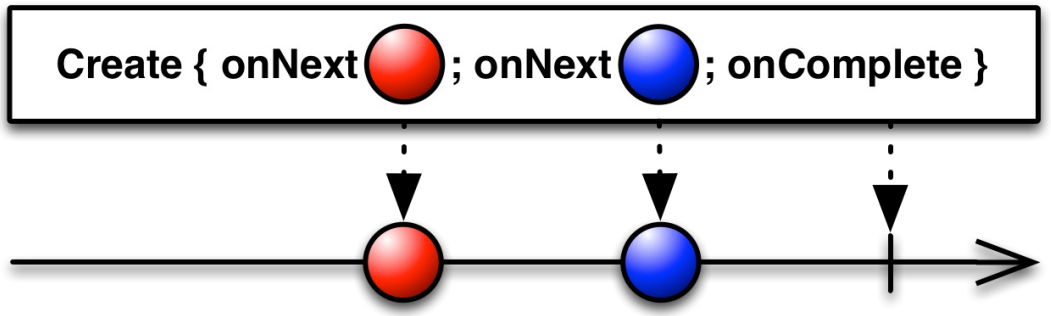
RxJavaOperator 이해와 사용법

manasobi RxJava

Qiita Yuki_Yamada Edited at 2016-10-27

1. 생성

1.1 create



[ReactiveX - Create operator][create] 각각의 Observable의 subscribe에 의해 Observable을 생성하는 오퍼레이터. 비동기처리에서 에러 핸들링이 필요할 때 사용.

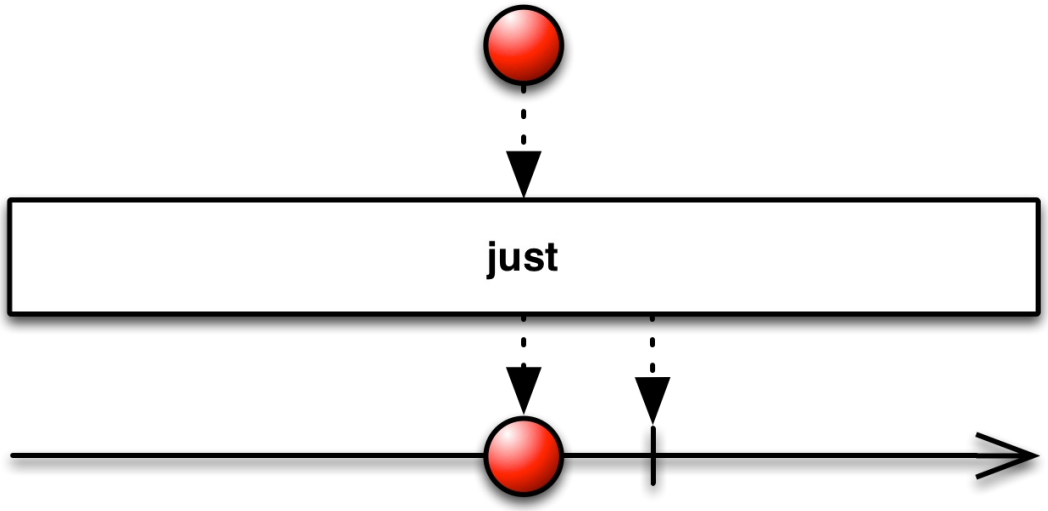
```
Observable<String> observable = Observable.create(
    new Observable.OnSubscribe<String>() {
        @Override
        public void call(Subscriber<? super String> subscriber) {
            try {
                FileInputStream fileInputStream;
                fileInputStream = openFileInput("MyFile.txt");
                byte[] readBytes = new byte[fileInputStream.availab
                fileInputStream.read(readBytes);
                subscriber.onNext(new String(readBytes));

                fileInputStream = openFileInput("2ndMyFile.txt");
                byte[] readBytes2nd = new byte[fileInputStream.avai
                fileInputStream.read(readBytes2nd);
                subscriber.onNext(new String(readBytes2nd));

                subscriber.onCompleted();
            } catch (Exception e) {
```

```
                subscriber.onError(e);
            }
        }
    }
```

create를 사용하는 경우는 예최처리가 발생하는 경우라고 생각한다. 자기가 subscribe 발화를 제어하는 느낌이다. 공식 샘플에는 for문에서 observable을 이용하고 있는데 Iterator를 가지고 있지 않는 연속된 데이타를 Observable로 하고 싶은 경우에 사용한다고 할 수 있겠다.

