

硕 士 研 究 生 读 书 报 告



题 目 探索RxJava

作者姓名 李晓雯

作者学号 21651112

指导教师 李启雷

学科专业 移动互联网与游戏开发

所在学院 软件学院

提交日期 二〇一七 年 一 月

Explore RxJava

A Dissertation Submitted to

Zhejiang University

in partial fulfillment of the requirements for

the degree of

Master of Engineering

Major Subject: Software Engineering

Advisor: Qilei Li

By

Xiaowen Li

Zhejiang University, P.R. China

2017

摘 要

RxJava是Java虚拟机实现的反应扩展。RxJava是一个库，用于通过使用观察序列构成异步和基于事件的程序库，可以展示创建于任何进程的任何异步数据流，并且能够被任何人和任何线程使用。RxJava通过观察者模式的设计方案来实现数据/事件的队列，并且加入了operator机制，来让开发者更好地对队列进行自定义化地组合。本文分析了当前Android上的线程操作的不足，介绍了RxJava的概念和基本实现，通过介绍观察者模式，解释了RxJava中包含的Observables，Observers，Observable Creation and Subscribers，Schedulers，Operators。本文介绍了为什么在Android中使用RxJava以及如何使用，最后介绍RxJava的防止内存泄漏机制。

**关键词**：RxJava，响应式编程，异步请求，Android

Abstract

RxJava is a Java VM implementation of ReactiveX, RxJava is a library that let’s you represent anything as an asynchronous data- stream that can be created on any thread, functionally transformed, and consumed by everyone on any thread. It extends the observer pattern to support sequences of data/events and adds operators that allow you to compose sequences together declaratively. This paper analyzes the current thread operations for Android, introduced the concept and basic realization of RxJava,Observer, Observer, Observable Creation and Subscribers, Schedulers, Operators included in RxJava are introduced by introducing the Observer pattern.This paper explains why the use of RxJava in Android and how to use, and finally introduced RxJava how to prevent memory leak mechanism.

**Keywords**：RxJava, Reactive Programming, Asynchrons request, Android

# RxJava简介

ReactiveX是一个专注于异步编程与控制可观察数据（或者事件）流的API。它组合了观察者模式，迭代器模式和函数式编程的优秀思想。实时数据处理是一件普通的现象，有一个高效、干净和可扩展的方式来处理这些情景是重要的。使用Observables和Operators来熟练操作它们。ReactiveX提供一个可组合又灵活的API来创建和处理数据流，同时简化了异步编程带来的一些担忧，如：线程创建和并发问题。

RxJava是ReactiveX在Java上的开源的实现。Observable（被观察者）和Subscriber（订阅者）是两个主要的类。在RxJava上，一个Observable是一个发出数据流或者事件的类，Subscriber是一个对这些发出的items（数据流或者事件）进行处理（采取行动）的类。一个Observable的标准流发出一个或多个item，然后成功完成或者出错。一个Observable可以有多个 Subscribers，并且通过Observable发出的每一个item，该item 将会被发送到Subscriber.onNext()方法来进行处理。一旦Observable不再发出items，它将会调用Subscriber.onCompleted()方法，或如果有一个出错的话Observable会调用Subscriber.onError()方法。Observable和Subscriber可以做任何事情，Observable可以是一个数据库查询，Subscriber用来显示查询结果；Observable可以是屏幕上的点击事件，Subscriber用来响应点击事件；Observable可以是一个网络请求，Subscriber用来显示请求结果。Observable和Subscriber是独立于中间的变换过程的。在Observable和Subscriber中间可以增减任何数量的map。整个系统是高度可组合的，操作数据是一个很简单的过程。

# Observables

Rxjava的异步数据流是由Observables发射的。[reactive extensions website](http://reactivex.io/intro.html)称Observable是“异步/发射对偶于Iterable的同步/获取”。虽然Java的Iterable接口对偶于RxJava的Observables是不完美的,但有助于介绍Observables和异步数据流,所以有必要介绍一下Java的Iterable接口是如何工作的。

每当我们使用for-each语法遍历集合时,我们都会利用Iterables接口的优势。如果我们正在构建我们的HackerNews客户端,我们可能需要遍历故事列表并打印这些故事的标题：

for (Story story : stories) {

Log.i(TAG, story.getTitle());

