicPlayer->play();
364 }
365 
366 void MainWindow::btn_previous_clicked()
367 {
368     musicPlayer->stop();
369     int count = mediaPlaylist->mediaCount();
370     if (0 == count)
371         return;
372 
373     /* 列表上一个 */
374     mediaPlaylist->previous();
375     musicPlayer->play();
376 }
377 
378 void MainWindow::mediaPlayerStateChanged(
379         QMediaPlayer::State
380         state)
381 {
382     switch (state) {
383     case QMediaPlayer::StoppedState:
384         pushButton[1]->setChecked(false);
385         break;
386 
387     case QMediaPlayer::PlayingState:
388         pushButton[1]->setChecked(true);
389         break;
390 
391     case QMediaPlayer::PausedState:
392         pushButton[1]->setChecked(false);
393         break;
394     }
395 }
396 
397 void MainWindow::listWidgetCliked(QListWidgetItem *item)
398 {
399     musicPlayer->stop();
400     mediaPlaylist->setCurrentIndex(listWidget->row(item));
401     musicPlayer->play();
```

```
402 }
403
404 void MainWindow::mediaPlaylistCurrentIndexChanged(
405         int index)
406 {
407     if (-1 == index)
408         return;
409
410     /* 设置列表正在播放的项 */
411     listWidget->setCurrentRow(index);
412 }
413
414 void MainWindow::musicPlayerDurationChanged(
415         qint64 duration)
416 {
417     durationSlider->setRange(0, duration / 1000);
418     int second  = duration / 1000;
419     int minute = second / 60;
420     second %= 60;
421
422     QString mediaDuration;
423     mediaDuration.clear();
424
425     if (minute >= 10)
426         mediaDuration = QString::number(minute, 10);
427     else
428         mediaDuration = "0" + QString::number(minute, 10);
429
430     if (second >= 10)
431         mediaDuration = mediaDuration
432                 + ":" + QString::number(second, 10);
433     else
434         mediaDuration = mediaDuration
435                 + ":0" + QString::number(second, 10);
436
437     /* 显示媒体总长度时间 */
438     label[3]->setText(mediaDuration);
439 }
440
441 void MainWindow::mediaPlayerPositionChanged(
442         qint64 position)
443 {
444     if (!durationSlider->isSliderDown())
445         durationSlider->setValue(position/1000);
```

```
446
447     int second  = position / 1000;
448     int minute = second / 60;
449     second %= 60;
450
451     QString mediaPosition;
452     mediaPosition.clear();
453
454     if (minute >= 10)
455         mediaPosition = QString::number(minute, 10);
456     else
457         mediaPosition = "0" + QString::number(minute, 10);
458
459     if (second >= 10)
460         mediaPosition = mediaPosition
461                 + ":" + QString::number(second, 10);
462     else
463         mediaPosition = mediaPosition
464                 + ":0" + QString::number(second, 10);
465
466     /* 显示现在播放的时间 */
467     label[2]->setText(mediaPosition);
468 }
469
470 void MainWindow::resizeEvent(QResizeEvent *event)
471 {
472     Q_UNUSED(event);
473     listMask->setGeometry(0,
474                           listWidget->height() - 50,
475                           310,
476                           50);
477 }
478
479 void MainWindow::durationSliderReleased()
480 {
481     /* 设置媒体播放的位置 */
482     musicPlayer->setPosition(durationSlider->value() * 1000);
483 }
484
485 void MainWindow::scanSongs()
486 {
487     QDir dir(QCoreApplication::applicationDirPath()
488             + "/myMusic");
489     QDir dirbsolutePath(dir.absolutePath());
```

```
490     /* 如果目录存在 */
491     if (dirbsolutePath.exists()) {
492         /* 定义过滤器 */
493         QStringList filter;
494         /* 包含所有.mp3 后缀的文件 */
495         filter << "*.mp3";
496         /* 获取该目录下的所有文件 */
497         QFileInfoList files =
498                 dirbsolutePath.entryInfoList(filter, QDir::Files);
499         /* 遍历 */
500         for (int i = 0; i < files.count(); i++) {
501             MediaObjectInfo info;
502             /* 使用 utf-8 编码 */
503             QString fileName = QString::fromUtf8(files.at(i)
504                                                  .fileName()
505                                                  .replace(".mp3", "")
506                                                  .toUtf8()
507                                                  .data());
508             info.fileName = fileName + "\n"
509                     + fileName.split("-").at(1);
510             info.filePath = QString::fromUtf8(files.at(i)
511                                               .filePath()
512                                               .toUtf8()
513                                               .data());
514             /* 媒体列表添加歌曲 */
515             if (mediaPlaylist->addMedia(
516                     QUrl::fromLocalFile(info.filePath))) {
517                 /* 添加到容器数组里储存 */
518                 mediaObjectInfo.append(info);
519                 /* 添加歌曲名字至列表 */
520                 listWidget->addItem(info.fileName);
521             } else {
522                 qDebug()<<
523                           mediaPlaylist->errorString()
524                           .toUtf8().data()
525                         << endl;
526                 qDebug()<< "  Error number:"
527                          << mediaPlaylist->error()
528                          << endl;
529             }
530         }
531     }
532 }
```

```
533
534 void MainWindow::mediaPlayerInit()
535 {
536     musicPlayer = new QMediaPlayer(this);
537     mediaPlaylist = new QMediaPlaylist(this);
538     /* 确保列表是空的 */
539     mediaPlaylist->clear();
540     /* 设置音乐播放器的列表为 mediaPlaylist */
541     musicPlayer->setPlaylist(mediaPlaylist);
542     /* 设置播放模式，Loop 是列循环 */
543     mediaPlaylist->setPlaybackMode(QMediaPlaylist::Loop);
544 }
```

第 10 行，布局初始化，第一步我们先建立好界面，确定好布局再实现功能，一般流程都这样，布局 msuicLayout()的内容比较多，也不难，但是比较复杂，如果我们有第七章的基础，看这种布局是没有难度的，这里就不多解释了。没有好看的布局也能完成本例。如果您喜欢这种布局方法，您需要多花点时间去研究如何布局才好看，这些没有固定的方法，完全是一个人的审美感。

第 13 行，媒体先初始化，初始化媒体播放器与媒体播放列表。

第 16 行，扫描歌曲，初始化本地歌曲，从固定的文件夹里找歌曲文件，这里笔者设计从固定的文件夹里找歌曲文件。也有些读者可能会说可以自由选择文件夹吗？答案是可以的！使用 QFileDilog 类打开选择歌曲目录或者文件即可！但是在嵌入式里，一般是初始化界里面就已经有歌曲在界面里的了，无需用户再去打开，打开歌曲这种操作是极少会使用的！

第 26 至 47 行，信号槽连接，这里使用了各种各样的信号。这里有必要说明一下，笔者设计了单击歌曲列表就能播放歌曲了。如果在电脑上可能有些播放器软件会双击才能播放歌曲，在嵌入式的触摸屏里，只有单击！没有双击，没有用户会双击歌曲的列表的。这些特殊的地方我们需要在嵌入式里考虑！

第 333~376 行，点击播放按钮，上一曲，下一曲，Qt 的媒体类已经提供了 previous()，next()，stop()，play()，pause()这些方法直接可以使用。除了实现好看的界面之外，这部分内容也是本例实现播放器重要的部分！

其他部分是实现播放状态的切换及扩进度条的显示进度处理，请参阅源码理解。

main.cpp 内容如下，主要是加载 qss 样式文件。没有什么可讲解。

```
1  #include "mainwindow.h"
2
3  #include <QApplication>
4  #include <QFile>
5
6  int main(int argc, char *argv[])
7  {
8      QApplication a(argc, argv);
9      /* 指定文件 */
```

```
10      QFile file(":/style.qss");
11
12      /* 判断文件是否存在 */
13      if (file.exists() ) {
14          /* 以只读的方式打开 */
15          file.open(QFile::ReadOnly);
16          /* 以字符串的方式保存读出的结果 */
17          QString styleSheet = QLatin1String(file.readAll());
18          /* 设置全局样式 */
19          qApp->setStyleSheet(styleSheet);
20          /* 关闭文件 */
21          file.close();
22      }
23
24      MainWindow w;
25      w.show();
26      return a.exec();
27  }
```

style.qss 样式文件如下。素材已经在源码处提供。注意下面的 style.qss 不能有注释！

```
1   QWidget {
2   background: "#25242a"
3   }
4
5   QWidget#listMask {
6   border-image: url(:/images/mask.png);
7   background-color: transparent;
8   }
9
10  QListWidget#listWidget {
11  color:white;
12  font-size: 15px;
13  border:none;
14  }
15
16  QListWidget#listWidget:item:active {
17  background: transparent;
18  }
19
20  QListWidget#listWidget:item {
21  background: transparent;
22  height:60;
23  }
24
```

```
25  QListWidget#listWidget:item:selected {
26  color:#5edcf3;
27  background: transparent;
28  }
29
30  QListWidget#listWidget:item:hover {
31  background: transparent;
32  color:#5edcf3;
33  border:none;
34  }
35
36  QPushButton#btn_play {
37  border-image:url(:/images/btn_play1.png);
38  }
39
40  QPushButton#btn_play:hover {
41  border-image:url(:/images/btn_play2.png);
42  }
43
44  QPushButton#btn_play:checked {
45  border-image:url(:/images/btn_pause1.png);
46  }
47
48  QPushButton#btn_play:checked:hover {
49  border-image:url(:/images/btn_pause2.png);
50  }
51
52  QPushButton#btn_previous {
53  border-image:url(:/images/btn_previous1.png);
54  }
55
56  QPushButton#btn_previous:hover {
57  border-image:url(:/images/btn_previous2.png);
58  }
59
60  QPushButton#btn_next {
61  border-image:url(:/images/btn_next1.png);
62  }
63
64  QPushButton#btn_next:hover {
65  border-image:url(:/images/btn_next2.png);
66  }
67
68  QPushButton#btn_favorite {
```

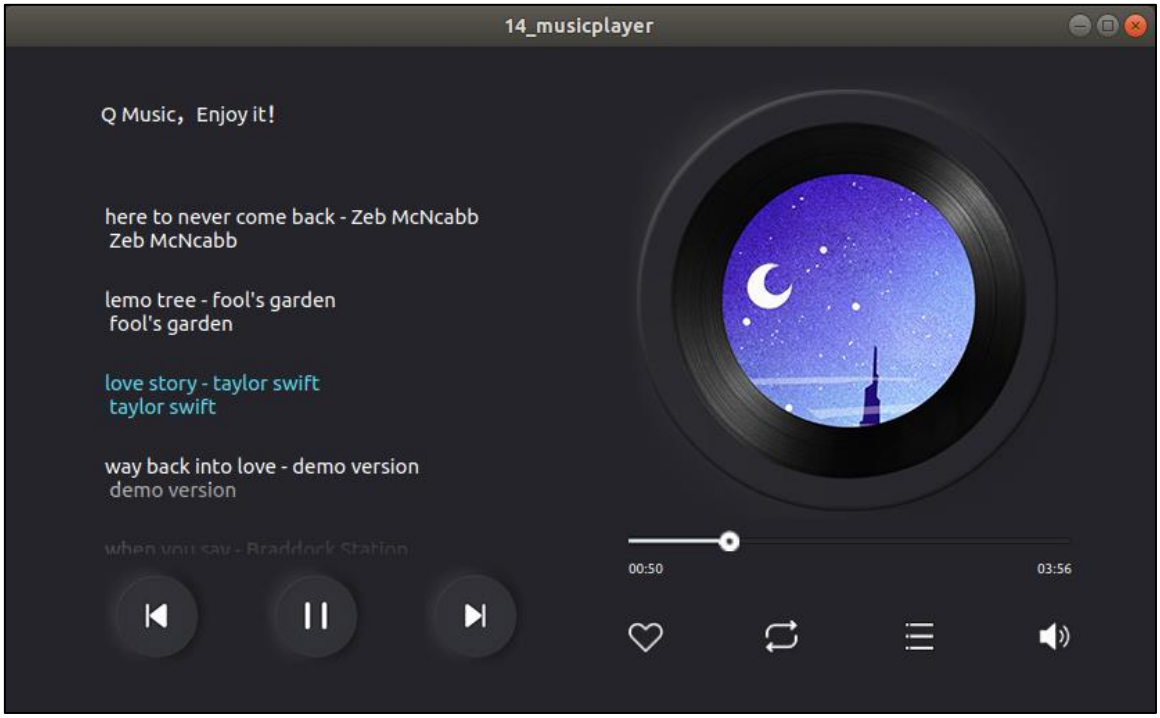
```
69  border-image:url(:/images/btn_favorite_no.png);
70  }
71
72  QPushButton#btn_favorite:checked {
73  border-image:url(:/images/btn_favorite_yes.png);
74  }
75
76  QPushButton#btn_menu {
77  border-image:url(:/images/btn_menu1.png);
78  }
79
80  QPushButton#btn_menu:hover {
81  border-image:url(:/images/btn_menu2.png);
82  }
83
84  QPushButton#btn_mode {
85  border-image:url(:/images/btn_listcircle1.png);
86  }
87
88  QPushButton#btn_mode:hover {
89  border-image:url(:/images/btn_listcircle2.png);
90  }
91
92  QPushButton#btn_mode {
93  border-image:url(:/images/btn_listcircle1.png);
94  }
95
96  QPushButton#btn_mode:hover {
97  border-image:url(:/images/btn_listcircle2.png);
98  }
99
100 QPushButton#btn_volume {
101 border-image:url(:/images/btn_volume1.png);
102 }
103
104 QPushButton#btn_volume:hover {
105 border-image:url(:/images/btn_volume2.png);
106 }
107
108 QSlider#durationSlider:handle:horizontal {
109 border-image:url(:/images/handle.png);
110 }
111
112 QSlider#durationSlider:sub-page:horizontal {
```

```
113 border-image:url(:/images/sub-page.png);
114 }
```

12.3.2 程序运行效果

先点击构建项目，项目构建完成后，再将本例的 myMusic 歌曲文件夹拷贝到可执行程序的文件夹同一级目录下，也就是 build-14_musicplayer-Desktop_Qt_5_12_9_GCC_64bit-Debug 目录下（windows 需要进入到 debug 目录）。再点击运行，就出现歌曲在列表里，如下图，点击播放即可播放歌曲，上一曲，下一曲也可以用。注意右下角的某些按钮功能，在本例没有继续去实现，比如音量控制，可以直接加一个垂直方向的滑条，然后控制媒体的软件音量即可。留给读者自由发挥，可以基于本例去开发，就当读者练习吧。本例的界面开发笔者还是比较满意的，前面的界面都比较普通，笔者个人 Qt 开发重要的是界面与稳定性，功能是次要的！因为功能都不难实现。

注意歌曲格式应严格为歌名 + “-” + 歌手名称.mp3，例如江南 - 林俊杰.mp3。中间的“-”是英文字符的短横杠，这样的目的是能够将歌曲名称和歌手分开显示到界面上。



12.4 视频播放器

与音乐播放器一样使用 QMediaPlayer 类，不同的是需要使用 setVideoOutput(QVideoWidget*) 设置一个视频输出窗口，好让视频在此窗口显示，其他步骤基本都一样。

12.4.1 应用实例

本例设计一个比较好看且简洁的视频播放器，界面是笔者原创界面。

本例目的：视频播放器的设计与使用。

例 15_videoplayer，视频播放器（难度：中等）。项目路径为 Qt/2/15_videoplayer。注意本例有用到 qss 样式文件，关于如何添加资源文件与 qss 文件请参考 7.1.3 小节。音乐播放器的功能这些都为大家所熟知，不用笔者介绍了。

项目文件 15_videoplayer.pro 文件第一行添加的代码部分如下。

15_videoplayer.pro 编程后的代码

```
1  QT       += core gui multimedia multimediawidgets
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
13 # You can also make your code fail to compile if it uses deprecated APIs.
14 # In order to do so, uncomment the following line.
15 # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18 SOURCES += \
19     main.cpp \
20     mainwindow.cpp
21
22 HEADERS += \
23     mainwindow.h
24
25 # Default rules for deployment.
26 qnx: target.path = /tmp/${TARGET}/bin
27 else: unix:!android: target.path = /opt/${TARGET}/bin
28 !isEmpty(target.path): INSTALLS += target
29
30 RESOURCES += \
31     res.qrc
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   15_videoplayer
```

```
    * @brief        mainwindow.h
    * @author       Deng Zhimao
    * @email        1252699831@qq.com
    * @net          www.openedv.com
    * @date         2021-04-27
    ****************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QMediaPlayer>
6   #include <QMediaPlaylist>
7   #include <QPushButton>
8   #include <QSlider>
9   #include <QVBoxLayout>
10  #include <QHBoxLayout>
11  #include <QListWidget>
12  #include <QLabel>
13  #include <QSpacerItem>
14  #include <QVideoWidget>
15  #include <QDebug>
16
17  /* 媒体信息结构体 */
18  struct MediaObjectInfo {
19      /* 用于保存视频文件名 */
20      QString fileName;
21      /* 用于保存视频文件路径 */
22      QString filePath;
23  };
24
25  class MainWindow : public QMainWindow
26  {
27      Q_OBJECT
28
29  public:
30      MainWindow(QWidget *parent = nullptr);
31      ~MainWindow();
32
33  private:
34      /* 媒体播放器，用于播放视频 */
35      QMediaPlayer *videoPlayer;
36
37      /* 媒体列表 */
```

```
38      QMediaPlaylist *mediaPlaylist;
39
40      /* 视频显示窗口 */
41      QVideoWidget *videoWidget;
42
43      /* 视频列表 */
44      QListWidget *listWidget;
45
46      /* 播放进度条 */
47      QSlider *durationSlider;
48
49      /* 音量条 */
50      QSlider *volumeSlider;
51
52      /* 视频播放器按钮 */
53      QPushButton *pushButton[5];
54
55      /* 水平布局 */
56      QHBoxLayout *hBoxLayout[3];
57
58      /* 水平容器 */
59      QWidget *hWidget[3];
60
61      /* 标签文本 */
62      QLabel *label[2];
63
64      /* 垂直容器 */
65      QWidget *vWidget[2];
66
67      /* 垂直界面 */
68      QVBoxLayout *vBoxLayout[2];
69
70      /* 视频布局函数 */
71      void videoLayout();
72
73      /* 主窗体大小重设大小函数重写 */
74      void resizeEvent(QResizeEvent *event);
75
76      /* 媒体信息存储 */
77      QVector<MediaObjectInfo> mediaObjectInfo;
78
79      /* 扫描本地视频文件 */
```

```
80      void scanVideoFiles();
81
82      /* 媒体初始化 */
83      void mediaPlayerInit();
84  private slots:
85      /* 播放按钮点击 */
86      void btn_play_clicked();
87
88      /* 下一个视频按钮点击 */
89      void btn_next_clicked();
90
91      /* 音量加 */
92      void btn_volmeup_clicked();
93
94      /* 音量减 */
95      void btn_volmedown_clicked();
96
97      /* 全屏 */
98      void btn_fullscreen_clicked();
99
100     /* 媒体状态改变 */
101     void mediaPlayerStateChanged(QMediaPlayer::State);
102
103     /* 列表单击 */
104     void listWidgetCliked(QListWidgetItem*);
105
106     /* 媒体列表项改变 */
107     void mediaPlaylistCurrentIndexChanged(int);
108
109     /* 媒体总长度改变 */
110     void musicPlayerDurationChanged(qint64);
111
112     /* 媒体播放位置改变 */
113     void mediaPlayerPositionChanged(qint64);
114
115     /* 播放进度条松开 */
116     void durationSliderReleased();
117
118     /* 音量条松开 */
119     void volumeSliderReleased();
120 };
121 #endif // MAINWINDOW_H
```

头文件里主要是声明界面所使用的元素及一些槽函数。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   15_videoplayer
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-27
*****************************************************************/
#include "mainwindow.h"
#include <QCoreApplication>
#include <QFileInfoList>
#include <QDir>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    /* 视频播放器布局初始化 */
    videoLayout();

    /* 媒体初始化 */
    mediaPlayerInit();

    /* 扫描本地视频 */
    scanVideoFiles();

    /* 设置按钮的属性 */
    pushButton[0]->setCheckable(true);
    pushButton[4]->setCheckable(true);

    /* 按钮连接信号槽 */
    connect(pushButton[0], SIGNAL(clicked()),
            this, SLOT(btn_play_clicked()));
    connect(pushButton[1], SIGNAL(clicked()),
            this, SLOT(btn_next_clicked()));
    connect(pushButton[2], SIGNAL(clicked()),
            this, SLOT(btn_volmedown_clicked()));
    connect(pushButton[3], SIGNAL(clicked()),
            this, SLOT(btn_volmeup_clicked()));
    connect(pushButton[4], SIGNAL(clicked()),
```

```
32              this, SLOT(btn_fullscreen_clicked()));
33
34      /* 列表连接信号槽 */
35      connect(listWidget, SIGNAL(itemClicked(QListWidgetItem*)),
36              this, SLOT(listWidgetCliked(QListWidgetItem*)));
37
38      /* 媒体连接信号槽 */
39      connect(videoPlayer,
40              SIGNAL(stateChanged(QMediaPlayer::State)),
41              this,
42              SLOT(mediaPlayerStateChanged(QMediaPlayer::State)));
43      connect(mediaPlaylist,
44              SIGNAL(currentIndexChanged(int)),
45              this,
46              SLOT(mediaPlaylistCurrentIndexChanged(int)));
47      connect(videoPlayer, SIGNAL(durationChanged(qint64)),
48              this,
49              SLOT(musicPlayerDurationChanged(qint64)));
50      connect(videoPlayer,
51              SIGNAL(positionChanged(qint64)),
52              this,
53              SLOT(mediaPlayerPositionChanged(qint64)));
54
55      /* slider 信号槽连接 */
56      connect(durationSlider, SIGNAL(sliderReleased()),
57              this, SLOT(durationSliderReleased()));
58      connect(volumeSlider, SIGNAL(sliderReleased()),
59              this, SLOT(volumeSliderReleased()));
60  }
61
62  MainWindow::~MainWindow()
63  {
64  }
65
66  void MainWindow::videoLayout()
67  {
68      /* 设置位置与大小,这里固定为 800, 480 */
69      this->setGeometry(0, 0, 800, 480);
70      //    this->setMinimumSize(800, 480);
71      //    this->setMaximumSize(800, 480);
72      QPalette pal;
73      pal.setColor(QPalette::WindowText, Qt::white);
74
75      for (int i = 0; i < 3; i++) {
```

```
76          /* 水平容器 */
77          hWidget[i] = new QWidget();
78          hWidget[i]->setAutoFillBackground(true);
79          /* 水平布局 */
80          hBoxLayout[i] = new QHBoxLayout();
81      }
82
83      for (int i = 0; i < 2; i++) {
84          /* 垂直容器 */
85          vWidget[i] = new QWidget();
86          vWidget[i]->setAutoFillBackground(true);
87          /* 垂直布局 */
88          vBoxLayout[i] = new QVBoxLayout();
89      }
90
91      for (int i = 0; i < 2; i++) {
92          label[i] = new QLabel();
93      }
94
95      for (int i = 0; i < 5; i++) {
96          pushButton[i] = new QPushButton();
97          pushButton[i]->setMaximumSize(44, 44);
98          pushButton[i]->setMinimumSize(44, 44);
99      }
100
101     /* 设置 */
102     vWidget[0]->setObjectName("vWidget0");
103     vWidget[1]->setObjectName("vWidget1");
104     hWidget[1]->setObjectName("hWidget1");
105     hWidget[2]->setObjectName("hWidget2");
106     pushButton[0]->setObjectName("btn_play");
107     pushButton[1]->setObjectName("btn_next");
108     pushButton[2]->setObjectName("btn_volumedown");
109     pushButton[3]->setObjectName("btn_volumeup");
110     pushButton[4]->setObjectName("btn_screen");
111
112     QFont font;
113
114     font.setPixelSize(18);
115     label[0]->setFont(font);
116     label[1]->setFont(font);
117
118     pal.setColor(QPalette::WindowText, Qt::white);
```

```
119     label[0]->setPalette(pal);
120     label[1]->setPalette(pal);
121
122     label[0]->setText("00:00");
123     label[1]->setText("/00:00");
124
125     durationSlider = new QSlider(Qt::Horizontal);
126     durationSlider->setMaximumHeight(15);
127     durationSlider->setObjectName("durationSlider");
128
129     volumeSlider = new QSlider(Qt::Horizontal);
130     volumeSlider->setRange(0, 100);
131     volumeSlider->setMaximumWidth(80);
132     volumeSlider->setObjectName("volumeSlider");
133     volumeSlider->setValue(50);
134
135     listWidget = new QListWidget();
136     listWidget->setObjectName("listWidget");
137     listWidget->setVerticalScrollBarPolicy(
138               Qt::ScrollBarAlwaysOff);
139     listWidget->setHorizontalScrollBarPolicy(
140               Qt::ScrollBarAlwaysOff);
141     //listWidget->setFocusPolicy(Qt::NoFocus);
142     videoWidget = new QVideoWidget();
143     videoWidget->setStyleSheet("border-image: none;"
144                           "background: transparent;"
145                           "border:none");
146
147     /* H0 布局 */
148     vWidget[0]->setMinimumSize(300, 480);
149     vWidget[0]->setMaximumWidth(300);
150     videoWidget->setMinimumSize(500, 480);
151
152     hBoxLayout[0]->addWidget(videoWidget);
153     hBoxLayout[0]->addWidget(vWidget[0]);
154
155     hWidget[0]->setLayout(hBoxLayout[0]);
156     hBoxLayout[0]->setContentsMargins(0, 0, 0, 0);
157
158     setCentralWidget(hWidget[0]);
159
160     /* V0 布局 */
161     QSpacerItem *vSpacer0 = new
162           QSpacerItem(0, 80,
```

```
163                     QSizePolicy::Minimum,
164                     QSizePolicy::Maximum);
165     vBoxLayout[0]->addWidget(listWidget);
166     vBoxLayout[0]->addSpacerItem(vSpacer0);
167     vBoxLayout[0]->setContentsMargins(0, 0, 0, 0);
168
169     vWidget[0]->setLayout(vBoxLayout[0]);
170
171     /* V1 布局 */
172     /* 底板部件布局 */
173     hWidget[1]->setMaximumHeight(15);
174     hWidget[2]->setMinimumHeight(65);
175     vBoxLayout[1]->addWidget(hWidget[1]);
176     vBoxLayout[1]->addWidget(hWidget[2]);
177     vBoxLayout[1]->setAlignment(Qt::AlignCenter);
178
179     vWidget[1]->setLayout(vBoxLayout[1]);
180     vWidget[1]->setParent(this);
181     vWidget[1]->setGeometry(0, this->height() - 80, this->width(), 80);
182     vBoxLayout[1]->setContentsMargins(0, 0, 0, 0);
183     /* 位于最上层 */
184     vWidget[1]->raise();
185
186     /* H1 布局 */
187     hBoxLayout[1]->addWidget(durationSlider);
188     hBoxLayout[1]->setContentsMargins(0, 0, 0, 0);
189     hWidget[1]->setLayout(hBoxLayout[1]);
190
191     /* H2 布局 */
192     QSpacerItem *hSpacer0 = new
193           QSpacerItem(300, 80,
194                     QSizePolicy::Expanding,
195                     QSizePolicy::Maximum);
196
197     hBoxLayout[2]->addSpacing(20);
198     hBoxLayout[2]->addWidget(pushButton[0]);
199     hBoxLayout[2]->addSpacing(10);
200     hBoxLayout[2]->addWidget(pushButton[1]);
201     hBoxLayout[2]->addSpacing(10);
202     hBoxLayout[2]->addWidget(pushButton[2]);
203     hBoxLayout[2]->addWidget(volumeSlider);
204     hBoxLayout[2]->addWidget(pushButton[3]);
205     hBoxLayout[2]->addWidget(label[0]);
```

```
206     hBoxLayout[2]->addWidget(label[1]);
207     hBoxLayout[2]->addSpacerItem(hSpacer0);
208     hBoxLayout[2]->addWidget(pushButton[4]);
209     hBoxLayout[2]->addSpacing(20);
210     hBoxLayout[2]->setContentsMargins(0, 0, 0, 0);
211     hBoxLayout[2]->setAlignment(Qt::AlignLeft | Qt::AlignTop);
212
213     hWidget[2]->setLayout(hBoxLayout[2]);
214 }
215
216 void MainWindow::mediaPlayerInit()
217 {
218     videoPlayer = new QMediaPlayer(this);
219     mediaPlaylist = new QMediaPlaylist(this);
220     /* 确保列表是空的 */
221     mediaPlaylist->clear();
222     /* 设置视频播放器的列表为 mediaPlaylist */
223     videoPlayer->setPlaylist(mediaPlaylist);
224     /* 设置视频输出窗口 */
225     videoPlayer->setVideoOutput(videoWidget);
226     /* 设置播放模式，Loop 是列循环 */
227     mediaPlaylist->setPlaybackMode(QMediaPlaylist::Loop);
228     /* 设置默认软件音量为 50% */
229     videoPlayer->setVolume(50);
230 }
231
232 void MainWindow::resizeEvent(QResizeEvent *event)
233 {
234     Q_UNUSED(event);
235     vWidget[1]->setGeometry(0, this->height() - 80, this->width(), 80);
236 }
237
238 void MainWindow::btn_play_clicked()
239 {
240     int state = videoPlayer->state();
241     switch (state) {
242     case QMediaPlayer::StoppedState:
243         /* 媒体播放 */
244         videoPlayer->play();
245         break;
246
247     case QMediaPlayer::PlayingState:
248         /* 媒体暂停 */
```

```
249         videoPlayer->pause();
250         break;
251
252     case QMediaPlayer::PausedState:
253         /* 设置视频输出窗口 */
254         videoPlayer->play();
255         break;
256     }
257 }
258
259 void MainWindow::btn_next_clicked()
260 {
261     videoPlayer->stop();
262     int count = mediaPlaylist->mediaCount();
263     if (0 == count)
264         return;
265
266     /* 列表下一个 */
267     mediaPlaylist->next();
268     videoPlayer->play();
269 }
270
271 void MainWindow::btn_volmeup_clicked()
272 {
273     /* 点击每次音量+5 */
274     volumeSlider->setValue(volumeSlider->value() + 5);
275     videoPlayer->setVolume(volumeSlider->value());
276 }
277
278 void MainWindow::btn_fullscreen_clicked()
279 {
280     /* 全屏/非全屏操作 */
281     vWidget[0]->setVisible(!pushButton[4]->isChecked());
282 }
283
284 void MainWindow::btn_volmedown_clicked()
285 {
286     /* 点击每次音量-5 */
287     volumeSlider->setValue(volumeSlider->value() - 5);
288     videoPlayer->setVolume(volumeSlider->value());
289 }
290
291 void MainWindow::mediaPlayerStateChanged(
```

```
292         QMediaPlayer::State
293         state)
294 {
295     switch (state) {
296     case QMediaPlayer::StoppedState:
297         pushButton[0]->setChecked(false);
298         break;
299
300     case QMediaPlayer::PlayingState:
301         pushButton[0]->setChecked(true);
302         break;
303
304     case QMediaPlayer::PausedState:
305         pushButton[0]->setChecked(false);
306         break;
307     }
308 }
309
310 void MainWindow::listWidgetCliked(QListWidgetItem *item)
311 {
312     videoPlayer->stop();
313     mediaPlaylist->setCurrentIndex(listWidget->row(item));
314     videoPlayer->play();
315 }
316
317 void MainWindow::mediaPlaylistCurrentIndexChanged(
318         int index)
319 {
320     if (-1 == index)
321         return;
322
323     /* 设置列表正在播放的项 */
324     listWidget->setCurrentRow(index);
325 }
326
327 void MainWindow::musicPlayerDurationChanged(
328         qint64 duration)
329 {
330     durationSlider->setRange(0, duration / 1000);
331     int second  = duration / 1000;
332     int minute = second / 60;
333     second %= 60;
334
335     QString mediaDuration;
```

```
336     mediaDuration.clear();
337
338     if (minute >= 10)
339         mediaDuration = QString::number(minute, 10);
340     else
341         mediaDuration = "0" + QString::number(minute, 10);
342
343     if (second >= 10)
344         mediaDuration = mediaDuration
345                 + ":" + QString::number(second, 10);
346     else
347         mediaDuration = mediaDuration
348                 + ":0" + QString::number(second, 10);
349
350     /* 显示媒体总长度时间 */
351     label[1]->setText("/" + mediaDuration);
352 }
353
354 void MainWindow::mediaPlayerPositionChanged(
355         qint64 position)
356 {
357     if (!durationSlider->isSliderDown())
358         durationSlider->setValue(position / 1000);
359
360     int second  = position / 1000;
361     int minute = second / 60;
362     second %= 60;
363
364     QString mediaPosition;
365     mediaPosition.clear();
366
367     if (minute >= 10)
368         mediaPosition = QString::number(minute, 10);
369     else
370         mediaPosition = "0" + QString::number(minute, 10);
371
372     if (second >= 10)
373         mediaPosition = mediaPosition
374                 + ":" + QString::number(second, 10);
375     else
376         mediaPosition = mediaPosition
377                 + ":0" + QString::number(second, 10);
378
379     /* 显示现在播放的时间 */
```

```
380     label[0]->setText(mediaPosition);
381 }
382
383 void MainWindow::durationSliderReleased()
384 {
385     /* 设置媒体播放的位置 */
386     videoPlayer->setPosition(durationSlider->value() * 1000);
387 }
388
389 void MainWindow::volumeSliderReleased()
390 {
391     /* 设置音量 */
392     videoPlayer->setVolume(volumeSlider->value());
393 }
394
395 void MainWindow::scanVideoFiles()
396 {
397     QDir dir(QCoreApplication::applicationDirPath()
398             + "/myVideo");
399     QDir dirbsolutePath(dir.absolutePath());
400     /* 如果目录存在 */
401     if (dirbsolutePath.exists()) {
402         /* 定义过滤器 */
403         QStringList filter;
404         /* 包含所有 xx 后缀的文件 */
405         filter << "*.mp4" << "*.mkv" << "*.wmv" << "*.avi";
406         /* 获取该目录下的所有文件 */
407         QFileInfoList files =
408                 dirbsolutePath.entryInfoList(filter, QDir::Files);
409         /* 遍历 */
410         for (int i = 0; i < files.count(); i++) {
411             MediaObjectInfo info;
412             /* 使用 utf-8 编码 */
413             info.fileName = QString::fromUtf8(files.at(i)
414                                               .fileName()
415                                               .toUtf8()
416                                               .data());
417             info.filePath = QString::fromUtf8(files.at(i)
418                                               .filePath()
419                                               .toUtf8()
420                                               .data());
421             /* 媒体列表添加视频 */
422             if (mediaPlaylist->addMedia(
```

```
423                     QUrl::fromLocalFile(info.filePath())) {
424                 /* 添加到容器数组里储存 */
425                 mediaObjectInfo.append(info);
426                 /* 添加视频名字至列表 */
427                 listWidget->addItem(info.fileName);
428             } else {
429                 qDebug()<<
430                           mediaPlaylist->errorString()
431                           .toUtf8().data()
432                         << endl;
433                 qDebug()<< "  Error number:"
434                         << mediaPlaylist->error()
435                         << endl;
436             }
437         }
438     }
439 }
```

与上一小节音乐播放器的一样，在构造函数里布局初始化，然后执行扫描本地视频文件。之后就是一些信号槽的连接，基本上就是这么一个流程了。

第 395~439 行，扫本地目录的视频文件，通过过滤文件名的后缀，将视频文件名添加至媒体列表里，就可以点击播放了，需要更多的格式的视频文件，可以自己尝试修改需要过滤的文件名。

main.cpp 内容如下，主要是加载 qss 样式文件。没有什么可讲解。

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4   #include <QFile>
5
6   int main(int argc, char *argv[])
7   {
8       QApplication a(argc, argv);
9       /* 指定文件 */
10      QFile file(":/style.qss");
11
12      /* 判断文件是否存在 */
13      if (file.exists() ) {
14          /* 以只读的方式打开 */
15          file.open(QFile::ReadOnly);
16          /* 以字符串的方式保存读出的结果 */
17          QString styleSheet = QLatin1String(file.readAll());
18          /* 设置全局样式 */
19          qApp->setStyleSheet(styleSheet);
```

```
20          /* 关闭文件 */
21          file.close();
22      }
23
24      MainWindow w;
25      w.show();
26      return a.exec();
27  }
```

style.qss 样式文件如下。素材已经在源码处提供。注意下面的 style.qss 不能有注释！

```
1   QWidget {
2   border-image:url(:/images/bg.png);
3   }
4
5   QLabel {
6   border-image:none;
7   }
8
9   QWidget#hWidget1 {
10  border-image:none;
11  background:transparent;
12  }
13
14  QWidget#hWidget2 {
15  border-image:none;
16  background:transparent;
17  }
18
19  QWidget#vWidget1 {
20  border-image:url(:/images/mask.png);
21  background:#24252a;
22  }
23
24  QWidget#vWidget0 {
25  border-image:none;
26  }
27
28  QListWidget#listWidget {
29  color:white;
30  font-size: 15px;
31  border:none;
32  background: "#20ffffff";
33  border-image:none;
34  }
35
```

```
36  QListWidgetlistWidget:item:active {
37  background: transparent;
38  }
39
40  QListWidget#listWidget:item {
41  background: transparent;
42  height:60;
43  }
44
45  QListWidget#listWidget:item:selected {
46  color:#5edcf3;
47  background: transparent;
48  }
49
50  QListWidget#listWidget:item:hover {
51  background: transparent;
52  color:#5edcf3;
53  border:none;
54  }
55
56
57  QPushButton#btn_play {
58  border-image:url(:/icons/btn_play1.png);
59  }
60
61  QPushButton#btn_play:hover {
62  border-image:url(:/icons/btn_play2.png);
63  }
64
65  QPushButton#btn_play:checked {
66  border-image:url(:/icons/btn_pause1.png);
67  }
68
69  QPushButton#btn_play:checked:hover {
70  border-image:url(:/icons/btn_pause2.png);
71  }
72
73  QPushButton#btn_next {
74  border-image:url(:/icons/btn_next1.png);
75  }
76
77  QPushButton#btn_next:hover {
78  border-image:url(:/icons/btn_next2.png);
79  }
```

```
80
81  QPushButton#btn_volumedown {
82  border-image:url(:/icons/btn_volumedown1.png);
83  }
84
85  QPushButton#btn_volumedown:hover {
86  border-image:url(:/icons/btn_volumedown2.png);
87  }
88
89  QPushButton#btn_volumeup {
90  border-image:url(:/icons/btn_volumeup1.png);
91  }
92
93  QPushButton#btn_volumeup:hover {
94  border-image:url(:/icons/btn_volumeup2.png);
95  }
96
97  QSlider#durationSlider:handle:horizontal {
98  border-image:url(:/icons/handle.png);
99  }
100
101 QSlider#durationSlider {
102 border-image:none;
103 }
104
105 QSlider#durationSlider:add-page:horizontal {
106 border-image:url(:/images/add_page.png);
107 }
108
109 QSlider#volumeSlider {
110 border-image:none;
111 }
112 QSlider#volumeSlider:handle:horizontal {
113 border-image:url(:/icons/handle.png);
114 }
115
116 QSlider#volumeSlider:handle:horizontal {
117 background:transparent;
118 }
119
120 QSlider#volumeSlider:add-page:horizontal {
121 border-image:url(:/images/add_page.png);
122 }
123
```

```
124 QPushButton#btn_screen {
125 border-image:url(:/icons/btn_fullscreen1.png);
126 }
127
128 QPushButton#btn_screen:hover {
129 border-image:url(:/icons/btn_fullscreen2.png);
130 }
131
132 QPushButton#btn_screen:checked {
133 border-image:url(:/icons/btn_screen1.png);
134 }
135
136 QPushButton#btn_screen:checked:hover {
137 border-image:url(:/icons/btn_screen2.png);
138 }
```

12.4.2 程序运行效果

先点击构建项目，项目构建完成后，再将本例的 myVideo 视频文件夹拷贝到可执行程序的文件夹同一级目录下，也就是 build-15_videoplayer-Desktop_Qt_5_12_9_GCC_64bit-Debug 目录下。再重新运行，就出现视频文件在列表里，如下图，点击播放即可播放视频。

默认列表，未播放前。

开始播放，未全屏状态。

全屏状态。

12.5 录音

Qt 提供了 QAudioRecorder 类录制音频，继承于 QmediaRecorder 类，音频输入可以使用 QAudioRecorder 或者 QAudioInput 类实现。QAudioRecorder 是高级的类，输入的音频数据直接保存为一个音频文件。而 QAudioInput 则是低层次的实现，从类的名称可以知道它是与输入输

出流有关的，它可以将音频录制的数据写入一个流设备。本小节将介绍使用 QAudioRecorder 录制音频并保存成一个 mp3 文件。并使用 QAudioProbe 类探测音频数据缓冲区里的实时音量大小设计一个实用的录音应用程序。

12.5.1 应用实例

本例设计一个实用的录音界面，界面是笔者原创界面。**本例适用于正点原子 ALPHA 开发板，已经测试。**Windows 与 Ubuntu 下请读者使用 Qt 官方的 audiorecorder 例子自行测试，Windows 系统上的声卡设置比较复杂，不详解，笔者只确保正点原子 I.MX6U ALPHA 开发板正常运行此应用程序。Mini 板没有声卡，请使用 USB 声卡插到正点原子 I.MX6U 开发板进行自行测试!!!

本例目的：录音程序的设计与使用。

例 16_audiorecorder，录音程序（难度：难）。项目路径为 Qt/2/16_audiorecorder。注意本例有用到 qss 样式文件，关于如何添加资源文件与 qss 文件请参考 7.1.3 小节。本例设计一个录音程序，录音功能部分直接参考 Qt 官方的 audiorecorder 例程，界面设计由笔者设计。在 Qt 官方的 audiorecorder 例程里（自行在 Qt 官方的 audiorecorder 例程里打开查看），我们可以看到官方的例程录音设置都是通过 QComboBox 来选择的，当然这个只是一个 Qt 官方的例子，有很大的参考性。如果运用到实际项目里我们需要做一定的修改。如果是面向用户，我们就不需要暴露那么多信息给用户，同时也可以避免用户操作失败等等。所以笔者参考 Qt 官方的例程重新设计了一个录音例程。源码如下，界面效果如 12.5.2 小节。

项目文件 16_audiorecorder 文件第一行添加的代码部分如下。

16_audiorecorder.pro 编程后的代码

```
1  QT       += core gui multimedia
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
13 # You can also make your code fail to compile if it uses deprecated APIs.
14 # In order to do so, uncomment the following line.
15 # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18 SOURCES += \
19     main.cpp \
```

```
20      audiorecorder.cpp
21
22  HEADERS += \
23      audiorecorder.h
24
25  # Default rules for deployment.
26  qnx: target.path = /tmp/$${TARGET}/bin
27  else: unix:!android: target.path = /opt/$${TARGET}/bin
28  !isEmpty(target.path): INSTALLS += target
29
30  RESOURCES += \
31      res.qrc
```

在头文件“audiorecorder.h”具体代码如下。

audiorecorder.h 编程后的代码

```
/*****************************************************************
Copyright (C) 2017 The Qt Company Ltd.
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   16_audiorecorder
* @brief        audiorecorder.h
* @author       Deng Zhimao
* @email        1252699831@qq.com
* @net          www.openedv.com
* @date         2021-05-10
*****************************************************************/
1   #ifndef AUDIORECORDER_H
2   #define AUDIORECORDER_H
3
4   #include <QMainWindow>
5   #include <QListWidget>
6   #include <QVBoxLayout>
7   #include <QPushButton>
8   #include <QLabel>
9   #include <QAudioRecorder>
10  #include <QAudioProbe>
11  #include <QAudioBuffer>
12  #include <QMediaPlaylist>
13  #include <QMediaPlayer>
14  #include <QProgressBar>
15
16  /* 媒体信息结构体 */
17  struct MediaObjectInfo {
18      /* 用于保存视频文件名 */
19      QString fileName;
```

```
20      /* 用于保存视频文件路径 */
21      QString filePath;
22  };
23
24  class AudioRecorder : public QMainWindow
25  {
26      Q_OBJECT
27
28  public:
29      AudioRecorder(QWidget *parent = nullptr);
30      ~AudioRecorder();
31
32  private:
33      /* 布局初始化 */
34      void layoutInit();
35
36      /* 主 Widget*/
37      QWidget *mainWidget;
38
39      /* 录音列表 */
40      QListWidget *listWidget;
41
42      /* 底部的 Widget,用于存放按钮 */
43      QWidget *bottomWidget;
44
45      /* 中间的显示录制时长的 Widget 容器 */
46      QWidget *centerWidget;
47
48      /* 垂直布局 */
49      QVBoxLayout *vBoxLayout;
50
51      /* 录音 Level 布局 */
52      QHBoxLayout *levelHBoxLayout;
53
54      /* 水平布局 */
55      QHBoxLayout *hBoxLayout;
56
57      /* 录音按钮 */
58      QPushButton *recorderBt;
59
60      /* 上一首按钮 */
61      QPushButton *previousBt;
62
```

```
63      /* 下一首按钮 */
64      QPushButton *nextBt;
65
66      /* 删除按钮 */
67      QPushButton *removeBt;
68
69      /* 录音类 */
70      QAudioRecorder *m_audioRecorder = nullptr;
71
72      /* 用于探测缓冲区的 level */
73      QAudioProbe *m_probe = nullptr;
74
75      /* 扫描录音文件 */
76      void scanRecordFiles();
77
78      /* 录音设置容器，保存录音设备的可用信息，
79       * 本例使用默认的信息，即可录音 */
80      QList<QVariant>devicesVar;
81      QList<QVariant>codecsVar;
82      QList<QVariant>containersVar;
83      QList<QVariant>sampleRateVar;
84      QList<QVariant>channelsVar;
85      QList<QVariant>qualityVar;
86      QList<QVariant>bitratesVar;
87
88      /* 媒体播放器，用于播放视频 */
89      QMediaPlayer *recorderPlayer;
90
91      /* 媒体列表 */
92      QMediaPlaylist *mediaPlaylist;
93
94      /* 录音媒体信息存储 */
95      QVector<MediaObjectInfo> mediaObjectInfo;
96
97      /* 用于显示录音时长 */
98      QLabel *countLabel;
99
100     /* 用于显示录音 level，最多四通道 */
101     QProgressBar *progressBar[4];
102
103     /* 清空录音 level */
104     void clearAudioLevels();
```

```
105
106 private slots:
107     /* 点击录音按钮槽函数 */
108     void recorderBtClicked();
109
110     /* 播放列表点击 */
111     void listWidgetCliked(QListWidgetItem*);
112
113     /* 当前媒体状态改变 */
114     void mediaPlayerStateChanged(QMediaPlayer::State);
115
116     /* 媒体列表改变 */
117     void mediaPlaylistCurrentIndexChanged(int);
118
119     /* 当前列表项改变 */
120     void listWidgetCurrentItemChange(QListWidgetItem*,
121                                      QListWidgetItem*);
122
123     /* 上一首按钮点击 */
124     void previousBtClicked();
125
126     /* 下一首按钮点击 */
127     void nextBtClicked();
128
129     /* 删除按钮点击 */
130     void removeBtClicked();
131
132     /* 更新录音时长 */
133     void updateProgress(qint64);
134
135     /* 在列表里显示播放时间 */
136     void recorderPlayerPositionChanged(qint64);
137
138     /* 更新录音 level */
139     void processBuffer(const QAudioBuffer&);
140 };
141 #endif // AUDIORECORDER_H
```

头文件里主要是声明界面所使用的元素及一些槽函数。重点是 QAudioRecorder 与 QAudioProbe 对象的声明。

在源文件“audiorecorder.cpp”具体代码如下。

audiorecorder.cpp 编程后的代码

```
/*****************************************************************
```

```
    Copyright (C) 2017 The Qt Company Ltd.
    Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
    * @projectName   16_audiorecorder
    * @brief        audiorecorder.cpp
    * @author       Deng Zhimao
    * @email        1252699831@qq.com
    * @net          www.openedv.com
    * @date         2021-05-10
    ****************************************************************/
1   #include "audiorecorder.h"
2   #include <QDebug>
3   #include <QUrl>
4   #include <QDateTime>
5   #include <QDir>
6   #include <QCoreApplication>
7
8   static qreal getPeakValue(const QAudioFormat &format);
9   static QVector<qreal> getBufferLevels(const QAudioBuffer &buffer);
10
11  template <class T>
12  static QVector<qreal> getBufferLevels(const T *buffer, int frames, int
channels);
13
14  AudioRecorder::AudioRecorder(QWidget *parent)
15      : QMainWindow(parent)
16  {
17      /* 初始化布局 */
18      layoutInit();
19
20      /* 录制音频的类 */
21      m_audioRecorder = new QAudioRecorder(this);
22
23      /* 用于探测缓冲区的数据 */
24      m_probe = new QAudioProbe(this);
25
26      /* 信号槽连接，更新录音 level 显示 */
27      connect(m_probe, &QAudioProbe::audioBufferProbed,
28              this, &AudioRecorder::processBuffer);
29
30      /* 设置探测的对象 */
31      m_probe->setSource(m_audioRecorder);
32
33      /* 播放器 */
```

```
34      recorderPlayer = new QMediaPlayer(this);
35
36      /* 播放列表 */
37      mediaPlaylist = new QMediaPlaylist(this);
38
39      recorderPlayer->setPlaylist(mediaPlaylist);
40
41      /* 设置播放模式，默认是列表播放 */
42      mediaPlaylist->setPlaybackMode(QMediaPlaylist::CurrentItemOnce);
43
44      /* 扫描本地声卡设备 */
45      devicesVar.append(QVariant(QString()));
46      for (auto &device: m_audioRecorder->audioInputs()) {
47          devicesVar.append(QVariant(device));
48          //qDebug()<<"本地声卡设备: "<<device<<endl;
49      }
50
51      /* 音频编码 */
52      codecsVar.append(QVariant(QString()));
53      for (auto &codecName: m_audioRecorder->supportedAudioCodecs()) {
54          codecsVar.append(QVariant(codecName));
55          //qDebug()<<"音频编码: "<<codecName<<endl;
56      }
57
58      /* 容器/支持的格式 */
59      containersVar.append(QVariant(QString()));
60      for (auto &containerName: m_audioRecorder->supportedContainers())
{
61          containersVar.append(QVariant(containerName));
62          //qDebug()<<"支持的格式: "<<containerName<<endl;
63      }
64
65      /* 采样率 */
66      sampleRateVar.append(QVariant(0));
67      for (int sampleRate: m_audioRecorder->supportedAudioSampleRates())
{
68          sampleRateVar.append(QVariant(sampleRate));
69          //qDebug()<<"采样率: "<<sampleRate<<endl;
70      }
71
72      /* 通道 */
73      channelsVar.append(QVariant(-1));
74      channelsVar.append(QVariant(1));
```

```
75      channelsVar.append(QVariant(2));
76      channelsVar.append(QVariant(4));
77
78      /* 质量 */
79      qualityVar.append(QVariant(int(QMultimedia::LowQuality)));
80      qualityVar.append(QVariant(int(QMultimedia::NormalQuality)));
81      qualityVar.append(QVariant(int(QMultimedia::HighQuality)));
82
83      /* 比特率 */
84      bitratesVar.append(QVariant(0));
85      bitratesVar.append(QVariant(32000));
86      bitratesVar.append(QVariant(64000));
87      bitratesVar.append(QVariant(96000));
88      bitratesVar.append(QVariant(128000));
89
90      /* 初始化时扫描已经录制的录音 mp3 文件 */
91      scanRecordFiles();
92
93      /* 录音类信号槽连接 */
94      connect(m_audioRecorder, &QAudioRecorder::durationChanged,
95              this, &AudioRecorder::updateProgress);
96
97      /* 列表信号槽连接 */
98      connect(listWidget, SIGNAL(itemClicked(QListWidgetItem*)),
99              this, SLOT(listWidgetCliked(QListWidgetItem*)));
100     connect(listWidget, SIGNAL(currentItemChanged(QListWidgetItem*,
101                                                   QListWidgetItem*)),
102             this, SLOT(listWidgetCurrentItemChange(QListWidgetItem*,
103                                                    QListWidgetItem*)));
104
105     /* 媒体连接信号槽 */
106     connect(recorderPlayer,
107             SIGNAL(stateChanged(QMediaPlayer::State)),
108             this,
109             SLOT(mediaPlayerStateChanged(QMediaPlayer::State)));
110     connect(mediaPlaylist,
111             SIGNAL(currentIndexChanged(int)),
112             this,
113             SLOT(mediaPlaylistCurrentIndexChanged(int)));
114     connect(recorderPlayer, SIGNAL(positionChanged(qint64)),
115             this,
116             SLOT(recorderPlayerPositionChanged(qint64)));
117
```

```
118     /* 按钮 */
119     connect(recorderBt, SIGNAL(clicked()), this,
SLOT(recorderBtClicked()));
120     connect(nextBt, SIGNAL(clicked()), this, SLOT(nextBtClicked()));
121     connect(previousBt, SIGNAL(clicked()), this,
SLOT(previousBtClicked()));
122     connect(removeBt, SIGNAL(clicked()), this,
SLOT(removeBtClicked()));
123 }
124
125 AudioRecorder::~AudioRecorder()
126 {
127 }
128
129 void AudioRecorder::layoutInit()
130 {
131     this->setGeometry(0, 0, 800, 480);
132
133     mainWidget = new QWidget();
134     setCentralWidget(mainWidget);
135
136     vBoxLayout = new QVBoxLayout();
137     bottomWidget = new QWidget();
138     listWidget = new QListWidget();
139     listWidget->setFocusPolicy(Qt::NoFocus);
140     listWidget->setVerticalScrollBarPolicy(
141                 Qt::ScrollBarAlwaysOff);
142     listWidget->setHorizontalScrollBarPolicy(
143                 Qt::ScrollBarAlwaysOff);
144
145     /* 垂直布局 */
146     vBoxLayout->addWidget(listWidget);
147     vBoxLayout->addWidget(bottomWidget);
148     vBoxLayout->setContentsMargins(0, 0, 0, 0);
149     mainWidget->setLayout(vBoxLayout);
150
151     bottomWidget->setMinimumHeight(80);
152     bottomWidget->setMaximumHeight(80);
153     bottomWidget->setStyleSheet("background:#cccccc");
154
155     /* 水平布局 */
156     hBoxLayout = new QHBoxLayout();
157
158     /* 按钮，录音、上一首、下一首、删除项按钮 */
```

```
159     recorderBt = new QPushButton();
160     previousBt = new QPushButton();
161     nextBt = new QPushButton();
162     removeBt = new QPushButton();
163
164     recorderBt->setCheckable(true);
165     recorderBt->setObjectName("recorderBt");
166     recorderBt->setFocusPolicy(Qt::NoFocus);
167     recorderBt->setMaximumSize(60, 60);
168     recorderBt->setMinimumSize(60, 60);
169
170     hBoxLayout->setContentsMargins(0, 0, 0, 0);
171
172     bottomWidget->setLayout(hBoxLayout);
173     hBoxLayout->addWidget(recorderBt);
174     hBoxLayout->addWidget(previousBt);
175     hBoxLayout->addWidget(nextBt);
176     hBoxLayout->addWidget(removeBt);
177
178     nextBt->setMaximumSize(50, 50);
179     removeBt->setMaximumSize(50, 50);
180     previousBt->setMaximumSize(50, 50);
181
182     previousBt->setObjectName("previousBt");
183     removeBt->setObjectName("removeBt");
184     nextBt->setObjectName("nextBt");
185
186     previousBt->setFocusPolicy(Qt::NoFocus);
187     removeBt->setFocusPolicy(Qt::NoFocus);
188     nextBt->setFocusPolicy(Qt::NoFocus);
189
190     /* 显示录音时长与录音 Level */
191     centerWidget = new QWidget(this);
192     centerWidget->setGeometry(width()/ 2 - 150,
193                               height() /2 - 100,
194                               300,
195                               200);
196     centerWidget->setStyleSheet("QWidget { background:#8823242a;"
197                                 "border-radius:10px}");
198     countLabel = new QLabel(centerWidget);
199     countLabel->setGeometry(0,
200                             0,
201                             300,
202                             50);
```

```
203     countLabel->setStyleSheet("QLabel {font-size:
30px;color:#eeeeee;"
204                               "font: blod;background:transparent}");
205     countLabel->setAlignment(Qt::AlignCenter);
206     levelHBoxLayout = new QHBoxLayout();
207
208     for (int i = 0; i < 4; i++) {
209         progressBar[i] = new QProgressBar();
210         progressBar[i]->setOrientation(Qt::Vertical);
211         progressBar[i]->setRange(0, 100);
212         progressBar[i]->setVisible(false);
213         progressBar[i]->setMaximumWidth(centralWidget()->width());
214         levelHBoxLayout->addWidget(progressBar[i]);
215         levelHBoxLayout->setContentsMargins(5, 50, 5, 5);
216         progressBar[i]->setStyleSheet("QWidget
{ background:#22eeeeee;"
217                                       "border-radius:0px}");
218     }
219     centerWidget->setLayout(levelHBoxLayout);
220     centerWidget->hide();
221     countLabel->raise();
222
223
224 }
225
226 void AudioRecorder::recorderBtClicked()
227 {
228     /* 录音前停止正在播放的媒体 */
229     if (recorderPlayer->state() != QMediaPlayer::StoppedState)
230         recorderPlayer->stop();
231     /* 如果录音已经停止，则开始录音 */
232     if (m_audioRecorder->state() == QMediaRecorder::StoppedState) {
233         /* 设置默认的录音设备 */
234         m_audioRecorder->setAudioInput(devicesVar.at(0).toString());
235
236         /* 下面的是录音设置，都是选择默认,可根据录音可用项，自行修改 */
237         QAudioEncoderSettings settings;
238         settings.setCodec(codecsVar.at(0).toString());
239         settings.setSampleRate(sampleRateVar[0].toInt());
240         settings.setBitRate(bitratesVar[0].toInt());
241         settings.setChannelCount(channelsVar[0].toInt());
242         settings.setQuality(QMultimedia::EncodingQuality(
243                                 qualityVar[0].toInt()));
244         /* 以恒定的质量录制，可选恒定的比特率 */
```

