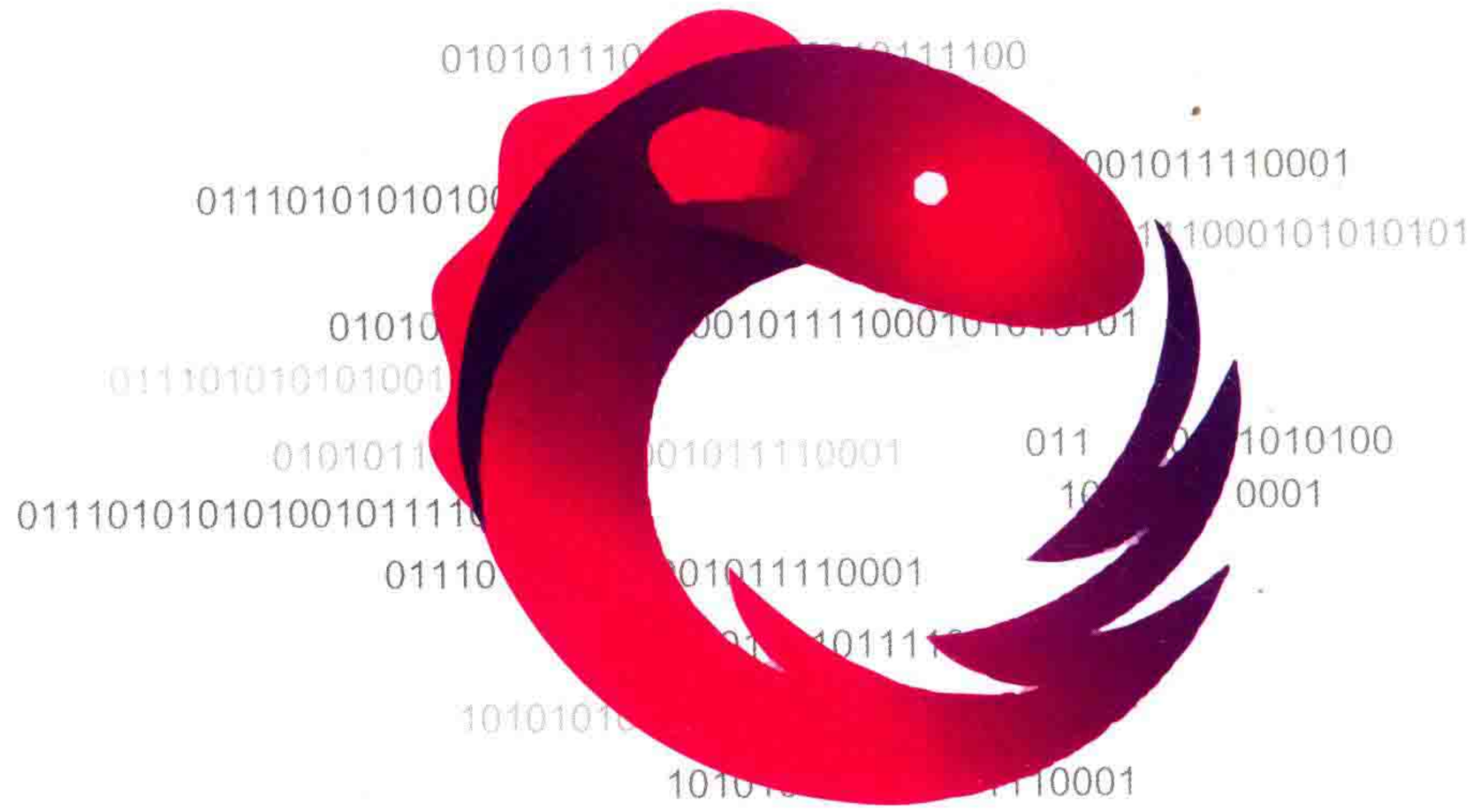
Broadview®
www.broadview.com.cn

RxJava 2.x 实战

沈哲 编著



本书通过完整的体系，深入浅出地讲解了RxJava 2.x的方方面面

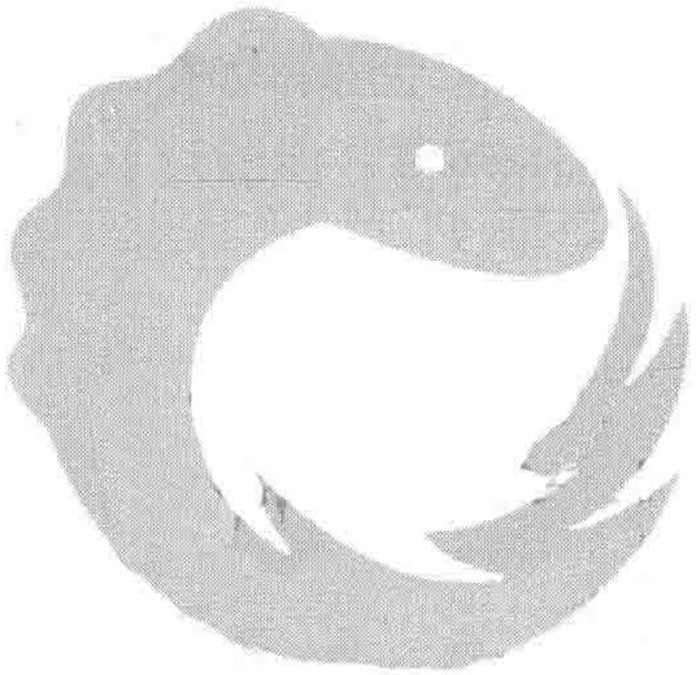
并通过案例详细讲解了如何解决一些生产环境的实战问题

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 2.x 实战

沈哲 编著



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

内 容 简 介

本书首先讲解了函数式响应式编程的概念，包括 Observables、Subject、Processor 等，以及 RxJava 的优点和用途。然后讲解了 RxJava 中必不可少的操作符，包括创建操作符、变换操作符、过滤操作符、条件操作符、布尔操作符、合并操作符和连接操作符。接着详细介绍了 RxJava 的线程操作及线程模型，用大量的实例，讲解了 RxJava 在 Android 上和在 Java 后端的使用。最后，介绍了 Java 8 的函数式编程的特性，以及对未来编程方式的展望。

本书适合 Android 开发工程师、Java 后端开发工程师，以及对函数响应式编程、感兴趣的 IT 从业人员。

图书在版编目（CIP）数据

RxJava 2.x 实战 / 沈哲编著. —北京：电子工业出版社，2018.4