}

正如我们前面看到的这段代码，Iterables暴露出Iterator，可以访问一个聚合对象中的元素，也决定不再有未被使用的元素遗留在集合里。从客户和接口交互的角度看，任何实现了Iterable接口的对象，都可以访问已经预先定义好终止点的数据流。

Observables在这方面很像Iterables：它们也可以使用预先定义好终止点的数据流。

Obervables和Iterators关键的不同点是，Observables提供了异步数据流的使用，而Iterables提供的是同步。使用来自Iterator的数据会阻塞线程直到元素被返回。另一方面，想要消费Observable发射出数据的对象，需要注册才能接收。

为了使这个区别更明显，请再思考上面的在集合Collection中打印HackerNews故事标题的那段代码。被打印的故事不在内存中，每个故事不得不取自网络，同时我们还想在main线程上打印故事。在这里，我们需要故事流是一个异步流，那么使用Iterable来读取数据流中的每一个元素是不合适的。

相反地，我们应该使用Observable去读取由HackerNews API返回的每一个故事。我们知道通过调用Iterator的next()方法可以访问Iterable数据流中的元素。然而，我们不知道如何去读取Observable异步数据流中的元素。接下来带来RxJava中的第二个基本概念：Observer。

# Observers

Observers是Observables异步数据流的消费者。Observers能以它们想要的方式响应由Observable发射出的数据。例如，下面的Observer打印了由Observable发射出的故事标题:

storiesObservable.subscribe(new Observer<Story>() {

public void onNext(Story story) {

Log.i(TAG, story.getTitle());}});

这段代码和之前的for-each代码是非常相似的。在这两段代码中，我们都消费了一个已经预先定义好终止点的数据流。当我们使用for-each语法遍历集合时，iterator.hasNext()方法返回false时遍历停止。相似的，在前面的这段代码中，当异步数据流中没有元素时，onCompleted()方法被调用。

两段代码的主要区别是，当我们遍历集合时，我们同步地打印故事的标题，而当我们订阅stringsObservable时，我们是异步地打印故事的标题。Observer也能处理任何可能发生在Observable发出数据时的异常。Observer在onError()方法中处理这些errors。

为了看清楚为什么这是RxJava一个非常有用的特性，脑补一下由Observable发出的这些故事对象，是由JSON响应转换过来的。如果HackerNews API返回了一个有缺陷的JSON，那么转换成故事对象会发生异常，Observer可以调用onError()方法来处理JSON被解析时抛出的异常。

此时此刻，前面提到RxJava的两个概念会更清晰。让我们再来简单看看它的定义：  
RxJava is a library that allows us to represent any operation as an asynchronous data stream that can be created on any thread, declaratively composed, and consumed by multiple objects on any thread.我们仅仅看到了Observables允许我们把任何操作描绘成一个异步数据流。Observables和Iterables在他们提供有终止点的数据流使用上比较相似。我们现在也知道了Observables和Iterables的一个重要不同点：Observables暴露异步数据流然而Iterables暴露同步数据流。

Observers可以消费由Observables发射出的异步数据流。可以有多个Observers注册去接收由Observable发射的数据。Observers能处理任何可能发生在Observable发射数据时的异常，Observers也知道什么时候结束。

RxJava的定义到这里仍然会有一些是不清楚的。RxJava到底是如何允许我们把任何操作描绘成异步数据流的？换句话说，Observables是如何发射数据组成它们的异步数据流的？这些数据来自哪？这些问题我们将要在下一部分讲解。

# Observable Creation and Subscribers

Observables发出异步数据流。Observables发射他们数据的方式和Iterable接口显示他们数据流的方式很相似。Iterables接口和Iterators是迭代器模式的一部分，迭代器模式的主要目的是：  
提供一种方法顺序访问一个聚合对象中的各个元素，而又不暴露其内部的表示。

迭代器模式允许任何对象在不暴露底层实现情况下提供读取它的元素的方法。相似地，Observables以完全隐藏的方式提供了对异步数据流元素的使用，且独立于数据流被创建的过程。这也意味着Observables可以执行几乎任何的操作。

接下来用一个例子体现Observable的灵活性。Observables通常是由函数对象创建，这个函数对象可以取异步数据流中的数据并通知Subscriber这些数据可用。此外，Subscriber也就是Observer可以把它自己从Observable发射的数据中unsubscribe。

