深入浅出RxJava(二：操作符)

标签： 响应式函数编程 RxJava

2015-03-06 08:04 18657人阅读 评论(26) 收藏 举报

分类： RxJava（6）

目录(?) [+]

在第一篇blog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）

```
[java]
01. 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：

```
[java]
01. query("Hello, world!")
02.     .subscribe(urls -> {
03.         for (String url : urls) {
04.             System.out.println(url);
05.         }
06.     });
```

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：

```
[java]
01. Observable.from("url1", "url2", "url3")
02.     .subscribe(url -> System.out.println(url));
```

我们来把这个方法使用到刚才的场景：

```
[java]
01. query("Hello, world!")
02.     .subscribe(urls -> {
03.         Observable.from(urls)
04.             .subscribe(url -> System.out.println(url));
05.     });
```

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

hi大头鬼hi

访问： 285482次
积分： 1589
等级： BLOC 4
排名： 第15832名

原创： 14篇 转载： 0篇
译文： 12篇 评论： 174条

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：

[java]

```
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：

[java]

```
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：

[java]

```
// 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。

[java]

```
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:

[java]

```
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！

```java
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：

```java
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：

```java
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。

原文链接

顶 14 踩 4

上一篇 android-gradle-深入浅出-五:build type
下一篇 深入浅出RxJava三--响应式的好处

我的同类文章

RxJava（6）

- RxJava使用场景小结 11-30 阅读 6625
- RxJava基本流程和lift源码分析 2015-11-30 阅读 3841
- 如何升级到RxAndroid 10-19 阅读 3096
- 深入浅出RxJava四-在Andro... 2015-04-13 阅读 22315
- 深入浅出RxJava三--响应式... 2015-04-05 阅读 14525
- 深入浅出RxJava（一：基础... 2014-12-09 阅读 64614

主题推荐 java url 搜索引擎 工作 class

猜你在找

Winform项目实战在线考试系统开发
大数据之编程语言：Scala
C#.NET_面向对象编程技术
Python自动化开发基础 装饰器-异常处理-面向对象编程
大数据编程语言：Java基础
java asp 分析各种搜索引擎的关键字自动识别url 中关
RxJava 操作符 take
RxJava 操作符 combineLatest
RxJava 操作符 interval
RxJava入门学习------①操作符

查看评论

18楼 bfbx5173 2016-02-03 14:27发表

装逼有风险，使用需谨慎

Re: 鱼塘鱼汤 昨天 19:47发表

回复bfbx5173：个人觉得能不用就最好别用那玩意儿。如果要做一些热修复类似的事情，就悲剧了。

17楼 bfbx5173 2016-02-03 14:26发表

如果你习惯使用 Retrolambda ，你也可以直接把代码写成上面这种简洁的形式。而如果你看到这里还不知道什么是 Retrolambda ，我不建议你现在就去学习它。原因有两点：1. Lambda 是把双刃剑，它让你的代码简洁的同时，降低了代码的可读性，因此同时学习 RxJava 和 Retrolambda 可能会让你忽略 RxJava 的一些技术细节；2. Retrolambda 是 Java 6/7 对 Lambda 表达式的非官方兼容方案，它的向后兼容性和稳定性是无法保障的，因此对于企业项目，使用 Retrolambda 是有风险的。所以，与很多 RxJava 的推广者不同，我并不推荐在学习 RxJava 的同时一起学习 Retrolambda。事实上，我个人虽然很欣赏 Retrolambda，但我从来不用它。 http://gank.io/post/560e15be2dca930e00da1083

16楼 XBlithe 2016-02-02 17:27发表

对于lambda这个不是很了解,所以看起来非常吃力.

15楼 lyclovezmy 2016-01-30 20:30发表

怎么老有人说在这里说不理解lambda。。。
不理解就去看啊

14楼 -琥珀川- 2016-01-26 11:31发表

2016-1-26学习

13楼 zcmain 2016-01-11 16:40发表

没接触过 lambda 看起来比较吃力

12楼 Franous 2015-12-03 16:57发表

lambda虽然简洁，但不适用在初学者教程中，比如 .subscribe(url -> System.out.println(url)); 这一句，我就不知道url是String类型还是Object类型，使用lambda必须对这个库非常熟悉才行，但是这里又是基础教程，相信熟悉这个库的使用者一般不太会到这里溜达吧

11楼 有妻徒刑岁月长 2015-12-03 11:55发表

这个例子有下载地址么

10楼 LittleCococ 2015-12-02 19:38发表

关于操作符，我觉得可以类比javascript的Ramda
比如 flatmap就相当于Ramda的flatten，Ramda的flatten介绍就相当直观。
http://ramdajs.com/0.18.0/docs/#flatten

9楼 xueerfei 2015-11-03 21:02发表

弱弱的问下
Observable<List<String>> query(String text);
这个声明是有问题的，请问咋样声明这个query呢

Re: 我是东方谁是不败 2015-11-19 12:48发表

回复xueerfei：木有问题。public Observable<List<String>> query(String text) { return Observable.create(subscribe -> { subscribe.onNext("xxx"); subscribe.onCompleted(); });}

Re: 4805凯盛 3天前 10:54发表

回复我是东方谁是不败：onNext里面 不应该是List集合吗 为什么是“XXX”

Re: ZYJWR 2016-01-27 16:33发表

回复我是东方谁是不败：拷贝进去会报错啊，请问怎么回事

Re: 我是东方谁是不败 2016-01-28 13:44发表

回复ZYJWR：首先你要引用了rxjava和rxandroid包，还有我用了lambda，你改一下

8楼 男子汉大豆腐 2015-10-28 15:05发表

lambda混合到语法中，增加了理解难度

Re: Pisces0Su 2016-02-17 10:11发表

就是说嘛，深入浅出RXJava你搞的全是lambda表达式，这让我情何以堪。。。

7楼 androidstackoverflow 2015-10-06 14:46发表

这篇blog不打算将自定义操作符，如果你想的话，清自行Google吧。两个错别字。“将”—>“讲”，“清”—>“请”

6楼 androidstackoverflow 2015-10-06 14:42发表

1.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。应该修改为收到一些单个的字符串，去掉“列”
2.这篇blog不打算将自定义操作符，需要修改“将”—>“讲”

5楼 挤不上公交车的路人甲 2015-09-20 17:59发表

请问 lz，我看有的demo里面用的是 Observer，这和Subscriber有什么区别吗

Re: wuxiaoming1992 6天前 16:45发表

Subscriber实现了Observer，多出了几个方法,onstart之类的

4楼 在下雨了 2015-09-09 14:12发表

好屌的样子，看是看懂了，但是不知道实际应用场景如何使用，还希望提供更多的使用场景的demo。

3楼 米莱尼 2015-09-06 17:16发表

原来是翻译的文章啊

2楼 Justlove_DK 2015-08-17 17:04发表

Rxjava和RxAndroid有什么区别啊

1楼 晴天小子 2015-03-06 15:11发表

RxAndroid的jar包两M多是不是有点大呢

Re: hi大头鬼hi 2015-03-06 23:10发表

回复晴天小子：http://mvnrepository.com/artifact/io.reactivex/rxjava/1.0.7只有700k啊

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap