**[从零开始的RxJava2.0教程(三)响应式的好处](http://blog.csdn.net/qq_35064774/article/details/53065360)**

# 2. 错误处理

到目前为止，我都没怎么介绍 onComplete 和 onError 方法。这两个方法用来通知订阅者，数据发送完成或出现错误。

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

@Override

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

e.onNext("exception:" + (1 / 0));

e.onComplete();

}

}, BackpressureStrategy.BUFFER)

.subscribe(new Subscriber<String>() {

@Override

public void onSubscribe(Subscription s) {

s.request(1);

}

@Override

public void onNext(String s) {

System.out.println(s);

}

@Override

public void onError(Throwable t) {

t.printStackTrace();

System.out.println("onError");

}

@Override

public void onComplete() {

System.out.println("on complete");

}

});

上面的代码中，发射数据时，做了一个(1 / 0)的运算，但这明显是会抛出除零异常的。所以，上述代码最后会打印 onError。   
而如果改成(1 / 1)，则打印的是 exception:1 和 on complete。

这样的设计有以下几个优点:

只要发生错误，onError()一定会被调用。   
这极大的简化了错误处理。只需要在一个地方处理错误即可以。

操作符不需要处理异常。   
将异常处理交给订阅者来做，一旦有调用链中有一个抛出了异常，就会直接执行onError()方法，停止数据传送。

你能够知道什么时候订阅者已经接收了全部的数据。

# 3. 调度器

假设你编写的 **[Android](http://lib.csdn.net/base/android" \o "Android知识库" \t "http://blog.csdn.net/qq_35064774/article/details/_blank)** App 需要从网络请求数据。网络请求是耗时的操作，因此你不得不在子线程中加载数据。那么问题来了！

在Android中写多线程不是一件容易的事，尤其是嵌套数据获取，比如要获取用的资料，其中有一项是头像，但得到的一般是头像的url地址，你还需要在资料获取成功后，在发送一次请求，这样就导致代码看起来很乱。

使用RxJava你可以随意的切换线程。

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

@Override

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

e.onNext("将会在3秒后显示");

SystemClock.sleep(3000);

e.onNext("ittianyu");

e.onComplete();

}

}, BackpressureStrategy.BUFFER)

.subscribeOn(Schedulers.io())

.observeOn(AndroidSchedulers.mainThread())

.subscribe(new Consumer<String>() {

@Override

public void accept(String s) throws Exception {

Toast.makeText(RxJava2Activity.this, s, Toast.LENGTH\_SHORT).show();

}

});

上述代码中，Flowable总共发射了两个数据，但中间延时了3秒，如果在主线程中延时，那将会导致UI卡顿，这是绝对不能容忍的。   
所以在订阅之前，我们使用 subscribeOn(Schedulers.io()) 指定了发送数据是在io线程(某个子线程)，然后调用observeOn(AndroidSchedulers.mainThread()) 指定订阅者在主线程执行。

对了，要使用 AndroidSchedulers 还需要引入 RxAndroid:

compile 'io.reactivex.rxjava2:rxandroid:2.0.0'