下面的代码会告诉你，如何创建Observable来发射从API服务器取回的HackerNews的故事：

Observable.create(new Observable.OnSubscribe<Story>() { //1

@Override

public void call(Subscriber<Story> subscriber) {

if (!subscriber.isUnsubscribed()) { //2

try {

Story topStory = hackerNewsRestAdapter.getTopStory(); //3

subscriber.onNext(topStory); //4

Story newestStory = hackerNewsRestAdapter.getNewestStory();

subscriber.onNext(newestStory);

subscriber.onCompleted(); //5

} catch (JsonParseException e) {

subscriber.onError(e); //6}}}});

让我们一步一步地浏览这段代码发生了什么：

OnSubscribe提供了这段代码什么时候被执行的线索：通过调用Observable.subscribe()方法，Observer注册去接收由这个Observable发射的数据时。

在发射数据前，我们可以检测Subscriber是否已经解订阅。请记住：Subscriber就像是Observer，它可以从发射数据的Observable中unsubscribe。

调用这个方法我们取出了HackerNews的数据。注意这是一个同步的方法调用，线程将阻塞直到故事被返回。

我们通知了已经订阅Observable的Observer，有新故事可用。Observer会被Subscriber封装并被传递进call()方法。Subscriber的封装，进一步简化了对封装好的Observer的调用。

当Observable的数据流中不再有故事发出时，我们会通知Observer调用onCompleted()方法。

如果JSON解析出现异常，我们会通知Observer调用onError()方法。

在Activitys中创建Observables可能导致内存泄露前面的代码如果在Activity中调用可能导致内存泄漏。

正如你之前看到的那段代码一样，Observables能通过相当多的操作创建。Observable被创建的灵活性和Iterables接口很相似。实现了Iterable接口的任何对象, 都可以暴露出同步数据流。同样地，Observable的数据流能被任何对象创建，只要将对象传递进Observable.OnSubscribe方法中即可。

到这里，聪明的读者可能会好奇Observables是否真的发射出了异步数据流。思考前面的例子，他们可能想知道，当Observable.subscribe()方法被调用时，是否在Observable.OnSubscribe函数对象中的call()方法也被调用了？方法的调用是否锁住了hackerNewsRestAdapter中的同步方法？是不是直到调用Observable.subscribe()阻塞了主线程Observable才完成发射由hackerNewsRestAdapter返回的故事？

这确实是一个好问题。在这里Observable.subscribe()方法会阻塞主线程。然而RxJava的另一个部分能阻止它的发生：Scheduler。

# Schedulers

Schedulers决定了Observables在哪个线程发射他们的异步数据流，也决定了Observers在哪个线程消费这些数据流。对Observable应用正确的Scheduler会阻止Observable.OnSubscribe的call()方法运行在main线程上：

Observable.create(new Observable.OnSubscribe<Story>() {  
 // ...}).subscribeOn(Schedulers.io());

顾名思义，Schedulers.io()会返回一个Scheduler，用来调度Observable.OnSubscribe对象里面的call()方法跑在一个I/O线程上。

使用Scheduler还有另一个方法:observeOn()。使用这个方法决定Observer在哪一个线程上消费由Observable subscribeOn()方法发出的数据，因此你能把observeOn()方法关联到通过subscribeOn()返回的Observable上:

Observable.create(new Observable.OnSubscribe<Story>() {  
 //...}).subscribeOn(Schedulers.io()).observeOn(AndroidSchedulers.mainThread());

AndroidSchedulers.mainThread()实际上不属于RxJava库，但是可以通过依赖RxAndroid库来使用。调用observeOn()方法主要点在于，你可以修改Observer消费由Observable发出的数据的线程。

subscribeOn()和observeOn()两个方法是你能修改由Observable发出的数据流通用的方式：operators。现在，让我们回到RxJava的定义：

RxJava is a library that allows us to represent any operation as an asynchronous data stream that can be created on any thread, declaratively composed, and consumed by multiple objects on any thread.

