

我的微博

hi大头鬼hi

我的其他资料

我的Github

文章分类

Android (15)

Animation (1)

Gesture (1)

RxJava (7)

Square (2)

Gradle (8)

open resty (1)

weex (1)

m (1)

阅读排行

深入浅出RxJava (一: 基

深入浅出RxJava四-在An

深入浅出RxJava(二: 操

深入浅出RxJava三--响应

RxJava使用场景小结

Android热更新实现原理

RxJava基本流程和lift源码

Android实现类似QQ的消

Otto使用入门

深入浅出Android Gradle:

评论排行

深入浅出RxJava (一: 基

深入浅出RxJava(二: 操

深入浅出RxJava四-在An

深入浅出RxJava三--响应

RxJava基本流程和lift源码

RxJava使用场景小结

Android热更新实现原理

Android实现类似QQ的消

使用动画和fragment改善

深入浅出Android Gradle:

【系列直播】微信公众号开发技术实战 CSDN日报20170421 —— 《程序员必知的七个图形工具》 程序员4月书讯: Angular来了!

深入浅出RxJava四-在Android中使用响应式编程

标签: android RxJava RxAndroid 响应式编程

2015-04-13 22:41 86632人阅读 评论(32) 收藏 举报

分类: RxJava (6)

目录(?)

原文链接

在第1, 2, 3篇中, 我大概介绍了RxJava是怎么使用的。下面我会介绍如何在Android中使用RxJava.

RxAndroid

RxAndroid是RxJava的一个针对Android平台的扩展。它包含了一些能够简化Android开发的工具。

首先, AndroidSchedulers提供了针对Android的线程系统的调度器。需要在UI线程中运行某些代码? 很简单, 只需要使用AndroidSchedulers.mainThread():

```
1 retrofitService.getImage(url)
2   .subscribeOn(Schedulers.io())
3   .observeOn(AndroidSchedulers.mainThread())
4   .subscribe(bitmap -> myImageView.setImageBitmap(bitmap));
```

如果你已经创建了自己的Handler, 你可以使用HandlerThreadScheduler1将一个调度器链接到你的handler上。

接着要介绍的就是AndroidObservable, 它提供了跟多的功能来配合Android的生命周期。bindActivity()和bindFragment()方法默认使用AndroidSchedulers.mainThread()来执行观察者代码, 这两个方法会在Activity或者Fragment结束的时候通知被观察者停止发出新的消息。

```
1 AndroidObservable.bindActivity(this, retrofitService.getImage(url))
2   .subscribeOn(Schedulers.io())
3   .subscribe(bitmap -> myImageView.setImageBitmap(bitmap);
```

我自己也很喜欢AndroidObservable.fromBroadcast()方法, 它允许你创建一个类似BroadcastReceiver的Observable对象。下面的例子展示了如何在网络变化的时候被通知到:

```
1 IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);
2 AndroidObservable.fromBroadcast(context, filter)
3   .subscribe(intent -> handleConnectivityChange(intent));
```

最后要介绍的是ViewObservable,使用它可以给View添加了一些绑定。对于View的点击事件, 可以使用ViewObservable.clicks(), 或者你想监听TextView的内部事件:

```
1 ViewObservable.clicks(mCardNameEditText, false)
2   .subscribe(view -> handleClick(view));
```

Retrofit

大名鼎鼎的Retrofit库内置了对RxJava的支持。通常调用发可以通过使用

关闭

创意办公室设计

http://blog.csdn.net/lzyzd/article/details/45033611

1/6

个人资料



hi大头鬼hi



访问： 798342次

积分： 2518

等级： BLOG > 5

排名： 第12799名

原创： 0篇

转载： 0篇

译文： 12篇

评论： 360条

文章搜索

文章存档

2016年07月 (1)

2015年12月 (1)

2015年11月 (4)

2015年10月 (1)

2015年09月 (1)

展开

推荐文章

- * 探索通用可编程数据平面
- * 这是一份很有诚意的 Protocol Buffer 语法详解
- * CSDN日报20170420 ——《开发和产品之间的恩怨从何来?》
- * Android图片加载框架最全解析——从源码的角度理解Glide的执行流程
- * 如果两个程序员差不多，选写作能力更好的那个
- * 从构造函数看线程安全

最新评论

深入浅出RxJava(二：操作符)
Jeff169: lambda可读性有点差啊
楼主

深入浅出RxJava（一：基础篇）
birdlovestudy:
@dantingjuan:..from.filter(过滤符合要求的).first（发射满足条件的第一个...

