[RxJava 2.0 使用详解](http://blog.csdn.net/aiynmimi/article/details/53382567)

## 最基本的的使用

我们知道一个简单的RxJava的应用，需要一个观察者或者订阅者Observer，一个被观察者Observable，最后调用subscribe()方法将两者绑定起来！   
示例：

//创建观察者或者订阅者

Observer<String> observer = new Observer<String>() {

@Override

public void onSubscribe(Disposable d) {

//Disposable是1.x的Subscription改名的，因为Reactive-Streams规范用这个名称，为了避免重复

//这个回调方法是在2.0之后新添加的

//可以使用d.dispose()方法来取消订阅

}

@Override

public void onNext(String value) {

Log.e("onNext", value);

}

@Override

public void onError(Throwable e) {

Log.e("onError", e.getMessage());

}

@Override

public void onComplete() {

Log.e("onComplete", "complete");

}

};

//创建被观察者

Observable observable = Observable.create(new ObservableOnSubscribe() {

@Override

public void subscribe(ObservableEmitter e) throws Exception {

e.onNext("Hello World!");

}

});

observable.subscribe(observer);

这是一个非常简单的例子，由于1.x中Observable不能合理的背压，导致了无法意料的 MissingBackpressureException，所以在2.x中，添加了**Flowable来支持背压，而把Observable设计成非背压的。**   
还有一点需要注意的就是，在上边注释中也有，onSubscribe(Disposable d)**这个回调方法是在2.x中添加的，Dispose参数是由1.x中的Subscription改名的，为了避免名称冲突！**

所以上边的例子在2.x中，最好这么写：

//创建订阅者

Subscriber<String> subscriber = new Subscriber<String>() {

@Override

public void onSubscribe(Subscription s) {

//这一步是必须，我们通常可以在这里做一些初始化操作，调用request()方法表示初始化工作已经完成

//调用request()方法，会立即触发onNext()方法

//在onComplete()方法完成，才会再执行request()后边的代码

s.request(Long.MAX\_VALUE);

}

@Override

public void onNext(String value) {

Log.e("onNext", value);

}

@Override

public void onError(Throwable t) {

Log.e("onError", t.getMessage());

}

@Override

public void onComplete() {

//由于Reactive-Streams的兼容性，方法onCompleted被重命名为onComplete

Log.e("onComplete", "complete");

}

};

Flowable.create(new FlowableOnSubscribe<String>() {

@Override

public void subscribe(FlowableEmitter<String> e) throws Exception {

e.onNext("Hello,I am China!");

}

}, BackpressureStrategy.BUFFER)

.subscribe(subscriber);

在2.x中，我们在onSubscribe()回调中必须调用s.request()方法去请求资源，参数就是要请求的数量，一般如果不限制请求数量，可以写成Long.MAX\_VALUE，之后会立即触发onNext()方法！所以**当你在**onSubscribe()/onStart()**中做了一些初始化的工作，而这些工作是在**request()**后面时，会出现一些问题，在**onNext()**执行时，你的初始化工作的那部分代码还没有执行。为了避免这种情况，请确保你调用**request()**时，已经把所有初始化工作做完了。**

## 更简洁的写法

Flowable.just("Hello,I am China!")

.subscribe(subscriber);

//.subscribeWith(subscriber)//在1.x中此方法返回Subscription，而在2.x中是没有返回值的

//所以增加subscribeWith()方法，用来返回一个Disposable对象

//使得用户可以CompositeDisposable.add()方法添加对象。1.x为CompositeSubscription

//其他subscribe()重载方法返回Disposable

RxJava提供了just()方法来创建一个发射字符串的Flowable，然后调用subcribe()即可！   
这里还有一个需要注意的问题，就是在注释中写的subcribe()**方法有多种重载方法，只有**subscribe(subscriber)**这个重载方法时没有返回值的，但是在1.x中，此方法返回Subscription（上边也提到过，在2.x中改名为Disposable），用户经常添加**Subscription**到**CompositeSubscription**（2.x中改名为CompositeDisposable），为了弥补这一点，我们增加了**E subscribeWith(E subscriber)**方法，返回一个Disposable对象，使得用户可以CompositeDisposable.add()方法添加对象。**

而对于 Subscriber 来说，我们目前仅仅关心onNext方法。所以又可以这样写：

Flowable.just("Hello,I am China!")