这一节我们讲述了RxJava是如何允许我们在任意一个线程上创建并消费异步数据流的。这个定义里面还有一个地方是我们不是很清楚的组合。不清楚的这个组合其实是和操作符相关的。

# Operators

在前面章节中我们讨论了Schedulers，在Observable.subscribeOn()和Observable.observeOn()方法中使用。这两个方法都是operators。Operators允许我们组合Observables。为了更好的理解operators，让我们来简短地分解定义中的组合。

组合Observable仅仅只是“制作”一个复杂的Observable。在复杂的功能实现上，由operators组合而成的Obserable和函数组合是非常相似的。在函数组合中，复杂的功能是通过一个函数的输出作为另一个函数的输入实现的。

例如，考虑Math.ceil(int x)函数。它返回要么比x大，要么与x相等的integer类型的整数。例如，Math.ceil(1.2)将要返回2.0。现在，假设我们有一个takeTwentyPercent(double x)方法简单地返回传进去值的20%。如果我们想要写一个方法计算小费，我们能组合Math.ceil()和takeTwentyPercent()去定义这个函数:

double calculateGenerousTip(double bill) {

return takeTwentyPercent(Math.ceil(bill));}

复杂的函数calculateGenerousTip()是传递Math.ceil(bill)的返回值作为takeTwentyPercent()输入得到的结果组成的。

操作符允许我们像calculateGenerousTip()方法的组成那样组合Observables。一个操作符是被应用于一个源Observable并且作为应用的结果它将返回一个新的Observable。例如，看下面的代码，源Observable是storiesObservable:

Observable<String> ioStoriesObservable = storiesObservable

.subscribeOn(Schedulers.io());

ioStoriesObservable，作为应用了subscribeOn操作符的结果返回的当然是一个Observable了。在操作符被应用之后，返回的Observable是更加复杂的：它表现不同于源Observable的地方在于它发射出的数据在一个I/O线程。

我们可以拿subscribeOn操作符返回的Observable和另一个操作符去进一步地组合成最终我们想要订阅的Observable。把subscribeOn和observeOn两个操作符链接在一起，确保故事标题的异步流在后台线程发射并且消费在main线程：

Observable<String> androidFriendlyStoriesObservable = storiesObservable

.subscribeOn(Schedulers.io())

.observeOn(AndroidSchedulers.mainThread());

我们看到Observable的组合就像calculateGenerousTip()方法的组合一样，传递Math.ceil()方法的输出作为takeTwentyPercent()方法的输入。相似地，传递应用了subscribeOn操作符的输出作为应用observeOn操作符的输入来组合成androidFriendlyStoriesObservable。

operators的这种组合方式声明了Observable。当我们使用operator时，我们可以简单地指定我们想要组合的Observable，而不是提供一个跳出组成我们Observable的行为实现。例如，当我们应用observeOn和subsceibeOn操作符时，我们就不是被迫的和Threads,Executors，Handlers一起工作。反而，我们可以简单地将Scheduler传递进这些操作符中并且这个Scheduler有责任确保组合成我们想要的Observable。这样，RxJava帮我们避免了复杂的和易于出错的异步数据的传输。

组合一个“android friendly”的Observable，在后台线程发射数据并且在主线程将这些数据传递给Observer就像你使用操作符开始的一样。联系上下文，看一看操作符是如何被使用的，是一种学习操作符是如何工作的并且它是如何帮助你的非常好的方式。这是下一章节我们将要详细讨论的事情。

现在，我们简单地介绍另一个在出现在我们HackerNews故事例子中的操作符。map操作符允许我们把一个发射故事的Observable变成一个发射这些故事标题的Observable。看起来像这样:

Observable.create(new Observable.OnSubscribe<Story>() {

// Emitting story objects... })

.map(new Func1<Story, String>() {

@Override

public String call(Story story) {

return story.getTitle();}})

map操作符会返回一个新的Observable用来发射故事的标题，故事由Observable.create()创建的Observable发出。

此时此刻，我们已经足够的了解了RxJava是如何允许我们整洁地和声明性地处理异步数据。由于操作符的强大，我们能以一个发射HackerNews故事的Observable和在UI线程上消费的需求开始，应用一系列的操作符，在I/O线程发射HackerNews的故事, 但是将这些故事的标题传递给UI线程上的Observers。看起来像这样:

Observale.create(new Observable.OnSubscribe<Story>() {

//Emitting story objects..})

.map(new Func1<Story, String>() {

@Override

public String call(Story story) {

return story.getTitle(); }})

.subscribeOn(Schedulers.io())

.observeOn(AndroidSchedulers.mainThread());

