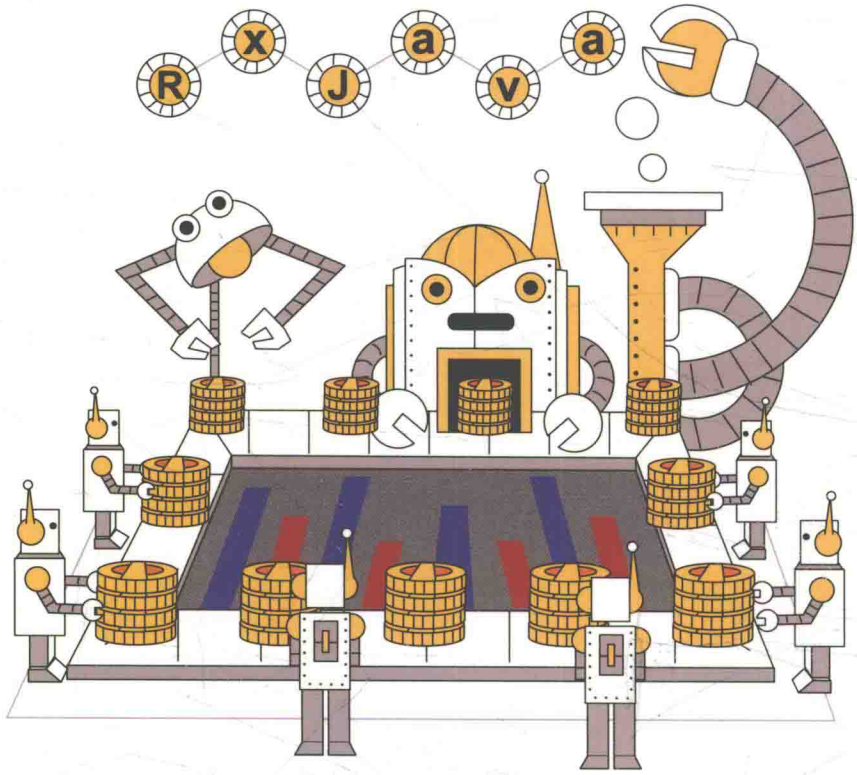
Broadview®
www.broadview.com.cn



RxJava 响应式编程

李衍顺 著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

RxJava
响应式编程

李衍顺 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

响应式编程是一种基于异步数据流概念的编程模式。在开发手机 App、Web App 时，要想保证对用户请求的实时响应，给用户带来流畅的体验，响应式编程是一个不错的选择，RxJava 则是这种编程模式的 Java 实现。本书主要介绍如何使用 RxJava 进行响应式编程。全书一共 6 章，从响应式编程与 RxJava 的概念，到 RxJava 的操作符和源码，以及各种 Scheduler 的特点和适用场景，均做了较细致的讲解。本书还用一章的篇幅给出了几个 RxJava 的实用案例，帮助读者理解概念，上手操作。

本书适合 RxJava 的初学者，以及对 RxJava 有初步了解并想要进一步深入学习的读者阅读。

图书在版编目（CIP）数据

RxJava 响应式编程 / 李衍顺著. —北京：电子工业出版社，2018.4
ISBN 978-7-121-33640-9

Ⅰ. ①R… Ⅱ. ①李… Ⅲ. ①移动电话机－应用程序－程序设计 ②JAVA 语言－程序设计

Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字（2018）第 022880 号

策划编辑：许　艳
责任编辑：张春雨
印　　刷：三河市君旺印务有限公司
装　　订：三河市君旺印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编　100036
开　　本：787×980　1/16　印张：14.25　字数：299 千字　印数：2500 册
版　　次：2018 年 4 月第 1 版
印　　次：2018 年 4 月第 1 次印刷
定　　价：49.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819　faq@phei.com.cn。

前 言

毫无疑问，RxJava是一个非常优秀的开源库，清晰的流式操作和便捷的线程切换为Java和Android开发者提供了有力的帮助。网上有大量介绍RxJava的文章，开发者可以很容易地查找到相关的学习资料。但是由于RxJava入门比较困难，而且缺乏一本系统地介绍RxJava的中文书籍，所以给很多初学者带来了困扰，不少人浅尝辄止，放弃了深入学习和使用RxJava的机会，这十分可惜。本书作为一本入门书，比较适合RxJava的初学者以及对RxJava有初步了解并想要进一步学习RxJava的读者。