//替代1.x中的action1,接收一个参数，如果是两个参数action2使用BiCustomer，而且删除了action3-9

//多个参数用Custom<Object[]>

.subscribe(new Consumer<String>() {

@Override

public void accept(String s) throws Exception {

Log.e("consumer", s);

}

});

需要注意的问题：**在1.x的API中，这里是Action1，在2.x中使用Consumer来代替，如果是两个参数，则用BiConsumer来代替Action2，而且在2.x中删除了Action3-9，如果是多个参数则用**Custom<Object[]>**代替ActionN。**

RxJava还有一个API能达到类似的效果，就是from()**，但是因为在使用java8编译时，javac不能够区分功能接口类型，所以它在2.x中被拆分为：**fromArray**,**fromIterable**,**fromFuture   
所以上边又可以这样写：

Flowable.fromArray("Hello,I am China!")

.subscribe(new Consumer<String>() {

@Override

public void accept(String s) throws Exception {

Log.e("consumer", s);

}

});

# 操作符

## map

首先看一个map的例子

Flowable.just("Hello,I am China!")

//将1.x中的Func1,2改为Function和BiFunction，Func3-9改为Function3-9

//多参数FuncN改为Function<Object[],R>

//这个第一个泛型为接收参数的数据类型，第二个泛型为转换后要发射的数据类型

.map(new Function<String, String>() {

@Override

public String apply(String s) throws Exception {

return s+"\_\_by Mars";

}

})

.subscribe(new Consumer<String>() {

@Override

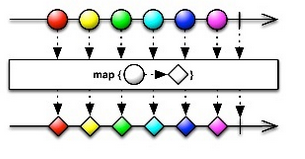
public void accept(String s) throws Exception {

Log.e("consumer", s);

}

});\*/

可以看出，例子中**map()将一个字符串对象，转换为另一个字符串对象返回，当然我们也可以将其转换为与之不同的对象，对应的返回的Flowable对象参数也会变为转换后的对象。另外Function的泛型第一个为接收参数的数据类型，第二个为转换后要发射的数据类型。**   
需要注意的问题：**在2.x中将1.x的**Func1**和**Func2**改为**Function**和**BiFunction**，**Func3-9**改为**Function3-9**，多参数**FuncN**改为**Function<Object[],R>

map()的逻辑操作图：   


## flatMap

首先看一个例子：

ArrayList<String[]> list=new ArrayList<>();String[] words1={"Hello,","I am","China!"};String[] words2={"Hello,","I am","Beijing!"};

list.add(words1);

list.add(words2);

Flowable.fromIterable(list)

.flatMap(new Function<String[], Publisher<String>>() {

@Override

public Publisher<String> apply(String[] strings) throws Exception {

return Flowable.fromArray(strings);

}

})

.subscribe(new Consumer<String>() {

@Override

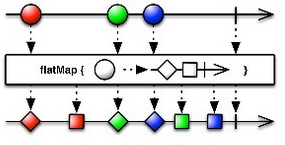
public void accept(String s) throws Exception {

Log.e("consumer", s);

}

});

从上边这个例子可以看出，**flatMap和map还是有共同点的，都是将一个对象转换为另一个对象，不同的是map只是一对一的转换，而flatMap可以是一对多的转换，并且是转换为另外一个Flowable对象！**

