무료 전자 책

배우기

rx-java

Free unaffiliated eBook created from
Stack Overflow contributors.

#rx-java

1: rx-java

rx-java .

RxJava Reactive Extensions : Java VM .

RxJava .

1.x		1.3.0	2017-05-05
2.x		2.1.1	2017-06-21

Examples

rx-java

1. ```
   compile 'io.reactivex:rxjava2:rxjava:2.1.1'
   ```

2. ```
   <dependency>
       <groupId>io.reactivex.rxjava2</groupId>
       <artifactId>rxjava</artifactId>
       <version>2.1.1</version>
   </dependency>
   ```

3. ```
   <dependency org="io.reactivex.rxjava2" name="rxjava" rev="2.1.1" />
   ```

4. JFrog

   ```
   repositories {
   maven { url 'https://oss.jfrog.org/libs-snapshot' }
   }

   dependencies {
       compile 'io.reactivex:rxjava:2.0.0-SNAPSHOT'
   }
   ```

5. jar Maven `pom` .

   ```
   <?xml version="1.0"?>
   <project xmlns="http://maven.apache.org/POM/4.0.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
       <modelVersion>4.0.0</modelVersion>
       <groupId>com.netflix.rxjava.download</groupId>
       <artifactId>rxjava-download</artifactId>
       <version>1.0-SNAPSHOT</version>
       <name>Simple POM to download rxjava and dependencies</name>
   ```
   ```

```
    <url>http://github.com/ReactiveX/RxJava</url>
    <dependencies>
        <dependency>
            <groupId>io.reactivex</groupId>
            <artifactId>rxjava</artifactId>
            <version>2.0.0</version>
            <scope/>
        </dependency>
    </dependencies>
</project>
```

:

```
$ mvn -f download-rxjava-pom.xml dependency:copy-dependencies
```

`rxjava-*.jar  ./target/dependency/.`

Java 6 .

, !

`Hello, World!  Hello, World!`

```
public void hello() {
  Observable.just("Hello, World!") // create new observable
    .subscribe(new Action1<String>() { // subscribe and perform action

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

    });
}
```

Java 8

```
public void hello() {
      Observable.just("Hello, World!") // create new observable
        .subscribe(onNext -> { // subscribe and perform action
             System.out.println(onNext);
        });
}
```

RxJava

RxJava `Observables` and `Subscribers` . `Observable` `Subscriber` .

`Observable` . Observables . . `Observable` `Observable` .

.

```
Observable<Integer> integerObservable = Observable.just(1, 2, 3); // Integer observable
```

```
Observable<String> stringObservable = Observable.just("Hello, ", "World", "!"); // String
observable
```

integerObservable stringObservable Rx just . Observable .

Subscriber . Subscriber Subscriber . Observable onNext() , onCompleted() , onError()
Subscriber .

```
Subscriber<Integer> mSubscriber = new Subscriber<Integer>() {
        // NOTE THAT ALL THESE ARE CALLED BY THE OBSERVABLE
        @Override
        public void onCompleted() {
            // called when all objects are emitted
            System.out.println("onCompleted called!");
        }

        @Override
        public void onError(Throwable throwable) {
            // called when an error occurs during emitting objects
            System.out.println("onError called!");
        }

        @Override
        public void onNext(Integer integer) {
            // called for each object that is emitted
            System.out.println("onNext called with: " + integer);
        }
    };
```

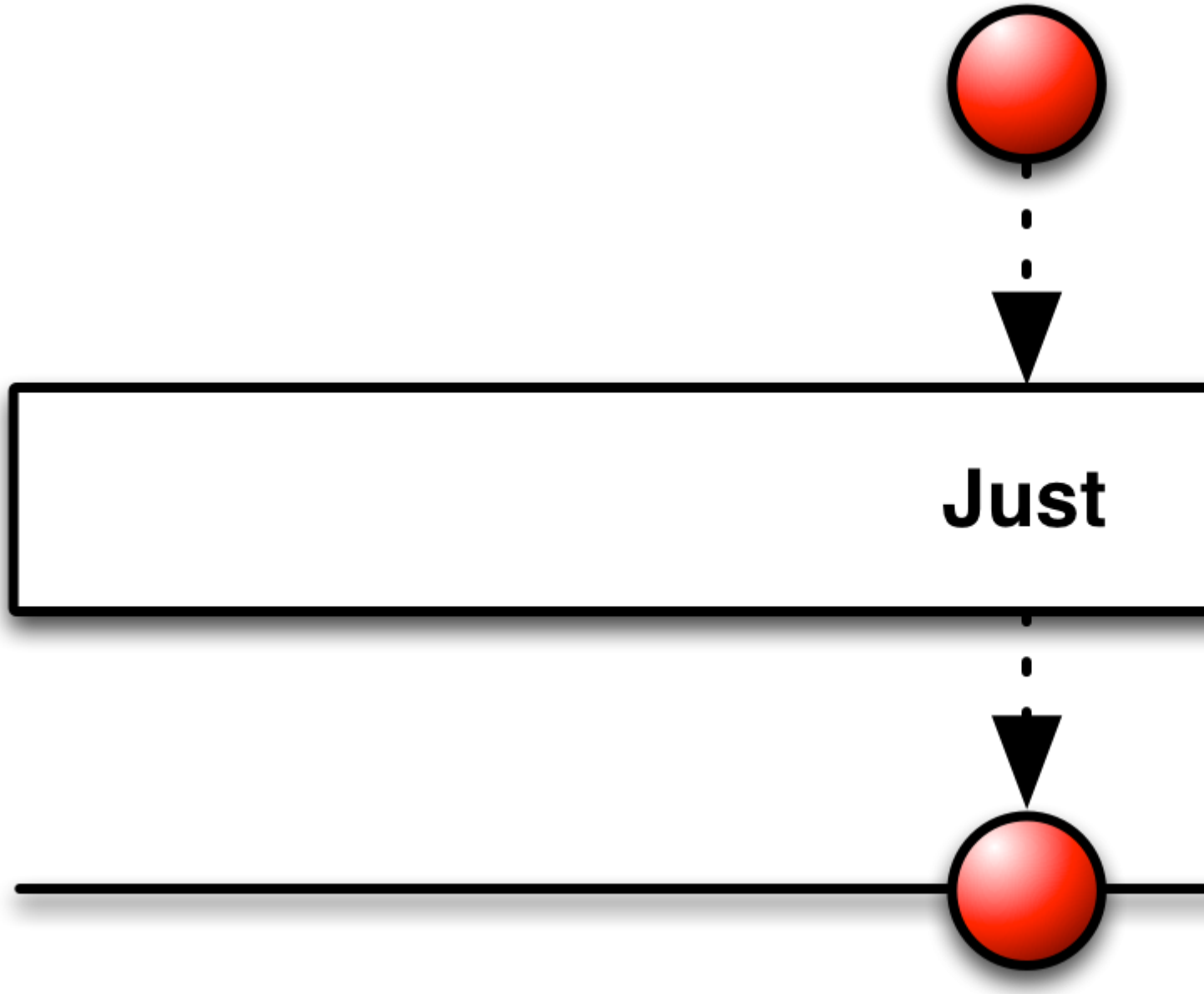
Subscriber . Subscriber subscribe Subscriber .

```
integerObservable.subscribe(mSubscriber);
```

