**[从零开始的RxJava2.0教程(一)基础](http://blog.csdn.net/qq_35064774/article/details/53057332)**

# 2. 基础

RxJava 2.0 最核心的是Publisher和Subscriber。Publisher可以发出一系列的事件，而Subscriber负责和处理这些事件。

平常用得最多的Publisher是Flowable，它支持背压。

要使用RxJava 2，你需要先引入相应的jar包。

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

compile 'org.reactivestreams:reactive-streams:1.0.0'

注意，和1.x中不一样，2.0有一个依赖包。

# 3. Hello RxJava 2

创建一个Flowable对象很简单，直接调用Flowable.create即可。

// create a flowable

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

@Override

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

e.onNext("hello RxJava 2");

e.onComplete();

}

}, BackpressureStrategy.BUFFER);

上述代码仅仅是发射了一个字符串"hello RxJava 2"。   
下面我们还需要创建一个Subscriber。

// create

Subscriber subscriber = new Subscriber<String>() {

@Override

public void onSubscribe(Subscription s) {

System.out.println("onSubscribe");

s.request(Long.MAX\_VALUE);

}

@Override

public void onNext(String s) {

System.out.println(s);

}

@Override

public void onError(Throwable t) {

}

@Override

public void onComplete() {

System.out.println("onComplete");

}

};

需要注意的是，在onSubscribe中，我们需要调用request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。如果你不调用request，Subscriber的onNext和onComplete方法将不会被调用。

onNext方法里面传入的参数就是Flowable中发射出来的。

为了让”发射器”和”接收器”工作起来，我们还需要把他们组装在一起。

flowable.subscribe(subscriber);

一旦 flowable.subscribe 被执行，就会分别打印 hello RxJava 2 和 onComplete。

# 4. 更简洁的代码

上面一大串代码仅仅就达到了打印两个字符串的效果，你可能会想:“RxJava只不过是把事情变复杂了”。   
或许是这样的，但RxJava也提供了很多便利的方法来做这种简单的事情。

Flowable<String> flowable = Flowable.just("hello RxJava 2")；

我们可以直接调用Flowable.just创建一个发射字符串的”发射器”。

而对于 Subscriber 来说，我们目前仅仅关心onNext方法。所以可以简写成下面这样。

Consumer consumer = new Consumer<String>() {

@Override

public void accept(String s) throws Exception {

System.out.println(s);

}

};

当然这只是一个 Consumer，但 subscribe 方法提供了重载，让我们可以只传入一个Consumer。

所以订阅代码是这样的。

flowable.subscribe(consumer);

如果省去单独定义变量，最终可以写成下面这样。

Flowable.just("hello RxJava 2")

.subscribe(new Consumer<String>() {

@Override

public void accept(String s) throws Exception {

System.out.println(s);

}

});

# 5. 变换

让我们做一些更有意思的事情把！   
比如我想在hello RxJava 2后面加上我的签名，你可能会去修改Flowable传入的参数:

Flowable.just("hello RxJava 2 -ittianyu")

.subscribe(s -> System.out.println(s));

这当然是可以的，但是这样做，就导致所有的接收者都会受到影响。我只想针对某个订阅者做修改，那么你可能会写出这样的代码:

Flowable.just("hello RxJava 2")

.subscribe(s -> System.out.println(s + " -ittianyu"));

这样的方式仍然不让人满意，因为我希望订阅者做的事越少越好，因为一般来说，订阅者都是在主线程中执行的。这个时候我们就可以利用操作符在数据传递的途中进行变换。

# 6. 操作符

操作符是为了解决 Flowable 对象变换问题而设计的，操作符可以在传递的途中对数据进行修改。   
RxJava提供了很多实用的操作符。比如 map 操作符，可以把一个事件转换成另一个事件。

Flowable.just("map")

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

@Override

public String apply(String s) throws Exception {

return s + " -ittianyu";

}

})

.subscribe(new Consumer<String>() {

@Override

public void accept(String s) throws Exception {

System.out.println(s);

}

});

上面代码中， map 是把传递过来的结果末尾加上了签名，然后在传递给了订阅者。   
是不是觉得神奇？   
map的作用就变换 Flowable 然后返回一个指定类型的 Flowable 对象。

# 7. map 操作符进阶

map 操作符更神奇的地方是，你可以返回任意类型的 Flowable，也就是说你可以使用 map 操作符发射一个新的数据类型的 Flowable 对象。

比如上面的例子，订阅者想要得到字符串的hashcode。

Flowable.just("map1")

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

@Override

public Integer apply(String s) throws Exception {

return s.hashCode();

}

})

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

@Override

public String apply(Integer integer) throws Exception {

return integer.toString();

}

})

.subscribe(new Consumer<String>() {

@Override

public void accept(String s) throws Exception {

System.out.println(s);

}

});

这里用了两个map，一个是把字符串转成hashcode，另一个是把hashcode 转成字符串。

# 8. 总结

1. 你可以在 Publisher 中查询数据库或者从网络上获取数据，然后在 Subscriber 中显示。
2. Publisher 不只有一种，事实上 Flowable 和 Processor 所有的子类都属于 Publisher。
3. 在数据发射途中，你可以利用操作符对数据进行变换。