内容结构

本书第1章从响应式编程入手，介绍了RxJava及RxJava的组成部分，帮读者初步了解RxJava。

第2章配合官方的示意图分类介绍了RxJava的大部分操作符。这一章的篇幅比较多，读者在阅读的时候可能无法全部记住，可以在需要时随时翻阅查询。

第3章就各种Scheduler的特点和适合的使用场景做了介绍，帮助读者根据实际需要选择最合适的Scheduler。

只知道轮子怎么跑还不够，还有必要知道轮子是如何造的，第4章结合源码研究了RxJava的实现原理。了解原理一方面可以让我们避免用错操作符或者Scheduler，另一方面

如果碰到 RxJava 中的 bug，也有助于我们定位 bug。发现 bug 后可以到 GitHub 上发起一个 issue，而且最好能够提一个附带的 pull request 来修复这个 bug。

第 5 章给出了一些实例和基于 RxJava 的开源库的使用示例，以帮助读者更好地将 RxJava 应用于实际开发中。

第 6 章介绍了 RxJava 2 相对于 RxJava 1 的改进之处，如果读者已经掌握了 RxJava 1，那么 RxJava 2 也可以很容易地上手。

给初学者的建议

RxJava 这种响应式编程方式跟大多数人习惯的命令式编程方式有较大的区别，所以初学者首先需要完成编程思想上的转变，理解 RxJava 的思想。如可以将 Observable 看作工厂的原材料生产机器，发送出来的数据即为原材料，整个链式操作可以视为原材料经过一条流水线，每个操作符为流水线上的一个车间，每个车间都会对原材料做一定的加工，最终的 Subscriber 可以视为最终消费者，会接收加工后的成品。

其次就是了解 RxJava 的操作符都有哪些，都有什么样的作用。你不需要一开始就将每个操作符都记住，但是可以大体上记住都有什么功能的操作符，这样在需要时就能够想起哪个操作符能够满足当下的需求。关于操作符的详细使用方式可以参阅第 2 章。

接下来就是实践环节了。初期可以尝试应用 RxJava 写一些小程序，并参阅网上的一些开源代码，看看别人都是怎么应用 RxJava 的。初步掌握之后就可以逐渐将 RxJava 引入到项目中，来解决一些工作过程中遇到的实际问题。只看不做永远都是眼高手低，只有将 RxJava 真正地应用到实际开发工作中，不断犯错、不断改进才能真正达到融会贯通的地步，才能真正地掌握 RxJava 的使用技巧。

最后，如果想要进一步学习 RxJava，可以阅读源代码，可以深入地跟踪一个操作符的实现过程来了解其原理。如果有可能，可以参与到 RxJava 的 bug 修复或者新功能开发中，在 GitHub 上给 RxJava 提 pull request，上面有很多大神会给你提各种修改意见，理解他们的思路绝对会让你受益匪浅。

读者服务

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

下载资源：本书如提供示例代码及资源文件，均可在 下载资源 处下载。

提交勘误：您对书中内容的修改意见可在 提交勘误 处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。

交流互动：在页面下方 读者评论 处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/33640

目 录

第 1 章 走进 RxJava 的世界 1

1.1 响应式编程 1
1.2 什么是 RxJava 4
1.3 Observable 和 Subscriber 5
1.3.1 Single：单个数据的生产者 8
1.3.2 Completable：单个事件的生产者 9
1.4 在 Android 工程中引入 RxJava 11

第 2 章 RxJava 中的操作符 12

2.1 创建 Observable 的操作符 12
2.1.1 range 13
2.1.2 defer 和 just 13
2.1.3 from 16
2.1.4 interval 17
2.1.5 repeat 和 timer 19
2.2 转化 Observable 的操作符 21
2.2.1 buffer 21

2.2.2 flatMap 23
2.2.3 groupBy 25
2.2.4 map 28
2.2.5 cast 29
2.2.6 scan 31
2.2.7 window 32
2.3 过滤操作符 35
2.3.1 debounce 35
2.3.2 distinct 39
2.3.3 elementAt 40
2.3.4 filter 41
2.3.5 first 和 last 43
2.3.6 skip 和 take，skipLast 和 takeLast 45
2.3.7 sample 和 throttleFirst 46
2.4 组合操作符 48
2.4.1 combineLatest 48
2.4.2 join 和 groupJoin 51
2.4.3 merge 和 mergeDelayError 55
2.4.4 startWith 58
2.4.5 switch 59
2.4.6 zip 和 zipWith 61
2.5 错误处理操作符 64
2.5.1 onErrorReturn 64
2.5.2 onErrorResumeNext 66
2.5.3 onExceptionResumeNext 67
2.5.4 retry 70
2.6 辅助操作符 72
2.6.1 delay 72
2.6.2 do 74
2.6.3 materialize 和 dematerialize 78