多个操作符链在一起很混乱，一些Android的开发者推荐使用Retrolambda插件，这个库把Java8的lambda特性兼容到Java6，一个被Android完全支持的Java版本。Dan Lew在他的博客中推荐这个库。然而，Jake Wharton大神，指出了一个使用Retrolambda的重要缺陷：IDE中的代码不会匹配跑在设备上的代码，因为Retrolambda为了移植lambda特性重写了字节码。

在决定是否使用Retrolambda之前需要记住的是，Android Studio能折叠被传递进多种多样的RxJava方法的函数对象，使这些对象看起来像lambda。对于我来说，这能减轻对Retrolambda的需要。

# RxJava在Android中的使用

为什么要在Android中使用RxJava？

Reactive Extensions(Rx)是一系列接口和方法，为开发者提供了一种易懂且迅速简单易维护的方法。RxJava就是干这事儿的，提供一系列tools来帮你写出简洁的代码。举个例子:我们需要从数据库获取一组用户的链表数据，并展示出来。我们可以用AsyncTask后台查询数据库，获得的结果给Ui的适配器展示出来。单示例代码：

public class SampleTask extends AsyncTask<Void,Void,List<Users>> {

private final SampleAdapter mAdapter;

public SampleTask(SampleAdapter sampleAdapter) {

mAdapter = sampleAdapater;

}

@Override

protected List<Users> doInBackground(Void... voids) {

List<Users> users = getUsersFromDatabase();

return users; }

@Override

protected void onPostExecute(List<Users> users) {

super.onPostExecute(products);

if(users == null) {

showEmptyUsersMessageView(); return; }

for(User user : users){

mAdapter.add(user); }

mAdapter.notifyDataSetChanged();}}

现在有个新需求，要求只显示非guest的user，我们处理的方法是，在添加到adapter前加个条件判断是不是guset，或者改变数据库查询的条件。或者从数据库中获取另外的其他信息，跟user一并在这个adapter中显示出来呢？**这就是我们为什么要用RxJava了**：

public Observable<List<User>> fetchUsersFromDatabase() {

return Observable.create(new Observable.OnSubscribe<List<User>(){

@Override

public void call(Subscriber<? super List<User>> subscriber){

// Fetch information from database

subscriber.onNext(getUserList());

subscriber.onCompleted();}});}

像这样被调用：

fetchUsersFromDatabase()

.subscribeOn(Schedulers.io())

.observeOn(AndroidSchedulers.mainThread())

.subscribe(new Subscriber<List<User>>() {

public void onNext(List<User> users) {

//Do whatever you want with each user } });

当不显示guests时，RxJava很方便：

fetchUsersFromDatabase()

.filter(new Func1<User, Boolean>() {

@Override

public Boolean call(User user) {

return !user.isGuest();}})

.subscribeOn(Schedulers.io())

.observeOn(AndroidSchedulers.mainThread())

.subscribe(new Subscriber<User>() {

public void onCompleted() {}

public void onNext(User user) {

//Do whatever you want with each user }} );

传统的方式，即便是个简单的变更，为了保持优雅的接口化编程，我们也得创建新接口，重构代码来实现过滤。但是使用RxJava让这一切变得优雅了，我们只需要一个被观察者用来获取所有的信息，让后你就可以尽情的用这些[**方法**](https://github.com/ReactiveX/RxJava/wiki/Filtering-Observables)来过滤获取你想要的数据。这个问题完美搞定，当需要增加另外的查询结果和user一同显示在这个adapter中时：

fetchUsersFromDatabase()

.zipWith(fetchSomethingElseFromDatabase(), (users, somethingElse) -> {

/\*here combine users and something else into a new object\*/

})

.subscribe( o -> {

/\*use the combine object from users and something else to fill the adapter

\*/});

Rxjava让Android中的线程操作变得简单：

在Android开发中有一个常见的场景是需要在后台线程去分担一定量的工作，一旦该任务完成，会将结果回调到主线程去显示结果。

在Android中，我们有多种方法来做这样的事：用AsyncTasks，Loaders，Services 等。然而，这些解决方式通常不是最好的。Asynctasks 很容易导致内存泄露，CursorLoaders与ContentProvider需要大量的配置和设置样板代码，还有Services的目的是为了长时间在后台运营的，而不是处理快速完成的操作，如：做一个网络请求或者从数据库加载内容。

RxJava可以解决这些问题。下面这样的布局有一个按钮去开始一个长时间运行的操作，并且始终显示进度条，这样我们可以确保我们的操作是运行在后台线程的而不是在主线程。