```
245
settings.setEncodingMode(QMultimedia::ConstantQualityEncoding);
246         QString container = containersVar.at(0).toString();
247         m_audioRecorder->setEncodingSettings(settings,
248                                              QVideoEncoderSettings(),
249                                              container);
250         m_audioRecorder->setOutputLocation(
251                     QUrl::fromLocalFile(tr("./Sounds/%1.mp3")
252                                         .arg(QDateTime::currentDateTime()
253                                              .toString())));
254         /* 开始录音 */
255         m_audioRecorder->record();
256         /* 显示录制时长标签 */
257         countLabel->clear();
258         centerWidget->show();
259     } else {
260         /* 停止录音 */
261         m_audioRecorder->stop();
262         /* 重设录音 level */
263         clearAudioLevels();
264         /* 隐藏录制时长标签 */
265         centerWidget->hide();
266         /* 重新扫描录音文件 */
267         scanRecordFiles();
268     }
269 }
270
271 void AudioRecorder::scanRecordFiles()
272 {
273     mediaPlaylist->clear();
274     listWidget->clear();
275     mediaObjectInfo.clear();
276     /* 录音文件保存在当前 Sounds 文件夹下 */
277     QDir dir(QCoreApplication::applicationDirPath()
278             + "/Sounds");
279     QDir dirbsolutePath(dir.absolutePath());
280
281     /* 如果文件夹不存在，则创建一个 */
282     if(!dirbsolutePath.exists())
283         dirbsolutePath.mkdir(dirbsolutePath.absolutePath());
284
285     /* 定义过滤器 */
286     QStringList filter;
```

```
287     /* 包含所有 xx 后缀的文件 */
288     filter<<"*.mp3";
289     /* 获取该目录下的所有文件 */
290     QFileInfoList files =
291             dirbsolutePath.entryInfoList(filter, QDir::Files);
292     /* 遍历 */
293     for (int i = 0; i < files.count(); i++) {
294         MediaObjectInfo info;
295         /* 使用 utf-8 编码 */
296         info.fileName = QString::fromUtf8(files.at(i)
297                                           .fileName()
298                                           .toUtf8()
299                                           .data());
300         info.filePath = QString::fromUtf8(files.at(i)
301                                           .filePath()
302                                           .toUtf8()
303                                           .data());
304         /* 媒体列表添加音频 */
305         if (mediaPlaylist->addMedia(
306                 QUrl::fromLocalFile(info.filePath))) {
307             /* 添加到容器数组里储存 */
308             mediaObjectInfo.append(info);
309             /* 添加音频名字至列表 */
310             listWidget->addItem(
311                     new QListWidgetItem(QIcon(":/icons/play.png"),
312                                         info.fileName));
313         } else {
314             qDebug()<<
315                     mediaPlaylist->errorString()
316                     .toUtf8().data()
317                   << endl;
318             qDebug()<< "  Error number:"
319                   << mediaPlaylist->error()
320                   << endl;
321         }
322     }
323 }
324
325 void AudioRecorder::listWidgetCliked(QListWidgetItem *item)
326 {
327     /* item->setIcon 为设置列表里的图标状态 */
328     for (int i = 0; i < listWidget->count(); i++) {
329         listWidget->item(i)->setIcon(QIcon(":/icons/play.png"));
```

```
330     }
331
332     if (recorderPlayer->state() != QMediaPlayer::PlayingState) {
333         recorderPlayer->play();
334         item->setIcon(QIcon(":/icons/pause.png"));
335     } else {
336         recorderPlayer->pause();
337         item->setIcon(QIcon(":/icons/play.png"));
338     }
339 }
340
341 void AudioRecorder::listWidgetCurrentItemChange(
342         QListWidgetItem *currentItem,
343         QListWidgetItem *previousItem)
344 {
345     if (mediaPlaylist->mediaCount() == 0)
346         return;
347
348     if (listWidget->row(previousItem) != -1)
349         previousItem->setText(mediaObjectInfo
350                               .at(listWidget->row(previousItem))
351                               .fileName);
352
353     /* 先暂停播放媒体 */
354     if (recorderPlayer->state() == QMediaPlayer::PlayingState)
355         recorderPlayer->pause();
356
357     /* 设置当前媒体 */
358     mediaPlaylist->
359             setCurrentIndex(listWidget->row(currentItem));
360 }
361
362 void AudioRecorder::mediaPlayerStateChanged(
363         QMediaPlayer::State
364         state)
365 {
366     for (int i = 0; i < listWidget->count(); i++) {
367         listWidget->item(i)
368                 ->setIcon(QIcon(":/icons/play.png"));
369     }
370
371     /* 获取当前项，根据当前媒体的状态，然后设置不同的图标 */
372     if (mediaPlaylist->currentIndex() == -1)
373         return;
```

```
374     QListWidgetItem *item = listWidget->item(
375                 mediaPlaylist->currentIndex());
376
377     switch (state) {
378     case QMediaPlayer::PausedState:
379     case QMediaPlayer::PlayingState:
380         item->setIcon(QIcon(":/icons/pause.png"));
381         break;
382     case QMediaPlayer::StoppedState:
383         item->setIcon(QIcon(":/icons/play.png"));
384         break;
385     }
386 }
387
388 void AudioRecorder::mediaPlaylistCurrentIndexChanged(
389         int index)
390 {
391     if (-1 == index)
392         return;
393 }
394
395 void AudioRecorder::previousBtClicked()
396 {
397     /* 上一首操作 */
398     recorderPlayer->stop();
399     int count = listWidget->count();
400     if (0 == count)
401         return;
402     if (listWidget->currentRow() == -1)
403         listWidget->setCurrentRow(0);
404     else {
405         if (listWidget->currentRow() - 1 != -1)
406             listWidget->setCurrentRow(
407                         listWidget->currentRow() - 1);
408         else
409             listWidget->setCurrentRow(listWidget->count() - 1);
410     }
411     mediaPlaylist->setCurrentIndex(listWidget->currentRow());
412     recorderPlayer->play();
413 }
414
415 void AudioRecorder::nextBtClicked()
416 {
417     /* 下一首操作 */
```

```
418     recorderPlayer->stop();
419
420     /* 获取列表的总数目 */
421     int count = listWidget->count();
422
423     /* 如果列表的总数目为 0 则返回 */
424     if (0 == count)
425         return;
426
427     if (listWidget->currentRow() == -1)
428         listWidget->setCurrentRow(0);
429     else {
430         if (listWidget->currentRow() + 1 < listWidget->count())
431             listWidget->setCurrentRow(
432                     listWidget->currentRow() + 1);
433         else
434             listWidget->setCurrentRow(0);
435     }
436     mediaPlaylist->setCurrentIndex(listWidget->currentRow());
437     recorderPlayer->play();
438 }
439
440 void AudioRecorder::removeBtClicked()
441 {
442     int index = listWidget->currentRow();
443     if (index == -1)
444         return;
445
446     /* 移除媒体的项 */
447     mediaPlaylist->removeMedia(index);
448
449     /* 指向要删除的文件 */
450     QFile file(mediaObjectInfo.at(index).filePath);
451
452     /* 移除录音文件 */
453     file.remove();
454
455     /* 删除列表选中的项 */
456     listWidget->takeItem(index);
457
458     /* 删除后设置当前项为删除项的前一个 */
459     if (index - 1 != -1)
460         listWidget->setCurrentRow(index - 1);
```

```
461 }
462
463 void AudioRecorder::updateProgress(qint64 duration)
464 {
465     if (m_audioRecorder->error()
466             != QMediaRecorder::NoError)
467         return;
468
469     /* 显示录制时长 */
470     countLabel->setText(tr("已录制 %1 s")
471                         .arg(duration / 1000));
472 }
473
474 void AudioRecorder::recorderPlayerPositionChanged(
475         qint64 position)
476 {
477     /* 格式化时间 */
478     int p_second  = position / 1000;
479     int p_minute = p_second / 60;
480     p_second %= 60;
481
482     QString mediaPosition;
483     mediaPosition.clear();
484
485     if (p_minute >= 10)
486         mediaPosition = QString::number(p_minute, 10);
487     else
488         mediaPosition = "0" + QString::number(p_minute, 10);
489
490     if (p_second >= 10)
491         mediaPosition = mediaPosition
492                 + ":" + QString::number(p_second, 10);
493     else
494         mediaPosition = mediaPosition
495                 + ":0" + QString::number(p_second, 10);
496
497
498     int d_second  =  recorderPlayer->duration() / 1000;
499     int d_minute = d_second / 60;
500     d_second %= 60;
501
502     QString mediaDuration;
503     mediaDuration.clear();
504
```

```
505     if (d_minute >= 10)
506         mediaDuration = QString::number(d_minute, 10);
507     else
508         mediaDuration = "0" + QString::number(d_minute, 10);
509
510     if (d_second >= 10)
511         mediaDuration = mediaDuration
512                 + ":" + QString::number(d_second, 10);
513     else
514         mediaDuration = mediaDuration
515                 + ":0" + QString::number(d_second, 10);
516
517     QString fileNmae = mediaObjectInfo
518             .at(listWidget->currentRow()).fileName + "\t";
519     /* 显示媒体总长度时间与播放的当前位置 */
520     listWidget->currentItem()->setText(fileNmae
521                                        + mediaPosition
522                                        +"/" + mediaDuration);
523 }
524
525 void AudioRecorder::clearAudioLevels()
526 {
527     for (int i = 0; i < 4; i++)
528         progressBar[i]->setValue(0);
529 }
530
531 // This function returns the maximum possible sample value for a given
audio format
532 qreal getPeakValue(const QAudioFormat& format)
533 {
534     // Note: Only the most common sample formats are supported
535     if (!format.isValid())
536         return qreal(0);
537
538     if (format.codec() != "audio/pcm")
539         return qreal(0);
540
541     switch (format.sampleType()) {
542     case QAudioFormat::Unknown:
543         break;
544     case QAudioFormat::Float:
545         if (format.sampleSize() != 32) // other sample formats are not
supported
546             return qreal(0);
```

```
547         return qreal(1.00003);
548     case QAudioFormat::SignedInt:
549         if (format.sampleSize() == 32)
550             return qreal(INT_MAX);
551         if (format.sampleSize() == 16)
552             return qreal(SHRT_MAX);
553         if (format.sampleSize() == 8)
554             return qreal(CHAR_MAX);
555         break;
556     case QAudioFormat::UnSignedInt:
557         if (format.sampleSize() == 32)
558             return qreal(UINT_MAX);
559         if (format.sampleSize() == 16)
560             return qreal(USHRT_MAX);
561         if (format.sampleSize() == 8)
562             return qreal(UCHAR_MAX);
563         break;
564     }
565
566     return qreal(0);
567 }
568
569 // returns the audio level for each channel
570 QVector<qreal> getBufferLevels(const QAudioBuffer& buffer)
571 {
572     QVector<qreal> values;
573
574     if (!buffer.format().isValid() || buffer.format().byteOrder() !=
QAudioFormat::LittleEndian)
575         return values;
576
577     if (buffer.format().codec() != "audio/pcm")
578         return values;
579
580     int channelCount = buffer.format().channelCount();
581     values.fill(0, channelCount);
582     qreal peak_value = getPeakValue(buffer.format());
583     if (qFuzzyCompare(peak_value, qreal(0)))
584         return values;
585
586     switch (buffer.format().sampleType()) {
587     case QAudioFormat::Unknown:
588     case QAudioFormat::UnSignedInt:
589         if (buffer.format().sampleSize() == 32)
```

```
590             values = getBufferLevels(buffer.constData<quint32>(),
buffer.frameCount(), channelCount);
591         if (buffer.format().sampleSize() == 16)
592             values = getBufferLevels(buffer.constData<quint16>(),
buffer.frameCount(), channelCount);
593         if (buffer.format().sampleSize() == 8)
594             values = getBufferLevels(buffer.constData<quint8>(),
buffer.frameCount(), channelCount);
595         for (int i = 0; i < values.size(); ++i)
596             values[i] = qAbs(values.at(i) - peak_value / 2) / (peak_value
/ 2);
597         break;
598     case QAudioFormat::Float:
599         if (buffer.format().sampleSize() == 32) {
600             values = getBufferLevels(buffer.constData<float>(),
buffer.frameCount(), channelCount);
601             for (int i = 0; i < values.size(); ++i)
602                 values[i] /= peak_value;
603         }
604         break;
605     case QAudioFormat::SignedInt:
606         if (buffer.format().sampleSize() == 32)
607             values = getBufferLevels(buffer.constData<qint32>(),
buffer.frameCount(), channelCount);
608         if (buffer.format().sampleSize() == 16)
609             values = getBufferLevels(buffer.constData<qint16>(),
buffer.frameCount(), channelCount);
610         if (buffer.format().sampleSize() == 8)
611             values = getBufferLevels(buffer.constData<qint8>(),
buffer.frameCount(), channelCount);
612         for (int i = 0; i < values.size(); ++i)
613             values[i] /= peak_value;
614         break;
615     }
616
617     return values;
618 }
619
620 template <class T>
621 QVector<qreal> getBufferLevels(const T *buffer, int frames, int
channels)
622 {
623     QVector<qreal> max_values;
624     max_values.fill(0, channels);
```

```
625
626     for (int i = 0; i < frames; ++i) {
627         for (int j = 0; j < channels; ++j) {
628             qreal value = qAbs(qreal(buffer[i * channels + j]));
629             if (value > max_values.at(j))
630                 max_values.replace(j, value);
631         }
632     }
633
634     return max_values;
635 }
636
637 void AudioRecorder::processBuffer(const QAudioBuffer& buffer)
638 {
639     /* 根据通道数目需要显示 count 个 level */
640     int count = buffer.format().channelCount();
641     for (int i = 0; i < 4; i++) {
642         if (i < count)
643             progressBar[i]->setVisible(true);
644         else
645             progressBar[i]->setVisible(false);
646     }
647
648     /* 设置 level 的值 */
649     QVector<qreal> levels = getBufferLevels(buffer);
650     for (int i = 0; i < levels.count(); ++i)
651         progressBar[i]->setValue(levels.at(i) * 100);
652 }
```

布局与播放录音部分的代码笔者不再解释，与音乐播放器和视频播放器的原理一样。只是换了个样式而已。

第 21 行，初始化录音类对象，m_audioRecorder。录音的功能全靠这个类了，要完成录音的工作，我们只需要关注这个类。剩下的都是其他功能的实现。

第 44~88 行，本例将录音设置的参数，参数可以从 Qt 提供的 API 里如 supportedAudioCodecs（）表示可用支持的编码方式，详细请参考代码，全部使用 QVariant 容器来储存。这样可以不必要暴露太多接口给用户修改，以免出错。

第 222~269，是录音功能的重要代码，一般是通过 QAudioEncoderSettings 来设置输入音频设置，主要是**编码格式、采样率、通道数、音频质量**等设置（音频格式 Qt 提供了如 setCodes（）等方式可以直接设置，音频相关知识不探讨，我们只需要知道这个流程即可。）。这些参考 Qt 官方的 audiorecorder 例程，使用默认的设置，也就是 Default 项，Qt 自动选择系统音频输入设备输入，同时自动确定底层的采样参数等。在 Linux 里，Qt 扫描声卡的设备项很多，让用户选择可能会导致出错。所以笔者测试使用默认的设置，即可录音，无需用户自行选择和修改。

同时设置了录音保存的文件名为 xx.mp3。根据系统时间命名录音文件，如果不指定录音文件名，在 Linux 下一般保存为 clip_0001.mov，clip_0002.mov 等录音文件（mov 格式为苹果操作系统常用音视频格式）。Windows 一般保存为 clip_0001.wav 格式文件。

设置完成录音项后，使用 QAudioRecorder 的 record()、pause()和 stop()函数即可完成录音。本例没有使用 pause()也就是录音暂停，根据情景无需录音暂停。record()和 stop()是开始录音和录音停止。

第 23~28 行，QAudioProbe 类型的对象用于探测缓冲区的数据。setSource()是指定探测的对象。

第 525~652 行，这部分代码是直接拷贝 Qt 官方的代码进行修改，代码比较复杂（闲时自行理解），我们只需要知道它是从缓冲区里获取通道数与音频的实时音量。

整个代码主要完成录音的功能是 QAudioRecorder、QAudioEncoderSettings 和 QAudioProbe 类。其他代码可以没有也可以完成本例录音的功能。QAudioRecorder 负责录音，QAudioProbe 类负责获取通道数，与输入的实时音量。只要掌握了这两个类，设计一个好看的录音应用界面不在话下。

main.cpp 内容如下，主要是加载 qss 样式文件。

```
1  #include "audiorecorder.h"
2
3  #include <QApplication>
4  #include <QFile>
5
6  int main(int argc, char *argv[])
7  {
8      QApplication a(argc, argv);
9      /* 指定文件 */
10     QFile file(":/style.qss");
11
12     /* 判断文件是否存在 */
13     if (file.exists() ) {
14         /* 以只读的方式打开 */
15         file.open(QFile::ReadOnly);
16         /* 以字符串的方式保存读出的结果 */
17         QString styleSheet = QLatin1String(file.readAll());
18         /* 设置全局样式 */
19         qApp->setStyleSheet(styleSheet);
20         /* 关闭文件 */
21         file.close();
22     }
23
24     AudioRecorder w;
25     w.show();
```

```
26      return a.exec();
27  }
```

style.qss 样式文件如下。素材已经在源码处提供。注意下面的 style.qss 不能有注释！

```
1   QTabBar:tab {
2   height:0; width:0;
3   }
4
5   QWidget {
6   background:#e6e6e6;
7   }
8
9   QListWidget {
10  border:none;
11  }
12
13  QPushButton#recorderBt {
14  border-image:url(:/icons/recorder_stop1.png);
15  background:transparent;
16  }
17
18  QPushButton#recorderBt:hover {
19  border-image:url(:/icons/recorder_stop2.png);
20  }
21
22  QPushButton#recorderBt:checked {
23  border-image:url(:/icons/recorder_start1.png);
24  }
25
26  QPushButton#recorderBt:checked:hover {
27  border-image:url(:/icons/recorder_start2.png);
28  }
29
30  QListWidget {
31  color:black;
32  font-size: 20px;
33  border:none;
34  icon-size:40px;
35  }
36
37  QListWidget:item:active {
38  background: transparent;
39  }
40
41  QListWidget:item {
```

```
42  background: transparent;
43  height:60;
44  }
45
46  QListWidget:item:selected {
47  color:red;
48  background: transparent;
49  }
50
51  QListWidget:item:hover {
52  background: transparent;
53  color:red;
54  border:none;
55  }
56
57  QPushButton#nextBt {
58  border-image:url(:/icons/btn_next1.png);
59  }
60
61  QPushButton#nextBt:hover {
62  border-image:url(:/icons/btn_next2.png);
63  }
64
65  QPushButton#previousBt {
66  border-image:url(:/icons/btn_previous1.png);
67  }
68
69  QPushButton#previousBt:hover {
70  border-image:url(:/icons/btn_previous2.png);
71  }
72
73  QPushButton#removeBt {
74  border-image:url(:/icons/remove1.png);
75  }
76
77  QPushButton#removeBt:hover {
78  border-image:url(:/icons/remove2.png);
79  }
80
81  QProgressBar::chunk {
82  background-color: #f6f6f6;
83  height: 8px;
84  margin: 0.5px;
85  border-radius:0px;
```

```
86  }
87
88  QProgressBar {
89  color:transparent;
90  }
```

12.5.2 程序运行效果

本例适用于正点原子 I.MX6U ALPHA 开发板!请使用正点原子 **I.MX6U 的出厂系统**进行测试!

请使用正点原子的 I.MX6U 的出厂时的系统测试!

请使用正点原子的 I.MX6U 的出厂时的系统测试!

请使用正点原子的 I.MX6U 的出厂时的系统测试!

重要的事情是说三遍!

开始录音前，需要根据正点原子 I.MX6U 用户快速体验手册，第 3.15 小节进行测试板子的录音功能。确保能正常录音，再交叉编译此 Qt 应用程序到开发板上运行。如何交叉编译 Qt 应用程序到开发板，请看【正点原子】I.MX6U 出厂系统 Qt 交叉编译环境搭建 V1.x 版本。

在正点原子 I.MX6U 开发板上运行此录音程序，需要先配置是麦克风（板子上的麦头）或者是 Line_in 输入方式。

如果是麦头录音，则在板子上运行开启麦头录音的脚本。

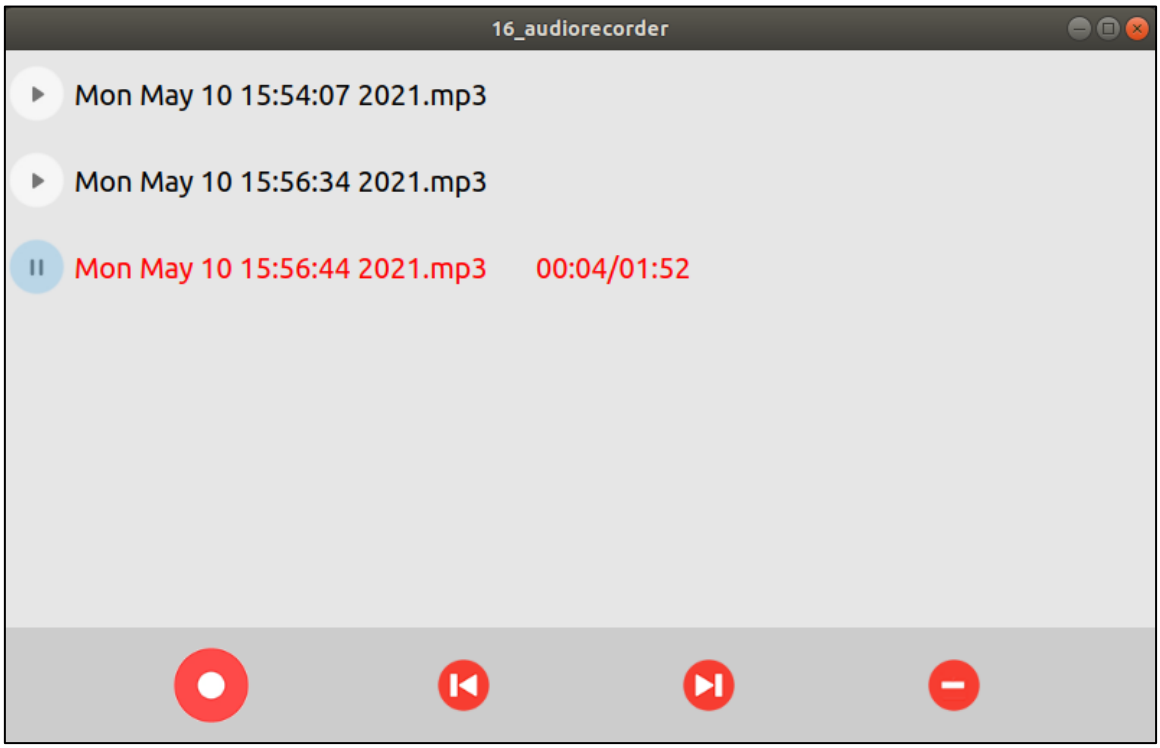
```
/home/root/shell/audio/mic_in_config.sh
```

如果是 Line_in 的方式录音，请使用一条 3.5mm 的两头公头的音频线，一头对板子上 Line_in 接口。另一头连接手机或者电脑的音频输出设备。手机或者电脑开始播放音乐，音乐尽量调高！执行下面的脚本，开启板子以 line_in 的方式录音。

```
/home/root/shell/audio/line_in_config.sh
```

点击左下角的录音按钮开始录音（Ubuntu 上模拟效果图）。可以看到两个通道的实时音量柱状图，用于显示实时音量输入的大小。

开如播放录音。（Ubuntu 上模拟效果图）。



第十三章 数据库

数据库是什么？简易言之，就是保存数据的文件。可以存储大量数据，包括插入数据、更新数据、截取数据等。用专业术语来说，数据库是“按照数据结构来组织、存储和管理数据的仓库”。是一个长期存储在计算机内的、有组织的、可共享的、统一管理的大量数据的集合。

什么时候需要数据库？在嵌入式里，存储大量数据，或者记录数据，就需要用到数据库。举个简单的例子，比如手机的闹钟就使用到了数据库，我们设置的闹钟数据将会保存到数据库里，闹钟程序运行时会从数据库里读取出上次保存的闹钟数据。如果没有数据库，则闹钟程序关机了数据不保存在物理储存设备里，下次运行闹钟时就没有上次设置的闹钟数据，这显然是不合理的。所以我们需要用到数据库。

本章认为读者已经基本了解数据库，已经对数据库有一定的认识，如果没有对数据库了解，请自行学习，毕竟本书是讲 Qt 的，不是讲数据库，数据库知识很多，而我们只是讲解 Qt 怎么去用数据库，对数据库的简单操作！目的就是在 Qt 里使用数据库！

想要在项目中使用 Qt SQL 模块，需要在项目配置文件里添加如下语句。

```
QT          += core gui sql
```

13.1 Qt SQL 简介

Qt SQL 模块为数据库提供了编程支持，Qt 支持很多种常见的数据库，如 MySQL、Oracle、MS SQL Server、SQLite 等。Qt SQL 模块里包含了很多个类，可以轻松实现数据库的连接、执行 SQL 语句，获取数据库里的数据与界面显示等功能，一般数据与界面之间会采用 Model/View 架构，从而很方便的显示数据界面和操作数据库。

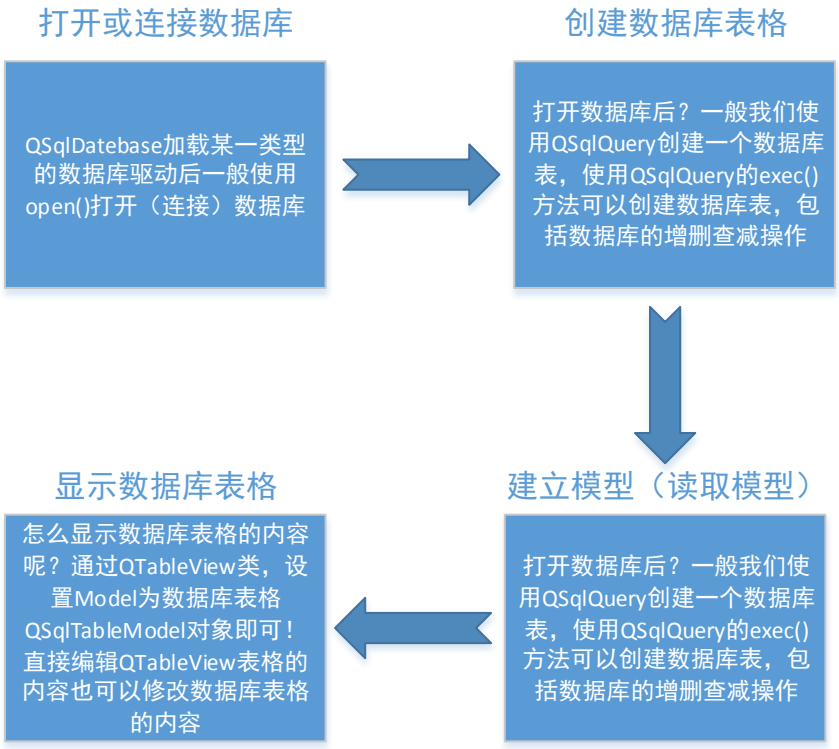
在嵌入式里，一般常用的数据库就是 Sqlite3。SQLite 是非常小的，是轻量级的，完全配置时小于 400KiB，省略可选功能配置时小于 250KiB。SQLite 是一个进程内的库，实现了自给自足的、无服务器的、零配置的、事务性的 SQL 数据库引擎。它是一个零配置的数据库，这意味着与其他数据库不一样，您不需要在系统中配置。就像其他数据库，SQLite 引擎不是一个独立的进程，可以按应用程序需求进行静态或动态连接。SQLite 可以直接访问其存储文件。

本章主要对 Sqlite3 进行实验。需要用其他数据库的请自行学习，在我们正点原子里 Linux 开发板里就是用 sqlite3，文件系统里不提供其他数据库类型。嵌入式一般是用 sqlite3，如需要其他类型数据库，请自行移植与学习！

13.2 应用实例

本章不讲解数据库中的语法，本书认为读者是已经数据库语法有一定了解的了，请知悉！详细声明请看本章前言！本章前言。

Model(模型)，复杂的事情往往可以简单化，Qt 提供了 QSqlDatabase 类用于建立数据库的连接，往往以指定加载的数据库驱动，然后设置数据库的登录参数，如主机地址，用户名、登录密码等。这些都是服务器类型的数据库所需要做的操作。恰好单机型（本地数据库类型）的 Sqlite3 数据库不需要设置登录参数就可以方便的打开数据库进行操作了。在 QSqlDatabase 连接数据库后，用 QSqlTableModel 从数据库里读取出表格模型，然后通过 Qt 的 QTableView 类显示数据库的内容在我们面前。需要对数据库的数据进行修改可以使用 QSqlQuery，或者直接修改 QSqlTableModel 对象，修改里面的模型数据即可！Qt 对数据库的基本操作流程大概是这样子，当然 Qt 提供了很多操作数据库的类，我们只讲解基本的与常用的就已经足够了。下面用个图示来再对上面的操作稍微了解一下。



13.2.1 实用闹钟（非 QTableView 显示）

一般显示数据库表格会使用 QTableView 显示，但是 QTableView 适合专业看数员看且适用于对界面操作要求不高的开发人员看。如果直接用这种表格展示给一般用户看，估计用户看数据得头皮发麻。本例就如本章开头所说，结合数据库开发一个闹钟实例，记录新建的闹钟数据，可以对闹钟进行增、删、改等操作。注意（闹钟不做响铃操作设计。可后期使用本例自行开发，本例主要讲解不使用 QTableView 如何对数据库表格的操作）。本小节开发的闹钟实例很好理解，它与手机的闹钟操作基本一模一样，读者理解起来不会很吃力。本例程序篇幅过长，请注意，我们只需要关注 mainwindow.h 和 mainwindow.cpp 这两个文件即可！其他的.h 和.cpp 文件是笔者为了界面的好看参考了一些博客而设计的程序。主要实现了数字选择器的功能和闹钟开关按钮的功能，因为 Qt C++里它本身没有这种好看的控件，所以得自行设计。由于篇幅过长，数字选择器与闹钟开关的代码不作分析（有兴趣自行分析），我们可以直接将它们当作普通的控件来用即可！重点是 mainwindow.h 和 mainwindow.cpp 里的数据库操作。笔者写这个例子都要好长时间，希望读者不要一口吃个胖老虎，急于求成，本例界面看似简单，可能大多数读者可能对数据库并不是很了解！理解这个例子时，笔者担心是在看天书一样！抓住我们想要理解的重点即可！不必要每句都去理解！

本例目的：了解不使用 QTableView 的情况下，也能把数据表完好展示在用户面前。

例 17_sqlite_alarm，实用闹钟（难度：很难【为什么定义为很难，笔者认为大多数读者对数据库没有一定的了解】）。项目路径为 Qt/2/17_sqlite_alarm。

项目文件 17_sqlite_alarm 文件第一行添加的代码部分如下。

17_sqlite_alarm.pro 编程后的代码

```
1   QT       += core gui sql
2
3   greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5   CONFIG += c++11
6
7   # The following define makes your compiler emit warnings if you use
8   # any Qt feature that has been marked deprecated (the exact warnings
9   # depend on your compiler). Please consult the documentation of the
10  # deprecated API in order to know how to port your code away from it.
11  DEFINES += QT_DEPRECATED_WARNINGS
12
13  # You can also make your code fail to compile if it uses deprecated APIs.
14  # In order to do so, uncomment the following line.
15  # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16  #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18  SOURCES += \
19      main.cpp \
20      mainwindow.cpp \
21      numberpicker.cpp \
22      switchbutton.cpp
23
24  HEADERS += \
25      mainwindow.h \
26      numberpicker.h \
27      switchbutton.h
28
29  # Default rules for deployment.
30  qnx: target.path = /tmp/$${TARGET}/bin
31  else: unix:!android: target.path = /opt/$${TARGET}/bin
32  !isEmpty(target.path): INSTALLS += target
33
34  RESOURCES += \
35      res.qrc
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   17_sqlite_example
* @brief         mainwindow.h
* @author        Deng Zhimao
```

```
    * @email        1252699831@qq.com
    * @net          www.openedv.com
    * @date         2021-05-15
    ************************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QSqlDatabase>
5   #include <QSqlQuery>
6   #include <QMainWindow>
7   #include <QDialog>
8   #include <QHBoxLayout>
9   #include <QVBoxLayout>
10  #include <QPushButton>
11  #include <QListWidget>
12  #include <QLabel>
13  #include <QTime>
14  #include <QSqlTableModel>
15  #include "numberpicker.h"
16  #include "switchbutton.h"
17
18  class NumberPicker;
19  class SwitchButton;
20
21  /* ListWiget 项结构体 */
22  struct ItemObjectInfo {
23      /* 闹钟开关 */
24      SwitchButton *switchButton;
25      /* Widget 容器 */
26      QWidget *widget;
27      /* 水平布局 */
28      QHBoxLayout *hBoxLayout;
29  };
30
31
32  class MainWindow : public QMainWindow
33  {
34      Q_OBJECT
35
36  public:
37      MainWindow(QWidget *parent = nullptr);
38      ~MainWindow();
39
40  private:
```

```
41
42      /* 数据库连接类 */
43      QSqlDatabase sqlDatabase;
44
45      /* 数据库操作模型 */
46      QSqlTableModel *model;
47
48      /* 时针选择器 */
49      NumberPicker *hourPicker;
50
51      /* 分钟选择器 */
52      NumberPicker *minutePicker;
53
54      /* 弹出选择时间对话框 */
55      QDialog *alarmDialog;
56
57      /* 水平布局 */
58      QHBoxLayout *hBoxLayout[3];
59
60      /* 垂直布局 */
61      QVBoxLayout *vBoxLayout[2];
62
63      /* 显示闹钟列表 */
64      QListWidget *listWidget;
65
66      /* 主 Widget */
67      QWidget *mainWidget;
68
69      /* 底部 Wiget */
70      QWidget *bottomWidget;
71
72      /* 弹出对话框布局窗口选择时间容器 */
73      QWidget *timeWidget;
74
75      /* 弹出对话框布局窗口按钮容器 */
76      QWidget *btWidget;
77
78      /* 添加闹钟按钮 */
79      QPushButton *addAlarm;
80
81      /* 确认按钮 */
82      QPushButton *yesButton;
```

```
83
84      /* 取消按钮 */
85      QPushButton *cancelButton;
86
87      /* listWiget 项信息存储 */
88      QVector<ItemObjectInfo> itemObjectInfo;
89
90  private slots:
91      /* 添加闹钟按钮被点击 */
92      void addAlarmClicked();
93
94      /* 列表被点击 */
95      void listWidgetItemClicked(QListWidgetItem *);
96
97      /* 确认按钮被点击 */
98      void yesButtonClicked();
99
100     /* 取消按钮被点击 */
101     void cancelButtonClicked();
102
103     /* 开关按钮点击 */
104     void switchButtonClicked(bool);
105 };
106 #endif // MAINWINDOW_H
```

头文件主要声明布局用的类和数据库，重要关注是 QSqlDatabase 和 QSqlTableModel。这里声明的是全局变量。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   17_sqlite_example
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-05-15
*****************************************************************/
#include "mainwindow.h"
#include <QDebug>
#include <QSqlError>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    /* 设置主窗体的显示位置与大小 */
    this->setGeometry(0, 0, 800, 480);

```

```
11      /* 查看本机可用的数据库驱动 */
12      QStringList drivers = QSqlDatabase::drivers();
13      foreach(QString driver, drivers) {
14          qDebug()<<driver;
15      }
16
17      /* 以 QSQLITE 驱动方式打开或者创建数据库 */
18      sqlDatabase = QSqlDatabase::addDatabase("QSQLITE");
19      sqlDatabase.setDatabaseName("alarm.db");
20      /* 以 open 的方式打开 alarm.db 数据库，则会创建一个 alarm.db */
21      if (!sqlDatabase.open())
22          qDebug()<<"连接数据库错误"<<sqlDatabase.lastError()<<endl;
23      else
24          qDebug()<<"连接数据库成功"<<endl;
25
26      QSqlQuery query(sqlDatabase);
27      /* 使用指令式创建表 */
28      query.exec("create table alarm (id int primary key, time vchar(15),
flag vchar(5))");
29      /* 以指令的方式插入数据 */
30      //query.exec("insert into alarm values(0, '06:00', 'false')");
31
32      model = new QSqlTableModel(this, sqlDatabase);
33
34      /* 模型设置表的名字，需要与数据库的表的名字相同 */
35      model->setTable("alarm");
36
37      /* 如果有修改则同步修改到数据库，
38       * 注意这个规则需要与 tabview 这样的控件才生效，
39       * 因为 tabview 可以直接编辑表里的内容 */
40      model->setEditStrategy(QSqlTableModel::OnFieldChange);
41
42      /* 成功则返回 true，查看数据库里是否有 alarm 这个表格 */
43      model->select();
44
45      /* 如果数据表数据为空，则添加两个闹钟 */
46      if (model->rowCount() == 0) {
47          /* 插入一行 */
48          model->insertRow(model->rowCount());
49          /* 在该行插入数据 */
50          model->setData(model->index(0, 0), 1);
51          model->setData(model->index(0, 1), "06:00");
52          model->setData(model->index(0, 2), "false");
53          /* 插入数据后记得提交 */
54          model->submit();
55
56          /* 再插入一行 */
57          model->insertRow(model->rowCount());
58          model->setData(model->index(1, 0), 2);
59          model->setData(model->index(1, 1), "18:00");
60          model->setData(model->index(1, 2), "true");
61          /* 提交 */
62          model->submit();
63      }
64
65      hourPicker = new NumberPicker(this);
66      hourPicker->setRange(0, 24);
```

```
67
68      minutePicker = new NumberPicker(this);
69      minutePicker->setRange(0, 60);
70
71      /* 标签，用于显示时&分 */
72      QLabel *label[3];
73      label[0] = new QLabel();
74      label[1] = new QLabel();
75      label[2] = new QLabel();
76
77      QFont font;
78      font.setBold(true);
79      font.setPixelSize(10);
80      QPalette pal;
81      pal.setBrush(QPalette::WindowText, QColor(0, 0, 0));
82
83      label[0]->setFont(font);
84      label[1]->setFont(font);
85      label[2]->setFont(font);
86
87      label[0]->setText(" ");
88      label[1]->setText("时");
89      label[2]->setText("分");
90
91      /* 主布局初始化 */
92      listWidget = new QListWidget();
93      mainWidget = new QWidget();
94      bottomWidget = new QWidget();
95      alarmDialog = new QDialog(this);
96      timeWidget = new QWidget();
97      btWidget = new QWidget();
98      addAlarm = new QPushButton();
99      yesButton = new QPushButton();
100     cancelButton = new QPushButton();
101     vBoxLayout[0] = new QVBoxLayout();
102     vBoxLayout[1] = new QVBoxLayout();
103     hBoxLayout[0] = new QHBoxLayout();
104     hBoxLayout[1] = new QHBoxLayout();
105     hBoxLayout[2] = new QHBoxLayout();
106
107     addAlarm->setMaximumSize(84, 84);
108     addAlarm->setObjectName("addAlarm");
109     addAlarm->setMinimumSize(84, 84);
110     bottomWidget->setMinimumHeight(84);
111     bottomWidget->setMaximumHeight(84);
112     yesButton->setText("确认");
113     cancelButton->setText("取消");
114     yesButton->setMaximumSize(100, 50);
115     yesButton->setMinimumSize(100, 50);
116     cancelButton->setMinimumSize(100, 50);
117     cancelButton->setMaximumSize(100, 50);
118     btWidget->setMaximumHeight(70);
119     btWidget->setMinimumHeight(70);
120     alarmDialog->setMinimumSize(300, 300);
121     alarmDialog->setMaximumSize(300, 300);
122     alarmDialog->setModal(true);
123     yesButton->setObjectName("yesButton");
124     cancelButton->setObjectName("cancelButton");
125
```

```
126     /* 主布局 */
127     vBoxLayout[0]->addWidget(listWidget);
128     vBoxLayout[0]->addWidget(bottomWidget);
129     vBoxLayout[0]->setContentsMargins(0, 0, 0, 0);
130
131     mainWidget->setLayout(vBoxLayout[0]);
132
133     setCentralWidget(mainWidget);
134
135     /* 底部按钮布局 */
136     hBoxLayout[0]->addWidget(addAlarm);
137     hBoxLayout[0]->setContentsMargins(0, 0, 0, 0);
138     bottomWidget->setLayout(hBoxLayout[0]);
139
140     /* 对话框布局 */
141     vBoxLayout[1]->addWidget(timeWidget);
142     vBoxLayout[1]->addWidget(btWidget);
143     vBoxLayout[1]->setContentsMargins(0, 0, 0, 0);
144     alarmDialog->setLayout(vBoxLayout[1]);
145
146     hBoxLayout[1]->addWidget(label[0]);
147     hBoxLayout[1]->addWidget(hourPicker);
148     hBoxLayout[1]->addWidget(label[1]);
149     hBoxLayout[1]->addWidget(minutePicker);
150     hBoxLayout[1]->addWidget(label[2]);
151     hBoxLayout[1]->setContentsMargins(0, 0, 0, 0);
152     timeWidget->setLayout(hBoxLayout[1]);
153
154     hBoxLayout[2]->addWidget(yesButton);
155     hBoxLayout[2]->addWidget(cancelButton);
156
157     btWidget->setLayout(hBoxLayout[2]);
158
159     /* 打印出闹钟数据库里的信息 */
160     for (int i = 0; i < model->rowCount(); i++) {
161         for (int j = 0; j < 3; j++) {
162             QModelIndex qindex = model->index(i, j);
163             switch (j) {
164             case 0:
165                 qDebug()<<"第"<<model->data(qindex).toInt()<<"行数据";
166                 break;
167             case 1:
168                 listWidget->addItem(model->data(qindex).toString());
169                 qDebug()<<"闹钟时间为："<<model->data(qindex).toString();
170                 break;
171             case 2:
172                 qDebug()<<"闹钟状态为："
173                       <<model->data(qindex).toString()<<endl;
174                 if (model->data(qindex).toString() != "true")
175                     listWidget->item(i)
176                             ->setTextColor(QColor(22, 22, 22, 60));
177                 else
178                     listWidget->item(i)
179                             ->setTextColor(QColor(22, 22, 22, 225));
180                 break;
181             default:
182                 break;
183             }
```

```
184         }
185     }
186 
187     /* 在列表里添加闹钟开关 */
188     for (int i = 0; i < model->rowCount(); i++) {
189         ItemObjectInfo info;
190         info.widget = new QWidget();
191         info.switchButton = new SwitchButton();
192         info.hBoxLayout = new QHBoxLayout();
193         info.switchButton->setMaximumSize(55, 30);
194         info.switchButton->setMinimumSize(55, 30);
195         info.hBoxLayout->setContentsMargins(0, 0, 0, 0);
196         info.hBoxLayout->setAlignment(Qt::AlignRight);
197         info.hBoxLayout->addWidget(info.switchButton);
198         info.widget->setLayout(info.hBoxLayout);
199         listWidget->setItemWidget(listWidget->item(i),
200                                   info.widget);
201         itemObjectInfo.append(info);
202 
203         /* 连接信号槽 */
204         connect(info.switchButton,
205                 SIGNAL(toggled(bool)),
206                 this,
207                 SLOT(switchButtonClicked(bool)));
208 
209         /* 获取数据库里的闹钟开关状态 */
210         QModelIndex qindex = model->index(i, 2);
211         if (model->data(qindex).toBool())
212             /* 设置列表里的闹钟开关按钮状态 */
213             info.switchButton->setToggle(true);
214     }
215 
216     /* 按钮 */
217     connect(addAlarm, SIGNAL(clicked()), this,
218             SLOT(addAlarmClicked()));
219 
220     connect(yesButton, SIGNAL(clicked()), this,
221             SLOT(yesButtonClicked()));
222 
223     connect(cancelButton, SIGNAL(clicked()), this,
224             SLOT(cancelButtonClicked()));
225 
226     /* 列表 */
227     connect(listWidget,
228             SIGNAL(itemClicked(QListWidgetItem*)),
229             this,
230             SLOT(listWidgetItemClicked(QListWidgetItem*)));
231 }
232 
233 MainWindow::~MainWindow()
234 {
235     /* 关闭数据库 */
236     sqlDatabase.close();
237 }
238 
239 void MainWindow::addAlarmClicked()
240 {
241     /* 选择时间对话框里显示当前系统时间 */
```

```
242     hourPicker->setValue(QTime::currentTime().hour());
243     minutePicker->setValue(QTime::currentTime().minute());
244 
245     /* 取消按钮显示文本为"取消" */
246     cancelButton->setText("取消");
247 
248     /* 如果是点击添加闹钟的按钮，则设置闹钟列表的索引 index 为-1 */
249     listWidget->setCurrentRow(-1);
250 
251     /* 显示对话框 */
252     alarmDialog->show();
253 }
254 
255 void MainWindow::listWidgetItemClicked(QListWidgetItem *item)
256 {
257     /* 从被点击项里获取闹钟数据 */
258     QStringList list =
259 
listWidget->item(listWidget->row(item))->text().split(":");
260 
261     /* 选择时间对话框里显示被选择项的时间 */
262     hourPicker->setValue(list.at(0).toInt());
263     minutePicker->setValue(list.at(1).toInt());
264 
265     /* 取消按钮显示文本为"删除" */
266     cancelButton->setText("删除");
267 
268     /* 显示闹钟选择对话框 */
269     alarmDialog->show();
270 
271     /* 作用使其失去选择 */
272     listWidget->clearSelection();
273 }
274 
275 void MainWindow::yesButtonClicked()
276 {
277     /* 获取数值选择值的数据，转为字符串 */
278     QString hour;
279     QString minute;
280 
281     if (hourPicker->readValue() < 10)
282         hour = "0" + QString::number(hourPicker->readValue()) + ":";
283     else
284         hour = QString::number(hourPicker->readValue()) + ":";
285 
286     if (minutePicker->readValue() < 10)
287         minute = "0" + QString::number(minutePicker->readValue());
288     else
289         minute = QString::number(minutePicker->readValue());
290 
291     /* 如果不是选中闹钟列表的数据 */
292     if (listWidget->currentRow() == -1) {
293         /* 插入一行数据，闹钟时间为选择的闹钟时间 */
294         int row = model->rowCount();
295 
296         /* 插入数据到数据库 */
297         model->insertRow(row);
```

```
298         model->setData(model->index(row, 0), row + 1);
299         model->setData(model->index(row, 1), hour + minute);
300         model->setData(model->index(row, 2), "true");
301         model->submit();
302 
303         /* 添加闹钟到列表 */
304         listWidget->addItem(hour + minute);
305 
306         /* 添加到容器 */
307         ItemObjectInfo info;
308         info.widget = new QWidget();
309         info.switchButton = new SwitchButton();
310         info.hBoxLayout = new QHBoxLayout();
311         info.switchButton->setMaximumSize(55, 30);
312         info.switchButton->setMinimumSize(55, 30);
313         info.hBoxLayout->setContentsMargins(0, 0, 0, 0);
314         info.hBoxLayout->setAlignment(Qt::AlignRight);
315         info.hBoxLayout->addWidget(info.switchButton);
316         info.widget->setLayout(info.hBoxLayout);
317         info.switchButton->setToggle(true);
318 
319         /* 连接信号槽 */
320         connect(info.switchButton, SIGNAL(toggled(bool)), this,
321                 SLOT(switchButtonClicked(bool)));
322 
323         listWidget->setItemWidget(
324                     listWidget->item(listWidget->count() - 1),
325                     info.widget);
326         itemObjectInfo.append(info);
327     } else {
328         /* 修改数据（更新闹钟数据） */
329         int row =  listWidget->currentRow();
330         model->setData(model->index(row, 0), row + 1);
331         model->setData(model->index(row, 1), hour + minute);
332         model->setData(model->index(row, 2), "true");
333         model->submit();
334 
335         /* 设置当前项的闹钟文本 */
336         listWidget->currentItem()->setText(hour + minute);
337     }
338 
339     /* 再确保提交 */
340     if (model->isDirty())
341         model->submitAll();
342 
343     /* 关闭对话框 */
344     alarmDialog->close();
345 }
346 
347 void MainWindow::cancelButtonClicked()
348 {
349     if (cancelButton->text() == "删除") {
350         /* 删除数据库整一行数据 */
351         model->removeRow(listWidget->currentRow());
352         model->submit();
353         /* 执行上面语句 */
354         model->select();
355         itemObjectInfo.remove(listWidget->currentRow());
```

```
356         listWidget->takeItem(listWidget->currentRow());
357     }
358
359     /* 再确保提交 */
360     if (model->isDirty())
361         model->submitAll();
362
363     /* 关闭对话框 */
364     alarmDialog->close();
365 }
366
367
368 /* 当点击闹钟开关时，将闹钟开关状态同步更新到数据库里 */
369 void MainWindow::switchButtonClicked(bool checked)
370 {
371     listWidget->clearSelection();
372
373     SwitchButton *button = (SwitchButton *)sender();
374     for (int i = 0; i < itemObjectInfo.count(); i++) {
375         if (button == itemObjectInfo.at(i).switchButton) {
376             if (checked) {
377                 model->setData(model->index(i, 2), "true");
378                 listWidget->item(i)
379                         ->setTextColor(QColor(22, 22, 22, 225));
380             } else {
381                 model->setData(model->index(i, 2), "false");
382                 listWidget->item(i)
383                         ->setTextColor(QColor(22, 22, 22, 60));
384             }
385
386             model->submit();
387             break;
388         }
389     }
390 }
```

第 5~231 行，数据库的连接、建立模型和界面布局等。界面布局这些不再详细说，这些在前面章节已经讲过很多次。在这部分代码里，我们发现没有用到 QTableView 来展示我们的闹钟数据，原因很简单，因为我们的界面需要适合大众眼光，而不是展示一个表格，应该展示一个闹钟列表，进而笔者设计使用了 QListWidget 这个控件。恰好与手机里的闹钟的列表相似。

12~15 行，查看本地主机可用的数据库驱动。一般 Qt 安装时都会自带 Sqlite3 驱动。注意了，如果本地主机没有可用的数据库，则实验不可操作！不可生搬硬套到其他开发板子测试。所以查看本地主机的数据库操作是调试时候必须的！

18~24 行，添加一个数据库，以 QSQLITE 驱动方式打开或者连接名字为 alarm.db 的数据库文件。数据库存储的形式为一个 alarm.db 文件。

26~28 行，在数据库里创建一个名字为 alarm 的表格。如果已经创建，也会覆盖这个表格名字，但是不会覆盖表格内容。必须先创建表格，才可以对表格的数据进行操作（增删查减等）。

32~43 行，新建模型 model，使用通过 setTable()设置的表中的数据填充模型，使用指定的过滤器和排序条件，如果成功返回 true;否则返回 false。注意:调用 select()将恢复任何未提交的更改，并删除任何插入的列。

46~63 行，笔者在这里判断，如果是刚运行该程序，发现数据库表中没有数据，则默认设置两行闹钟数据，数据 ID 为 1，闹钟时间为 06:00，状态为关；另一条是数据 ID 为 2，闹钟时间为 18:00，状态为开。到这里我们就已经学会在数据库里插入数据，记得插入数据后需要手动执行 submit()函数，表示提交。不提交是不会记录保存到数据库里的。

328~336 行，直接设置数据库里的行数据，即可覆盖该行数据的内容。

347~363 行，model 对象直接移除某一行的数据就完成删除数据库里某行内容。注意可以移除一行或者一列，闹钟数据是以每行记录保存，所以这里是移除一行。移除之后记得提交。

其他的内容都是一些逻辑与界面设计的内容，重点讲解的是 Qt 对数据库操作的步骤。其他内容请根据源码的注释理解即可。或者运行程序去理解本例逻辑。

main.cpp 内容如下，主要是加载 qss 样式文件。

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4   #include <QFile>
5
6   int main(int argc, char *argv[])
7   {
8       QApplication a(argc, argv);
9       /* 指定文件 */
10      QFile file(":/style.qss");
11
12      /* 判断文件是否存在 */
13      if (file.exists() ) {
14          /* 以只读的方式打开 */
15          file.open(QFile::ReadOnly);
16          /* 以字符串的方式保存读出的结果 */
17          QString styleSheet = QLatin1String(file.readAll());
18          /* 设置全局样式 */
19          qApp->setStyleSheet(styleSheet);
20          /* 关闭文件 */
21          file.close();
22      }
23
24      MainWindow w;
25      w.show();
26      return a.exec();
27  }
```

style.qss 样式文件如下。素材已经在源码处提供。注意下面的 style.qss 不能有注释！

```
1   QListWidget {
2   font-size: 30px;
3   outline:none;
```

```
4   }
5
6   QListWidget::item:active {
7   background: transparent;
8   }
9
10  QListWidget::item {
11  height:80;
12  }
13
14  QListWidget::item:selected:hover {
15  background:#22222222;
16  }
17
18  QListWidget::item:selected {
19  background:transparent;
20  color:#ee222222;
21  }
22
23  QPushButton#addAlarm {
24  border-image:url(:/icons/addalarm1.png);
25  background:transparent;
26  outline: none;
27  }
28
29  QPushButton#addAlarm:hover {
30  border-image:url(:/icons/addalarm2.png);
31  }
32
33  QPushButton#yesButton {
34  border: 1px solid #22222222;
35  border-radius: 25px;
36  background:#22222222;
37  outline:none;
38  }
39
40  QPushButton#yesButton:pressed {
41  background:#44222222;
42  color:white;
43  }
44
45  QPushButton#cancelButton {
46  border: 1px solid #22222222;
47  border-radius: 25px;
```

```
48  background:#22222222;
49  outline:none;
50  }
51
52  QPushButton#cancelButton:pressed {
53  background:#44222222;
54  color:white;
55  }
56
57  QScrollBar:vertical {
58  width:30px;
59  background:rgba(255, 255, 255, 100%)
60  }
61
62  QScrollBar::handle:vertical {
63  width:30px;
64  background:rgba(200, 200, 200, 20%);
65  border-radius:15px;
66  }
67
68  QScrollBar::add-line:vertical {
69  width:0px; height:0px;
70  }
71  QScrollBar::sub-line:vertical {
72  width:0px;
73  height:0px;
74  }
75  QScrollBar::handle:vertical:hover {
76  width:30px;
77  background:rgba(200, 200, 200, 80%);
78  border-radius:15px;
79  }
80  QScrollBar::add-page:vertical,QScrollBar::sub-page:vertical {
81  background:rgba(255, 255, 255, 100%)
82  }
```

其中数字选择器与闹钟开关按钮的代码，代码笔者参考了一些博客优化后写成的代码。有兴趣可以细读代码，不作为本章注释讲解的代码。

数字选择器的作用是选择闹钟的时间，效果如下，通过上下滑动可以选择数字作为时钟的时针和分针数据。通过点击对话框确认后存储到数据库里。

数字选择器的头文件“numberpicker.h”代码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   NumberPicker
* @brief        numberpicker.h
* @author       Deng Zhimao
* @email        1252699831@qq.com
* @net          www.openedv.com
* @date         2021-05-14
*****************************************************************/
#ifndef NUMBERPICKER_H
#define NUMBERPICKER_H

#include <QMainWindow>
#include <QPropertyAnimation>

class NumberPicker : public QWidget
{
    Q_OBJECT

    Q_PROPERTY(int deviation READ readDeviation WRITE setDeviation )
public:
    NumberPicker(QWidget *parent = nullptr);
    ~NumberPicker();

    /* 设置最大值与最小值的范围 */
    void setRange(int min, int max);

    /* 读取当前值 */
    int readValue();

protected:
```

```
23      void mousePressEvent(QMouseEvent *);
24
25      void mouseMoveEvent(QMouseEvent *);
26
27      void mouseReleaseEvent(QMouseEvent *);
28
29      void wheelEvent(QWheelEvent *);
30
31      void paintEvent(QPaintEvent *);
32
33  public:
34      /* 描绘数字 */
35      void paintNum(QPainter &painter, int num, int deviation);
36
37      /* 使选中的数字回到屏幕中间 */
38      void homing();
39
40      /* 鼠标移动偏移量，默认为 0 */
41      int readDeviation();
42
43      /* 设置偏移量 */
44      void setDeviation(int n);
45
46      /* 设置字体大小 */
47      void setNumSize(int);
48
49      /* 设置间隔大小 */
50      void setInterval(int);
51
52      /* 设置分格数量，一般设置为 3、5、7... */
53      void setDevide(int);
54
55      /* 设置数字颜色,设置 rgb 的数值 */
56      void setNumberColor(QRgb rgb);
57
58      /* 设置当前值 */
59      void setValue(int value);
60
61  signals:
62
63      void currentValueChanged(int value);
64
65      void deviationChange(int deviation);
```

```
66
67  private:
68      /* 最小值 */
69      int minRange;
70
71      /* 最大值 */
72      int maxRange;
73
74      /* 当前选中的值 */
75      int currentValue;
76
77      /* 鼠标是否按下 */
78      bool isDragging;
79
80      /* 偏移量,记录鼠标按下后移动的垂直距离 */
81      int deviation;
82
83      /* 鼠标按下的垂直位置 */
84      int mouseSrcPos;
85
86      /* 数字大小 */
87      int numSize;
88
89      /* 动画 */
90      QPropertyAnimation *homingAni;
91
92      /* 间隔大小 */
93      int interval;
94
95      /* 分格数量 */
96      int devide;
97
98      /* 数字颜色 */
99      QColor numberColor;
100 };
101 #endif // NUMBERPICKER_H
```

数字选择器的源文件“numberpicker.cpp”代码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   NumberPicker
* @brief         numberpicker.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
```