RxJava基本流程和lift源码分析是个纠结人: 大神，很给力，结合之前看的，居然看懂了



1 2 3 4 5 6 7 8 9 10

```
1 @GET("/user/{id}/photo")
2 void getUserPhoto(@Path("id") int id, Callback<Photo> cb);
```

使用RxJava，你可以直接返回一个Observable对象。

```
1 @GET("/user/{id}/photo")
2 Observable<Photo> getUserPhoto(@Path("id") int id);
```

现在你可以随意使用Observable对象了。你不仅可以获取数据，还可以进行变换。

Retrofit对Observable的支持使得它可以很简单的将多个REST请求结合起来。比如我们有一个请求是获取照片的，还有一个请求是获取元数据的，我们就可以将这两个请求并发的发出，并且等待两个结果都返回之后再做处理：

```
1 Observable.zip(
2     service.getUserPhoto(id),
3     service.getPhotoMetadata(id),
4     (photo, metadata) -> createPhotoWithData(photo, metadata))
5     .subscribe(photoWithData -> showPhoto(photoWithData));
```

在第二篇里我展示过一个类似的例子（使用flatMap()）。这里我只是想展示以下使用RxJava+Retrofit地组合多个REST请求。

遗留代码，运行极慢的代码

Retrofit可以返回Observable对象，但是如果你使用的别的库并不支持这样怎么办？或者说一个内部的内码，你想把他们转换成Observable的？有什么简单的办法没？

绝大多数时候Observable.just() 和 Observable.from() 能够帮助你从遗留代码中创建 Observable 对象：

```
1 private Object oldMethod() { ... }
2
3 public Observable<Object> newMethod() {
4     return Observable.just(oldMethod());
5 }
```

上面的例子中如果oldMethod()足够快是没有什么问题的，但是如果很慢呢？调用oldMethod()将会阻塞住他所在的线程。

为了解决这个问题，可以参考我一直使用的方法-使用defer()来包装缓慢的代码：

```
1 private Object slowBlockingMethod() { ... }
2
3 public Observable<Object> newMethod() {
4     return Observable.defer(() -> Observable.just(slowBlockingMethod()));
5 }
```

现在，newMethod()的调用不会阻塞了，除非你订阅返回的observable对象。

生命周期

我把最难的不分留在了最后。如何处理Activity的生命周期？主要就是两个问题：

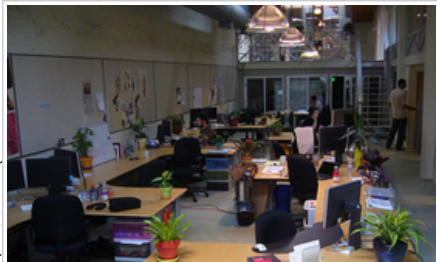
1.在configuration改变（比如转屏）之后继续之前的Subscription。

比如你使用Retrofit发出了一个REST请求，接着想在listview中展示结果。如果在网络请求的时候用户旋转了屏幕怎么办？你当然想继续刚才的请求，但是怎么搞？

2.Observable持有Context导致的内存泄露

这个问题是因为创建subscription的时候，以某种方式持有了context的容易发生！如果Observable没有及时结束，内存占用就会越来越大。不幸的是，没有银弹来解决这两个问题，但是这里有一些指导方案你可以

第一个问题的解决方案就是使用RxJava内置的缓存机制，这样你就可以unsubscribe/resubscribe，却不用重复运行得到Observable的代码。c（甚至你调用了unsubscribe也不会停止）。这就是说你可以在Activity一个新的Observable对象。



创意办公室设计



100% 关注



怎么实现呢？现...

深入浅出RxJava（一：基础篇）
阳光沙滩: 楼主，我想实现遍历一个list，list中的某个数据满足一定条件后就不执行后面的了，该怎么实现呢？现...

深入浅出RxJava三--响应式的好处
xuliangxu: 使用subscribeOn()指定观察者代码运行的线程，使用observerOn()指定订阅者运行的...

深入浅出RxJava（一：基础篇）
mm860659: 虽说这是翻译的文章，但确实不错，浅显易懂，百度的其他文章写得都不如这个。

RxJava基本流程和lift源码分析
Newbie00001: 分析的真不错。32个赞

```
1 Observable<Photo> request = service.getUserPhoto(id).cache();
2 Subscription sub = request.subscribe(photo -> handleUserPhoto(photo));
3
4 // ...When the Activity is being recreated...
5 sub.unsubscribe();
6
7 // ...Once the Activity is recreated...
8 request.subscribe(photo -> handleUserPhoto(photo));
```

注意，两次sub是使用的同一个缓存的请求。当然在哪里去存储请求的结果还是要你自己来做，和所有其他的生命周期相关的解决方案一延虎，必须在生命周期外的某个地方存储。（retained fragment或者单例等等）。