2.6.4 subscribeOn 和 observeOn 80
2.6.5 timeInterval 和 timeStamp 82
2.6.6 timeout 84
2.6.7 using 87
2.7 条件操作 90
2.7.1 all 90
2.7.2 amb 92
2.7.3 contains 93
2.7.4 isEmpty 94
2.7.5 defaultIfEmpty 95
2.7.6 sequenceEqual 97
2.7.7 skipUntil 和 skipWhile 98
2.7.8 takeUntil 和 takeWhile 100
2.8 聚合操作符 102
2.8.1 concat 102
2.8.2 count 104
2.8.3 reduce 105
2.8.4 collect 106
2.9 与 Connectable Observable 相关的操作符 107
2.9.1 publish 和 connect 108
2.9.2 refCount 110
2.9.3 replay 111
2.10 自定义操作符 114
2.10.1 lift 115
2.10.2 compose 117

第 3 章 使用 Scheduler 进行线程调度 119

3.1 什么是 Scheduler 119
3.2 Scheduler 的类型 121
3.2.1 computation 121

3.2.2 newThread 122
3.2.3 io 122
3.2.4 immediate 123
3.2.5 trampoline 123
3.2.6 from 123
3.3 总结 125

第 4 章 RxJava 的实现原理 126

4.1 数据的发送和接收 126
4.1.1 创建 Observable 的过程 127
4.1.2 订阅的过程 128
4.2 操作符的实现 130
4.2.1 lift 的工作原理 130
4.2.2 map 的工作原理 132
4.2.3 flatMap 的工作原理 135
4.2.4 merge 的工作原理 136
4.2.5 concat 的工作原理 139
4.3 Scheduler 的工作原理 144
4.3.1 Scheduler 源码 144
4.3.2 subscribeOn 的工作原理 152
4.3.3 observeOn 的工作原理 156

第 5 章 RxJava 的应用实例 161

5.1 计算 π 的值 161
5.2 图片的三级缓存 165
5.2.1 内存缓存 167
5.2.2 外存缓存 169
5.2.3 网络缓存 172
5.2.4 缓存管理 173

5.2.5 封装 176
5.2.6 运行测试 178
5.3 结合 Retrofit 和 OkHttp 访问网络 181
5.3.1 卡片类的定义 181
5.3.2 配置 OkHttp 183
5.3.3 配置 Retrofit 186
5.4 使用 RxLifecycle 避免内存泄漏 189
5.4.1 修改 demo 工程 189
5.4.2 绑定其他生命周期 191
5.5 使用 RxBinding 绑定各种 View 事件 193
5.5.1 绑定点击事件 194
5.5.2 绑定 TextWatcher 196
5.5.3 绑定 OnPageChangeListener 197
第 6 章 RxJava 2 的改进 200
6.1 Observable 和 Flowable 200
6.2 null 的使用 203
6.3 Single 和 Completable 205
6.4 Maybe 207
6.5 Subscriber 208
6.5.1 DefaultSubscriber 209
6.5.2 ResourceSubscriber 210
6.5.3 DisposableSubscriber 211
6.6 Action 和 Function 212
6.7 错误处理 214
6.8 Scheduler 216

第 1 章
走进 RxJava 的世界

1.1 响应式编程

什么是响应式编程？响应式编程是一种以异步数据流为核心的编程方式。这里的数据一般是一些事件；而流则是在时间序列上的一系列的事件。任何东西都可以转化为数据流，如变量、用户输入事件、数据结构等。

我们可以很灵活地操纵数据流，如可以将两个甚至多个数据流融合成一个数据流，可以从数据流中过滤出感兴趣的事件，还可以将数据流中的事件转化为其他新的事件。数据流中的事件通常可以分成三种类型：普通事件、错误事件和结束事件。以用户的键盘输入事件为例，当用户依次敲击“A”、“B”、“C”键的时候，就会产生三个输入事件，计算机接收到这些事件并对其做出响应——将字母“A”、“B”、“C”显示在显示器上。当用户敲击回车键时，可以将其作为一个结束事件来表示数据流的结束，即用户输入结束。而在输入过程中发生的任何错误都可以作为数据流中的错误事件。