```
    * @net          www.openedv.com
    * @date         2021-05-14
    ************************************************************************/
1   #include <QMouseEvent>
2   #include <QDebug>
3   #include "numberpicker.h"
4   #include <QPainter>
5
6   NumberPicker::NumberPicker(QWidget *parent) :
7       /* 最小值默认为 0 */
8       minRange(0),
9
10      /* 最大值默认 60 */
11      maxRange(60),
12
13      /* 当前值默认 0 */
14      currentValue(0),
15
16      /* 按下标志位为假 */
17      isDragging(false),
18
19      /* 默认偏移量为 0 */
20      deviation(0),
21
22      /* 数值越大 */
23      numSize(15),
24
25      /* 间隔为 1 */
26      interval(1),
27
28      /* 默认分成 3 格 */
29      devide(3),
30
31      /* 默认颜色黑色 */
32      numberColor(0, 0, 0)
33  {
34      setParent(parent);
35      setMinimumSize(50, 150);
36      homingAni = new QPropertyAnimation(this, "deviation");
37      homingAni->setDuration(300);
38      homingAni->setEasingCurve(QEasingCurve::OutQuad);
39  }
40
41  NumberPicker::~NumberPicker()
42  {
43
44  }
45
46  void NumberPicker::setRange(int min, int max)
47  {
48      minRange = min;
49      maxRange = max;
50      if (currentValue < min) {
51          currentValue = min;
52      }
53      if (currentValue > max) {
54          currentValue = max;
55      }
```

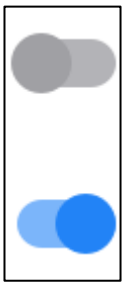
```
56      repaint();
57  }
58
59  int NumberPicker::readValue()
60  {
61      return currentValue;
62  }
63
64  void NumberPicker::mousePressEvent(QMouseEvent *e)
65  {
66      homingAni->stop();
67      isDragging = true;
68      mouseSrcPos = e->pos().y();
69      QWidget::mousePressEvent(e);
70  }
71
72  void NumberPicker::mouseMoveEvent(QMouseEvent *e)
73  {
74      if (isDragging){
75          deviation = e->pos().y() - mouseSrcPos;
76
77          /* 若移动速度过快，则进行限制 */
78          if (deviation > (height() - 1) / devide) {
79              deviation = (height() - 1) / devide;
80          } else if (deviation < -(height() - 1) / devide) {
81              deviation = -( height() - 1) / devide;
82          }
83
84          emit deviationChange(deviation / ((height() - 1) / devide));
85          repaint();
86      }
87  }
88
89  void NumberPicker::mouseReleaseEvent(QMouseEvent *)
90  {
91      if (isDragging) {
92          isDragging = false;
93          homing();
94      }
95  }
96
97  void NumberPicker::wheelEvent(QWheelEvent *e)
98  {
99      if (e->delta() > 0) {
100         deviation = (this->height() - 1) / devide;
101     } else {
102         deviation = -(this->height() - 1) / devide;
103     }
104
105     homing();
106     repaint();
107 }
108
109 void NumberPicker::paintEvent(QPaintEvent *)
110 {
111     QPainter painter(this);
112     painter.setRenderHint(QPainter::Antialiasing, true);
113     int Height = height() - 1;
114
```

```
115     if (deviation >= Height / devide && currentValue > minRange ) {
116         mouseSrcPos += Height / devide;
117         deviation -= Height / devide;
118         currentValue -= interval;
119         /* 负数处理 */
120         if (currentValue < 0)
121             currentValue = maxRange + currentValue;
122     }
123 
124     if (deviation <= -Height / devide && currentValue < maxRange ) {
125         mouseSrcPos -= Height / devide;
126         deviation += Height / devide;
127         currentValue += interval;
128     }
129 
130     if (qAbs(int(currentValue)) >= int(maxRange))
131         currentValue = minRange;
132 
133     paintNum(painter, qAbs(int(currentValue + maxRange) % maxRange),
134             deviation);
135 
136     paintNum(painter,
137             qAbs((currentValue - interval + maxRange) % maxRange),
138             deviation - Height / devide);
139 
140     paintNum(painter,
141             qAbs((currentValue + interval + maxRange) % maxRange),
142             deviation + Height / devide);
143 
144     for (int i = 2; i <= devide / 2; ++i) {
145         if (qAbs(currentValue - interval * i) >= minRange) {
146             paintNum(painter,
147                     qAbs((currentValue - interval * i + maxRange)
148                         % maxRange),
149                     deviation - Height / devide * i);
150         }
151 
152         if (qAbs(currentValue + interval * i) <= maxRange) {
153             paintNum(painter,
154                     qAbs((currentValue + interval * i + maxRange)
155                         % maxRange),
156                     deviation + Height / devide * i);
157         }
158     }
159 }
160 
161 void NumberPicker::paintNum(QPainter &painter, int num, int deviation)
162 {
163     int Width = width() - 1;
164     int Height = height() - 1;
165 
166     /* 偏移量越大，数字越小 */
167     //int size = (Height - qAbs(deviation)) / numSize;
168     int size = (Height - qAbs(deviation)) * numSize / 80;
169     int transparency = 255 - 255 * qAbs(deviation) / Height;
170     int height = Height / devide;
171     int y = Height / 2 + deviation - height / 2;
172 
173     QFont font;
```

```
174     font.setPixelSize(size);
175     painter.setFont(font);
176     painter.setPen(QColor(numberColor.red(),
177                           numberColor.green(),
178                           numberColor.blue(),
179                           transparency));
180 
181     if ( y >= 0 && y + height < Height) {
182         //painter.drawRect(0, y, Width, height);
183         if (num < 10)
184             painter.drawText(QRectF(0, y, Width, height),
185                              Qt::AlignCenter,
186                              "0" + QString::number(num, 'f', 0));
187         else
188             painter.drawText(QRectF(0, y, Width, height),
189                              Qt::AlignCenter,
190                              QString::number(num, 'f', 0));
191     }
192 }
193 
194 void NumberPicker::homing()
195 {
196     if (deviation > height() / 10) {
197         homingAni->setStartValue((height() - 1 ) / 8 - deviation);
198         homingAni->setEndValue(0);
199         currentValue -= interval;
200     } else if (deviation > -height() / 10) {
201         homingAni->setStartValue(deviation);
202         homingAni->setEndValue(0);
203     } else if (deviation < -height() / 10) {
204         homingAni->setStartValue(-(height() - 1) / 8 - deviation);
205         homingAni->setEndValue(0);
206         currentValue += interval;
207     }
208 
209     emit currentValueChanged(currentValue);
210     homingAni->start();
211 }
212 
213 int NumberPicker::readDeviation()
214 {
215     return deviation;
216 }
217 
218 void NumberPicker::setDeviation(int n)
219 {
220     deviation = n;
221     repaint();
222 }
223 
224 void NumberPicker::setNumSize(int size)
225 {
226     numSize = size;
227     repaint();
228 }
229 
230 void NumberPicker::setInterval(int n)
231 {
232     interval = n;
233     repaint();
```

```
234 }
235
236 void NumberPicker::setDevide(int n)
237 {
238     devide = n;
239     repaint();
240 }
241
242 void NumberPicker::setNumberColor(QRgb rgb)
243 {
244     numberColor.setRgb(rgb);
245     repaint();
246 }
247
248 void NumberPicker::setValue(int value)
249 {
250     if (value < minRange || value > maxRange) {
251         qDebug()<<"数值设置必须在"<<minRange
252                 <<"和"<<maxRange<<"之间"<<endl;
253         return;
254     }
255     currentValue = value;
256     repaint();
257 }
```

开关按钮的效果如下。运行时有动画效果，类似 IOS 手册里的开关按钮一样。



开关按钮的头文件“switchbutton.h”代码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   17_sqlite_example
* @brief         switchbutton.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-05-14
*****************************************************************/
#ifndef SWITCHBUTTON_H
#define SWITCHBUTTON_H

#include <QWidget>
#include <QTimer>

class SwitchButton : public QWidget
{
```

```
9       Q_OBJECT
10
11  public:
12      explicit SwitchButton(QWidget *parent = nullptr);
13
14      /* 返回开关状态 - 打开: true 关闭: false */
15      bool isToggled() const;
16
17      /* 设置开关状态 */
18      void setToggle(bool checked);
19
20      /* 设置背景颜色 */
21      void setBackgroundColor(QColor color);
22
23      /* 设置选中颜色 */
24      void setCheckedColor(QColor color);
25
26      /* 设置不可用颜色 */
27      void setDisbaledColor(QColor color);
28
29  protected:
30      /* 绘制开关 */
31      void paintEvent(QPaintEvent *event) Q_DECL_OVERRIDE;
32
33      /* 鼠标按下事件 */
34      void mousePressEvent(QMouseEvent *event) Q_DECL_OVERRIDE;
35
36      /* 鼠标释放事件 - 切换开关状态、发射 toggled()信号 */
37      void mouseReleaseEvent(QMouseEvent *event) Q_DECL_OVERRIDE;
38
39      /* 大小改变事件 */
40      void resizeEvent(QResizeEvent *event) Q_DECL_OVERRIDE;
41
42      /* 缺省大小 */
43      QSize sizeHint() const Q_DECL_OVERRIDE;
44      QSize minimumSizeHint() const Q_DECL_OVERRIDE;
45
46  signals:
47      /* 状态改变时，发射信号 */
48      void toggled(bool checked);
49
50  private slots:
```

```
51      /* 状态切换时，用于产生滑动效果 */
52      void onTimeout();
53
54  private:
55      /* 是否选中 */
56      bool m_bChecked;
57
58      /* 背景颜色 */
59      QColor m_background;
60
61      /* 选中颜色 */
62      QColor m_checkedColor;
63
64      /* 不可用颜色 */
65      QColor m_disabledColor;
66
67      /* 拇指颜色 */
68      QColor m_thumbColor;
69
70      /* 圆角 */
71      qreal m_radius;
72
73      /* x 点坐标 */
74      qreal m_nX;
75
76      /* y 点坐标 */
77      qreal m_nY;
78
79      /* 高度 */
80      qint16 m_nHeight;
81
82      /* 外边距 */
83      qint16 m_nMargin;
84
85      /* 定时器 */
86      QTimer m_timer;
87  };
88  #endif // SWITCHBUTTON_H
```

开关按钮的源文件“switchbutton.cpp”代码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
```

```
    * @projectName   17_sqlite_example
    * @brief         switchbutton.cpp
    * @author        Deng Zhimao
    * @email         1252699831@qq.com
    * @net           www.openedv.com
    * @date          2021-05-14
    ****************************************************************/
1   #include "switchbutton.h"
2
3   #include <QPainter>
4   #include <QMouseEvent>
5
6   SwitchButton::SwitchButton(QWidget *parent)
7       : QWidget(parent),
8         m_bChecked(false),
9         m_background(Qt::gray),
10        m_checkedColor(34, 131, 246),
11        m_disabledColor(190, 190, 190),
12        m_thumbColor(Qt::gray),
13        m_radius(12.5),
14        m_nHeight(16),
15        m_nMargin(3)
16  {
17      /* 鼠标滑过光标形状 - 手型 */
18      setCursor(Qt::PointingHandCursor);
19
20      /* 连接信号槽 */
21      connect(&m_timer, SIGNAL(timeout()),
22              this, SLOT(onTimeout()));
23  }
24
25  /* 绘制开关 */
26  void SwitchButton::paintEvent(QPaintEvent *event)
27  {
28      Q_UNUSED(event)
29
30      QPainter painter(this);
31      painter.setPen(Qt::NoPen);
32      painter.setRenderHint(QPainter::Antialiasing);
33
34      QPainterPath path;
35      QColor background;
36      QColor thumbColor;
37      qreal dOpacity;
```

```
38      /* 可用状态 */
39      if (isEnabled()) {
40          /* 打开状态 */
41          if (m_bChecked) {
42              background = m_checkedColor;
43              thumbColor = m_checkedColor;
44              dOpacity = 0.600;
45              /* 关闭状态 */
46          } else {
47              background = m_background;
48              thumbColor = m_thumbColor;
49              dOpacity = 0.800;
50          }
51          /* 不可用状态 */
52      } else {
53          background = m_background;
54          dOpacity = 0.260;
55          thumbColor = m_disabledColor;
56      }
57      /* 绘制大椭圆 */
58      painter.setBrush(background);
59      painter.setOpacity(dOpacity);
60      path.addRoundedRect(QRectF(m_nMargin,
61                                 m_nMargin, width() - 2 * m_nMargin,
62                                 height() - 2 * m_nMargin),
63                          m_radius, m_radius);
64      painter.drawPath(path.simplified());
65
66      /* 绘制小椭圆 */
67      painter.setBrush(thumbColor);
68      painter.setOpacity(1.0);
69      painter.drawEllipse(QRectF(m_nX - (m_nHeight / 2),
70                                 m_nY - (m_nHeight / 2),
71                                 height(),
72                                 height()));
73  }
74
75  /* 鼠标按下事件 */
76  void SwitchButton::mousePressEvent(QMouseEvent *event)
77  {
78      if (isEnabled()) {
79          if (event->buttons() & Qt::LeftButton) {
80              event->accept();
```

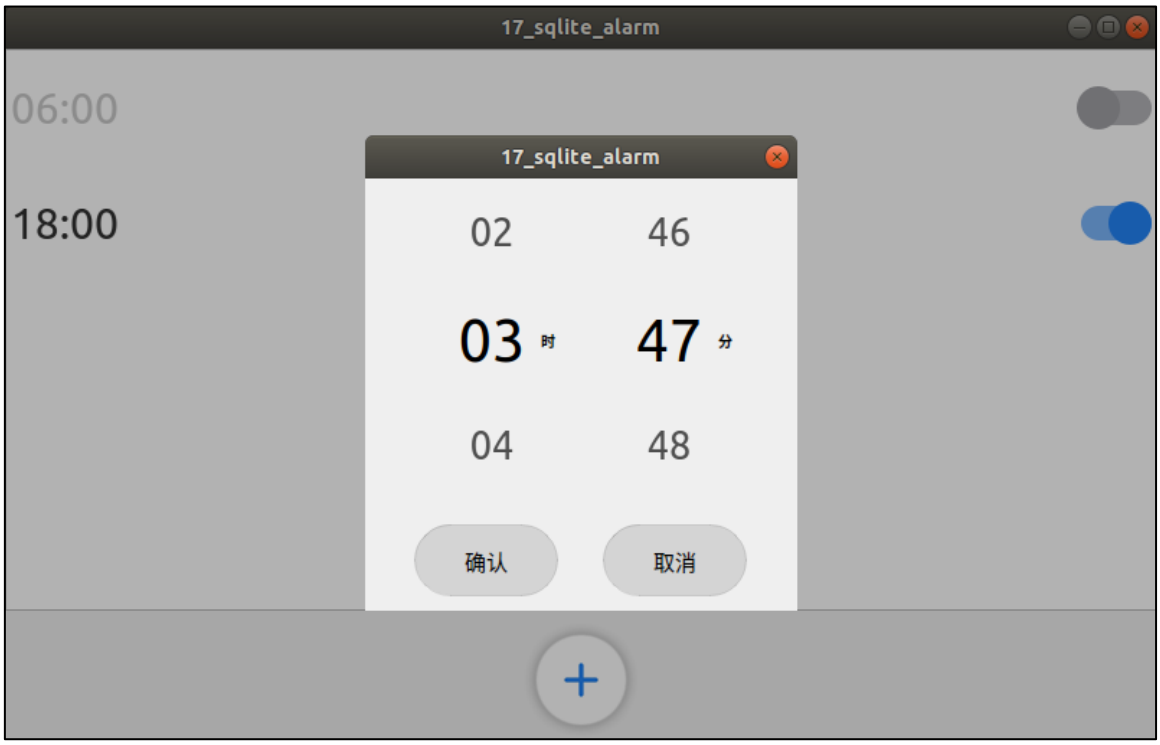
```
81          } else {
82              event->ignore();
83          }
84      }
85  }
86
87  /* 鼠标释放事件 - 切换开关状态、发射 toggled()信号 */
88  void SwitchButton::mouseReleaseEvent(QMouseEvent *event)
89  {
90      if (isEnabled()) {
91          if ((event->type() == QMouseEvent::MouseButtonRelease)
92                  && (event->button() == Qt::LeftButton)) {
93              event->accept();
94              m_bChecked = !m_bChecked;
95              emit toggled(m_bChecked);
96              m_timer.start(10);
97          } else {
98              event->ignore();
99          }
100     }
101 }
102
103 /* 大小改变事件 */
104 void SwitchButton::resizeEvent(QResizeEvent *event)
105 {
106     m_nX = m_nHeight / 2;
107     m_nY = m_nHeight / 2;
108     QWidget::resizeEvent(event);
109 }
110
111 /* 默认大小 */
112 QSize SwitchButton::sizeHint() const
113 {
114     return minimumSizeHint();
115 }
116
117 /* 最小大小 */
118 QSize SwitchButton::minimumSizeHint() const
119 {
120     return QSize(2 * (m_nHeight + m_nMargin),
121                  m_nHeight + 2 * m_nMargin);
122 }
123
124 /* 切换状态 - 滑动 */
```

```
125 void SwitchButton::onTimeout()
126 {
127     if (m_bChecked) {
128         m_nX += 1;
129         if (m_nX >= width() - m_nHeight - m_nHeight / 2 ) {
130             m_timer.stop();
131             m_nX -= 1;
132         }
133     } else {
134         m_nX -= 1;
135         if (m_nX <= m_nHeight / 2) {
136             m_timer.stop();
137             m_nX += 1;
138         }
139     }
140     update();
141 }
142
143 /* 返回开关状态 - 打开: true 关闭: false */
144 bool SwitchButton::isToggled() const
145 {
146     return m_bChecked;
147 }
148
149 /* 设置开关状态 */
150 void SwitchButton::setToggle(bool checked)
151 {
152     m_bChecked = checked;
153     m_timer.start(10);
154 }
155
156 /* 设置背景颜色 */
157 void SwitchButton::setBackgroundColor(QColor color)
158 {
159     m_background = color;
160 }
161
162 /* 设置选中颜色 */
163 void SwitchButton::setCheckedColor(QColor color)
164 {
165     m_checkedColor = color;
166 }
167
168 /* 设置不可用颜色 */
```

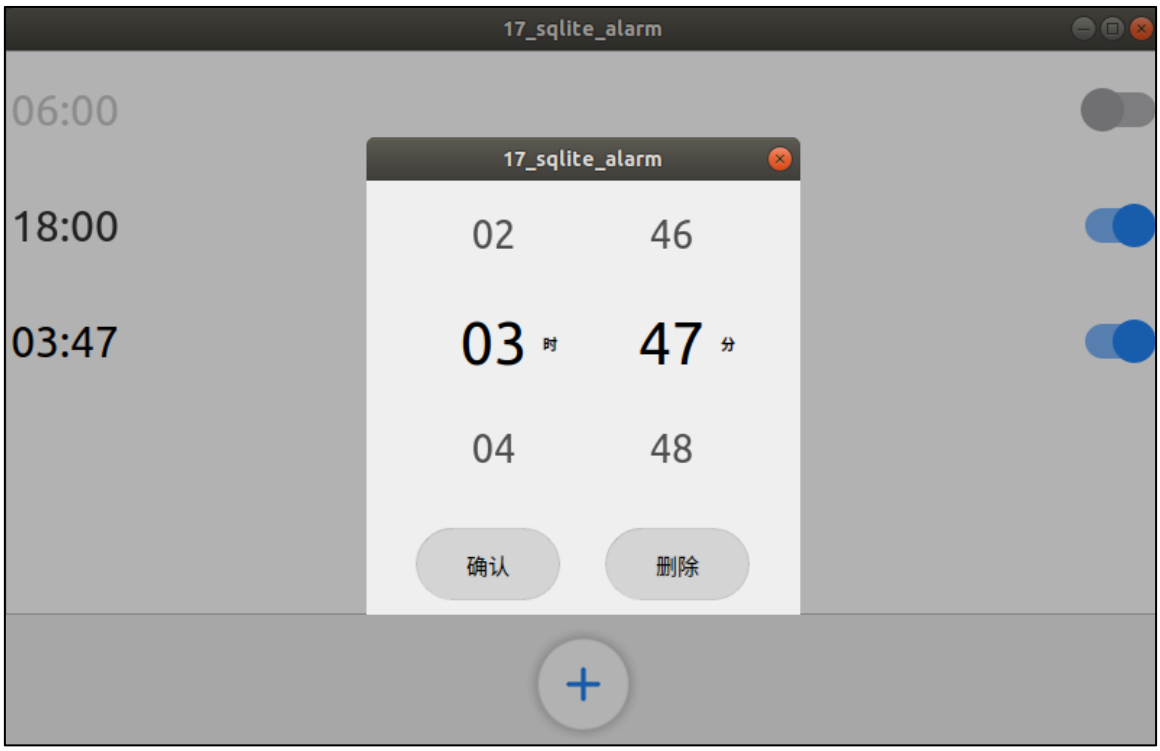
```
169 void SwitchButton::setDisbaledColor(QColor color)
170 {
171     m_disabledColor = color;
172 }
```

13.2.1.1 程序运行效果

点击程序正下方的“+”按钮则开始添加闹钟数据，根据当前系统的时间或者滑动选择闹钟的时间，点击确认即新增一条闹钟数据。



修改或者删除数据，点击要修改闹钟的数据项目，弹出的对话框，可以重新设置闹钟或者删除闹钟。并保存记录到数据库里，下次运行打开时会从数据库里读取保存的闹钟数据。



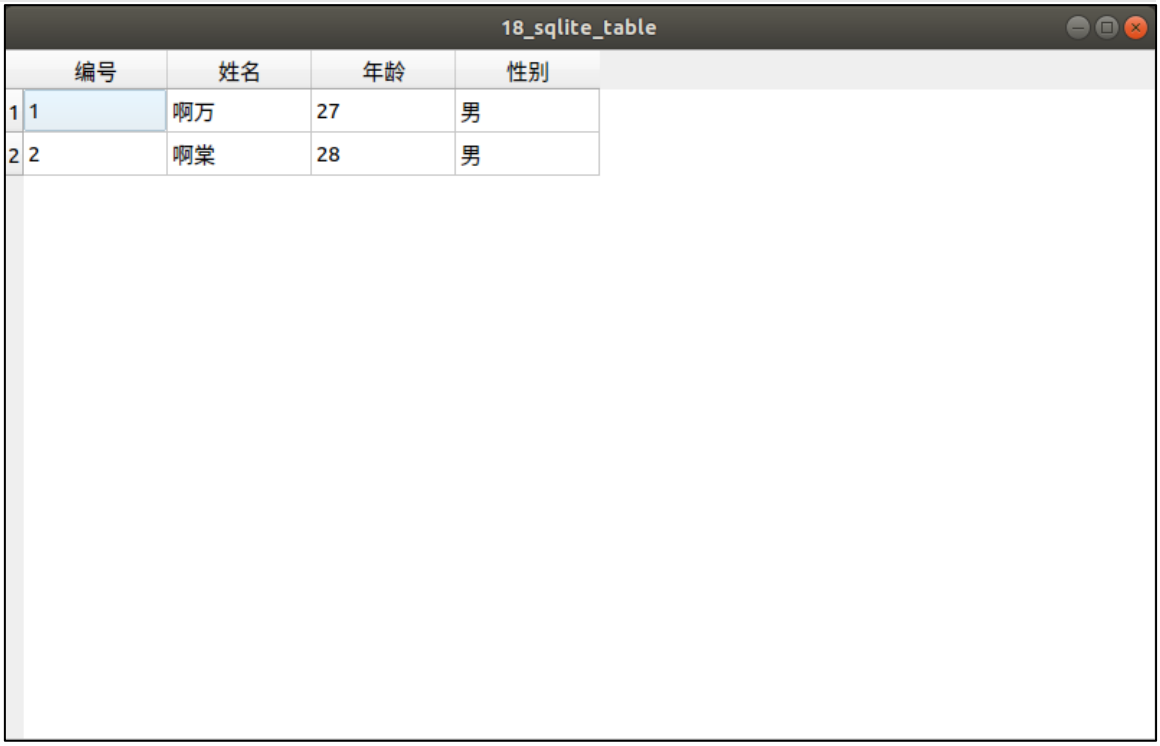
13.2.2 数据库表格（QTableView 显示）

本小节设计一个生活中的例子，使用数据库修改/查询员工的编号、姓名、年龄、性别与照片信息。

本例将数据库的内容显示到 QTableView 上。如果只是简单的显示数据库的内容到 QTableView 上，可以使用下面的方法，此方法 QTableView 上可以看到员工的编号、姓名、年龄、性别信息，同时可以双击表格进行项修改，修改完成将自动保存到数据库里。

```
1    /* 初始化表格模型 */
2    QSqlTableModel *model = new QSqlTableModel(this, sqlDatabase);
3
4    /* 设置要选中的表格名称 */
5    model->setTable("employee");
6    /* 如果有修改则同步修改到数据库，
7     * 注意这个规则需要与 tabview 这样的控件才生效，
8     * 因为 tabview 可以直接编辑表里的内容 */
9    model->setEditStrategy(QSqlTableModel::OnFieldChange);
10   /* 成功则返回 true，查看数据库里是否有 employee 这个表格 */
11   model->select();
12   /* 设置表格的头信息，若不设置则显示数据库里的英文字段头信息 */
13   model->setHeaderData(model->fieldIndex("id"),
14                        Qt::Horizontal, tr("编号"));
15   model->setHeaderData(model->fieldIndex("name"),
16                        Qt::Horizontal, tr("姓名"));
```

```
17    model->setHeaderData(model->fieldIndex("age"),
18                         Qt::Horizontal, tr("年龄"));
19    model->setHeaderData(model->fieldIndex("sex"),
20                         Qt::Horizontal, tr("性别"));
21
22    QTableView *view = new QTableView;
23
24    /* 设置表格的模型为 model */
25    view->setModel(model);
26    /* 不显示图片路径信息行 */
27    view->hideColumn(4);
28    /* 表格居中 */
29    setCentralWidget(view);
30    return;
```



上面的程序可以修改数据库的内容也可以查看。但是看不到员工的照片信息。本例就讲解如何将数据库数据显示到 QTableView 上，及查看选择的员工项的全部信息。介绍 Qt 如何使用数据库存储照片的信息。我们知道数据库类型有个 BLOB 数据类型可以用于存储照片信息。但是本例并不那样做，当数据库数据很多时，将照片（二进制数据）存储到数据库里就不是一个明智的选择了。大字段数据会加重数据库的负担，拖慢数据库，数据库文件越小访问肯定越快，数据库也不用遍历那么多内容，或者加载那么大的数据到内存里，造成响应不及时等。计算机可能处理速度很快，但是对于普通的单核和多核 ARM 开发板来说速度可能会跟不上啊！所以数据库最好是存储照片的路径。照片路径属于字符串文本，不会占用太多空间。

好了现在上例子，例子就是查询员工的编号、姓名、年龄、性别与照片信息，简单实用好理解，不复杂。

本例目的：用 QTableView 显示数据库表的数据，显示员工的信息。

例 18_sqlite_table，查询数据表（难度：一般）。项目路径为 Qt/2/ 18_sqlite_table。

项目文件 18_sqlite_table 文件第一行添加的代码部分如下。

18_sqlite_table.pro 编程后的代码

```
1  QT       += core gui sql
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
13 # You can also make your code fail to compile if it uses deprecated APIs.
14 # In order to do so, uncomment the following line.
15 # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18 SOURCES += \
19     main.cpp \
20     mainwindow.cpp
21
22 HEADERS += \
23     mainwindow.h
24
25 # Default rules for deployment.
26 qnx: target.path = /tmp/$${TARGET}/bin
27 else: unix:!android: target.path = /opt/$${TARGET}/bin
28 !isEmpty(target.path): INSTALLS += target
29
30 RESOURCES +=
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*************************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   18_sqlite_table
```

```
    * @brief        mainwindow.h
    * @author       Deng Zhimao
    * @email        1252699831@qq.com
    * @net          www.openedv.com
    * @date         2021-05-18
    ****************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QSqlDatabase>
5   #include <QSqlQuery>
6   #include <QMainWindow>
7   #include <QLabel>
8   #include <QSqlTableModel>
9   #include <QHBoxLayout>
10  #include <QVBoxLayout>
11  #include <QGridLayout>
12  #include <QTableView>
13  #include <QComboBox>
14  #include <QLineEdit>
15  #include <QDataWidgetMapper>
16  #include <QSqlQueryModel>
17  #include <QItemSelectionModel>
18  #include <QSpinBox>
19
20  class MainWindow : public QMainWindow
21  {
22      Q_OBJECT
23
24  public:
25      MainWindow(QWidget *parent = nullptr);
26      ~MainWindow();
27
28  private:
29
30      /* 数据库连接类 */
31      QSqlDatabase sqlDatabase;
32
33      /* 用于查询数据 */
34      QSqlQueryModel *sqlQueryModel;
35
36      /* 数据映射 */
37      QDataWidgetMapper *dataWidgetMapper;
38
```

```
39      /* 选择模型 */
40      QItemSelectionModel * itemSelectionModel;
41
42      /* 水平布局 */
43      QHBoxLayout *hBoxLayout[2];
44
45      /* 垂直布局 */
46      QVBoxLayout *vBoxLayout;
47
48      /* 网格布局 */
49      QGridLayout *gridLayout;
50
51      /* 用于显示的表格*/
52      QTableView *tableView;
53
54      /* 主 Widget */
55      QWidget *mainWidget;
56
57      /* 底部容器 */
58      QWidget *bottomWidget;
59
60      /* 底部网格布局容器 */
61      QWidget *gridWidget;
62
63      /* 照片容器 */
64      QWidget *photoWidget;
65
66      /* Label，用于显示照片 */
67      QLabel *imageLabel;
68
69      /* Label,底部显示文本 */
70      QLabel *label[4];
71
72      /* 性别下拉选择框，选择信息 */
73      QComboBox *comboBox;
74
75      /* 数值选择框，[0, 100] */
76      QSpinBox *spinBox[2];
77
78      /* 单行输入框  */
79      QLineEdit *lineEdit;
80
```

```
81  private slots:
82      /* 表格当前行变化执行的槽函数 */
83      void on_currentRowChanged(const QModelIndex&, const QModelIndex&);
84  };
85  #endif // MAINWINDOW_H
```

头文件主要声明布局用的类和数据库，重要关注是 QSqlDatabase、QSqlQueryModel 、QdataWidgetMapper 和 QItemSelectionModel。这里声明的是全局变量。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   18_sqlite_table
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-05-18
*****************************************************************/
1   #include "mainwindow.h"
2   #include <QDebug>
3   #include <QHeaderView>
4   #include <QSqlError>
5   #include <QApplication>
6   #include <QSqlRecord>
7
8   MainWindow::MainWindow(QWidget *parent)
9       : QMainWindow(parent)
10  {
11      /* 设置主窗体的显示位置与大小 */
12      this->setGeometry(0, 0, 800, 480);
13
14      /* 查看本机可用的数据库驱动 */
15      QStringList drivers = QSqlDatabase::drivers();
16      foreach(QString driver, drivers) {
17          qDebug()<<driver;
18      }
19
20      /* 以 QSQLITE 驱动方式打开或者创建数据库 */
21      sqlDatabase = QSqlDatabase::addDatabase("QSQLITE");
22      sqlDatabase.setDatabaseName("employee.db");
23      /* 以 open 的方式打开 employee.db 数据库，则会创建一个 employee.db */
24      if (!sqlDatabase.open())
25          qDebug()<<"连接数据库错误"<<sqlDatabase.lastError()<<endl;
```

```
26      else
27          qDebug()<<"连接数据库成功"<<endl;
28
29      QSqlQuery query(sqlDatabase);
30      /* 使用指令式创建表 */
31      query.exec("create table employee (id int primary key, name vchar(10),
"
32                 "age int, sex vchar(3), photo text)");
33
34      QStringList photoPath;
35      /* 当前可执行程序的路径 */
36      QString path(QApplication::applicationDirPath());
37      photoPath<< path + "/photos/啊万.jpg"<< path + "/photos/啊棠.jpg";
38
39      /* 以指令的方式插入数据，如果数据已经存在则不会成功不能插入 */
40      query.exec(tr("insert into employee values(1, '啊万', 27, '男',
'%1')").arg(photoPath[0]));
41      query.exec(tr("insert into employee values(2, '啊棠', 28, '男',
'%1')").arg(photoPath[1]));
42
43      //    /* 初始化表格模型 */
44      //    QSqlTableModel *model = new QSqlTableModel(this, sqlDatabase);
45
46      //    /* 设置要选中的表格名称 */
47      //    model->setTable("employee");
48      //    /* 如果有修改则同步修改到数据库，
49      //     * 注意这个规则需要与 tabview 这样的控件才生效，
50      //     * 因为 tabview 可以直接编辑表里的内容 */
51      //    model->setEditStrategy(QSqlTableModel::OnFieldChange);
52      //    /* 成功则返回 true，查看数据库里是否有 employee 这个表格 */
53      //    model->select();
54      //    /* 设置表格的头信息，若不设置则显示数据库里的英文字段头信息 */
55      //    model->setHeaderData(model->fieldIndex("id"),
56      //                         Qt::Horizontal, tr("编号"));
57      //    model->setHeaderData(model->fieldIndex("name"),
58      //                         Qt::Horizontal, tr("姓名"));
59      //    model->setHeaderData(model->fieldIndex("age"),
60      //                         Qt::Horizontal, tr("年龄"));
61      //    model->setHeaderData(model->fieldIndex("sex"),
62      //                         Qt::Horizontal, tr("性别"));
63
64      //    QTableView *view = new QTableView;
```

```
65
66      //    /* 设置表格的模型为 model */
67      //    view->setModel(model);
68      //    /* 不显示图片路径信息行 */
69      //    view->hideColumn(4);
70      //    /* 表格居中 */
71      //    setCentralWidget(view);
72      //    return;
73
74      /* QSqlQueryModel 适合用于查询数据，不能修改数据 */
75      sqlQueryModel = new QSqlQueryModel(this);
76
77      /* 选择编号，姓名，年龄和性别的内容，显示到 tableView 上，
78       * 图片最后通过数据选择再读取 Label 上 */
79      sqlQueryModel->setQuery("select id, name, age, sex from employee");
80
81      if (sqlQueryModel->lastError().isValid())
82          qDebug()<<"选择数据失败！"<<endl;
83
84      sqlQueryModel->setHeaderData(0, Qt::Horizontal, "编号");
85      sqlQueryModel->setHeaderData(1, Qt::Horizontal, "姓名");
86      sqlQueryModel->setHeaderData(2, Qt::Horizontal, "年龄");
87      sqlQueryModel->setHeaderData(3, Qt::Horizontal, "性别");
88
89      tableView = new QTableView();
90      tableView->setModel(sqlQueryModel);
91
92      /* 设置显示平均分列 */
93      tableView->horizontalHeader()
94              ->setSectionResizeMode(QHeaderView::Stretch);
95
96      mainWidget = new QWidget();
97      bottomWidget = new QWidget();
98      gridWidget = new QWidget();
99      photoWidget = new QWidget();
100     imageLabel = new QLabel();
101
102     /* 设置照片属性 */
103     imageLabel->setScaledContents(true);
104     imageLabel->setMaximumSize(200, 200);
105
106
```

```
107     vBoxLayout = new QVBoxLayout();
108     hBoxLayout[0] = new QHBoxLayout();
109     hBoxLayout[1] = new QHBoxLayout();
110     gridLayout = new QGridLayout();
111
112     for (int i = 0; i < 4; i++)
113         label[i] = new QLabel();
114
115     for (int i = 0; i < 2; i++) {
116         spinBox[i] = new QSpinBox();
117         spinBox[i]->setRange(1, 100);
118     }
119
120     comboBox = new QComboBox();
121     comboBox->addItem("男");
122     comboBox->addItem("女");
123
124     lineEdit = new QLineEdit();
125
126     bottomWidget->setMinimumHeight(this->height() / 2 - 30);
127     gridWidget->setMaximumWidth(this->width() / 2 - 30);
128
129     /* 垂直布局 */
130     vBoxLayout->addWidget(tableView);
131     vBoxLayout->addWidget(bottomWidget);
132
133     mainWidget->setLayout(vBoxLayout);
134     setCentralWidget(mainWidget);
135
136     /* 水平布局 */
137     hBoxLayout[0]->addWidget(gridWidget);
138     hBoxLayout[0]->addWidget(photoWidget);
139     bottomWidget->setLayout(hBoxLayout[0]);
140
141     QStringList list;
142     list<<"姓名："<<"编号："<<"年龄："<<"性别：";
143
144     /* 网格布局 */
145     for (int i = 0; i < 4; i++) {
146         gridLayout->addWidget(label[i], i, 0);
147         label[i]->setText(list[i]);
148         switch (i) {
149         case 0:
```

```
150             gridLayout->addWidget(lineEdit, i, 1);
151             break;
152         case 1:
153             gridLayout->addWidget(spinBox[0], i, 1);
154             break;
155         case 2:
156             gridLayout->addWidget(spinBox[1], i, 1);
157             break;
158         case 3:
159             gridLayout->addWidget(comboBox, i, 1);
160             break;
161         default:
162             break;
163         }
164     }
165
166     gridWidget->setLayout(gridLayout);
167     hBoxLayout[1]->addWidget(imageLabel);
168     photoWidget->setLayout(hBoxLayout[1]);
169
170     itemSelectionModel = new QItemSelectionModel(sqlQueryModel);
171     tableView->setSelectionModel(itemSelectionModel);
172
173     /* 信号槽连接，表示表中行数据变化时，触发槽函数 */
174     connect(itemSelectionModel,
175             SIGNAL(currentRowChanged(QModelIndex, QModelIndex)),
176             this,
177             SLOT(on_currentRowChanged(QModelIndex, QModelIndex)));
178
179     dataWidgetMapper = new QDataWidgetMapper(this);
180     /* 设置为自动提交 */
181
dataWidgetMapper->setSubmitPolicy(QDataWidgetMapper::AutoSubmit);
182     dataWidgetMapper->setModel(sqlQueryModel);
183     /* 创建数据映射，将前面的数据库内容映射到控件上 */
184     dataWidgetMapper->addMapping(lineEdit, 1);
185     dataWidgetMapper->addMapping(spinBox[0], 0);
186     dataWidgetMapper->addMapping(spinBox[1], 2);
187     dataWidgetMapper->addMapping(comboBox, 3);
188 }
189
190 MainWindow::~MainWindow()
191 {
192     /* 关闭数据库 */
```

```
193     sqlDatabase.close();
194 }
195
196 void MainWindow::on_currentRowChanged(const QModelIndex &current,
197                                       const QModelIndex &previous)
198 {
199     Q_UNUSED(previous)
200     /* 更新数据映射行号，初始化时映射到第 0 行 */
201     dataWidgetMapper->setCurrentModelIndex(current);
202     /* 获取当前行号 */
203     int row =  itemSelectionModel->currentIndex().row();
204     /* 获取当前模型记录 */
205     QSqlRecord record = sqlQueryModel->record(row);
206     /* 获取 id 信息 */
207     int id = record.value("id").toInt();
208     QSqlQuery query;
209     /* 使用 bindValue 绑定 prepare 里语句的值，需要使用":"，":"是占位符 */
210     query.prepare("select photo from employee where id = :ID");
211     query.bindValue(":ID", id);
212     query.exec();
213     /* 返回到选择的第一条记录，因为 id 是唯一的，也只有一条记录 */
214     query.first();
215
216     /* 获取字段为 photo 的值，也就是存储照片的路径 */
217     QVariant temp = query.value("photo");
218     if (!temp.isValid()) {
219         qDebug()<<"数据无效！"<<endl;
220         return;
221     }
222
223     /* 清空图片显示 */
224     imageLabel->clear();
225
226     QImage image(temp.toString());
227
228     if (image.isNull()) {
229         qDebug()<<"未找到"<<temp.toString()<<endl;
230         return;
231     }
232
233     /* 显示照片 */
234     imageLabel->setPixmap(QPixmap::fromImage(image));
235 }
```

第 15~41 行，连接数据库，创建数据库表，插入数据，与上一小节实用闹钟基本一样，不再赘述。

第 43~72 行，被注释行，这部分程序就是本小节开头所说的，如果只想显示数据库表里的数据就打开这个被注释掉的内容即可。

第 75~168 行，布局及一些设置的内容，不再解释。布局前面入门篇已经讲过。

第 170~187 行，重点关注这几行代码，170~177 行，QItemSelectionModel 将 sqlQueryModel 作为项的选择模型，然后，tableView 设置项的选择模型为 itemSelectionModel，这个目的就是使用 itemSelection 的行发生的变化的信号 currentRowChanged()。

第 179~187 行，这里主要是是将 sqlQueryModel 的数据通过 dataWidgetMapper 这个对象映射到我们的普通控件类上。比如映射第 4 个数据（index = 3）性别数据到 comboBox 上。可以看出 dataWidgetMapper 只是一个搬运工而已，将指定的数据搬到我们需要显示的控件上。

第 196~235 行，当我们点击表中的数据，行发生变化后，则这个槽函数触发，流程是从当前选择的行里，使用 QSqlRecord 记录当前行的数据，然后从当前行的数据提取出 id 的编号，再使用 QSqlQuery 的 exec()方法执行 sql 的 select 语句获取出 photo 字段照片的路径数据，最后将获取的照片路径数据初始化一个 QImage 对象显示到 imageLable 上，这样就实现了数据显示的功能。并且映射到控件上的内容也发生了改变。

当我们的数据库表足够大时，我们若想使用一些按钮来点击切换查询当前项的数据可以使用按钮连接到 dataWidgetMapper 的 toFirst()、toLast()、toNext()和 toPrevious()槽函数。意思是跳转到第一行，最后一行，下一行和前一行。最后将 tableView 设置为 dataWidgetMapper 当前行就会触发 currentRowChanged()，就能实现按钮控制查询数据了。至于读者想加什么功能由读者自由设计，本例只是写了个大概框架。

main.cpp 内容如下，没有修改。

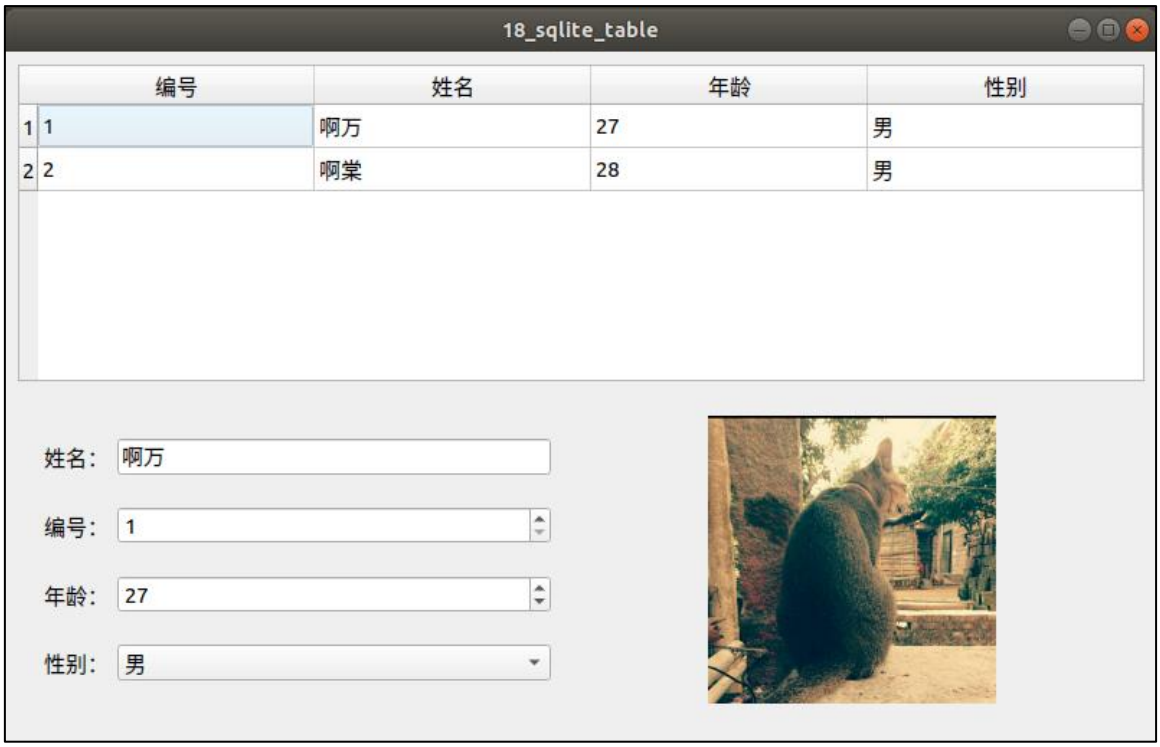
```
1  #include "mainwindow.h"
2
3  #include <QApplication>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8      MainWindow w;
9      w.show();
10     return a.exec();
11 }
```

13.2.2.1 程序运行效果

运行本例时，先点击构建，构建完成后将本项目下的整个 photos 文件夹拷贝到构建出来的 build-18_sqlite_table-Desktop_Qt_5_12_9_GCC_64bit-Debug 目录下。因为本程序会从数据库里读取出员工的照片路径信息，所以需要提前将照片文件夹放至可执行程序同一级目录下。点击表中的项，当切换员工的信息里，我们可以看到左下角被映射的内容发生了改变，变成了当前选

择行的员工信息，右下角的照片头像也变成了该员工的照片信息。本例实现的就是从数据库里取数据并显示到 QTableView 及搭配其他控件使用的例子。

员工啊万的信息：



员工啊棠信息：

第三篇 进阶篇

前面两篇都是在 Ubutnu 和 Windows 基本都是可以直接操作的，当然也是可以在开发板上操作，只是不依赖开发板。进阶篇则对开发板依赖性比较强。请用正点原子 I.MX6U 开发板进行实验，注意是用出厂系统，包括出厂内核（zImage modules）、设备树（dtb）出厂文件系统！

重要的事情得说三遍！

本篇使用的是正点原子 I.MX6U 出厂系统进行实验！

本篇使用的是正点原子 I.MX6U 出厂系统进行实验！

本篇使用的是正点原子 I.MX6U 出厂系统进行实验！

为什么强调是要使用默认的出厂系统？因为初学者做 Linux 驱动实验时会移植自己的内核，包括设备树和文件系统。这些与出厂系统有很大的差异！无法保证您的驱动是否可用！或者文件系统 Qt 库完整！好比如说，初学者连自己的屏触摸驱动都没有移植或者没写对，到时触摸一出问题本章无法继续！也不需要自己移植 Qt 到其他文件系统里测试。如果您能力可以，请自行测试！使用其他文件系统缺少 Qt 库的情况是发生在经常初学者身上的！本章针对正点原子的 I.MX6U 出厂系统进行实验，笔者在 Ubuntu 上实验，也已经在正点原子 I.MX6U 开发板上出厂系统验证可行！

本章将编写与开发板相关的实例，涉及到的都是初学者或者开发者经常需要使用的硬件接口或者硬件资源！例子比较精简实用，对开发实际项目有很大的帮助！

第十四章 I.MX6U Qt 开发

本章开始写与正点原子 I.MX6U 板子相关的实例。**所有例子都是基于正点原子 I.MX6U 的出厂系统上进行。请不要使用其他系统或者自己开发的内核设备树等**！否则可能驱动与应用对应不上没法操作硬件设备。本章适用于正点原子 ALPHA 或者 MINI 开发板。

搭建 I.MX6U 的 Qt 开发环境，正点原子已经早有相关文档，写的很详细。请大家先熟悉正点原子 I.MX6U 的 Qt 开发环境。交叉编译 Qt 应用程序有两种方法。一种是直接在终端使用命令行编译 Qt 项目。一种是在 Qt Creator 里搭建交叉编译套件的方法来编译 Qt 项目。请大家根据个人的喜好可两者都熟悉或者使用其中一种即可！本教程更倾向使用命令行编译 Qt 项目，在 Ubuntu 直接运行仿真，再交叉编译到开发板上运行查看实际效果即可！

14.1 使用命令行编译

请参考开发板光盘 A-基础资料/【正点原子】I.MX6U 用户快速体验 V1.x.pdf 第四章第 4.6 小节，使用命令行编译的方法。不过此方法需要拷贝执行文件到板子上运行。

14.2 在 Qt Creator 搭建交叉环境搭建

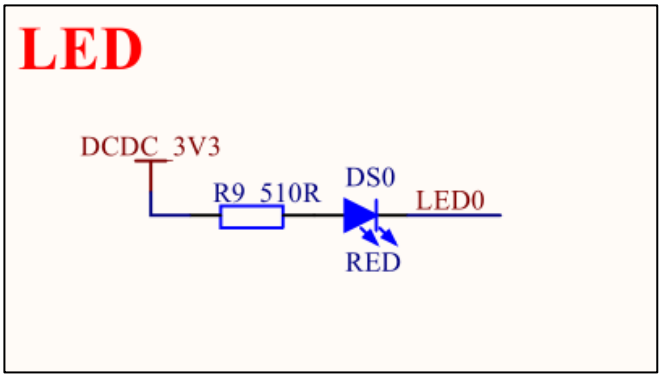
请参考开发板光盘 A-基础资料/【正点原子】I.MX6U 出厂系统 Qt 交叉编译环境搭建 V1.x.pdf。推荐使用在 Qt Creator 搭建调试环境。可以远程连接开发板运行。但若结果不正确时（比如需要环境变量，路径等），建议拷贝执行文件到开发板上运行。

第十五章 Qt 控制 LED

本章开始使用 Qt 应用到正点原子的嵌入式 I.MX6ULL 开发板上，凡事是先易后难，我们也是从最简单的点亮 LED 说起。介绍如何使用 Qt 知识应用到正点原子的嵌入式 I.MX6ULL 开发板，亦可参考来修改到其他平台的嵌入式 Linux 开发板上。

15.1 资源简介

在正点原子的 I.MX6U 开发板，ALPHA 和 MINI Linux 开发板板载资源上有一个 LED。如下图原理图（下图为 ALPHA 开发板的 LED 原理图）。



15.2 应用实例

想要控制这个 LED，首先我们正点原子的出厂内核已经默认将这个 LED 注册成了 gpio-leds 类型设备。所以我们可以直接在应用层接口直接可以操作这个 LED 设备。我们在 Qt 里有很多种方法可以控制正点原子 I.MX6U 的 LED 设备。如可以用 C 语言的读写函数读写来控制 LED 的状态，或者直接使用 system()函数启动一个进程执行相关指令直接控制 LED 等。

我们介绍最简单的方法控制开发板上的 LED，就是使用 Qt 的操作文件的类直接控制 LED。因为 Linux 上一切皆文件，所有的东西都当作文件来处理。

下面将贴上代码，其中不会再去讲如何搭建工程，不会贴上实验现象图。代码注释详细，不额外说明。实现现象请自行编译到开发板上运行查看。（笔者都有在正点原子 I.MX6U 开发板上实验，确保准确性）。项目虽然简单，但是在嵌入式里基本都是从点亮一个 LED 里开始说起。只有我们会操作一个 IO，剩下的基本都不会难！

项目简介：设置一个按钮，点击即可控制 LED 状态反转（点亮或者熄灭 LED）。项目看来很起来很简单，实际上有些需要注意的地方，我们在改变 LED 的状态时，需要先去读取 LED 的状态，防止外界（外面应用程序）将 LED 的状态改变了。否则我们反转操作将不成立。在 C++里一般使用 get()和 set()方法来获取和设置。我们的 LED 程序里也有这种方法。所以需要写好一个让人看得懂的程序是有“方法”的。不能将程序功能写在一堆，最好是分开写，留有接口。让后面的人看懂！

例 01_led，控制 LED（难度简单）。项目路径为 Qt/3/01_led。3 是代表第三篇的例程父目录。

在源文件“mainwindow.h”的代码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   01_led
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
```

```
 * @date          2021-03-08
 ************************************************************************/
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QPushButton>
#include <QFile>

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    /* 按钮 */
    QPushButton *pushButton;

    /* 文件 */
    QFile file;

    /* 设置 LED 的状态 */
    void setLedState();

    /* 获取 LED 的状态 */
    bool getLedState();

private slots:
    void pushButtonClicked();
};
#endif // MAINWINDOW_H
```

在头文件“mainwindow.h”里第 24 行声明一个设置 LED 状态方法，另一个是获取状态的方法。另外声明一个槽函数，作用是点击切换 LED 的状态。

在源文件“mainwindow.cpp”的代码如下。

```
/************************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   01_led
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
```

```
    * @net            www.openedv.com
    * @date           2021-03-08
    ************************************************************************/
1   #include "mainwindow.h"
2   #include <QDebug>
3   #include <QGuiApplication>
4   #include <QScreen>
5   #include <QRect>
6
7   MainWindow::MainWindow(QWidget *parent)
8       : QMainWindow(parent)
9   {
10      /* 获取屏幕的分辨率，Qt 官方建议使用这
11       * 种方法获取屏幕分辨率，防上多屏设备导致对应不上
12       * 注意，这是获取整个桌面系统的分辨率
13       */
14      QList <QScreen *> list_screen =  QGuiApplication::screens();
15
16      /* 如果是 ARM 平台，直接设置大小为屏幕的大小 */
17  #if __arm__
18      /* 重设大小 */
19      this->resize(list_screen.at(0)->geometry().width(),
20                   list_screen.at(0)->geometry().height());
21      /* 默认是出厂系统的 LED 心跳的触发方式,想要控制 LED,
22       * 需要改变 LED 的触发方式，改为 none，即无 */
23      system("echo none > /sys/class/leds/sys-led/trigger");
24  #else
25      /* 否则则设置主窗体大小为 800x480 */
26      this->resize(800, 480);
27  #endif
28
29      pushButton = new QPushButton(this);
30
31      /* 居中显示 */
32      pushButton->setMinimumSize(200, 50);
33      pushButton->setGeometry((this->width() - pushButton->width()) /2 ,
34                              (this->height() - pushButton->height()) /2,
35                              pushButton->width(),
36                              pushButton->height()
37                              );
38      /* 开发板的 LED 控制接口 */
39
file.setFileName("/sys/devices/platform/leds/leds/sys-led/brightness");
```

```
40
41      if (!file.exists())
42          /* 设置按钮的初始化文本 */
43          pushButton->setText("未获取到 LED 设备！");
44
45      /* 获取 LED 的状态 */
46      getLedState();
47
48      /* 信号槽连接 */
49      connect(pushButton, SIGNAL(clicked()),
50              this, SLOT(pushButtonClicked()));
51  }
52
53  MainWindow::~MainWindow()
54  {
55  }
56
57  void MainWindow::setLedState()
58  {
59      /* 在设置 LED 状态时先读取 */
60      bool state = getLedState();
61
62      /* 如果文件不存在，则返回 */
63      if (!file.exists())
64          return;
65
66      if(!file.open(QIODevice::ReadWrite))
67          qDebug()<<file.errorString();
68
69      QByteArray buf[2] = {"0", "1"};
70
71      /* 写 0 或 1 */
72      if (state)
73          file.write(buf[0]);
74      else
75          file.write(buf[1]);
76
77      /* 关闭文件 */
78      file.close();
79
80      /*重新获取 LED 的状态 */
81      getLedState();
82  }
```

```
83
84  bool MainWindow::getLedState()
85  {
86      /* 如果文件不存在，则返回 */
87      if (!file.exists())
88          return false;
89
90      if(!file.open(QIODevice::ReadWrite))
91          qDebug()<<file.errorString();
92
93      QTextStream in(&file);
94
95      /* 读取文件所有数据 */
96      QString buf = in.readLine();
97
98      /* 打印出读出的值 */
99      qDebug()<<"buf: "<<buf<<endl;
100     file.close();
101     if (buf == "1") {
102         pushButton->setText("LED 点亮");
103         return true;
104     } else {
105         pushButton->setText("LED 熄灭");
106         return false;
107     }
108 }
109
110 void MainWindow::pushButtonClicked()
111 {
112     /* 设置 LED 的状态 */
113     setLedState();
114 }
```

第 9 行~24 行，界面初始化设置，在嵌入式里，根据实际的屏的大小，设置全屏显示。按钮居中显示。

第 23 行，因为出厂系统里配置 LED 的触发方式为心跳方式，要想控制此 LED，需要将 LED 的触发方式改为 none，即是无触发方式。为了方便，笔者直接使用 system()函数，用指令的方式改变 LED 的触发方式。

第 54~82 行设置 LED 的方法，写入“0”或“1”代表开和关。写入之前先读取 LED 的状态，预防在用户其他地方有设置过 LED。

第 84 行~108 行，获取 LED 的状态。

第 110 行~114 行设置 LED 的状态，此方法为槽函数，由点击按钮触发。

至此常规的控制一个 IO，大概流程已经完成。

15.3 程序运行效果

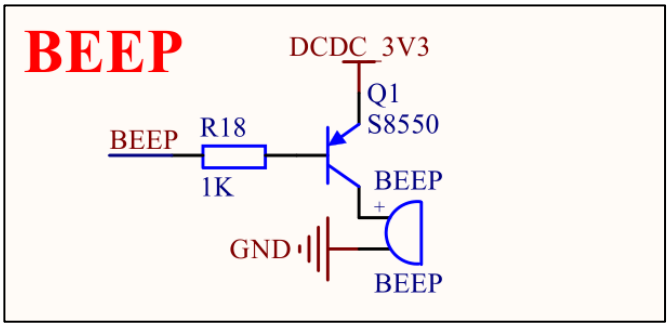
下面为 Ubuntu 上仿真界面的效果，由于 Ubuntu 不是“开发板”，所以在读取 LED 设备时会读取失败。实际在板上运行图略。交叉编译程序到正点原子 I.MX6U 开发板上运行即可控制 LED 的状态。

第十六章 Qt 控制 BEEP

本章是 Qt 控制蜂鸣器实验，原理和上一章点亮 LED 一样。

16.1 资源简介

在正点原子的 I.MX6U 开发板，ALPHA 和 MINI Linux 开发板板载资源上有一个蜂鸣器（BEEP）。如下图原理图。此蜂鸣器直接接在一个 GPIO 上，并不是接在 PWM 上，管脚资源限制。所以我们的操作与上一小节是一样的（下图为 ALPHA 开发板的 BEEP 原理图）。



16.2 应用实例

想要控制这个蜂鸣器（BEEP），首先我们正点原子的出厂内核已经默认将这个 LED 注册成了 gpio-leds 类型设备。所以实例与上一小节 LED 实例是一样的。

项目简介：设置一个按钮，点击即可控制 BEEP 状态反转（打开蜂鸣器或者关闭蜂鸣器）。例 02_beep，控制 BEEP（难度简单）。项目路径为 Qt/3/02_beep。

在源文件“mainwindow.cpp”的代码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   02_beep
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-03-11
*****************************************************************/
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QPushButton>
#include <QFile>

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
```

```
15
16  private:
17      /* 按钮 */
18      QPushButton *pushButton;
19
20      /* 文件 */
21      QFile file;
22
23      /* 设置 BEEP 的状态 */
24      void setBeepState();
25
26      /* 获取 BEEP 的状态 */
27      bool getBeepState();
28
29  private slots:
30      /* 槽函数 */
31      void pushButtonClicked();
32  };
33  #endif // MAINWINDOW_H
```

在头文件“mainwindow.h”里第 24~27 行声明一个设置蜂鸣器状态方法，另一个是获取状态的方法。另外第 31 声明一个槽函数，作用是点击切换蜂鸣器的状态。

在源文件“mainwindow.cpp”的代码如下。

```
    /*****************************************************************
    Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
    * @projectName   02_beep
    * @brief         mainwindow.cpp
    * @author        Deng Zhimao
    * @email         1252699831@qq.com
    * @net           www.openedv.com
    * @date          2021-03-11
    *****************************************************************/
1   #include "mainwindow.h"
2   #include <QDebug>
3   #include <QGuiApplication>
4   #include <QScreen>
5   #include <QRect>
6
7   MainWindow::MainWindow(QWidget *parent)
8       : QMainWindow(parent)
9   {
10      /* 获取屏幕的分辨率，Qt 官方建议使用这
11       * 种方法获取屏幕分辨率，防上多屏设备导致对应不上
```

```
12      * 注意，这是获取整个桌面系统的分辨率
13      */
14     QList <QScreen *> list_screen =  QGuiApplication::screens();
15
16     /* 如果是 ARM 平台，直接设置大小为屏幕的大小 */
17 #if __arm__
18     /* 重设大小 */
19     this->resize(list_screen.at(0)->geometry().width(),
20                  list_screen.at(0)->geometry().height());
21 #else
22     /* 否则则设置主窗体大小为 800x480 */
23     this->resize(800, 480);
24 #endif
25
26     pushButton = new QPushButton(this);
27
28     /* 居中显示 */
29     pushButton->setMinimumSize(200, 50);
30     pushButton->setGeometry((this->width() - pushButton->width()) /2 ,
31                             (this->height() - pushButton->height()) /2,
32                             pushButton->width(),
33                             pushButton->height()
34                             );
35     /* 开发板的蜂鸣器控制接口 */
36
file.setFileName("/sys/devices/platform/leds/leds/beep/brightness");
37
38     if (!file.exists())
39         /* 设置按钮的初始化文本 */
40         pushButton->setText("未获取到 BEEP 设备！");
41
42     /* 获取 BEEP 的状态 */
43     getBeepState();
44
45     /* 信号槽连接 */
46     connect(pushButton, SIGNAL(clicked()),
47             this, SLOT(pushButtonClicked()));
48 }
49
50
51 MainWindow::~MainWindow()
52 {
53 }
```

```
54
55  void MainWindow::setBeepState()
56  {
57      /* 在设置 BEEP 状态时先读取 */
58      bool state = getBeepState();
59
60      /* 如果文件不存在，则返回 */
61      if (!file.exists())
62          return;
63
64      if(!file.open(QIODevice::ReadWrite))
65          qDebug()<<file.errorString();
66
67      QByteArray buf[2] = {"0", "1"};
68
69      if (state)
70          file.write(buf[0]);
71      else
72          file.write(buf[1]);
73
74      file.close();
75
76      getBeepState();
77  }
78
79  bool MainWindow::getBeepState()
80  {
81      /* 如果文件不存在，则返回 */
82      if (!file.exists())
83          return false;
84
85      if(!file.open(QIODevice::ReadWrite))
86          qDebug()<<file.errorString();
87
88      QTextStream in(&file);
89
90      /* 读取文件所有数据 */
91      QString buf = in.readLine();
92
93      /* 打印出读出的值 */
94      qDebug()<<"buf: "<<buf<<endl;
95      file.close();
96      if (buf == "1") {
```

```
97          pushButton->setText("BEEP 开");
98          return true;
99      } else {
100         pushButton->setText("BEEP 关");
101         return false;
102     }
103 }
104
105 void MainWindow::pushButtonClicked()
106 {
107     /* 设置蜂鸣器的状态 */
108     setBeepState();
109 }
```

解释与上一小节 LED 的一样。

第 7~48 行，界面初始化设置，在嵌入式里，根据实际的屏的大小，设置全屏显示。按钮居中显示。

第 55~77 行设置蜂鸣器的方法，写入“0”或“1”代表开和关。写入之前先读取蜂鸣器的状态，预防在用户其他地方有设置过蜂鸣器。

第 79~103 行获取 BEEP 的状态。

第 105~109 行设置蜂鸣器的状态，此方法为槽函数，由点击按钮触发。

16.3 程序运行效果

下面为 Ubuntu 上仿真界面的效果，由于 Ubuntu 不是“开发板”，所以在读取 BEEP 设备时会读取失败。实际在板上运行图略。交叉编译程序到正点原子 I.MX6U 开发板上运行即可控制蜂鸣器的状态。


02_beep
未获取到BEEP设备!