1.2 just

Observable로 하는 값을 직접 지정.



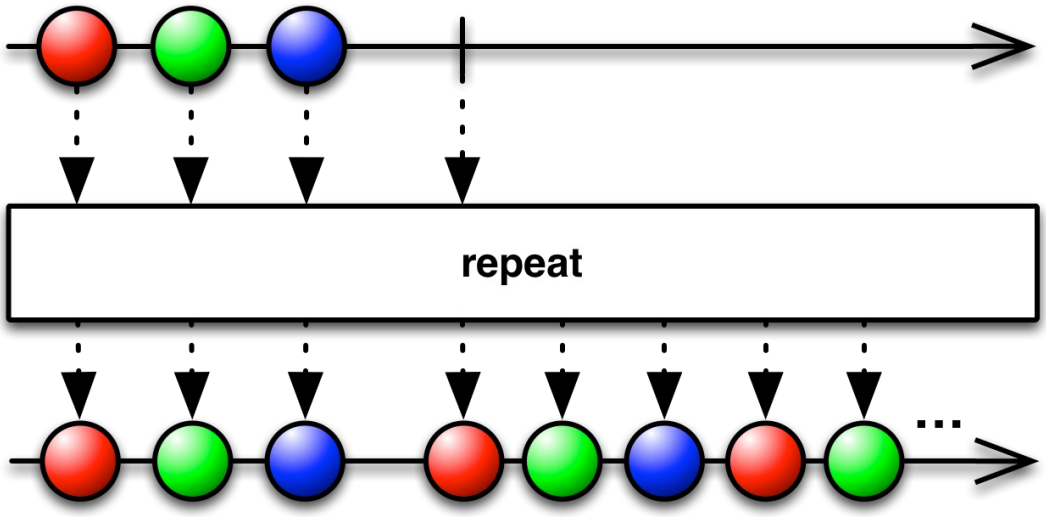
[ReactiveX - Just operator][just] 그림에는 하나밖에 Observable에 흐르고 있지 않지만, 10개까지 지정 가능하다.

```
Observable.just(1,2,3,4,5,6,7,8,9)
```

테스트 등에 사용될 수 있다.

1.3 repeat

onCompleted를 통과했지만 한번더 Subscribe를 해준다.

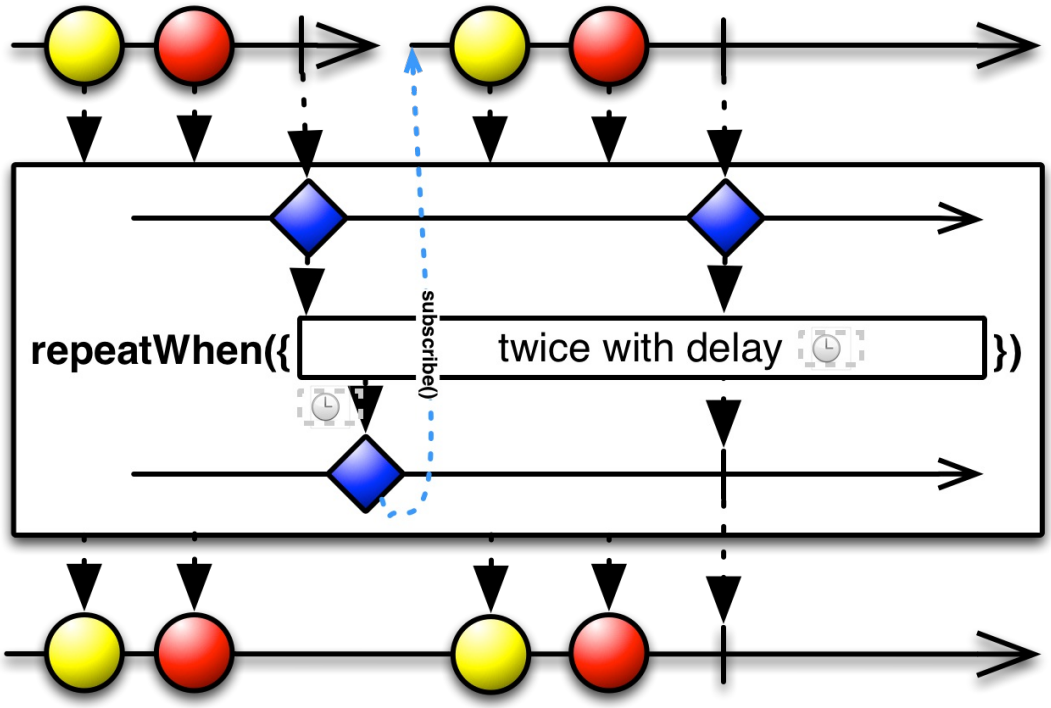


빨강, 초록, 파랑을 Observable로 흐르게하고, repeatOperator를 통과했기때문에 같은 데이터가 반복해서 subscribe된다.

```
Observable.just(1,2,3,4,5,6,7,8,9)
          .repeat(5)
```

1.4 repeatWhen

repeat와 유사 함수로 repeat하는 간격을 지정.

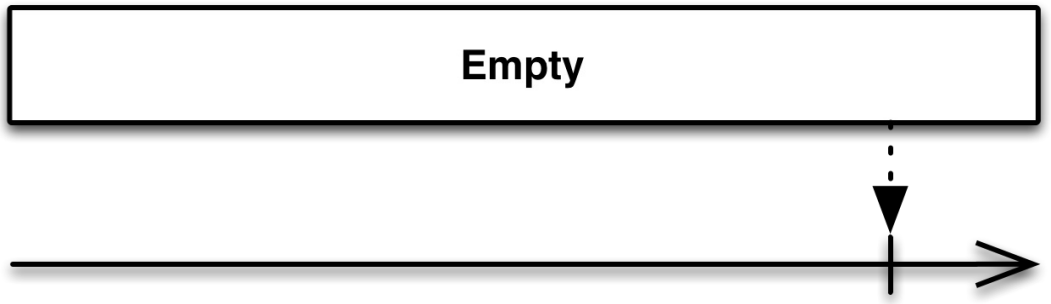


데이터는 노란, 빨간 공. 이것이 막대기에서 종료하면 파란 마름모가 발행되고 delay시간이 포함되어 한번 더 subscribe된다.

```
Observable
    .just(1,2,3,4,5,6,7,8,9)
    .repeatWhen(new Func1<Observable<? extends Void>, Observable<?>
        @Override
        public Observable<?> call(Observable<? extends Void> observ
            return observable.delay(5, TimeUnit.DAYS);
        }
    });
```

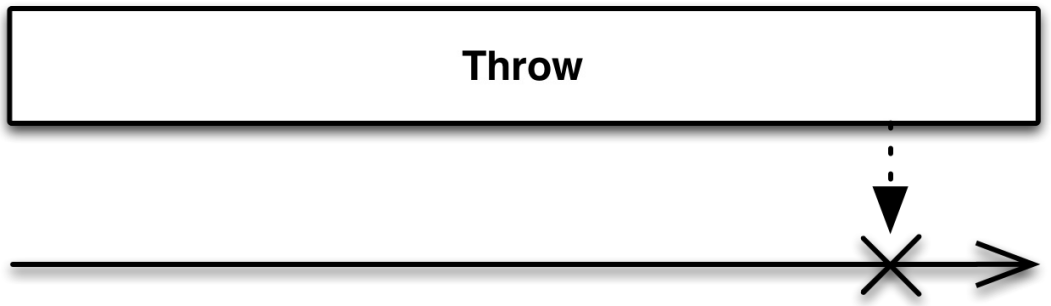
1.5 Empty/Never/Throw

- Empty는 onComplete만 발행한다.
- Never는 그것조차 호출하지 않는 공백의 Observable이다.
- Throw는 onError를 발행하는 Observable를 발행한다.



최후에 발화하고 있다. 이것이 OnCompleted이다.

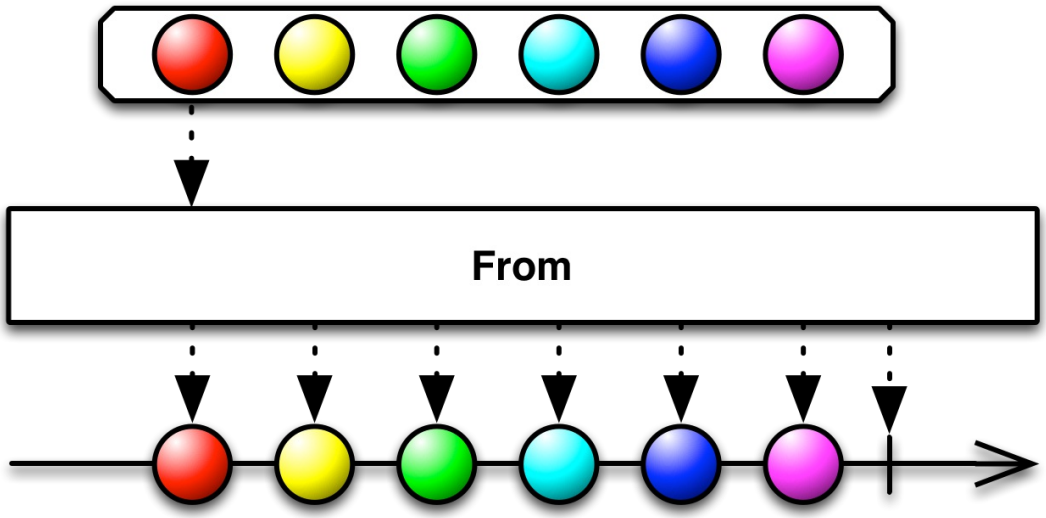
보는 것처럼 발화조차 하지 않는다.



에러의 발화를 하고있다.

1.6 from

Iterator를 가진 오브젝트로부터 Observable을 생성한다.



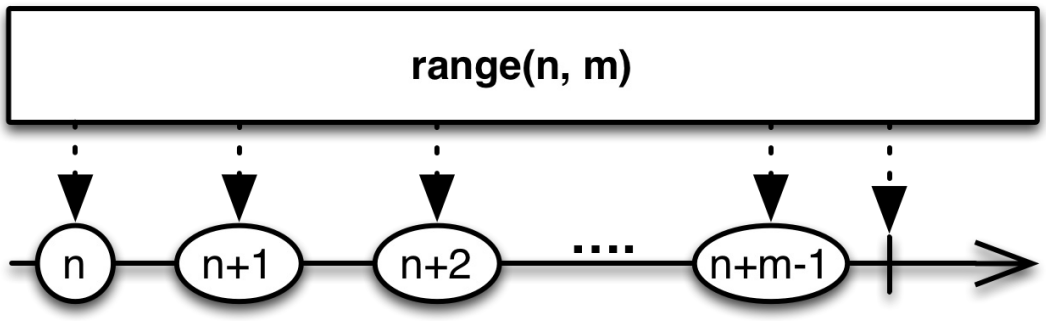

```
String[] array = new String[]{"sasaki","ササキ","ささき","佐々木"};
Observable.from(array)
```

맵으로부터 데이터를 취득하는 경우는 아래와 같다.

```
Observable.from(map.entrySet())
```

1.7 range

int로 시작, 종료를 지정하고 그 사이의 숫자의 Observable을 생성한다.

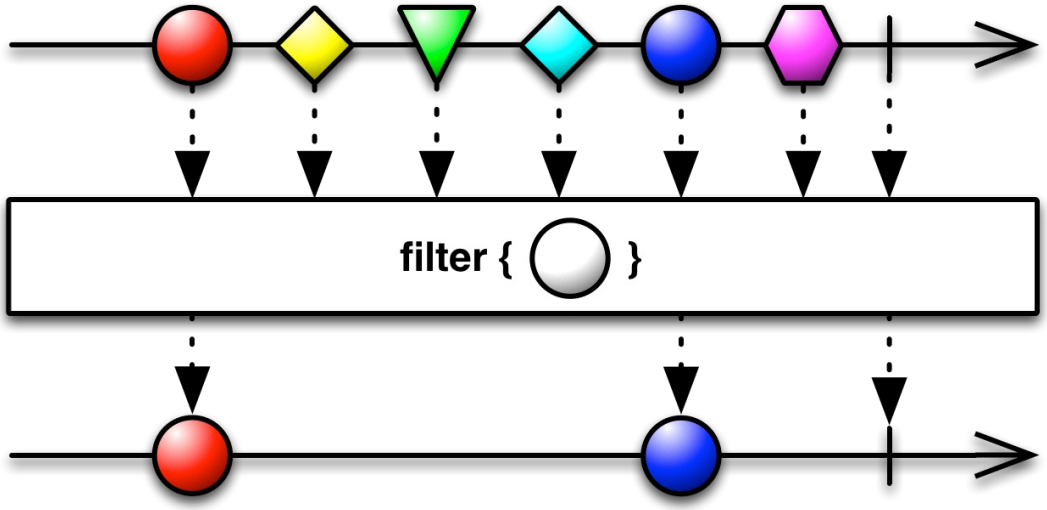



```
Observable.range(1,10);
```

2. 필터링

흘러들어온 데이터에 대하여 onNext를 발행할까 어떨까라는 처리를 주로하는 오퍼레이터이다.

2.1 filter



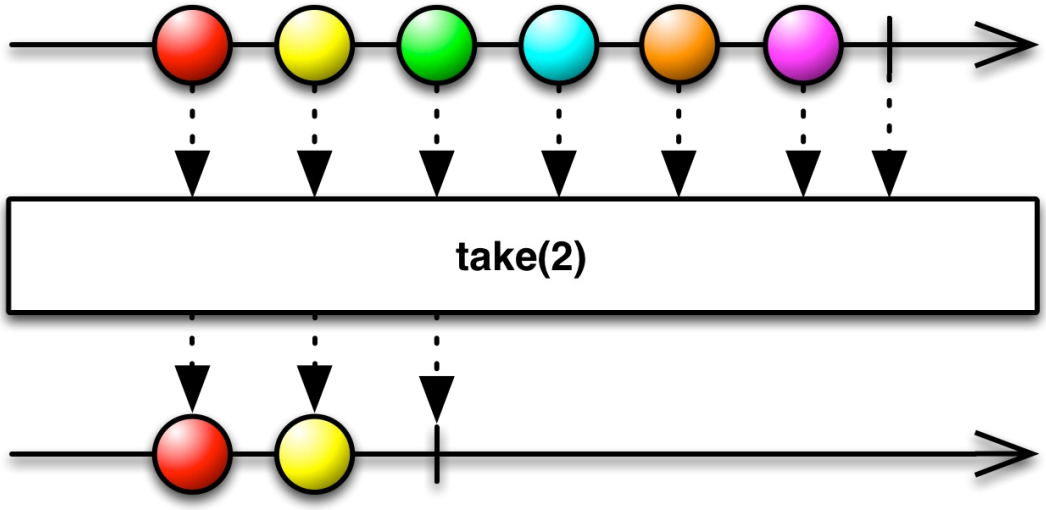
[ReactiveX - Filter operator][filter] 이 그림에서는 전달된 Observer로부터 동그라미만을 필터링해서, 동그라미 모양만을 onNext한다.

```
Observable.just(1,10,4,21,19,2,13,9)
    .filter(new Func1<Integer, Boolean>() {
        @Override
        public Boolean call(Integer item) {
            return (item < 15);
        }
    })
```

함수 Func1에서 반환 값이 true이면 그 값이 onNext에 도달한다.

2.2 take

개수를 지정해서 꺼낸다.