我们为什么要使用响应式编程呢？考虑这样的一种场景：用户登录一个购物客户端，这时客户端会将自己的用户名和密码通过网络请求发送出去，然后通过注册一个回调接口来监听请求的结果，因为结果只有成功和失败两种，所以回调里需要有 onSuccess 和 onError 两个接口来分别处理这两种情况，如代码 1-1-1 所示。

代码 1-1-1

```
void sendRequest(String userName,String passwrod) {
    client.sendLoginRequest(userName, passwrod, new Callback() {
```

```
        public void onSuccess(UserInfo info) {
            //处理登录成功的操作
        }
        public void onError(Exception e) {
            //处理错误的操作
        }
    });
}
```

通过注册一个回调接口来监听请求的结果，这其实也算一种响应式编程了。如果在登录成功后，我们要根据用户的 ID 来请求用户的购物记录，又该怎么办呢？很明显，应该在 onSuccess 方法里面继续请求。同登录请求一样，也需要注册一个回调接口来监听请求的结果，让我们继续在代码 1-1-1 的基础上添加代码，如代码 1-1-2 所示。登录成功后我们发出了查询购物记录的请求，并在 onSuccess 方法里将购物记录展示到 UI 上面。

代码 1-1-2

```
client.sendLoginRequest(userName, passwrod, new LoginCallback() {
    public void onSuccess(UserInfo info) {
        client.sendRecordRequest(info.ID, new RecordCallback() {
            public void onSuccess(List<Record> list) {
                //在 UI 上展示购物记录
                view.update(list);
            }
            public void onError(Exception e) {
                //处理错误的操作
            }
        });
    }
    public void onError(Exception e) {
        //处理错误的操作
    }
});
```

到这一步感觉还不错——除了有回调的嵌套外。如果我们有更进一步的需求：要求只展示最近一个月的购物记录（不考虑服务器端的实现），该怎么办呢？看来只能继续在 onSuccess 方法里改了，如代码 1-1-3 所示。为了看起来简单，我们忽略了外层的嵌套。

代码 1-1-3

```
public void onSuccess(List<Record> list) {
    Iterator<Integer> iterator = list.iterator();
    while (iterator.hasNext()) {
        Record record = iterator.next();
        if (record.time < getTime())
            iterator.remove();
    }
    //在 UI 上展示购买记录
}
```

新的需求又来了：每条购买记录里可能包含多个物品，之前只展示了每条购买记录，现在我们需要将所有购买记录里的所有物品都展示到 UI 上。继续改，如代码 1-1-4 所示。

代码 1-1-4

```
public void onSuccess(List<Record> list) {
    Iterator<Integer> iterator = list.iterator();
    while (iterator.hasNext()) {
        Record record = iterator.next();
        if (record.time < getTime())
            iterator.remove();
    }
    List<Item> itemList = new ArrayList<>();
    for (Record record : list) {
        for (Item item : record.getItemList()) {
            itemList.add(item);
        }
    }
    //在 UI 上展示物品记录
    view.update(itemList);
}
```

终于再次改完了，但是还可能有其他需求出现，如只显示金额大于 100 元的物品、将所有相同的物品合并后显示、有些物品需要请求网络获取更详细的信息，等等。看到这里，你会不会很崩溃？传统的命令式编程在处理这样的需求时就是非常痛苦的，但是使用响应式编程就可以游刃有余地处理。如果使用响应式编程，这些要求可以很容易地串成一条链式调用，如图

1-1-1 所示。读者可能不是很理解图中每一步的操作符是什么意思，不要担心，相信你读完本书后就可以很轻松地将该图中的链式调用实现出来。

图 1-1-1

通过上面的实例和图 1-1-1，我们可以看到响应式编程提高了代码的抽象层级，所以我们只需要关注与业务逻辑相关的事件，而不必去纠缠里面的一些细节。不仅如此，响应式编程的链式调用还可以消除回调嵌套，所以使用响应式编程写出来的代码往往会更加简明易懂。响应式编程非常适合以下几种情况：

1. 鼠标的点击、移动事件，键盘的输入事件，移动设备上的各种触摸和手势事件。
2. 当用户的位置改变时，其移动设备上的 GPS 信号和陀螺仪信号。
3. 各种耗时的操作，如读取硬盘内容以及从网络请求数据，这些操作一般都是异步的。
4. 涉及数据的转化、组合、过滤等操作的场景。

