[深入浅出RxJava三--响应式的好处](http://blog.csdn.net/lzyzsd/article/details/44891933)

在第一篇中，我介绍了RxJava的基础知识。第二篇中，我向你展示了操作符的强大。但是你可能仍然没被说服。这篇里面，我讲向你展示RxJava的其他的一些好处，相信这篇足够让你去使用Rxjava.

## 错误处理

到目前为止，我们都没怎么介绍onComplete()和onError()函数。这两个函数用来通知订阅者，被观察的对象将停止发送数据以及为什么停止（成功的完成或者出错了）。

下面的代码展示了怎么使用这两个函数：

Observable.just("Hello, world!")

.map(s -> potentialException(s))

.map(s -> anotherPotentialException(s))

.subscribe(new Subscriber<String>() {

@Override

public void onNext(String s) { System.out.println(s); }

@Override

public void onCompleted() { System.out.println("Completed!"); }

@Override

public void onError(Throwable e) { System.out.println("Ouch!"); }

});

代码中的potentialException() 和 anotherPotentialException()有可能会抛出异常。每一个Observerable对象在终结的时候都会调用onCompleted()或者onError()方法，所以Demo中会打印”Completed!”或者”Ouch!”。

这种模式有以下几个优点：

1.只要有异常发生onError()一定会被调用

这极大的简化了错误处理。只需要在一个地方处理错误即可以。

2.操作符不需要处理异常

将异常处理交给订阅者来做，Observerable的操作符调用链中一旦有一个抛出了异常，就会直接执行onError()方法。

3.你能够知道什么时候订阅者已经接收了全部的数据。

知道什么时候任务结束能够帮助简化代码的流程。（虽然有可能Observable对象永远不会结束）

我觉得这种错误处理方式比传统的错误处理更简单。传统的错误处理中，通常是在每个回调中处理错误。这不仅导致了重复的代码，并且意味着每个回调都必须知道如何处理错误，你的回调代码将和调用者紧耦合在一起。

使用RxJava，Observable对象根本不需要知道如何处理错误！操作符也不需要处理错误状态-一旦发生错误，就会跳过当前和后续的操作符。所有的错误处理都交给订阅者来做。

## 调度器

假设你编写的**[Android](http://lib.csdn.net/base/android" \o "Android知识库" \t "http://blog.csdn.net/lzyzsd/article/details/_blank)** app需要从网络请求数据（感觉这是必备的了，还有单机么？）。网络请求需要花费较长的时间，因此你打算在另外一个线程中加载数据。那么问题来了！

编写多线程的Android应用程序是很难的，因为你必须确保代码在正确的线程中运行，否则的话可能会导致app崩溃。最常见的就是在非主线程更新UI。

使用RxJava，你可以使用subscribeOn()指定观察者代码运行的线程，使用observerOn()指定订阅者运行的线程：

myObservableServices.retrieveImage(url)

.subscribeOn(Schedulers.io())

.observeOn(AndroidSchedulers.mainThread())

.subscribe(bitmap -> myImageView.setImageBitmap(bitmap));

是不是很简单？任何在我的Subscriber前面执行的代码都是在I/O线程中运行。最后，操作view的代码在主线程中运行.

最棒的是我可以把subscribeOn()和observerOn()添加到任何Observable对象上。这两个也是操作符！。我不需要关心Observable对象以及它上面有哪些操作符。仅仅运用这两个操作符就可以实现在不同的线程中调度。

如果使用AsyncTask或者其他类似的，我将不得不仔细设计我的代码，找出需要并发执行的部分。使用RxJava，我可以保持代码不变，仅仅在需要并发的时候调用这两个操作符就可以。

## 订阅（Subscriptions）

当调用Observable.subscribe()，会返回一个Subscription对象。这个对象代表了被观察者和订阅者之间的联系。

ubscription subscription = Observable.just("Hello, World!")

.subscribe(s -> System.out.println(s));

你可以在后面使用这个Subscription对象来操作被观察者和订阅者之间的联系.

subscription.unsubscribe();

System.out.println("Unsubscribed=" + subscription.isUnsubscribed());// Outputs "Unsubscribed=true"

RxJava的另外一个好处就是它处理unsubscribing的时候，会停止整个调用链。如果你使用了一串很复杂的操作符，调用unsubscribe将会在他当前执行的地方终止。不需要做任何额外的工作！

## 总结

记住这个系列仅仅是对RxJava的一个入门介绍。RxJava中有更多的我没介绍的功能等你探索（比如backpressure）。当然我也不是所有的代码都使用响应式的方式–仅仅当代码复杂到我想将它分解成简单的逻辑的时候，我才使用响应式代码。

最初，我的计划是这篇文章作为这个系列的总结，但是我收到许多请求我介绍在Android中使用RxJava，所以你可以继续阅读第四篇了。我希望这个介绍能让你开始使用RxJava。如果你想学到更多，我建议你阅读RxJava的官方wiki。