# **RxJava2 vs RxJava1**

RxJava2相比RxJava1，它的改动还是很大的：

## **Observable and Flowable**

在前一个版本里backpressure被集成到了Observable中，官方也提供了很多方法让我们来处理backpressure问题。但是有一些特殊的场景根本无法用其来解决，最常见的例如UI事件。而不处理backpressure有可能导致MissingBackpressureException的出现。

关于backpressure的概念可以看一下[RxJava中backpressure这个概念的理解](http://www.dundunwen.com/article/275b1d92-f9da-4bb8-b111-3aa8a6ace245.html" \t "http://www.jianshu.com/p/_blank)

为了解决这个问题，在RxJava2里，引入了Flowable这个类：Observable不包含backpressure处理，而Flowable包含。

例如:

Flowable<Long> flowable =

Flowable.create((FlowableOnSubscribe<Long>) e -> {

Observable.interval(10, TimeUnit.MILLISECONDS)

.take(Integer.MAX\_VALUE)

.subscribe(e::onNext);

}, FlowableEmitter.BackpressureMode.DROP);

Observable<Long> observable =

Observable.create((ObservableOnSubscribe<Long>) e -> {

Observable.interval(10, TimeUnit.MILLISECONDS)

.take(Integer.MAX\_VALUE)

.subscribe(e::onNext);

});

两个对象都以10毫秒一次派发数据，假设订阅他们的方法都是:

i -> {

Thread.sleep(100);

Log.v("TEST", "out : " + i);

}

以100毫秒一次消费数据，消费数据的效率是生产的1/10。那么

对于observable  
他会按照0,1,2,3,4...的顺序依次消费，并输出log，而没有消费的数据将会都存在内存中。如果在RxJava1中，内存数据超过128个时将会抛出MissingBackpressureException错误；而在RxJava2中并不会报错，数据会一直放到内存中，直到发生OutOfMemoryError。

对于flowable, 在创建时我们设定了FlowableEmitter.BackpressureMode.DROP，一开始他会输出0,1,2,3....127但之后会忽然跳跃到966,967,968 ...。中间的部分数据由于缓存不了，被抛弃掉了。

## **Single**

和Observable，Flowable一样会发送数据，不同的是订阅后只能接受到一次:

Single<Long> single = Single.just(1l);

single.subscribe(new SingleObserver<Long>() {

@Override

public void onSubscribe(Disposable d) {

}

@Override

public void onSuccess(Long value) {

// 和onNext是一样的

}

@Override

public void onError(Throwable e) {

}

});

普通Observable可以使用toSingle转换:Observable.just(1).toSingle()

## **Completable**

与Single类似，只能接受到完成(onComplete)和错误(onError)

同样也可以由普通的Observable转换而来:Observable.just(1).toCompletable()

可订阅的对象在RxJava1中只有Observable一种，之前我们经常会直接把数据源称作Observable。而在RxJava2中扩充成了4种，因此在之后还是把他们统称为数据源为宜。

## **Base reactive interfaces**

和Flowable的接口Publisher类似，Observable、Single、Completable也有类似的基类

interface ObservableSource<T> {

void subscribe(Observer<? super T> observer);

}

interface SingleSource<T> {

void subscribe(SingleObserver<? super T> observer);

}

interface CompletableSource {

void subscribe(CompletableObserver observer);

}

因此许多操作符接受的参数从以前的具体对象，变成了现在的接口:

Flowable<R> flatMap(

Function<? super T, ? extends Publisher<? extends R>> mapper

);

Observable<R> flatMap(

Function<? super T, ? extends ObservableSource<? extends R>> mapper

);

------// 以前

Observable<R> flatMap(Func1<? super T, ? extends Observable<? extends R>> func) {

由于接收的都是接口，在使用其他遵循Reactive-Streams设计的第三方库的时候，就不需要把他自定义的Flowable转换成标准Flowable了。

## **Subjects and Processors**

io.reactivex.subjects.AsyncSubject,  
io.reactivex.subjects.BehaviorSubject,  
io.reactivex.subjects.PublishSubject,  
io.reactivex.subjects.ReplaySubject,  
io.reactivex.subjects.UnicastSubject

在RxJava2中依然存在，但现在他们不支持backpressure。新出现的

io.reactivex.processors.AsyncProcessor,  
io.reactivex.processors.BehaviorProcessor,  
io.reactivex.processors.PublishProcessor,  
io.reactivex.processors.ReplayProcessor  
io.reactivex.processors.UnicastProcessor

支持backpressure

## **Other classes**

rx.observables.ConnectableObservable变成了io.reactivex.observables.ConnectableObservable<T>和io.reactivex.flowables.ConnectableFlowable<T>