.

```
onNext called with: 1
onNext called with: 2
onNext called with: 3
onCompleted called!
```

Observable . Observable , onNext, onComplete onError . observable onNext . Observable onComplete . Observable onError . Rx . "Just" .



, , X . "Just" onNext onComplete . "" . ReactiveX RexX RxJava . RxMarbles .

rx-java : https://riptutorial.com/ko/rx-java/topic/974/rx-java-

2: Android with RxJava

RxAndroid . 0.25.0 1.x .

1.0 .

Examples

RxAndroid - AndroidSchedulers

Android RxJava .

gradle RxJava RxAndroid :

```
 // use the last version
compile 'io.reactivex.rxjava2:rxjava:2.1.1'
compile 'io.reactivex.rxjava2:rxandroid:2.0.1'
```

RxJava RxAndroid UI .

:

```
Observable.just("one", "two", "three", "four", "five")
        .subscribeOn(Schedulers.newThread())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(
            data -> doStuffOnMainThread(),
            error -> handleErrorOnMainThread()
        )
```

`Looper` .

```
Looper backgroundLooper = // ...
Observable.just("one", "two", "three", "four", "five")
        .subscribeOn(Schedulers.newThread())
        .observeOn(AndroidSchedulers.from(backgroundLooper))
        .subscribe(
            data -> doStuffOnMainThread(),
            error -> handleErrorOnMainThread()
        )
```

RxJava .

RxLifecycle

RxLifecycle Android .

Observable / .

.

```
// use the last version available
compile 'com.trello:rxlifecycle:0.6.1'
compile 'com.trello:rxlifecycle-components:0.6.1'
```

`Rx*` .

- `RxActivity` / `support.RxFragmentActivity` / `support.RxAppCompatActivity`
- `RxFragment` / `support.RxFragment`
- `RxDialogFragment` / `support.RxDialogFragment`
- `support.RxAppCompatDialogActivity`

Observable :

```
someObservable
    .compose(bindToLifecycle())
    .subscribe();
```

`onCreate()` `onDestroy()` .

:

- `onStart()` -> `onStop()`
- `onResume()` -> `onPause()`
- `onAttach()` -> `onDetach()` *( )*
- `onViewCreated()` -> `onDestroyView()` *( )*

.

:

```
someObservable
    .compose(bindUntilEvent(ActivityEvent.DESTROY))
    .subscribe();
```

:

```
someObservable
    .compose(bindUntilEvent(FragmentEvent.DESTROY_VIEW))
    .subscribe();
```

(lifecycle `lifecycle()` `lifecycle()` .

RxLifecycle .

```
.compose(RxLifecycleAndroid.bindActivity(lifecycle))
```

`Single` `Completable` bind `forSingle()` `forCompletable` .

```
someSingle
    .compose(bindToLifecycle().forSingle())
    .subscribe();
```

Navi .

Rxpermissions

Android M RxJava .

.

Rxjava

```
dependencies {
    compile 'com.tbruyelle.rxpermissions:rxpermissions:0.8.0@aar'
}
```

Rxjava2

```
dependencies {
    compile 'com.tbruyelle.rxpermissions2:rxpermissions:0.8.1@aar'
}
```

(Retrolambda) :

```
// Must be done during an initialization phase like onCreate
RxPermissions.getInstance(this)
    .request(Manifest.permission.CAMERA)
    .subscribe(granted -> {
        if (granted) { // Always true pre-M
           // I can control the camera now
        } else {
           // Oups permission denied
        }
    });
```

: https://github.com/tbruyelle/RxPermissions .

Android with RxJava : https://riptutorial.com/ko/rx-java/topic/7125/android-with-rxjava

3: RxJava2

rxjava version2 Flowable Subscriber .

rxjava2 .

```
    <dependency>
        <groupId>io.reactivex.rxjava2</groupId>
        <artifactId>rxjava</artifactId>
        <version>2.0.8</version>
    </dependency>
```

Examples

TestProducer Integer Subscriber . Flowable<Integer> . request(long) Integer Subscription .

Subscription subscriber request() onNext() onNext() . , outStandingRequests isProducing .

```
class TestProducer extends Flowable<Integer> {
    static final Logger logger = LoggerFactory.getLogger(TestProducer.class);
    final int from, to;

    public TestProducer(int from, int to) {
        this.from = from;
        this.to = to;
    }

    @Override
    protected void subscribeActual(Subscriber<? super Integer> subscriber) {
        subscriber.onSubscribe(new Subscription() {

            /** the next value. */
            public int next = from;
            /** cancellation flag. */
            private volatile boolean cancelled = false;
            private volatile boolean isProducing = false;
            private AtomicLong outStandingRequests = new AtomicLong(0);

            @Override
            public void request(long n) {
                if (!cancelled) {

                    outStandingRequests.addAndGet(n);

                    // check if already fulfilling request to prevent call  between request()
an subscriber .onNext()
                    if (isProducing) {
                        return;
                    }

                    // start producing
                    isProducing = true;

                    while (outStandingRequests.get() > 0) {
                        if (next > to) {
```

```
                            logger.info("producer finished");
                            subscriber.onComplete();
                            break;
                        }
                        subscriber.onNext(next++);
                        outStandingRequests.decrementAndGet();
                    }
                    isProducing = false;
                }
            }

            @Override
            public void cancel() {
                cancelled = true;
            }
        });
    }
}
```

Consumer `DefaultSubscriber<Integer>` `DefaultSubscriber<Integer>` . Integer .

```
class TestConsumer extends DefaultSubscriber<Integer> {

    private static final Logger logger = LoggerFactory.getLogger(TestConsumer.class);

    @Override
    protected void onStart() {
        request(1);
    }

    @Override
    public void onNext(Integer i) {
        logger.info("consuming {}", i);
        if (0 == (i % 5)) {
            try {
                Thread.sleep(500);
            } catch (InterruptedException ignored) {
                // can be ignored, just used for pausing
            }
        }
        request(1);
    }

    @Override
    public void onError(Throwable throwable) {
        logger.error("error received", throwable);
    }

    @Override
    public void onComplete() {
        logger.info("consumer finished");
    }
}
```

.

```
public static void main(String[] args) {
    try {
        final TestProducer testProducer = new TestProducer(1, 1_000);
```

```
        final TestConsumer testConsumer = new TestConsumer();

        testProducer
                .subscribeOn(Schedulers.computation())
                .observeOn(Schedulers.single())
                .blockingSubscribe(testConsumer);

    } catch (Throwable t) {
        t.printStackTrace();
    }
}
```

rxjava2 Flowable .

RxJava2 : https://riptutorial.com/ko/rx-java/topic/9810/rxjava2---

4: RxJava

Examples

RxJava

Retrofit2 RxJava.

RxJava Retrofit RxJava .

```
compile 'com.squareup.retrofit2:adapter-rxjava:2.1.0'
```

.

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.example.com")
    .addCallAdapterFactory(RxJavaCallAdapterFactory.create())
    .build();
```

API `Observable` . :

```
public interface GitHubService {
  @GET("users/{user}/repos")
  Observable<List<Repo>> listRepos(@Path("user") String user);
}
```

`Observable Single` .

RxJava . `flatMap` .

```
api.getRepo(repoId).flatMap(repo -> api.getUser(repo.getOwnerId())
    .subscribe(/*do something with the result*/);
```

`zip` . :

```
Observable.zip(api.getRepo(repoId1), api.getRepo(repoId2), (repo1, repo2) ->
    {
        //here you can combine the results
    }).subscribe(/*do something with the result*/);
```

RxJava : https://riptutorial.com/ko/rx-java/topic/2950/--rxjava

5:

Scheduler RxJava JVM . , TestScheduler . Schedulers .

Examples

TestSubscriber

TestSubscribers Action <?> Observable .

1,2,3 4 onNext Observable .

```
TestSubscriber<Integer> ts = TestSubscriber.create();
Observable.just(1,2,3,4).subscribe(ts);
ts.assertValues(1,2,3,4); // Success
```

assertValues assertValues . .

```
TestSubscriber<Integer> ts = TestSubscriber.create();
Observable.just(1,2,3,4).subscribe(ts);
ts.assertValues(1,2,3); // Fail
```

assertValues assertValues equals . .

```
TestSubscriber<Object> ts = TestSubscriber.create();
Observable.just(new Object(), new Object()).subscribe(ts);
ts.assertValues(new Object(), new Object()); // Fail
```

equals Observable .

```
public class Room {

    public String floor;
    public String number;

    @Override
    public boolean equals(Object o) {
        if (o == this) {
            return true;
        }
        if (o instanceof Room) {
            Room that = (Room) o;
            return (this.floor.equals(that.floor))
                    && (this.number.equals(that.number));
        }
        return false;
    }
}

TestSubscriber<Room> ts = TestSubscriber.create();
```

```
Observable.just(new Room("1", "10")).subscribe(ts);
ts.assertValue(new Room("1", "10"); // Success
```

assertValue assertValue .

.

```
TestSubscriber<Integer> ts = TestSubscriber.create();
Observable.just(1,2,3,4).subscribe(ts);
List<Integer> onNextEvents = ts.getOnNextEvents();
List<Throwable> onErrorEvents = ts.getOnErrorEvents();
List<Notification<Integer>> onCompletedEvents = ts.getOnCompletedEvents();
```

getOnNextEvents (getOn*Events) .

```
Observable<Integer> obs = Observable.just(1,2,3,4)
    .filter( x -> x % 2 == 0);

// note that we instanciate TestSubscriber via the constructor here
TestSubscriber<Integer> ts = new TestSubscriber();
obs.subscribe(ts);

// Note that we are not using Observable#forEach here
// but java.lang.Iterable#forEach.
// You should never use Observable#forEach unless you know
// exactly what you're doing
ts.getOnNextEvents()
    .forEach( integer -> assertTrue(integer % 2 == 0));
```

Observable#error

.

```
Observable<Integer> obs = Observable.error(new Exception("I am a Teapot"));

TestSubscriber<Integer> ts = new TestSubscriber<>();
obs.subscribe(ts);

ts.assertError(Exception.class);
```

.

```
Exception e = new Exception("I am a Teapot");
Observable<Integer> obs = Observable.error(e);

TestSubscriber<Integer> ts = new TestSubscriber<>();
obs.subscribe(ts);

ts.assertError(e);
```

TestScheduler

TestSchedulers Busy Wait, Threads Observables . . .

TestSchedulers RxJava .

```
TestScheduler testScheduler = new TestScheduler();
TestSubscriber<Integer> subscriber = TestSubscriber.create();
Observable.just(1,2,3)
          .delay(10, TimeUnit.SECONDS, testScheduler)
          .subscribe(subscriber);

try {
    Thread.sleep(TimeUnit.SECONDS.toMillis(11));
} catch (InterruptedException ignored) { }
subscriber.assertValues(1,2,3); // fails

testScheduler.advanceTimeBy(10, TimeUnit.SECONDS);
subscriber.assertValues(1,2,3); // success
```

TestScheduler . .

```
testScheduler.advanceTimeBy(amount, timeUnit);
testScheduler.advanceTimeTo(when, timeUnit);
testScheduler.triggerActions();
```

TestScheduler .

, TestScheduler . . RxJava Schedulers.io () / computation () / etc . RxJava Hooks . Scheduler .

```
public final class TestSchedulers {

    public static TestScheduler test() {
        final TestScheduler testScheduler = new TestScheduler();
        RxJavaHooks.reset();
        RxJavaHooks.setOnComputationScheduler((scheduler) -> {
            return testScheduler;
        });
        RxJavaHooks.setOnIOScheduler((scheduler) -> {
            return testScheduler;
        });
        RxJavaHooks.setOnNewThreadScheduler((scheduler) -> {
            return testScheduler;
        });
        return testScheduler;
    }
}
```

. . TestScheduler triggerAction , . .

```
TestScheduler testScheduler = new TestScheduler();
TestSubscriber<Integer> subscriber = TestSubscriber.create();
Observable.just(1,2,3)
          .delay(10, TimeUnit.SECONDS, testScheduler)
          .subscribe(subscriber);
testScheduler.advanceTimeBy(9, TimeUnit.SECONDS);
subscriber.assertValues(); // success (delay hasn't finished)
```

```
testScheduler.advanceTimeBy(10, TimeUnit.SECONDS);
subscriber.assertValues(1,2,3); // success (delay has finished)
```

(RxJava)

: https://riptutorial.com/ko/rx-java/topic/5207/-

6:

Examples

Backpressure `Observable` .

.

```
PublishSubject<Integer> source = PublishSubject.create();

source
.observeOn(Schedulers.computation())
.subscribe(v -> compute(v), Throwable::printStackTrace);

for (int i = 0; i < 1_000_000; i++) {
    source.onNext(i);
}

Thread.sleep(10_000);
```

1 . `compute(int)` `Observable` . for `onNext` ing .

, . Rx.NET RxJava . , . 10 1000 1000 `OutOfMemoryError` GC .

`onErrorXXX` `onErrorXXX` `onErrorXXX` `onBackpressureXXX` `onBackpressureXXX` .

`PublishSubject` . , `interval` , .

RxJava `observeOn` `observeOn` `MissingBackpressureException` . .

(`MissingBackpressureException` `MissingBackpressureException`) `MissingBackpressureException` . :

```
Observable.range(1, 1_000_000)
.observeOn(Schedulers.computation())
.subscribe(v -> compute(v), Throwable::printStackTrace);

Thread.sleep(10_000);
```

. "" `observeOn` `observeOn` `range` .

(,). `range` `Producer` (`Subscriber`) `setProducer` `observeOn` `observeOn` . , `observeOn` `Producer.request(n)` `range` (, `onNext`) `onNext` . `request` `observeOn` .

( ). .

```
Observable.range(1, 1_000_000)
.subscribe(new Subscriber<Integer>() {
    @Override
    public void onStart() {
        request(1);
    }
```

```
    public void onNext(Integer v) {
        compute(v);

        request(1);
    }

    @Override
    public void onError(Throwable ex) {
        ex.printStackTrace();
    }

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

onStart range onNext . compute(int) range range . range onNext StackOverflowError .

(trampolining) . range onNext() request(1) onNext() onNext() onNext() . .

```
@Override
public void onNext(Integer v) {
    request(1);

    compute(v);
}
```

onStart . Observable Subscriber request(1) . onNext request(1) .

```
Observable.range(1, 1_000_000)
.subscribe(new Subscriber<Integer>() {

    String name;

    @Override
    public void onStart() {
        request(1);

        name = "RangeExample";
    }

    @Override
    public void onNext(Integer v) {
        compute(name.length + v);

        request(1);
    }

    // ... rest is the same
});
```

, onStart NullPointerException . request(1) onNext onNext name onNext onStart post request
.

onStart request() . request() ( ) .

onBackpressureXXX

MissingBackpressureException MissingBackpressureException observeOn . PublishSubject , timer() interval() create() create() PublishSubject .

.

. observeOn 16 observeOn .

RxJava . bufferSize , prefetch capacityHint capacityHint . observeOn observeOn .

```
PublishSubject<Integer> source = PublishSubject.create();

source.observeOn(Schedulers.computation(), 1024 * 1024)
      .subscribe(e -> { }, Throwable::printStackTrace);

for (int i = 0; i < 1_000_000; i++) {
    source.onNext(i);
}
```

. .

/

(/) MissingBackpressureException .

```
PublishSubject<Integer> source = PublishSubject.create();

source
      .buffer(1024)
      .observeOn(Schedulers.computation(), 1024)
      .subscribe(list -> {
          list.parallelStream().map(e -> e * e).first();
      }, Throwable::printStackTrace);

for (int i = 0; i < 1_000_000; i++) {
    source.onNext(i);
}
```

(Observable) throttleFirst (throttleFirst , throttleLast , throttleWithTimeout) .

```
PublishSubject<Integer> source = PublishSubject.create();

source
      .sample(1, TimeUnit.MILLISECONDS)
      .observeOn(Schedulers.computation(), 1024)
      .subscribe(v -> compute(v), Throwable::printStackTrace);

for (int i = 0; i < 1_000_000; i++) {
    source.onNext(i);
}
```

MissingBackpressureException .

onBackpressureBuffer ()

. JVM .

```
 Observable.range(1, 1_000_000)
           .onBackpressureBuffer()
           .observeOn(Schedulers.computation(), 8)
           .subscribe(e -> { }, Throwable::printStackTrace);
```

, observeOn MissingBackpressureException onBackpressureBuffer 100 observeOn .

onBackpressureBuffer . , backpressure . range .

onBackpressureBuffer 4 onBackpressureBuffer

onBackpressureBuffer (int capacity)

BufferOverflowError .

```
Observable.range(1, 1_000_000)
          .onBackpressureBuffer(16)
          .observeOn(Schedulers.computation())
          .subscribe(e -> { }, Throwable::printStackTrace);
```

. onBackpressureBuffer onBackpressureBuffer " " .

onBackpressureBuffer (int capacity, Action0 onOverflow)

() . .

onBackpressureBuffer (int capacity, Action0 onOverflow, BackpressureOverflow.Strategy)

. BackpressureOverflow.Strategy BackpressureOverflow 4 .

- ON_OVERFLOW_ERROR : , BufferOverflowException ON_OVERFLOW_ERROR .
- ON_OVERFLOW_DEFAULT : ON_OVERFLOW_ERROR ON_OVERFLOW_ERROR
- ON_OVERFLOW_DROP_LATEST : .
- ON_OVERFLOW_DROP_OLDEST : .

```
Observable.range(1, 1_000_000)
          .onBackpressureBuffer(16, () -> { },
              BufferOverflowStrategy.ON_OVERFLOW_DROP_OLDEST)
          .observeOn(Schedulers.computation())
          .subscribe(e -> { }, Throwable::printStackTrace);
```

., BufferOverflowException .

onBackpressureDrop ()

. ON_OVERFLOW_DROP_LATEST 0 onBackpressureBuffer .

(: GPS) .

```
 component.mouseMoves()
 .onBackpressureDrop()
 .observeOn(Schedulers.computation(), 1)
 .subscribe(event -> compute(event.x, event.y));
```

interval() . .

```
 Observable.interval(1, TimeUnit.MINUTES)
 .onBackpressureDrop()
 .observeOn(Schedulers.io())
 .doOnNext(e -> networkCall.doStuff())
 .subscribe(v -> { }, Throwable::printStackTrace);
```

onBackpressureDrop(Action1<? super T> onDrop) (shared) . (:).

onBackpressureLatest ()

. 1 ON_OVERFLOW_DROP_OLDEST onBackpressureBuffer .

onBackpressureDrop , . .

, , .

```
component.mouseClicks()
.onBackpressureLatest()
.observeOn(Schedulers.computation())
.subscribe(event -> compute(event.x, event.y), Throwable::printStackTrace);
```

onBackpressureDrop .

Observable . . "" / "" .

just :

```
Observable.just(1).subscribe(new Subscriber<Integer>() {
    @Override
    public void onStart() {
        request(0);
    }

    @Override
    public void onNext(Integer v) {
        System.out.println(v);
    }
```

```
    // the rest is omitted for brevity
}
```

onStart . just .

, just Subscriber :

```
int counter;

int computeValue() {
   return ++counter;
}

Observable<Integer> o = Observable.just(computeValue());

o.subscribe(System.out:println);
o.subscribe(System.out:println);
```

, 1 2 1 . .

```
int temp = computeValue();

Observable<Integer> o = Observable.just(temp);
```

computeValue .

fromCallable

fromCallable .

```
Observable<Integer> o = Observable.fromCallable(() -> computeValue());
```

computeValue 1 2 . fromCallable . . just map :

```
Observable.just("This doesn't matter").map(ignored -> computeValue())...
```

just computeValue .

