

# Flutter 异步编程

本节开始讲 Flutter 的异步编程，首先我们先了解一下基本知识：

- 什么是异步编程？

异步是相对于同步来说的，同步代码必须得等代码运行完后才能执行下一步的代码，而异步代码不必等代码运行完，就可以直接运行下一步的，就叫异步代码。

- Flutter 是运行在 '单线程' 上的

这里的 '单线程' 是带有单引号的，并不是说 Flutter 只有一个线程，后面也会讲 Flutter 的线程模型，而是指 Flutter 的 Dart 代码是运行在单线程上的。

- 单线程？异步？

你可能会有疑问，Flutter 是运行在 '单线程' 上的，竟然还能异步？当然能异步，JS也是单线程，也可以异步啊。对于为何 Flutter 能够实现异步的原因，后面的章节会有讲到。

- Dart 代码运行在 Isolate 上

Dart 代码运行的环境叫 Isolate，Dart 代码默认跑在 root Isolate 上。当 Dart 代码正在运行时，同一个 Isolate 中的其他代码无法同时运行。Flutter 可以拥有多个 Isolates，但 Isolates 之间不能共享内存。

- 异步操作的结果：Future 对象

当写异步函数的时候，可以像普通函数一样返回结果，但是异步函数的返回的结果要用 Future 来包装。

Flutter 里的异步编程需要用到 Future 对象，代表的是异步操作的结果。

Future 的使用方法是：在异步编程里，要用 Future 来包装返回的结果，如：

- Future 表示返回结果是类型T的异步操作。
- Future 表示没有返回结果的异步操作。

除了返回期望的结果，也可能会抛异常出来。

当调用一个返回类型是 Future 对象的函数时，会发生如下两步：

1. 函数会被加入到一个待执行的队列里，同时立即返回一个 uncompleted Future object（未完成的Future对象）
2. 当队列里的函数执行完后，才会返回带有值的 complete Future object

- Future API

Future API 是用来写异步代码的。请和上面的 Future 对象区别开，Future 对象是一个对象，Future API 指的是 then 等操作符。Future 对象可以使用 Future API。

例如：

```
//函数返回一个Future<String>对象
Future<String> gatherNewsReports(){
    ...
}

//Future<String>对象 使用 Future API的then操作符
gatherNewsReports().then(...);
```

- `async` 和 `await`

`async` 和 `await` 是 Dart 语言用来支持异步编程的关键字，是 Dart 1.9 之后才加入的。这两个关键字使得 Dart 可以用同步代码的方式来写异步代码，而且不需要使用 Future API，极大的降低了异步代码的复杂度，而且方便阅读。给函数加上 `async` 关键字，使函数变为异步函数，`await` 关键字只能在异步函数里用。

有了 `async` 和 `await` 之后，基本上就没有必要使用 Future API 了。

- 如果想暂停代码运行直到 complete Future object 返回

有两种方法：

1. 在 `async` 函数里使用 `await` 关键字,例如：

```
void gather() async{
    var content = await
gatherNewsReports();
}

Future<String> gatherNewsReports(){
    ...
}
```

2. 对 Future 对象使用 `then` 方法，例如：

```
void gather() {  
    gatherNewsReports().then(...);  
}  
  
Future<String> gatherNewsReports(){  
    ...  
}
```

从这里看出，还是使用 `async` 和 `await` 比较方便。

- 在异步函数里使用 `try catch` 捕获异常

异步函数里有可能会抛出异常，所以使用 `try catch` 来捕获异常，代码如下：

```
void gather() async{  
    try{  
        var content = await  
gatherNewsReports();  
    } catch(e){  
  
    }  
}  
  
Future<String> gatherNewsReports(){  
    ...  
}
```

- 为了让 Flutter 代码可以并行运行，可以创建自定义的 `Isolate`  
创建 `Isolate` 会在接下来的几节讲。

将完这些基本知识，你可能还有疑问：

Flutter明明是单线程的，怎么还有异步操作？

想弄懂这个问题，请看后面章节讲的事件循环(event loop)及代码运行顺序

## async 和 await 的使用

### async 函数（异步函数）

异步函数是函数标有 async 修饰符的函数。async 只能用来修饰函数。

例如：

```
foo() async => 42;
```

当调用异步函数时，异步函数里的代码都是同步执行的，直到遇到 await，await 会立即返回一个 uncompleted Future 对象，但函数的主体也会暂停执行，直到 uncompleted Future 执行完成，返回一个带结果的 complete Future 对象，函数的主体才会接着运行。如上面的代码，调用 foo() 后会返回一个 Future，这个 Future 的结果的值是 42。

如果不用 async 关键字的话，可以这么写：

```
foo() => new Future(() => 42);
```

可以看到不用 async 的话，写起来会复杂很多，而且不能用 await，来看一下 await 的使用方法。

### await 表达式

await 表达式允许你用写同步代码的方式来写异步代码。

await 会阻塞当前的操作，直到结果返回才会执行下一步。而且请注意，await 只能在 async 里面使用。

为什么要用 `await` 阻塞当前操作，这里有一个例子：假设要从服务器获取一个信息，并打印出来：

- 因为从服务器获取一个信息是一个耗时的操作，所以这个操作必须是异步的，所以使用 `async` 写成 异步函数
- 要把获取的信息打印出来，肯定要等到数据返回，所以这里需要 `await` 阻塞当前操作，直到数据回来

代码如下：

```
void getFromServer() async{
    var content =await fetch();
    print(content);
}

String fetch(){
    return 'infomation';
}
```

如果 `await` 后面的表达式返回值不是 `Future` 对象，那么会自动被 `Future` 对象封装。

在代码运行到 `await` 之后，`fetch()` 被调用，产生 `Future` 对象，同时代码运行会暂停，等这个 `Future` 对象运行完成，当 `Future` 对象运行完成后，会返回结果，然后接着运行下一句 `print(content)`。

`await` 的作用就是等 `Future` 运行完成。

为什么要这样子呢？

想象一下，如果 `await` 不会暂停运行，直接运行下一步，我本来要打印获取到的内容，结果内容的值还没返回回来，肯定会运行失败，所以 `await` 暂停运行，等结果回来，很有必要。

同时，要注意 `await` 的用法

`await` 只能在 `async` 函数里使用，如果用在普通函数里，会报语法错误。