链滴

RxJava 操作符之 -- 变换操作符

作者：hiquanta
原文链接：https://ld246.com/article/1490866021238
来源网站：链滴

buffer

定期收集Observable的数据放进一个数据包裹，然后发射这些数据包裹，而不是一次发射一个值。

<!--more-->

```
Observable.range(1,5).buffer(2).subscribe(new Observer<List<Integer>>() {
            @Override
            public void onCompleted() {
                System.out.println("onCompleted");
            }

            @Override
            public void onError(Throwable e) {
                System.out.println("onError");
            }

            @Override
            public void onNext(List<Integer> i) {
                System.out.println("onNext"+i);
            }
        });
```

结果

onNext[1, 2]

onNext[3, 4]

onNext[5]

onCompleted

flatMap

FlatMap将一个发射数据的Observable变换为多个Observables，然后将它们发射的数据合并后放进一个单独的Observable

```
Observable.just("大","小")
            .flatMap(new Func1<String, Observable<String>>() {

                @Override
                public Observable<String> call(String t) {
                    return Observable.just(t+"sb");
                }
            }).subscribe(new Action1<String>() {

                @Override
                public void call(String t) {
                    System.out.println(t);
                }
            });
```

结果

```
大sb
小sb
```

map

对Observable发射的每一项数据应用一个函数，执行变换操作

```
Observable.just("小","大")
        .map(new Func1<String, String>() {

            @Override
            public String call(String t) {

                return t+"sb";
            }
        }).subscribe(new Action1<String>() {

            @Override
            public void call(String t) {
                System.out.println(t);

            }
        });
```

结果

```
小sb
大sb
```

groupBy

将一个Observable分拆为一些Observables集合，它们中的每一个发射原始Observable的一个子序列

```
Observable.interval(1, TimeUnit.SECONDS).take(10).groupBy(new Func1<Long, Long>() {

      @Override
      public Long call(Long t) {
        //按照key为0,1,2分为3组
        return t % 3;
      }
    }).subscribe(new Action1<GroupedObservable<Long, Long>>() {

      @Override
      public void call(GroupedObservable<Long, Long> result) {
        result.subscribe(new Action1<Long>() {

          @Override
          public void call(Long t) {
            System.out.println("key:" + result.getKey() +", value:" + t);
```

```
        }
      });
    }
  });
    try {
      Thread.sleep(Integer.MAX_VALUE);
    } catch (InterruptedException e1) {
      e1.printStackTrace();
    }
```

结果

```
key:0, value:0
key:1, value:1
key:2, value:2
key:0, value:3
key:1, value:4
key:2, value:5
key:0, value:6
key:1, value:7
key:2, value:8
key:0, value:9
```

scan

连续地对数据序列的每一项应用一个函数，然后连续发射结果

```
Observable.just(1,2,3,4,5).scan(new Func2<Integer, Integer, Integer>() {

            @Override
            public Integer call(Integer t1, Integer t2) {
                // TODO Auto-generated method stub
                return t1+t2;
            }
        }).subscribe(new Subscriber<Integer>() {
            @Override
            public void onNext(Integer item) {
                System.out.println("Next: " + item);
            }

            @Override
            public void onError(Throwable error) {
                System.err.println("Error: " + error.getMessage());
            }

            @Override
            public void onCompleted() {
                System.out.println("Sequence complete.");
            }
        });
```

结果

```
Next: 1
Next: 3
Next: 6
Next: 10
Next: 15
Sequence complete.
```

window

定期将来自原始Observable的数据分解为一个Observable窗口，发射这些窗口，而不是每次发射一数据

```
Observable.interval(1, TimeUnit.SECONDS).take(12)
    .window(3,TimeUnit.SECONDS)
    .subscribe(new Action1<Observable<Long>>() {

      @Override
      public void call(Observable<Long> t) {
          System.out.println("subdivide begin.....");
                t.subscribe(new Action1<Long>() {
                    @Override
                    public void call(Long aLong) {
                        System.out.println("Next:" + aLong);
                    }
                });
      }
    });
    try {
      Thread.sleep(Integer.MAX_VALUE);
    } catch (InterruptedException e1) {
      e1.printStackTrace();
    }
```

结果

```
subdivide begin.....
Next:0
Next:1
subdivide begin.....
Next:2
Next:3
Next:4
subdivide begin.....
Next:5
Next:6
Next:7
subdivide begin.....
Next:8
Next:9
Next:10
subdivide begin.....
Next:11
```

代码