...

, Iterable from :

```
 Observable.from(Arrays.asList(1, 2, 3, 4, 5)).subscribe(System.out::println);
```

( ) 2~10 just from .

from(Iterable) . . .

Iterable Observable C # (yield return yield break). Google Guava AbstractIterable

IxJava `Ix.generate()` `Ix.forloop()` . `Iterable` :

```
Iterable<Integer> iterable = () -> new Iterator<Integer>() {
    @Override
    public boolean hasNext() {
        return true;
    }

    @Override
    public Integer next() {
        return 1;
    }
};

Observable.from(iterable).take(5).subscribe(System.out::println);
```

classic for-loop `iterator` . `Observable` 5 . `Observable` .

create (SyncOnSubscribe)

() . , `get` `read` . `Iterable` .

RxJava `SyncOnSubscribe` . .

```
SyncOnSubscribe<Integer, InputStream> binaryReader = SyncOnSubscribe.createStateful(
     () -> new FileInputStream("data.bin"),
     (inputstream, output) -> {
         try {
             int byte = inputstream.read();
             if (byte < 0) {
                 output.onCompleted();
             } else {
                 output.onNext(byte);
             }
         } catch (IOException ex) {
             output.onError(ex);
         }
         return inputstream;
     },
     inputstream -> {
         try {
             inputstream.close();
         } catch (IOException ex) {
             RxJavaHooks.onError(ex);
         }
     }
 );

 Observable<Integer> o = Observable.create(binaryReader);
```

`SyncOnSubscribe` 3 .

`FileInputStream` . .

`onXXX` `Observer` . . `onNext` `onNext` , `onError` `onCompleted` . , read , `onCompleted()` , read `IOException` `onError` .

( ) . . SyncOnSubscribe .

JVM throw throw try-catch es .

.

```
SyncOnSubscribe.createStateful(
     () -> 0,
     (current, output) -> {
         output.onNext(current);
         return current + 1;
     },
     e -> { }
);
```

current 0 current 1 .

SyncOnSubscribe AsyncOnSubscribe long Observable . Observable .

```
 AsyncOnSubscribe.createStateful(
     () -> 0,
     (state, requested, output) -> {
         output.onNext(Observable.range(state, (int)requested));
         return state + 1;
     },
     e -> { }
 );
```

. . , . .

( )

Observable API ( ).

RxJava create(emitter) . .

- Emitter<T> ,
- Emitter.BackpressureMode . onBackpressureXXX . MissingBackpressureException MissingBackpressureException .

. NONE Observable onBackpressureXXX .

GUI . API addListener / removeListener .

```
Observable.create(emitter -> {
    ActionListener al = e -> {
        emitter.onNext(e);
    };

    button.addActionListener(al);

    emitter.setCancellation(() ->
        button.removeListener(al));
```

```
}, BackpressureMode.BUFFER);
```

Emitter . onNext , onError onCompleted . API (:) setCancellation (Subscription setSubscription) onError / setCancellation . onCompleted Emitter .

. CompositeSubscription CompositeSubscription .

```
Observable.create(emitter -> {
    CompositeSubscription cs = new CompositeSubscription();

    Worker worker = Schedulers.computation().createWorker();

    ActionListener al = e -> {
        emitter.onNext(e);
    };

    button.addActionListener(al);

    cs.add(worker);
    cs.add(Subscriptions.create(() ->
        button.removeActionListener(al));

    emitter.setSubscription(cs);

}, BackpressureMode.BUFFER);
```

Observable API .

```
Observable.create(emitter -> {

    someAPI.remoteCall(new Callback<Data>() {
        @Override
        public void onSuccess(Data data) {
            emitter.onNext(data);
            emitter.onCompleted();
        }

        @Override
        public void onFailure(Exception error) {
            emitter.onError(error);
        }
    });

}, BackpressureMode.LATEST);
```

. , , API , previoius . LATEST . 128 ( ) BUFFER .

: https://riptutorial.com/ko/rx-java/topic/2341/

7:

Examples

RxJava . Executor , .

Scheduler .

- (FIFO).
-

Scheduler (: delay) subscribeOn / observeOn .

Scheduler . delay . Scheduler .

subscribeOn Observable . Scheduler .

observeOn Observable . observeOn Scheduler . observeOn .

subscribeOn

```
// this lambda will be executed in the `Schedulers.io()`
Observable.fromCallable(() -> Thread.currentThread().getName())
          .subscribeOn(Schedulers.io())
          .subscribe(System.out::println);
```

observeOn

```
Observable.fromCallable(() -> "Thread -> " + Thread.currentThread().getName())
         // next tasks will be executed in the io scheduler
         .observeOn(Schedulers.io())
         .map(str -> str + " -> " + Thread.currentThread().getName())
          // next tasks will be executed in the computation scheduler
         .observeOn(Schedulers.computation())
         .map(str -> str + " -> " + Thread.currentThread().getName())
         // next tasks will be executed in the io scheduler
         .observeOn(Schedulers.newThread())
         .subscribe(str -> System.out.println(str + " -> " +
Thread.currentThread().getName()));
```

