链滴

RxJava 操作符之 -- 过滤操作符

作者：hiquanta

原文链接：https://ld246.com/article/1490866172175

来源网站：链滴

debounce

仅在过了一段指定的时间还没发射数据时才发射一个数据

<!--more-->

```
Observable.create(new Observable.OnSubscribe<Integer>() {

        @Override
        public void call(Subscriber<? super Integer> t) {
            try {
                //产生结果的间隔时间分别为100、200、300...900毫秒
                for (int i = 1; i < 10; i++) {
                    t.onNext(i);
                    Thread.sleep(i * 100);
                }
                t.onCompleted();
            }catch(Exception e){
                t.onError(e);
            }
        }
    }).subscribeOn(Schedulers.newThread())
    .debounce(400,TimeUnit.MILLISECONDS)
    .subscribe(  new Action1<Integer>() {
        @Override
        public void call(Integer integer) {
            System.out.println("Next:" + integer);
        }
    }, new Action1<Throwable>() {
        @Override
        public void call(Throwable throwable) {
            System.out.println("Error:" + throwable.getMessage());
        }
    }, new Action0() {
        @Override
        public void call() {
            System.out.println("completed!");
        }
    });
     try {
            Thread.sleep(Integer.MAX_VALUE);
        } catch (InterruptedException e1) {
            e1.printStackTrace();
        }
```

结果

```
Next:5
Next:6
Next:7
Next:8
Next:9
```

completed!

distinct

抑制（过滤掉）重复的数据项

```
Observable.just(1, 2, 1, 1, 2, 3).distinct()
        .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: 2
Next: 3
Sequence complete.
```

elementAt

只发射第N项数据

```
Observable.just(1, 2, 3, 4, 5, 6).elementAt(2)
        .subscribe(new Action1<Integer>() {
            @Override
            public void call(Integer integer) {
                System.out.println("Next:" + integer);
            }
        }, new Action1<Throwable>() {
            @Override
            public void call(Throwable throwable) {
                System.out.println("Error:" + throwable.getMessage());
            }
        }, new Action0() {
            @Override
            public void call() {
                System.out.println("completed!");
            }
        });
```

结果

```
Next:3
completed!
```

filter

只发射通过了谓词测试的数据项

```
Observable.just(1, 2, 3, 4, 5).filter(new Func1<Integer, Boolean>() {
        @Override
        public Boolean call(Integer item) {
            return item < 4;
        }
    }).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: 2
Next: 3
Sequence complete.
```

first

只发射第一项（或者满足某个条件的第一项）数据

```
Observable.just(1, 2, 3).first().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
Sequence complete.
```

ignoreElements

不发射任何数据，只发射Observable的终止通知

```
Observable.just(1, 2, 3).ignoreElements()
            .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

```
Sequence complete.
```

last

只发射最后一项（或者满足某个条件的最后一项）数据

```
Observable.just(1, 2, 3).last().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: 3
Sequence complete.
```

sample

定期发射Observable最近发射的数据项

```
Observable.create(new Observable.OnSubscribe<Integer>() {
    @Override
    public void call(Subscriber<? super Integer> subscriber) {
      if (subscriber.isUnsubscribed())
        return;
      try {
        // 前8个数字产生的时间间隔为1秒，后一个间隔为3秒
        for (int i = 1; i < 9; i++) {
          subscriber.onNext(i);
          Thread.sleep(1000);
        }
        Thread.sleep(2000);
        subscriber.onNext(9);
        subscriber.onCompleted();
      } catch (Exception e) {
        subscriber.onError(e);
      }
    }
  }).subscribeOn(Schedulers.newThread())
      .sample(2200, TimeUnit.MILLISECONDS) // 采样间隔时间为2200毫秒
      .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.");
        }
```

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

结果

```
Next: 3
Next: 5
Next: 7
Next: 8
Sequence complete.
```

skip

抑制Observable发射的前N项数据

```
Observable.just(1,2,3,4,5,6,7).skip(3)
        .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: 4
Next: 5
Next: 6
Next: 7
Sequence complete.
```

skipLast

```
Observable.just(1, 2, 3, 4, 5, 6, 7).skipLast(3)
                .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: 2
Next: 3
Next: 4
Sequence complete.
```

take

只发射前面的N项数据

```
Observable.just(1, 2, 3, 4, 5, 6, 7, 8).take(4)
        .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: 2
Next: 3
Next: 4
Sequence complete.
```

takeFirst

takeFirst操作符类似于take操作符，同时也类似于first操作符，都是获取源Observable产生的结果列中符合指定条件的前一个或多个，与first操作符不同的是，first操作符如果获取不到数据，则会抛出NSuchElementException异常，而takeFirst则会返回一个空的Observable，该Observable只有onCompleted通知而没有onNext通知。

```
Observable.just(1,2,3,4,5,6,7).takeFirst(new Func1<Integer, Boolean>() {
            @Override
            public Boolean call(Integer integer) {
                //获取数值大于3的数据
                return integer>3;
            }
        })
         .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: 4
Sequence complete.
```

takeLast

发射Observable发射的最后N项数据

```
Observable.just(1, 2, 3, 4, 5, 6, 7).takeLast(2)
            .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: 6
Next: 7
Sequence complete.
```