秒懂Flutter之如何进行异步编程（future，async, await）

09/09/2021 · 1002点热度 · 0人点赞 · 0条评论

概述

说实话，在使用了几个月的Flutter后，我爱上了它，至少爱上了dart处理网络请求的方式。今天我想叨逼叨一下Flutter中异步编程这块

异步编程

异步编程是个古老的话题，感兴趣的同学请移步：维基百科。经过多年的发展，目前主要有如下几种解决方案：

- Threading 多线程
- Callbacks 回调
- Futures，Promises
- Reactive Extensions Rx系列， 例如rxjava
- Coroutines 协程

若对协程有兴趣，请查看我的另一篇文章：秒懂Kotlin之小白都看的懂的协程教程(part1)

dart 采用了第三种方式：Futures。由于Dart的单线程模型，所以使用比Java简单很多，我敢保证Flutter绝对比原生开发简单，没试过的同学建议大胆尝试一下，毕竟IT这个行当就是要不断的折腾。

Future

Dart 使用Future来处理异步编程。顾名思义，它是一个未来才能获得结果的东东，所以如果某个操作是耗时的，你可以将这个耗时操作包装成一个Future，等未来某个时间点操作完成后，你就可以从这个Future里面取出结果了。

Dart 的Future前后支持两套使用方法。在1.9之前使用Future API，1.9之后加入了 Asynchrony support （async, await） 。现在官方强烈推荐使用async与await这种方法，本文我们着重聊第二种方法，对于第一种方法浅尝辄止。

第一种： Future API 和 callbacks

这种方法的缺点有：

- 使用回调，所以仍然需要以异步的方式写代码
- 需要学习各种API的使用方法，如 `then` 、 `catchError` 、 `whenComplete` 等。

虽然各种API方法的名称已经做到顾名思义了，但要熟练掌握仍然有一定的学习成本。

我们简单看一个使用实例：

```
Future.value('ShuSheng007')
    .then((value) => print(value))
    .catchError((e,stackTrace) =>print('Exception:$e StackTrace:$stackTrace'))
    .whenComplete(() => print('无论如何都执行'));
```

执行结果为：

```
I/flutter (16437): ShuSheng007
I/flutter (16437): 无论如何都执行
```

一般的使用，上面的方法就够了。 如果Future 内部逻辑执行正常，就会在 `then` 方法中获得结果，如果期间Future或者 `then` 中发生了异常就会进入 `catchError` 内，最后无论成功失败，都会执行 `whenComplete` 方法。

值得注意的是，上面3个方法可以一不同的顺序，不同的个数组合，所以也很灵活。

第二种：async与await

这个就牛逼了，用同步的方式写异步，还不用学习新的API，关键是你写的代码很容易理解，那附带的结果就是更容易维护。与kotlin协程有点像，但是比协程简单。

例如我有一个从网络获取姓名并打印的需求

```
Future<void> printUserName() async {
  String name = await getUserName();
  print('my name is :$name');
}

//模拟从网上获取数据
Future<String> getUserName() =>
    Future.delayed(Duration(seconds: 1), () => 'ShuSheng007');
```

当点击UI上的一个按钮时调用 `printUserName` 方法。

```
 FlatButton(
   onPressed: () {
     printUserName();
     print('我写在打印名字下面却先执行了');
   },
   child: Text('Run'),
 )
```

输出结果：

```
I/flutter ( 4787): 开始打印名字
I/flutter ( 4787): 我写在打印名字下面却先执行了
I/flutter ( 4787): my name is :ShuSheng007
```

我们可以看到，在Flutter中完成耗时任务非常简单，如果在Android中就需要使用RxJava等方式进行线程的切换。

如何使用

1. 在方法的 `{}` 前面加上 `async` 关键字
2. 使用了 `async` 关键字的方法返回类型是一个 `Future<T>` ， T为此方法具体的返回值类型，如果没有返回则为 `void` ， 即 `Future<void>` 。
3. 在 `{}` 就可以使用 `await` 关键字来从 `Future` 中获取到值了

当 `printUserName` 方法被调用时，当没有遇到 `await` 时正常执行，当遇到 `await` 时就它就被暂停了，程序会接着执行此方法下面的代码，例如此处的

```
print('我写在打印名字下面却先执行了');
```

当await那个Future执行完毕， `printUserName` 方法就会被唤起，接着执行下面的代码。

异常处理

异步支持使得异步编程的异常处理也变的非常简单，只要加上 `await` 关键字，就与处理同步方法完全一样了

```
try {
  print('开始打印名字');
  String name = await getUserName();
  print('my name is :$name');
} catch (e) {
  print(e);
} finally {
}
```

只要在try 块里发生的异常都可以被捕获到，如果 `getUserName()` 发生了未捕获的异常也可以在catch中捕获到，是不是与同步无异呢？但是一定要注意，想要捕获 `getUserName()` 里的异常，必须使用 `await` 关键字，即使你不关心其返回结果，但是要捕获异常，也必须要有 `await` 。

关于dart的异常处理传送门：秒懂Flutter之Dart中如何处理异常（Exception）

并发执行任务

有时我们需要同时请求多个互不相关的接口，然后等所有接口的值都返回后再展示页面，在Android原生中，我们可以使用RxJava的 `zip` 操作符，那在dart中怎么办呢?

可以使用 `Future.wait` 方法，如下所示：

```
Future<void> printUserName() async {
  List<Object> results= await Future.wait([getUserName(),getUserAge()]);
  print('my name is :${results[0]},I am ${results[1]} years old');
}

Future<String> getUserName() =>
   Future.delayed(Duration(seconds: 1), () => 'ShuSheng007');
Future<int> getUserAge() =>
   Future.delayed(Duration(seconds: 2), () => 18);
```

输出结果：

```
I/flutter ( 4787): my name is :ShuSheng007,I am 18 years old
```

此方法在大约2秒后返回了结果。

Stream

敬请期待...

总结

写了一段时间Flutter，觉得Java真的是太严格了，虽然用的不爽，不过我还是很支持Java语法的严谨性，毕竟只有这样才能造就庞大的工程项目，小伙伴你们觉得Java和Dart哪个好用呢?

标签： Async Dart flutter Future