Scheduler .

```
Observable.just(1)
          // the onNext method of the delay operator will be executed in a new thread
          .delay(1, TimeUnit.SECONDS, Schedulers.newThread())
          .subscribe(System.out::println);
```

:

```
TestScheduler testScheduler = Schedulers.test();
EventBus sut = new DefaultEventBus(testScheduler);
```

```
TestSubscriber<Event> subscriber = new TestSubscriber<Event>();
sut.get().subscribe(subscriber);
sut.publish(event);
testScheduler.advanceTimeBy(1, TimeUnit.SECONDS);
```

:

```
this.poolName = schedulerFig.getIoSchedulerName();
final int poolSize = schedulerFig.getMaxIoThreads();
final BlockingQueue<Runnable> queue = new ArrayBlockingQueue<Runnable>(poolSize);
final MaxSizeThreadPool threadPool = new MaxSizeThreadPool( queue, poolSize );
this.scheduler = Schedulers.from(threadPool);
```

:

```
final Subscription subscribe = socket.webSocketObservable()
        .subscribeOn(Schedulers.io())
        .doOnNext(new Action1<RxEvent>() {
            @Override
            public void call(RxEvent rxEvent) {
                System.out.println("Event: " + rxEvent);
            }
        })
        .subscribe();
```

: https://riptutorial.com/ko/rx-java/topic/2321/

8:

.

Examples

,

Observable Subscriber .

```
Observable<Integer> integerObservable = Observable.just(1, 2, 3); // creating a simple Integer
observable
Subscriber<String> mSubscriber = new Subscriber<String>() {
    @Override
    public void onCompleted() {
        System.out.println("onCompleted called!");
    }

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

    @Override
    public void onNext(String string) {
        System.out.println("onNext called with: " + string);
    }
}; // a simple String subscriber

integerObservable
    .map(new Func1<Integer, String>() {
        @Override
        public String call(Integer integer) {
            switch (integer) {
                case 1:
                    return "one";
                case 2:
                    return "two";
                case 3:
                    return "three";
                default:
                    return "zero";
            }
        }
}).subscribe(mSubscriber);
```

.

```
onNext called with: one
onNext called with: two
onNext called with: three
onCompleted called!
```

`map` `Integer` observable `String` observable .

chained .

```
integerObservable // emits 1, 2, 3
            .map(i -> i + 10) // adds 10 to each item; emits 11, 12, 13
            .filter(i -> i > 11) // emits items that satisfy condition; 12, 13
            .last() // emits last item in observable; 13
            // unlimited operators can be added ...
            .subscribe(System.out::println); // prints 13
```

`Observable` `Subscriber` .

flatMap

`flatMap` `Observable` 0 , 1 .

`Observable` .

```
 public Observable<String> perform(int i) {
      // ...
 }

 Observable.just(1, 2, 3)
           .flatMap(i -> perform(i))
           .subscribe(result -> System.out.println("result ->" + result);
```

`flatMap` `perform` emited `perform` . `perform` (`concatMap`).

`Observable` , `flatMap` . . **Observable**

:

```
 Observable.just(1, 2, 3)
           .subscribe(i -> perform(i));
```

:

```
 Observable.just(1, 2, 3)
           .flatMap(i -> perform(i))
           .subscribe();
```

Reactivex.io : http://reactivex.io/documentation/operators/flatmap.html

`filter` .

, `filter` `false` .

:

```
List<Integer> integers = Arrays.asList(0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
```

```
Observable.from(integers)
    .filter(number -> {
        return (number  % 2 == 0);
        // odd numbers will return false, that will cause them to be filtered
    })
    .map(i -> {
        return Math.pow(i, 2); // take each number and multiply by power of 2
    })
    .subscribe(onNext -> {
         System.out.println(onNext); // print out the remaining numbers
    });
```

.

```
0.0
4.0
16.0
36.0
64.0
```

map map map map . .

.map() .

:

```
List<Integer> numbers = Arrays.asList(1, 2, 3);
Observable.from(numbers)
    .map(number -> {
        return number.toString(); // convert each integer into a string and return it
    })
    .subscribe(onNext -> {
         System.out.println(onNext); // print out the strings
    });
```

.

```
"1"
"2"
"3"
```

Observable List<Integer> . List<Integer> List<String> .subscribe String

doOnNext

Observable doOnNext doOnNext . , , .

```
Observable.range(1, 3)
    .doOnNext(value -> System.out.println("before transform: " + value))
    .map(value -> value * 2)
    .doOnNext(value -> System.out.println("after transform: " + value))
    .subscribe();
```

doOnNext

Observable Observable.empty() onCompleted .

```
Observable.empty()
    .doOnNext(item -> System.out.println("item: " + item))
    .subscribe();
```

repeat Observable .

```
Observable.just(1, 2, 3)
    .repeat()
    .subscribe(
        next -> System.out.println("next: " + next),
        error -> System.out.println("error: " + error),
        () -> System.out.println("complete")
    );
```

```
next: 1
next: 2
next: 3
next: 1
next: 2
next: 3
```

.

