链滴

# RxJava 操作符之 -- 结合操作符

作者：hiquanta
原文链接：https://ld246.com/article/1490866320900
来源网站：链滴

# And/Then/When

使用Pattern和Plan作为中介，将两个或多个Observable发射的数据集合并到一起

它们属于rxjava-joins模块，不是核心RxJava包的一部分。

<!--more-->

# combineLatest

当两个Observables中的任何一个发射了数据时，使用一个函数结合每个Observable发射的最近数据，并且基于这个函数的结果发射数据。

```
// 产生0,5,10,15,20数列
        Observable<Long> observable1 = Observable
                .interval(0, 1000, TimeUnit.MILLISECONDS)
                .map(new Func1<Long, Long>() {
                    @Override
                    public Long call(Long aLong) {
                        return aLong * 5;
                    }
                }).take(5);

        // 产生0,10,20,30,40数列
        Observable<Long> observable2 = Observable
                .interval(500, 1000, TimeUnit.MILLISECONDS)
                .map(new Func1<Long, Long>() {
                    @Override
                    public Long call(Long aLong) {
                        return aLong * 10;
                    }
                }).take(5);

        Observable.combineLatest(observable1, observable2,
                new Func2<Long, Long, Long>() {
                    @Override
                    public Long call(Long aLong, Long aLong2) {
                        return aLong + aLong2;
                    }
                }).subscribe(new Subscriber<Long>() {
            @Override
            public void onCompleted() {
                System.out.println("Sequence complete.");
            }

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

            @Override
            public void onNext(Long aLong) {
                System.out.println("Next: " + aLong);
```

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

## 结果

```
Next: 0
Next: 5
Next: 15
Next: 20
Next: 30
Next: 35
Next: 45
Next: 50
Next: 60
Sequence complete.
```

## join （不是太好理解）

任何时候，只要在另一个Observable发射的数据定义的时间窗口内，这个Observable发射了一条数，就结合两个Observable发射的数据

```
//产生0,5,10,15,20数列
    Observable<Long> observable1 = Observable.timer(0, 1000, TimeUnit.MILLISECONDS)
            .map(new Func1<Long, Long>() {
                @Override
                public Long call(Long aLong) {
                    return aLong * 5;
                }
            });

    //产生0,10,20,30,40数列
    Observable<Long> observable2 = Observable.timer(500, 1000, TimeUnit.MILLISECONDS)
            .map(new Func1<Long, Long>() {
                @Override
                public Long call(Long aLong) {
                    return aLong * 10;
                }
            });

    observable1.join(observable2, new Func1<Long, Observable<String>>() {

   @Override
   public Observable<String> call(Long t) {
     // TODO Auto-generated method stub
     return Observable.just(t.toString());
   }
```

```
    }, new Func1<Long, Observable<Long>>() {
        @Override
        public Observable<Long> call(Long aLong) {
            //使Observable延迟600毫秒执行
            return Observable.just(aLong).delay(600, TimeUnit.MILLISECONDS);
        }
    }, new Func2<Long, Long, String>() {
        @Override
        public String call(Long aLong, Long aLong2) {
            return aLong +":" +aLong2;
        }
    }).subscribe(new Subscriber<String>() {
        @Override
        public void onCompleted() {
            System.out.println("Sequence complete.");
        }

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

        @Override
        public void onNext(String aLong) {
            System.out.println("Next: " + aLong);
        }
    });
    try {
  Thread.sleep(Integer.MAX_VALUE);
 } catch (InterruptedException e1) {
  e1.printStackTrace();
 }
```

## 结果

```
Next: 5:0
Next: 10:10
Next: 15:20
Next: 20:30
Next: 25:40
Next: 30:50
Next: 35:60
Next: 40:70
Next: 45:80
Next: 50:90
Next: 55:100
Next: 60:110
Next: 65:120
Next: 70:130
Next: 75:140
...
```

如何理解RxJava中的join操作

# merge

合并多个Observables的发射物

```
// 产生0,5,10,15,20数列
        Observable<Long> observable1 = Observable.timer(0, 1000,
                TimeUnit.MILLISECONDS).map(new Func1<Long, Long>() {
            @Override
            public Long call(Long aLong) {
                return aLong * 5;
            }
        });

        // 产生0,10,20,30,40数列
        Observable<Long> observable2 = Observable.timer(500, 10000,
                TimeUnit.MILLISECONDS).map(new Func1<Long, Long>() {
            @Override
            public Long call(Long aLong) {
                return aLong * 10;
            }
        });

        Observable.merge(observable1, observable2).subscribe(
                new Subscriber<Long>() {
                    @Override
                    public void onNext(Long 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: 0
Next: 0
Next: 5
Next: 10
Next: 15
Next: 20
```

```
Next: 25
Next: 30
Next: 35
Next: 40
Next: 45
Next: 50
Next: 10
Next: 55
Next: 60
Next: 65
Next: 70
Next: 75
Next: 80
```

# StartWith

在数据序列的开头插入一条指定的项

```
Observable.just(10, 20, 30).startWith(2, 3, 4)
                .subscribe(new Subscriber<Integer>() {
                    @Override
                    public void onCompleted() {
                        System.out.println("Sequence complete.");
                    }

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

                    @Override
                    public void onNext(Integer value) {
                        System.out.println("Next:" + value);
                    }
                });
```

## 结果

```
Next:2
Next:3
Next:4
Next:10
Next:20
Next:30
Sequence complete.
```

# switchOnNext (不是太理解)

将一个发射多个Observables的Observable转换成另一个单独的Observable，后者发射那些Observables最近发射的数据项

```
// 每隔500毫秒产生一个observable
        Observable<Observable<Long>> observable = Observable
                .timer(0, 20, TimeUnit.MILLISECONDS)
                .map(new Func1<Long, Observable<Long>>() {
                    @Override
                    public Observable<Long> call(Long aLong) {
                        // 每隔200毫秒产生一组数据（0,10,20,30,40)
                        return Observable.timer(0, 2000, TimeUnit.MILLISECONDS)
                                .map(new Func1<Long, Long>() {
                                    @Override
                                    public Long call(Long aLong) {
                                        return aLong * 10;
                                    }
                                }).take(5);
                    }
                }).take(2);

        Observable.switchOnNext(observable).subscribe(new Subscriber<Long>() {
            @Override
            public void onCompleted() {
                System.out.println("Sequence complete.");
            }

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

            @Override
            public void onNext(Long aLong) {
                System.out.println("Next:" + aLong);
            }
        });
        try {
            Thread.sleep(Integer.MAX_VALUE);
        } catch (InterruptedException e1) {
            e1.printStackTrace();
        }
```

## 结果

```
Next:0
Next:0
Next:10
Next:20
Next:30
Next:40
Sequence complete.
```

# zip

通过一个函数将多个Observables的发射物结合到一起，基于这个函数的结果为每个结合体发射单个据项。

```
Observable<Integer> observable1 = Observable.just(10,20,30);
        Observable<Integer> observable2 = Observable.just(4, 8, 12, 16);
        Observable.zip(observable1, observable2, new Func2<Integer, Integer, Integer>() {
            @Override
            public Integer call(Integer integer, Integer integer2) {
                return integer + integer2;
            }
        }).subscribe(new Subscriber<Integer>() {
            @Override
            public void onCompleted() {
                System.out.println("Sequence complete.");
            }

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

            @Override
            public void onNext(Integer value) {
                System.out.println("Next:" + value);
            }
        });
```

## 结果

```
Next:14
Next:28
Next:42
Sequence complete.
```