ISBN 978-7-121-33722-2

Ⅰ. ①R… Ⅱ. ①沈… Ⅲ. ①JAVA 语言—程序设计 Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字(2018)第 031196 号

责任编辑：安　娜
印　　刷：三河市华成印务有限公司
装　　订：三河市华成印务有限公司
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：26.75　字数：497 千字
版　　次：2018 年 4 月第 1 版
印　　次：2018 年 4 月第 1 次印刷
定　　价：79.00 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

编写这本书的目的

笔者从 2015 年开始关注 RxJava 以及响应式编程，当时网上的资料很少。三年后的今天，我们可以看到越来越多的 App 都在使用 Rx 相关的技术。在 Java 后端，Spring 5 也开始支持响应式编程。在开源的技术社区里，Rx、响应式编程、函数式编程都是热门话题。我们公司开发的 App，笔者也会强制大家使用 RxJava 或者 RxSwfit。本书通过完整的体系介绍了 RxJava 的方方面面。

对于不了解响应式编程的开发者，RxJava 的入门可能会有一些难度。笔者结合自身的学习以及使用情况，尽可能使用通俗易懂的讲解方式带领大家学习 RxJava。同时，本书还附有丰富的例子，从 Android 开发到 Java 后端的开发，相信能够让大家感受到 RxJava 的魅力。