第十七章 Serial Port

Qt 提供了串口类，可以直接对串口访问。我们可以直接使用 Qt 的串口类编程即可，十分方便。Qt 串口类不仅在 Windows 能用，还能在 Linux 下用，虽然串口编程不是什么新鲜事儿，既然 Qt 提供了这方面的接口，我们就充分利用起来，这将会使我们的开发十分方便！其实 Qt 也提供了相关的 Qt 串口的例子，我们也可以直接参考来编程，笔者根据实际情况，化繁为易，直接写了个简单的例子给大家参考。

17.1 资源简介

在正点原子的 I.MX6U 开发板的出厂系统里，默认已经配置了两路串口可用。一路是调试串口 UART1(对应系统里的节点/dev/ttymxc0)，另一路是 UART3(对应系统里的节点/dev/ttymxc2)。由于 UART1 已经作为调试串口被使用。所以我们只能对 UART3 编程，(如需要使用多路串口，请自行设计底板与系统)。

17.2 应用实例

项目简介：Qt 串口的使用示例，应用到正点原子 I.MX6U 开发板上。

例 03_serialport，Qt 串口编程（难度：一般）。项目路径为 Qt/3/03_serialport。

在 03_serialport.pro 里，我们需要使用串口，需要在 pro 项目文件中添加串口模块的支持，如下。

```
1  # 添加串口模块支持
2  QT       += core gui serialport
3
4  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
5
6  CONFIG += c++11
7
8  # The following define makes your compiler emit warnings if you use
9  # any Qt feature that has been marked deprecated (the exact warnings
10 # depend on your compiler). Please consult the documentation of the
11 # deprecated API in order to know how to port your code away from it.
12 DEFINES += QT_DEPRECATED_WARNINGS
13
14 # You can also make your code fail to compile if it uses deprecated APIs.
15 # In order to do so, uncomment the following line.
16 # You can also select to disable deprecated APIs only up to a certain
version of Qt.
17 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
18
19 SOURCES += \
20     main.cpp \
21     mainwindow.cpp
22
23 HEADERS += \
24     mainwindow.h
25
26 # Default rules for deployment.
27 qnx: target.path = /tmp/$${TARGET}/bin
28 else: unix:!android: target.path = /opt/$${TARGET}/bin
29 !isEmpty(target.path): INSTALLS += target
```

第 2 行，添加的 serialport 就是串口模块的支持。

在头文件“mainwindow.h”的代码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   03_serialport
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-03-12
*****************************************************************/
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSerialPort>
#include <QSerialPortInfo>
#include <QPushButton>
#include <QTextBrowser>
#include <QTextEdit>
#include <QVBoxLayout>
#include <QLabel>
#include <QComboBox>
#include <QGridLayout>
#include <QMessageBox>
#include <QDebug>

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    /* 串口对象 */
    QSerialPort *serialPort;

    /* 用作接收数据 */
    QTextBrowser *textBrowser;

    /* 用作发送数据 */
```

```
33      QTextEdit *textEdit;
34
35      /* 按钮 */
36      QPushButton *pushButton[2];
37
38      /* 下拉选择盒子 */
39      QComboBox *comboBox[5];
40
41      /* 标签 */
42      QLabel *label[5];
43
44      /* 垂直布局 */
45      QVBoxLayout *vboxLayout;
46
47      /* 网络布局 */
48      QGridLayout *gridLayout;
49
50      /* 主布局 */
51      QWidget *mainWidget;
52
53      /* 设置功能区域 */
54      QWidget *funcWidget;
55
56      /* 布局初始化 */
57      void layoutInit();
58
59      /* 扫描系统可用串口 */
60      void scanSerialPort();
61
62      /* 波特率项初始化 */
63      void baudRateItemInit();
64
65      /* 数据位项初始化 */
66      void dataBitsItemInit();
67
68      /* 检验位项初始化 */
69      void parityItemInit();
70
71      /* 停止位项初始化 */
72      void stopBitsItemInit();
73
74  private slots:
```

```
75      void sendPushButtonClicked();
76      void openSerialPortPushButtonClicked();
77      void serialPortReadyRead();
78  };
79  #endif // MAINWINDOW_H
```

上面代码是在 mianwindow.h 里声明需要用到的变量，方法及槽函数。

mainwindow.cpp 的代码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   03_serialport
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-03-12
*****************************************************************/
1   #include "mainwindow.h"
2   #include <QDebug>
3   #include <QGuiApplication>
4   #include <QScreen>
5   #include <QRect>
6
7   MainWindow::MainWindow(QWidget *parent)
8       : QMainWindow(parent)
9   {
10      /* 布局初始化 */
11      layoutInit();
12
13      /* 扫描系统的串口 */
14      scanSerialPort();
15
16      /* 波特率项初始化 */
17      baudRateItemInit();
18
19      /* 数据位项初始化 */
20      dataBitsItemInit();
21
22      /* 检验位项初始化 */
23      parityItemInit();
24
25      /* 停止位项初始化 */
26      stopBitsItemInit();
27  }
```

```
28
29  void MainWindow::layoutInit()
30  {
31      /* 获取屏幕的分辨率，Qt 官方建议使用这
32       * 种方法获取屏幕分辨率，防上多屏设备导致对应不上
33       * 注意，这是获取整个桌面系统的分辨率
34       */
35      QList <QScreen *> list_screen =  QGuiApplication::screens();
36
37      /* 如果是 ARM 平台，直接设置大小为屏幕的大小 */
38  #if __arm__
39      /* 重设大小 */
40      this->resize(list_screen.at(0)->geometry().width(),
41                   list_screen.at(0)->geometry().height());
42  #else
43      /* 否则则设置主窗体大小为 800x480 */
44      this->resize(800, 480);
45  #endif
46      /* 初始化 */
47      serialPort = new QSerialPort(this);
48      textBrowser = new QTextBrowser();
49      textEdit = new QTextEdit();
50      vboxLayout = new QVBoxLayout();
51      funcWidget = new QWidget();
52      mainWidget = new QWidget();
53      gridLayout = new QGridLayout();
54
55      /* QList 链表，字符串类型 */
56      QList <QString> list1;
57      list1<<"串口号:"<<"波特率:"<<"数据位:"<<"检验位:"<<"停止位:";
58
59      for (int i = 0; i < 5; i++) {
60          label[i] = new QLabel(list1[i]);
61          /* 设置最小宽度与高度 */
62          label[i]->setMinimumSize(80, 30);
63          /* 自动调整 label 的大小 */
64          label[i]->setSizePolicy(
65                      QSizePolicy::Expanding,
66                      QSizePolicy::Expanding
67                      );
68          /* 将 label[i]添加至网格的坐标(0, i) */
69          gridLayout->addWidget(label[i], 0, i);
```

```
70      }
71
72      for (int i = 0; i < 5; i++) {
73          comboBox[i] = new QComboBox();
74          comboBox[i]->setMinimumSize(80, 30);
75          /* 自动调整 label 的大小 */
76          comboBox[i]->setSizePolicy(
77                      QSizePolicy::Expanding,
78                      QSizePolicy::Expanding
79                      );
80          /* 将 comboBox[i]添加至网格的坐标(1, i) */
81          gridLayout->addWidget(comboBox[i], 1, i);
82      }
83
84      /* QList 链表，字符串类型 */
85      QList <QString> list2;
86      list2<<"发送"<<"打开串口";
87
88      for (int i = 0; i < 2; i++) {
89          pushButton[i] = new QPushButton(list2[i]);
90          pushButton[i]->setMinimumSize(80, 30);
91          /* 自动调整 label 的大小 */
92          pushButton[i]->setSizePolicy(
93                      QSizePolicy::Expanding,
94                      QSizePolicy::Expanding
95                      );
96          /* 将 pushButton[0]添加至网格的坐标(i, 5) */
97          gridLayout->addWidget(pushButton[i], i, 5);
98      }
99      pushButton[0]->setEnabled(false);
100
101     /* 布局 */
102     vboxLayout->addWidget(textBrowser);
103     vboxLayout->addWidget(textEdit);
104     funcWidget->setLayout(gridLayout);
105     vboxLayout->addWidget(funcWidget);
106     mainWidget->setLayout(vboxLayout);
107     this->setCentralWidget(mainWidget);
108
109     /* 占位文本 */
110     textBrowser->setPlaceholderText("接收到的消息");
111     textEdit->setText("www.openedv.com");
112
```

```
113     /* 信号槽连接 */
114     connect(pushButton[0], SIGNAL(clicked()),
115             this, SLOT(sendPushButtonClicked()));
116     connect(pushButton[1], SIGNAL(clicked()),
117             this, SLOT(openSerialPortPushButtonClicked()));
118
119     connect(serialPort, SIGNAL(readyRead()),
120             this, SLOT(serialPortReadyRead()));
121 }
122
123 void MainWindow::scanSerialPort()
124 {
125     /* 查找可用串口 */
126     foreach (const QSerialPortInfo &info,
127             QSerialPortInfo::availablePorts()) {
128         comboBox[0]->addItem(info.portName());
129     }
130 }
131
132 void MainWindow::baudRateItemInit()
133 {
134     /* QList 链表，字符串类型 */
135     QList <QString> list;
136     list<<"1200"<<"2400"<<"4800"<<"9600"
137       <<"19200"<<"38400"<<"57600"
138      <<"115200"<<"230400"<<"460800"
139      <<"921600";
140     for (int i = 0; i < 11; i++) {
141         comboBox[1]->addItem(list[i]);
142     }
143     comboBox[1]->setCurrentIndex(7);
144 }
145
146 void MainWindow::dataBitsItemInit()
147 {
148     /* QList 链表，字符串类型 */
149     QList <QString> list;
150     list<<"5"<<"6"<<"7"<<"8";
151     for (int i = 0; i < 4; i++) {
152         comboBox[2]->addItem(list[i]);
153     }
154     comboBox[2]->setCurrentIndex(3);
155 }
156
```

```
157 void MainWindow::parityItemInit()
158 {
159     /* QList 链表，字符串类型 */
160     QList <QString> list;
161     list<<"None"<<"Even"<<"Odd"<<"Space"<<"Mark";
162     for (int i = 0; i < 5; i++) {
163         comboBox[3]->addItem(list[i]);
164     }
165     comboBox[3]->setCurrentIndex(0);
166 }
167
168 void MainWindow::stopBitsItemInit()
169 {
170     /* QList 链表，字符串类型 */
171     QList <QString> list;
172     list<<"1"<<"2";
173     for (int i = 0; i < 2; i++) {
174         comboBox[4]->addItem(list[i]);
175     }
176     comboBox[4]->setCurrentIndex(0);
177 }
178
179 void MainWindow::sendPushButtonClicked()
180 {
181     /* 获取 textEdit 数据,转换成 utf8 格式的字节流 */
182     QByteArray data = textEdit->toPlainText().toUtf8();
183     serialPort->write(data);
184 }
185
186 void MainWindow::openSerialPortPushButtonClicked()
187 {
188     if (pushButton[1]->text() == "打开串口") {
189         /* 设置串口名 */
190         serialPort->setPortName(comboBox[0]->currentText());
191         /* 设置波特率 */
192         serialPort->setBaudRate(comboBox[1]->currentText().toInt());
193         /* 设置数据位数 */
194         switch (comboBox[2]->currentText().toInt()) {
195         case 5:
196             serialPort->setDataBits(QSerialPort::Data5);
197             break;
198         case 6:
199             serialPort->setDataBits(QSerialPort::Data6);
```

```
200             break;
201         case 7:
202             serialPort->setDataBits(QSerialPort::Data7);
203             break;
204         case 8:
205             serialPort->setDataBits(QSerialPort::Data8);
206             break;
207         default: break;
208         }
209         /* 设置奇偶校验 */
210         switch (comboBox[3]->currentIndex()) {
211         case 0:
212             serialPort->setParity(QSerialPort::NoParity);
213             break;
214         case 1:
215             serialPort->setParity(QSerialPort::EvenParity);
216             break;
217         case 2:
218             serialPort->setParity(QSerialPort::OddParity);
219             break;
220         case 3:
221             serialPort->setParity(QSerialPort::SpaceParity);
222             break;
223         case 4:
224             serialPort->setParity(QSerialPort::MarkParity);
225             break;
226         default: break;
227         }
228         /* 设置停止位 */
229         switch (comboBox[4]->currentText().toInt()) {
230         case 1:
231             serialPort->setStopBits(QSerialPort::OneStop);
232             break;
233         case 2:
234             serialPort->setStopBits(QSerialPort::TwoStop);
235             break;
236         default: break;
237         }
238         /* 设置流控制 */
239         serialPort->setFlowControl(QSerialPort::NoFlowControl);
240         if (!serialPort->open(QIODevice::ReadWrite))
241             QMessageBox::about(NULL, "错误",
242                                "串口无法打开！可能串口已经被占用！");
```

```
243         else {
244             for (int i = 0; i < 5; i++)
245                 comboBox[i]->setEnabled(false);
246             pushButton[1]->setText("关闭串口");
247             pushButton[0]->setEnabled(true);
248         }
249     } else {
250         serialPort->close();
251         for (int i = 0; i < 5; i++)
252             comboBox[i]->setEnabled(true);
253         pushButton[1]->setText("打开串口");
254         pushButton[0]->setEnabled(false);
255     }
256 }
257
258 void MainWindow::serialPortReadyRead()
259 {
260     /* 接收缓冲区中读取数据 */
261     QByteArray buf = serialPort->readAll();
262     textBrowser->insertPlainText(QString(buf));
263 }
264
265 MainWindow::~MainWindow()
266 {
267 }
```

第 29~121 行，界面布局初始化设置，在嵌入式里，根据实际的屏的大小，设置全屏显示。其中我们用到垂直布局和网格布局，如果布局这方面内容理解不了，请回到第七章 7.5 小节学习布局内容，学以致用理解的时候到了。

第 123~130 行，查找系统可用的串口，并添加串口名到 comboBox[0]中。

第 132~144 行，波特率初始化，预设常用的波特率，115200 作为默认选项。并添加波特率到 comboBox[1]中。

第 146~155 行，数据位项初始化，设置默认数据位为 8。

第 157~166 行，校验位项初始化，默认无校验位。

第 168~177 行，停止位项初始化，默认停止位为 1。

第 179~184 行，发送数据，点击发送按钮时触发。

第 186~256 行，打开或者关闭串口。以我们设置的项使用 Qt 串口提供的设置串口的方法如 setDataBits(QSerialPort::DataBits)等，按第 188~239 行步骤设置完串口需要配置的参数就可以打开或者关闭串口了。

第 258~263 行，从缓冲区里读出数据，并显示到 textBrowser 里。

17.3 程序运行效果

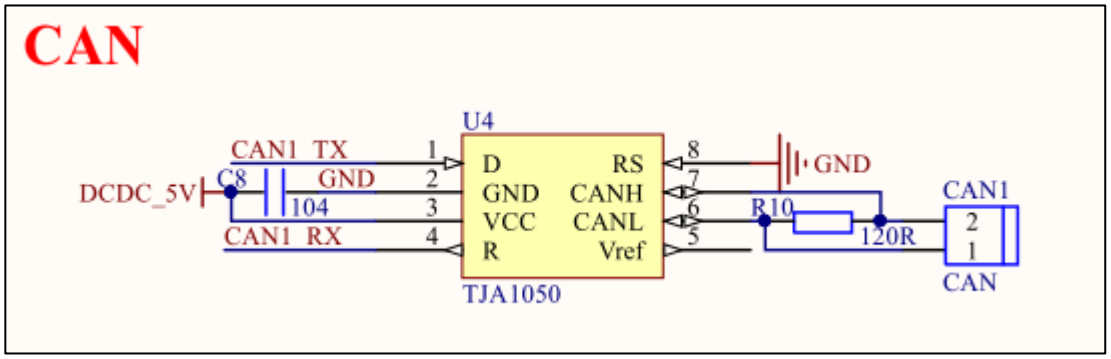
下面为 Ubuntu 上仿真界面的效果，请将程序交叉编译后到开发板运行，用串口线连接开发板的 UART3 到电脑串口，在电脑用正点原子的 XCOM 上位机软件(或者本程序亦可当上位机软件)，设置相同的串口参数，选择串口号为 ttymxc2（注意 ttymxc0 已经作为调试串口被使用了！），点击打开串口就可以进行消息收发了。默认参数为波特率为 115200，数据位为 8，校验为 None，停止位为 1，流控为关闭。

第十八章 CAN Bus

从 Qt5.8 开始，提供了 CAN Bus 类，很庆幸，正点原子的 I.MX6U 出厂系统里 Qt 版本是 QT5.12.9。我们可以直接使用 Qt 的提供的 CAN 相关类编程即可。假设您的 Qt 版本没有 CAN Bus，可以参考 Linux 应用编程来操控开发板的 CAN，目前我们主要讲解 Qt 相关的 CAN 编程。其实 Qt 也提供了相关的 Qt CAN 的例子，我们也可以直接参考来编程。笔者根据实际情况，化繁为易，直接写了个简单的例子给大家参考。最重要的一点，读者应会使用 CAN 相关测试工具，我们应该提前去熟悉，并且读者手上需要有测试 CAN 的仪器！否则写好程序，却无法测试，这就有些尴尬了。

18.1 资源简介

正点原子 I.MX6U 开发板底板上预留了一路 CAN 接口（6U 芯片最大支持两路）。如下图。在正点原子【正点原子】I.MX6U 用户快速体验 V1.x.pdf 里也有相关的 CAN 测试方法。这里就不多介绍 CAN 了，笔者默认读者是会使用 CAN 的。同时不对 CAN 总线协议进行讲解，主要是讲解如何在 Qt 里对 CAN 编程。



18.2 应用实例

项目简介：本例适用于正点原子 I.MX6U 开发板。不适用于 Windows。因为 Windows 没有 CAN 设备。虽然 Windows 可以外接 USB CAN 模块，但是这些模块都是某些厂商开发的，需要有相应的固件才能驱动 CAN 设备。所以编写的例子不一定适用于 Windows 下的 CAN。笔者写的例子已经在正点原子 I.MX6U 开发板上验证了，确保正常使用！

在正点原子 I.MX6U 板上，需要使用 CAN 必须初始化 CAN。它的开启与关闭都是由系统完成。I.MX6U 为普通 CAN，非 FD CAN，最大比特率为 1000kBit/s。

在系统执行要在 100 毫秒后自动从“总线关闭”错误中恢复，并以比特率 1000000，可以使用以下命令开启 CAN。

```
ip link set up can0 type can bitrate 1000000 restart-ms 100
```

例 04_socketcan，Qt CAN 编程（难度：较难）。项目路径为 Qt/3/04_ socketcan。

04_socketcan.pro 要想使用 Qt 的 QCanBus，需要在 pro 项目文件里添加相应的模块支持。同时还需要添加对应的头文件，详细请看项目里的代码。

```
1  QT       += core gui serialbus
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
```

```
13  # You can also make your code fail to compile if it uses deprecated APIs.
14  # In order to do so, uncomment the following line.
15  # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16  #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18  SOURCES += \
19      main.cpp \
20      mainwindow.cpp
21
22  HEADERS += \
23      mainwindow.h
24
25  # Default rules for deployment.
26  qnx: target.path = /tmp/${TARGET}/bin
27  else: unix:!android: target.path = /opt/${TARGET}/bin
28  !isEmpty(target.path): INSTALLS += target
```

第 1 行，添加的 serialbus 就是添加串行总线模块的支持。

在头文件“mainwindow.h”的代码如下。一些声明。

```
    /*****************************************************************
    Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
    * @projectName   04_socketcan
    * @brief         mainwindow.h
    * @author        Deng Zhimao
    * @email         1252699831@qq.com
    * @net           www.openedv.com
    * @date          2021-03-15
    *****************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QCanBusDevice>
6   #include <QCanBus>
7   #include <QPushButton>
8   #include <QTextBrowser>
9   #include <QLineEdit>
10  #include <QVBoxLayout>
11  #include <QLabel>
12  #include <QComboBox>
13  #include <QGridLayout>
14  #include <QMessageBox>
15  #include <QDebug>
```

```
16
17  class MainWindow : public QMainWindow
18  {
19      Q_OBJECT
20
21  public:
22      MainWindow(QWidget *parent = nullptr);
23      ~MainWindow();
24
25  private:
26      /* CAN 设备 */
27      QCanBusDevice *canDevice;
28
29      /* 用作接收数据 */
30      QTextBrowser *textBrowser;
31
32      /* 用作发送数据 */
33      QLineEdit *lineEdit;
34
35      /* 按钮 */
36      QPushButton *pushButton[2];
37
38      /* 下拉选择盒子 */
39      QComboBox *comboBox[3];
40
41      /* 标签 */
42      QLabel *label[4];
43
44      /* 垂直布局 */
45      QVBoxLayout *vboxLayout;
46
47      /* 网络布局 */
48      QGridLayout *gridLayout;
49
50      /* 主布局 */
51      QWidget *mainWidget;
52
53      /* 设置功能区域 */
54      QWidget *funcWidget;
55
56      /* 布局初始化 */
57      void layoutInit();
58
```

```
59      /* 插件类型项初始化 */
60      void pluginItemInit();
61
62      /* 比特率项初始化 */
63      void bitrateItemInit();
64
65  private slots:
66      /* 发送消息 */
67      void sendFrame();
68
69      /* 接收消息 */
70      void receivedFrames();
71
72      /* 插件发生改变 */
73      void pluginChanged(int);
74
75      /* 处理 can 错误 */
76      void canDeviceErrors(QCanBusDevice::CanBusError) const;
77
78      /* 连接或者断开 can */
79      void connectDevice();
80  };
81  #endif // MAINWINDOW_H
82
```

上面代码是在 mianwindow.h 里声明需要用到的变量，方法及槽函数。

mainwindow.cpp 的代码如下。

```
    /*****************************************************************
    Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
    * @projectName   04_socketcan
    * @brief         mainwindow.cpp
    * @author        Deng Zhimao
    * @email         1252699831@qq.com
    * @net           www.openedv.com
    * @date          2021-03-15
    *****************************************************************/
1   #include "mainwindow.h"
2   #include <QGuiApplication>
3   #include <QScreen>
4
5   MainWindow::MainWindow(QWidget *parent)
6       : QMainWindow(parent)
7   {
8       /* 使用系统指令比特率初始化 CAN,默认为 1000000bits/s */
```

```
9       system("ifconfig can0 down");
10      system("ip link set up can0 type can bitrate 1000000 restart-ms 100");
11
12      /* 布局初始化 */
13      layoutInit();
14
15      /* 可用插件初始化 */
16      pluginItemInit();
17
18      /* 可用接口项初始化 */
19      pluginChanged(comboBox[0]->currentIndex());
20
21      /* 比特率项初始化 */
22      bitrateItemInit();
23  }
24
25  MainWindow::~MainWindow()
26  {
27  }
28
29  static QString frameFlags(const QCanBusFrame &frame)
30  {
31      /* 格式化接收到的消息 */
32      QString result = QLatin1String(" --- ");
33
34      if (frame.hasBitrateSwitch())
35          result[1] = QLatin1Char('B');
36      if (frame.hasErrorStateIndicator())
37          result[2] = QLatin1Char('E');
38      if (frame.hasLocalEcho())
39          result[3] = QLatin1Char('L');
40
41      return result;
42  }
43
44  /* 发送消息 */
45  void MainWindow::sendFrame()
46  {
47      if (!canDevice)
48          return;
49      /* 读取 QLineEdit 的文件 */
50      QString str = lineEdit->text();
51      QByteArray data = 0;
```

```
52      QString strTemp = nullptr;
53      /* 以空格分隔 lineEdit 的内容，并存储到字符串链表中 */
54      QStringList strlist = str.split(' ');
55      for (int i = 1; i < strlist.count(); i++) {
56          strTemp = strTemp + strlist[i];
57      }
58      /* 将字符串的内容转为 QByteArray 类型 */
59      data = QByteArray::fromHex(strTemp.toLatin1());
60
61      bool ok;
62      /* 以 16 进制读取要发送的帧内容里第一个数据，并作为帧 ID */
63      int framId = strlist[0].toInt(&ok, 16);
64      QCanBusFrame frame = QCanBusFrame(framId, data);
65      /* 写入帧 */
66      canDevice->writeFrame(frame);
67  }
68
69  /* 接收消息 */
70  void MainWindow::receivedFrames()
71  {
72      if (!canDevice)
73          return;
74
75      /* 读取帧 */
76      while (canDevice->framesAvailable()) {
77          const QCanBusFrame frame = canDevice->readFrame();
78          QString view;
79          if (frame.frameType() == QCanBusFrame::ErrorFrame)
80              view = canDevice->interpretErrorFrame(frame);
81          else
82              view = frame.toString();
83
84          const QString time = QString::fromLatin1("%1.%2  ")
85                  .arg(frame.timeStamp()
86                       .seconds(), 10, 10, QLatin1Char(' '))
87                  .arg(frame.timeStamp()
88                       .microSeconds() / 100, 4, 10, QLatin1Char('0'));
89
90          const QString flags = frameFlags(frame);
91          /* 接收消息框追加接收到的消息 */
92          textBrowser->insertPlainText(time + flags + view + "\n");
93      }
94  }
```

```
95
96  void MainWindow::layoutInit()
97  {
98      /* 获取屏幕的分辨率，Qt 官方建议使用这
99       * 种方法获取屏幕分辨率，防上多屏设备导致对应不上
100      * 注意，这是获取整个桌面系统的分辨率
101      */
102     QList <QScreen *> list_screen =  QGuiApplication::screens();
103
104     /* 如果是 ARM 平台，直接设置大小为屏幕的大小 */
105 #if __arm__
106     /* 重设大小 */
107     this->resize(list_screen.at(0)->geometry().width(),
108                  list_screen.at(0)->geometry().height());
109 #else
110     /* 否则则设置主窗体大小为 800x480 */
111     this->resize(800, 480);
112 #endif
113     /* 对象初始化 */
114     textBrowser = new QTextBrowser();
115     lineEdit = new QLineEdit();
116     vboxLayout = new QVBoxLayout();
117     funcWidget = new QWidget();
118     mainWidget = new QWidget();
119     gridLayout = new QGridLayout();
120
121     /* QList 链表，字符串类型 */
122     QList <QString> list1;
123     list1<<"插件类型:"<<"可用接口:"<<"比特率 bits/sec:";
124
125     for (int i = 0; i < 3; i++) {
126         label[i] = new QLabel(list1[i]);
127         /* 设置最小宽度与高度 */
128         label[i]->setMinimumSize(120, 30);
129         label[i]->setMaximumHeight(50);
130         /* 自动调整 label 的大小 */
131         label[i]->setSizePolicy(QSizePolicy::Expanding,
132                                 QSizePolicy::Expanding);
133         /* 将 label[i]添加至网格的坐标(0, i) */
134         gridLayout->addWidget(label[i], 0, i);
135     }
136     label[3] = new QLabel();
```

```
137     label[3]->setMaximumHeight(30);
138
139     for (int i = 0; i < 3; i++) {
140         comboBox[i] = new QComboBox();
141         comboBox[i]->setMinimumSize(120, 30);
142         comboBox[i]->setMaximumHeight(50);
143         /* 自动调整 label 的大小 */
144         comboBox[i]->setSizePolicy(QSizePolicy::Expanding,
145                                    QSizePolicy::Expanding);
146         /* 将 comboBox[i]添加至网格的坐标(1, i) */
147         gridLayout->addWidget(comboBox[i], 1, i);
148     }
149
150     /* QList 链表，字符串类型 */
151     QList <QString> list2;
152     list2<<"发送"<<"连接 CAN";
153
154     for (int i = 0; i < 2; i++) {
155         pushButton[i] = new QPushButton(list2[i]);
156         pushButton[i]->setMinimumSize(120, 30);
157         pushButton[i]->setMaximumHeight(50);
158         /* 自动调整 label 的大小 */
159         pushButton[i]->setSizePolicy(QSizePolicy::Expanding,
160                                      QSizePolicy::Expanding);
161         /* 将 pushButton[0]添加至网格的坐标(i, 3) */
162         gridLayout->addWidget(pushButton[i], i, 3);
163     }
164     pushButton[0]->setEnabled(false);
165
166     /* 布局 */
167     vboxLayout->addWidget(textBrowser);
168     vboxLayout->addWidget(lineEdit);
169     funcWidget->setLayout(gridLayout);
170     vboxLayout->addWidget(funcWidget);
171     vboxLayout->addWidget(label[3]);
172     mainWidget->setLayout(vboxLayout);
173     this->setCentralWidget(mainWidget);
174
175     /* 设置文本 */
176     textBrowser->setPlaceholderText("系统时间 帧 ID 长度 数据");
177     lineEdit->setText("123 aa 77 66 55 44 33 22 11");
178     label[3]->setText(tr("未连接！"));
179
```

```
180     connect(pushButton[1], SIGNAL(clicked()),
181             this, SLOT(connectDevice()));
182     connect(pushButton[0], SIGNAL(clicked()),
183             this, SLOT(sendFrame()));
184 }
185
186 /* 从系统中读取可用的插件，并显示到 comboBox[0] */
187 void MainWindow::pluginItemInit()
188 {
189     comboBox[0]->addItems(QCanBus::instance()->plugins());
190     for (int i = 0; i < QCanBus::instance()->plugins().count(); i++) {
191         if (QCanBus::instance()->plugins().at(i) == "socketcan")
192             comboBox[0]->setCurrentIndex(i);
193     }
194     connect(comboBox[0], SIGNAL(currentIndexChanged(int)),
195             this, SLOT(pluginChanged(int)));
196 }
197
198 /* 插件类型改变 */
199 void MainWindow::pluginChanged(int)
200 {
201     QList<QCanBusDeviceInfo> interfaces;
202     comboBox[1]->clear();
203     /* 当我们改变插件时，我们同时需要将可用接口，从插件类型中读取出来 */
204     interfaces = QCanBus::instance()
205             ->availableDevices(comboBox[0]->currentText());
206     for (const QCanBusDeviceInfo &info : qAsConst(interfaces)) {
207         comboBox[1]->addItem(info.name());
208     }
209 }
210
211 /* 初始化一些常用的比特率，can 的比特率不是随便设置的，有相应的计算公式 */
212 void MainWindow::bitrateItemInit()
213 {
214     const QList<int> rates = {
215         10000, 20000, 50000, 100000, 125000,
216         250000, 500000, 800000, 1000000
217     };
218
219     for (int rate : rates)
220         comboBox[2]->addItem(QString::number(rate), rate);
221
222     /* 默认初始化以 1000000 比特率 */
```

```
223     comboBox[2]->setCurrentIndex(8);
224 }
225
226 /* 连接或断开 CAN */
227 void MainWindow::connectDevice()
228 {
229     if (pushButton[1]->text() == "连接 CAN") {
230         /* Qt 中的 QCanBusDevice::BitRateKey 不能设置比特率 */
231         QString cmd1 = tr("ifconfig %1 down")
232                .arg(comboBox[1]->currentText());
233         QString cmd2 =
234                tr("ip link set up %1 type can bitrate %2 restart-ms 100")
235                .arg(comboBox[1]->currentText())
236                .arg(comboBox[2]->currentText());
237         /* 使用系统指令以设置的比特率初始化 CAN */
238         system(cmd1.toStdString().c_str());
239         system(cmd2.toStdString().c_str());
240
241         QString errorString;
242         /* 以设置的插件名与接口实例化 canDevice */
243         canDevice = QCanBus::instance()->
244                createDevice(comboBox[0]->currentText(),
245                comboBox[1]->currentText(),
246                &errorString);
247
248         if (!canDevice) {
249             label[3]->setText(
250                        tr("Error creating device '%1', reason: '%2'")
251                        .arg(comboBox[0]->currentText())
252                    .arg(errorString));
253             return;
254         }
255
256         /* 连接 CAN */
257         if (!canDevice->connectDevice()) {
258             label[3]->setText(tr("Connection error: %1")
259                             .arg(canDevice->errorString()));
260             delete canDevice;
261             canDevice = nullptr;
262
263             return;
264         }
265
```

```
266         connect(canDevice, SIGNAL(framesReceived()),
267                 this, SLOT(receivedFrames()));
268         connect(canDevice,
269                 SIGNAL(errorOccurred(QCanBusDevice::CanBusError)),
270                 this,
271                 SLOT(canDeviceErrors(QCanBusDevice::CanBusError)));
272         /* 将连接信息插入到 label */
273         label[3]->setText(
274                     tr("插件类型为: %1, 已连接到 %2, 比特率为 %3 kBit/s")
275                     .arg(comboBox[0]->currentText())
276                 .arg(comboBox[1]->currentText())
277                 .arg(comboBox[2]->currentText().toInt() / 1000));
278         pushButton[1]->setText("断开 CAN");
279         /* 使能/失能 */
280         pushButton[0]->setEnabled(true);
281         comboBox[0]->setEnabled(false);
282         comboBox[1]->setEnabled(false);
283         comboBox[2]->setEnabled(false);
284     } else {
285         if (!canDevice)
286             return;
287
288         /* 断开连接 */
289         canDevice->disconnectDevice();
290         delete canDevice;
291         canDevice = nullptr;
292         pushButton[1]->setText("连接 CAN");
293         pushButton[0]->setEnabled(false);
294         label[3]->setText(tr("未连接! "));
295         comboBox[0]->setEnabled(true);
296         comboBox[1]->setEnabled(true);
297         comboBox[2]->setEnabled(true);
298     }
299 }
300
301 void MainWindow::canDeviceErrors(QCanBusDevice::CanBusError error)
const
302 {
303     /* 错误处理 */
304     switch (error) {
305     case QCanBusDevice::ReadError:
306     case QCanBusDevice::WriteError:
307     case QCanBusDevice::ConnectionError:
```

```
308     case QCanBusDevice::ConfigurationError:
309     case QCanBusDevice::UnknownError:
310         label[3]->setText(canDevice->errorString());
311         break;
312     default:
313         break;
314     }
315 }
316
```

第 9~10 行，使用系统的 CAN 硬件，必须初始化系统的 CAN。在项目里添加相应的开启 CAN 的指令。第一个指令是先关闭本地的 CAN，因为只有关闭 CAN，才能以新的速率再开启。

第 12~22 行，构造函数里界面初始化，以及 QComboBox 里的项初始化。

第 29~42 行，格式化帧处理函数。

第 45~67 行，发送消息，将 lineEdit 的文本进行处理后，第一个作为 CAN 的帧 ID，后面 8 个数据作为需要要发送的数据。每帧只能发送 8 个数据。

第 70~94 行，接收消息，读取帧并格式化处理，显示到 textBrowser 里。

第 96~184 行，界面布局初始化设置，在嵌入式里，根据实际的屏的大小，设置全屏显示。其中我们用到垂直布局和网格布局，如果布局这方面内容理解不了，请回到第七章 7.5 小节学习布局内容。

第 187~196 行，可用插件初始化，检查系统 QCanBus 提供的插件。在 Linux 里使用的插件类型是 SocketCAN, SocketCAN 插件支持 Linux 内核和用于所用 CAN 硬件的 SocketCAN 设备驱动程序。下面程序遍历可用的 CAN 插件，并设置 socketcan 为当前插件。注意，只能使用 SocketCAN 访问本地硬件 CAN，其他插件是不同类型的 CAN 驱动程序所使用的。请自行测试。

第 199~209 行，当插件类型改变时，我们需要更新可用接口。

第 212~224 行，常用的比特率初始化。

第 227~299 行，连接/断开 CAN，很遗憾 Qt 的 QCanBusDevice::BitRateKey 不能设置比特率，因为系统的 CAN 需要使用 ip 指令以一个比特率才能进行初始化，Qt 需要系统 CAN 起来才能进行操作。所以需要使用系统指令设置 CAN。

第 301~315 行，错误处理，CAN 设备可能遇到错误，打印错误的信息。

18.3 程序运行效果

在 Ubuntu 上运行 界面效果如下，因为 Ubutnu 没有 CAN 设备，所以在可用接口处是不可选的。请把程序交叉编译到开发板上运行。与 CAN 仪器以相同的比特率通信，插件类型默认是（必须是）socketcan，可用接口为 can0，即可发送消息与接收消息。

下图最上面的是接收消息框，“123 aa 77 66 55 44 33 22 11”这个是需要发送的帧，“123”为帧 ID，后面的为 8 个字节数据，每个字节需要以空格隔开。点击连接后，发送按钮才能使用。


04_socketcan
系统时间 帧ID 长度 数据
123 aa 77 66 55 44 33 22 11
插件类型:
可用接口:
比特率bits/sec:
发送
socketcan
1000000
连接CAN
未连接!

第十九章 Camera

此章节例程适用于 Ubuntu 和正点原子 I.MX6U 开发板，不适用于 Windows（需要自行修改才能适用 Windows，Windows 上的应用不在我们讨论范围）！

19.1 资源简介

正点原子 I.MX6U 开发板底板上有一路“CSI”摄像头接口。支持正点原子的 OV5640、OV2640 和 OV7725(不带 FIFO)。同时有 USB 接口，可以接 USB 免驱摄像头。例程兼容 USB 摄像头与正点原子的 OV5640、OV2640 和 OV7725 摄像头。

出厂系统请更新到正点原子 I.MX6U 最新的出厂系统，在驱动层正点原子对 OV5640、OV2640 和 OV7725 摄像头维护、优化或者添加支持。

19.2 环境搭建

Qt 里也有一个 QCamera 类。没错，确实可以使用这个 QCamera 类来开发摄像头。但是这个类在正点原子的 I.MX6U 开发板 4.1.15 内核版本上不能使用 OV5640、OV2640 和 OV7725，可以使用 USB 免驱摄像头。因为 OV5640、OV2640 和 OV7725 的驱动默认是读取 YUYV 格式数据，而 QCamera 里读取的数据是 RGB 格式数据，它们可能数据对不上就无法使用了！但是也不建议修改驱动的方法来使用 QCamera，防止 QCamera 在某些方法上与驱动层不对应，导致使用报错。

实际上我们使用 V4l2 编程就可以对摄像头进行编程，效果会比在 Qt 显示流畅，因为 V4l2 的数据直接可以显示在 fb0(也就是屏上)。而经过 Qt 还需要在控件上处理，所以效率会慢一些！在正点原子 I.MX6U 开发板上或者说是嵌入式上，比较低性能的 CPU 对于处理图像等大数据还是比较吃力的。所以我们需要去优化，不同的编程方式，对数据的处理不同，写出来的效果也会不同！

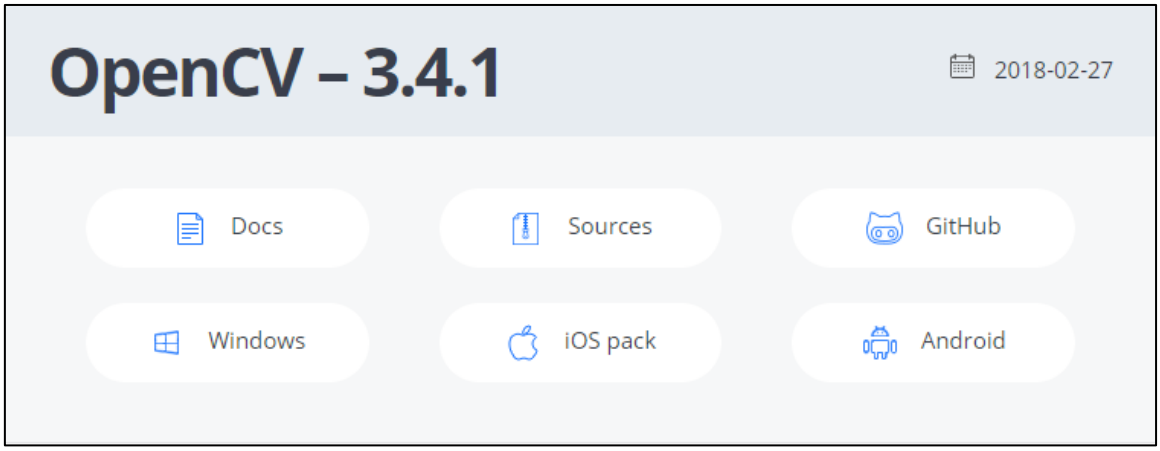
下面主要介绍 Qt + OpenCV 调用摄像头，效果肯定不能与使用 V4l2 直接显示在 fb0 上相比。在这个开发板的 CPU 上显示效果还是比较好的，还能接受，流畅度一般。这个可能是 OpenCV 在对摄像头数据做了一定的处理才会变的慢了。

要想在 Ubuntu 上使用 OpenCV，那么我们的 Ubuntu 上必须有 OpenCV 的库，如果您不想在 Ubuntu 安装 OpenCV，就可以跳过这小节，直接用出厂系统提供的交叉编译工具链，里面已经提供有 OpenCV。在 Ubuntu 上安装 OpenCV 只是方便我们测试界面，编写的程序也可以在 Ubuntu 上运行。安装的步骤也比较简单。

正点原子 I.MX6U 出厂系统的 OpenCV 版本为 3.1.0。也不一定非要与正点原子 I.MX6U 出厂系统里的 OpenCV 相同版本，我们只是在 Ubuntu 上运行 OpenCV。其中用到的 API 在 3.1.0 版本与 3.4.1 版本基本没有什么区别。3.1.0 版本的 OpenCV 与 3.4.1 版本的 OpenCV 绝大多数核心 API 都相同，不必要担心找不到相同的 API。

这里笔者选择安装到 OpenCV 版本为 3.4.1 版本，因为 3.1.0 版本在配置 cmake 时下载的第三方库因为网络的原因难下载，导致 cmake 配置不过去。(PS:如果不是因为编译不过去，笔者会选择与开发板相同的版本的 OpenCV 的)。

进入 OpenCV 的官网 https://opencv.org/releases。下载 3.4.1 版本的 OpenCV，如下图。我们选择 Sources(源码)进行下载。



右键选择复制下载链接，用迅雷下载会比较快。下载完成后我们拷贝下载的文件到 Ubuntu 上进行解压。或者直接在正点原子的 I.MX6U 的光盘资料路径下找到开发板光盘 A-基础资料/1、例程源码/7、第三方库源码/opencv-3.4.1.tar.gz。然后拷贝到 Ubuntu 下。

如下图我们已经下载好文件，并拷贝下载好的文件到 Ubuntu 的家目录下。

```
alientek@ubuntu:~$ ls opencv-3.4.1.tar.gz
opencv-3.4.1.tar.gz
alientek@ubuntu:~$
```

执行下面的指令进行解压。解压将会得到一个 opencv-3.4.1 文件夹，我们使用 cd 指令进入此文件夹。

```
tar xf opencv-3.4.1.tar.gz
cd opencv-3.4.1
```

```
alientek@ubuntu:~$ tar xf opencv-3.4.1.tar.gz
alientek@ubuntu:~$ cd opencv-3.4.1/
alientek@ubuntu:~/opencv-3.4.1$
```

安装 cmake，用于生成编译 OpenCV 所需要的文件。

```
sudo apt-get install cmake
```

新建一个 build 目录，并进入，用于编译生成的文件。

```
mkdir build
cd build
```

```
alientek@ubuntu:~/opencv-3.4.1$ mkdir build
alientek@ubuntu:~/opencv-3.4.1$ cd build/
alientek@ubuntu:~/opencv-3.4.1/build$
```

执行 cmake 配置编译。注意下面的指令“..”不要漏了！这里表示上一层目录。cmake 会从上一层目录下找配置项，并配置到当前目录。

```
cmake ..
```

在配置的过程中 cmake 会下载一些库，如 ippicv_2017u3_lnx_intel64_general_20170822.tgz，需要一段时间，请等待，如果不能下载成功请重复尝试。

cmake 配置成功如下图。

```
--
--   Python (for build):           /usr/bin/python3
--
--   Java:
--     ant:                        NO
--     JNI:                        NO
--     Java wrappers:              NO
--     Java tests:                 NO
--
--   Matlab:                       NO
--
--   Install to:                   /usr/local
-- -----------------------------------------------------------------
--
-- Configuring done
-- Generating done
-- Build files have been written to: /home/alientek/opencv-3.4.1/build
alientek@ubuntu:~/opencv-3.4.1/build$
alientek@ubuntu:~/opencv-3.4.1/build$ ls
3rdparty        CMakeDownloadLog.txt   CMakeVars.txt            CTestTestfile.cmake  data     lib       OpenCVConfig.cmake             opencv_tests_config.hpp
apps            CMakeFiles             configured               custom_hal.hpp       doc      Makefile  OpenCVConfig-version.cmake     test-reports
bin             cmake_install.cmake    CPackConfig.cmake        cvconfig.h           include  modules   OpenCVGenPkgConfig.info.cmake  unix-install
CMakeCache.txt  cmake_uninstall.cmake  CPackSourceConfig.cmake  cv_cpu_config.h      junk     opencv2   OpenCVModules.cmake            version_string.tmp
alientek@ubuntu:~/opencv-3.4.1/build$ _
```

执行 make 开始编译。输入下面的指令。

```
make -j 16                    // 以实际分配给虚拟机的核心数为准，最佳为分配给虚拟机核心数据的 2 倍。
```

笔者的虚拟机最大分配了 16 个核心，笔者个人的电脑并不快，就是核心多，所以编译就快。编译完成耗时约 5 分钟。不要只输入 make，否则将编译很久！需要加参数 -j n，n 请根据个人虚拟机的实际情况。

```
alientek@ubuntu:~/opencv-3.4.1/build$ make -j 16
Scanning dependencies of target zlib
Scanning dependencies of target libwebp
Scanning dependencies of target libjasper
Scanning dependencies of target opencv_perf_core_pch_dephelp
Scanning dependencies of target ittnotify
Scanning dependencies of target libjpeg
Scanning dependencies of target opencv_imgproc_pch_dephelp
Scanning dependencies of target gen-pkgconfig
Scanning dependencies of target opencv_videoio_pch_dephelp
Scanning dependencies of target opencv_test_core_pch_dephelp
Scanning dependencies of target libprotobuf
Scanning dependencies of target opencv_ts_pch_dephelp
Scanning dependencies of target opencv_core_pch_dephelp
Scanning dependencies of target opencv_imgcodecs_pch_dephelp
Scanning dependencies of target opencv_flann_pch_dephelp
[  0%] Building CXX object modules/imgproc/CMakeFiles/opencv_imgproc_pch_dephelp.dir/opencv_imgproc_pch_dephelp.cxx.o
[  0%] Building C object 3rdparty/ittnotify/CMakeFiles/ittnotify.dir/src/ittnotify/ittnotify_static.c.o
[  1%] Building CXX object modules/core/CMakeFiles/opencv_perf_core_pch_dephelp.dir/opencv_perf_core_pch_dephelp.cxx.o
[  1%] Building CXX object modules/ts/CMakeFiles/opencv_ts_pch_dephelp.dir/opencv_ts_pch_dephelp.cxx.o
[  1%] Building CXX object modules/core/CMakeFiles/opencv_test_core_pch_dephelp.dir/opencv_test_core_pch_dephelp.cxx.o
[  1%] Generate opencv.pc
Scanning dependencies of target opencv_highgui_pch_dephelp
[  1%] Building C object 3rdparty/libjpeg/CMakeFiles/libjpeg.dir/jaricom.c.o
[  1%] Building C object 3rdparty/libwebp/CMakeFiles/libwebp.dir/src/dec/alpha_dec.c.o
[  1%] Building CXX object modules/flann/CMakeFiles/opencv_flann_pch_dephelp.dir/opencv_flann_pch_dephelp.cxx.o
[  1%] Building CXX object modules/imgcodecs/CMakeFiles/opencv_imgcodecs_pch_dephelp.dir/opencv_imgcodecs_pch_dephelp.cxx.o
```

```
[ 99%] Building CXX object modules/features2d/CMakeFiles/opencv_test_features2d.dir/test/test_keypoints.cpp.o
[ 99%] Built target opencv_perf_calib3d
[ 99%] Building CXX object modules/features2d/CMakeFiles/opencv_test_features2d.dir/test/test_main.cpp.o
[ 99%] Building CXX object modules/calib3d/CMakeFiles/opencv_test_calib3d.dir/test/test_fundam.cpp.o
[ 99%] Building CXX object modules/calib3d/CMakeFiles/opencv_test_calib3d.dir/test/test_homography_decomp.cpp.o
[100%] Building CXX object modules/calib3d/CMakeFiles/opencv_test_calib3d.dir/test/test_homography.cpp.o
[100%] Building CXX object modules/calib3d/CMakeFiles/opencv_test_calib3d.dir/test/test_main.cpp.o
[100%] Building CXX object modules/features2d/CMakeFiles/opencv_test_features2d.dir/test/test_matchers_algorithmic.cpp.o
[100%] Building CXX object modules/calib3d/CMakeFiles/opencv_test_calib3d.dir/test/test_modelest.cpp.o
[100%] Building CXX object modules/features2d/CMakeFiles/opencv_test_features2d.dir/test/test_mser.cpp.o
[100%] Building CXX object modules/calib3d/CMakeFiles/opencv_test_calib3d.dir/test/test_posit.cpp.o
[100%] Building CXX object modules/stitching/CMakeFiles/opencv_perf_stitching.dir/perf/perf_estimators.cpp.o
[100%] Building CXX object modules/calib3d/CMakeFiles/opencv_test_calib3d.dir/test/test_reproject_image_to_3d.cpp.o
[100%] Building CXX object modules/stitching/CMakeFiles/opencv_perf_stitching.dir/perf/perf_main.cpp.o
[100%] Building CXX object modules/calib3d/CMakeFiles/opencv_test_calib3d.dir/test/test_solvepnp_ransac.cpp.o
[100%] Building CXX object modules/calib3d/CMakeFiles/opencv_test_calib3d.dir/test/test_stereomatching.cpp.o
[100%] Building CXX object modules/calib3d/CMakeFiles/opencv_test_calib3d.dir/test/test_undistort.cpp.o
[100%] Building CXX object modules/features2d/CMakeFiles/opencv_test_features2d.dir/test/test_nearestneighbors.cpp.o
[100%] Building CXX object modules/calib3d/CMakeFiles/opencv_test_calib3d.dir/test/test_undistort_badarg.cpp.o
[100%] Building CXX object modules/features2d/CMakeFiles/opencv_test_features2d.dir/test/test_orb.cpp.o
[100%] Building CXX object modules/stitching/CMakeFiles/opencv_perf_stitching.dir/perf/perf_matchers.cpp.o
[100%] Building CXX object modules/stitching/CMakeFiles/opencv_perf_stitching.dir/perf/perf_stich.cpp.o
[100%] Building CXX object modules/calib3d/CMakeFiles/opencv_test_calib3d.dir/test/test_undistort_points.cpp.o
[100%] Linking CXX executable ../../bin/opencv_test_features2d
[100%] Built target opencv_test_features2d
[100%] Linking CXX executable ../../bin/opencv_test_calib3d
[100%] Built target opencv_test_calib3d
[100%] Linking CXX executable ../../bin/opencv_perf_stitching
[100%] Built target opencv_perf_stitching
alientek@ubuntu:~/opencv-3.4.1/build$
```

执行下面的指令安装，安装到系统目录，需要加 sudo 权限。

```
sudo make install
```

```
alientek@ubuntu:~/opencv-3.4.1/build$ sudo make install
[sudo] alientek 的密码：
[  0%] Built target gen-pkgconfig
[  2%] Built target zlib
[  5%] Built target libjpeg
[  8%] Built target libtiff
[ 16%] Built target libwebp
[ 19%] Built target libjasper
[ 21%] Built target libpng
[ 26%] Built target IlmImf
[ 31%] Built target libprotobuf
[ 31%] Built target ittnotify
[ 31%] Built target opencv_test_core_pch_dephelp
[ 31%] Built target opencv_core_pch_dephelp
```

安装完成如下。可以看到库被安装到/usr/local/lib 下，头文件被安装在/usr/local/include 下。

```
-- Installing: /usr/local/share/OpenCV/lbpcascades/lbpcascade_frontalcatface.xml
-- Installing: /usr/local/share/OpenCV/lbpcascades/lbpcascade_frontalface.xml
-- Installing: /usr/local/share/OpenCV/lbpcascades/lbpcascade_frontalface_improved.xml
-- Installing: /usr/local/share/OpenCV/lbpcascades/lbpcascade_profileface.xml
-- Installing: /usr/local/share/OpenCV/lbpcascades/lbpcascade_silverware.xml
-- Installing: /usr/local/bin/opencv_traincascade
-- Set runtime path of "/usr/local/bin/opencv_traincascade" to "/usr/local/lib"
-- Installing: /usr/local/bin/opencv_createsamples
-- Set runtime path of "/usr/local/bin/opencv_createsamples" to "/usr/local/lib"
-- Installing: /usr/local/bin/opencv_annotation
-- Set runtime path of "/usr/local/bin/opencv_annotation" to "/usr/local/lib"
-- Installing: /usr/local/bin/opencv_visualisation
-- Set runtime path of "/usr/local/bin/opencv_visualisation" to "/usr/local/lib"
-- Installing: /usr/local/bin/opencv_interactive-calibration
-- Set runtime path of "/usr/local/bin/opencv_interactive-calibration" to "/usr/local/lib"
-- Installing: /usr/local/bin/opencv_version
-- Set runtime path of "/usr/local/bin/opencv_version" to "/usr/local/lib"
alientek@ubuntu:~/opencv-3.4.1/build$
```

我们只需要知道安装的库路径和头文件路径即可在 Qt 里调用 Ubuntu 安装的 OpenCV。头文件作用来编写程序，库路径用来运行程序时调用。我们只要在 Qt 的 pro 项目文件里指定这两个路径即可。

19.3 应用实例

请根据【正点原子】I.MX6U 出厂系统 Qt 交叉编译环境搭建 V1.x.pdf(x >= 6)的文档搭建好 I.MX6U 的交叉编译环境。交叉编译工具链里已经有 OpenCV，所以我们只要在我们搭建的交叉环境下就可以调用 OpenCV 的相关 API 进行编写 Qt 项目了。

