

Dart 简介及基础语法

因为 Flutter 是使用 Dart 语言开发的，所以在开始 Flutter 之前，有必要先了解一下 Dart。

Dart 简介

Dart 是 Google 开发的计算机编程语言，可以用于 移动端、PC 端、WEB、服务器的开发，是全栈式编程语言。虽然语言都是 Dart，但在每个平台上使用的框架不一样，例如在 Flutter 上使用的 Dart for the Flutter，在 Web 端使用的是 Dart for the web，在服务器使用的是 Server-side Dart：

平台	开发语言	框架
Flutter(Android/iOS/Linux/MacOS/Windows/Web)	Dart	Dart for the Flutter
Web	Dart	Dart for the web
服务器	Server-side Dart	

Dart 的历史

- 2011.10：Dart 语言发布
- 2013.10：Dart 1.0
- 2018.8：Dart 2.0

- 2018.12 : Dart 2.1
- 2019.2.27 : Dart 2.2
- 2019.4.21 : Dart 2.3

可能你是在听说了 Flutter 之后，才开始接触 Dart 语言，但是 Dart 语言已经存在并发展很久了，Dart 早在 2011 年就发布了，现在 Dart 的最新版本是 2.3 。

Dart 语言优势

- Dart 中的所有东西都是对象，包括数字、函数等，它们都继承自 Object，并且对象的默认值都是 null（包括数字）。
- Dart 既可以支持 JIT（动态编译），也可以支持 AOT（静态编译）。
- Dart 是强类型语言，但是由于 Dart 可以推断类型，所以也可以支持动态类型，例如 var、dynamic。
- Dart 有强大的异步编程能力。

Dart 用法

我们有了其他语言的基础，学习 Dart 语法很快的，看下面的一段代码，显示了很多 Dart 用法：

```
// 导入库
import 'package:flutter/material.dart';

/**
 * 入口函数
 */
void main() => runApp(MyApp());

/**
```

```

* 定义一个 MyApp Widget
*/
class MyApp extends StatelessWidget {

  var content = 'Dart 语法'; // 声明并初始化变量
  String _name = "by 小德";

  @override
  Widget build(BuildContext context) {

    print('display $content');

    // return a Widget
    return MaterialApp(
      title: "Flutter Demo",
      theme: ThemeData(
        primaryColor: Colors.blue,
      ),
      home: Scaffold(
        appBar: AppBar(title: Text("Flutter
Dart 语法")),
        body: Text(content+_name)
      ),
    );
  }
}

```

- import

导入库。

- 注释

注释，有两种：

语法	含义
<code>/** .. **/</code>	多行注释
<code>//</code>	单行注释

- `=>`

Dart 的箭头语法，即 Lambda 表达式。

- `class`

Dart 的类。

- `extends`

继承。

- `var`

一种声明变量而不指定其类型的方法。

- 以下划线 `_` 开头的类或成员变量是私有的

Dart 没有 `Public`、`Protected`、`Private` 的关键字，在 Dart 里，类或成员变量默认都是 `Public` 的，以下划线 `_` 开头的就是私有的，例如成员变量 `_name` 就是私有的。

- `String`

在声明变量时也可以指定具体类型，`String` 是字符串，还有其他的数据类型：`int`、`double`等。

- `'...'` 或 `"..."`

字符串的使用，用单引号或双引号都行。

- `$variableName` 或 `${expression}`

字符串插值：将变量的值直接插入字符串中。

变量声明

以下是 Dart 中所有变量声明的方法：

```
var content = 'Dart 语法'; // Declare and
initialize a variable.
var switchOn = false;
var current = 0;

String name = "by 小德";
int count = 0;

dynamic example = 'example';

Object index = 100;
```

总共有四种：

1. var

```
var content = 'Dart 语法'; // Declare and
initialize a variable.
var switchOn = false;
var current = 0;
```

使用 var 来声明变量，不需要特别指定变量的数据类型，因为 Dart 会自动推断其数据类型，所以可以使用 var 来定义任何的变量。

问题来了，var 为什么能定义所有的变量？

因为，`var` 并不是直接存储值，而是存储的值的对象的引用，例如：`var content = 'Dart 语法'` 这句，是名字为 `content` 的 `var` 变量存储了值为 `'Dart 语法'` 的 `String` 对象的引用，所以 `var` 才能定义任何变量。