[ReactiveX - Take operator][take]

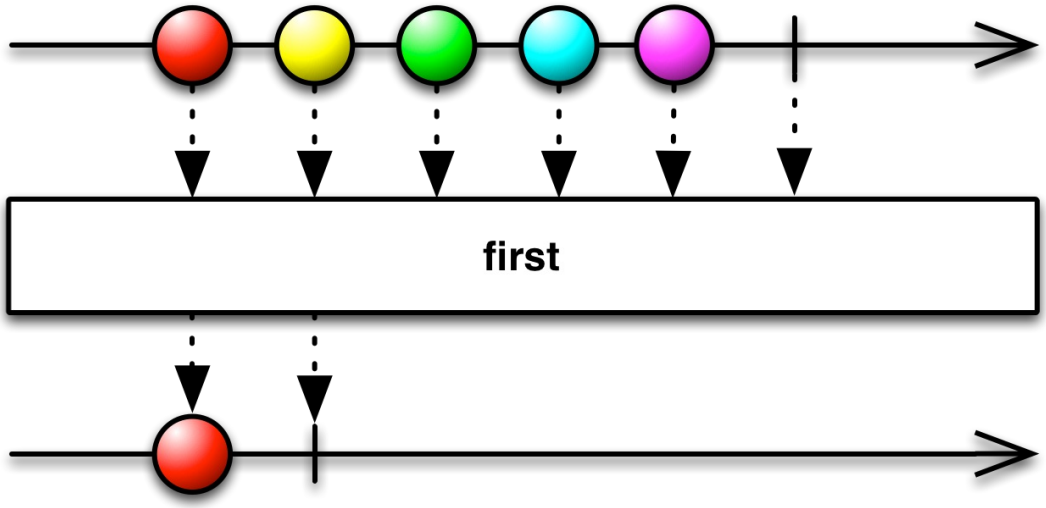
```
Observable.just(1,10,4,21,19,2,13,9)
          .take(2)
```

takeLast를 이용하면 마지막 데이터로부터 데이터를 취득한다.

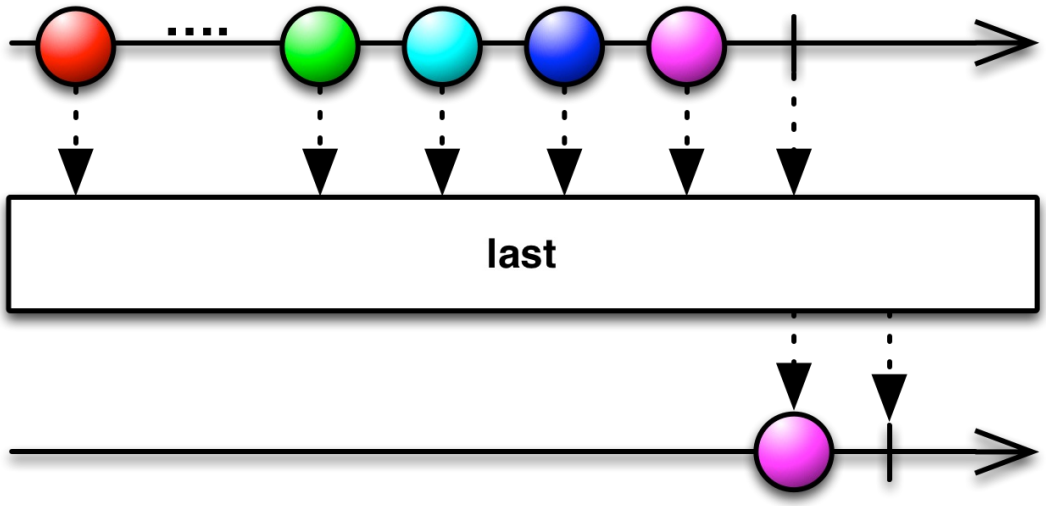
```
Observable.range(1,10)
          .takeLast(3)
```