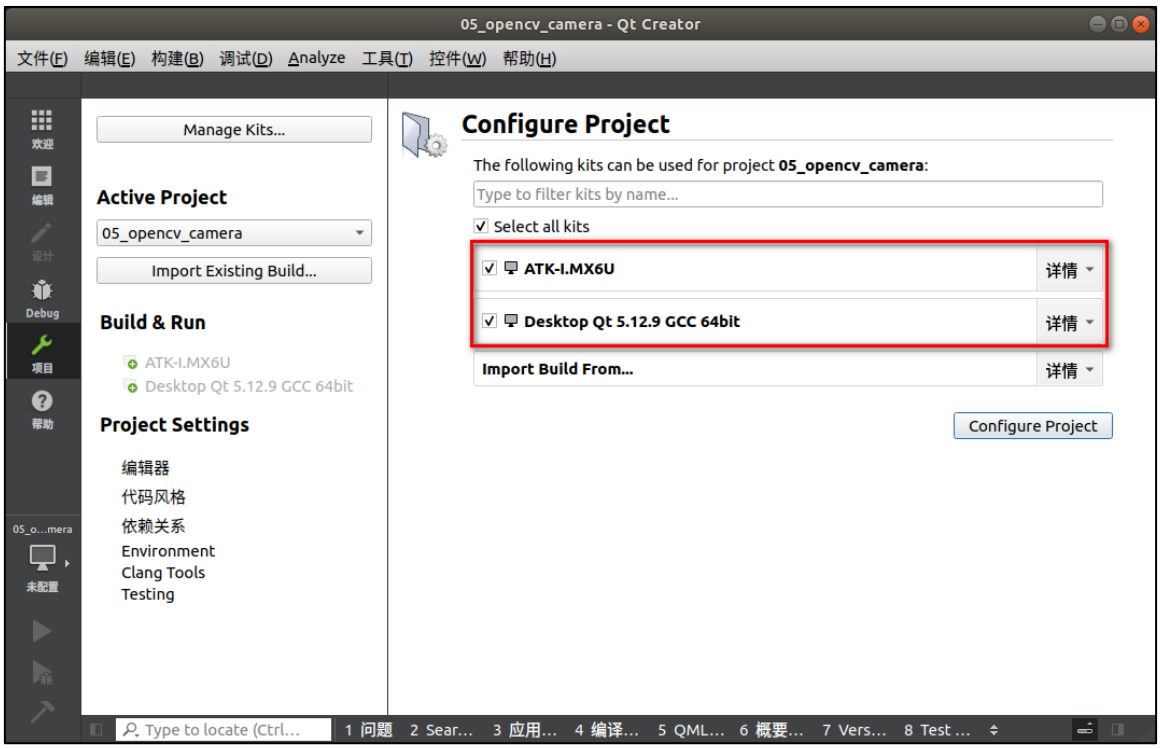
项目简介：Qt 加 OpenCV 打开摄像头采集图像并拍照。

例 05_opencv_camera，Qt Camera 编程（难度：**较难**）。项目路径为 Qt/3/05_opencv_camera。

用脚本打开 Qt Creator，必须按出厂系统 Qt 交叉编译环境搭建 V1.x.pdf(x >= 6)的文档搭建好交叉编译环境，用脚本启动时，脚本有相应的环境变量，编译时会用到。

```
/opt/Qt5.12.9/Tools/QtCreator/bin/qtcreator.sh &
```

编写程序应使用我们搭建的 ATK-I.MX6U 套件编写，否则若选择了 Desktop Qt 5.12.9 GCC 64bit，如果我们的 Ubuntu 没有安装 OpenCV 就会使用不了 OpenCV。如果您在 19.2 小节已经安装过 OpenCV，那么下面两个套件都可一起选。本次笔者两个一起选，因为笔者有 USB 摄像头可以在 Ubutnu 上使用 OpenCV 测试，编写的程序交叉编译后在 I.MX6U 开发板使用 USB 免驱摄像头或者正点原子 OV5640/OV7725(不带 FIFO 款)/OV2640 测试成功！



下面开始编写程序。首先我们需要在项目 pro 文件添加 OpenCV 库的支持及头文件路径。

05_opencv_camera.pro 文件如下，添加以下内容，这里主要是判断交叉编译器的类型，然后链接到不同的头文件路径与库。

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# The following define makes your compiler emit warnings if you use
# any Qt feature that has been marked deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0

TARGET_ARCH = $${QT_ARCH}
contains(TARGET_ARCH, arm){
    CONFIG += link_pkgconfig
```

```
21      PKGCONFIG += opencv
22      INCLUDEPATH +=
/opt/fsl-imx-x11/4.1.15-2.1.0/sysroots/cortexa7hf-neon-poky-linux-gnuea
bi/usr/include
23  } else {
24      LIBS += -L/usr/local/lib \
25              -lopencv_core \
26              -lopencv_highgui \
27              -lopencv_imgproc \
28              -lopencv_videoio \
29              -lopencv_imgcodecs
30
31      #INCLUDEPATH 可写可不写，系统会到找到此路径
32      INCLUDEPATH += /usr/local/include
33  }
34
35  SOURCES += \
36      camera.cpp \
37      main.cpp \
38      mainwindow.cpp
39
40  HEADERS += \
41      camera.h \
42      mainwindow.h
43
44  # Default rules for deployment.
45  qnx: target.path = /tmp/$${TARGET}/bin
46  else: unix:!android: target.path = /opt/$${TARGET}/bin
47  !isEmpty(target.path): INSTALLS += target
```

第 18 行，获取编译器的类型。

第 19 行，判断交叉编译器的类型是否为 arm。

第 22 行，arm 对应 opencv 的头文件路径，可以不写，编译不会报错，但是我们想查看对应的头文件，就不得不包括这个路径了，否则跳转不过去！

第 24~32 行，添加库的支持。-L 后面指的是库文件路径，-l 后面的是相关库参数（l 是大字母“L”的小写字母“l”，不是一），如果不会写库的名称，可以参考 Ubuntu 的 OpenCV 安装路径下的/usr/local/lib/pkgconfig/opencv.pc 文件。

camera.h 文件，此文件声明了一个 Camera 类，其内容如下，比较简单。

```
9   #ifndef CAMERA_H
10  #define CAMERA_H
11
12  #include <QImage>
13  #include <QTimer>
```

```
14  /* 使用命名空间 cv 下的 VideoCapture 与 Mat 类 */
15  namespace cv {
16  class VideoCapture;
17  class Mat;
18  }
19
20  class Camera : public QObject
21  {
22      Q_OBJECT
23  public:
24      explicit Camera(QObject *parent = nullptr);
25      ~Camera();
26
27  signals:
28      /* 声明信号，用于传递有图片信号时显示图像 */
29      void readyImage(const QImage&);
30
31  public slots:
32      /* 用于开启定时器 */
33      bool cameraProcess(bool);
34
35      /* 选择摄像头 */
36      void selectCameraDevice(int);
37
38  private slots:
39      /* 定时器时间到处理函数，发送图像数据信号 */
40      void timerTimeOut();
41
42  private:
43      /* 声明 OpenCV 的 cv 命名空间下的 VideoCapture 对象 */
44      cv::VideoCapture * capture;
45
46      /* 定时器 */
47      QTimer * timer;
48
49      /* 图像转换处理函数 */
50      QImage matToQImage(const cv::Mat&);
51  };
52
53  #endif // CAMERA_H
54
```

camera.cpp 类的定义如下。

```
9   #include "camera.h"
```

```
10  #include "opencv2/core/core.hpp"
11  #include "opencv2/highgui/highgui.hpp"
12  #include <QImage>
13  #include <QDebug>
14
15  Camera::Camera(QObject *parent) :
16      QObject(parent)
17  {
18      /* 实例化 */
19      capture = new cv::VideoCapture();
20      timer = new QTimer(this);
21
22      /* 信号槽连接 */
23      connect(timer, SIGNAL(timeout()), this, SLOT(timerTimeOut()));
24  }
25
26  Camera::~Camera()
27  {
28      delete capture;
29      capture = NULL;
30  }
31
32  void Camera::selectCameraDevice(int index)
33  {
34      /* 如果有其他摄像头打开了，先释放 */
35      if (capture->isOpened()) {
36          capture->release();
37      }
38
39      /* 打开摄像头设备 */
40      capture->open(index);
41  }
42
43  bool Camera::cameraProcess(bool bl)
44  {
45      if (bl) {
46          /* 为什么是 33？1000/33 约等于 30 帧，也就是一秒最多显示 30 帧  */
47          timer->start(33);
48      } else {
49          timer->stop();
50      }
51      /* 返回摄像头的状态 */
52      return capture->isOpened();
```

```
53  }
54
55  void Camera::timerTimeOut()
56  {
57      /* 如果摄像头没有打开，停止定时器，返回 */
58      if (!capture->isOpened()) {
59          timer->stop();
60          return;
61      }
62
63      static cv::Mat frame;
64      *capture >> frame;
65      if (frame.cols)
66          /* 发送图片信号 */
67          emit readyImage(matToQImage(frame));
68  }
69
70  QImage Camera::matToQImage(const cv::Mat &img)
71  {
72      /* USB 摄像头和 OV5640 等都是 RGB 三通道，不考虑单/四通道摄像头 */
73      if(img.type() == CV_8UC3) {
74          /* 得到图像的的首地址 */
75          const uchar *pimg = (const uchar*)img.data;
76
77          /* 以 img 构造图片 */
78          QImage qImage(pimg, img.cols, img.rows, img.step,
79                        QImage::Format_RGB888);
80
81          /* 在不改变实际图像数据的条件下，交换红蓝通道 */
82          return qImage.rgbSwapped();
83      }
84
85      /* 返回 QImage */
86      return QImage();
87  }
```

mainwindow.h 头文件代码如下。

```
/*************************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   05_opencv_camera
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
```

```
 * @date          2021-03-17
 ************************************************************************/
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QComboBox>
#include <QPushButton>
#include <QVBoxLayout>
#include <QLabel>
#include <QScrollArea>
#include <QDebug>

class Camera;

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    /* 主容器，Widget 也可以当作一种容器 */
    QWidget *mainWidget;

    /* 滚动区域，方便开发高分辨率 */
    QScrollArea *scrollArea;

    /* 将采集到的图像使用 Widget 显示 */
    QLabel *displayLabel;

    /* 界面右侧区域布局 */
    QHBoxLayout *hboxLayout;

    /* 界面右侧区域布局 */
    QVBoxLayout *vboxLayout;

    /* 界面右侧区域容器 */
    QWidget *rightWidget;
```

```
42
43      /* 界面右侧区域显示拍照的图片 */
44      QLabel *photoLabel;
45
46      /* 界面右侧区域摄像头设备下拉选择框 */
47      QComboBox *comboBox;
48
49      /* 两个按钮，一个为拍照按钮，另一个是开启摄像头按钮 */
50      QPushButton *pushButton[2];
51
52      /* 拍照保存的照片 */
53      QImage saveImage;
54
55      /* 摄像头设备 */
56      Camera *camera;
57
58      /* 布局初始化 */
59      void layoutInit();
60
61      /* 扫描是否存在摄像头 */
62      void scanCameraDevice();
63
64  private slots:
65      /* 显示图像 */
66      void showImage(const QImage&);
67
68      /* 设置按钮文本 */
69      void setButtonText(bool);
70
71      /* 保存照片到本地 */
72      void saveImageToLocal();
73  };
74  #endif // MAINWINDOW_H
```

mainwindow.cpp 源文件代码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   05_opencv_camera
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-03-17
*****************************************************************/
```

```
1   #include "mainwindow.h"
2   #include <QGuiApplication>
3   #include <QScreen>
4   #include <QFile>
5   #include <QPixmap>
6   #include <QBuffer>
7   #include "camera.h"
8
9   MainWindow::MainWindow(QWidget *parent)
10      : QMainWindow(parent)
11  {
12      /* 布局初始化 */
13      layoutInit();
14
15      /* 扫描摄像头 */
16      scanCameraDevice();
17  }
18
19  MainWindow::~MainWindow()
20  {
21  }
22
23  void MainWindow::layoutInit()
24  {
25      /* 获取屏幕的分辨率，Qt 官方建议使用这
26       * 种方法获取屏幕分辨率，防上多屏设备导致对应不上
27       * 注意，这是获取整个桌面系统的分辨率
28       */
29      QList <QScreen *> list_screen =  QGuiApplication::screens();
30
31      /* 如果是 ARM 平台，直接设置大小为屏幕的大小 */
32  #if __arm__
33      /* 重设大小 */
34      this->resize(list_screen.at(0)->geometry().width(),
35                   list_screen.at(0)->geometry().height());
36  #else
37      /* 否则则设置主窗体大小为 800x480 */
38      this->resize(800, 480);
39  #endif
40
41      /* 实例化与布局，常规操作 */
42      mainWidget = new QWidget();
43      photoLabel = new QLabel();
```

```
44      rightWidget = new QWidget();
45      comboBox = new QComboBox();
46      pushButton[0] = new QPushButton();
47      pushButton[1] = new QPushButton();
48      scrollArea = new QScrollArea();
49      displayLabel = new QLabel(scrollArea);
50      vboxLayout = new QVBoxLayout();
51      hboxLayout = new QHBoxLayout();
52
53      vboxLayout->addWidget(photoLabel);
54      vboxLayout->addWidget(comboBox);
55      vboxLayout->addWidget(pushButton[0]);
56      vboxLayout->addWidget(pushButton[1]);
57
58      rightWidget->setLayout(vboxLayout);
59
60      hboxLayout->addWidget(scrollArea);
61      hboxLayout->addWidget(rightWidget);
62      mainWidget->setLayout(hboxLayout);
63
64      this->setCentralWidget(mainWidget);
65
66      pushButton[0]->setMaximumHeight(40);
67      pushButton[0]->setMaximumWidth(200);
68
69      pushButton[1]->setMaximumHeight(40);
70      pushButton[1]->setMaximumWidth(200);
71
72      comboBox->setMaximumHeight(40);
73      comboBox->setMaximumWidth(200);
74      photoLabel->setMaximumSize(100, 75);
75      scrollArea->setMinimumWidth(this->width()
76                                  - comboBox->width());
77
78      /* 显示图像最大画面为 xx */
79      displayLabel->setMinimumWidth(scrollArea->width() * 0.75);
80      displayLabel->setMinimumHeight(scrollArea->height() * 0.75);
81      scrollArea->setWidget(displayLabel);
82
83      /* 居中显示 */
84      scrollArea->setAlignment(Qt::AlignCenter);
85
86      /* 自动拉伸 */
87      photoLabel->setScaledContents(true);
```

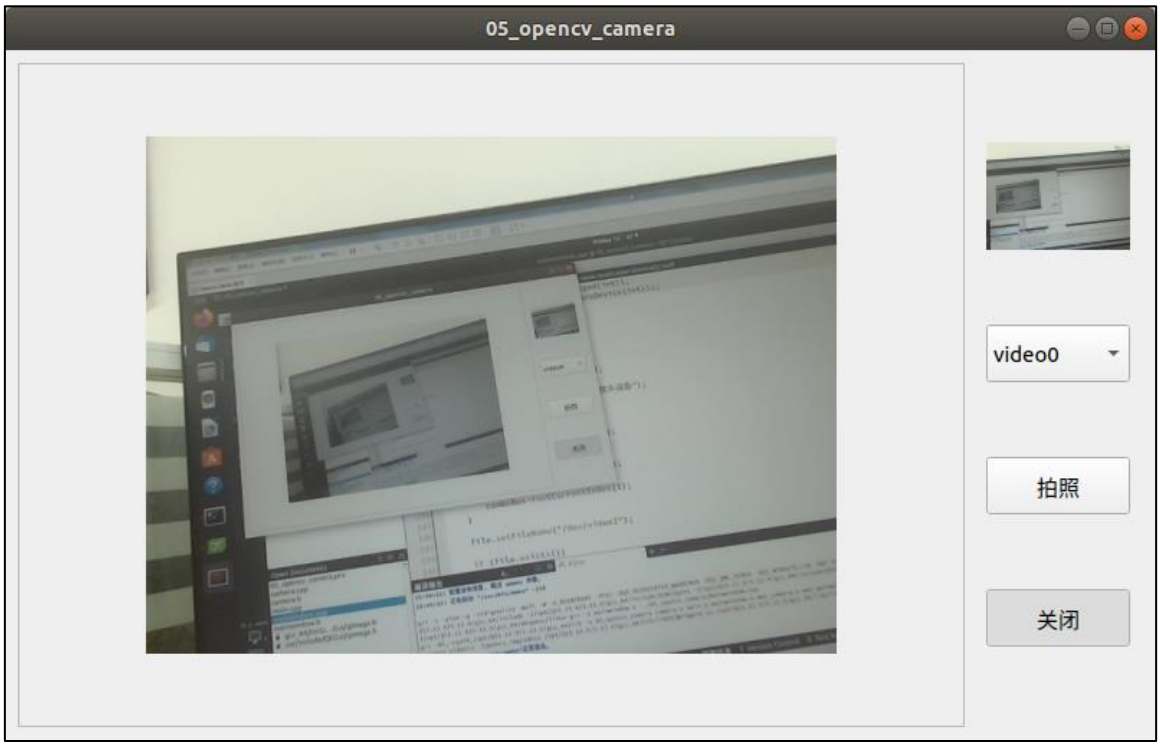
```
88      displayLabel->setScaledContents(true);
89
90      /* 设置一些属性 */
91      pushButton[0]->setText("拍照");
92      pushButton[0]->setEnabled(false);
93      pushButton[1]->setText("开始");
94      pushButton[1]->setCheckable(true);
95
96      /* 摄像头 */
97      camera = new Camera(this);
98
99      /* 信号连接槽 */
100     connect(camera, SIGNAL(readyImage(QImage)),
101             this, SLOT(showImage(QImage)));
102     connect(pushButton[1], SIGNAL(clicked(bool)),
103             camera, SLOT(cameraProcess(bool)));
104     connect(pushButton[1], SIGNAL(clicked(bool)),
105             this, SLOT(setButtonText(bool)));
106     connect(pushButton[0], SIGNAL(clicked()),
107             this, SLOT(saveImageToLocal()));
108
109 }
110
111 void MainWindow::scanCameraDevice()
112 {
113     /* 如果是 Windows 系统，一般是摄像头 0 */
114 #if win32
115     comboBox->addItem("windows 摄像头 0");
116     connect(comboBox,
117             SIGNAL(currentIndexChanged(int)),
118             camera, SLOT(selectCameraDevice(int)));
119 #else
120     /* QFile 文件指向/dev/video0 */
121     QFile file("/dev/video0");
122
123     /* 如果文件存在 */
124     if (file.exists())
125         comboBox->addItem("video0");
126     else {
127         displayLabel->setText("无摄像头设备");
128         return;
129     }
130
```

```
131     file.setFileName("/dev/video1");
132
133     if (file.exists()) {
134         comboBox->addItem("video1");
135         /* 开发板 ov5640 等设备是 1 */
136         comboBox->setCurrentIndex(1);
137     }
138
139     file.setFileName("/dev/video2");
140
141     if (file.exists())
142         /* 开发板 USB 摄像头设备是 2 */
143         comboBox->addItem("video2");
144
145 #if !__arm__
146     /* ubuntu 的 USB 摄像头一般是 0 */
147     comboBox->setCurrentIndex(0);
148 #endif
149
150     connect(comboBox,
151             SIGNAL(currentIndexChanged(int)),
152             camera, SLOT(selectCameraDevice(int)));
153 #endif
154 }
155
156 void MainWindow::showImage(const QImage &image)
157 {
158     /* 显示图像 */
159     displayLabel->setPixmap(QPixmap::fromImage(image));
160     saveImage = image;
161
162     /* 判断图像是否为空，空则设置拍照按钮不可用 */
163     if (!saveImage.isNull())
164         pushButton[0]->setEnabled(true);
165     else
166         pushButton[0]->setEnabled(false);
167 }
168
169 void MainWindow::setButtonText(bool bl)
170 {
171     if (bl) {
172         /* 设置摄像头设备 */
173         camera->selectCameraDevice(comboBox->currentIndex());
```

```
174         pushButton[1]->setText("关闭");
175     } else {
176         /* 若关闭了摄像头则禁用拍照按钮 */
177         pushButton[0]->setEnabled(false);
178         pushButton[1]->setText("开始");
179     }
180 }
181
182 void MainWindow::saveImageToLocal()
183 {
184     /* 判断图像是否为空 */
185     if (!saveImage.isNull()) {
186         QString fileName =
187                 QCoreApplication::applicationDirPath() + "/test.png";
188         qDebug()<<"正在保存"<<fileName<<"图片,请稍候..."<<endl;
189
190         /* save(arg1, arg2, arg3)重载函数，arg1 代表路径文件名，
191          * arg2 保存的类型，arg3 代表保存的质量等级 */
192         saveImage.save(fileName, "PNG", -1);
193
194         /* 设置拍照的图像为显示在 photoLabel 上 */
195         photoLabel->setPixmap(QPixmap::fromImage(QImage(fileName)));
196
197         qDebug()<<"保存完成！"<<endl;
198     }
199 }
200
```

第 111~154 行，判断 linux 下的设备/dev/video*。细心的同学发现，这个程序是 Linux 下用的。当然 Windows 也是可以使用 OpenCV 的，需要自己修改 pro 文件链接到 Windows 的 OpenCV 库-L 需要修改为-LD，Windows 下的库文件是 dll 类型，此外不考虑 macOS 系统，具体情况笔者没得实验。

19.4 程序运行效果



选择合适的摄像头设备，（注意如果在 Ubuntu 使用 USB 摄像头，需要设置 USB 的兼容性为 3.0 反之 2.0，具体需要看不同摄像头设备，点击连接摄像头到虚拟机。可以先使用 Ubuntu18.04 自带的茄子拍照软件，检测摄像头是否可用）。点击拍照，可以看程序输出的 Debug 信息，保存照片的路径为当前可执行程序的路径，保存照片名称为 test.png，右上角显示保存照片的缩略图，再次点击拍照则会替换已经保存过的照片。若想要保存多个照片可自行设计。

若在正点原子 I.MX6U 开发板上运行此程序，先插上摄像头，确保摄像头能用，注意不要选择 video0，video0 是 NXP 的 pxp 驱动产生的节点，不是摄像头，否则会报错。I.MX6U 开发板是单核 A7 的 CPU，性能有限，所以流畅度一般，还可以。但是在保存照片时会比 PC 电脑慢好多，不过也不能太勉强这个 6ULL 芯片了啦，能拍照已经不错了，或者大家可以对保存照片步骤进行优化，开启一个线程进行优化，剩下的交给大家了，大家可以的。

OpenCV 在不设置摄像头分辨率时会采用默认分辨率 640*480 30fps(绝大多数摄像头都是支持这个分辨率)。USB 免驱摄像头可以使用下面的方法来设置分辨率。

```
capture->open(1);
capture ->set(CV_CAP_PROP_FRAME_WIDTH, 320);
capture ->set(CV_CAP_PROP_FRAME_HEIGHT, 240);
```

但是正点原子 6ULL 开发板上的 OV5640/OV2640/OV7725（不带 FIFO 款）摄像头就不可以直接使用些方法设置采集分辨率了，因为驱动里设置分辨率的方法与标准的 V4L2 设置分辨有些差异。也可以直接使用正点原子 I.MX6U 里的摄像头采集分辨率设置软件 camera_settings 直接设置，注意在处理图像中的 matToQImage 函数需要以确认的分辨率进行转换使用了。

总结，想要在 Qt 中使用 OpenCV，那么我们的开发板文件系统里或者 Ubuntu 系统必须要有 OpenCV 的库。对于某些非通用的 USB 摄像头来说，因为驱动层限制不能直接使用 Qt 自带的 QCamera 类。

第二十章 USB Bluetooth

Qt 官方提供了蓝牙的相关类和 API 函数，也提供了相关的例程给我们参考。笔者根据 Qt 官方的例程编写出适合我们 Ubuntu 和正点原子 I.MX6U 开发板的例程。注意 Windows 上不能使用 Qt 的蓝牙例程，因为底层需要有 BlueZ 协议栈，而 Windows 没有。Windows 可能需要去移植。笔者就不去探究了。确保我们正点原子 I.MX6U 开发板与 Ubuntu 可用即可，所以大家还是老实的用 Ubuntu 来开发吧！

20.1 资源简介

在正点原子 IMX6U 开发板上虽然没有带板载蓝牙，但是可以外接免驱 USB 蓝牙，直接在 USB 接口插上一个 USB 蓝牙模块就可以进行本章节的实验了。详细请看【正点原子】I.MX6U 用户快速体验 V1.x.pdf 的第 3.29 小节蓝牙测试，先了解蓝牙是如何在 Linux 上如何使用的，切记先看正点原子快速体验文档，了解用哪种蓝牙芯片，和怎么测试蓝牙的。本 Qt 教程就不再介绍了。

20.2 应用实例

项目简介：Qt 蓝牙聊天。将蓝牙设置成一个服务器，或者用做客户端，连接手机即可通信。

例 06_bluetooth_chat，Qt 蓝牙聊天（难度：难）。项目路径为 Qt/3/06_bluetooth_chat。

Qt 使用蓝牙，需要在项目文件加上相应的蓝牙模块。添加的代码如下红色加粗部分。

06_bluetooth_chat.pro 文件代码如下。

```
1   QT      += core gui bluetooth
2
3   greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5   CONFIG += c++11
6
7   # The following define makes your compiler emit warnings if you use
8   # any Qt feature that has been marked deprecated (the exact warnings
9   # depend on your compiler). Please consult the documentation of the
10  # deprecated API in order to know how to port your code away from it.
11  DEFINES += QT_DEPRECATED_WARNINGS
12
13  # You can also make your code fail to compile if it uses deprecated APIs.
14  # In order to do so, uncomment the following line.
15  # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16  #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18  SOURCES += \
19      chatclient.cpp \
20      chatserver.cpp \
21      main.cpp \
22      mainwindow.cpp \
23      remoteselector.cpp
24
25  HEADERS += \
26      chatclient.h \
27      chatserver.h \
28      mainwindow.h \
```

```
29      remoteselector.h
30
31  # Default rules for deployment.
32  qnx: target.path = /tmp/$${TARGET}/bin
33  else: unix:!android: target.path = /opt/$${TARGET}/bin
34  !isEmpty(target.path): INSTALLS += target
```

第 18~29 行，可以看到我们的项目组成文件。一个客户端，一个服务端，一个主界面和一个远程选择蓝牙的文件。总的看起来有四大部分，下面就介绍这四大部分的文件。

chatclient.h 的代码如下。

```
    /*********************************************************************
    Copyright (C) 2015 The Qt Company Ltd.
    Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
    * @projectName   06_bluetooth_chat
    * @brief        chatclient.h
    * @author       Deng Zhimao
    * @email        1252699831@qq.com
    * @net          www.openedv.com
    * @date         2021-03-20
    ********************************************************************/
1   #ifndef CHATCLIENT_H
2   #define CHATCLIENT_H
3
4   #include <qbluetoothserviceinfo.h>
5   #include <QBluetoothSocket>
6   #include <QtCore/QObject>
7
8   QT_FORWARD_DECLARE_CLASS(QBluetoothSocket)
9
10  class ChatClient : public QObject
11  {
12      Q_OBJECT
13
14  public:
15      explicit ChatClient(QObject *parent = nullptr);
16      ~ChatClient();
17
18      /* 开启客户端 */
19      void startClient(const QBluetoothServiceInfo &remoteService);
20
21      /* 停止客户端 */
22      void stopClient();
23
24  public slots:
```

```
25      /* 发送消息 */
26      void sendMessage(const QString &message);
27
28      /* 主动断开连接 */
29      void disconnect();
30
31  signals:
32      /* 接收到消息信号 */
33      void messageReceived(const QString &sender, const QString &message);
34
35      /* 连接信号 */
36      void connected(const QString &name);
37
38      /* 断开连接信号 */
39      void disconnected();
40
41  private slots:
42      /* 从 socket 里读取消息 */
43      void readSocket();
44
45      /* 连接 */
46      void connected();
47
48  private:
49      /* socket 通信 */
50      QBluetoothSocket *socket;
51  };
52
53  #endif // CHATCLIENT_H
```

chatclient.h 文件主要是客户端的头文件，其中写一些接口，比如开启客户端，关闭客户端，接收信号与关闭信号等等。

chatclient.cpp 的代码如下。

```
/*****************************************************************
Copyright (C) 2015 The Qt Company Ltd.
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   06_bluetooth_chat
* @brief        chatclient.cpp
* @author       Deng Zhimao
* @email        1252699831@qq.com
* @net          www.openedv.com
* @date         2021-03-20
*****************************************************************/
```

```
1   #include "chatclient.h"
2   #include <qbluetoothsocket.h>
3
4   ChatClient::ChatClient(QObject *parent)
5       :   QObject(parent), socket(0)
6   {
7   }
8
9   ChatClient::~ChatClient()
10  {
11      stopClient();
12  }
13
14  /* 开启客户端 */
15  void ChatClient::startClient(const QBluetoothServiceInfo
&remoteService)
16  {
17      if (socket)
18          return;
19
20      // Connect to service
21      socket = new
QBluetoothSocket(QBluetoothServiceInfo::RfcommProtocol);
22      qDebug() << "Create socket";
23      socket->connectToService(remoteService);
24      qDebug() << "ConnectToService done";
25
26      connect(socket, SIGNAL(readyRead()),
27              this, SLOT(readSocket()));
28      connect(socket, SIGNAL(connected()),
29              this, SLOT(connected()));
30      connect(socket, SIGNAL(disconnected()),
31              this, SIGNAL(disconnected()));
32  }
33
34  /* 停止客户端 */
35  void ChatClient::stopClient()
36  {
37      delete socket;
38      socket = 0;
39  }
40
41  /* 从 Socket 读取消息 */
```

```
42  void ChatClient::readSocket()
43  {
44      if (!socket)
45          return;
46
47      while (socket->canReadLine()) {
48          QByteArray line = socket->readLine();
49          emit messageReceived(socket->peerName(),
50                               QString::fromUtf8(line.constData(),
51                                                 line.length()));
52      }
53  }
54
55  /* 发送的消息 */
56  void ChatClient::sendMessage(const QString &message)
57  {
58      qDebug()<<"Sending data in client: " + message;
59
60      QByteArray text = message.toUtf8() + '\n';
61      socket->write(text);
62  }
63
64  /* 主动连接 */
65  void ChatClient::connected()
66  {
67      emit connected(socket->peerName());
68  }
69
70  /* 主动断开连接*/
71  void ChatClient::disconnect() {
72      qDebug()<<"Going to disconnect in client";
73      if (socket) {
74          qDebug()<<"diconnecting...";
75          socket->close();
76      }
77  }
```

chatclient.cpp 文件主要是客户端的 chatclient.h 头文件的实现。代码参考 Qt 官方 btchat 例子，代码比较长，也有相应的注释了，大家自由查看。主要我们关注的是下面的代码。

第 15~32 行，我们需要开启客户端模式，那么我们需要将扫描服务器（手机蓝牙）的结果，实例化一个蓝牙 socket，使用 socket 连接传入来的服务器信息，即可将本地蓝牙当作客户端，实现了客户端创建。

chatserver.h 代码如下。

```
/*****************************************************************
```

```
    Copyright (C) 2015 The Qt Company Ltd.
    Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
    * @projectName   06_bluetooth_chat
    * @brief        chatserver.h
    * @author       Deng Zhimao
    * @email        1252699831@qq.com
    * @net          www.openedv.com
    * @date         2021-03-20
    ****************************************************************/

1   #ifndef CHATSERVER_H
2   #define CHATSERVER_H
3
4   #include <qbluetoothserviceinfo.h>
5   #include <qbluetoothaddress.h>
6   #include <QtCore/QObject>
7   #include <QtCore/QList>
8   #include <QBluetoothServer>
9   #include <QBluetoothSocket>
10
11
12  class ChatServer : public QObject
13  {
14      Q_OBJECT
15
16  public:
17      explicit ChatServer(QObject *parent = nullptr);
18      ~ChatServer();
19
20      /* 开启服务端 */
21      void startServer(const QBluetoothAddress &localAdapter =
QBluetoothAddress());
22
23      /* 停止服务端 */
24      void stopServer();
25
26  public slots:
27      /* 发送消息 */
28      void sendMessage(const QString &message);
29
30      /* 服务端主动断开连接 */
31      void disconnect();
32
33  signals:
```

```
34      /* 接收到消息信号 */
35      void messageReceived(const QString &sender, const QString &message);
36
37      /* 客户端连接信号 */
38      void clientConnected(const QString &name);
39
40      /* 客户端断开连接信号 */
41      void clientDisconnected(const QString &name);
42
43  private slots:
44
45      /* 客户端连接 */
46      void clientConnected();
47
48      /* 客户端断开连接 */
49      void clientDisconnected();
50
51      /* 读 socket */
52      void readSocket();
53
54  private:
55      /* 使用 rfcomm 协议 */
56      QBluetoothServer *rfcommServer;
57
58      /* 服务器蓝牙信息 */
59      QBluetoothServiceInfo serviceInfo;
60
61      /* 用于保存客户端 socket */
62      QList<QBluetoothSocket *> clientSockets;
63
64      /* 用于保存客户端的名字 */
65      QList<QString> socketsPeername;
66  };
67
68  #endif // CHATSERVER_H
```

chatserver.h 文件主要是服务端的头文件，其中写一些接口，比如开启服务端，关闭服务端，接收信号与关闭信号等等。

chatserver.cpp 代码如下。

```
/*****************************************************************
Copyright (C) 2015 The Qt Company Ltd.
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   06_bluetooth_chat
```

```
    * @brief        chatserver.cpp
    * @author       Deng Zhimao
    * @email        1252699831@qq.com
    * @net          www.openedv.com
    * @date         2021-03-20
    ****************************************************************/

#include "chatserver.h"

#include <qbluetoothserver.h>
#include <qbluetoothsocket.h>
#include <qbluetoothlocaldevice.h>

static const QLatin1String
serviceUuid("e8e10f95-1a70-4b27-9ccf-02010264e9c8");
ChatServer::ChatServer(QObject *parent)
    :   QObject(parent), rfcommServer(0)
{
}

ChatServer::~ChatServer()
{
    stopServer();
}

/* 开启服务端，设置服务端使用 rfcomm 协议与 serviceInfo 的一些属性 */
void ChatServer::startServer(const QBluetoothAddress& localAdapter)
{
    if (rfcommServer)
        return;

    rfcommServer = new
QBluetoothServer(QBluetoothServiceInfo::RfcommProtocol, this);
    connect(rfcommServer, SIGNAL(newConnection()), this,
SLOT(clientConnected()));
    bool result = rfcommServer->listen(localAdapter);
    if (!result) {
        qWarning()<<"Cannot bind chat server
to"<<localAdapter.toString();
        return;
    }

```

```
32
//serviceInfo.setAttribute(QBluetoothServiceInfo::ServiceRecordHandle,
(uint)0x00010010);
33
34      QBluetoothServiceInfo::Sequence classId;
35
36
classId<<QVariant::fromValue(QBluetoothUuid(QBluetoothUuid::SerialPort)
);
37
serviceInfo.setAttribute(QBluetoothServiceInfo::BluetoothProfileDescrip
torList,
38                              classId);
39
40
classId.prepend(QVariant::fromValue(QBluetoothUuid(serviceUuid)));
41
42      serviceInfo.setAttribute(QBluetoothServiceInfo::ServiceClassIds,
classId);
43
serviceInfo.setAttribute(QBluetoothServiceInfo::BluetoothProfileDescrip
torList,classId);
44
45      serviceInfo.setAttribute(QBluetoothServiceInfo::ServiceName,
tr("Bt Chat Server"));
46
serviceInfo.setAttribute(QBluetoothServiceInfo::ServiceDescription,
47                              tr("Example bluetooth chat server"));
48      serviceInfo.setAttribute(QBluetoothServiceInfo::ServiceProvider,
tr("qt-project.org"));
49
50      serviceInfo.setServiceUuid(QBluetoothUuid(serviceUuid));
51
52      QBluetoothServiceInfo::Sequence publicBrowse;
53      publicBrowse<<
QVariant::fromValue(QBluetoothUuid(QBluetoothUuid::PublicBrowseGroup));
54      serviceInfo.setAttribute(QBluetoothServiceInfo::BrowseGroupList,
55                              publicBrowse);
56
57      QBluetoothServiceInfo::Sequence protocolDescriptorList;
58      QBluetoothServiceInfo::Sequence protocol;
59      protocol<<
QVariant::fromValue(QBluetoothUuid(QBluetoothUuid::L2cap));
60      protocolDescriptorList.append(QVariant::fromValue(protocol));
```

```
61      protocol.clear();
62      protocol<<
QVariant::fromValue(QBluetoothUuid(QBluetoothUuid::Rfcomm))
63              <<
QVariant::fromValue(quint8(rfcommServer->serverPort()));
64      protocolDescriptorList.append(QVariant::fromValue(protocol));
65
serviceInfo.setAttribute(QBluetoothServiceInfo::ProtocolDescriptorList,
66                               protocolDescriptorList);
67
68      serviceInfo.registerService(localAdapter);
69  }
70
71  /* 停止服务端 */
72  void ChatServer::stopServer()
73  {
74      // Unregister service
75      serviceInfo.unregisterService();
76
77      // Close sockets
78      qDeleteAll(clientSockets);
79
80      // Close server
81      delete rfcommServer;
82      rfcommServer = 0;
83  }
84
85  /* 主动断开连接 */
86  void ChatServer::disconnect()
87  {
88      qDebug()<<"Going to disconnect in server";
89
90      foreach (QBluetoothSocket *socket, clientSockets) {
91          qDebug()<<"sending data in server!";
92          socket->close();
93      }
94  }
95
96  /* 发送消息 */
97  void ChatServer::sendMessage(const QString &message)
98  {
99      qDebug()<<"Going to send message in server: " << message;
100     QByteArray text = message.toUtf8() + '\n';
101
```

```
102     foreach (QBluetoothSocket *socket, clientSockets) {
103         qDebug()<<"sending data in server!";
104         socket->write(text);
105     }
106     qDebug()<<"server sending done!";
107 }
108
109 /* 客户端连接 */
110 void ChatServer::clientConnected()
111 {
112     qDebug()<<"clientConnected";
113
114     QBluetoothSocket *socket = rfcommServer->nextPendingConnection();
115     if (!socket)
116         return;
117
118     connect(socket, SIGNAL(readyRead()), this, SLOT(readSocket()));
119     connect(socket, SIGNAL(disconnected()), this,
SLOT(clientDisconnected()));
120     clientSockets.append(socket);
121     socketsPeername.append(socket->peerName());
122     emit clientConnected(socket->peerName());
123 }
124
125 /* 客户端断开连接 */
126 void ChatServer::clientDisconnected()
127 {
128     QBluetoothSocket *socket = qobject_cast<QBluetoothSocket
*>(sender());
129     if (!socket)
130         return;
131
132     if (clientSockets.count() != 0) {
133         QString peerName;
134
135         if (socket->peerName().isEmpty())
136             peerName =
socketsPeername.at(clientSockets.indexOf(socket));
137         else
138             peerName = socket->peerName();
139
140         emit clientDisconnected(peerName);
141
142         clientSockets.removeOne(socket);
```

```
143         socketsPeername.removeOne(peerName);
144     }
145
146     socket->deleteLater();
147
148 }
149
150 /* 从 Socket 里读取数据 */
151 void ChatServer::readSocket()
152 {
153     QBluetoothSocket *socket = qobject_cast<QBluetoothSocket
*>(sender());
154     if (!socket)
155         return;
156
157     while (socket->bytesAvailable()) {
158         QByteArray line = socket->readLine().trimmed();
159         qDebug()<<QString::fromUtf8(line.constData(),
line.length())<<endl;
160         emit messageReceived(socket->peerName(),
161                              QString::fromUtf8(line.constData(),
line.length()));
162         qDebug()<<QString::fromUtf8(line.constData(),
line.length())<<endl;
163     }
164 }
```

chatserver.cpp 文件主要是服务端的 chatserver.h 头文件的实现。代码也是参考 Qt 官方 btchat 例子，代码比较长，也有相应的注释了，大家自由查看。主要我们关注的是下面的代码。

第 19~69 行，我们需要开启服务端模式，那么我们需要将本地的蓝牙 localAdapter 的地址传入，创建一个 QBluetoothServer 对象 rfcommServer。在 19 至 69 行代码很长，其中使用了 serviceInfo.setAttribute()设置了许多参数，这个流程是官方给出的流程，我们只需要了解下就可以了。大体流程：使用了 QBluetoothServiceInfo 类允许访问服务端蓝牙服务的属性，其中有设置蓝牙的 UUID 为文件开头定义的 serviceUuid，设置 serviceUuid 的目的是为了区分其他蓝牙，用于搜索此类型蓝牙，但是作用并不是很大，因为我们的手机并不一定开启了这个 uuid 标识。最后必须用 registerService()启动蓝牙。

第 36 行，转换串行端口（SerialPort），转换成 classId，然后再设置串行端口服务。通信原理就是串行端口连接到 RFCOMM server channel。（PS：蓝牙使用的协议多且复杂，本教程并不能清晰解释这种原理，如果有错误，欢迎指出）。

remoteselector.h 代码如下。

```
/****************************************************************
Copyright (C) 2015 The Qt Company Ltd.
```

```
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   06_bluetooth_chat
* @brief        remoteselector.h
* @author       Deng Zhimao
* @email        1252699831@qq.com
* @net          www.openedv.com
* @date         2021-03-20
*****************************************************************/

#ifndef REMOTESELECTOR_H
#define REMOTESELECTOR_H

#include <qbluetoothuuid.h>
#include <qbluetoothserviceinfo.h>
#include <qbluetoothservicediscoveryagent.h>
#include <QListWidgetItem>

/* 声明一个蓝牙适配器类 */
class RemoteSelector : public QObject
{
    Q_OBJECT

public:
    explicit RemoteSelector(QBluetoothAddress&,
                            QObject *parent = nullptr);
    ~RemoteSelector();

    /* 开启发现蓝牙 */
    void startDiscovery(const QBluetoothUuid &uuid);

    /* 停止发现蓝牙 */
    void stopDiscovery();

    /* 蓝牙服务 */
    QBluetoothServiceInfo service() const;

signals:
    /* 找到新服务 */
    void newServiceFound(QListWidgetItem*);

    /* 完成 */
    void finished();

```

```
35  private:
36      /* 蓝牙服务代理，用于发现蓝牙服务 */
37      QBluetoothServiceDiscoveryAgent *m_discoveryAgent;
38
39      /* 服务信息 */
40      QBluetoothServiceInfo m_serviceInfo;
41
42  private slots:
43      /* 服务发现完成 */
44      void serviceDiscovered(const QBluetoothServiceInfo &serviceInfo);
45
46      /* 蓝牙发现完成 */
47      void discoveryFinished();
48
49  public:
50      /* 键值类容器 */
51      QMap<QString, QBluetoothServiceInfo> m_discoveredServices;
52  };
53
54  #endif // REMOTESELECTOR_H
55
```

remoteselector.h 翻译成远程选择器，代码也是参考 Qt 官方 btchat 例子，这个头文件定义了开启蓝牙发现模式，蓝牙关闭发现模式，还有服务完成等等，代码有注释，请自由查看。

remoteselector.cpp 代码如下。

```
    /****************************************************************************
    Copyright (C) 2015 The Qt Company Ltd.
    Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
    * @projectName   06_bluetooth_chat
    * @brief         remoteselector.cpp
    * @author        Deng Zhimao
    * @email         1252699831@qq.com
    * @net           www.openedv.com
    * @date          2021-03-20
    ****************************************************************************/
1   #include "remoteselector.h"
2
3   /* 初始化本地蓝牙 */
4   RemoteSelector::RemoteSelector(QBluetoothAddress &localAdapter,
QObject *parent)
5       : QObject(parent)
6   {
7       m_discoveryAgent = new
QBluetoothServiceDiscoveryAgent(localAdapter);
```

```
8
9       connect(m_discoveryAgent,
SIGNAL(serviceDiscovered(QBluetoothServiceInfo)),
10              this, SLOT(serviceDiscovered(QBluetoothServiceInfo)));
11      connect(m_discoveryAgent, SIGNAL(finished()), this,
SLOT(discoveryFinished()));
12      connect(m_discoveryAgent, SIGNAL(canceled()), this,
SLOT(discoveryFinished()));
13  }
14
15  RemoteSelector::~RemoteSelector()
16  {
17      delete m_discoveryAgent;
18  }
19
20  /* 开启发现模式，这里无需设置过滤 uuid，否则搜索不到手机
21   * uuid 会过滤符合条件的 uuid 服务都会返回相应的蓝牙设备
22   */
23  void RemoteSelector::startDiscovery(const QBluetoothUuid &uuid)
24  {
25      Q_UNUSED(uuid);
26      qDebug()<<"startDiscovery";
27      if (m_discoveryAgent->isActive()) {
28          qDebug()<<"stop the searching first";
29          m_discoveryAgent->stop();
30      }
31
32      //m_discoveryAgent->setUuidFilter(uuid);
33
m_discoveryAgent->start(QBluetoothServiceDiscoveryAgent::FullDiscovery)
;
34  }
35
36  /* 停止发现 */
37  void RemoteSelector::stopDiscovery()
38  {
39      qDebug()<<"stopDiscovery";
40      if (m_discoveryAgent){
41          m_discoveryAgent->stop();
42      }
43  }
44
45  QBluetoothServiceInfo RemoteSelector::service() const
46  {
```

```
47      return m_serviceInfo;
48  }
49
50  /* 扫描蓝牙服务信息 */
51  void RemoteSelector::serviceDiscovered(const QBluetoothServiceInfo
&serviceInfo)
52  {
53  #if 0
54      qDebug() << "Discovered service on"
55              << serviceInfo.device().name() <<
serviceInfo.device().address().toString();
56      qDebug() << "\tService name:" << serviceInfo.serviceName();
57      qDebug() << "\tDescription:"
58              <<
serviceInfo.attribute(QBluetoothServiceInfo::ServiceDescription).toStri
ng();
59      qDebug() << "\tProvider:"
60              <<
serviceInfo.attribute(QBluetoothServiceInfo::ServiceProvider).toString(
);
61      qDebug() << "\tL2CAP protocol service multiplexer:"
62              << serviceInfo.protocolServiceMultiplexer();
63      qDebug() << "\tRFCOMM server channel:" <<
serviceInfo.serverChannel();
64  #endif
65
66      QMapIterator<QString, QBluetoothServiceInfo>
i(m_discoveredServices);
67      while (i.hasNext()){
68          i.next();
69          if (serviceInfo.device().address() ==
i.value().device().address()){
70              return;
71          }
72      }
73
74      QString remoteName;
75      if (serviceInfo.device().name().isEmpty())
76          remoteName = serviceInfo.device().address().toString();
77      else
78          remoteName = serviceInfo.device().name();
79
80      qDebug()<<"adding to the list....";
81      qDebug()<<"remoteName: "<< remoteName;
```

```
82      QListWidgetItem *item =
83              new QListWidgetItem(QString::fromLatin1("%1%2")
84                                  .arg(remoteName,
serviceInfo.serviceName()));
85      m_discoveredServices.insert(remoteName, serviceInfo);
86      emit newServiceFound(item);
87  }
88
89  /* 发现完成 */
90  void RemoteSelector::discoveryFinished()
91  {
92      qDebug()<<"discoveryFinished";
93      emit finished();
94  }
```

remoteselector .cpp 是 remoteselector.h 的实现代码。主要看以下几点。

第 4~13 行，初始化本地蓝牙，实例化对象 discoveryAgent（代理对象），蓝牙主要通过本地代理对象去扫描其他蓝牙。

第 32 行，这里官方 Qt 代码设计是过滤 uuid。只有符合对应的 uuid 的蓝牙，才会返回结果。因为我们要扫描我们的手机，所以这里我们要把它注释掉。手机的 uuid 没有设置成设定的 uuid，如果设置了 uuid 过滤，手机就扫描不出了。（uuid 指的是唯一标识，手机蓝牙有很多 uuid，不同的 uuid 有不同的作用，指示着不同的服务）。

其他代码都是一些逻辑性的代码，比较简单，请自由查看。

mainwindow.h 代码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   06_bluetooth_chat
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-03-19
*****************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <qbluetoothserviceinfo.h>
6   #include <qbluetoothsocket.h>
7   #include <qbluetoothhostinfo.h>
8   #include <QDebug>
9   #include <QTabWidget>
10  #include <QHBoxLayout>
```

```
11  #include <QVBoxLayout>
12  #include <QPushButton>
13  #include <QListWidget>
14  #include <QTextBrowser>
15  #include <QLineEdit>
16
17  class ChatServer;
18  class ChatClient;
19  class RemoteSelector;
20
21  class MainWindow : public QMainWindow
22  {
23      Q_OBJECT
24
25  public:
26      MainWindow(QWidget *parent = nullptr);
27      ~MainWindow();
28
29  public:
30      /* 暴露的接口，主动连接设备*/
31      Q_INVOKABLE void connectToDevice();
32
33  signals:
34      /* 发送消息信号 */
35      void sendMessage(const QString &message);
36
37      /* 连接断开信号 */
38      void disconnect();
39
40      /* 发现完成信号 */
41      void discoveryFinished();
42
43      /* 找到新服务信号 */
44      void newServicesFound(const QStringList &list);
45
46  public slots:
47      /* 停止搜索 */
48      void searchForDevices();
49
50      /* 开始搜索 */
51      void stopSearch();
52
53      /* 找到新服务 */
```

```
54      void newServiceFound(QListWidgetItem*);
55
56      /* 已连接 */
57      void connected(const QString &name);
58
59      /* 显示消息 */
60      void showMessage(const QString &sender, const QString &message);
61
62      /* 发送消息 */
63      void sendMessage();
64
65      /* 作为客户端断开连接 */
66      void clientDisconnected();
67
68      /* 主动断开连接 */
69      void toDisconnected();
70
71      /* 作为服务端时，客户端断开连接 */
72      void disconnected(const QString &name);
73
74  private:
75      /* 选择本地蓝牙 */
76      int adapterFromUserSelection() const;
77
78      /* 本地蓝牙的 Index */
79      int currentAdapterIndex;
80
81      /* 蓝牙本地适配器初始化 */
82      void localAdapterInit();
83
84      /* 布局初始化 */
85      void layoutInit();
86
87      /* 服务端*/
88      ChatServer *server;
89
90      /* 多个客户端 */
91      QList<ChatClient *> clients;
92
93      /* 远程选择器，使用本地蓝牙去搜索蓝牙，可过滤蓝牙等 */
94      RemoteSelector *remoteSelector;
95
```

```
96      /* 本地蓝牙 */
97      QList<QBluetoothHostInfo> localAdapters;
98
99      /* 本地蓝牙名称 */
100     QString localName;
101
102     /* tabWidget 视图，用于切换页面 */
103     QTabWidget *tabWidget;
104
105     /* 3 个按钮，扫描按钮，连接按钮，发送按钮 */
106     QPushButton *pushButton[5];
107
108     /* 2 个垂直布局，一个用于页面一，另一个用于页面二 */
109     QVBoxLayout *vBoxLayout[2];
110
111     /* 2 个水平布局，一个用于页面一，另一个用于页面二 */
112     QHBoxLayout *hBoxLayout[2];
113
114     /* 页面一和页面二容器 */
115     QWidget *pageWidget[2];
116
117     /* 用于布局，pageWidget 包含 subWidget */
118     QWidget *subWidget[2];
119
120     /* 蓝牙列表 */
121     QListWidget *listWidget;
122
123     /* 显示对话的内容 */
124     QTextBrowser *textBrowser;
125
126     /* 发送消息输入框 */
127     QLineEdit *lineEdit;
128
129 };
130 #endif // MAINWINDOW_H
```

mainwindow.h 是整个代码重要的文件，这里使用了客户端类，服务端类和远程服务端类。前面介绍的客户端类，服务端类和远程服务端类都是为 mainwindow.h 服务的。我们在编程的时候可以不用改动客户端类，服务端类和远程服务端类了，直接像 mainwindow.h 一样使用它们的接口就可以编程了。

其中笔者还在 mainwindow.h 使用了很多控件，这些控件都是界面组成的重要元素。并不复杂，如果看不懂界面布局，或者理解不了界面布局，请回到本教程的第七章学习基础，本教程不再一一说明这种简单的布局了。

mainwindow.cpp 代码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   06_bluetooth_chat
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-03-19
*****************************************************************/
#include "mainwindow.h"
#include "remoteselector.h"
#include "chatserver.h"
#include "chatclient.h"
#include <qbluetoothuuid.h>
#include <qbluetoothserver.h>
#include <qbluetoothservicediscoveryagent.h>
#include <qbluetoothdeviceinfo.h>
#include <qbluetoothlocaldevice.h>
#include <QGuiApplication>
#include <QScreen>
#include <QRect>
#include <QTimer>
#include <QDebug>
#include <QTabBar>
#include <QHeaderView>
#include <QTableView>


static const QLatin1String
serviceUuid("e8e10f95-1a70-4b27-9ccf-02010264e9c8");

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    /* 本地蓝牙初始化 */
    localAdapterInit();

    /* 界面布局初始化 */
    layoutInit();
```

```
31  }
32
33  MainWindow::~MainWindow()
34  {
35      qDeleteAll(clients);
36      delete server;
37  }
38
39  /* 初始化本地蓝牙，作为服务端 */
40  void MainWindow::localAdapterInit()
41  {
42      /* 查找本地蓝牙的个数 */
43      localAdapters = QBluetoothLocalDevice::allDevices();
44      qDebug() << "localAdapter: " << localAdapters.count();
45
46      QBluetoothLocalDevice localDevice;
47
localDevice.setHostMode(QBluetoothLocalDevice::HostDiscoverable);
48
49      QBluetoothAddress adapter = QBluetoothAddress();
50      remoteSelector = new RemoteSelector(adapter, this);
51      connect(remoteSelector,
52              SIGNAL(newServiceFound(QListWidgetItem*)),
53              this, SLOT(newServiceFound(QListWidgetItem*)));
54
55      /* 初始化服务端 */
56      server = new ChatServer(this);
57
58      connect(server, SIGNAL(clientConnected(QString)),
59              this, SLOT(connected(QString)));
60
61      connect(server, SIGNAL(clientDisconnected(QString)),
62              this, SLOT(disconnected(QString)));
63
64      connect(server, SIGNAL(messageReceived(QString, QString)),
65              this, SLOT(showMessage(QString, QString)));
66
67      connect(this, SIGNAL(sendMessage(QString)),
68              server, SLOT(sendMessage(QString)));
69
70      connect(this, SIGNAL(disconnect()),
71              server, SLOT(disconnect()));
72
73      server->startServer();
```

```
74
75      /* 获取本地蓝牙的名称 */
76      localName = QBluetoothLocalDevice().name();
77  }
78
79  void MainWindow::layoutInit()
80  {
81      /* 获取屏幕的分辨率，Qt 官方建议使用这
82       * 种方法获取屏幕分辨率，防上多屏设备导致对应不上
83       * 注意，这是获取整个桌面系统的分辨率
84       */
85      QList <QScreen *> list_screen =  QGuiApplication::screens();
86
87      /* 如果是 ARM 平台，直接设置大小为屏幕的大小 */
88  #if __arm__
89      /* 重设大小 */
90      this->resize(list_screen.at(0)->geometry().width(),
91                   list_screen.at(0)->geometry().height());
92  #else
93      /* 否则则设置主窗体大小为 800x480 */
94      this->resize(800, 480);
95  #endif
96
97      /* 主视图 */
98      tabWidget = new QTabWidget(this);
99
100     /* 设置主窗口居中视图为 tabWidget */
101     setCentralWidget(tabWidget);
102
103     /* 页面一对象实例化 */
104     vBoxLayout[0] = new QVBoxLayout();
105     hBoxLayout[0] = new QHBoxLayout();
106     pageWidget[0] = new QWidget();
107     subWidget[0] = new QWidget();
108     listWidget = new QListWidget();
109     /* 0 为扫描按钮，1 为连接按钮 */
110     pushButton[0] = new QPushButton();
111     pushButton[1] = new QPushButton();
112     pushButton[2] = new QPushButton();
113     pushButton[3] = new QPushButton();
114     pushButton[4] = new QPushButton();
115
```

```
116     /* 页面二对象实例化 */
117     hBoxLayout[1] = new QHBoxLayout();
118     vBoxLayout[1] = new QVBoxLayout();
119     subWidget[1] = new QWidget();
120     textBrowser = new QTextBrowser();
121     lineEdit = new QLineEdit();
122     pushButton[2] = new QPushButton();
123     pageWidget[1] = new QWidget();
124
125
126     tabWidget->addTab(pageWidget[1], "蓝牙聊天");
127     tabWidget->addTab(pageWidget[0], "蓝牙列表");
128
129     /* 页面一 */
130     vBoxLayout[0]->addWidget(pushButton[0]);
131     vBoxLayout[0]->addWidget(pushButton[1]);
132     vBoxLayout[0]->addWidget(pushButton[2]);
133     vBoxLayout[0]->addWidget(pushButton[3]);
134     subWidget[0]->setLayout(vBoxLayout[0]);
135     hBoxLayout[0]->addWidget(listWidget);
136     hBoxLayout[0]->addWidget(subWidget[0]);
137     pageWidget[0]->setLayout(hBoxLayout[0]);
138     pushButton[0]->setMinimumSize(120, 40);
139     pushButton[1]->setMinimumSize(120, 40);
140     pushButton[2]->setMinimumSize(120, 40);
141     pushButton[3]->setMinimumSize(120, 40);
142     pushButton[0]->setText("开始扫描");
143     pushButton[1]->setText("停止扫描");
144     pushButton[2]->setText("连接");
145     pushButton[3]->setText("断开");
146
147     /* 页面二 */
148     hBoxLayout[1]->addWidget(lineEdit);
149     hBoxLayout[1]->addWidget(pushButton[4]);
150     subWidget[1]->setLayout(hBoxLayout[1]);
151     vBoxLayout[1]->addWidget(textBrowser);
152     vBoxLayout[1]->addWidget(subWidget[1]);
153     pageWidget[1]->setLayout(vBoxLayout[1]);
154     pushButton[4]->setMinimumSize(120, 40);
155     pushButton[4]->setText("发送");
156     lineEdit->setMinimumHeight(40);
157     lineEdit->setText("正点原子论坛网址 www.openedv.com");
158
```

```
159     /* 设置表头的大小 */
160     QString str = tr("QTabBar::tab {height:40; width:%1};")
161             .arg(this->width()/2);
162     tabWidget->setStyleSheet(str);
163 
164     /* 开始搜寻蓝牙 */
165     connect(pushButton[0], SIGNAL(clicked()),
166             this, SLOT(searchForDevices()));
167 
168     /* 停止搜寻蓝牙 */
169     connect(pushButton[1], SIGNAL(clicked()),
170             this, SLOT(stopSearch()));
171 
172     /* 点击连接按钮，本地蓝牙作为客户端去连接外界的服务端 */
173     connect(pushButton[2], SIGNAL(clicked()),
174             this, SLOT(connectToDevice()));
175 
176     /* 点击断开连接按钮，断开连接 */
177     connect(pushButton[3], SIGNAL(clicked()),
178             this, SLOT(toDisconnected()));
179 
180     /* 发送消息 */
181     connect(pushButton[4], SIGNAL(clicked()),
182             this, SLOT(sendMessage()));
183 }
184 
185 /* 作为客户端去连接 */
186 void MainWindow::connectToDevice()
187 {
188     if (listWidget->currentRow() == -1)
189         return;
190 
191     QString name = listWidget->currentItem()->text();
192     qDebug() << "Connecting to " << name;
193 
194     // Trying to get the service
195     QBluetoothServiceInfo service;
196     QMapIterator<QString,QBluetoothServiceInfo>
197             i(remoteSelector->m_discoveredServices);
198     bool found = false;
199     while (i.hasNext()){
200         i.next();
201 
```

```
202         QString key = i.key();
203 
204         /* 判断连接的蓝牙名称是否在发现的设备里 */
205         if (key == name) {
206             qDebug() << "The device is found";
207             service = i.value();
208             qDebug() << "value: " << i.value().device().address();
209             found = true;
210             break;
211         }
212     }
213 
214     /* 如果找到，则连接设备 */
215     if (found) {
216         qDebug() << "Going to create client";
217         ChatClient *client = new ChatClient(this);
218         qDebug() << "Connecting...";
219 
220         connect(client, SIGNAL(messageReceived(QString,QString)),
221                 this, SLOT(showMessage(QString,QString)));
222         connect(client, SIGNAL(disconnected()),
223                 this, SLOT(clientDisconnected()));;
224         connect(client, SIGNAL(connected(QString)),
225                 this, SLOT(connected(QString)));
226         connect(this, SIGNAL(sendMessage(QString)),
227                 client, SLOT(sendMessage(QString)));
228         connect(this, SIGNAL(disconnect()),
229                 client, SLOT(disconnect()));
230 
231         qDebug() << "Start client";
232         client->startClient(service);
233 
234         clients.append(client);
235     }
236 }
237 
238 /* 本地蓝牙选择，默认使用第一个蓝牙 */
239 int MainWindow::adapterFromUserSelection() const
240 {
241     int result = 0;
242     QBluetoothAddress newAdapter = localAdapters.at(0).address();
243     return result;
244 }
245 
```

```
246 /* 开始搜索 */
247 void MainWindow::searchForDevices()
248 {
249     /* 先清空 */
250     listWidget->clear();
251     qDebug() << "search for devices!";
252     if (remoteSelector) {
253         delete remoteSelector;
254         remoteSelector = NULL;
255     }
256 
257     QBluetoothAddress adapter = QBluetoothAddress();
258     remoteSelector = new RemoteSelector(adapter, this);
259 
260     connect(remoteSelector,
261             SIGNAL(newServiceFound(QListWidgetItem*)),
262             this, SLOT(newServiceFound(QListWidgetItem*)));
263 
264     remoteSelector->m_discoveredServices.clear();
265     remoteSelector->startDiscovery(QBluetoothUuid(serviceUuid));
266     connect(remoteSelector, SIGNAL(finished()),
267             this, SIGNAL(discoveryFinished()));
268 }
269 
270 /* 停止搜索 */
271 void MainWindow::stopSearch()
272 {
273     qDebug() << "Going to stop discovery...";
274     if (remoteSelector) {
275         remoteSelector->stopDiscovery();
276     }
277 }
278 
279 /* 找到蓝牙服务 */
280 void MainWindow::newServiceFound(QListWidgetItem *item)
281 {
282     /* 设置项的大小 */
283     item->setSizeHint(QSize(listWidget->width(), 50));
284 
285     /* 添加项 */
286     listWidget->addItem(item);
287 
288     /* 设置当前项 */
```

```
289     listWidget->setCurrentRow(listWidget->count() - 1);
290
291     qDebug() << "newServiceFound";
292
293     // get all of the found devices
294     QStringList list;
295
296     QMapIterator<QString, QBluetoothServiceInfo>
297             i(remoteSelector->m_discoveredServices);
298     while (i.hasNext()){
299         i.next();
300         qDebug() << "key: " << i.key();
301         qDebug() << "value: " << i.value().device().address();
302         list << i.key();
303     }
304
305     qDebug() << "list count: "  << list.count();
306
307     emit newServicesFound(list);
308 }
309
310 /* 已经连接 */
311 void MainWindow::connected(const QString &name)
312 {
313     textBrowser->insertPlainText(tr("%1:已连接\n").arg(name));
314     tabWidget->setCurrentIndex(0);
315     textBrowser->moveCursor(QTextCursor::End);
316 }
317
318 /* 接收消息 */
319 void MainWindow::showMessage(const QString &sender,
320                              const QString &message)
321 {
322     textBrowser->insertPlainText(QString::fromLatin1("%1: %2\n")
323                                  .arg(sender, message));
324     tabWidget->setCurrentIndex(0);
325     textBrowser->moveCursor(QTextCursor::End);
326 }
327
328 /* 发送消息 */
329 void MainWindow::sendMessage()
330 {
331     showMessage(localName, lineEdit->text());
332     emit sendMessage(lineEdit->text());
```

```
333 }
334
335 /* 作为客户端断开连接 */
336 void MainWindow::clientDisconnected()
337 {
338     ChatClient *client = qobject_cast<ChatClient *>(sender());
339     if (client) {
340         clients.removeOne(client);
341         client->deleteLater();
342     }
343
344     tabWidget->setCurrentIndex(0);
345     textBrowser->moveCursor(QTextCursor::End);
346 }
347
348 /* 主动断开连接 */
349 void MainWindow::toDisconnected()
350 {
351     emit disconnect();
352     textBrowser->moveCursor(QTextCursor::End);
353     tabWidget->setCurrentIndex(0);
354 }
355
356 /* 作为服务端时，客户端断开连接 */
357 void MainWindow::disconnected(const QString &name)
358 {
359     textBrowser->insertPlainText(tr("%1:已断开\n").arg(name));
360     tabWidget->setCurrentIndex(0);
361     textBrowser->moveCursor(QTextCursor::End);
362 }
```

mainwindow.cpp 则是整个项目的核心文件，包括处理界面点击的事件，客户端连接，服务端连接，扫描蓝牙，断开蓝牙和连接蓝牙等。设计这样的一个逻辑界面并不难，只要我们前面第七章 Qt 控件打下了基础。上面的代码注释详细，请自由查看。

