RxJava2.0 操作符之 -- 异常操作符

作者：hiquanta
原文链接：https://ld246.com/article/1500604893933
来源网站：链滴

onErrorReturn

让Observable遇到错误时发射一个特殊的项并且正常终止

```
Observable.create(new ObservableOnSubscribe() {
    public void subscribe(@NonNull ObservableEmitter e) throws Exception {
            e.onNext(1);

 throw new RuntimeException("error");

  }
}).onErrorReturn(new Function, Integer>() {
    public Integer apply(@NonNull Throwable throwable) throws Exception {
        return 5;
  }
}).subscribe(RxUtils.getObserver());
```

```
    onSubscribe
    Thread:Thread[main,5,main]
    onNext:1
    Thread:Thread[main,5,main]
    onNext:5
    Thread:Thread[main,5,main]
    onComplete
    Thread:Thread[main,5,main]
```

onErrorResumeNext

让Observable在遇到错误时开始发射第二个Observable的据序列。

```
    Observable.create(new ObservableOnSubscribe() {
        public void subscribe(@NonNull ObservableEmitter e) throws Exception {
            e.onNext(3);
     throw new RuntimeException("error");
      }
    }).onErrorResumeNext(new Function, ObservableSourceextends Integer>>() {
        public ObservableSourceextends Integer> apply(@NonNull Throwable throwable) throw
 Exception {
            return Observable.just(5,6,7);
      }
    }).subscribe(RxUtils.getObserver());
```

```
    onSubscribe
    Thread:Thread[main,5,main]
    onNext:3
    Thread:Thread[main,5,main]
    onNext:5
    Thread:Thread[main,5,main]
    onNext:6
    Thread:Thread[main,5,main]
```

```
onNext:7
Thread:Thread[main,5,main]
onComplete
Thread:Thread[main,5,main]
```

onExceptionResumeNext

和onErrorResumeNext类似，onExceptionResumeNex方法返回一个镜像原有Observable行为的新Observable，使用一个备用的Observable，不同的是，如果onError收到Throwable不是## 一个Exception，它会将错误传递给观者的onError方法，不会使用备用的Observable。

```
Observable.create(new ObservableOnSubscribe() {
    public void subscribe(@NonNull ObservableEmitter e) throws Exception {
        e.onNext(3);
 throw new RuntimeException("error");
  }
}).onExceptionResumeNext(Observable.just(5,6,7)).subscribe(RxUtils.getObserver());

onSubscribe
Thread:Thread[main,5,main]
onNext:3
Thread:Thread[main,5,main]
onNext:5
Thread:Thread[main,5,main]
onNext:6
Thread:Thread[main,5,main]
onNext:7
Thread:Thread[main,5,main]
onComplete
Thread:Thread[main,5,main]
```

retry

如果原始Observable遇到错误，重新订阅它期望它能正常终止

```
Observable.create(new ObservableOnSubscribe() {
    public void subscribe(@NonNull ObservableEmitter e) throws Exception {
        try {
            for (int i = 0; i < 10; i++) {
                if (i == 4) {
                    throw new Exception(
                            "this is number 4 error!  ");
  }
                e.onNext(i);
  }
            e.onComplete();
  } catch (Throwable ex) {
            e.onError(ex);
  }
    }
```

```
}).retry(2).subscribe(RxUtils.getObserver());

onSubscribe
Thread:Thread[main,5,main]
onNext:0
Thread:Thread[main,5,main]
onNext:1
Thread:Thread[main,5,main]
onNext:2
Thread:Thread[main,5,main]
onNext:3
Thread:Thread[main,5,main]
onNext:0
Thread:Thread[main,5,main]
onNext:1
Thread:Thread[main,5,main]
onNext:2
Thread:Thread[main,5,main]
onNext:3
Thread:Thread[main,5,main]
onNext:0
Thread:Thread[main,5,main]
onNext:1
Thread:Thread[main,5,main]
onNext:2
Thread:Thread[main,5,main]
onNext:3
Thread:Thread[main,5,main]
onError:java.lang.Exception: this is number 4 error!
Thread:Thread[main,5,main]
```

retryWhen

retryWhen和retry类似，区别是，retryWhen将onError的Throwable传递给一个函数，这个函数产生另一个Observable，retryWhen观察它的结果再决定是不是要重新订阅原的

Observable，如果这个Observable发射了一项数据，它就新订阅，如果这个Observable发射的是onError通知，它就这个通知传递给观察者然后终止。

```
Observable.create(new ObservableOnSubscribe() {
    public void subscribe(@NonNull ObservableEmitter e) throws Exception {
        try {
            for (int i = 0; i < 10; i++) {
                if (i == 4) {
                    throw new Exception(
                            "this is number 4 error!  ");
  }
                e.onNext(i);
  }
```

```
            e.onComplete();
  } catch (Throwable ex) {
            e.onError(ex);
  }
    }
}).retryWhen(new Function, ObservableSource>() {
       public ObservableSource apply(@NonNull Observable throwableObservable) throws Exc
ption {
           return Observable.just(12,13);
  }
   }).subscribe(RxUtils.getObserver());
 try {
        Thread.sleep(Integer.MAX_VALUE);
  } catch (InterruptedException e1) {
        // TODO Auto-generated catch block
  e1.printStackTrace();
  }

onSubscribe
Thread:Thread[main,5,main]
onNext:0
Thread:Thread[main,5,main]
onNext:1
Thread:Thread[main,5,main]
onNext:2
Thread:Thread[main,5,main]
onNext:3
Thread:Thread[main,5,main]
onNext:0
Thread:Thread[main,5,main]
onNext:1
Thread:Thread[main,5,main]
onNext:2
Thread:Thread[main,5,main]
onNext:3
Thread:Thread[main,5,main]
onComplete
Thread:Thread[main,5,main]

Process finished with exit code 1
```