2. 明确数据类型

```
String name = "by 小德";  
int count = 0;
```

就是在声明变量的时候，使用明确的数据类型。

3. dynamic

```
dynamic example = 'example';
```

意思是数据类型是动态可变的，也可以定义任何变量，但是和 `var` 不同的是，`var` 一旦赋值后，就不能改变数据类型了，例如以下用法就是错误的

```
var content = 'Dart 语法';  
content = 1; // 错误的使用方法，content为  
String，不能赋值数字类型
```

但是 `dynamic` 就可以，`dynamic` 可以这么使用：

```
dynamic example = 'example';  
example = 1; // 这个使用方法正确，因为 dynamic  
的类型是动态可变的
```

4. Object

```
Object index = 100;
```

Dart 里所有东西都是对象，是因为 Dart 的所有东西都继承自 Object，因此 Object 可以定义任何变量，而且赋值后，类型也可以更改：

```
Object index = 100;  
index = 'string';//    因为 'String' 也是  
Object
```

注意：请不要滥用 dynamic，一般情况下都可以用 Object 代替 dynamic。

那什么情况下可以用 dynamic 呢？

当这个变量没法用 Dart 的类型来表示时，比如 Native 和 Flutter 交互，从 Native 传来的数据。所以你会看到 PlatformChannel 里有很多地方使用到了 dynamic。

变量：final 和 const

如果你不想更改变量的值，那么你可以用 final 和 const：

```
final content = 'Dart 语法';  
static const bool switchOn = false;
```

使用时有以下几点：

- 使用 final 和 const 的时候可以把 var 省略
- final 和 const 变量只能赋值一次，而且只能在声明的时候就赋值
- const 是隐式的 final
- 在使用 const 的时候，如果变量是类里的变量，必须加 static，是全局变量时不需要加，例如：

```
import 'package:flutter/material.dart';

const demoConst = 'demo'; // 这里不用加 static

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {

    static content = 'Dart 语法'; // 这里必须加
static
    ...
}
```

那么 final 和 const 有什么区别呢？

const 是编译时常量，在编译的时候就初始化了，但是 final 变量是当类创建的时候才初始化。

Dart 支持的数据类型

Dart 支持以下的数据类型：

	含义	使用
int	整数，范围为 -2^{63} 到 $2^{63} - 1$.	int x = 1; // 没有小数点就是 int
double	浮点数，64位（双精度） 浮点数	double y = 1.1; // 有小数点就是浮点数
num	num 是数字类型，既可以表示整数，也可以表示浮点数，具体看赋的值	num x = 1; // num x 是整数 num y = 1.1; // num y 是浮点数
String	字符串 Dart 字符串采用 UTF-16 编码	var s1 = 'string';

可以使用单引号或双引号 `var s2 = "string";`
来创建字符串

<code>bool</code>	布尔值	<code>var isTrue = true;</code>
	<code>List<E></code>	
<code>List</code>	E 表示 List 里的数据类型 用中括号来赋值	<code>List<int> list = [1, 2, 3];</code>
	<code>Set<E></code>	<code>Set<String> halogens =</code>
<code>Set</code>	E 表示 Set 里的数据类型 用大括号来赋值	<code>{'fluorine', 'chlorine', 'bromine', 'iodine', 'astatine'};</code>
	<code>Map<K, V></code>	<code>Map<String,String> gifts = {</code>
<code>Map</code>	K 是 Key 的数据类型,V是 Value 的数据类型	<code>// Key: Value 格式 'first': 'partridge', 'second': 'turtledoves', 'fifth': 'golden rings'};</code>
	表示采用 UTF-32 的字符串，用于显示 Unicode 因为Dart字符串是UTF-16，因此在Dart中表示 32位的Unicode值需要 Runes这个特殊语法。	<code>Runes input = new Runes('\u{1f600}'); print(new String.fromCharCode(input)); 打印出来的是笑脸emoji：</code>
<code>Runes</code>		

函数

在 Dart 中函数也是对象，函数的类型是 `Function`。

写一个函数

函数按照如下的格式来写：

```
返回类型 函数名(函数参数){  
  
}
```

这里写一个名字为 say 的函数：

```
bool say(String msg , String from, int clock){  
    print(msg+" from " + from + " at " +  
clock?.toString());  
    return true;  
}
```

函数的类型：Function

上面写了一个函数 say，写出如下的代码，判断函数 say 的类型是不是 Function：

```
print(say is Function);
```

输出的是 true。

而且请注意，这里在判断函数的类型的时候，直接用的是函数的名字，也就是说函数的名字就像变量名一样可以直接拿来使用，当直接用函数名来使用时，就是一个对象，其类型是 Function，要调用这个函数的话，就需要传入参数来使用，下面是使用函数的方法：

```
say('Hello Flutter', 'XiaoDe',10);
```