读者对象

1）Android 开发工程师。

2）Java 后端开发工程师。

3）对函数式编程、响应式编程感兴趣的 IT 从业人员。

如何阅读本书

本书共分为 18 章。

第 1 章讲述了 RxJava 的来龙去脉，以及什么是响应式编程、什么是函数式编程。

第 2 章至第 7 章讲述了 RxJava 的基础概念，以及 RxJava 常用的操作符。

第 8 章至第 10 章为 RxJava 的高级部分。

第 11 章至第 12 章为 RxAndroid 的使用篇，介绍了常用的 RxBinding、Retrofit 等框架。

第 13 章至第 15 章为 RxJava 的实战篇，介绍了如何实现一个基于 RxJava 的 Event Bus 框架，以及 Spring Boot 如何与 RxJava 相结合使用。

第 16 章至第 18 章介绍了 Java 8 函数式编程的特性和 Kotlin，并展望未来。

勘误和支持

本书相关例子的源码都在 GitHub 上，地址：https://github.com/fengzhizi715/RxJavaInAction。

由于笔者水平有限，编写本书时难免会出现错误或者纰漏，恳请读者批评指正。读者可以关注笔者的公众号与笔者进行互动。或者通过邮箱：fengzhizi715@126.com，有关本书的任何问题都可以反馈给笔者，笔者期待与您的技术交流。

致谢

首先要感谢我的家人，最主要是感谢我的妻子。在写书期间，恰逢儿子的出生，她承担了绝大部分照顾儿子的责任。

感谢公司的支持与同事的帮助，特别是刘志强帮我整理了很多 RxJava 相关的资料，以及对本书部分章节进行了试读，并提出意见。

感谢 www.bsfans.com 罗波同学提供 UI 支持。

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。

- **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/33722

目　录

第1章

RxJava简介

1.1 你需要了解的函数响应式编程

如果你曾经使用过 Java，那么你一定听说过面向对象的编程思想，也可能听说过 AOP（Aspect Orient Programming，面向切面编程）的编程思想。本书要讲的是全新的编程思想，跟它们没有丝毫的联系。

1. 响应式编程（Reactive Programming，简称 RP）

在计算机中，响应式编程是一种面向数据流和变化传播的编程范式。这意味着可以在编程语言中很方便地表达静态或动态的数据流，而相关的计算模型会自动将变化的值通过数据流进行传播。

传统的编程方式是顺序执行的，需要等待直至完成上一个任务之后才会执行下一个任务。无论是提升机器的性能还是代码的性能，本质上都需要依赖上一个任务的完成。如果需要响应迅速，就得把同步执行的方式换成异步执行，方法执行变成消息发送。这就是异步编程的方式，它是响应式编程的重要特性之一。

响应式编程有以下几个特点。

◎ 异步编程：提供了合适的异步编程模型，能够挖掘多核 CPU 的能力、提高效率、降低延迟和阻塞等。

◎ 数据流：基于数据流模型，响应式编程提供一套统一的 Stream 风格的数据处理接口。与 Java 8 中的 Stream 相比，响应式编程除了支持静态数据流，还支持动态数据流，并且允许复用和同时接入多个订阅者。

◎ 变化传播：简单来说就是以一个数据流为输入，经过一连串操作转化为另一个数据流，然后分发给各个订阅者的过程。这就有点像函数式编程中的组合函数，将多个函数串联起来，把一组输入数据转化为格式迥异的输出数据。

响应式编程一方面在用户界面编程领域及基于实时系统的动画方面都有广泛的应用；另一方面，在处理嵌套回调的异步事件、复杂的列表过滤和变换的时候也都有良好的表现。

现在的 App 无论是原生 H5、HTML5 还是 Hybird，都会和与数据事件相关的 UI 事件进行大量的交互，使用响应式编程会显得更加得心应手。

这些年来前端比较流行的响应式设计，实际上是指网页能够自动调整布局和样式以适配不同尺寸的屏幕，与我们谈论的响应式编程是两个完全不同的概念。

2. 函数式编程（Functional Programming，简称 FP）

随着硬件能力的不断提升，单核 CPU 的计算能力几乎达到极限，CPU 已经进入了多核时代，程序员转而通过并发编程、分布式系统来应对越来越复杂的计算任务。

然而并发编程并不是银弹，作为一种基于共享内存的并发编程，多线程编程有常见的死锁、线程饥饿、竞争条件等问题，而且多线程的 Bug 也难以重现和定位。于是，函数式编程开始兴起。