20.3 程序运行效果

本例程运行后，默认开启蓝牙的服务端模式，可以用手机安装蓝牙调试软件（安卓手机如蓝牙调试宝、蓝牙串口助手）。当我们点击蓝牙列表页面时，点击扫描后请等待扫描的结果，选中需要连接的蓝牙再点击连接。

下面程序效果是 Ubuntu 虚拟机上连接 USB 蓝牙模块，用手机连接后运行的蓝牙聊天第一页效果图。

下面程序效果是 Ubuntu 虚拟机上连接 USB 蓝牙模块运行的蓝牙聊天第二页效果图。

安卓手机可以用蓝牙调试宝等软件进行配对连接。IOS 手机请下载某些蓝牙调试软件测试即可。手机接收到的消息如下。

在笔者测试的过程中，发现在 Ubuntu 上运行蓝牙聊天程序不太好用，需要开启扫描后，才能连接得上，而且接收的消息反应比较慢，有可能是虚拟机的原因吧。不过在正点原子 I.MX6U 开发板上运行没有问题。先按照详细请看【正点原子】I.MX6U 用户快速体验 V1.x.pdf 的第 3.29 小节蓝牙测试开启蓝牙，启用蓝牙被扫描后，先进行配对，手机用蓝牙调试软件就可以连接上进行聊天了。

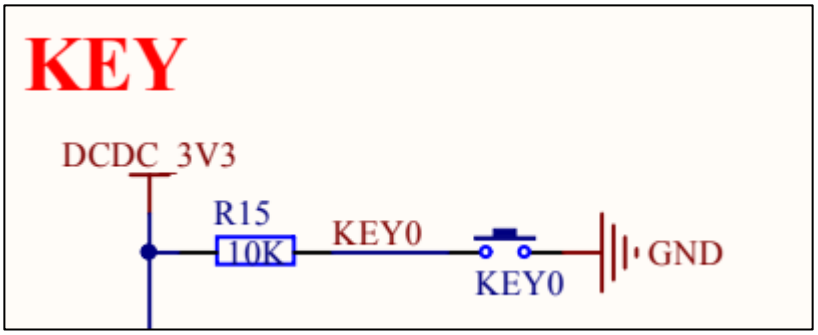
注意：本程序需要在确保蓝牙能正常使用的情况下才能运行，默认使用第一个蓝牙，如果 Ubuntu 上查看有两个蓝牙，请不要插着 USB 蓝牙启动电脑，先等 Ubuntu 启动后再插蓝牙模块。连接前应先配对，连接不上的原因可能或者蓝牙质量问题，或者系统里的软件没有开启蓝牙，或者使用的手机蓝牙调试软件不支持 SPP（串行端口）蓝牙调试等，请退出重试等。程序仅供学习与参考。

第二十一章 USER-KEY

本章是按键实验，介绍如何在 Qt 应用上使用正点原子嵌入式 I.MX6ULL Linux 开发板上的按键。

21.1 资源简介

在正点原子的 I.MX6U 开发板，ALPHA 和 MINI Linux 开发板板载资源上有一个按键。如下图原理图（下图为 ALPHA 开发板的 KEY0 按键原理图）。



21.2 应用实例

想要监测这个 KEY0，首先正点原子的出厂内核已经默认将这个按键注册成了 gpio-keys 类型设备，键值为 114 也就是对应 Qt 的 Key_VolumeDown 键值。也就是说我们可以直接当这个按键是我们普通键盘的音量减键使用（注意键值为 114 的按键比较特殊，并不能直接在我们普通的键盘上找到，有些笔记本电脑使用 FN + F6 等组合直接代替了音量减键）。

我们在本例中使用 Key_Down（键盘方向键↓）在 Windows/Ubuntu 上测试，在开发板上还是使用 KEY0 按键测试。

在开发板监测这个 KEY0 有很多方法。比如使用 C 语言开一个线程监测这个按键，或者按本例重写键盘事件来监测 KEY0 按键按下或者松开。

项目简介：监测 KEY0 按键的按下和松开。使用一个标签文本，通过按键按下来改变标签文本的文字属性。

例 07_key，监测 KEY0（难度简单）。项目路径为 Qt/3/07_key。

在源文件“mainwindow.h”的代码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   07_key
* @brief         mainwindow.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-04-19
*****************************************************************/
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QKeyEvent>
```

```
6   #include <QLabel>
7   #include <QDebug>
8   #include <QEvent>
9
10  class MainWindow : public QMainWindow
11  {
12      Q_OBJECT
13
14  public:
15      MainWindow(QWidget *parent = nullptr);
16      ~MainWindow();
17
18  private:
19      /* 标签文本 */
20      QLabel *label;
21
22      /* 重写按键事件 */
23      void keyPressEvent(QKeyEvent *event);
24      void keyReleaseEvent(QKeyEvent *event);
25  };
26
27  #endif // MAINWINDOW_H
```

第 23~24 行，声明需要重写的按键事件类型。分别是按下事件和松开事件。通过重写这两个事件可以监测到键盘或 KEY0 按下的状态。

在源文件“mainwindow.cpp”的代码如下。

```
    /*************************************************************
    Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
    * @projectName   07_key
    * @brief         mainwindow.cpp
    * @author        Deng Zhimao
    * @email         1252699831@qq.com
    * @net           www.openedv.com
    * @date          2021-04-19
    *************************************************************/
1   #include "mainwindow.h"
2   #include <QGuiApplication>
3   #include <QScreen>
4   #include <QRect>
5
6   MainWindow::MainWindow(QWidget *parent)
7       : QMainWindow(parent)
8   {
9       /* 获取屏幕的分辨率，Qt 官方建议使用这
```

```
10      * 种方法获取屏幕分辨率，防上多屏设备导致对应不上
11      * 注意，这是获取整个桌面系统的分辨率
12      */
13     QList <QScreen *> list_screen =  QGuiApplication::screens();
14
15     /* 如果是 ARM 平台，直接设置大小为屏幕的大小 */
16 #if __arm__
17     /* 重设大小 */
18     this->resize(list_screen.at(0)->geometry().width(),
19                  list_screen.at(0)->geometry().height());
20 #else
21     /* 否则则设置主窗体大小为 800x480 */
22     this->setGeometry(0, 0, 800, 480);
23 #endif
24
25     /* 标签实例化 */
26     label = new QLabel(this);
27
28     /* 设置默认文本 */
29 #if __arm__
30     label->setText("VolumeDown 松开状态");
31 #else
32     label->setText("Down 按键松开");
33 #endif
34
35     /* 设置对齐方式 */
36     label->setAlignment(Qt::AlignCenter);
37
38     /* 居中显示 */
39     setCentralWidget(label);
40 }
41
42 MainWindow::~MainWindow()
43 {
44 }
45
46 void MainWindow::keyPressEvent(QKeyEvent *event)
47 {
48 #if __arm__
49     /* 判断按下的按键，也就是板子 KEY0 按键 */
50     if(event->key() == Qt::Key_VolumeDown) {
51         /* 设置 label 的文本 */
```

```
52          label->setText("VolumeDown 按键按下");
53      }
54  #else
55      /* 判断按下的按键，也就是"↓"方向键 */
56      if(event->key() == Qt::Key_Down) {
57          /* 设置 label 的文本 */
58          label->setText("Down 按键按下");
59      }
60
61  #endif
62      /* 保存默认事件 */
63      QWidget::keyPressEvent(event);
64  }
65
66  void MainWindow::keyReleaseEvent(QKeyEvent *event)
67  {
68  #if __arm__
69      /* 判断松开的按键，也就是板子 KEY0 按键 */
70      if(event->key() == Qt::Key_VolumeDown) {
71          /* 设置 label 的文本 */
72          label->setText("VolumeDown 按键松开");
73      }
74  #else
75      /* 判断按下的按键，也就是"↓"方向键 */
76      if(event->key() == Qt::Key_Down) {
77          /* 设置 label 的文本 */
78          label->setText("Down 按键松开");
79      }
80  #endif
81      /* 保存默认事件 */
82      QWidget::keyReleaseEvent(event);
83  }
```

第 9 行~23 行，界面初始化设置，在嵌入式里，根据实际的屏的大小，设置全屏显示。按钮居中显示。

第 46~83 行，重写按下事件和松开事件，通过判断 event->key()等哪个按键，就可以知道是哪个按键按下或者松开了。并设置了标签文本的属性。

21.3 程序运行效果

Ubuntu/Windows 上当焦点聚集到此应用程序窗口时，按下方向键“↓”，标签文本的值会改变为“Down 按键按下”，当松开方向键“↓”，标签的文本值会改变为默认状态“Down 按键松开”。

同理在开发板上，按下 KEY0 键则会打印“VolumeDown 按键按下”，松开 KEY0 键会打印“VolumeDown 按键松开”。

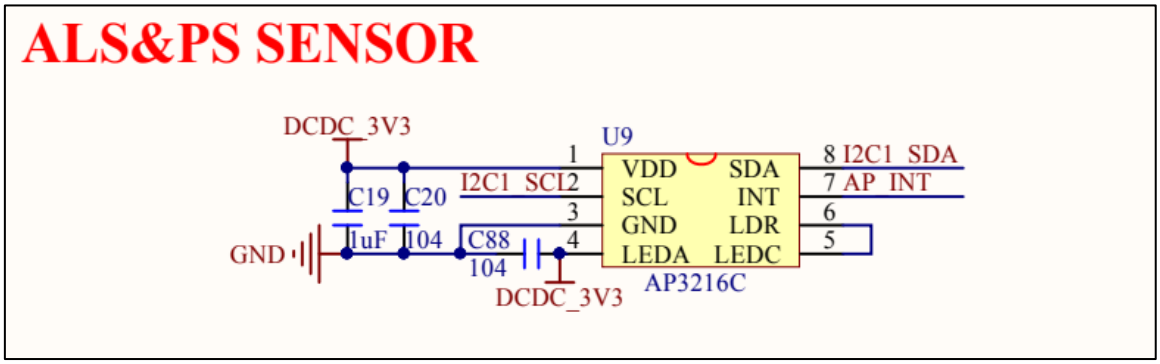
第二十二章 AP3216C

本章是 AP3216C 实验，介绍如何在 Qt 应用上获取正点原子嵌入式 I.MX6ULL Linux 开发板上的三合一环境传感器的数据。注意，既然是从开发板上获取数据，那么需要使用开发板，在正点原子 I.MX6ULL ALPHA 开发板上有这个 AP3216C 传感器，MINI 底板没有这个 AP3216C 传感器。不过本章实验程序在没有传感器的情况下也是可以运行的，可以在 Window/Ubuntu/ARM Linux 上运行看看界面效果，也可以直接看第 22.3 小节的程序运行后的效果图。获取数据的原理和第十五章 LED 章节原理一样都是从开发板获取数据，不同的是第十五章使用 QFile 这个类直接访问文件，本例介绍另外一种方法，就是使用 C 语言的 open()方法访问数据。本章没有使用很多新的知识，笔者于是花时间设计一个新界面，将界面拆分成一段段小知识。让读者明白要设计这么一个界面需要经过哪些步骤。

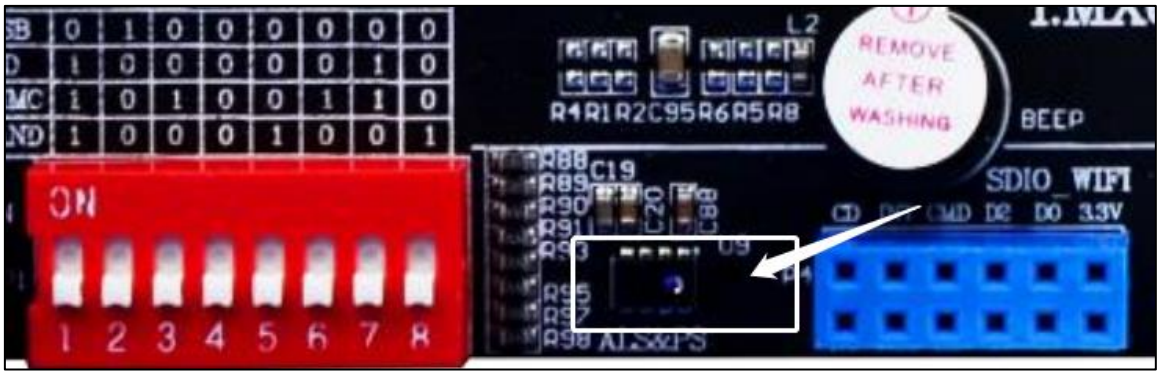
更多 AP3216C 的信息请看【正点原子】I.MX6U 用户快速体验 V1.x.pdf 的第 3.20 小节。

22.1 资源简介

在正点原子 I.MX6ULL ALPHA 开发板底板上有一个三合一环境传感器，也就是在拨码开关旁边的传感器，采用的是 I2C 接口。（注意：I.MX6ULL MINI 开发板没有这个传感器）。下图为 I.MX6ULL ALPHA 开发板的三合一环境传感器原理图。



开发板实物图位置。



22.2 应用实例

在正点原子 I.MX6U 出厂系统里，已经编写了 AP3216C 的驱动，并注册成了杂项设备，可以在/sys/class/misc 下找到 ap3216c 节点。我们直接用 Qt 通过访问节点文件的方式来获取 AP3216C 的传感器数据。读取数据流程解释：数据由驱动层传到 Linux 应用层，Qt 应用程序从应用层读取传感器数据。

项目简介：Qt 读取三合一环境传感器的数据。

例 08_ii2c_ap3216c_sensor，读取三合一环境传感器的数据（难度：一般）。项目路径为 Qt/3/08_ii2c_ap3216c_sensor。

项目文件 08_ii2c_ap3216c_sensor.pro 文件如下。

```
1  QT       += core gui
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
```

```
9   # depend on your compiler). Please consult the documentation of the
10  # deprecated API in order to know how to port your code away from it.
11  DEFINES += QT_DEPRECATED_WARNINGS
12
13  # You can also make your code fail to compile if it uses deprecated APIs.
14  # In order to do so, uncomment the following line.
15  # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16  #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0
17
18  SOURCES += \
19      ap3216c.cpp \
20      arcgraph.cpp \
21      glowtext.cpp \
22      main.cpp \
23      mainwindow.cpp
24
25  HEADERS += \
26      ap3216c.h \
27      arcgraph.h \
28      glowtext.h \
29      mainwindow.h
30
31  # Default rules for deployment.
32  qnx: target.path = /tmp/$${TARGET}/bin
33  else: unix:!android: target.path = /opt/$${TARGET}/bin
34  !isEmpty(target.path): INSTALLS += target
35
36  include(headview/headview.pri)
```

从上面的项目 pro 文件可以看出，本例使用的文件比较多。

第 36 行，使用到 pri 文件，pri 文件的语法和 pro 文件相同，通常它是由 pro 文件改写得到的，该类型文件类似于 C++中的头文件，可以在 pro 文件中使用 include 将其包含进来，相当于文件引入，当一个项目文件非常多时，或有些项目文件需要重复使用，为了方便管理就可以使用此方法。

项目里文件很多，我们一个一个分析，最终我们只需关注“mainwindow.h”和“mainwindow.cpp”文件，程序的主要流程都在这两个文件里。分析完了这两个文件再到其他文件。

在源文件“mainwindow.h”的代码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   08_spi_sensor
```

```
    * @brief        mainwindow.h
    * @author       Deng Zhimao
    * @email        1252699831@qq.com
    * @net          www.openedv.com
    * @date         2021-05-21
    ****************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QLabel>
6   #include <QVBoxLayout>
7   #include <QHBoxLayout>
8   #include "arcgraph.h"
9   #include "glowtext.h"
10  #include "ap3216c.h"
11  #include "headview/headview.h"
12  class ArcGraph;
13  class GlowText;
14  class Ap3216c;
15  class HeadView;
16
17  class MainWindow : public QMainWindow
18  {
19      Q_OBJECT
20
21  public:
22      MainWindow(QWidget *parent = nullptr);
23      ~MainWindow();
24
25  private:
26      ArcGraph *arcGraph[3];
27      GlowText *glowText[3];
28
29      QVBoxLayout *vBoxLayout;
30      QHBoxLayout *hBoxLayout[5];
31
32      GlowText *test;
33
34      /* 容器作用，用于布局 */
35      QWidget *widget[6];
36
37      /* 标签文本 */
38      QLabel *label[3];
```

```
39
40      /* i2c 传感器类 */
41      Ap3216c *ap3216c;
42
43      /* 视图表头 */
44      HeadView *headView;
45
46  private slots:
47      /* 获取 ap3216 传感器数据 */
48      void getAp3216cData();
49  };
50  #endif // MAINWINDOW_H
```

在“mainwindow.h”的头文件里，我们看到使用了 ArcGraph、GlowText、Ap3216c 和 HeadView 自定义的类。它们是蓝色科技弧形视图、发光文本、Ap3216c 类和视图表头。不同的类分开来写这样可以很方便地管理我们的项目。这些类在后面已经贴上代码和图加上一些解释方便给读者查阅。“mainwindow.h”头文件的解释就到这里了。

“mainwindow.cpp”文件主要承担着布局及数据显示的功能。

```
/*********************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   08_spi_sensor
* @brief         mainwindow.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-05-21
*********************************************************************/
1   #include "mainwindow.h"
2   #include <QDebug>
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       this->resize(800, 480);
7       this->setStyleSheet("background:#011753");
8
9       for (int i = 0; i < 6; i++)
10          widget[i] = new QWidget();
11
12      for (int i = 0; i < 3; i++)
13          arcGraph[i] = new ArcGraph();
14
15      for (int i = 0; i < 5; i++)
16          hBoxLayout[i] = new QHBoxLayout();
17
```

```
18      headView = new HeadView();
19
20      QFont font;
21      font.setPixelSize(18);
22
23      QPalette pal;
24      pal.setColor(QPalette::WindowText, Qt::white);
25
26      QStringList list;
27      list<<"环境光强度："<<"接近距离："<<"红外强度：";
28      for (int i = 0; i < 3; i++) {
29          label[i] = new QLabel();
30          glowText[i] = new GlowText();
31          glowText[i]->setMinimumWidth(30);
32          label[i]->setText(list[i]);
33          label[i]->setFont(font);
34          label[i]->setPalette(pal);
35          label[i]->adjustSize();
36      }
37
38      vBoxLayout = new QVBoxLayout();
39
40      /* 垂直布局，将主窗体为上下两部分，方便布局 */
41      vBoxLayout->addWidget(headView);
42      vBoxLayout->addWidget(widget[1]);
43      vBoxLayout->addWidget(widget[2]);
44      widget[0]->setLayout(vBoxLayout);
45
46      /* 主布局设置为 widget[0] */
47      setCentralWidget(widget[0]);
48
49      /* 设置 widget[1]的高度，不会随界面的大小而变化 */
50      widget[2]->setFixedHeight(150);
51
52      /* 三个蓝色科技感弧形图布局，采用水平布局 */
53      hBoxLayout[0]->addWidget(arcGraph[0]);
54      hBoxLayout[0]->addWidget(arcGraph[1]);
55      hBoxLayout[0]->addWidget(arcGraph[2]);
56      widget[1]->setLayout(hBoxLayout[0]);
57
58      /* 数据文字容器水平布局， */
59      hBoxLayout[1]->addWidget(widget[3]);
60      hBoxLayout[1]->addWidget(widget[4]);
```

```
61      hBoxLayout[1]->addWidget(widget[5]);
62      hBoxLayout[1]->setContentsMargins(0, 40, 0, 0);
63
64      widget[2]->setLayout(hBoxLayout[1]);
65
66      /* als 布局 */
67      hBoxLayout[2]->addWidget(label[0]);
68      hBoxLayout[2]->addWidget(glowText[0]);
69      hBoxLayout[2]->setAlignment(Qt::AlignTop | Qt::AlignHCenter);
70      widget[3]->setLayout(hBoxLayout[2]);
71
72      /* ps 布局 */
73      hBoxLayout[3]->addWidget(label[1]);
74      hBoxLayout[3]->addWidget(glowText[1]);
75      hBoxLayout[3]->setAlignment(Qt::AlignTop | Qt::AlignHCenter);
76      widget[4]->setLayout(hBoxLayout[3]);
77
78      /* ir 布局 */
79      hBoxLayout[4]->addWidget(label[2]);
80      hBoxLayout[4]->addWidget(glowText[2]);
81      hBoxLayout[4]->setAlignment(Qt::AlignTop | Qt::AlignHCenter);
82      widget[5]->setLayout(hBoxLayout[4]);
83
84      ap3216c = new Ap3216c(this);
85      /* 只能在开发板上开启获取数据，Ubuntu 上是没有 ap3216c 传感器的 */
86  #if __arm__
87      ap3216c->setCapture(true);
88  #endif
89
90      connect(ap3216c, SIGNAL(ap3216cDataChanged()),
91              this, SLOT(getAp3216cData()));
92  }
93
94  MainWindow::~MainWindow()
95  {
96  }
97
98  void MainWindow::getAp3216cData()
99  {
100     static QString als = ap3216c->alsData();
101     if (als != ap3216c->alsData()) {
102         als = ap3216c->alsData();
103         arcGraph[0]->setangleLength(als.toUInt() * 360 / 65535);
104     }
```

```
105
106     static QString ps = ap3216c->psData();
107     if (ps != ap3216c->psData()) {
108         ps = ap3216c->psData();
109         arcGraph[1]->setangleLength(ps.toUInt() * 360 / 1023);
110     }
111
112     static QString ir = ap3216c->irData();
113     if (ir != ap3216c->irData()) {
114         ir = ap3216c->irData();
115         arcGraph[2]->setangleLength(ir.toUInt() * 360 / 1023);
116     }
117
118     glowText[0]->setTextData(als);
119     glowText[1]->setTextData(ps);
120     glowText[2]->setTextData(ir);
121 }
```

第 98 行之前都是一些布局及变量声明使用的内容。

第 98~121 行，若收到 Ap3216c 类发送过来的信号，则显示数据。显示在发数据在 ArcGraph 和 GlowText 类上。其中 ArcGraph 是一个弧形视图，通过 setangleLength()方法，设置传入弧的角度大小，就可以画出一段弧，用图形的方式显示给用户看，比数字更直观。其中 65535 和 1023 数值的由来是环境光传感器具有 16 位的分辨率，接近传感器和红外传感器具有 10 位分辨率。也就是 2 的 16 次方减一，与 2 的 10 次方减一。

主要的流程基本介绍完，我们开始分步介绍 ArcGraph、GlowText、Ap3216c 和 HeadView 类。

HeadView 类主要功能是显示如下的一个表头。可以很方便地移植到其他项目里用。下图的背景颜色可忽略。实际这个类的背景颜色是透明的。

“headview.h”头文件内容如下。文件比较简单，不再解释。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   headview
* @brief        headview.h
* @author       Deng Zhimao
* @email        1252699831@qq.com
* @net          www.openedv.com
* @date         2021-05-22
*****************************************************************/
1   #ifndef HEADVIEW_H
2   #define HEADVIEW_H
```

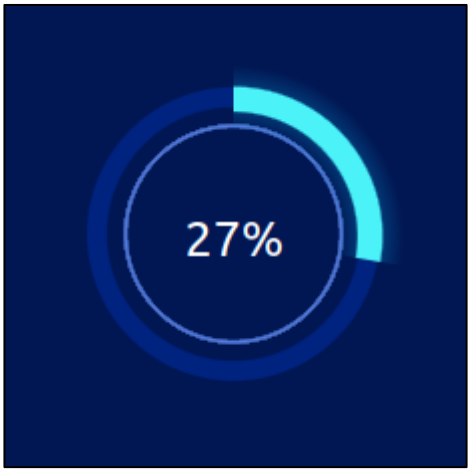
```
3
4   #include <QWidget>
5   #include <QLabel>
6   #include <QVBoxLayout>
7   #include <QHBoxLayout>
8
9
10  class HeadView : public QWidget
11  {
12      Q_OBJECT
13
14  public:
15      HeadView(QWidget *parent = nullptr);
16      ~HeadView();
17
18  private:
19      QWidget *widget;
20      QLabel *textLabel;
21      QWidget *iconWidget;
22      QWidget *lineWidget;
23
24      QHBoxLayout *hBoxLayout;
25      QVBoxLayout *vBoxLayout;
26  };
27  #endif // HEADVIEW_H
```

“headview.cpp”源文件内容如下。

```
    /**************************************************************************
    Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
    * @projectName   headview
    * @brief         headview.cpp
    * @author        Deng Zhimao
    * @email         1252699831@qq.com
    * @net           www.openedv.com
    * @date          2021-05-22
    **************************************************************************/
1   #include "headview.h"
2
3   HeadView::HeadView(QWidget *parent)
4       : QWidget(parent)
5   {
6       this->setAttribute(Qt::WA_TranslucentBackground, true);
7       widget = new QWidget();
8
9       iconWidget = new QWidget(this);
```

```
10      iconWidget->setFixedSize(48, 48);
11
iconWidget->setStyleSheet("background:url(:/images/dataviewicon.png)");
12
13      textLabel = new QLabel(this);
14      textLabel->setFixedSize(200, 48);
15      textLabel->setText("数据可视化情况");
16      textLabel->setStyleSheet("QLabel {font-size: 20px; color: white}");
17
18      lineWidget = new QWidget(this);
19      lineWidget->setFixedHeight(2);
20      lineWidget->setStyleSheet("QWidget {background: #eeeeeeee}");
21
22      vBoxLayout = new QVBoxLayout();
23      vBoxLayout->addWidget(widget);
24      vBoxLayout->addWidget(lineWidget);
25      vBoxLayout->setContentsMargins(0, 0, 0, 0);
26
27      hBoxLayout = new QHBoxLayout();
28      hBoxLayout->addWidget(iconWidget);
29      hBoxLayout->addWidget(textLabel);
30      hBoxLayout->setContentsMargins(0, 0, 0, 0);
31      hBoxLayout->setAlignment(Qt::AlignLeft);
32      widget->setLayout(hBoxLayout);
33
34      this->setLayout(vBoxLayout);
35      this->adjustSize();
36      this->setMaximumHeight(48);
37  }
38
39  HeadView::~HeadView()
40  {
41  }
```

ArcGraph 类是一个蓝色科技感弧形视图，这里运用了 QPainter 为画图，在第九章画图章节我们已经了解过 QPainter 的内容了。不详细解释。ArcGraph 类实现的效果如下。（注背景实际上是透明的）。



“arcgraph.h”头文件内容如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   08_spi_sensor
* @brief         arcgraph.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-05-21
*****************************************************************/
#ifndef ARCGRAPH_H
#define ARCGRAPH_H

#include <QWidget>
#include <QPainter>
#include <QPaintEvent>

/* 蓝色科技感弧形视图 */
class ArcGraph : public QWidget
{
    Q_OBJECT

public:
    ArcGraph(QWidget *parent = nullptr);
    ~ArcGraph();

    void setstartAngle(int);
    void setangleLength(int);

private:
    void paintEvent(QPaintEvent *event);
    int startAngle;
```

```
23      int angleLength;
24  };
25  #endif // ARCGRAPH_H
```

“arcgraph.cpp”源文件内容如下。

```
    /****************************************************************
    Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
    * @projectName   08_ii2_ap3216c_sensor
    * @brief         arcgraph.cpp
    * @author        Deng Zhimao
    * @email         1252699831@qq.com
    * @net           www.openedv.com
    * @date          2021-05-22
    ****************************************************************/
1   #include "arcgraph.h"
2
3   ArcGraph::ArcGraph(QWidget *parent)
4       : QWidget(parent),
5         startAngle(90),
6         angleLength(100)
7   {
8       this->setMinimumSize(100, 100);
9       setAttribute(Qt::WA_TranslucentBackground, true);
10  }
11
12  ArcGraph::~ArcGraph()
13  {
14  }
15
16  void ArcGraph::setstartAngle(int angle)
17  {
18      startAngle = angle;
19      this->repaint();
20  }
21
22  void ArcGraph::setangleLength(int length)
23  {
24      angleLength = length;
25      this->repaint();
26  }
27
28  void ArcGraph::paintEvent(QPaintEvent *event)
29  {
30      QPainter painter(this);
31
```

```
32      /* 保存状态 */
33      painter.save();
34
35      /* 设置抗锯齿 */
36      painter.setRenderHints(QPainter::Antialiasing, true);
37
38      /* 最外层的圆 */
39      QRect drawRect = event->rect();
40      QRadialGradient gradient1(drawRect.center(),
41                                drawRect.width() / 2,
42                                drawRect.center());
43      gradient1.setColorAt(0, Qt::transparent);
44      gradient1.setColorAt(0.5, Qt::transparent);
45      gradient1.setColorAt(0.51, QColor("#00237f"));
46      gradient1.setColorAt(0.58, QColor("#00237f"));
47      gradient1.setColorAt(0.59, Qt::transparent);
48      gradient1.setColorAt(1, Qt::transparent);
49      painter.setBrush(gradient1);
50      painter.setPen(Qt::NoPen);
51      painter.drawEllipse(drawRect);
52
53      /* 里层的圆 */
54      QRadialGradient gradient2(drawRect.center(),
55                                drawRect.width() / 2,
56                                drawRect.center());
57      gradient2.setColorAt(0, Qt::transparent);
58      gradient2.setColorAt(0.420, Qt::transparent);
59      gradient2.setColorAt(0.421, QColor("#885881e3"));
60      gradient2.setColorAt(0.430, QColor("#5881e3"));
61      gradient2.setColorAt(0.440, QColor("#885881e3"));
62      gradient2.setColorAt(0.441, Qt::transparent);
63      gradient2.setColorAt(1, Qt::transparent);
64      painter.setBrush(gradient2);
65      painter.setPen(Qt::NoPen);
66      painter.drawEllipse(drawRect);
67
68      /* 数字 */
69      QFont font;
70      font.setPixelSize(drawRect.width() / 10);
71      painter.setPen(Qt::white);
72      painter.setFont(font);
73      painter.drawText(drawRect, Qt::AlignCenter,
74                       QString::number(angleLength * 100 / 360) + "%");
```

```
75
76      /* 发光背景圆 */
77      painter.translate(drawRect.width() >> 1, drawRect.height() >> 1);
78      int radius = drawRect.width() / 2;
79      /* radius<< 1（左移 1 位）相当于 radius*2 */
80      QRectF rect(-radius, -radius, radius << 1, radius << 1);
81
82      QRadialGradient gradient3(0, 0, radius);
83      gradient3.setColorAt(0, Qt::transparent);
84      gradient3.setColorAt(0.42, Qt::transparent);
85      gradient3.setColorAt(0.51, QColor("#500194d3"));
86      gradient3.setColorAt(0.55, QColor("#22c1f3f9"));
87      gradient3.setColorAt(0.58, QColor("#500194d3"));
88      gradient3.setColorAt(0.68, Qt::transparent);
89      gradient3.setColorAt(1.0, Qt::transparent);
90      painter.setBrush(gradient3);
91      QPainterPath path1;
92      path1.arcTo(rect, startAngle, -angleLength);
93      painter.setPen(Qt::NoPen);
94      painter.drawPath(path1);
95
96      /* 发光圆/弧 */
97      QRadialGradient gradient4(0, 0, radius);
98      gradient4.setColorAt(0, Qt::transparent);
99      gradient4.setColorAt(0.49, Qt::transparent);
100     gradient4.setColorAt(0.50, QColor("#4bf3f9"));
101     gradient4.setColorAt(0.59, QColor("#4bf3f9"));
102     gradient4.setColorAt(0.60, Qt::transparent);
103     gradient4.setColorAt(1.0, Qt::transparent);
104     painter.setBrush(gradient4);
105     QPainterPath path2;
106     path2.arcTo(rect, startAngle, -angleLength);
107     painter.setPen(Qt::NoPen);
108     painter.drawPath(path2);
109
110     /* 恢复状态 */
111     painter.restore();
112
113     /* 设置事件对象的 accept 标志 */
114     event->accept();
115 }
```

GlowText 类是发光文字效果类，模拟出文字发光效果。可以看到下图的数字 100 微微发着青色的光，（文档可能效果不是很明显）在 Qt C++里想要实现这种效果需要自己实现。通过重写类或者自己设计程序实现。GlowText 是笔者设计发光效果的类。

“glowtext.h”头文件内容如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   GlowText
* @brief        glowtext.h
* @author       Deng Zhimao
* @email        1252699831@qq.com
* @net          www.openedv.com
* @date         2021-05-21
*****************************************************************/
#ifndef GLOWTEXT_H
#define GLOWTEXT_H

#include <QWidget>
#include <QLabel>

class GlowText : public QWidget
{
    Q_OBJECT

public:
    GlowText(QWidget *parent = nullptr);
    ~GlowText();

    void setTextColor(QColor);
    void setFontSize(int);
    void setTextData(QString);

private:
    /* 文本背景 */
    QLabel *textLabelbg;

    /* 文本标签 */
    QLabel *textLabel;
```

```
25
26      /* 字体颜色 */
27      QColor textColor;
28
29      /* 文本字体大小 */
30      int fontSize;
31
32      /* 文本内容 */
33      QString textData;
34  };
35  #endif // GLOWTEXT_H
```

“glowtext.cpp”源文件内容如下。

```
/****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   GlowText
* @brief         glowtext.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-05-21
****************************************************************/
1   #include "glowtext.h"
2   #include <QDebug>
3   #include <QGraphicsBlurEffect>
4
5   GlowText::GlowText(QWidget *parent)
6       : QWidget(parent),
7         textColor("#4bf3f9"),
8         fontSize(18),
9         textData("100")
10  {
11      QFont font;
12      font.setPixelSize(fontSize);
13      QPalette pal;
14      pal.setColor(QPalette::WindowText, textColor);
15      textLabelbg = new QLabel(this);
16      textLabelbg->setAttribute(Qt::WA_TranslucentBackground, true);
17      textLabelbg->setPalette(pal);
18      textLabelbg->setFont(font);
19      textLabelbg->setText(textData);
20      textLabelbg->setAlignment(Qt::AlignCenter);
21
22      /* 设置模糊特效 */
```

```
23      QGraphicsBlurEffect *ef = new QGraphicsBlurEffect();
24      ef->setBlurRadius(25);
25      ef->setBlurHints(QGraphicsBlurEffect::QualityHint);
26      textLabelbg->setGraphicsEffect(ef);
27
28      textLabel = new QLabel(this);
29      textLabel->setAttribute(Qt::WA_TranslucentBackground, true);
30      textLabel->setPalette(pal);
31      textLabel->setFont(font);
32      textLabel->setText(textData);
33      textLabel->setAlignment(Qt::AlignCenter);
34      textLabelbg->adjustSize();
35      textLabel->adjustSize();
36
37      this->resize(textLabel->size().width() + 10,
38                   textLabel->size().height() + 10);
39      /* 背景透明化 */
40      this->setAttribute(Qt::WA_TranslucentBackground, true);
41  }
42
43  GlowText::~GlowText()
44  {
45  }
46
47  void GlowText::setTextColor(QColor color)
48  {
49      QPalette pal;
50      pal.setColor(QPalette::WindowText, color);
51      textLabelbg->setPalette(pal);
52      textLabel->setPalette(pal);
53  }
54
55  void GlowText::setFontSize(int size)
56  {
57      QFont font;
58      font.setPixelSize(size);
59
60      textLabelbg->setFont(font);
61      textLabel->setFont(font);
62
63      textLabel->adjustSize();
64      textLabelbg->adjustSize();
65      this->resize(textLabel->size().width() + 10,
66                   textLabel->size().height() + 10);
```

```
67  }
68
69  void GlowText::setTextData(QString text)
70  {
71      textLabelbg->setText(text);
72      textLabel->setText(text);
73
74      textLabel->adjustSize();
75      textLabelbg->adjustSize();
76      this->resize(textLabel->size().width() + 10,
77                   textLabel->size().height() + 10);
78  }
```

Ap3216c 类的作用就是从驱动层提供给 Linux 应用层的接口获取数据。

“ap3216c.h”头文件内容如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   sensor
* @brief         ap3216c.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2020-07-10
*****************************************************************/
1   #ifndef AP3216C_H
2   #define AP3216C_H
3
4   #include <QObject>
5   #include <QTimer>
6
7   class Ap3216c : public QObject
8   {
9       Q_OBJECT
10
11  public:
12      explicit Ap3216c(QObject *parent = 0);
13      ~Ap3216c();
14
15      Q_INVOKABLE void setCapture(bool str);
16
17      QString alsData();
18      QString psData();
19      QString irData();
20
21  private:
```

```
22      QTimer *timer;
23      QString alsdata;
24      QString psdata;
25      QString irdata;
26
27      QString readAlsData();
28      QString readPsData();
29      QString readIrData();
30
31      Q_PROPERTY(QString alsData READ alsData NOTIFY ap3216cDataChanged)
32      Q_PROPERTY(QString psData READ psData NOTIFY ap3216cDataChanged)
33      Q_PROPERTY(QString irData READ irData NOTIFY ap3216cDataChanged)
34
35  public slots:
36      void timer_timeout();
37
38  signals:
39      void ap3216cDataChanged();
40
41  };
42
43  #endif // AP3216C_H
```

“ap3216c.cpp”源文件内容如下。

```
    /*****************************************************************
    Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
    * @projectName   sensor
    * @brief         ap3216c.cpp
    * @author        Deng Zhimao
    * @email         1252699831@qq.com
    * @net           www.openedv.com
    * @date          2020-07-10
    *****************************************************************/
1   #include "ap3216c.h"
2   #include <stdio.h>
3   #include <string.h>
4   #include <sys/types.h>
5   #include <sys/stat.h>
6   #include <fcntl.h>
7   #include <unistd.h>
8   #include <QDebug>
9
10  Ap3216c::Ap3216c(QObject *parent) : QObject (parent)
11  {
12      timer = new QTimer();
```

```
13      connect(timer, SIGNAL(timeout()), this, SLOT(timer_timeout()));
14  }
15
16  Ap3216c::~Ap3216c()
17  {
18
19  }
20
21  void Ap3216c::timer_timeout()
22  {
23      alsdata = readAlsData();
24      psdata = readPsData();
25      irdata = readIrData();
26      emit ap3216cDataChanged();
27  }
28
29  QString Ap3216c::readIrData()
30  {
31      char const *filename = "/sys/class/misc/ap3216c/ir";
32      int err = 0;
33      int fd;
34      char buf[10];
35
36      fd = open(filename, O_RDONLY);
37      if(fd < 0) {
38          close(fd);
39          return "open file error!";
40      }
41
42      err = read(fd, buf, sizeof(buf));
43      if (err < 0) {
44          close(fd);
45          return "read data error!";
46      }
47      close(fd);
48
49      QString irValue = buf;
50      QStringList list = irValue.split("\n");
51      return list[0];
52  }
53
54  QString Ap3216c::readPsData()
55  {
56      char const *filename = "/sys/class/misc/ap3216c/ps";
```

```
57      int err = 0;
58      int fd;
59      char buf[10];
60
61      fd = open(filename, O_RDONLY);
62      if(fd < 0) {
63          close(fd);
64          return "open file error!";
65      }
66
67      err = read(fd, buf, sizeof(buf));
68      if (err < 0) {
69          close(fd);
70          return "read data error!";
71      }
72      close(fd);
73
74      QString psValue = buf;
75      QStringList list = psValue.split("\n");
76      return list[0];
77  }
78
79  QString Ap3216c::readAlsData()
80  {
81      char const *filename = "/sys/class/misc/ap3216c/als";
82      int err = 0;
83      int fd;
84      char buf[10];
85
86      fd = open(filename, O_RDONLY);
87      if(fd < 0) {
88          close(fd);
89          return "open file error!";
90      }
91
92      err = read(fd, buf, sizeof(buf));
93      if (err < 0) {
94          close(fd);
95          return "read data error!";
96      }
97      close(fd);
98
99      QString alsValue = buf;
100     QStringList list = alsValue.split("\n");
```

```
101     return list[0];
102 }
103
104 QString Ap3216c::alsData()
105 {
106     return alsdata;
107 }
108
109 QString Ap3216c::irData()
110 {
111     return irdata;
112 }
113
114 QString Ap3216c::psData()
115 {
116     return psdata;
117 }
118
119 void Ap3216c::setCapture(bool str)
120 {
121     if(str)
122         timer->start(500);
123     else
124         timer->stop();
125 }
```

上面通过 C 语言的接口访问节点文件的方法来获取数据。需要包含 C 语言的头文件，如第 1~7 行。

总结，从上面来看，设计一个界面的代码往往比实现这个读取数据的功能复杂多。所以还是印证了那句笔者说的话，“美化界面比功能实现要耗时”!

22.3 程序运行效果

Ubuntu 运行效果图如下(下图为初始化数据)。要想获取传感器数据必须使用正点原子 I.MX6ULL APLPA 开发板，交叉编译到开发板上运行！

在正点原子 I.MX6ULL APLPA 开发板上运行的情况时，当我们用手接近三合一环境传感器时，界面上的数据会发生变化，数据默认设置为 500ms 采集一次。


08_li2c_ap3216c_sensor
数据可视化情况
27%
27%
27%
环境光强度： 100
接近距离： 100
红外强度： 100

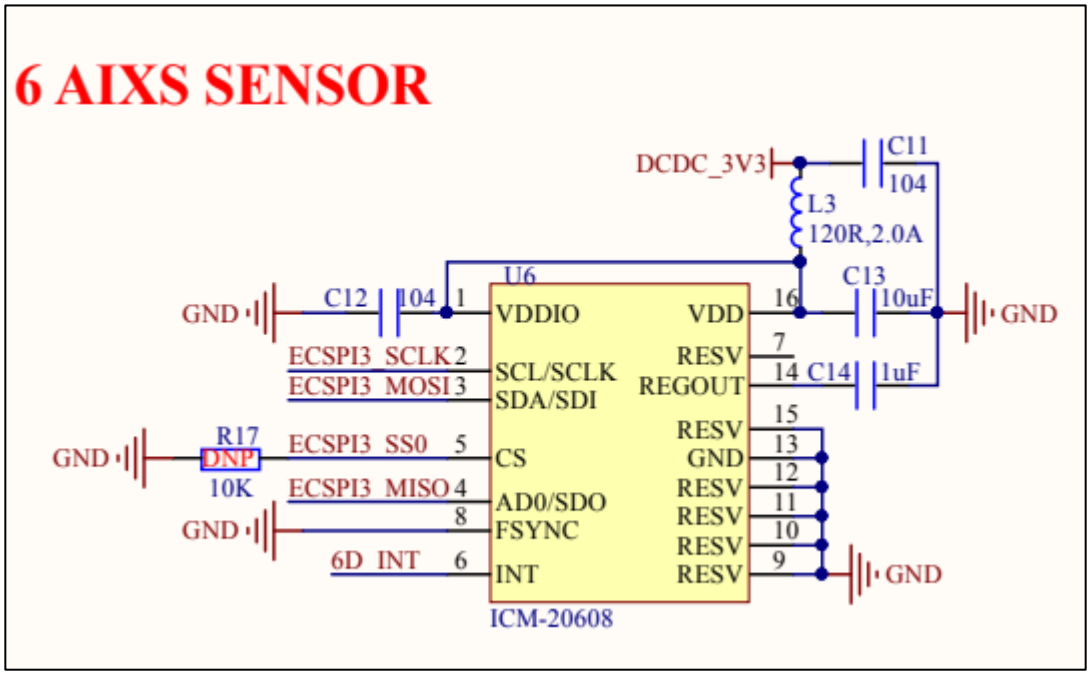
第二十三章 ICM20608

本章是 ICM20608 实验，与上一章类似，都是获取传感器的数据，ICM20608 是一款六轴 MEMS 传感器，包括三轴加速度和三轴陀螺仪。因为与上一章类似，故本章只提供获取 ICM20608 的接口，不再重复设计界面程序。

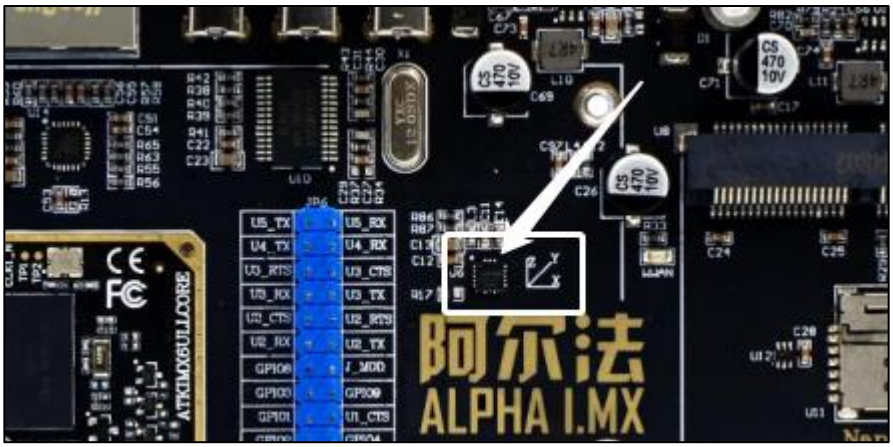
更多 ICM20608 的信息请看【正点原子】I.MX6U 用户快速体验 V1.x.pdf 的第 3.21 小节。

23.1 资源简介

在正点原子 I.MX6ULL ALPHA 开发板底板上有一个 6 轴 MEMS 传感器，也就是在底板上晶振旁边的传感器，采用的是 spi 接口。（注意：I.MX6ULL MINI 开发板没有这个传感器）。下图为 I.MX6ULL ALPHA 开发板的 6 轴 MEMS 传感器原理图。



开发板底板实物图。



23.2 应用接口

本实验提供 ICM20608 的 Qt 访问出厂系统 ICM20608 驱动设备文件/dev/icm20608 的接口。接口类 Icm20608 根据【正点原子】I.MX6U 嵌入式 Linux 驱动开发指南 V1.x.pdf 第六十二章 Linux SPI 驱动实验编写，想要理解原理请看【正点原子】I.MX6U 嵌入式 Linux 驱动开发指南，第六十二章，已经写得很详细。对于 Qt 访问底层驱动接口，我们需要对驱动有一点了解，否则学习起来是比较难的！建议读者全方面学习一下正点原子的 I.MX6U 配套教程。

源码路径为 Qt/3/09_spi_icm20608_sensor。

“icm20608.h”，头文件如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   sensor
* @brief         icm20608.h
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2020-07-10
*****************************************************************/
#ifndef ICM20608_H
#define ICM20608_H

#include <QObject>
#include <QTimer>

class Icm20608 : public QObject
{
    Q_OBJECT

public:
    explicit Icm20608(QObject *parent = 0);
    ~Icm20608();

    Q_INVOKABLE void setCapture(bool str);

private:
    QTimer *timer;
    QString gxdata;
    QString gydata;
    QString gzdata;
    QString axdata;
    QString aydata;
    QString azdata;
    QString tempdata;

    QString gxData();
    QString gyData();
    QString gzData();
    QString axData();
    QString ayData();
    QString azData();
    QString tempData();

```

```
35      void icm20608ReadData();
36
37      Q_PROPERTY(QString gxData READ gxData NOTIFY icm20608DataChanged)
38      Q_PROPERTY(QString gyData READ gyData NOTIFY icm20608DataChanged)
39      Q_PROPERTY(QString gzData READ gzData NOTIFY icm20608DataChanged)
40      Q_PROPERTY(QString axData READ axData NOTIFY icm20608DataChanged)
41      Q_PROPERTY(QString ayData READ ayData NOTIFY icm20608DataChanged)
42      Q_PROPERTY(QString azData READ azData NOTIFY icm20608DataChanged)
43      Q_PROPERTY(QString tempData READ tempData NOTIFY
icm20608DataChanged)
44
45  public slots:
46      void timer_timeout();
47
48
49  signals:
50      void icm20608DataChanged();
51  };
52
53  #endif // ICM20608_H
```

“icm20608.h”，源文件如下。

```
    /*****************************************************************
    Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
    * @projectName   sensor
    * @brief         icm20608.cpp
    * @author        Deng Zhimao
    * @email         1252699831@qq.com
    * @net           www.openedv.com
    * @date          2020-07-10
    *****************************************************************/
1   #include "icm20608.h"
2   #include <stdio.h>
3   #include <string.h>
4   #include <sys/types.h>
5   #include <sys/stat.h>
6   #include <fcntl.h>
7   #include <unistd.h>
8   #include <QDebug>
9
10  Icm20608::Icm20608(QObject *parent) : QObject (parent)
11  {
12      timer = new QTimer();
13  #if __arm__
14      system("insmod /home/root/driver/icm20608/icm20608.ko");
```

```
15  #endif
16      connect(timer,SIGNAL(timeout()),this,SLOT(timer_timeout()));
17  }
18
19  Icm20608::~Icm20608()
20  {
21
22  }
23
24  void Icm20608::icm20608ReadData()
25  {
26      int fd;
27      char const *filename = "/dev/icm20608";
28      signed int databuf[7];
29      unsigned char data[14];
30      signed int gyro_x_adc, gyro_y_adc, gyro_z_adc;
31      signed int accel_x_adc, accel_y_adc, accel_z_adc;
32      signed int temp_adc;
33
34      float gyro_x_act, gyro_y_act, gyro_z_act;
35      float accel_x_act, accel_y_act, accel_z_act;
36      float temp_act;
37
38      int ret = 0;
39
40      fd = open(filename, O_RDWR);
41      if(fd < 0) {
42          printf("can't open file %s\r\n", filename);
43          return;
44      }
45
46      ret = read(fd, databuf, sizeof(databuf));
47      /* 若读取成功 */
48      if (ret == 0) {
49          gyro_x_adc = databuf[0];
50          gyro_y_adc = databuf[1];
51          gyro_z_adc = databuf[2];
52          accel_x_adc = databuf[3];
53          accel_y_adc = databuf[4];
54          accel_z_adc = databuf[5];
55          temp_adc = databuf[6];
56
57          /* 实际值 */
58          gyro_x_act = (float)(gyro_x_adc)  / 16.4;
```

```
59          gyro_y_act = (float)(gyro_y_adc)  / 16.4;
60          gyro_z_act = (float)(gyro_z_adc)  / 16.4;
61          accel_x_act = (float)(accel_x_adc) / 2048;
62          accel_y_act = (float)(accel_y_adc) / 2048;
63          accel_z_act = (float)(accel_z_adc) / 2048;
64          temp_act = ((float)(temp_adc) - 25 ) / 326.8 + 25;
65
66          gxdata = QString::number(gyro_x_act, 'f', 2);
67          gydata = QString::number(gyro_y_act, 'f', 2);
68          gzdata = QString::number(gyro_z_act, 'f', 2);
69          axdata = QString::number(accel_x_act, 'f', 2);
70          aydata = QString::number(accel_y_act, 'f', 2);
71          azdata = QString::number(accel_z_act, 'f', 2);
72          tempdata = QString::number(temp_act, 'f', 2);
73      }
74      close(fd);
75      emit icm20608DataChanged();
76  }
77
78  QString Icm20608::gxData()
79  {
80      return  gxdata;
81  }
82
83  QString Icm20608::gyData()
84  {
85      return  gydata;
86  }
87
88  QString Icm20608::gzData()
89  {
90      return  gzdata;
91  }
92
93  QString Icm20608::axData()
94  {
95      return  axdata;
96  }
97
98  QString Icm20608::ayData()
99  {
100     return  aydata;
101 }
102
```