函数的参数：必选参数和可选参数

函数的参数分为两类：

1. 必选参数
2. 可选参数

必选参数是必填的，可选参数是选填的。

那如何知道参数是必选的还是可选的呢？

首先必选参数在前面，和普通的参数定义一样，后面跟可选参数，可选参数需要用 `{}` 或者 `[]` 包起来，用不同的括号，可选参数的意义与用法也不一样。

必选参数

必选参数就是使用平常的方法定义的函数参数，如下面的函数：

```
bool say(String msg , String from, int clock){  
    print(msg+" from " + from + " at " +  
clock?.toString());  
    return true;  
}
```

say 函数有三个参数：msg、from、clock，这三个参数都是必选参数。

必选参数必须在可选参数的前面。

可选参数

可选参数也分为两类：

1. **可选命名参数**：使用 `{}` 包起来的参数是可选命名参数
 2. **可选位置参数**：使用 `[]` 包起来的参数是可选位置参数
- 可选命名参数：`{}`

用 `{}` 包起来的参数是**可选命名参数**，前面讲数据类型的时候，使用 `{}` 来赋值的数据类型是 Map，所以可选的命名参数的类型也是 Map，因此调用函数时，可选参数的赋值必须是 `paramName: value` 这种 `key: value` 格式的，下面是可选命名参数的例子：

```
bool say(String msg , {String from, int
clock}){
    print(msg+" from " + from + " at " +
clock.toString());
    return true;
}
```

这里参数 from 就是可选命名参数。

要调用 say 函数，方法如下：

```
say('Hello Flutter');//    因为 from 和 clock
是可选参数，所以可以不填

say('Hello Flutter',from: 'XiaoMing');//对 用
命名参数格式 paramName: value 为 from 赋值
say('Hello Flutter',clock: 11);//
say('Hello Flutter',from: 'XiaoMing',clock:
11);//
```

同时还可以给命名参数加 @required ，意思是这个也是必填参数，例子如下：

```
bool say(String msg , {@required String from,
int clock}){
    print(msg+" from " + from + " at " +
clock.toString());
    return true;
}
```

这时候要调用 say 函数，方法如下：

```
say('Hello Flutter');// 错误调用方式，因为  
from 是必选参数，不填的话会报错
```

```
say('Hello Flutter',from: 'XiaoMing');// 正  
确调用方式
```

```
say('Hello Flutter',from: 'XiaoMing',clock:  
11);// 这个调用方式也是正确的
```

- 可选位置参数：[]

用 [] 包起来的参数是**可选位置参数**，前面讲数据类型的时候，使用 [] 来赋值的数据类型是 List，所以可选的命名参数的类型也是 List，所以赋值和参数是一一对应的，下面是可选位置参数的例子：

```
bool say(String msg , [String from , int  
clock]){  
    print(msg+" from " + from + " at " +  
clock.toString());  
    return true;  
}
```

要给可选位置参数赋值时，必选按照顺序来赋值：

```
say('Hello Flutter');// 因为 from 和 clock  
是可选参数，所以可以不填
```

```
say('Hello Flutter','XiaoMing',1);// 为可选  
位置参数赋值，只能一个参数一个参数对应的赋值，所以要全  
部赋值
```

```
say('Hello Flutter','XiaoMing')//  
say('Hello Flutter',1)// 因为 1 赋值给了  
from,但是 from 是String，所以会报错
```

可选参数的默认值：=

因为参数是可选的，那么参数的值很可能没有赋值，就是 null，用到的时候就有可能会触发 NPE，所以可以给可选参数赋默认值。

使用 = 为可选参数赋默认值

```
bool say(String msg , {String from = 'empty',
int clock = 0}){
    print(msg+" from " + from + " at " +
clock.toString());
    return true;
}
```

=> ： 箭头语法

=> 语法是 { return expr; } 的简写，因为 => 酷似箭头，也称箭头语法,也是 Lambda 表达式。

=> 语句后面只能跟一行代码，而且这一行代码只能一个表达式，而不能跟语句。表达式可以是函数、值。

例如，main() 函数这里：

```
void main() => runApp(MyApp());
```

就等价于：

```
void main(){
    return runApp(MyApp()); //runApp() 返回的是 void
}
```

类

Dart 中每个对象都是一个类的实例，所有类都继承自 Object。

自定一个类

如下，我定义一个类 Point：

```
class Point {  
  num x, y;  
  
  Point(num x, num y) {  
    // There's a better way to do this, stay  
tuned.  
    this.x = x;  
    this.y = y;  
  }  
}
```