第二个问题的解决方案就是在生命周期的某个时刻取消订阅。一个很常见的模式就是使用CompositeSubscription来持有所有的Subscriptions，然后在onDestroy()或者onDestroyView()里取消所有的订阅。

```
1 private CompositeSubscription mCompositeSubscription
2     = new CompositeSubscription();
3
4 private void doSomething() {
5     mCompositeSubscription.add(
6         AndroidObservable.bindActivity(this, Observable.just("Hello World"))
7         .subscribe(s -> System.out.println(s)));
8 }
9
10 @Override
11 protected void onDestroy() {
12     super.onDestroy();
13
14     mCompositeSubscription.unsubscribe();
15 }
```

你可以在Activity/Fragment的基类里创建一个CompositeSubscription对象，在子类中使用它。

注意! 一旦你调用了 CompositeSubscription.unsubscribe(), 这个CompositeSubscription对象就不可用了, 如果你还想使用CompositeSubscription，就必须在创建一个新的对象了。

两个问题的解决方案都需要添加额外的代码，如果谁有更好的方案，欢迎告诉我。

总结

RxJava还是一个很新的项目，RxAndroid更是。RxAndroid目前还在活跃开发中，也没有多少好的例子。我打赌一年之后我的一些建议就会被看做过时了。

顶

69

踩

5

上一篇

深入浅出RxJava三--响应式的好处

下一篇

Gradle Tips#1-tasks

我的同类文章

RxJava （6）			
• RxJava使用场景小结	2015-11-30	阅读 33298	• RxJava基本
• 如何升级到RxAndroid 1.0	2015-10-19	阅读 8201	• 深入浅出Rx
• 深入浅出RxJava(二：操作符)	2015-03-06	阅读 72251	• 深入浅出Rx





创意办公室设计





IoT公寓







云服务器性价比王 68元/月

立即购买

2核CPU 5MBGP带宽 4G内存 150G数据盘

参考知识库

猜你在找

Android中的数据存储在Android之数据存储Android核心技术——Android数据存储Android核心技术——网络应用Android开发之初窥门径

深入浅出RxJava三响应式的好处深入浅出RxJava三--响应式的好处深入浅出RxJava三--响应式的好处深入浅出RxJava三--响应式的好处RxJava响应式函数编程

1



6.80/件

长期供应 3色豪华彩色跳绳无线无绳款 健身

2



2.20/条

供应跳绳 轴承跳绳 考试专用跳绳 学生中考

3



90.00/件

厂家直销专业体能训练

广告

查看评论

21楼 冲浪的水手 2017-04-01 17:19发表

好多方法都没有了，楼主该说明一下了

20楼 骑着老虎去溜猫 2017-02-14 15:36发表

大神你要是写几个Demo 就好了 还要确实可读性不好

19楼 咖喱弓 2016-11-21 17:09发表

谢谢入门，没用Lambda敲了一遍你说的情况，回头要学一下Lambda了

18楼 Code4Android 2016-11-02 17:42发表

谢谢入门

17楼 xxiang1x 2016-10-27 15:35发表

从1到4看完了，学习了。

16楼 游鑫 2016-08-16 20:00发表

https://github.com/youxin11544/mvp_hybride_framework（这是一个Android MVP模型良好的架构设计,同时也做了Android和HTML 5交互架构，用到了RxJava+Retrofit+MVP+泛型缩减mvp+模板模式+命令模式+观察者模式+管理者模式 +简单工厂模式

15楼 不良、青年 2016-06-29 19:12发表

老大能否把写的demo上传一下，新手看的不全 不太理解

14楼 Joney小鬼 2016-05-23 22:16发表

在新的rxJava和rxAndroid中怎么也找不到AndroidObservable和ViewObservable，新版本中没有了吗？

13楼 BeiBeiXiaXueLe 2016-04-25 21:47发表

楼主，能给在说下关于缓存的问题吗，特此感谢

12楼 lei小歪 2016-04-13 17:53发表

使用rxjava一年多了，总结了些demo和资料，需要的朋友复制下面链接：https://github.com/cn-ljb/rxjava_for_android

11楼 ginngi 2016-03-31 19:12发表

一年后入门学习，感觉很有用。。。

10楼 野生ChaoS 2015-12-21 11:15发表

看了抛物线的文章再来看大头哥的，感觉就像是复习。不过为什么要用 Lan

关闭



创意办公室设计





* 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

全部主题	Hadoop	AWS	手机游戏	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							

[网站客服](#)
 [杂志客服](#)
 [微博客服](#)
 webmaster@csdn.net
 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 |

京 ICP 证 09002463 号 | Copyright © 1999-2016, CSDN.NET, All Rights Reserved