<Button

android:id="@+id/start\_btn"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center\_horizontal"

android:text="@string/start\_operation\_text" />

<ProgressBar

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center\_horizontal"

android:indeterminate="true" />

一旦按钮被点击，它会禁用按钮并开启长时间运行的操作，并且一旦这个操作完成便会显示一个Snackbar，然后按钮会重新变得可点击。这里是一个用AsyncTask实现这个“长期运行的操作”的例子。这个按钮new了一个SampleAsyncTask并执行。

public String longRunningOperation() {

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

// error

}

return "Complete!";

}

private class SampleAsyncTask extends AsyncTask {

protected String doInBackground(Void... params) {

return longRunningOperation();}

protected void onPostExecute(String result) {

Snackbar.make(rootView, result, Snackbar.LENGTH\_LONG).show();

startAsyncTaskButton.setEnabled(true);

}}

现在，将这个AsyncTask用RxJava来实现。首先，需要添加以下内容到app的 gradle.build文件下:compile 'io.reactivex:rxjava:+'。然后创建一个Observable来调用这个操作,可以使用Observable.create()方法来做到。

Final Observable operationObservable = Observable.create(new

Observable.OnSubscribe() {

public void call(Subscriber subscriber) {

subscriber.onNext(longRunningOperation());

subscriber.onCompleted();}});

创建Observable调用longRunningOperation()方法，将返回的结果作为参数给onNext()方法，然后调用onCompleted()来完成Observable（注：在Observable去订阅之前，操作是不会被调用的）。接下来，当button被点击时，需要订阅Observable。添加了一个新的button用RxJava版本来处理任务。

startRxOperationButton.setOnClickListener(new View.OnClickListener() {

public void onClick(final View v) {

v.setEnabled(false);

operationObservable.subscribe(new Subscriber() {

public void onCompleted() {

v.setEnabled(true);}

public void onNext(String value) {

Snackbar.make(rootView, value,

Snackbar.LENGTH\_LONG).show();}});}});

点击新button时发现的进度显示会冻结，UI变得反应迟钝。这是因为我们还没有定义Observable应该在什么线程上，以及应该在什么线程去订阅它。这是RxJava的Schedulers（调度器）功能。

对于任何Observable你可以定义在两个不同的线程，Observable会操作在它上面。使用Observable.observeOn()可以定义在一个线程上，可以用来监听和检查从Observable最新发出的items（Subscriber的onNext，onCompleted和onError方法会执行在observeOn所指定的线程上），并使用Observable.subscribeOn()来定义一个线程，将其运行我们Observable的代码（长时间运行的操作）。

RxJava默认情况下是单线程的，你会需要利用observeOn()和subscribeOn()方法为你的应用带来多线程操作。RxJava附带了几个现成的Schedulers给Observables使用，如：Schedulers.io()（用于I/O操作），Schedulers.computation()（计算工作），和 Schedulers.newThread()（为任务创建的新线程）。然而，从Android的角度来看，你可能想知道如何把订阅代码执行到主线程。我们可以用RxAndroid库来实现这一目标。

[RxAndroid](https://github.com/ReactiveX/RxAndroid)是一个对RxJava的轻量级扩展为了Android的主线程提供Scheduler，也能去创建一个Scheduler用于运行在任何给定的Android Handler类上。用新的 Schedulers，Observable创建之前能让我们将其修改为在后台线程执行我们的任务，并将我们的结果推到主线程上。

要在APP中用RxAndroid，只要在gradle.build文件中添加这行代码就行了：compile 'io.reactivex:rxandroid:1.0.1'。

final Observable operationObservable = Observable.create(new Observable.OnSubscribe() {

@Override

public void call(Subscriber subscriber) {

subscriber.onNext(longRunningOperation());

subscriber.onCompleted();

}

})

.subscribeOn(Schedulers.io()) // subscribeOn the I/O thread

.observeOn(AndroidSchedulers.mainThread()); // observeOn the UI Thread

修改Observable将用Schedulers.io()去订阅，并用AndroidSchedulers.mainThread()方法将观察的结果返回到UI线程上。现在，点击Rx操作的按钮，可以看到当操作运行时它将不再阻塞 UI线程。

上述的例子利用了Observable类来发出结果，当一个操作仅仅只需要发出一个结果然后就完成的情况我们可以有另外一个选择。RxJava介绍了Single类。Single类可以用于创建像下面这样的方法：