2.3 first/last/elementAt ..OrDefault

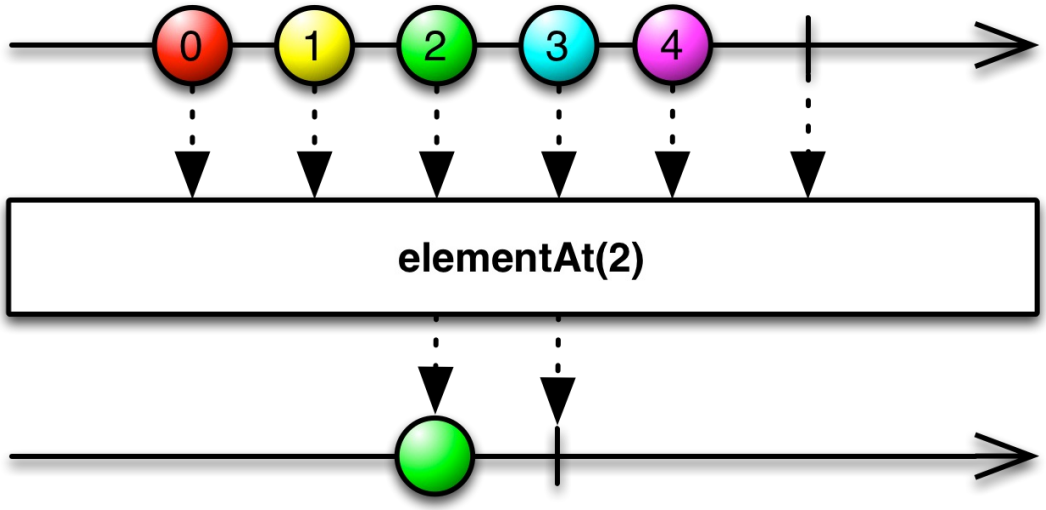
최초(최후, 또는 지정한 곳)만을 취득해서 onNext한다. 요소 하나만을 onNext하는 것이 특징이다.



최초의 빨간색만을 Observable에 흘려보낸다.



제일 마지막 핑크색만을 Observable로 보낸다.

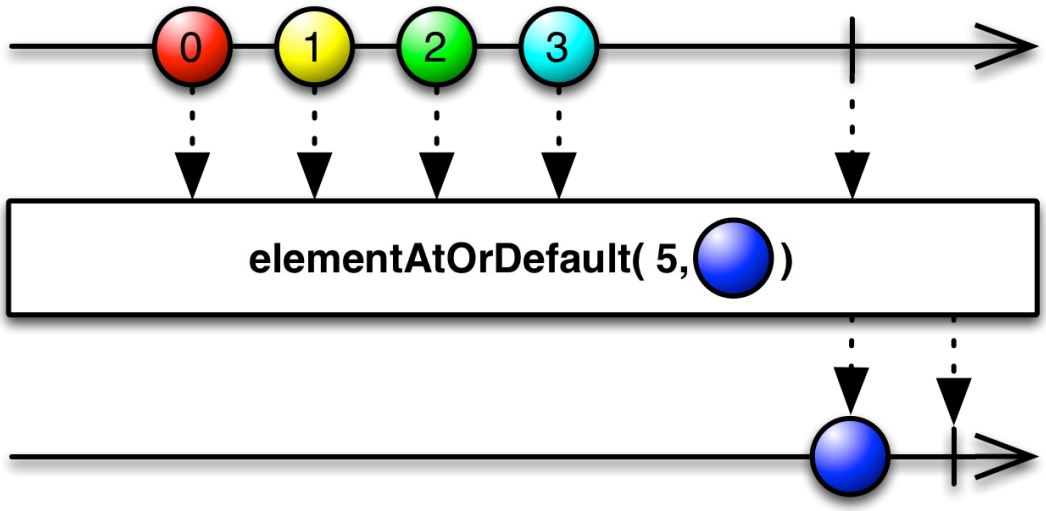


지정한 요소 번호 2번만을 Observable로 보낸다.

```
Observable.just(1,2,3,4,5,6).first();
Observable.just(1,2,3,4,5,6).last();
Observable.just(1,2,3,4,5,6).firstOrDefault(1);
```

OrDefault가 있어서 안전한 설계가 가능하다.

```
Observable.just(1,2,3,4,5,6).elementAt(3);
Observable.just(1,2,3,4,5,6).elementAtOrDefault(2,11);
```



요소 5번이 없더라도 예외가 발생하지않고 Default 값을 Observable에 값을 전달한다.

2.4 sample

일정 시간마다 스트림의 값을 취득하여 전달한다. 인수는 `sample(long,TimeUnit)` 이다.