链滴

# RxJava2.0 操作符之 -- 辅助操作符

作者：hiquanta

原文链接：https://ld246.com/article/1500433590953

来源网站：链滴

# Delay

## 延迟一段指定的时间再发射来自Observable的发射物

```
Observable.just(1, 2, 3, 4).delay(5, TimeUnit.SECONDS).subscribe(RxUtils.getObserver());
try {
    Thread.sleep(Integer.MAX_VALUE);
} catch (InterruptedException e1) {
    e1.printStackTrace();
}
```

# Do

## 注册一个动作作为原始Observable生命周期事件的一种占位,相当于注册回调

## 其中RxJava中的实现为doXXX 很简单 看名字就知道回调时和怎么用

```
  //doOnEach
//        Observable.just(1,2,3,4,5).doOnEach(RxUtils.getObserver()).subscribe();
  /**
 * onNext:1 Thread:Thread[main,5,main] onNext:2 Thread:Thread[main,5,main] onNext:3 Thre
d:Thread[main,5,main] onNext:4 Thread:Thread[main,5,main] onNext:5 Thread:Thread[main,5
main] onComplete Thread:Thread[main,5,main] */  //doOnNext
//        Observable.just(1,2,3,4,5).doOnNext(new Consumer() {
//            public void accept(@NonNull Integer integer) throws Exception {
//                if(integer>3)
//                    throw new RuntimeException("error");
//            }
//        }).subscribe(RxUtils.getObserver());
  /**
 * onSubscribe Thread:Thread[main,5,main] onNext:1 Thread:Thread[main,5,main] onNext:2 T
read:Thread[main,5,main] onNext:3 Thread:Thread[main,5,main] onError:java.lang.RuntimeExc
ption: error Thread:Thread[main,5,main] */  //doOnSubscribe
//        Observable observable = Observable.just(1, 2, 3, 4, 5, 6).doOnSubscribe(new Consumer
) {
//            public void accept(@NonNull Disposable disposable) throws Exception {
//                System.out.println("published");
//            }
//        });
//        observable.subscribe(RxUtils.getObserver());
//        observable.subscribe(RxUtils.getObserver());

  /**
 * published onSubscribe Thread:Thread[main,5,main] onNext:1 Thread:Thread[main,5,main]
nNext:2 Thread:Thread[main,5,main] onNext:3 Thread:Thread[main,5,main] onNext:4 Thread:T
read[main,5,main] onNext:5 Thread:Thread[main,5,main] onNext:6 Thread:Thread[main,5,main
 onComplete Thread:Thread[main,5,main] published onSubscribe Thread:Thread[main,5,main]
onNext:1 Thread:Thread[main,5,main] onNext:2 Thread:Thread[main,5,main] onNext:3 Thread:
```

```
hread[main,5,main] onNext:4 Thread:Thread[main,5,main] onNext:5 Thread:Thread[main,5,ma
n] onNext:6 Thread:Thread[main,5,main] onComplete Thread:Thread[main,5,main] */  //doOn
nsubscribe 2.0 has removed instead of
  DisposableSubscriber disposableSubscriber = new DisposableSubscriber() {
          public void onNext(Long along) {
              System.out.println(along);
  }

          public void onError(Throwable throwable) {
              System.out.println(throwable);
  }

          public void onComplete() {
              System.out.println("onComplete");
  }
      };

  Flowable.interval(1000, TimeUnit.MILLISECONDS).doOnCancel(new Action() {
          public void run() throws Exception {
              System.out.println("doOnCancel");
  }
      }).subscribe(disposableSubscriber);

  disposableSubscriber.dispose();
```

# Materialize/Dematerialize

**Materialize将数据项和事件通知都当做数据项发射，Dematerialize刚好相反。**

**Dematerialize操作符是Materialize的逆向过程，它将Materialize转换的结果还原成它原本的形式。**

```
    Observable.just(1,2,3,4).materialize().subscribe(RxUtils.>getObserver());
    Observable.just(1,2,3,4).materialize().dematerialize().subscribe(RxUtils.getObserver());
```

# ObserveOn_SubscribOn

# ObserveOn:指定一个观察者在哪个调度器上观察这个Observable

# SubscribOn:用来指定Observable在哪个线程上运行

```
  //  Observable.just(1,2,3,4,5).observeOn( Schedulers.newThread()).subscribeOn(Schedulers.c
mputation()).subscribe(RxUtils.getObserver());
  Observable.just(1,2,3,4,5).subscribeOn( Schedulers.newThread()).subscribeOn(Schedulers.co
putation()).subscribe(RxUtils.getObserver());
 try {
      Thread.sleep(Integer.MAX_VALUE);
  } catch (InterruptedException e1) {
```

```
        e1.printStackTrace();
    }
```