在函数式编程中，由于数据是不可变的（immutable），因此没有并发编程的问题，是线程安全的。它将计算机运算看作数学中函数的计算，主要特点是将计算过程分解成多个可复用的函数，并且避免了状态及变量的概念。函数式编程虽然也可以归结到面向过程的程序设计，但其思想更接近数学计算。

函数式编程具有以下特点。

◎ 函数是“第一等公民”：所谓“第一等公民”（First Class），指的是函数与其他数据类型一样，处于平等地位，可以赋值给其他变量，也可以作为参数，传入另一个函数，或者作为别的函数的返回值。

◎ 闭包和高阶函数：闭包是起函数的作用并可以像对象一样操作的对象。与此类似，FP 语言支持高阶函数。高阶函数可以用另一个函数（间接地，用一个表达式）作为输入参数，在某些情况下，它甚至返回一个函数作为输出参数。这两种结构结合在一起，即可以用优雅的方式进行模块化编程，这是使用 FP 的最大好处。

◎ 递归：把递归作为控制流程的机制。例如，在 Haskell 的世界中，没有变量赋值和流程跳转，如果要实现一些简单的功能，比如求一个数组中的最大值，则需要借助递归来实现。

◎ 惰性求值（Lazy Evaluation）：它表示为“延迟求值”和“最小化求值”。惰性求值使得代码具备了巨大的优化潜能。支持惰性求值的编译器会像数学家看待代数表达式那样看待函数式编程的程序，即抵消相同项从而避免执行无谓的代码，安排代码执行顺序从而实现更高的执行效率甚至是减少错误。惰性求值另一个重要的好处是它可以构造一个无限的数据类型，无须担心由无穷计算所导致的内存溢出错误。

◎ 没有“副作用”（Side Effect）：指的是函数内部与外部互动（最典型的情况，就是修改全局变量的值），产生运算以外的其他结果。函数式编程强调没有“副作用”，意味着函数要保持独立，所有功能就是返回一个新的值，没有其他行为，尤其是不得修改外部变量的值。。

既然函数式编程已经能够解决并发的问题，那为何还需要响应式编程呢？

3. 函数响应式编程（Functional Reactive Programming，简称 FRP）

函数响应式编程结合了函数式和响应式的优点，把函数范式里的一套思路和响应式编程合起来就是函数响应式编程。

我们知道，传统的面向对象编程是通过抽象出的对象关系来解决问题，函数式编程是通过函数（function）的组合来解决问题，响应式编程是通过函数式编程的方式来解决回调地狱（Callback Hell）的问题。

用传统的面向对象来处理异步事件不是很直观，处理并发也十分麻烦，所以才产生了函数响应式编程。

下面一起进入 RxJava 的世界，体验一下函数响应式编程带来的乐趣。

1.2 RxJava 简介

1. RxJava 产生的由来

RxJava 是 Reactive Extensions 的 Java 实现，用于通过使用 Observable/Flowable 序列来构建异步和基于事件的程序的库。

RxJava 扩展观察者模式以支持数据/事件序列，并添加允许你以声明方式组合序列的操作符，同时提取对低优先级的线程、同步、线程安全性和并发数据结构等问题的隐藏。

2. 什么是 Rx

ReactiveX 是 Reactive Extensions 的缩写，一般简写为 Rx，最初是 LINQ 的一个扩展，由微软架构师 Erik Meijer 领导的团队所开发，于 2012 年 11 月开源。Rx 是一个编程模型，目标是提供一致的编程接口，帮助开发者更方便地处理异步数据流。Rx 库支持 .NET、JavaScript 和 C++。由于 Rx 近几年越来越流行，因此现在已经支持几乎全部的流行编程语言。Rx 的大部分语言库由 ReactiveX 这个组织负责维护，比较流行的有 RxJava、RxJS 和 Rx.NET，社区网站是 http://reactivex.io/。

3. ReactiveX 的历史

微软给的定义是，Rx 是一个函数库，让开发者可以利用可观察序列和 LINQ 风格查询操作符来编写异步和基于事件的程序，通过使用 Rx，开发者可以用 Observables 表示异步数据流，用 LINQ 操作符查询异步数据流，用 Schedulers 参数化异步数据流的并发处理。可以这样定义 Rx：Rx = Observables + LINQ + Schedulers。