1.2 什么是 RxJava

RxJava 是一个非常著名的开源库，是 ReactiveX（Reactive Extensions）的一种 Java 实现。ReactiveX 是一种响应式扩展框架，有很多种实现，如 RxAndroid、RxJS、RxSwift、RxRuby、RxCpp、RxGo 等。最近几年 RxJava 在 Java 和 Android 的开发中得到了广泛的使用。截至本书写作时，RxJava 在 GitHub 上已经超过 28000 个 Star，想来突破 30000 也不是很遥远的事情。目

前 RxJava 有 1.x 和 2.x 两个主要的分支，分别代表着 RxJava 1 和 RxJava 2。由于 RxJava 1 发布的时间较早，使用也更广泛，所以本书的内容主要是针对 RxJava 1 的。但是 RxJava 2 已经发布，以后也会逐渐流行起来，所以在本书的最后一章会对 RxJava 2 做一些简单的介绍。

RxJava 可以看作由 Observable、Subscriber 和 Scheduler 组成的，它们的关系如图 1-2-1 所示。Subscriber 订阅到 Observable，Observable 会在默认或者指定的 Scheduler 上工作并产生数据流返回给 Subscriber，Subscriber 也会在默认或者指定的 Scheduler 上接收 Observable 发送过来的数据流。Scheduler 是对线程的一种抽象，不同的 Scheduler 代表了不同的线程。

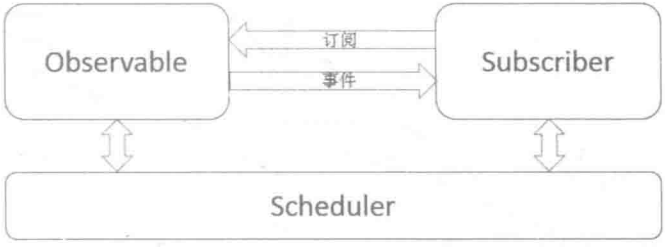


图 1-2-1

1.3　Observable 和 Subscriber

Observable 和 Subscriber 是 RxJava 的重要组成部分。Observable 提供了 subscribe 方法，当有 Subscriber 通过 subscribe 方法订阅到 Observable 时，Observable 就可以向 Subscriber 发送数据流。在 1.1 节中我们了解到响应式编程中的事件分为三类：普通事件、错误事件和结束事件，在 Subscriber 中有三个方法与这三种事件一一对应，Observable 会通过调用 Subscriber 的这三个方法来发送对应的事件。

1. onNext：当 Observable 要发送普通事件时，就会调用这个方法。这个方法可以被调用 0～*N* 次。

2. onError：当在 Observable 内部有异常或者错误产生的时候，就可以调用这个方法来向 Subscriber 发送错误事件。这个方法只能被调用 1 次。

3. onComplete：如果 Observable 已经发送完所有的数据，并且没有发生错误，这时就需要调用这个方法来向 Subscriber 发送结束事件。这个方法也只能调用 1 次，而且和 onError 是

互斥的关系，也就是说调用了 onError 后就不能调用 onComplete，反之亦然。在 onError 或者 onComplete 被调用之后，Observable 就失去了作用，不能再调用 onNext 来发送数据了。

Subscriber 还提供了 unsubscribe 方法，当 Subscriber 订阅到 Observable 之后，可以随时调用这个方法来终止对 Observable 的订阅。

代码 1-3-1 定义了一个方法，该方法创建一个 Observable 并将其返回。创建的 Observable 会发送随机生成的 5 个小于 10 的整数。当生成的随机数大于 8 时，我们假设有了异常，然后调用 onError 抛出一个异常。

代码 1-3-1

```
private Observable<Integer> createObserver() {
    return Observable.create(new Observable.OnSubscribe<Integer>() {
        @Override
        public void call(Subscriber<? super Integer> subscriber) {
            if (!subscriber.isUnsubscribed()) {
                for (int i = 0; i < 5; i++) {
                    int temp = new Random().nextInt(10);
                    if (temp > 8) {
                        //如果 value>8，则创建一个异常
                        subscriber.onError(new Throwable("value >8"));
                        break;
                    } else {
                        subscriber.onNext(temp);
                    }
                    // 没有发生异常，正常结束
                    if (i == 4) {
                        subscriber.onCompleted();
                    }
                }
            }
        }
    });
}
```

接下来建立一个 Subscriber 对象并将其注册给创建的 Observable 对象，然后接收其发送来的数据，参见代码 1-3-2。其中 log 函数是自定义的函数，会将结果输出。