## Serialize

## 强制一个Observable连续调用并保证行为正确

## 这个说明目前没有从文档中找到能够说明此功能的代码，自的测试代码也有点问题

## 如果你又好的方法，可以留言告诉我

## Subscribe

## 操作来自Observable的发射物和通知

## 一直在用这个方法，很简单，就不贴代码了

## 1.x 2.x 方法差异如下

**[RxJava 1.x BlockingObservable.forEach forEach subscribe](http://reactivex.io/documentation/operators/subscribe.html#collapseRxJava 1.x)**

**[RxJava 2.x blockingForEach blockingSubscribe forEachWhile safeSubscribe subscribe](http://reactivex.io/documentation/operators/subscribe.html#collapseRxJava 2.x)**

## TimeInterval

## 将一个发射数据的Observable转换为发射那些数据发射时间隔的对象

```
Observable.interval(3, TimeUnit.SECONDS)
            .timeInterval()
            .subscribe(RxUtils.>getObserver());
 try {
        Thread.sleep(Integer.MAX_VALUE);
    } catch (InterruptedException e1) {
        e1.printStackTrace();
    }
}

onSubscribe
Thread:Thread[main,5,main]
onNext:Timed[time=3007, unit=MILLISECONDS, value=0]
Thread:Thread[RxComputationThreadPool-1,5,main]
```

```
onNext:Timed[time=3000, unit=MILLISECONDS, value=1]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=3001, unit=MILLISECONDS, value=2]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=3000, unit=MILLISECONDS, value=3]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=3000, unit=MILLISECONDS, value=4]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=2999, unit=MILLISECONDS, value=5]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=3000, unit=MILLISECONDS, value=6]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=3001, unit=MILLISECONDS, value=7]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=3000, unit=MILLISECONDS, value=8]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=3000, unit=MILLISECONDS, value=9]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=3000, unit=MILLISECONDS, value=10]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=3000, unit=MILLISECONDS, value=11]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=2999, unit=MILLISECONDS, value=12]
```

# Timeout

## 对原始Observable的一个镜像，如果过了一个指定的时长仍有发射数据，它会发一个错误通知

```
Observable.interval(3, TimeUnit.SECONDS)
        .timeout(2,TimeUnit.SECONDS)
        .subscribe(RxUtils.getObserver());
try {
    Thread.sleep(Integer.MAX_VALUE);
} catch (InterruptedException e1) {
    e1.printStackTrace();
}

onSubscribe
Thread:Thread[main,5,main]
onError:java.util.concurrent.TimeoutException
Thread:Thread[RxComputationThreadPool-1,5,main]
```

# Timestamp

## 给Observable发射的数据项附加一个时间戳

```
Observable.interval(3, TimeUnit.SECONDS)
        .timestamp()
        .subscribe(RxUtils.>getObserver());
try {
    Thread.sleep(Integer.MAX_VALUE);
```

```
 } catch (InterruptedException e1) {
     e1.printStackTrace();
 }

onSubscribe
Thread:Thread[main,5,main]
onNext:Timed[time=1500432643307, unit=MILLISECONDS, value=0]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=1500432646306, unit=MILLISECONDS, value=1]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=1500432649306, unit=MILLISECONDS, value=2]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=1500432652306, unit=MILLISECONDS, value=3]
Thread:Thread[RxComputationThreadPool-1,5,main]
onNext:Timed[time=1500432655307, unit=MILLISECONDS, value=4]
```

# Using

# 创建一个只在Observable生命周期内存在的一次性资源

```
Observable.interval(3, TimeUnit.SECONDS)
        .using(new Callable() {
            public Long call() throws Exception {
                return Long.valueOf(2);
 }
        }, new Function, ObservableSource>() {
            public ObservableSource apply(@NonNull Long aLong) throws Exception {
                return new ObservableSource() {
                    public void subscribe(@NonNull Observersuper Long> observer) {
                        observer.onNext(Long.valueOf(2));
 }
                };
 }
        }, new Consumer() {
            public void accept(@NonNull Long aLong) throws Exception {
                    System.out.println(aLong);
 }
        })
        .subscribe(RxUtils.getObserver());
try {
    Thread.sleep(Integer.MAX_VALUE);
} catch (InterruptedException e1) {
    e1.printStackTrace();
}

    onNext:2
   Thread:Thread[main,5,main]
```