注：在 RxJava 2.x 中，Observables 包含了 Observables、Flowable、Single、Maybe、Completable 等。

ReactiveX.io 给的定义是，Rx 是一个使用可观察数据流进行异步编程的编程接口，ReactiveX 结合了观察者模式、迭代器模式和函数式编程的精华。

这个定义听上去很拗口，不过没有关系，在第 2 章和第 4 章会分别讲述 Observable 和 Scheduler。

4. Rx 模式

（1）使用观察者模式。

◎　创建：Rx 可以方便地创建事件流和数据流。
◎　组合：Rx 使用查询式的操作符组合和变换数据流。
◎　监听：Rx 可以订阅任何可观察的数据流并执行操作。

（2）简化代码。

◎　函数式风格：对可观察的数据流使用无副作用的输入/输出函数，避免了程序里错综复杂的状态。
◎　简化代码：Rx 的操作符通常可以将复杂的难题简化为很少的几行代码。
◎　异步错误处理：传统的 try/catch 没办法处理异步计算，Rx 提供了合适的错误处理机制。
◎　轻松使用并发：Rx 的 Observables（包括 Observable、Flowable、Single、Completable 和 Maybe）和 Schedulers 可以让开发者摆脱底层的线程同步和各种并发问题。

总而言之，RxJava 是 Reactive Extensions 在 JVM 平台上的一个实现，通过使用观察者序列来构建异步、基于事件的程序。RxJava 可以说是观察者设计模式的一个扩展，支持不同的数据/事件流和额外的操作类，允许通过声明式的方式构建不同的执行序列，通过抽象的方式屏蔽底层的多线程实现、同步、线程安全、并发数据结构、非阻塞 I/O 等逻辑。 RxJava 支持 Java 5 之后的版本，还支持跑在 JVM 上的各种其他语言，例如 Groovy、Clojure、JRuby、Kotlin 和 Scala 等。笔者建议尽量使用 Java 8 及以上的版本，因为能够使用 Lambda 表达式等新特性。

RxJava 使用 Observables 来访问多对象的异步序列。 Observables 是可组合的，Java Futures 模型在用于单一层面的异步执行方面比较方便，但是在需要异步嵌套执行时使用起来却很复杂，至少在 Java 8 出现之前是这样的。用 Futures 在构建有条件的异步执行流时会非常困难，因为在运行时每次请求的延迟都是不可控的。虽然理论上能够做到，但是过程会很复杂，有时候会在 Future.get()上一直阻塞，这样的结果显然无法体现异步执行的优势。在 Java 8 之后新增了 CompletableFuture，弥补了原先 Futures 模型的问题。

当然，RxJava 不并限于是在 Java 8 之后，还是之前使用。 Observables 本身是专门用于构建异步数据流和执行片段的，它很灵活，RxJava 的 Observables 除了支持 Futures 模型的所有功能外，还支持无限的数据流；Observables 本身是一个统一抽象，用于支持不用情况下的异步数

据序列的执行过程。

1.3 为何选择 RxJava

Rx 扩展了观察者模式用于支持数据和事件序列，添加了一些操作符，让你可以使用声明式的方式来组合这些序列，而无须关注底层的实现，如线程、同步、线程安全、并发数据结构和非阻塞 I/O。

Rx 的 Observable 模型让开发者可以像使用集合数据一样操作异步事件流，对异步事件流使用各种简单、可组合的操作。

◎ 可组合

对于单层的异步操作来说，Java 中 Future 对象的处理方式非常简单有效，但是一旦涉及嵌套，它们就开始变得异常烦琐和复杂。在 Java 8 之前，使用 Future 很难很好地组合带条件的异步执行流程。从另一方面来说，RxJava 的被观察者们（Observable/Flowable/Single/Completable/Maybe）一开始就是为组合异步数据流准备的。

◎ 更灵活

RxJava 的 Observable 不仅支持处理单独的标量值（就像 Future 可以做的），还支持数据序列，甚至是无穷的数据流。Observable 是一个抽象概念，适用于任何场景。Observable 拥有它的近亲 Iterable 的全部优雅与灵活。