```
103 QString Icm20608::azData()
104 {
105     return  azdata;
106 }
107
108 QString Icm20608::tempData()
109 {
110     return  tempdata;
111 }
112
113
114 void Icm20608::timer_timeout()
115 {
116     icm20608ReadData();
117 }
118
119 void Icm20608::setCapture(bool str)
120 {
121     if(str)
122         timer->start(500);
123     else
124         timer->stop();
125 }
```

第四篇 项目实战篇

终于到项目实战篇了，大家前面学习辛苦了！前面的算是一个个小小的项目。而真正的实战项目就是本篇的。项目实战没有新的 Qt 语法学习，都是通过结合我们前面学习过的内容，开发出本篇的项目的。本篇重点不是讲 Qt 的语法或者界面设计，这些都是可以在前面章节学习到的，看程序很容易理解。重点是项目流程，流程将会分步讲解。

注：本篇项目实战篇需要读者的有较深的理解能力及较好 Qt 的基础。急于做项目的读者可以直接使用例程，若是初学者学习，请在本教程前面打下扎实的基础再细读程序。因为本篇是项目篇，一个项目代码会很长很长，不是讲解语法或者用法的，就不全贴代码了。

第二十四章 智能家居物联网项目

本章介绍使用 Qt 开发智能家居中的一个物联应用。简单直白的说就是通过云服务器远程控制设备（与设备通信等）。本章可以直接做毕设，是毕设物联网项目的一大福音！本章将实现远程点亮开发板 LED 作为一个项目实例。

在生活中，我们可能使用过 WIFI 智能插座这款产品。智能家居中常用来控制电器开关。比如远程开热水器，远程打开空调，窗帘等等。这些 WIFI 智能插座的原理就是将 WIFI 插座注册到云服务器上，然后通过手机的 APP 来访问云服务器，然后控制 WIFI 插座。嗯，原理我们懂了。本章就是模仿？不，或者说是直接开发这样的一个项目。包括 WIFI 连网，注册到云服务器上，编写 Qt UI 通过网络来与云服务器通信，然后再下发指令到这个连网的设备，与之通信。恩本章的流程就是这些，带着这个项目流程，然后一步步看笔者是如何通过 Qt 实现的吧！

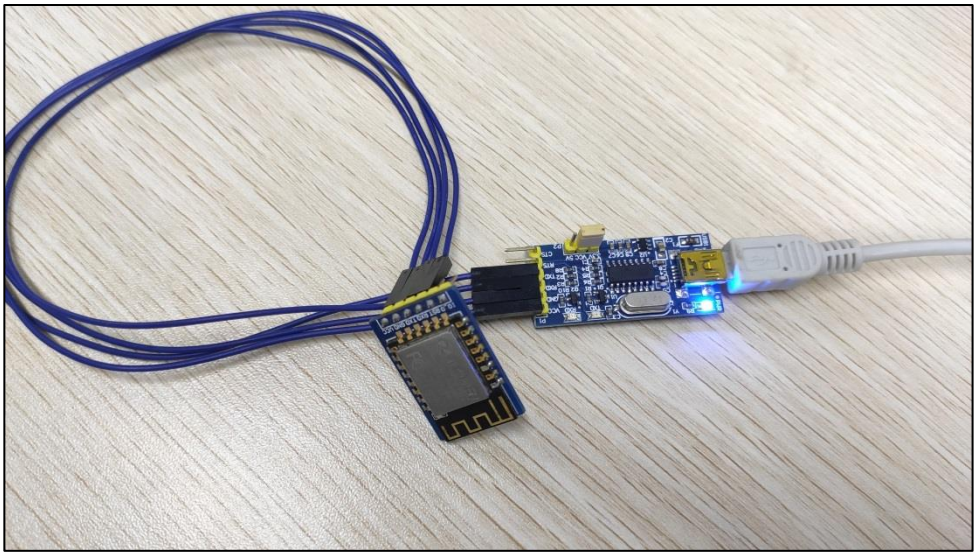
本章需要读者对正点原子的 wifi 模块 ATK-ESP8266 串口转 WIFI 有一定的了解。正点原子提供了 STM32 与 ESP8266 模块通信的例程，如果学习过 STM32 与 ESP8266 模块通信的例程的内容，理解起来则会更容易。建议参考文档：ATK-ESP8266 WIFI 用户手册_V1.x.pdf 及原子云平台 API 文档 V1.2.pdf。

24.1 项目硬件

- 必备硬件

本章需要正点原子 ATK-ESP8266 串口转 WIFI 模块（免费接入原子云）。另外还需要加上一个 USB-TTL 模块，外加一根 T 口 USB 连接线，可接入 PC（电脑）调试。

T 口连接线连接 USB-TTL 模块再连接 ATK-ESP8266 模块到 PC(电脑),用于在 PC(电脑)上直接使用串口调试/测试此模块。



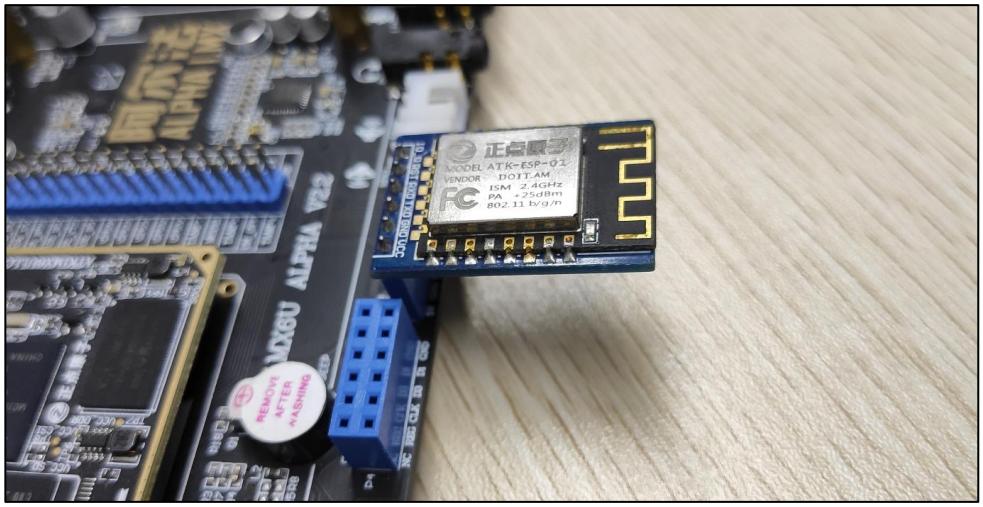
这里可能会有部分读者会问是否可以用其他 WIFI 模块，比如正点原子 Linux USB WIFI 模块，或者直接使用开发板联网接入到云设备呢？答案是不可以的！只有一些特定的设备，需要刷能接入云的固件才能接入服务器。

又有读者问是否可以直接购买一些 WIFI 插座来使用呢？答案是不可以的！因为这些 WIFI 插座也是一样，也是刷了固件，而且这些设备是连接到阿里云的。需要与特定的手机 APP 结合使用才能注册到阿里云服务器。也就是不能拿来二次开发了！

恰好我们正点原子有物联网模块 ESP8266 与 4G DTU 模块。本章主要讲解如何通过正点原子的串口转 WIFI ESP8266 模块来开发一个物联网的项目应用！

- 可选配件

本项目可以在正点原子 I.MX6U ALPHA | mini 开发板直接使用，下图为正点原子 ALPHA 开发板在底板上预留的 ATK MODULE 接口（串口接口）上的接法图。



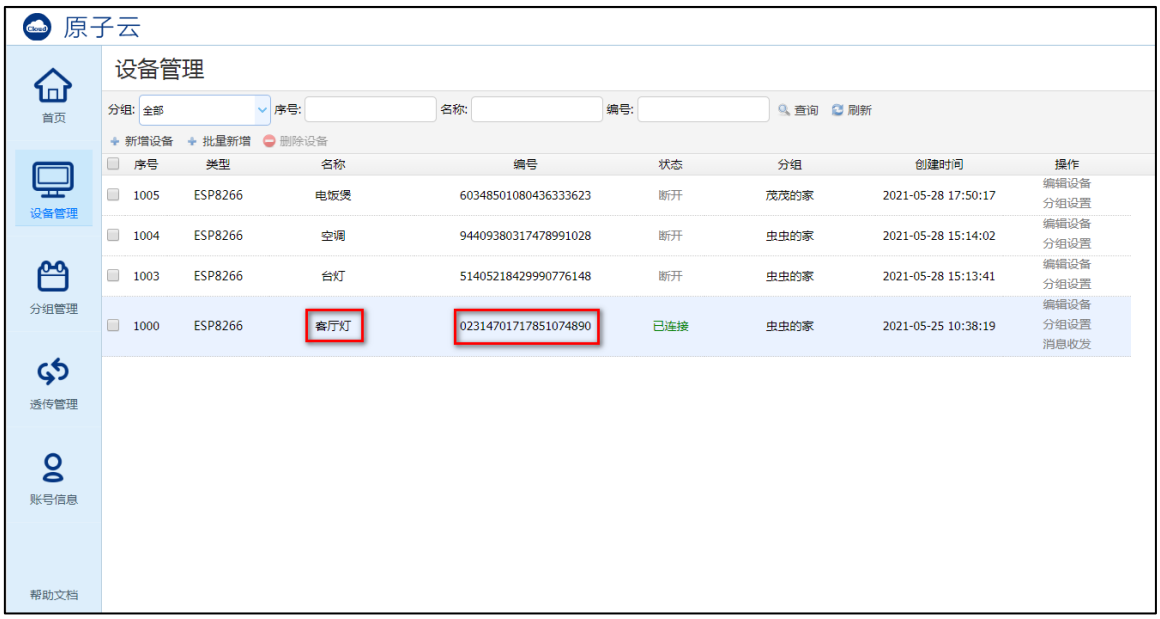
特别提醒：如果反复实验不正确时，因为 ATK-MODULE 这个接口，KEY 和 LED 脚如果有其他程序在使用，那么很可能会影响 ESP8266 模块的功能。刚好那两个脚接到了 ESP8266 的烧录固件 IO-0 脚与复位 RST 脚上，所以我们可以用杜邦线将模块重新连接到这个座子上，只接 VCC、GND、TX 和 RX 脚即可！

24.2 测试 WIFI 模块

要实现物联网功能，需要使用正点原子的 ATK-ESP8266 WIFI 模块。首先我们先测试正点原子的 ATK-ESP8266 WIFI 模块是否正常使用，及能否正常连接原子云服务器。ATK-ESP8266 WIFI 用户手册_V1.3.pdf 手册第 2.2.3 小节硬件连接，将 ATK-ESP8266 WIFI 模块连接到 PC（电脑），再查阅 ATK-ESP8266 WIFI 用户手册_V1.3.pdf 手册的第 2.2.9.1 小节，注册原子云帐号后，添加设备，然后按 2.2.9.1 小节测试连接本地 WIFI（自己的路由器发出的 WIFI，**注意不要使用中文名或者有空格的 WIFI**，确保路由器的 WIFI 能上网！）。请自行完成并成功连接到原子云。

24.3 WIFI 模块连接原子云

请先测试个人的 ATK-ESP8266 模块是否正常使用，及正常连接云。原子云的设备需要先分好组，各个设备命名如下。注意需要和笔者命名的名字一样，也就是至少有一个分组及一个名字为“客厅灯”的设备，并记住编号及密码（密码由云生成，默认“12345678”）。



源码路径 4/01_smarthome/esp8266/esp8266.cpp，内容如下。默认使用的 WIFI 模块串口通信波特率为 115200。在 Ubuntu 上设试 WIFI 模块时，一般串口名称为“ttyUSB0”，默认是没有权限访问这个/dev/ttyUSB0 设备的。所以我们需要使用下面的指令修改权限。（注意：本例适用于 I.MX6U Linux 开发板与 Ubuntu，Windows 不作测试！）。

```
sudo chmod 777 /dev/ttyUSB0
```

修改完成后查看源码内容如下。先看源码，不要急着运行！

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   esp8266
* @brief         esp8266.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-05-27
*****************************************************************/
#include "esp8266.h"
#include <unistd.h>
#include <QDebug>

Esp82266::Esp82266(QWidget *parent)
{
    Q_UNUSED(parent)
    /* 串口对象，用于与 Esp8266 模块通信 */
    serialPort = new QSerialPort(this);

    /* 定时器对象，用于定时发送设备在线的心跳包 */
    timer = new QTimer();

```

```
14      /* led对象，用于串口接收到云发过来的数据，然后控制板子的LED */
15      led = new Led(this);
16
17      /* 设置串口名 */
18  #if __arm__
19      serialPort->setPortName("ttymxc2");
20  #else
21      serialPort->setPortName("ttyUSB0");
22  #endif
23
24      /* 设置波特率 */
25      serialPort->setBaudRate(115200);
26
27      /* 设置数据位数 */
28      serialPort->setDataBits(QSerialPort::Data8);
29
30      /* 设置奇偶校验 */
31      serialPort->setParity(QSerialPort::NoParity);
32
33      /* 设置停止位 */
34      serialPort->setStopBits(QSerialPort::OneStop);
35
36      /* 设置流控制 */
37      serialPort->setFlowControl(QSerialPort::NoFlowControl);
38
39      if (!serialPort->open(QIODevice::ReadWrite))
40          qDebug()<<"串口无法打开！可能正在被使用！"<<endl;
41      else {
42          qDebug()<<"串口打开成功！"<<endl;
43      }
44
45      /* 开始连接云 */
46      connectToClound();
47
48      connect(serialPort, SIGNAL(readyRead()),
49              this, SLOT(serialPortReadyRead()));
50
51      connect(timer, SIGNAL(timeout()),
52              this, SLOT(onTimerTimeOut()));
53  }
54
55  void Esp82266::serialPortReadyRead()
56  {
```

```
57      /* 接收缓冲区中读取数据 */
58      QByteArray buf = serialPort->readAll();
59
60      QString temp = QString(buf);
61      readData.append(temp);
62      qDebug()<<temp<<endl;
63
64      if  (readData.contains("ready")) {
65          /* 如果复位成功 */
66          sendCmdToEsp8266("AT+CWMODE=1");
67          readData.clear();
68      }
69
70      if (readData.contains("OK") && readData.contains("AT+CWMODE")) {
71          qDebug()<<"设置 STA 模式成功"<<endl;
72          sendCmdToEsp8266("AT+CWJAP=\"ALIENTEK-YF\",\"15902020353\"");
73          qDebug()<<"开始连接 WIFI"<<endl;
74          readData.clear();
75      }
76
77      if  (temp.contains("WIFI GOT IP")) {
78          qDebug()<<"连接 WIFI 成功"<<endl;
79          sleep(2);
80          /* 原子云的设备号及密码 */
81
sendCmdToEsp8266("AT+ATKCLDSTA=\"02314701717851074890\",\"12345678\"");
82          qDebug()<<"开始连接原子云请等待"<<endl;
83      }
84
85      if (temp.contains("CLOUD CONNECTED")) {
86          qDebug()<<"连接原子云成功"<<endl;
87          sleep(2);
88          /* 15s 就发送一次心跳包 */
89          timer->start(15000);
90      }
91
92      if (temp == "开")
93          led->setLedState(true);
94      else if (temp == "关")
95          led->setLedState(false);
96
97  }
98
```

```
99  Esp82266::~Esp82266()
100 {
101     serialPort->close();
102     delete timer;
103     timer = nullptr;
104 }
105
106 void Esp82266::sendCmdToEsp8266(QString cmd)
107 {
108     cmd = cmd + "\r\n";
109
110     QByteArray data = cmd.toUtf8();
111     serialPort->write(data);
112 }
113
114 void Esp82266::connectToClound()
115 {
116     /* 重启模块，注意若已经连接上原子云，
117      * 需要重新上电或者短接 RST 脚来复位模块 */
118     sendCmdToEsp8266("AT+RST");
119 }
120
121 void Esp82266::sleep(int second)
122 {
123     usleep(second * 1000000);
124 }
125
126 void Esp82266::sendTextMessage(QString message)
127 {
128     serialPort->write(message.toLatin1());
129 }
130
131 void Esp82266::onTimerTimeOut()
132 {
133     sendTextMessage("online");
134     qDebug()<<"发送设备在线心跳包"<<endl;
135 }
```

查看源码，我们可以知道以下重要内容。

第 72 行，“ALIENTEK-YF”是笔者处的路由器发出的 WIFI 热点名称，密码是“15902020353”。请修改为自己的 WIFI 名称及连接密码！没有路由器，用手机开热点也可以。

第 81 行，是原子云上的“客厅灯”设备的编号“02314701717851074890”，及密码“12345678”，请填写自己的设备编号，及密码，注意这个设备需要命名为“客厅灯”，后面程序需要使用到它！

按照上面的程序就容易地可以连接上原子云了。也无需熟读 ATK-ESP8266 WIFI 用户手册_V1.3.pdf 手册。

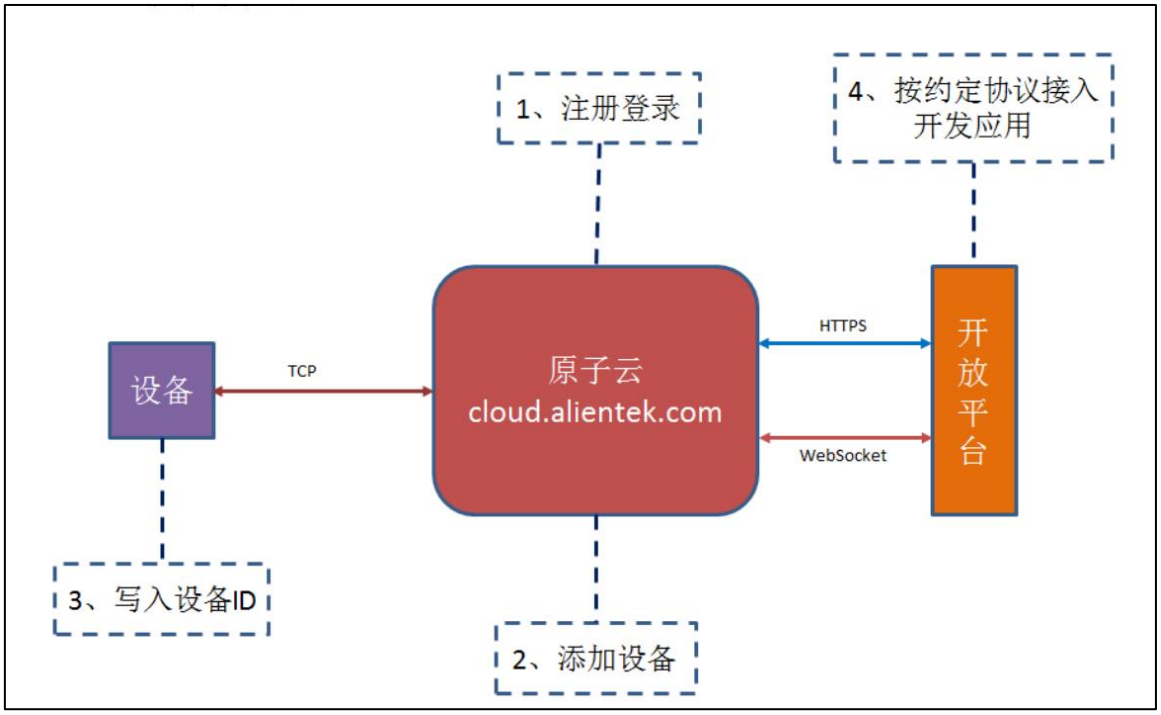
24.4 智能家居物联 UI 界面开发

项目路径为 4/01_smarthome/01_smarthome/01_smarthome.pro，先看项目界面。项目界面如下，采用暗黑主题设计，结合黄色作为亮色，让用户一目了然。界面笔者从一些智能家居界面中找到灵感的，编写设计完成的效果不错！请自行查阅源码，掌握了本教程前面第七章的内容，就可以理解这个界面是如何设计的。

24.5 原子云 API 接口

我们想要与原子云通信，那么必须先了解原子云平台的 API 接口。请参阅原子云平台 API 文档 V1.2.pdf 文档。原子云平台 API 写的非常详细了，请自行翻阅。需要我们从原子云平台了解原子云 API 的通信流程。

下图是原子云平台 API 的使用流程图。



我们写 Qt 应用就应该重点放在 HTTPS 与 WebSocket 方向上。查阅原子云平台 API 可以知道，下面是重点！一些帐号信息，与设备信息是通过 HTTPS 协议接口获取的，通信用 WebSocket 协议接口。那么我们就按原子云平台的协议流程编写应用程序。

源码路径为 4/01_smarthome/webapi/webapi.cpp。内容如下。

```
/*********************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   webapi
* @brief         webapi.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-05-27
*********************************************************************/
#include "webapi.h"
#include <QUuid>
#include <QRegularExpression>

Webapi::Webapi(QObject *parent)
{
    this->setParent(parent);
    /* 数组清空 */
    groupID.clear();
    deviceID.clear();
    deviceNumber.clear();

    timer = new QTimer();
```

```
14      connect(timer, SIGNAL(timeout()), this, SLOT(onTimerTimeOut()));
15
16      networkAccessManager = new QNetworkAccessManager(this);
17
18      orgURL = "https://cloud.alientek.com/api/orgs";
19      /* 请填写自己的 token 信息！！！ */
20      api_token = "bf591984c8fa417584d18f6328e0ef73";
21
22      /* 获取账号机构列表 */
23      getOrgURL();
24
25      QUuid uuid = QUuid::createUuid();
26      random_token = uuid.toString();
27
28      webSocket = new QWebSocket();
29      /* 需要加一些安全配置才能访问 https */
30      QSslConfiguration config;
31      config.setPeerVerifyMode(QSslSocket::VerifyNone);
32      config.setProtocol(QSsl::TlsV1SslV3);
33      webSocket->setSslConfiguration(config);
34
35      connect(webSocket, SIGNAL(connected()),
36              this, SLOT(webSocketConnected()));
37      connect(webSocket, SIGNAL(binaryMessageReceived(QByteArray)),
38              this, SLOT(onBinaryMessageReceived(QByteArray)));
39  }
40
41  Webapi::~Webapi()
42  {
43      delete timer;
44      delete webSocket;
45      webSocket = nullptr;
46  }
47
48  void Webapi::getOrgURL()
49  {
50      getDataFromWeb(QUrl(orgURL));
51  }
52
53  /* 获取设备分组列表 */
54  void Webapi::getGroupListUrl()
55  {
56      getDataFromWeb(QUrl(groupListUrl));
57  }
```

```
58
59  /* 获取设备的信息 */
60  void Webapi::getDevOfGroupUrl()
61  {
62      getDataFromWeb(QUrl(devOfGroupUrl));
63  }
64
65  /* 获取设备连接状态  */
66  void Webapi::getConStateUrl()
67  {
68      getDataFromWeb(QUrl(conStateUrl));
69  }
70
71  /* 从云服务器获取数据 */
72  void Webapi::getDataFromWeb(QUrl url)
73  {
74      /* 网络请求 */
75      QNetworkRequest networkRequest;
76
77      /* 需要加一些安全配置才能访问 https */
78      QSslConfiguration config;
79      config.setPeerVerifyMode(QSslSocket::VerifyNone);
80      config.setProtocol(QSsl::TlsV1SslV3);
81      networkRequest.setSslConfiguration(config);
82
83      /* 设置访问的地址 */
84      networkRequest.setUrl(url);
85
86      /* 网络响应 */
87      networkRequest.setHeader(QNetworkRequest::ContentTypeHeader,
88                               "application/json;charset=UTF-8");
89
90      /* 参数二为原子云帐号的 token 信息,填写自己的 */
91      networkRequest.setRawHeader("token", api_token.toLatin1());
92
93      QNetworkReply *newReply =
94              networkAccessManager->get(networkRequest);
95
96      connect(newReply, SIGNAL(finished()),
97              this, SLOT(replyFinished()));
98      connect(newReply, SIGNAL(readyRead()),
99              this, SLOT(readyReadData()));
100
```

```
101 }
102 void Webapi::replyFinished()
103 {
104     QNetworkReply *reply = (QNetworkReply *)sender();
105 
106     if (reply->url() == QUrl(orgURL)) {
107         /* 设备分组列表 ID */
108         getID(dataString, reply);
109     }
110 
111     if (reply->url() == QUrl(groupListUrl)) {
112         /* 列表 ID */
113         getID(dataString, reply);
114 
115         /* 获取到组 ID 再开启定时器 */
116         if (!timer->isActive())
117             timer->start(2000);
118     }
119 
120     /* 设备的信息 */
121     if (reply->url() == QUrl(devOfGroupUrl)) {
122         getID(dataString, reply);
123         getNumber(dataString);
124         getName(dataString);
125     }
126 
127     /* 设备的连接状态 */
128     if (reply->url() == QUrl(conStateUrl)) {
129         getConnectState(dataString);
130     }
131 
132     reply->deleteLater();
133     reply = nullptr;
134 }
135 void Webapi::readyReadData()
136 {
137     QNetworkReply *reply = (QNetworkReply *)sender();
138     QByteArray data = reply->readAll();
139     dataString =  QString(data);
140     qDebug()<<dataString<<endl;
141 }
142 
143 /* 获取 ID，包括分组 id，设备 id */
```

```
144 void Webapi::getID(QString data, QNetworkReply *reply)
145 {
146     /* 多个匹配，因为可能有多个合适的字段 */
147     QRegularExpression pattern("\"id\":(\\d+)");
148 
149     QRegularExpressionMatchIterator i = pattern.globalMatch(data);
150     while (i.hasNext()) {
151         QRegularExpressionMatch match = i.next();
152         if (match.hasMatch()) {
153             if (reply->url() == QUrl(orgURL)) {
154                 org_id =  match.captured(1);
155                 groupListUrl = "https://cloud.alientek.com/api/orgs/"
156                         + org_id + "/grouplist";
157                 getGroupListUrl();
158                 /* Socket 连接 */
159 
webSocket->open(QUrl(QString("wss://cloud.alientek.com/connection/%1/org/%2?token=%3")
160                                     .arg(api_token).arg(org_id).arg(random_token)));
161             }
162 
163             if (reply->url() == QUrl(groupListUrl)) {
164                 group_id = match.captured(1);
165                 /* 存储组 ID，再由定时器根据组的 ID 获取设备信息 */
166                 groupID.append(group_id);
167                 qDebug()<<"组 ID:"<<group_id<<endl;
168 
169             }
170 
171             if (reply->url() == QUrl(devOfGroupUrl)) {
172                 device_id = match.captured(1);
173                 /* 存储设备 ID,再由定时器根据设备的 ID 获取连接状态 */
174                 deviceID.append(device_id);
175                 qDebug()<<"设备 ID:"<<device_id<<endl;
176             }
177         }
178     }
179 }
180 
181 void Webapi::getNumber(QString data)
182 {
183     QRegularExpression pattern("\"number\":\"(\\d+)\"");
```

```
184
185     QRegularExpressionMatchIterator i = pattern.globalMatch(data);
186     while (i.hasNext()) {
187         QRegularExpressionMatch match = i.next();
188         if (match.hasMatch()) {
189             device_number = match.captured(1);
190             deviceNumber.append(device_number);
191             qDebug()<<"设备编号: "<<device_number<<endl;
192         }
193     }
194 }
195
196 void Webapi::getName(QString data)
197 {
198     /* 匹配中文字符，设备起名需要为中文 */
199     QRegularExpression pattern("\"name\":\"([\u4e00-\u9fa5]*)");
200
201     QRegularExpressionMatchIterator i = pattern.globalMatch(data);
202     while (i.hasNext()) {
203         QRegularExpressionMatch match = i.next();
204         if (match.hasMatch()) {
205             device_name = match.captured(1);
206             deviceName.append(device_name);
207             qDebug()<<"设备名称: "<<device_name<<endl;
208         }
209     }
210 }
211
212 /* 获取设备的连接状态 */
213 void Webapi::getConnectState(QString data)
214 {
215     QString pattern = "\"data\":\"(\\S*)\"";
216     QRegularExpression regularExpression(pattern);
217     QRegularExpressionMatch  match = regularExpression.match(data, 0);
218     if(match.hasMatch()) {
219         qDebug()<<"设备连接状态"<<match.captured(1);
220         deviceConnectState.append(match.captured(1));
221     }
222 }
223
224 void Webapi::webSocketConnected()
225 {
226     qDebug()<<"webSocket 连接原子云成功"<<endl;
```

```
227 }
228
229 void Webapi::onBinaryMessageReceived(QByteArray str)
230 {
231
232     QString temp(str);
233     if (temp.contains("online")) {
234         for (int i = 0; i < deviceNumber.count() ; i++) {
235             if (temp.contains(deviceNumber[i])) {
236                 /* 发送如客厅灯在线信号*/
237                 emit deviceStateChanged(deviceName[i] + "|在线");
238                 qDebug()<<deviceName[i] + "|在线"<<endl;
239                 break;
240             }
241         }
242     }
243 }
244
245 /* 延时函数 */
246 void Webapi::sleep(double second)
247 {
248     usleep(second * 1000000);
249 }
250
251 void Webapi::onTimerTimeOut()
252 {
253     static int i = 0;
254     if (i < groupID.count()) {
255         /* 获取分组下的设备列表 */
256         devOfGroupUrl = "https://cloud.alientek.com/api/orgs/"
257                 + org_id + "/groups/"
258                 + groupID[i] + "/devices";
259         dataString.clear();
260         getDevOfGroupUrl();
261     } else if (i >= groupID.count()
262                && i < groupID.count() + deviceID.count() ) {
263         timer->start(1000);
264         conStateUrl = "https://cloud.alientek.com/api/orgs/"
265                 + org_id + "/devicestate/"
266                 + deviceID[i - groupID.count()];
267         getConStateUrl();
268
269     } else {
```

```
270         /* 订阅设备的消息 */
271         for (int j = 0; j < deviceNumber.count(); j++) {
272             QByteArray cmd;
273             cmd[0] = 0x01;
274             sendCmd(deviceNumber[j], cmd);
275         }
276 
277         timer->stop();
278     }
279 
280     i++;
281 }
282 
283 /* 订阅指定设备的消息，cmd = 0x01 */
284 void Webapi::sendCmd(QString number, QByteArray cmd)
285 {
286     QStringList list = number.split("");
287     for (int i = 0; i < list.count(); i++) {
288         if (!list[i].isEmpty()) {
289             cmd.append(list[i]);
290         }
291     }
292 
293     webSocket->sendBinaryMessage(cmd);
294 }
295 
296 /* 发送消息到指定设备，cmd = 0x03 */
297 void Webapi::sendCmdMessage(QString number,
298                             QByteArray cmd, QString message)
299 {
300     QStringList list = number.split("");
301     for (int i = 0; i < list.count(); i++) {
302         if (!list[i].isEmpty()) {
303             cmd.append(list[i]);
304         }
305     }
306 
307     cmd.append(message);
308 
309     webSocket->sendBinaryMessage(cmd);
310 }
311 
312 void Webapi::whichDeviceNameSendCmd(QString name,
313                                     QString message) {
```

```
314
315     for (int i = 0; i < deviceName.count(); i++) {
316         if (name ==  deviceName[i]) {
317             QByteArray cmd;
318             cmd[0] = 0x03;
319             sendCmdMessage(deviceNumber[i], cmd, message);
320             break;
321         }
322     }
323 }
```

第 20 行，需要填写自己的原子云平台帐号 api_token 信息，请在原子云》帐号信息中查看！

剩余的代码都按照原子云平台 API 文档编写，首先是通过网络请求 networkRequest，访问需要访问的地址，然后通过网络回应对象 newReply 来接收网络回复的结果。结果是 JSION 格式的文本，笔者使用正则表达式提取回复的内容，作为下一个地址的参数，如此反复，就可以将原子云服务器的帐号下的设备信息提取出来。

第 159 行，提取出来的信息转交 webSocket 对象，让 webSocket 获取原子云平台的鉴权，就可以实现通信了。

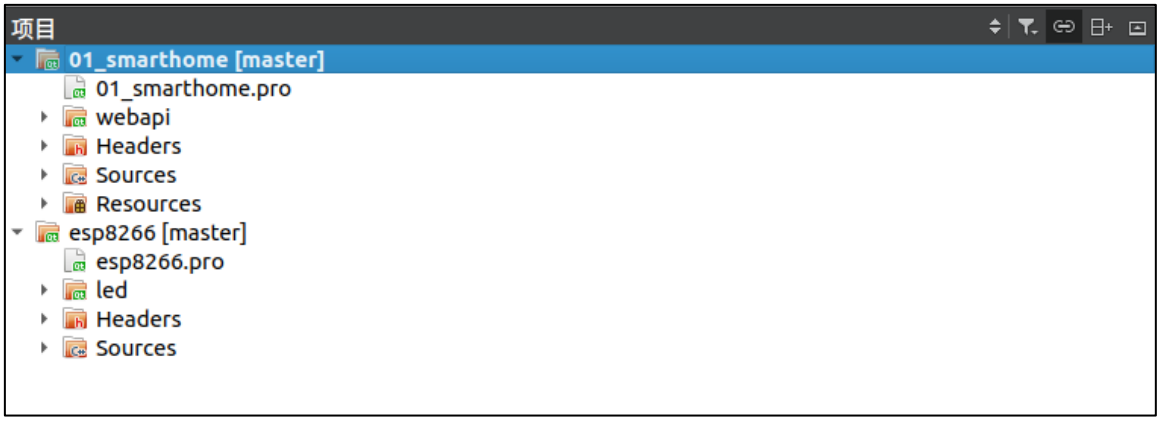
流程都是按照原子云平台 API 文档的走，剩下的就是 webSocket 通信了，与 TCP，UDP 的 socket 通信相似，这里就不多解释了，和第十一章的 TCP/UDP Socket 通信内容相似。重点是流程，再参考代码看。

24.6 物联网项目综合测试

打开 4/01_smarthome/01_smarthome/01_smarthome.pro 项目，此项目为智能家居物联网 UI 界面控制端。

打开 4/01_smarthome/esp8266/esp8266.pro 项目，此项目设备端（被控端）。

打开上面两个项目如下。



项目文件夹下内容解释：

01_smarthome 项目下：

- webapi 文件夹为原子云平台的应用程序，主要用来与原子云通信。
- Headers 文件夹为界面设计的头文件。

- Sources 文件夹为界面设计的源文件。

esp8266 项目下：

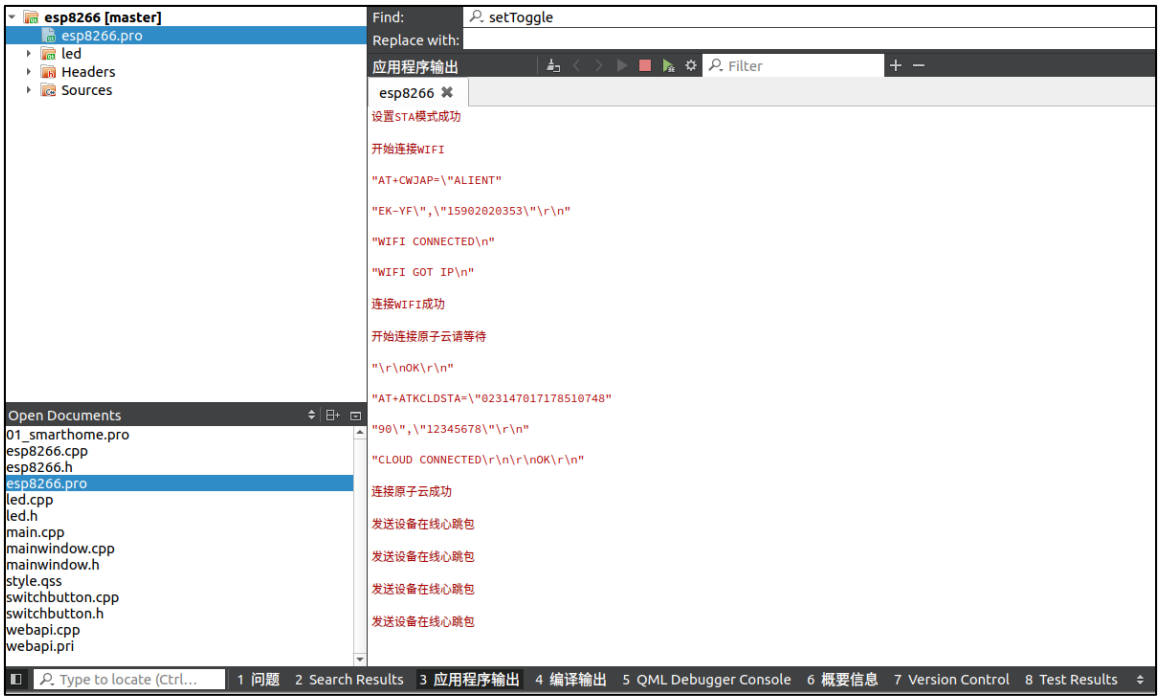
- led 文件夹为 I.MX6U 开发板控制 LED 的接口程序。
- Headers 文件夹为 esp8266 通信的头文件。
- Sources 文件夹为 esp8266 通信的源文件（使用串口通信）。

24.6.1 Ubuntu 上运行

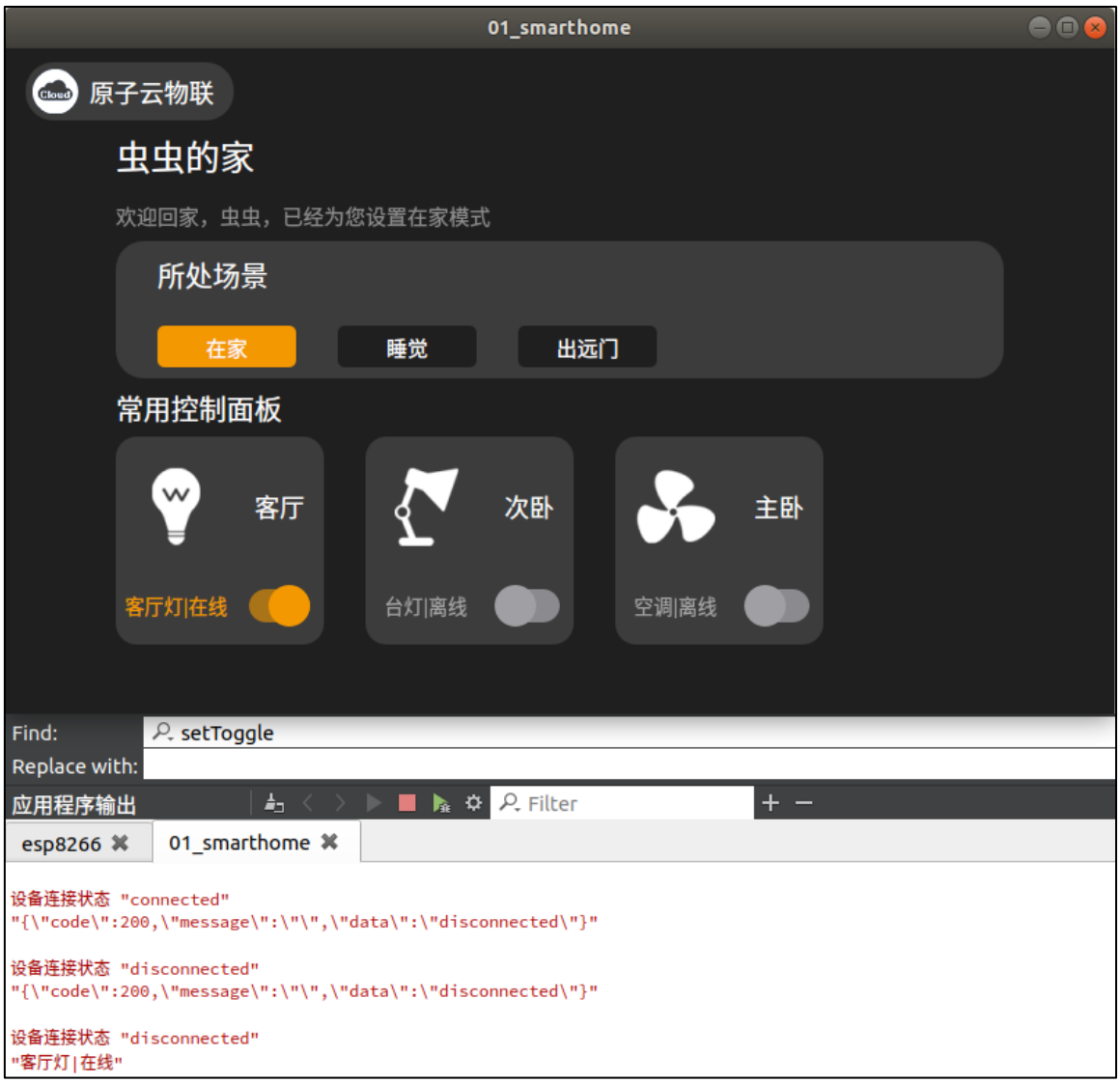
运行 esp8266.pro 项目前请确认 ESP8266 WIFI 模块已经使用 USB-TTL 模块通过 T 口 USB 连接线连接到 Ubuntu 上，并赋予/dev/ttyUSB0 权限才能访问这个串口。运行后串口终端打印信息如下。若模块已经连接上原子云，重新运行程序时，需要将模块断电复位才能再次通信！

注意，需要修改程序中的个人的本地 WIFI 帐号及密码，以及原子云上的设备的编号及密码，**特别注意，这个设备必须在一个新增的分组下**。设备在分组才能被后面的原子云 API 接口获取到。

运行 esp8266.pro 项目，Qt Creator 的应用程序窗口输出如下。连接原子云成功后就会启用定时器，每 15s 向原子云服务器发送一次心跳包，webSocket 应用程序收到后就会显示此设备在线。



运行 01_smarthome.pro 项目前请确认原子云》帐号信息处的 API TOKEN 信息，请填写自己的原子云帐号 API TOKEN。否则您将访问到笔者的原子云帐号的设备信息。



点击“客厅”开关按钮，可以看到 esp8266 设备应用程序收到开关信息。这个信息是通过 UI 界面应用程序发送到原子云服务器，然后原子云服务器转发给 WIFI 模块设备的。

24.6.2 ALPHA/Mini 开发板运行

正点原子的 ALPHA/Mini Linux 开发板交运行上面的 01_smarthome 项目和 esp8266 项目后，运行的结果与 Ubuntu 上面的是一样的，注意，主控开发板需要先连网(通过网线|USB WIFI 模块|4G 模块|SDIO WIFI 模块)，可以将正点原子的 ESP8266 WIFI 插在同一块开发板上进行实验，这样，点击 UI 界面上的开关按钮，就相当于发信息到原子云上，原子云再转发给 ESP8266 WIFI 模块，进而控制开发板上的 LED。这样就实现了将开发板实现连接到原子云里了。

第二十五章 语音识别项目

我们知道 AI 智能音箱已经在我们生活中不少见，也许我们都玩过，智能化非常高，功能强大，与我们平常玩的那种蓝牙音箱，Wifi 音箱有很大的区别，AI 智能在哪里呢？语音识别技术和云端技术，主要由主控芯片，麦克风阵列，功率放大，codec，触控电路，LED 阵列组成。

AI 音箱对传统音箱主要有两大块的技术区别，一块是语音信号的前处理，包括回声消除、波速成型、音源定位、降噪、去混响、自动语音电平控制这块是偏硬件的控制。还有一块是智能语音交互，包括语音关键词搜索、本地语音识别、声纹识别、语音合成。

AI 智能音箱的芯片方案商：联发科，全志科技，瑞芯微等等，语音识别都有现成的方案商。他们的麦克风阵列方案，有 2 麦，4 麦，6 麦，7 + 1 麦等等。

写上面这些是让读者了解一下专业 AI 音箱方案与我们在正点原子 Linux 开发板想实现语音识别的差别在哪里。我们在正点原子 Linux 开发板上实现语音识别项目（功能），就不能与专业的 AI 音箱对比了。硬件资源有限，开发板只有一个麦头（咪头座），没有那些硬件控制消除回声，降噪等等。不过笔者在上面调用百度语音 API 识别语音，识别率还是挺高的。

下面就与大家一起在正点原子 Linux IMX6U 开发板上实现语音识别功能吧！注意：正点原子 MINI I.MX6U 开发板没有音频芯片，不支持此实验，只有正点原子 I.MX6U ALPHA 开发板支持。

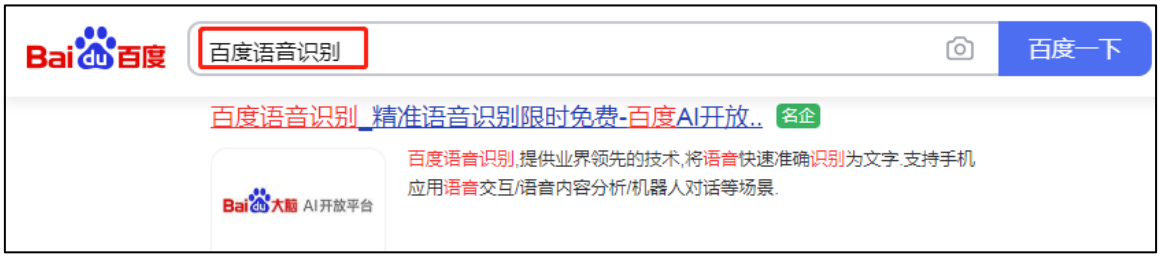
本章简介如下：

（1）介绍百度语音技术账号申请，及简单介绍调用流程。

（2）用 Qt 编写示例程序。流程如下，录制音频后，发送调用百度语音识别 API 接口，识别并返回结果。支持语音控制正点原子 I.MX6U 开发板上的 LED 控制，其他设备可以自行拓展。

25.1 语音识别产品申请帐号

语音识别技术产品，有讯飞，百度等厂家，我们可以购买或者免费试用他们的产品。可以直接到他们的官网上查看，有使用技术文档。下面我们以百度语音识别技术产品为例子。可以在浏览器输入搜索“百度语音识别”，就可以找到百度 AI 开放平台。



点击进去就可以看到他的技术文档链接位置。如下图。

或者直接打开 https://ai.baidu.com/ai-doc/SPEECH/Ek39uxgre 就可以跳转到百度 AI 开放平台》帮助文档》语音技术页面。如下图。

请仔细阅读百度语音技术的文档，里面写的非常详细，还有例子下载参考。

笔者阅读总结，想要使用百度语音识别接口，需要根据上面图中的新手指南注册百度帐号，领取免费额度及创建中文普通话应用（创建前先领取免费额度（180 天免费额度，可调用约 5

万次左右，详细请看免费额度说明））。记住自己的密钥。**请自行完成及创建百度帐号，按照百度帮助文档里的步骤，**领取免费额度及创建中文普通话应用，**获取密钥！程序里需要用到自己的密钥**。笔者提供的密钥是百度语音识别例程里的，如果开发次数超了可能就不能使用了。程序中只需要 API Key 与 Secret Key。注意获取 Access Token 时有效期为 30 天，到期后需要在程序里重新获取新的 token。

更多参考请查看百度 AI 接入指南。

注意，帮助文档里提及 SDK 包，有 LinuxC++SDK 包支持，但是目前仅支持 X64（x86-64）CPU 架构的 Linux 操作系统。LinuxSDK 仅支持在线语音识别，固定长语音模式。简单的说就是还不支持 ARM 架构的 SDK 包。

25.2 百度语音识别流程及示例简介

在百度 AI 帮助文档里可以看见如下重要信息。

请认真阅读调用流程，了解操作过程，对下面理解笔者编写 Qt 调用百度语音 API 的例子会有一定的帮助。

总结：调用流程需要仔细阅读，百度提供了示例 Demo 代码，可以看到里面支持很多种编程语言编写的 API 请求相关示例 demo 代码。没有直接 C++相关的代码。C 语言是 C++语言的子集，我们可以直接参考 C 语言编写的例子（请自行查阅及参考百度提供的 C 语言编写的 API 请求相关示例 demo 代码）来编写 Qt 调用语音识别 API。（备注：其他语言编写的例子不在我们教程范围。）识别的音频格式支持如上，我们可以知道一些重要的信息是支持采样率 16000、8000 的固定值，16bit 深的单声道，音频长度最长 60 秒。格式支持 wav，恰好正点原子 Linux I.MX6U 开发板系统支持 wav 格式播放及录制（详细请看【正点原子】I.MX6U 用户快速体验 V1.x.pdf 测试音频部分）。

备注：由于百度语音识别的 API 例子放在 github（开源网站），国外网站的原因，可能打开失败，请多次尝试，如果一直无法访问，那么我们直接往下看使用笔者编写 Qt 的示例吧。不能访问的话，笔者也没办法的。

25.3 百度短语音识别 API 接口

源码路径为 4/02_asr_demo/asr/asr.h，内容如下。asr 是语音识别功能 demo，（asr 译作自动语音识别技术即 automatic speech recognition）

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   asr
* @brief         asr.h
```

```
    * @author       Deng Zhimao
    * @email        1252699831@qq.com
    * @net          www.openedv.com
    * @date         2021-06-03
    ****************************************************************/
1   #ifndef ASR_H
2   #define ASR_H
3
4   #include <QWidget>
5
6   #include <QNetworkAccessManager>
7   #include <QNetworkReply>
8
9   #include <QJsonDocument>
10  #include <QJsonParseError>
11  #include <QJsonObject>
12  #include <QJsonArray>
13  #include <QHostInfo>
14
15  #include <QFile>
16
17  class Asr : public QWidget
18  {
19      Q_OBJECT
20
21  public:
22      Asr(QWidget *parent = nullptr);
23      ~Asr();
24
25      /* 请求网络 */
26      void requestNetwork(QString, QByteArray);
27
28      /* 获取识别结果 */
29      void getTheResult(QString fileName);
30
31  private:
32      /* 存储获取 tokenUrl 地址 */
33      QString tokenUrl;
34
35      /* 存储 serverapi 地址 */
36      QString serverApiUrl;
37
38      /* 最终需要访问 token 的地址 */
```

```
39      QString accessToken;
40
41      /* 获取 token 的接口*/
42      const QString token_org =
"https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials
&client_id=%1&client_secret=%2&";
43
44      /* 填写网页上申请的 appkey 如 g_api_key="g8eBUMSokVB1BHGmgxxxxxx" */
45      const QString api_key = "kVcnfD9iW2XVZSMaLMrtLYIz";
46
47      /* 填写网页上申请的 APP SECRET 如
$secretKey="94dc99566550d87f8fa8ece112xxxxx" */
48      const QString secret_key = "O9o1O213UgG5LFn0bDGNtoRN3VWl2du6";
49
50      /* 百度服务器 API 接口，发送语音可返回识别结果 */
51      const QString server_api =
"http://vop.baidu.com/server_api?dev_pid=1537&cuid=%1&token=%2";
52
53      /* 网络管理 */
54      QNetworkAccessManager *networkAccessManager;
55
56      QString getJsonValue(QByteArray ba, QString key);
57
58      QFile file;
59
60  private slots:
61
62      /* 准备读取响应返回来的数据 */
63      void readyReadData();
64
65      /* 响应完成处理 */
66      void replyFinished();
67
68  signals:
69      void asrReadyData(QString);
70
71  };
72  #endif // ASR_H
```

第 45 行，请填写读者自己在网页上申请的 API Key。以防万一示例中的 API Key 过期不可用！

第 47 行，请填写读者在网页上申请的 Secret Key。以防万一示例中的 Secret Key 过期不可用！

其他地址由来是见百度给出的 Demo 示例，及百度的帮助文档。这里就不详细说了。原理与上一章原子云 API 接口相似。不过百度语音识别需要通过自己的帐号，指定地址获取访问的 Token 源地址，然后将得到的 Access Token 地址与语音识别服务器地址拼接，发送语音到服务器，就可以返回识别的结果了。详细请参考源码 4/02_asr_demo/asr/asr.cpp。

25.4 录制 wav 音频

在 12.5 小节，已经介绍过开发板如何录制音频文件了，详细请看 12.5 小节，就不详细介绍了，注意需要修改的地方如下。因为百度语音识别支持采样率 16000、8000 的固定值，16bit 深的单声道，音频长度最长 60 秒。格式支持 wav，pcm 等格式。我们需要修改录制音频格式为 wav 格式，通道为单声道，采样率为 16000，源码如下，已用红色字体标出。录制的音频文件保存为本地 16k.wav 文件。

源码路径为 4/02_asr_demo/audiorecorder/audiorecorder.cpp。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   audiorecorder
* @brief         audiorecorder.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-06-04
*****************************************************************/
#include "audiorecorder.h"
#include <QDebug>
#include <QUrl>
#include <QDateTime>
#include <QDir>
#include <QCoreApplication>

static qreal getPeakValue(const QAudioFormat &format);
static QVector<qreal> getBufferLevels(const QAudioBuffer &buffer);

template <class T>
static QVector<qreal> getBufferLevels(const T *buffer, int frames, int channels);

AudioRecorder::AudioRecorder(QWidget *parent)
{
    Q_UNUSED(parent);

    /* 录制音频的类 */
    m_audioRecorder = new QAudioRecorder(this);

```

```
21      /* 用于探测缓冲区的数据 */
22      m_probe = new QAudioProbe(this);
23
24      /* 信号槽连接，更新录音 level 显示 */
25      connect(m_probe, &QAudioProbe::audioBufferProbed,
26              this, &AudioRecorder::processBuffer);
27
28      /* 设置探测的对象 */
29      m_probe->setSource(m_audioRecorder);
30
31      /* 扫描本地声卡设备 */
32      devicesVar.append(QVariant(QString()));
33      for (auto &device: m_audioRecorder->audioInputs()) {
34          devicesVar.append(QVariant(device));
35          //qDebug()<<"本地声卡设备："<<device<<endl;
36      }
37
38      /* 音频编码 */
39      codecsVar.append(QVariant(QString()));
40      for (auto &codecName: m_audioRecorder->supportedAudioCodecs()) {
41          codecsVar.append(QVariant(codecName));
42          //qDebug()<<"音频编码："<<codecName<<endl;
43      }
44
45      /* 容器/支持的格式 */
46      containersVar.append(QVariant(QString()));
47      for (auto &containerName: m_audioRecorder->supportedContainers())
{
48          containersVar.append(QVariant(containerName));
49          //qDebug()<<"支持的格式："<<containerName<<endl;
50      }
51
52      /* 采样率 */
53      sampleRateVar.append(QVariant(0));
54      /* 百度语音识别只支持 8000、 16000 采样率 */
55      sampleRateVar.append(QVariant(8000));
56      sampleRateVar.append(QVariant(16000));
57      for (int sampleRate: m_audioRecorder->supportedAudioSampleRates())
{
58          sampleRateVar.append(QVariant(sampleRate));
59          //qDebug()<<"采样率："<<sampleRate<<endl;
60      }
```

```
61
62
63      /* 通道 */
64      channelsVar.append(QVariant(-1));
65      channelsVar.append(QVariant(1));
66      channelsVar.append(QVariant(2));
67      channelsVar.append(QVariant(4));
68
69      /* 质量 */
70      qualityVar.append(QVariant(int(QMultimedia::LowQuality)));
71      qualityVar.append(QVariant(int(QMultimedia::NormalQuality)));
72      qualityVar.append(QVariant(int(QMultimedia::HighQuality)));
73
74      /* 比特率 */
75      bitratesVar.append(QVariant(0));
76      bitratesVar.append(QVariant(32000));
77      bitratesVar.append(QVariant(64000));
78      bitratesVar.append(QVariant(96000));
79      bitratesVar.append(QVariant(128000));
80
81      /* 录音类信号槽连接 */
82      connect(m_audioRecorder, &QAudioRecorder::durationChanged,
83              this, &AudioRecorder::updateProgress);
84  }
85
86  AudioRecorder::~AudioRecorder()
87  {
88  }
89
90
91  void AudioRecorder::startRecorder()
92  {
93      /* 备注：录音需要设置成 16000 采样率和通道数为 1，
94       * 保存为 wav 文件需要设置成 audio/x-wav（container 文件格式） */
95
96      /* 如果录音已经停止，则开始录音 */
97      if (m_audioRecorder->state() == QMediaRecorder::StoppedState) {
98          /* 设置默认的录音设备 */
99          m_audioRecorder->setAudioInput(devicesVar.at(0).toString());
100
101         /* 下面的是录音设置 */
102         QAudioEncoderSettings settings;
103         settings.setCodec(codecsVar.at(0).toString());
```

```
104         settings.setSampleRate(sampleRateVar[2].toInt());
105         settings.setBitRate(bitratesVar[0].toInt());
106         settings.setChannelCount(channelsVar[1].toInt());
107         settings.setQuality(QMultimedia::EncodingQuality(
108                                 qualityVar[0].toInt()));
109
110         /* 以恒定的质量录制，可选恒定的比特率 */
111
settings.setEncodingMode(QMultimedia::ConstantQualityEncoding);
112
113         /* I.MX6ULL 第 20 个支持的格式为 audio/x-wav */
114         QString container = containersVar.at(20).toString();
115
116         /* 使用配置 */
117         m_audioRecorder->setEncodingSettings(settings,
118                                              QVideoEncoderSettings(),
119                                              container);
120         /* 录音保存为 16k.wav 文件 */
121
m_audioRecorder->setOutputLocation(QUrl::fromLocalFile(tr("./16k.wav"))
);
122
123         /* 开始录音 */
124         m_audioRecorder->record();
125     }
126 }
127
128 void AudioRecorder::stopRecorder()
129 {
130     /* 停止录音 */
131     m_audioRecorder->stop();
132 }
133
134
135 void AudioRecorder::updateProgress(qint64 duration)
136 {
137     Q_UNUSED(duration);
138
139     if (m_audioRecorder->error()
140             != QMediaRecorder::NoError)
141         return;
142
143     /* 打印录制时长 */
```

```
144     //qDebug()<<duration / 1000<<endl;
145 }
146
147
148 void AudioRecorder::clearAudioLevels()
149 {
150     //...
151 }
152
153 // This function returns the maximum possible sample value for a given
audio format
154 qreal getPeakValue(const QAudioFormat& format)
155 {
156     // Note: Only the most common sample formats are supported
157     if (!format.isValid())
158         return qreal(0);
159
160     if (format.codec() != "audio/pcm")
161         return qreal(0);
162
163     switch (format.sampleType()) {
164     case QAudioFormat::Unknown:
165         break;
166     case QAudioFormat::Float:
167         if (format.sampleSize() != 32) // other sample formats are not
supported
168             return qreal(0);
169         return qreal(1.00003);
170     case QAudioFormat::SignedInt:
171         if (format.sampleSize() == 32)
172             return qreal(INT_MAX);
173         if (format.sampleSize() == 16)
174             return qreal(SHRT_MAX);
175         if (format.sampleSize() == 8)
176             return qreal(CHAR_MAX);
177         break;
178     case QAudioFormat::UnSignedInt:
179         if (format.sampleSize() == 32)
180             return qreal(UINT_MAX);
181         if (format.sampleSize() == 16)
182             return qreal(USHRT_MAX);
183         if (format.sampleSize() == 8)
184             return qreal(UCHAR_MAX);
185         break;
```

```
186     }
187
188     return qreal(0);
189 }
190
191 // returns the audio level for each channel
192 QVector<qreal> getBufferLevels(const QAudioBuffer& buffer)
193 {
194     QVector<qreal> values;
195
196     if (!buffer.format().isValid() || buffer.format().byteOrder() !=
QAudioFormat::LittleEndian)
197         return values;
198
199     if (buffer.format().codec() != "audio/pcm")
200         return values;
201
202     int channelCount = buffer.format().channelCount();
203     values.fill(0, channelCount);
204     qreal peak_value = getPeakValue(buffer.format());
205     if (qFuzzyCompare(peak_value, qreal(0)))
206         return values;
207
208     switch (buffer.format().sampleType()) {
209     case QAudioFormat::Unknown:
210     case QAudioFormat::UnSignedInt:
211         if (buffer.format().sampleSize() == 32)
212             values = getBufferLevels(buffer.constData<quint32>(),
buffer.frameCount(), channelCount);
213         if (buffer.format().sampleSize() == 16)
214             values = getBufferLevels(buffer.constData<quint16>(),
buffer.frameCount(), channelCount);
215         if (buffer.format().sampleSize() == 8)
216             values = getBufferLevels(buffer.constData<quint8>(),
buffer.frameCount(), channelCount);
217         for (int i = 0; i < values.size(); ++i)
218             values[i] = qAbs(values.at(i) - peak_value / 2) / (peak_value
/ 2);
219         break;
220     case QAudioFormat::Float:
221         if (buffer.format().sampleSize() == 32) {
222             values = getBufferLevels(buffer.constData<float>(),
buffer.frameCount(), channelCount);
223             for (int i = 0; i < values.size(); ++i)
```

```
224                 values[i] /= peak_value;
225         }
226         break;
227     case QAudioFormat::SignedInt:
228         if (buffer.format().sampleSize() == 32)
229             values = getBufferLevels(buffer.constData<qint32>(),
buffer.frameCount(), channelCount);
230         if (buffer.format().sampleSize() == 16)
231             values = getBufferLevels(buffer.constData<qint16>(),
buffer.frameCount(), channelCount);
232         if (buffer.format().sampleSize() == 8)
233             values = getBufferLevels(buffer.constData<qint8>(),
buffer.frameCount(), channelCount);
234         for (int i = 0; i < values.size(); ++i)
235             values[i] /= peak_value;
236         break;
237     }
238
239     return values;
240 }
241
242 template <class T>
243 QVector<qreal> getBufferLevels(const T *buffer, int frames, int
channels)
244 {
245     QVector<qreal> max_values;
246     max_values.fill(0, channels);
247
248     for (int i = 0; i < frames; ++i) {
249         for (int j = 0; j < channels; ++j) {
250             qreal value = qAbs(qreal(buffer[i * channels + j]));
251             if (value > max_values.at(j))
252                 max_values.replace(j, value);
253         }
254     }
255
256     return max_values;
257 }
258
259 void AudioRecorder::processBuffer(const QAudioBuffer& buffer)
260 {
261     /* 根据通道数目需要显示 count 个 level */
262     int count = buffer.format().channelCount();
263     /* 打印通道数 */
```

```
264     Q_UNUSED(count);
265     // qDebug()<<"通道数"<<count<<endl;
266
267     /* 设置 level 的值 */
268     QVector<qreal> levels = getBufferLevels(buffer);
269     for (int i = 0; i < levels.count(); ++i) {
270         /* 打印音量等级 */
271         // qDebug()<<"音量等级"<<levels.at(i)<<endl;
272     }
273 }
```

