[深入浅出RxJava(二：操作符)](http://blog.csdn.net/lzyzsd/article/details/44094895)

在[第一篇blog](http://blog.csdn.net/lzyzsd/article/details/41833541" \t "http://blog.csdn.net/lzyzsd/article/details/_blank)中，我介绍了RxJava的一些基础知识，同时也介绍了map()操作符。当然如果你并没有意愿去使用RxJava我一点都不诧异，毕竟才接触了这么点。看完这篇blog，我相信你肯定想立即在你的项目中使用RxJava了，这篇blog将介绍许多RxJava中的操作符，RxJava的强大性就来自于它所定义的操作符。  
  
首先先看一个例子：

## 准备工作

假设我有这样一个方法：  
这个方法根据输入的字符串返回一个网站的url列表（啊哈，搜索引擎）

Observable<List<String>> query(String text);

现在我希望构建一个健壮系统，它可以查询字符串并且显示结果。根据上一篇blog的内容，我们可能会写出下面的代码：

query("Hello, world!")

    .subscribe(urls -> {

**for** (String url : urls) {

            System.out.println(url);

        }

    });

这种代码当然是不能容忍的，因为上面的代码使我们丧失了变化数据流的能力。一旦我们想要更改每一个URL，只能在Subscriber中来做。我们竟然没有使用如此酷的map()操作符！！！  
  
当然，我可以使用map操作符，map的输入是urls列表，处理的时候还是要for each遍历，一样很蛋疼。  
  
万幸，还有Observable.from()方法，它接收一个集合作为输入，然后每次输出一个元素给subscriber：

Observable.from("url1", "url2", "url3")

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

我们来把这个方法使用到刚才的场景：

query("Hello, world!")

    .subscribe(urls -> {

        Observable.from(urls)

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

    });

虽然去掉了for each循环，但是代码依然看起来很乱。多个嵌套的subscription不仅看起来很丑，难以修改，更严重的是它会破坏某些我们现在还没有讲到的RxJava的特性。

## 改进

救星来了,他就是flatMap()。  
Observable.flatMap()接收一个Observable的输出作为输入，同时输出另外一个Observable。直接看代码：

query("Hello, world!")

    .flatMap(**new** Func1<List<String>, Observable<String>>() {

        @Override

**public** Observable<String> call(List<String> urls) {

**return** Observable.from(urls);

        }

    })

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

这里我贴出了整个的函数代码，以方便你了解发生了什么，使用lambda可以大大简化代码长度：

query("Hello, world!")

    .flatMap(urls -> Observable.from(urls))

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

flatMap()是不是看起来很奇怪？为什么它要返回另外一个Observable呢？理解flatMap的关键点在于，flatMap输出的新的Observable正是我们在Subscriber想要接收的。现在Subscriber不再收到List<String>，而是收到一些列单个的字符串，就像Observable.from()的输出一样。  
  
这部分也是我当初学RxJava的时候最难理解的部分，一旦我突然领悟了，RxJava的很多疑问也就一并解决了。

## 还可以更好

flatMap()实在不能更赞了，它可以返回任何它想返回的Observable对象。  
比如下面的方法：

// 返回网站的标题，如果404了就返回null

Observable<String> getTitle(String URL);

接着前面的例子，现在我不想打印URL了，而是要打印收到的每个网站的标题。问题来了，我的方法每次只能传入一个URL，并且返回值不是一个String，而是一个输出String的Observabl对象。使用flatMap()可以简单的解决这个问题。

query("Hello, world!")

    .flatMap(urls -> Observable.from(urls))

    .flatMap(**new** Func1<String, Observable<String>>() {

        @Override

**public** Observable<String> call(String url) {

**return** getTitle(url);

        }

    })

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

使用lambda:

query("Hello, world!")

    .flatMap(urls -> Observable.from(urls))

    .flatMap(url -> getTitle(url))

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

是不是感觉很不可思议？我竟然能将多个独立的返回Observable对象的方法组合在一起！帅呆了！  
不止这些，我还将两个API的调用组合到一个链式调用中了。我们可以将任意多个API调用链接起来。大家应该都应该知道同步所有的API调用，然后将所有API调用的回调结果组合成需要展示的数据是一件多么蛋疼的事情。这里我们成功的避免了callback hell（多层嵌套的回调，导致代码难以阅读维护）。现在所有的逻辑都包装成了这种简单的响应式调用。

## 丰富的操作符

目前为止，我们已经接触了两个操作符，RxJava中还有更多的操作符，那么我们如何使用其他的操作符来改进我们的代码呢？  
getTitle()返回null如果url不存在。我们不想输出"null"，那么我们可以从返回的title列表中过滤掉null值！

query("Hello, world!")

    .flatMap(urls -> Observable.from(urls))

    .flatMap(url -> getTitle(url))

    .filter(title -> title != **null**)

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

filter()输出和输入相同的元素，并且会过滤掉那些不满足检查条件的。  
  
如果我们只想要最多5个结果：

query("Hello, world!")

    .flatMap(urls -> Observable.from(urls))

    .flatMap(url -> getTitle(url))

    .filter(title -> title != **null**)

    .take(5)

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

take()输出最多指定数量的结果。  
  
如果我们想在打印之前，把每个标题保存到磁盘：

query("Hello, world!")

    .flatMap(urls -> Observable.from(urls))

    .flatMap(url -> getTitle(url))

    .filter(title -> title != **null**)

    .take(5)

    .doOnNext(title -> saveTitle(title))

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

doOnNext()允许我们在每次输出一个元素之前做一些额外的事情，比如这里的保存标题。  
  
看到这里操作数据流是多么简单了么。你可以添加任意多的操作，并且不会搞乱你的代码。  
  
RxJava包含了大量的操作符。操作符的数量是有点吓人，但是很值得你去挨个看一下，这样你可以知道有哪些操作符可以使用。弄懂这些操作符可能会花一些时间，但是一旦弄懂了，你就完全掌握了RxJava的威力。  
  
你甚至可以编写自定义的操作符！这篇blog不打算将自定义操作符，如果你想的话，清自行Google吧。

## 感觉如何？

好吧，你是一个怀疑主义者，并且还很难被说服，那为什么你要关心这些操作符呢？  
  
因为操作符可以让你对数据流做任何操作。  
  
将一系列的操作符链接起来就可以完成复杂的逻辑。代码被分解成一系列可以组合的片段。这就是响应式函数编程的魅力。用的越多，就会越多的改变你的编程思维。

另外，RxJava也使我们处理数据的方式变得更简单。在最后一个例子里，我们调用了两个API，对API返回的数据进行了处理，然后保存到磁盘。但是我们的Subscriber并不知道这些，它只是认为自己在接收一个Observable<String>对象。良好的封装性也带来了编码的便利！

在第三部分中，我将介绍RxJava的另外一些很酷的特性，比如错误处理和并发，这些特性并不会直接用来处理数据。