默认构造函数的写法

默认的构造函数就是使用类名作为函数名的构造函数，例如上面写的类 Point 的构造函数：Point(num x,num y)

构造函数的语法糖

Dart 里还有构造函数的语法糖，可以将构造函数改造为：

```
Point(this.x, this.y);
```

这样 Point 类就可以简化为：

```
class Point {  
  num x,y;  
  Point(this.x, this.y);  
}
```

这个语法糖会简化构造函数的赋值操作。

Widget 构造函数参数采用的是可选命名参数

因为 Widget 构造函数有很多参数，为了使用起来清晰，Widget 构造函数的参数采用的是可选命名参数。

后面会经常用到。

创建实例：不需要使用 new

```
Point point = Point(0,0);
```

创建类实例的时候，都要写 new，其实很麻烦的，而且也没有必要，所以 Dart 在创建实例的时候不再需要使用 new 。

使用类的变量

使用点 . 来引用实例变量或方法：

```
print(p.x);
```

操作符

Dart 中定义了很多的操作符，分为以下几类：

1. 算术运算符
2. 比较操作符
3. 类型判断符
4. 赋值操作符
5. 逻辑运算符
6. 按位与移位运算符
7. 条件运算符

8. 级联操作符

9. 其他操作符

1. 算术运算操作符

下面是 Dart 支持的常见的算术运算操作符：

操作符	含义	例子
+	加	<code>var a = 2 + 3;</code>
-	减	<code>var a = 2 - 3;</code>
-exper	负数	<code>var a = -1;</code>
*	乘	<code>var a = 2 * 3;</code>
/	除，精确除法	<code>var a = 5 / 2; // a的结果为 2.5</code>
~/	整除	<code>var a = 5 ~/ 2; // a的结果为 2</code>
%	取余	<code>var a = 5 % 2; // a的结果为 1</code>
++var		<code>var a = 1;</code> <code>var b = ++a; // b 的结果为 2,a 的结果为 2</code>
var++		<code>var a = 1;</code> <code>var b = a++; // b 的结果为 1,a 的结果为 2</code>
--var		<code>var a = 1;</code> <code>var b = --a; // b 的结果为 0,a 的结果为 0</code>
var--		<code>var a = 1;</code> <code>var b = a--; // b 的结果为 1,a 的结果为 0</code>

2.相等和大小关系操作符

下面是 Dart 支持的判断是否相等和大小关系的操作符：

操作符	含义	例子
==	是否相等	<code>assert(2 == 2);</code>
!=	不等于	<code>assert(2 != 3);</code>
>	大于	<code>assert(3 > 2);</code>

< 小于 `assert(2 < 3);`
>= 大于等于 `assert(3 >= 3);`
<= 小于等于 `assert(3 <= 3);`

3.类型判断操作符

下面是 Dart 支持的检查运行时类型的操作符：

操作符	含义	例子
<code>as</code> 类型转换		<code>(emp as Person).firstName = 'Bob';</code>
<code>is</code> 判断是否是某个类型,如果是的话,就返回 <code>true</code>		<code>if (emp is Person) { // 如果 emp 是 Person 类型 emp.firstName = 'Bob'; }</code>
<code>is!</code> 判断是否不是某个类型, 如果不是的话, 就返回 <code>true</code>		<code>if (emp is! Person) { // 如果 emp 不是 Person 类型 }</code>

上面的例子中，如果 `emp` 是 `null` 的话，`as` 的例子就会抛异常，`is` 和 `isn't` 的例子会返回 `false`。

4.赋值操作符

赋值操作符是 `=`。

如果只想当被赋值的变量为空的时候才赋值，可以使用 `??=`，例如：

```
var a,b;  
a = 1; //使用 = 赋值  
b ??= 1; // 当 b 是空的话才赋值, 否则不会赋值
```

= 还可以和其他操作符结合起来使用, 例如:

```
+= -= *= /= ~/= %=  
>>= <<= ^= &= ` ` ??=
```

这些组合起来的操作符, 意思是先进行操作, 然后在赋值, 例如:

组合操作符

例子

+= a += b ; 就等效于 a = a + b;

5.逻辑运算操作符

下面是 Dart 支持的逻辑运算操作符:

操作符

含义

例子

!	反转表达式 (将 !expr false 改为 true, 反 之亦然)	!(2 == 3); // 结果 为 true
---	--	----------------------------

&&	逻辑与	(2 == 2) && (3 == 3); // 结果为 true
&	逻辑或	(2 == 2) & (3 == 3); // 结果为 true
	逻辑或	(2 == 2) (3 == 3); // 结果为 true

6.按位与移位运算符

下面是 Dart 支持的按位与移位运算符:

操作符	含义	例子
&	按位与 对于每一个比特位，只有两个操作数相应的比特位都是1时，结果才为1，否则为0。	final value = 0x22; final bitmask = 0x0f; var result = value & bitmask;//结果为 0x02
		final value = 按位或，对于每一个 0x22; 比特位，当两个操作 final bitmask;// 数相应的比特位至少 bitmask 结果为 有一个1时，结果为 = 0x0f; 0x2f 1，否则为0。 var result = value
^	按位异或，对于每一个比特位，当两个操作数相应的比特位有且只有一个1时，结果为1，否则为0。	final value = 0x22; final bitmask = 0x0f; var result = value ^ bitmask;//结果 为 0x2d
		final value = 0x22; final bitmask = 0x0f; var result = value & ~bitmask;//结果 为 0x20
~expr	按位非，反转操作数的比特位，即0变成1，1变成0。	final value =

<< 左移	0x22; final bitmask = 0x0f; var result = value << 4; // 结果为 0x220
>> 右移	final bitmask = 0x0f; var result = value >> 4; // 结果为 0x02

7. 条件运算符

Dart 有两个运算符，可以让您使用更简单的表达式来代替可能需要 if-else 语句的表达式：

1. condition ? expr1 : expr2

如果 condition 是 true，返回 expr1，否则返回 expr2。

当你需要根据一个 boolean 表达式来赋值时，可以使用 ? :，例如：

```
var visibility = isPublic ? 'public' :  
'private';
```

2. expr1 ?? expr2

如果 expr1 为 null，就返回 expr2 的值，否则返回 expr1 的值。

如果需要根据一个 boolean 表达式是否为 null 来作为条件，可以使用 ??，例如：

```
String playerName(String name) => name ??  
'Guest';
```

8.级联操作符

级联操作符是 ..,允许你对同一对象进行一系列的操作。除了函数调用，您还可以访问同一对象上的字段。这通常可以为您节省创建临时变量的步骤，并允许您编写更多流畅的代码。

例如：

```
querySelector('#confirm') // Get an object.  
  ..text = 'Confirm' // Use its members.  
  ..classes.add('important')  
  ..onClick.listen((e) =>  
window.alert('Confirmed!'));
```

querySelector() 返回一个 selector 对象，后面的 ..text、..classes、..onClick就是在 selector 对象上进行的。

9.其他操作符

还有其他操作符：

操作符	含义	例子
()	函数调用	代表函数调用
.	访问列	

[] 表	引用列表中指定索引处的值
访问成员变量	访问表达式里的成员变量，例如 <code>foo.bar</code> , 表示访问 <code>foo</code> 表达式里的 <code>bar</code> 成员变量
有条件的成员变量访问	很像 <code>.</code> ，但是左边的表达式可以为 <code>null</code> ，例如 <code>foo?.bar</code> ，如果 <code>foo</code> 为 <code>null</code> ，则不会抛异常，而是返回 <code>null</code> ，如果 <code>foo</code> 不为 <code>null</code> ，则可以返回 <code>bar</code>

在说一下 Dart 里很好用但容易搞混的几个操作符：`?.`、`??`、`??=`

1. `?.`

想要访问表达式的某个属性时，就可以使用这个，可以有效避免 NPE。

例如：

```
var yourName = user?.name;
```

就等效于：

```
var yourName;
if(user == null){
    yourName = null;
}else{
    yourName = user.name;
}
```

2. `??`

在赋值时，可以使用 `??`，若发现为空，可以为其赋默认值。
例如：

```
var yourName = name ?? "Bob";
```

就是在为 yourName 赋值时，若 name 有值，就使用 name 的值，若 name 为空，则使用默认值 Bob，等效于：

```
var yourName;  
if(name == null){  
    yourName = "Bob";  
}else{  
    yourName = name;  
}
```

3. ??=

expr1 ??= expr2 等效于 expr1 = expr1 ?? expr2
就是判断 expr1 是否为null，如果为null的话，就使用默认值 expr2。

例如：

```
user ??= User();
```

等效于：

```
if(user == null) {  
    user = User();  
}
```