4/02_asr_demo/audiorecorder/audiorecorder.cpp 主要提供了一个 startRecorder() 和 stopRecorder()的接口，录音保存的文件为可执行程序当前路径下的 16k.wav 文件。startRecorder() 和 stopRecorder()分别是开始录音和停止录音。

第 54~56 行，增加 8000 和 16000 的支持项。

第 104 行，设置为下标为 2 的项，也就是 16000 采样率。

第 106 行，设置通道数下标为 1 的项，也就是单通道。

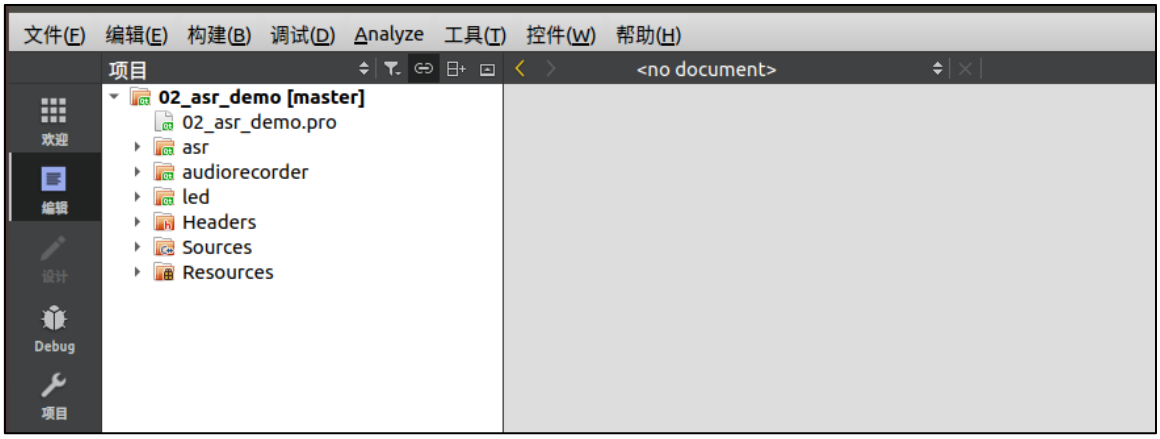
第 114 行，设置文件容器/格式，为 audio/x-wav 格式（项的下标为 20）。设置此格式会保存 wav 后缀的文件。

25.5 语音界面 UI 开发

项目路径为 4/02_asr_demo/02_asr_demo/02_asr_demo.pro，先看项目界面。项目界面如下，界面简洁大气，界面中间用了一个立体的素材，点击后可以旋转，给人一种智能化的感觉，点击时还会有音效提示，文本提示“请点击，开始说话…”，点击后，提示“正在听您说话，请继续…”，录制 8s 左右的音频，等待返回识别结果即可。编写设计完成的效果不错！请自行查阅源码，掌握了本教程前面第七章的内容，就可以理解这个界面是如何设计的。

25.6 语音识别项目综合测试

打开 4/02_asr_demo/02_asr_demo/02_asr_demo.pro 项目，此项目为语音识别 UI 界面。打开项目如下图。



项目文件夹下内容解释：

02_asr_demo 项目下：

- asr 文件夹为语音识别的应用程序，主要用来与将录制的音频发送到百度云语音识别服务器上，然后返回识别结果。
- aduiorecorder 文件夹为录制 wav 音频的文件夹。主要是用来录制 wav 音频。
- led 文件夹为 I.MX6U 开发板控制 LED 的接口程序。
- Headers 文件夹为界面设计的头文件。
- Sources 文件夹为界面设计的源文件。

25.6.1 Ubuntu 上运行

Ubuntu 运行后界面如下，注意，Ubuntu 需要联网！Ubuntu 上理论上是能录制音频识别返回结果的，但是教程主要写正点原子 I.MX6U 开发板上的语音识别项目。限于笔者手上没有可用电脑麦克风，估计读者也没有，电脑配置麦克风输入后可以自行测试。运行之后可以看到下面的界面。Windows 不作讲解！请到下面小节使用正点原子 I.MX6U ALPHA 开发板运行体验识别效果！

25.6.2 ALPHA 开发板上运行

本例适用于正点原子 I.MX6U ALPHA 开发板！请使用正点原子 **I.MX6U 的出厂系统**进行测试！

请使用正点原子的 I.MX6U 的出厂时的系统测试！

请使用正点原子的 I.MX6U 的出厂时的系统测试！

请使用正点原子的 I.MX6U 的出厂时的系统测试！

重要的事情是说三遍！

开始录音前，需要根据正点原子 I.MX6U 用户快速体验手册，第 3.15 小节进行测试板子的录音功能。确保能正常录音，再交叉编译此 Qt 应用程序到开发板上运行。如何交叉编译 Qt 应用程序到开发板，请看【正点原子】I.MX6U 出厂系统 Qt 交叉编译环境搭建 V1.x 版本。

在正点原子 I.MX6U 开发板上运行此录音程序，需要先配置是麦克风（板子上的麦头）。

麦头录音，则在板子上运行开启麦头录音的脚本。

```
/home/root/shell/audio/mic_in_config.sh
```

交叉编译到开发板上运行效果如下。下面的图都是开发板上的截图。

程序初始化时。（注意开发板先插上网线联网！确保能上网！）

点击中间的图标后，注意，请在点击 1.5~2s 后再说话，点击时有音效提醒，避免把音效录进去。整个录音过程是 8s 左右。

识别返回结果的过程很快，识别率也挺高，如下图，笔者说了一句“正点原子”，语音识别返回“正点原子”的结果。注意，识别是中文标准普通话。请尽量说一些日常话语，避免说生僻语句，特殊的方言等。识别常见问题请查看百度 AI 开发平台的帮助文档。

再点击，再次进行语音识别，话语中，包含“开灯”，那么即可点亮板子上的 LED。点亮后，再次进行语音识别，话语中包含“关灯”，即可熄灭板子上的 LED。

“开灯”识别结果。

“关灯”识别结果。

本示例仅供学习参考使用，如需要用到开发上，请购买百度或其他开放平台的语音识别产品。

第二十六章 APP 主界面开发项目

本章与大家一起开发 APP 主界面。Qt C++提供了像 QStackedWdget 与 QTableView 这种控件可以方便的切换页面，但是这种切换页面的方法比较生硬，不能像手机一样滑动，往往这种界面就会给用户较差的体验感。所以在传统的 Qt C++里（Qt Quick 除外），Qt 没有提供能够滑动的页面界面。如今是移动设备到处都是，指尖触控交互，好的操作界面能给用户优越的体验感。

在 Qt C++编程滑动屏幕界面这方面里，笔者也参考过许多网上的文章。发现很多都是使用 QPainter 结合 QMouseMove 事件来重绘屏幕或者移动屏幕的，这种方式代码量较长，而且不容易移植。很多都是固定了界面的页数及大小，不能使用布局等等。于是笔者结合自己的开发经验自己写一个滑动界面的类，可以很方便的增加页面，能跟随页面的大小变化而变化，是笔者原创的一个作品，可以方便大家移植到需要写 APP 主界面的程序里。同时笔者也花了时间写了一个好看的车载音乐主界面（只有界面，非功能的实现，笔者模仿网上的车载界面，用 Qt 实现的），读者可以参考源代码来开发自己的界面。同时也是笔者带领读者开发 APP 主界面的入门操作。不过这个入门操作是相当有难度了！因为这个界面比较复杂，内容较多。读者主要了解下 26.1 小节滑动界面的使用，然后自己写一个只带一两个按钮的界面自己经测试一下效果滑动效果，再细读源码。源码不细节讲解，最好的方式就是阅读源码理解整个流程。

26.1 滑动界面

本节代码程序是滑动页面的设计，下面贴出主程序代码。然后介绍实现思路。

项目路径为 4/03_appmainview/slidepage/slidepage.pro。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   slidepage
* @brief         slidepage.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-06-09
*****************************************************************/
#include "slidepage.h"
#include <QDebug>
#include <QPropertyAnimation>

SlidePage::SlidePage(QWidget *parent):
    QWidget(parent),
    pageIndex(0),
    pageCount(0),
    draggingFlag(false)
{
    pageIndicator.clear();
    this->setMinimumSize(400, 300);
    this->setAttribute(Qt::WA_TranslucentBackground, true);

    scrollArea = new QScrollArea(this);
    scrollArea->setAlignment(Qt::AlignCenter);

    mainWidget = new QWidget();
    mainWidget->setStyleSheet("background: transparent");

    scrollArea->setWidget(mainWidget);
    scrollArea->setStyleSheet("background: transparent");

    bottomWidget = new QWidget(this);
    bottomWidget->setStyleSheet("background: transparent");

    bottomHBoxLayout = new QHBoxLayout();
    bottomWidget->setLayout(bottomHBoxLayout);
    bottomHBoxLayout->setContentsMargins(0, 0, 0, 0);
    bottomHBoxLayout->setAlignment(Qt::AlignCenter);

```

```
32      /* 关闭滚动条显示 */
33      scrollArea->setVerticalScrollBarPolicy(
34                  Qt::ScrollBarAlwaysOff);
35      scrollArea->setHorizontalScrollBarPolicy(
36                  Qt::ScrollBarAlwaysOff);
37
38      /* 滚屏对象 */
39      scroller = QScroller::scroller(scrollArea);
40      QScroller::ScrollerGestureType gesture =
QScroller::LeftMouseButtonGesture;
41      scroller->grabGesture(scrollArea, gesture);
42
43      /* 获取属性 */
44      QScrollerProperties properties = scroller->scrollerProperties();
45
46      /* 设置滑动的时间，值越大，时间越短 */
47      properties.setScrollMetric(QScrollerProperties::SnapTime, 0.5);
48
49      /* 设置滑动速度 */
50      properties.setScrollMetric(QScrollerProperties::MinimumVelocity,
1);
51      scroller->setScrollerProperties(properties);
52
53      /* 布局 */
54      hBoxLayout = new QHBoxLayout();
55
56      hBoxLayout->setContentsMargins(0, 0, 0, 0);
57      hBoxLayout->setSpacing(0);
58
59      mainWidget->setLayout(hBoxLayout);
60
61      /* 定时器，用于判断用户是否是拖动屏幕，区分滑动，超过 300ms 表示拖动 */
62      timer = new QTimer(this);
63
64      connect(scrollArea->horizontalScrollBar(),
SIGNAL(valueChanged(int)), this, SLOT(hScrollBarValueChanged(int)));
65      connect(scroller, SIGNAL(stateChanged(QScroller::State)), this,
SLOT(onStateChanged(QScroller::State)));
66      connect(timer, SIGNAL(timeout()), this, SLOT(onTimerTimeOut()));
67      connect(this, SIGNAL(currentPageIndexChanged(int)), this,
SLOT(onCurrentPageIndexChanged(int)));
68  }
69
```

```
70  SlidePage::~SlidePage()
71  {
72  }
73
74  void SlidePage::addPage(QWidget *w)
75  {
76      /* 布局里添加页面 */
77      hBoxLayout->addWidget(w);
78      /* 页数加一 */
79      pageCount++;
80      QLabel *label = new QLabel();
81      label->setPixmap(QPixmap(":/icons/indicator1.png"));
82      pageIndicator.append(label);
83      bottomHBoxLayout->addWidget(label);
84  }
85
86  void SlidePage::resizeEvent(QResizeEvent *event)
87  {
88      Q_UNUSED(event)
89      scrollArea->resize(this->size());
90      /* mainWidget 需要比 scrollArea 小 */
91      mainWidget->resize(this->width() * pageCount, this->height() - 4);
92      if (pageCount == 0)
93          qDebug()<<"当前页面总数为 0，请使用 addPage()方法添加页面再使用!
"<<endl;
94      else
95          onCurrentPageIndexChanged(0);
96      bottomWidget->setGeometry(0, this->height() - 20, this->width(),
20);
97  }
98
99  void SlidePage::hScrollBarValueChanged(int)
100 {
101     /* 滑动时判断当前页的下标 */
102     pageIndex= scrollArea->horizontalScrollBar()->value() /
this->width();
103     pageIndex = scrollArea->horizontalScrollBar()->value()
104           >= (pageIndex * this->width() + this->width() * 0.5) ?
pageIndex + 1 : pageIndex;
105
106 }
107
108 void SlidePage::onStateChanged(QScroller::State state)
```

```
109 {
110     static int pressedValue = 0;
111     static int releasedValue = 0;
112     static int currentPageIndex = 0;
113 
114     /* 如果页面数为 0，返回，不做任何操作 */
115     if (pageCount == 0)
116         return;
117 
118     /* 松开 */
119     if (state == QScroller::Inactive) {
120         /* 停止定时器，防止检测到界面是缓慢拖动状态 */
121         timer->stop();
122         /* 记录松开时的坐标 */
123         releasedValue = QCursor::pos().x();
124 
125         if (pressedValue == releasedValue)
126             return;
127 
128         /* 判断按下与松开的距离，首先先判断是不是拖动状态，如果是拖动状态，
pageIndex 不会变化 */
129         if (!draggingFlag) {
130             if (pressedValue - releasedValue > 20 && currentPageIndex ==
pageIndex)
131                 pageIndex++;
132             else
133                 pageIndex--;
134         }
135 
136         /* 页面下标判断 */
137         if (pageIndex == -1)
138             pageIndex = 0;
139 
140         if (pageIndex >= pageCount)
141             pageIndex = pageCount - 1;
142 
143         /* 动画 */
144         QPropertyAnimation *animation = new
QPropertyAnimation(scrollArea->horizontalScrollBar(), "value");
145         animation->setDuration(200);
146 
animation->setStartValue(scrollArea->horizontalScrollBar()->value());
147         animation->setEasingCurve(QEasingCurve::OutCurve);
```

```
148         animation->setEndValue(pageIndex * this->width());
149         animation->start();
150 
151         if (currentPageIndex != pageIndex) {
152             /* 发送当前页面的位置信号 */
153             emit currentPageIndexChanged(pageIndex);
154         }
155 
156         /* 重新赋值*/
157         pressedValue = 0;
158         releasedValue = 0;
159         draggingFlag = false;
160     }
161 
162     /* 按下 */
163     if (state == QScroller::Pressed) {
164         pressedValue = QCursor::pos().x();
165         currentPageIndex = scrollArea->horizontalScrollBar()->value() /
this->width();
166         /* 按下如果超过 300ms，表示用户在拖动 */
167         timer->start(300);
168     }
169 }
170 
171 void SlidePage::onTimerTimeOut()
172 {
173     /* 拖动标志位 */
174     draggingFlag = true;
175     timer->stop();
176 }
177 
178 int SlidePage::getPageCount()
179 {
180     return pageCount;
181 }
182 
183 int SlidePage::getCurrentPageIndex()
184 {
185     return pageIndex;
186 }
187 
188 void SlidePage::onCurrentPageIndexChanged(int index)
189 {
```

```
190     for (int i = 0; i < pageIndicator.count(); i++) {
191         if (i == index)
192 
pageIndicator[i]->setPixmap(QPixmap(":/icons/indicator2.png"));
193         else
194 
pageIndicator[i]->setPixmap(QPixmap(":/icons/indicator1.png"));
195     }
196 }
```

可以看到主程序代码量不多，仅 200 行不到就可以完成这样的滑动页面设计。（上面代码单行较长，由于 Word 排版，会换行，若影响阅读，请打开源项目查看）。比网上用 QPainter 与 QMouseMove 实现页面滑动省了很多代码。

我们在第七章学习过了 QScrollArea，这是一个滑动的界面，初学时我们只知道它只能通过两边的滑块来滚动界面。实际上，查 Qt 帮助文档资料得知，Qt 有一个 QScroller 类就可以实现滑动界面。它能控制那种带滚动条的类的界面的滑动。详细可以看源码 39 行至 41 行。

滑动界面部分是由 QScroller 类控制。同时通过设置 QScrollArea-> horizontalScrollBar()这个滚动条的 value 值就可以控制界面所处位置了。原理看似简单，详细需要查看程序，请读者带着程序的原理通过细读程序源码来分析。

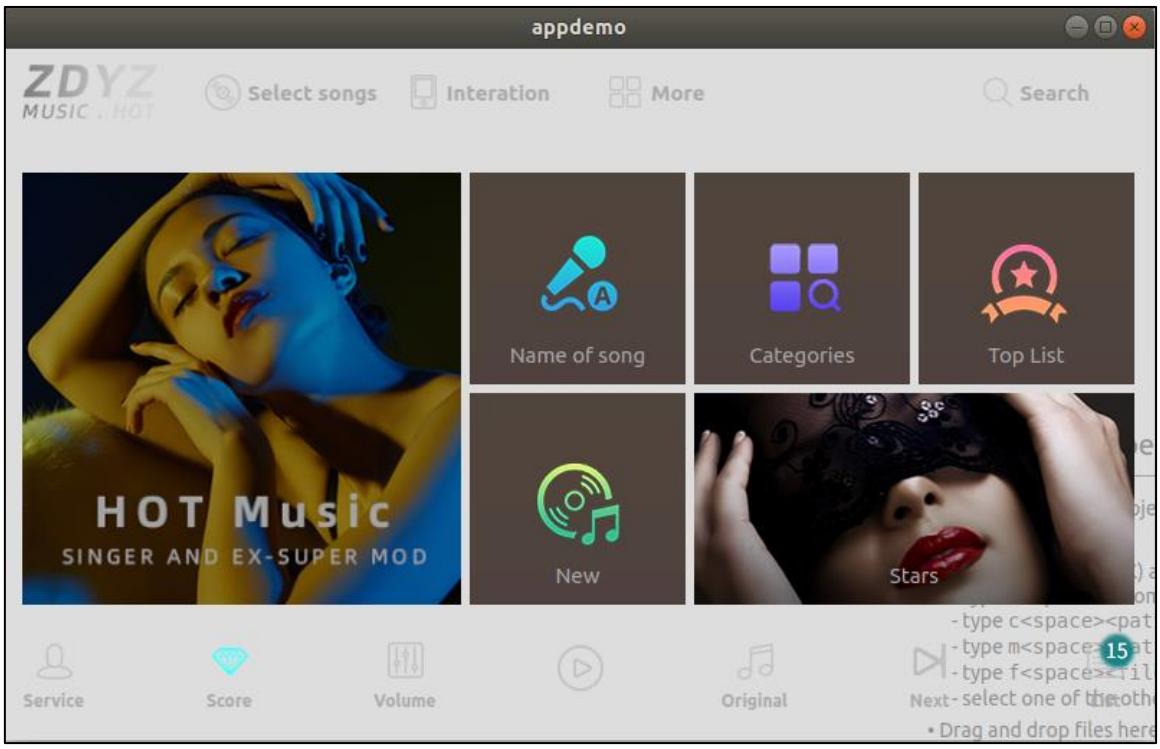
注意！运行这个项目时，您会发现这是一个空白的窗口！因为这只是一个滑动页面的类，我们还没有在里面添加内容！所以接着往下看，我们先开发一页 APP 界面。

运行项目效果如下。因为我们还没有给滑动页面类加内容。所以是无法滑动的。继续往下看。

26.2 APP 界面开发

本小节实现了一个车载音乐界面的界面开发。注意只是界面开发，不实现其功能。读者可以参考来实现自己的界面。开发界面除了自己的想法，还需要有美工基础，我们前面已经学习过布局了，APP 界面，主要都是一些布局的设计。这里就不贴代码了。（代码都是一些布局设置，与按钮排布，读者有前面的基础后，自行查阅项目源码）。

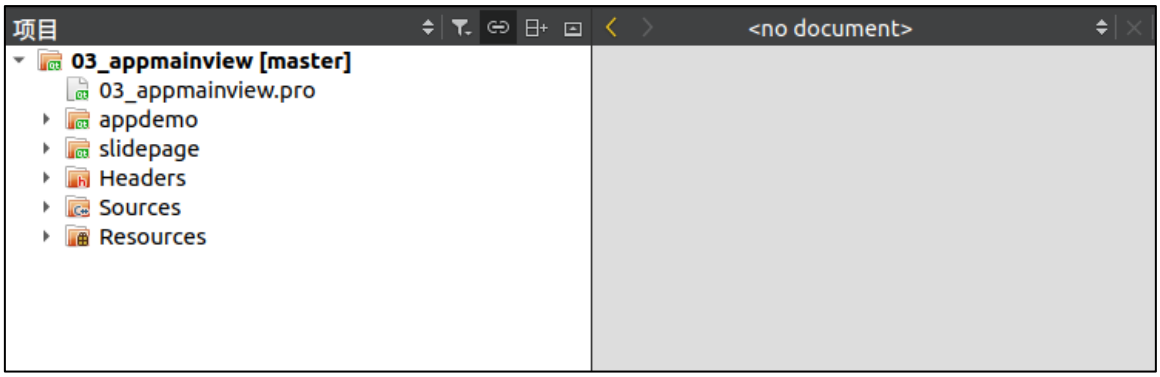
项目路径为 4/03_appmainview/appdemo/appdemo.pro。项目运行效果如下。



项目运行后，可以看到这样的一页这样的 APP 主界面，注意不能点击，这只是界面而已！读者就可以仿照这样的一个 APP 界面来开发自己的应用界面了。开发时再通过链接到界面上的按钮的点击信号，比如点击按钮后打开新的页面，这样就可以完成一个完整的点击交互事件了！

26.3 APP 主界面项目综合测试

项目路径为 4/03_appmainview/03_appmainview /03_appmainview.pro。打开此项目您将看到如下。



项目文件夹下内容解释：

03_appmainview.pro 项目下：

- appdemo 文件夹为车载音乐 APP 页面，只是界面，不带实际功能！
- slidepage 文件夹为笔者原创的一个滑动页面类，在这个类里，我们可以使用 addPage()方法来添加页面，当添加的页面大于 2 页时，就可以滑动切换页面了。
- Headers 文件夹为 03_appmainview.pro 的头文件。
- Sources 文件夹为 03_appmainview.pro 的源文件。

- Resource 文件夹为 03_appmainview.pro 的源文件。主要是存放一张背景图片。

  源程序路径为 4/03_appmainview/03_appmainview /widget.cpp。源码如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
* @projectName   03_appmainview
* @brief         widget.cpp
* @author        Deng Zhimao
* @email         1252699831@qq.com
* @net           www.openedv.com
* @date          2021-06-09
*****************************************************************/
#include "widget.h"
#include <QPushButton>
#include <QDebug>

AppMainView::AppMainView(QWidget *parent)
{
    this->setParent(parent);
    this->setGeometry(0, 0, 800, 480);
    this->setMinimumSize(800, 480);

    bgWidget = new QWidget(this);
    bgWidget->setStyleSheet("border-image: url(:/images/bg.png)");

    mySlidePage = new SlidePage(this);
    mySlidePage->resize(this->size());

    for (int i = 0; i < 3; i++) {
        appDemo[i] = new AppDemo();
        mySlidePage->addPage(appDemo[i]);
    }

}

AppMainView::~AppMainView()
{
}


void AppMainView::resizeEvent(QResizeEvent *event)
{
    Q_UNUSED(event)
    mySlidePage->resize(this->size());
    bgWidget->resize(this->size());
```

```
34  }
```

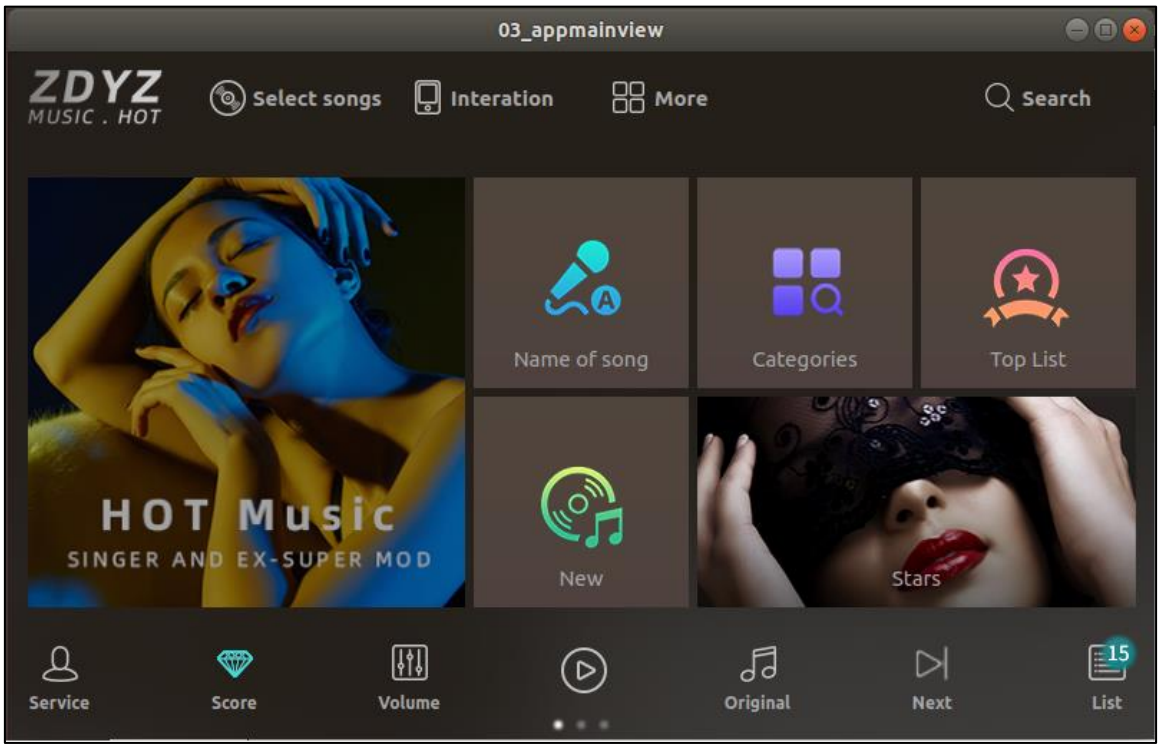
可以看到 01_appmainview.pro 项目里的内容较少，因为 01_appmainview.pro 这个项目是调用了前面两节的项目，所以在这个项目里看到的内容较少。

第 14 行，调用了 26.1 小节里的滑动界面类，实例化了一个对象 mySlidePage。我们只需要往这个对象了添加 Widget 类型对象，就相当于为这个滑动界面类添加了对象！使用笔者封装好的一个 addPage(QWidget *)就可以轻松的添加页面了。

17~20 行，APP 页面，这里是一个 AppDemo 类，然后实例化了 3 个对象，也就是说会有 3 个 APP 页面，而且他们都是一样的。第 19 行，如笔者所说，使用滑动界面对象 mySlidePage 使用方法 addPage()，将 3 个 APP 页面添加到滑动界面对象里，然后就可以左右滑动这个 APP 主界面了。

26.3.1 程序运行效果

程序运行效果如下。可以看到有 3 个页面，而且他们的样子都是一样的，这是笔者为了方便测试滑动页面的效果而添加这三个页面。在界面的下方，如果有细心的读者发现，这里还有三个分页逗点，可以说明这是三个页面。读者可以运行程序来体验滑动效果及界面设计的效果。可以交叉编译到开发板上运行，无论是界面设计感还是流畅度都很不错。



第二十七章 车牌识别项目

车牌识别在我们生活中经常看到。例如停车场，高速入口，这些地方就需要车牌识别。很多车牌方案商都有成熟的车牌识别技术，他们是靠这个吃饭的，不开源。当然网上也有一些开源的车牌识别算法可以参考，但是我们 Qt 教程里不是讲车牌识别算法，因为过于复杂，内容多。所以本教程也是使用方案商提供的接口来做车牌识别，例程比较简单，百度 AI 接口车牌识别率非常高，毕竟能让别人花钱的东西是不一样的。下面就让我们使用 Qt 来调用百度 AI 车牌识别的接口来做个例子吧，实际上与二十五章语音识别项目非常相似。

本章简介如下：

（1）介绍百度车牌识别功能申请，及简单介绍调用流程。

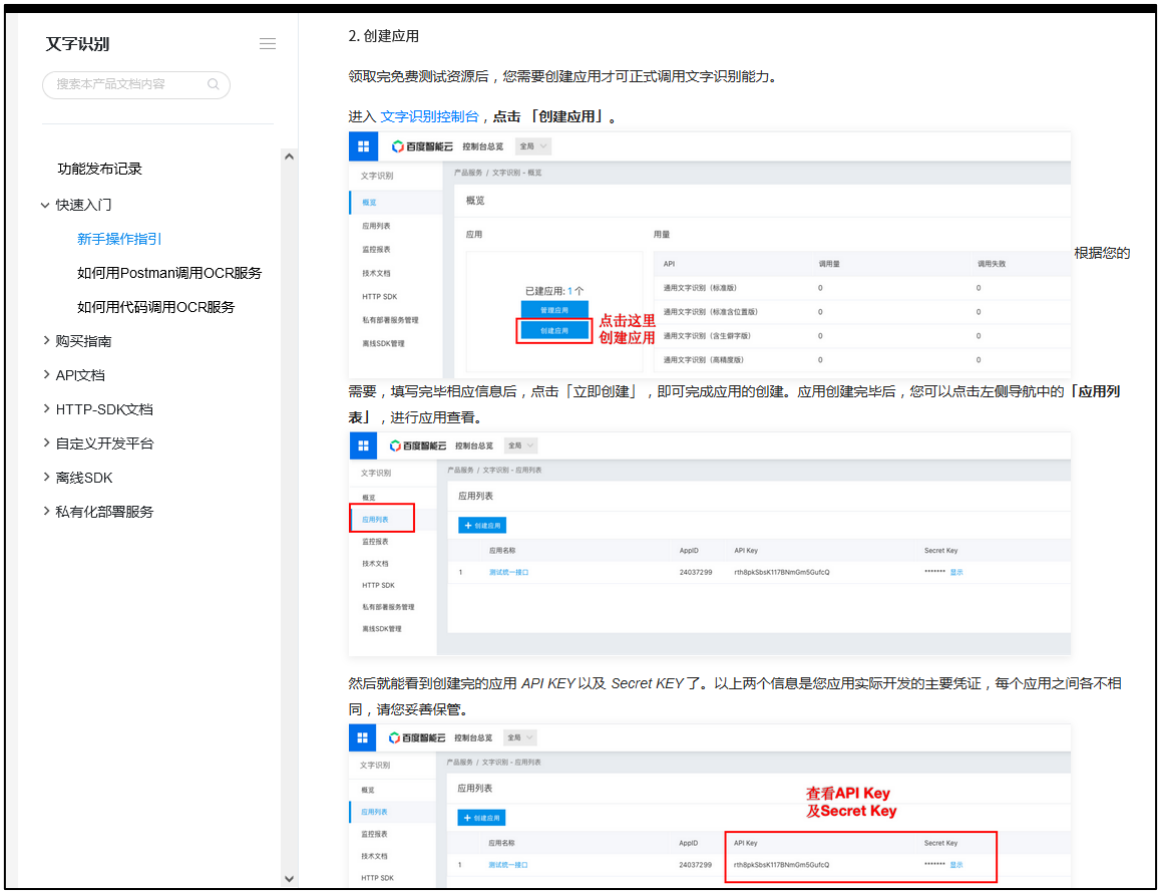
（2）用 Qt 编写示例程序。流程如下，将本地车牌照片（JPG），发送到百度车牌识别服务器，然后由百度服务器返回车牌识别结果。

27.1 车牌识别产品申请

在浏览器输入“百度车牌识别”搜索，击进去，

点击“技术文档”，其中您需要参阅下图的“技术文档”来查看使用手册。它里面会介绍 API 相关使用方法和领取免费的测试资源方法。

我们只需要领取车牌识别功能资源，根据百度提供的“技术文档”可以知道，在文字识别》购买指南》免费测试资源，可以看到未实名认证的用户免费领取车牌识别功能为 200 次/月，实名认证是 1000 次/月。根据自己所需，勾情实名认证。在快速入门》新手操作指引处，我们在领取资源页面找到车牌识别，勾选“车牌识别”，然后点 0 元领取即可。(PS 百度技术文档是教您全部领取的。)



点击上图的文字识别控制台（蓝色字体）就可以看跳转到创建文字识别的应用了。如下图。点击创建应用后。在文字识别应用列表中，默认会把文字识别的应用全部勾选了，如下图。接着填上相关信息，完成创建即可。

创建完成后，查看应用的 API Key 和 Secret Key。因为下面的程序需要用到。

27.2 百度车牌识别 API 接口

源码路径为 04/04_lpr_demo/ocr/ocr.h，内容如下。ocr 是笔者编写的车牌识别功能 demo，（ocr 原意光学字符识别即 Optical Character Recognition）。使用此程序需要修改自己应用的 client_id（API Key）和 client_secret（ Secret Key）。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 2021-2030. All rights reserved.
* @projectName   ocr
* @brief         ocr.h
* @author        Deng Zhimao
* @email         dengzhimao@alientek.com
* @link          www.openedv.com
* @date          2021-11-17
*****************************************************************/
#ifndef OCR_H
```

```
2   #define OCR_H
3
4   #include <QNetworkAccessManager>
5   #include <QNetworkReply>
6
7   #include <QJsonDocument>
8   #include <QJsonParseError>
9   #include <QJsonObject>
10  #include <QJsonArray>
11  #include <QHostInfo>
12
13  #include <QFile>
14  #include <QImage>
15
16  class Ocr : public QObject
17  {
18      Q_OBJECT
19
20  public:
21      Ocr(QObject *parent = nullptr);
22      ~Ocr();
23
24      /* 请求网络 */
25      void requestNetwork(QString, QByteArray);
26
27      /* 获取识别结果 */
28      void getTheResult(QString fileName);
29      void getTheResult(QImage image);
30
31  private:
32      /* 存储获取 tokenUrl 地址 */
33      QString tokenUrl;
34
35      /* 存储 serverapi 地址 */
36      QString serverApiUrl;
37
38      /* 最终需要访问 token 的地址 */
39      QString accessToken;
40
41      /* 获取 token 的接口*/
42      const QString token_org =
"https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials
&client_id=%1&client_secret=%2&";
43
44      /* 填写网页上申请的 appkey 如 api_key ="g8eBUMSokVB1BHGmgxxxxxx" */
45      const QString api_key = "填写自己的 APP KEY";
46
47      /* 填写网页上申请的 APP SECRET 如
secret_key="94dc99566550d87f8fa8ece112xxxxx" */
48      const QString secret_key = "填写自己的 APP SECRET";
49
50      /* 百度服务器 API 接口，发送图片可返回识别结果 */
51      const QString server_api =
"https://aip.baidubce.com/rest/2.0/ocr/v1/license_plate?access_token=%1
";
52
53      /* 网络管理 */
54      QNetworkAccessManager *networkAccessManager;
```

```
55
56      /* 处理 Json 数据 */
57      QString getJsonValue(QByteArray ba, QString key);
58
59      /* 处理 Json 数据 */
60      QString getJsonValue(QByteArray ba, QString key1, QString key2);
61
62  public slots:
63
64      /* 准备读取响应返回来的数据 */
65      void readyReadData();
66
67      /* 响应完成处理 */
68      void replyFinished();
69
70      /* 开始识别 */
71      void readyToDetection(QString);
72      void readyToDetection(QImage);
73
74  signals:
75      /* 识别到车牌，发送信号 */
76      void ocrReadyData(QString);
77
78  };
79  #endif // OCR_H
```

第 45 行，请填写读者自己在网页上申请的 API Key。本例不提供开放的 API Key，请填写自己个人申请的！

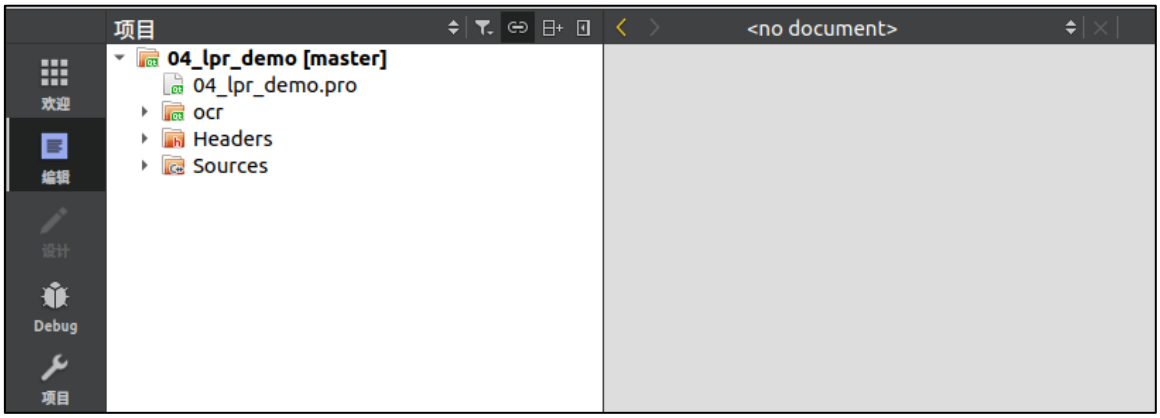
第 48 行，请填写读者在网页上申请的 Secret Key。本例不提供开放的 Secret Key，请填写自己个人申请的！

其他地址由来在百度车牌识别的帮助文档，它已经解释的很详细，我们只要阅读它的帮助文档可知，这里就不详细说了。发送要识别的图片到服务器，就可以返回识别的结果了。详细请参考源码 04/04_lpr_demo/ocr/ocr. cpp。

27.3 车牌识别项目综合测试

打开 04/ 04_lpr_demo/04_lpr_demo/ 04_lpr_demo.pro 项目，此项目为车牌识别主体程序。

打开项目如下图。



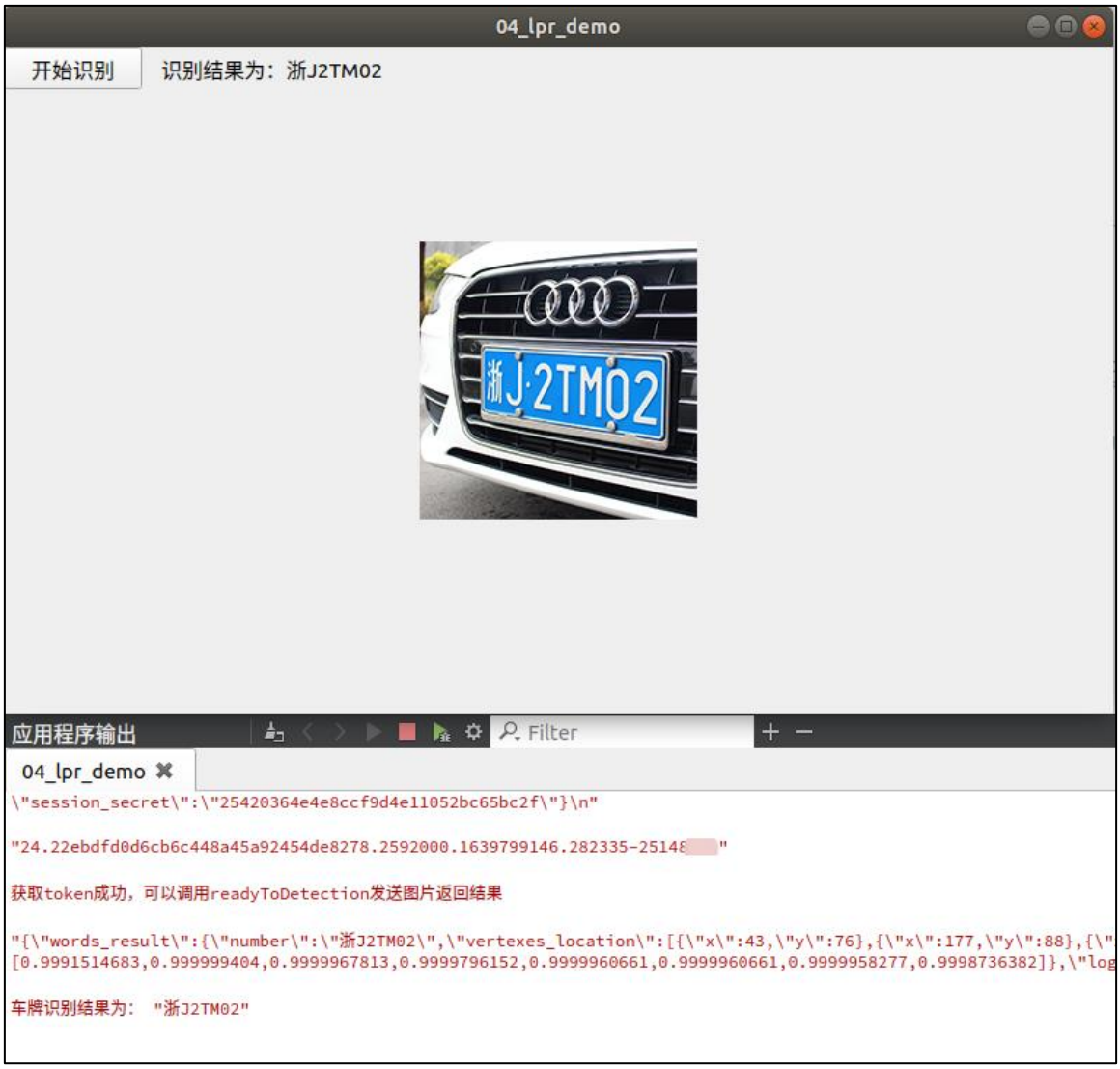
项目文件夹下内容解释，源码在工程中查看，有详细注释，这里就不贴代码了。

04_lpr_demo 项目下：

- ocr 文件夹为车牌识别的应用程序，主要发送本地图片到百度车牌识别服务器上，然后返回识别结果。
- Headers 文件夹为界面的头文件。
- Sources 文件夹为界面的源文件。

我们直接构建工程，注意：运行之前需要把项目下的 image 文件夹拷贝到可执行程序的同级目录下。此 image 文件夹目录存放的是要识别的车牌图片，若想替换自己的图片，需要把要识别的图片替换到 image 文件夹下的 carlpr.jpg，名字要相同。

运行结果如下。注意：Ubuntu 或者开发板需要联网！本程序适用于 Ubuntu18 和 I.MX6U 开发板，Ubuntu16 会报 Openssl 版本错误！请注意！这就是为什么建议需要读者使用和笔者相同版本 Ubuntu 来开发的原因了！毕竟初学者处理不同的开发环境还是有点难度的！



运行程序后，界面会显示要识别的车牌图片，如果没有显示，是因为您没有把 image 文件夹拷贝到可执行程序的同级路径下。程序运行会根据您在百度上申请的车牌识别服务上的 API

Key 与 Secert Key 来获取 token，获取 token 后，我们点击左上角的按钮就可以发送界面上显示的图片到百度车牌识别服务器上，服务器即会返回车牌识别结果。过程非常简单。如上图，识别的结果在界面的左上角。

总结，我们使用了百度车牌识别的接口来完成这个车牌识别的过程，在这里我们虽然我们不能学到车牌识别的算法，但是我们也学会了怎么根据百度 ai 的帮助文档去调用。并不是所有的车牌识别算法都开源，毕竟也是别人的成果，别人的成果也是要付出的。此例程仅供参考使用。实际用到项目上请购买相关的车牌识别产品。

本章可能遇到的问题？

Q: 是否可以采用摄像头捕获的图片，也就是拍照。

A: 当然是可以的啊，笔者考虑到某些读者没有摄像头，也不想把程序写的过于复杂。所以就使用本地的了。

第二十八章 视频监控项目

常见的视频监控和视频直播就是使用 RTMP 和 RTSP 流媒体协议等。

RTSP （Real-Time Stream Protocol）由 Real Networks 和 Netscape 共同提出的，基于文本的多媒体播放控制协议。RTSP 定义流格式，流数据经由 RTP 传输；RTSP 实时效果非常好，适合视频聊天，视频监控等方向。

RTMP（Real Time Message Protocol） 由 Adobe 公司提出，用来解决多媒体数据传输流的多路复用（Multiplexing）和分包（packetizing）的问题，优势在于低延迟，稳定性高，支持所有摄像头格式，浏览器加载 flash 插件就可以直接播放。

RTSP 和 RTMP 的区别：

RTSP 虽然实时性最好，但是实现复杂，适合视频聊天和视频监控；RTMP 强在浏览器支持好，加载 flash 插件后就能直接播放，所以非常火，相反在浏览器里播放 rtsp 就很困难了。

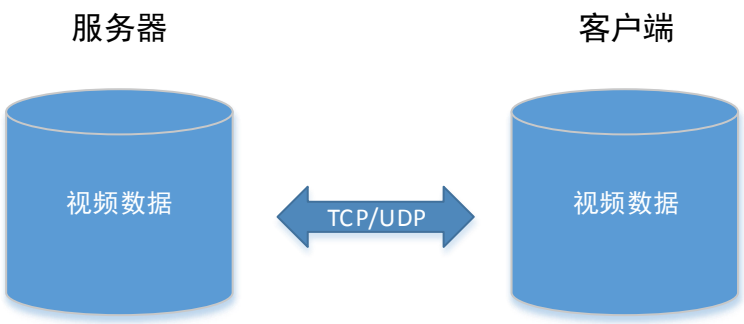
说了上面那么多，是为了让大家了解当下比较火的流媒体协议。一般这种协议需要搭配服务器如 Nginx 服务器和 ffmpeg 工具来使用。当然像 ffmpeg 这种强大的工具我们是写不出来的，这种强大的工具已经发展的很好，专门是做音视频处理方案的。有兴趣的可以了解下。在我们 I.MX6U 的出厂系统就支持 RTMP+FFMPEG+NGINX 推流了。这就需要一个客户端和一个服务端。毫无疑问，I.MX6U 充当服务器，客户端用现成的软件 VLC 播放器可以播放。如果我们不想用这种方式来做视频监控，有没有简单的一些方案在 Qt 里能使用的呢。答案是有的。看下面的方案。

在这本 Qt 教程里，前面第十一章，我们已经学习过网络编程。既然学习过网络编程，使用 UDP 或者 TCP 传输，使用 Qt 封装好 TCP/IP 协议 Socket 抽象层接口来通信。那么我们也可以使用它们来发送图像数据啊。这样，服务器和客户端我们完全可以使用 Qt 来写了。实际上 RTSP 协议和 RTMP 协议都使用 TCP/IP 协议，在传输层使用了 UDP 或 TCP，所以说 RTSP 和 RTMP 协议是更高的一层协议。

本章需要使用正点原子的 I.MX6U 开发板及正点原子 OV5640 摄像头。请使用正点原子最新的出厂系统固件（2021.11 月份有更新出厂固件，为了适配 Qt 教程和 C 应用教程使用 OV5640 摄像头做了优化，请在 https://alientek-linux.coding.net/public/下载更新您的出厂固件）。注：本次不对 USB 摄像头做适配。原因是 USB 摄像头采集的是 YUYV 数据，转 RGB 数据则会消耗大量 CPU，会卡顿很多，延时性大。推荐使用正点原子的 OV5640 摄像头，支持 RGB 格式采集，Qt 显示也是 RGB 格式的，所以采集的数据直接给屏幕显示即可，相对 YUYV 摄像头，消耗 CPU 更少，显示更流畅！

28.1 实验流程图

数据传输原理流程图。



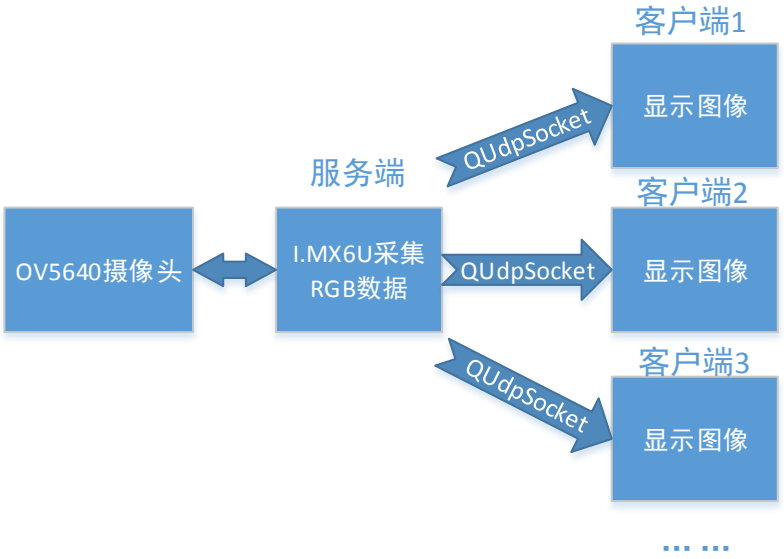
我们需要完成的任务如下。

（1） 服务器采集摄像头的数据。

（2） 处理视频数据转交给 Socket，由 TCP/UDP 传输。

（3） 客户端接收视频数据。

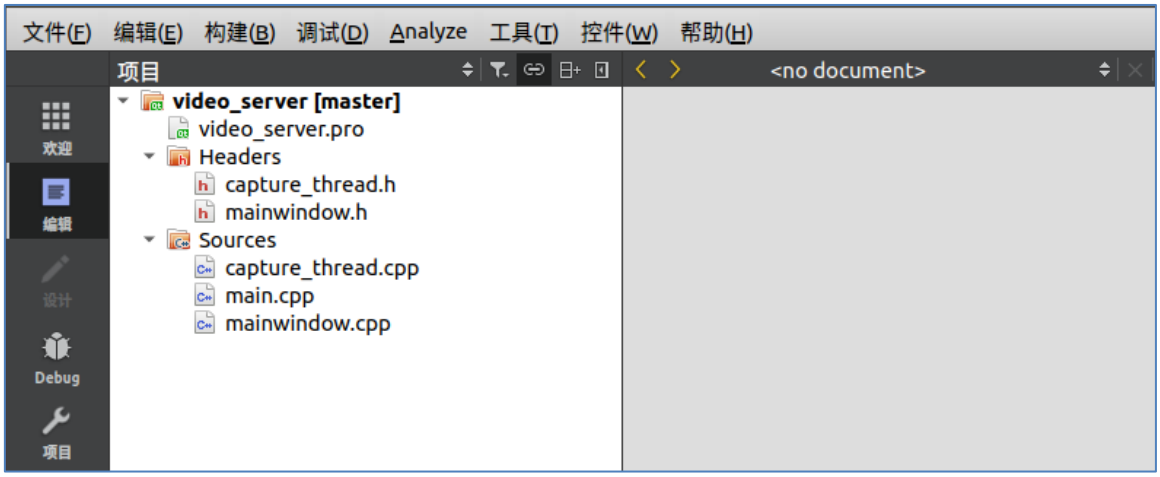
本章项目流程图。



本项目，使用的是 OV5640 摄像头（500 万像素），本次我们使用 Qt 的 QUdpSocket 传输，这样就可以有多个客户端可以在不用连接的情况下接收到图像数据。服务端为正点原子的 I.MX6U 开发板，客户端可以为计算机或者其他 I.MX6U 开发板或其他能运行 Qt 的 ARM 开发板。本章不再讲解 QUdpSocket 的用法，请在前面第十一章网络编程里学习！

28.2 视频监控之服务端

源码路径为 04/05_video_surveillance/video_server。服务端工程结构如下。



项目文件夹下内容解释，源码在工程中查看，有详细注释。

video_server 项目下：

- capture_thread.h 这个是摄像头捕获线程的头文件。摄像头采集数据，我们开启一个线程来获取。
- capture_thread.cpp 摄像头线程的主程序。

其他文件是界面文件，不用再介绍。

capture_thread.h 的内容如下。

```
/*****************************************************************
Copyright © Deng Zhimao Co., Ltd. 2021-2030. All rights reserved.
* @projectName   video_server
* @brief         capture_thread.h
* @author        Deng Zhimao
* @email         dengzhimao@alientek.com
* @link          www.openedv.com
* @date          2021-11-19
*****************************************************************/
#ifndef CAPTURE_THREAD_H
#define CAPTURE_THREAD_H

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#ifdef linux
#include <linux/fb.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <linux/videodev2.h>
#include <linux/input.h>
#endif

#include <QThread>
#include <QDebug>
#include <QPushButton>
#include <QImage>
#include <QByteArray>
```

```
24  #include <QBuffer>
25  #include <QTime>
26  #include <QUdpSocket>
27
28  #define VIDEO_DEV          "/dev/video1"
29  #define FB_DEV             "/dev/fb0"
30  #define VIDEO_BUFFER_COUNT  3
31
32  struct buffer_info {
33      void *start;
34      unsigned int length;
35  };
36
37  class CaptureThread : public QThread
38  {
39      Q_OBJECT
40
41  signals:
42      /* 准备图片 */
43      void imageReady(QImage);
44      void sendImage(QImage);
45
46  private:
47      /* 线程开启 flag */
48      bool startFlag = false;
49
50      /* 开启广播 flag */
51      bool startBroadcast = false;
52
53      /* 本地显示 flag  */
54      bool startLocalDisplay = false;
55      void run() override;
56
57  public:
58      CaptureThread(QObject *parent = nullptr) {
59          Q_UNUSED(parent);
60      }
61
62  public slots:
63      /* 设置线程 */
64      void setThreadStart(bool start) {
65          startFlag = start;
66          if (start) {
67              if (!this->isRunning())
68                  this->start();
69          } else {
70              this->quit();
71          }
72      }
73
74      /* 设置广播 */
75      void setBroadcast(bool start) {
76          startBroadcast = start;
77      }
78
79      /* 设置本地显示 */
80      void setLocalDisplay(bool start) {
81          startLocalDisplay = start;
```

```
82      }
83  };
84
85  #endif // CAPTURE_THREAD_H
```

可以看出服务端开辟了一个线程来采集数据，这样的界面和数据逻辑就分开了，界面就不会卡顿。线程部分我们也已经在第十章已经学习过了，比较简单。采集摄像头数据部分在源码里分析即可，有详细注释。

28.3 视频监控之客户端

源码路径为 04/05_video_surveillance/video_client/video_client.pro。客户端的 mainwindow.h 的源码如下。客户端代码量更小，只需要接收并处理图像数据即可！

```
    /**************************************************************
    Copyright © Deng Zhimao Co., Ltd. 2021-2030. All rights reserved.
    * @projectName   video_client
    * @brief         mainwindow.h
    * @author        Deng Zhimao
    * @email         dengzhimao@alientek.com
    * @link          www.openedv.com
    * @date          2021-11-20
    **************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QUdpSocket>
6   #include <QLabel>
7
8   class MainWindow : public QMainWindow
9   {
10      Q_OBJECT
11
12  public:
13      MainWindow(QWidget *parent = nullptr);
14      ~MainWindow();
15
16  private:
17      /* 用于接收数据 */
18      QUdpSocket *udpSocket;
19
20      /* 显示接收的图像数据 */
21      QLabel *videoLabel;
22
23      void resizeEvent(QResizeEvent *event) override;
24
25  private slots:
26      /* 图像更新 */
27      void videoUpdate();
28  };
29  #endif // MAINWINDOW_H
```

28.4 视频监控综合测试

交叉编译服务端程序到 I.MX6U 开发板，插上 OV5640 摄像头，确保开发板接上路由器，运行后界面如下。界面使用了两个 QCheckBox，一个 QCheckBox 用于开启本地图像显示，也就是显示摄像头捕获的内容。另一个 QCheckBox 用于开启 udp 广播，也就是将摄像头的捕获的内容发送出去。这两个功能默认没有选上，请先点击界面底部的“开启采集摄像头数据”进行数据采集，然后再选上需要开启的功能。注意，由于 I.MX6U 性能有限，同时开启本地显示和开启广播会造成图像稍延时。单独开启这两个功能，图像则十分流畅！

服务端：

客户端则可以在另一块开发板或者 Ubuntu/Window 上运行，而且可以多个客户端同时运行在不同的机器上。确保和服务端在同一个路由器下（同一局域网）。运行结果如下，接收到服务端的数据，并显示图像，和服务端显示的图像同步，效果不错。

客户端：

开发出来的视频监控项目，对单核 A7 的 CPU 来说，从流畅度这方面要比 RTMP 推流的好的多。流畅度是涉及到数据处理问题，使用 RTMP 如果不恰当，不在乎分辨率和图像的质量，那么它看起来肯定就会“慢”很多。

本项目可以应用于如做室外监控，查看来访客人等，很类似智能楼宇里终端的查看大门来访客人。甚至由读者结合其他例子开发相关的项目。

附录-A