链滴

RxJava 操作符之 -- 异常操作符

作者：hiquanta

原文链接：https://ld246.com/article/1490866905566

来源网站：链滴

```
Observable<Integer> observable = Observable.create(new Observable.OnSubscribe<Integer>
) {
        @Override
        public void call(Subscriber<? super Integer> subscriber) {
            if (subscriber.isUnsubscribed()) return;
            //循环输出数字
            try {
                for (int i = 0; i < 10; i++) {
                    if (i == 4) {
                        throw new Exception("this is number 4 error!  ");
                    }
                    subscriber.onNext(i);
                }
                subscriber.onCompleted();
            } catch (Exception e) {
                subscriber.onError(e);
            }
        }
    });

    observable.onErrorReturn(new Func1<Throwable, Integer>() {
        @Override
        public Integer call(Throwable throwable) {
            return 1004;
        }
    }).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:0
Next:1
Next:2
Next:3
Next:1004
Sequence complete.
```

onErrorResumeNext

onErrorResumeNext方法返回一个镜像原有Observable行为的新Observable，后者会忽略前者的onrror调用，不会将错误传递给观察者，作为替代，它会开始镜像另一个，备用的Observable

```
Observable<Integer> observable1 = Observable
        .create(new Observable.OnSubscribe<Integer>() {
            @Override
            public void call(Subscriber<? super Integer> subscriber) {
                if (subscriber.isUnsubscribed())
                    return;
                // 循环输出数字
                try {
                    for (int i = 0; i < 10; i++) {
                        if (i == 4) {
                            throw new Exception(
                                    "this is number 4 error!  ");
                        }
                        subscriber.onNext(i);
                    }
                    subscriber.onCompleted();
                } catch (Exception e) {
                    subscriber.onError(e);
                }
            }
        });

    observable1.onErrorResumeNext(
        new Func1<Throwable, Observable<? extends Integer>>() {
            @Override
            public Observable<? extends Integer> call(
                    Throwable throwable) {
                return Observable.just(100, 101, 102);
            }
        }).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:0
Next:1
Next:2
```

```
Next:3
Next:100
Next:101
Next:102
Sequence complete.
```

onExceptionResumeNext

和onErrorResumeNext类似，onExceptionResumeNext方法返回一个镜像原有Observable行为的Observable，也使用一个备用的Observable，不同的是，如果onError收到的Throwable不是一个Exception，它会将错误传递给观察者的onError方法，不会使用备用的Observable。

```
Observable<Integer> observable2 = Observable.create(new Observable.OnSubscribe<Intege
>() {
        @Override
        public void call(Subscriber<? super Integer> subscriber) {
            if (subscriber.isUnsubscribed()) return;
            //循环输出数字
            try {
                for (int i = 0; i < 10; i++) {
                    if (i == 4) {
                        throw new Exception("this is number 4 error!  ");
                    }
                    subscriber.onNext(i);
                }
                subscriber.onCompleted();
            } catch (Throwable e) {
                subscriber.onError(e);
            }
        }
    });

    observable2.onExceptionResumeNext(Observable.just(100, 101, 102)).subscribe(new Sub
criber<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:0
```

```
Next:1
Next:2
Next:3
Next:100
Next:101
Next:102
Sequence complete.
```

retry

如果原始Observable遇到错误，重新订阅它期望它能正常终止

```
Observable<Integer> observable = Observable
            .create(new Observable.OnSubscribe<Integer>() {
                @Override
                public void call(Subscriber<? super Integer> subscriber) {
                    if (subscriber.isUnsubscribed())
                        return;
                    // 循环输出数字
                    try {
                        for (int i = 0; i < 10; i++) {
                            if (i == 4) {
                                throw new Exception(
                                        "this is number 4 error!  ");
                            }
                            subscriber.onNext(i);
                        }
                        subscriber.onCompleted();
                    } catch (Throwable e) {
                        subscriber.onError(e);
                    }
                }
            });

        observable.retry(2).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:0
Next:1
Next:2
Next:3
Next:0
Next:1
Next:2
Next:3
Next:0
Next:1
Next:2
Next:3
Next:0
Next:1
Next:2
Next:3
Next:0
Next:1
Next:2
Next:3
Next:0
Next:1
Next:2
Next:3
Error: this is number 4 error!
```

retryWhen

retryWhen和retry类似，区别是，retryWhen将onError中的Throwable传递给一个函数，这个函数生另一个Observable，retryWhen观察它的结果再决定是不是要重新订阅原始的Observable。如果个Observable发射了一项数据，它就重新订阅，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<Integer> observable1 = Observable
        .create(new Observable.OnSubscribe<Integer>() {
            @Override
            public void call(Subscriber<? super Integer> subscriber) {
                if (subscriber.isUnsubscribed())
                    return;
                // 循环输出数字
                try {
                    for (int i = 0; i < 10; i++) {
                        if (i == 4) {
                            throw new Exception(
                                    "this is number 4 error!  ");
                        }
                        subscriber.onNext(i);
                    }
                    subscriber.onCompleted();
                } catch (Throwable e) {
                    subscriber.onError(e);
                }
            }
        });
```

```
observable1.retryWhen(
        new Func1<Observable<? extends Throwable>, Observable<?>>() {
            @Override
            public Observable<?> call(
                    Observable<? extends Throwable> observable) {
                System.out.println("delay retry by " + "1"
                        + " second(s)");
                return Observable.timer(1, TimeUnit.SECONDS);
            }
        }).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);
    }
});
try {
    Thread.sleep(Integer.MAX_VALUE);
} catch (InterruptedException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}ssss
```

结果

```
delay retry by 1 second(s)
Next:0
Next:1
Next:2
Next:3
Next:0
Next:1
Next:2
Next:3
Sequence complete.
```