Observable 是异步的双向 push，Iterable 是同步的单向 pull，可以通过表 1-1 进行对比。

表 1-1

事　　件	Iterable（pull）	Observable（push）
获取数据	T next()	Consumer<? super T> onNext
异常处理	throws Exception	Consumer<? super Throwable> onError
任务完成	!hasNext()	Action onComplete

◎ 无偏见

Rx 对于并发性或异步性没有任何特殊的偏好，Observable 可以用任何方式（如线程池、事件循环、非阻塞 I/O 或 Actor 模式）来满足我们的需求。无论我们选择怎样实现它，无论底层实

现是阻塞的还是非阻塞的，客户端代码都将与 Observable 的全部交互当成是异步的。

1. Observable 是如何实现的

public Observable<T> getData();

◎　它能与调用者在同一线程同步执行吗？
◎　它能异步地在单独的线程执行吗？
◎　它会将工作分发到多个线程，返回数据的顺序是任意的吗？
◎　它使用 Actor 模式而不是线程池吗？
◎　它使用 NIO 和事件循环执行异步网络访问吗？
◎　它使用事件循环将工作线程从回调线程分离出来吗？

从 Observer 的视角来看，这些都无所谓，重要的是：使用 Rx，可以改变我们的观念，可以在完全不影响 Observable 程序库使用者的情况下，彻底地改变 Observable 的底层实现。

2. 使用回调存在很多问题

回调在不阻塞任何事情的情况下，解决了 Future.get()过早阻塞的问题。响应结果一旦就绪，Callback 就会被调用，它们天生就是高效率的。不过，就像使用 Future 一样，对于单层的异步执行来说，回调很容易使用，对于嵌套的异步组合而言，它们显得非常笨拙。

3. Rx 是一个多语言的实现

Rx 在大量的编程语言中都有实现，并尊重实现语言的风格，而且更多的实现正在飞速增加。

4. 响应式编程

Rx 提供了一系列的操作符，我们可以使用它们来过滤（filter）、选择（select）、变换（transform）、结合（combine）和组合（compose）多个 Observable，这些操作符让执行和复合变得非常高效。

我们可以把 Observable 当作 Iterable 推送方式的等价物，使用 Iterable，消费者从生产者那拉取数据，线程阻塞直至数据准备好。使用 Observable，在数据准备好时，生产者将数据推送给消费者。数据可以同步或异步到达，这种方式更加灵活。

下面的例子展示了相似的高阶函数在 Iterable 和 Observable 上的应用。

```
// Iterable
```

```
getDataFromLocalMemory()
  .skip(10)
  .take(5)
  .map({ s -> return s + " transformed" })
  .forEach({ println "next => " + it })

// Observable
getDataFromNetwork()
  .skip(10)
  .take(5)
  .map({ s -> return s + " transformed" })
  .subscribe({ println "onNext => " + it })
```

Observable 类型给 GOF 的观察者模式添加了两种缺少的语义，这样就和 Iterable 类型中可用的操作一致了。

生产者可以发信号给消费者，通知它没有更多数据可用了（对于 Iterable，一个 for 循环正常完成表示没有数据了；对于 Observable，就是调用观察者的 onComplete 方法） 生产者可以发信号给消费者，通知它遇到了一个错误（对于 Iterable，在迭代过程中发生错误会抛出异常；对于 Observable，就是调用观察者（Observer）的 onError 方法）。有了这两种功能，Rx 就能使 Observable 与 Iterable 保持一致了，唯一的不同是数据流的方向。任何对 Iterable 的操作，都可以对 Observable 适用。

再举一个例子，某 App 从服务端获取酒店列表，并将价格大于等于 500 元的房间全部列出来展示在界面上。

```
new Thread() {
        @Override
        public void run() {
            super.run();
            //从服务端获取酒店列表
            List<Hotel> hotels = getHotelsFromServer();
            for (Hotel hotel : hotels) {
                List<Room> rooms = hotel.rooms;
                for (Room room : rooms) {
                    if (room.price >= 500) {
                        runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
```