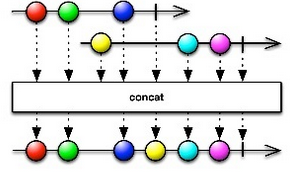
flatMap()的逻辑操作图：   


## lift和compose

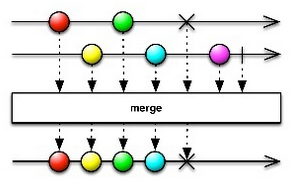
关于这些转换的使用和原理，可以参考扔物线的   
[给 Android 开发者的 RxJava 详解](http://gank.io/post/560e15be2dca930e00da1083" \l "toc_18)   
2.x中的用法基本相同

## concat和merge

## concat

逻辑操作图：   


## merge

逻辑操作图：   


上述所有逻辑操作图来自[这里](http://www.open-open.com/news/view/24a29ece)

# 其他api

Flowable.range(5,10)//从5开始数10个数(5——14)

.filter(new Predicate<Integer>() {//过滤为偶数

@Override

public boolean test(Integer integer) throws Exception {

return integer%2==0;

}

})

.take(2)//只要前2个数据

.subscribe(new Consumer<Integer>() {

@Override

public void accept(Integer integer) throws Exception {

Log.e("consumer", integer+"");

}

});

上边注释已经写的很清楚了！   
**range()方法，第一个参数为开始值，第二个参数为数量，所以别搞错了，以为第二个参数为结束值；filter()方法用于对数据进行过滤；take(n)方法用于取前n个值。**

# 在Android中的使用

RxJava在**[Android](http://lib.csdn.net/base/android" \o "Android知识库" \t "http://blog.csdn.net/aiynmimi/article/details/_blank)**中的使用，主要就体现在**异步**这一点。对应RxJava，RxAndroid也已经到2.x版本。   
我在上一篇博客中也提到过，**涉及两个比较核心的方法subscribeOn和observeOn这两个方法都传入一个Scheduler对象，subscribeOn指定发射事件的线程，observeOn指定消费事件的线程。**   
在2.x的API中仍然支持主要的默认scheduler: computation, io, newThread 和 trampoline，可以通过io.reactivex.schedulers.Schedulers这个实用的工具类来调度。

我们在android中主要就使用下边这两个就够了：   
①Schedulers.io(): I/O 操作（读写文件、读写**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/aiynmimi/article/details/_blank)**、网络信息交互等）所使用的 Scheduler。行为模式和 newThread() 差不多，区别在于 io() 的内部实现是是用一个无数量上限的线程池，可以重用空闲的线程，因此多数情况下 io() 比 newThread() 更有效率。不要把计算工作放在 io() 中，可以避免创建不必要的线程。   
②AndroidSchedulers.mainThread()，它指定的操作将在 Android 主线程运行。

这里一个最简单的例子：

Flowable.just("Hello,I am China!")

.subscribeOn(Schedulers.io())

.observeOn(AndroidSchedulers.mainThread())

.subscribe(subscriber)

所以在Android中创建Flowable时，即发射数据源的时候的耗时操作，可以指定在io()线程中，得到数据后，更新UI可以指定在mainThread()中。

当然现在最经典的就是RxAndroid和Retrofit的结合使用了：   
这里有一个比较牛逼的写法总结：   
[RxJava 与 Retrofit 结合的最佳实践](http://gank.io/post/56e80c2c677659311bed9841)   
这篇文章是基于1.x写的，不过在2.x中用法大同小异。   
另外需要注意的问题就是，retrofit现在还未支持RxJava2.x，不过不用担心，jake大神已经给我们写好了适配器：

compile 'com.jakewharton.retrofit:retrofit2-rxjava2-adapter:1.0.0'

在gradle中添加依赖即可！   
然后在创建Retrofit对象时，这样写：

Retrofit retrofit = new Retrofit.Builder()

.baseUrl(BASE\_URL)

.addConverterFactory(GsonConverterFactory.create())

.addCallAdapterFactory(RxJava2CallAdapterFactory.create())//1.X为RxJavaCallAdapterFactory

.build();

就可以在Retrofit2中尽情使用RxJava2了！