类似的还有rx.observables.GroupedObservable。

## **Functional interfaces**

需要注意的一点是，现在RxJava2的接口方法里加上了throws Exception:

ublic interface Consumer<T> {

void accept(T t) throws Exception;

}

意味着在这些方法里调用一些会发生异常的方法不需要try-catch了

## **Actions**

另外大部分接口方法都按照Java8的接口方法名进行了相应的修改，比如上面那个Consumer<T>接口原来叫Action1<T>，而Action2<T>改名成了BiConsumer

Action3-Action9被删掉了，大概因为没人用。。

## **Functions**

同上，基本就是名字的修改和不常用类的删除

## **Subscriber**

RxJava1里Subscriber只是一个空接口，在新版里Subscriber被赋予了更多的作用，有几个实现类可以供我们使用，例如

ResourceSubscriber<Integer> subscriber = new ResourceSubscriber<Integer>() {

@Override

public void onStart() {

request(Long.MAX\_VALUE);

}

@Override

public void onNext(Integer t) {

System.out.println(t);

}

@Override

public void onError(Throwable t) {

t.printStackTrace();

}

@Override

public void onComplete() {

System.out.println("Done");

}

};

request()方法可以控制当前subscriber需要接收几个事件。而且，还可以调用subscriber.dispose()来断开对信号的监听。

同时，onCompleted方法改成了onComplete。意味着完成时调用这个方法，而不是完成后。由于Subscription被改掉了(下面会讲到)，如果需要类似以前CompositeSubscription的用法，可以使用:

CompositeDisposable composite2 = new CompositeDisposable();

composite2.add(Flowable.range(1, 5).subscribeWith(subscriber));

注意这里需要使用subscribeWith而不是subscribe，因为subscribe方法现在返回void

## **Subscription**

在RxJava1里，Subscription起到的是订阅桥梁的作用。在2中，由于Subscription本身和Reactive-Streams里的另外一个同名概念冲突。因此把原本的Subscription改名成了Disposable。

除了上一节里subscribe(Subscriber )方法返回void，其他名为subscribe的方法都返回Disposable

相应的，

* CompositeSubscription 改名成了 CompositeDisposable
* SerialSubscription 和 MultipleAssignmentSubscription被合并到了SerialDisposable里。 set() 方法会处理掉旧的值，而replace()方法不会。
* RefCountSubscription被移除了。

## **Backpressure**

在第一节Observable and Flowable里已经说到了这个问题，在2中，Observable将不会处理backpressure，也就不会发生MissingBackpressureException问题，但内存仍然会缓存多余的数据。

而在使用Flowable时，如果配置Backpressure有问题，那么MissingBackpressureException依然存在

## **Schedulers**

RxJava2里仍然包含了computation, io, newThread 和 trampoline这些默认线程调度。而immediate被移除了，因为他经常被人错误使用。同时Schedulers.test也被移除了。

## **Entering the reactive world**

将普通方法转换成RxJava的数据源，在RxJava1中，提供了Observable.create()方法，但是这个方法过于强大，但使用时需要注意的东西太多经常会发生错误。

因此在RxJava2中，把原来的fromAsync重命名成了create，fromAsync是一个和create类似但更为简单和安全的方法。这样大部分旧代码都能够继续使用。

## **Leaving the reactive world**

之前如果想把数据源转换成普通的数据对象，需要先转换成BlockingObservable。而在2中，可以调用blockingXXX方法直接把数据源转换成对象:

List<Integer> list = Flowable.range(1, 100).toList().blockingFirst();

有一点需要特别注意，在RxJava2里，不建议在Subscriber里抛出错误，这意味着下面的代码可能有一天就不能继续运行了：

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

@Override

public void onSubscribe(Subscription s) {

s.request(Long.MAX\_VALUE);

}

public void onNext(Integer t) {

if (t == 1) {

throw new IllegalArgumentException();

}

}

public void onError(Throwable e) {

if (e instanceof IllegalArgumentException) {

throw new UnsupportedOperationException();

}

}

public void onComplete() {

throw new NoSuchElementException();

}

};

Flowable.just(1).subscribe(subscriber);

由于上面类似的代码实际中出现得很多，因此在2中提供了safeSubscribe方法，使用它就可以继续在subscriber里抛出错误。

当然，你可以绕过subscribe(subscriber)这个方法，使用类似:

Flowable.just(1).subscribe(subscriber::onNext, subscriber::onError, subscriber::onComplete);

这样的方法，之前的代码仍然可以继续throw错误。

## **Operator differences**

操作符的改动不大，大部分是扩充了参数数量。  
或者是加入prefetch代表可以加入预置数据。

## **总结**

可以明显的看到，RxJava2最大的改动就是对于backpressure的处理，为此将原来的Observable拆分成了新的Observable和Flowable，同时其他相关部分也同时进行了拆分。