Subscription subscription = Single.create(new Single.OnSubscribe() {

public void call(SingleSubscriber singleSubscriber) {

String value = longRunningOperation();

singleSubscriber.onSuccess(value); }})

.subscribeOn(Schedulers.io())

.observeOn(AndroidSchedulers.mainThread())

.subscribe(new Action1() {

public void call(String value) {

Snackbar.make(rootView, value,

Snackbar.LENGTH\_LONG).show();}}, new Action1() {

public void call(Throwable throwable) { // handle onError }});

当给一个Single类做订阅时，只有一个onSuccess的Action和onError的action。Single类有不同于Observable 的操作符，有几个操作符具有将Single转换到Observable的机制。例如：用Single.mergeWith()操作符，两个或更多同类型的Singles可以合并到一起去创建一个Observable，发出每个Single的结果给一个 Observable。

# 防止内存泄露

对于AsyncTasks所提到的缺点是，如果对于涉及了Activity或Fragment的处理不仔细的话，AsyncTasks可能会造成内存泄露。不幸的是，使用RxJava不会魔术般的缓解内存泄露危机，但是防止内存泄露是很简单的。

如果你一直在关注代码，你可能会注意到你调用的Observable.subscribe()的返回值是一个Subscription对象。Subscription类只有两个方法，unsubscribe()和isUnsubscribed()。为了防止可能的内存泄露，在你的Activity或Fragment的onDestroy 里，用 Subscription.isUnsubscribed()检查你的Subscription是否是unsubscribed。如果调用了 Subscription.unsubscribe()，Unsubscribing将会对items停止通知给你的Subscriber，并允许垃圾回收机制释放对象，防止任RxJava造成内存泄露。如果你正在处理多个 Observables和Subscribers，所有的Subscription对象可以添加到CompositeSubscription，然后可以使用CompositeSubscription.unsubscribe()方法在同一时间进行退订(unsubscribed)。

# 总结

RxJava通过观察者模式的设计方案来实现数据/事件的队列，并且加入了operator机制，来让开发者更好地对队列进行自定义化地组合。同时，RxJava对底层的线程、同步化，线程安全、多线程数据结构等组件也进行了一定程度的抽象提取。本读书报告通过阅读国外优秀的书籍和文章后，对RxJava的基础知识和在Android上的应用做了简单介绍。

Android开发，异步操作是必不可少的，异步操作很关键的一点是程序的简洁性，因为在调度过程比较复杂的情况下，异步代码经常会既难写也难被读懂。Android创造的AsyncTask和Handler ，其实都是为了让异步代码更加简洁。RxJava的优势也是简洁，但它的简洁的与众不同之处在于，随着程序逻辑变得越来越复杂，它依然能够保持简洁。对于Android开发者，学会使用RxJava对效率和代码质量的提升都有显著有好处。

参考文献

[1] RxJava for Android App Development．O’Reilly. [[EB](https://www.baidu.com/s?wd=EB&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3Phn3nHmvujT4mHbLuHbs0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHnzn1DdrHRs)/[OL](https://www.baidu.com/s?wd=OL&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3Phn3nHmvujT4mHbLuHbs0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHnzn1DdrHRs)]. http://www.oreilly.com/programming/free/rxjava-for-android-app-development.csp. 2017.1.2

[2] RxJava: Reactive Extensions for the JVM．Github. [[EB](https://www.baidu.com/s?wd=EB&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3Phn3nHmvujT4mHbLuHbs0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHnzn1DdrHRs)/[OL](https://www.baidu.com/s?wd=OL&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3Phn3nHmvujT4mHbLuHbs0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHnzn1DdrHRs)]. https://github.com/ReactiveX/RxJava. 2016.12.23

[3] Getting Started with RxJava and Android．[ALEX TOWNSEND](http://www.captechconsulting.com/search#q=Alex Townsend). [[EB](https://www.baidu.com/s?wd=EB&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3Phn3nHmvujT4mHbLuHbs0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHnzn1DdrHRs)/[OL](https://www.baidu.com/s?wd=OL&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3Phn3nHmvujT4mHbLuHbs0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHnzn1DdrHRs)]. <http://www.captechconsulting.com/blogs/getting-started-with-rxjava-and-android>. 2015.10.30