repeat .

```
Observable.just(1, 2, 3)
    // Repeat three times and complete
    .repeat(3)
    .subscribe(
        next -> System.out.println("next: " + next),
        error -> System.out.println("error: " + error),
        () -> System.out.println("complete")
    );
```

```
next: 1
next: 2
next: 3
next: 1
next: 2
next: 3
next: 1
next: 2
next: 3
complete
```

repeat Observable Observable Observable repeat . Observable.create Observable.create .

```
Observable.<Integer>create(subscriber -> {

    //Same as Observable.just(1, 2, 3) but with output message
    System.out.println("Subscribed");
    subscriber.onNext(1);
```

```
    subscriber.onNext(2);
    subscriber.onNext(3);
    subscriber.onCompleted();
})
        .repeat(3)
        .subscribe(
                next -> System.out.println("next: " + next),
                error -> System.out.println("error: " + error),
                () -> System.out.println("complete")
        );
```

```
Subscribed
next: 1
next: 2
next: 3
Subscribed
next: 1
next: 2
next: 3
Subscribed
next: 1
next: 2
next: 3
complete
```

repeat repeat .

```
Observable.<Integer>create(subscriber -> {
    System.out.println("Subscribed");
    subscriber.onNext(1);
    subscriber.onNext(2);
    subscriber.onNext(3);
    subscriber.onCompleted();
})
        .map(value -> value * 2) //First chain operator
        .map(value -> "modified " + value) //Second chain operator
        .repeat(3)
        .subscribe(
                next -> System.out.println("next: " + next),
                error -> System.out.println("error: " + error),
                () -> System.out.println("complete")
        );
```

```
Subscribed
next: modified 2
next: modified 4
next: modified 6
Subscribed
next: modified 2
next: modified 4
next: modified 6
Subscribed
next: modified 2
next: modified 4
next: modified 6
complete
```

repeat

map repeat Observable repeat .

```
Observable.<Integer>create(subscriber -> {
        //...
    })
    .map(value -> value * 2) //First chain operator
    .map(value -> "modified " + value) //Second chain operator
    .repeat(3)
    .subscribe(
        /*....*/
    );
```

.

```
Observable.<Integer>create(subscriber -> {
        //...
    })
    .repeat(3)
    .map(value -> value * 2) //First chain operator
    .map(value -> "modified " + value) //Second chain operator
    .subscribe(
        /*....*/
    );
```

: https://riptutorial.com/ko/rx-java/topic/2316/

9:

Examples

Observable

RxJava Observable . Observable.create . . .

Observable.just .

```
 Observable.just("Hello World").subscribe(System.out::println);
```

Observable.fromCallable .

```
Observable.fromCallable(() -> longComputation()).subscribe(System.out::println);
```

longComputation() Observable . .

Observable.defer Observable Observable.fromCallable Observable . .

```
Observable.defer(() -> {
          try {
                return Observable.just(longComputation());
          } catch(SpecificException e) {
                return Observable.error(e);
          }).subscribe(System.out::println);
```

Observables

Hot Cold .
Cold Observable () , Hot Observable .

```
/* Demonstration of a Cold Observable */
Observable<Long> cold = Observable.interval(500, TimeUnit.MILLISECONDS); // emits a long every
500 milli seconds
cold.subscribe(l -> System.out.println("sub1, " + l)); // subscriber1
Thread.sleep(1000); // interval between the two subscribes
cold.subscribe(l -> System.out.println("sub2, " + l)); // subscriber2
```

( ).

```
sub1, 0    -> subscriber1 starts
sub1, 1
```

```
sub1, 2
sub2, 0    -> subscriber2 starts
sub1, 3
sub2, 1
sub1, 4
sub2, 2
```

sub2   . Cold Observable .   .

:   .   .

Cold Observale publish Hot Observable   .

```
Observable.interval(500, TimeUnit.MILLISECONDS)
    .publish(); // publish converts cold to hot
```

publish   ConnectableObservable .

```
ConnectableObservable<Long> hot = Observable
                                    .interval(500, TimeUnit.MILLISECONDS)
                                    .publish(); // returns ConnectableObservable
hot.connect(); // connect to subscribe

hot.subscribe(l -> System.out.println("sub1, " + l));
Thread.sleep(1000);
hot.subscribe(l -> System.out.println("sub2, " + l));
```

.

```
sub1, 0  -> subscriber1 starts
sub1, 1
sub1, 2
sub2, 2  -> subscriber2 starts
sub1, 3
sub2, 3
```

sub2   sub1 .

! Observable Subscription .

```
ConnectableObservable<Long> hot = Observable
                                    .interval(500, TimeUnit.MILLISECONDS)
                                    .publish(); // same as above
Subscription subscription = hot.connect(); // connect returns a subscription object, which we
store for further use

hot.subscribe(l -> System.out.println("sub1, " + l));
Thread.sleep(1000);
hot.subscribe(l -> System.out.println("sub2, " + l));
Thread.sleep(1000);
subscription.unsubscribe(); // disconnect, or unsubscribe from subscription

System.out.println("reconnecting");
/* reconnect and redo */
subscription = hot.connect();
hot.subscribe(l -> System.out.println("sub1, " + l));
```

```
Thread.sleep(1000);
hot.subscribe(l -> System.out.println("sub2, " + l));
Thread.sleep(1000);
subscription.unsubscribe();
```

.

```
sub1, 0   -> subscriber1 starts
sub1, 1
sub1, 2
sub2, 2   -> subscriber2 starts
sub1, 3
sub2, 3
reconnecting  -> reconnect after unsubscribe
sub1, 0
...
```

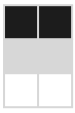
`Observable` "" .

`Hot Observable EventBus` . EventBus . RxBus .

: https://riptutorial.com/ko/rx-java/topic/1418/-

10:

- Subject <T, R> subject = AsyncSubject.create (); // AsyncSubject
- Subject <T, R> subject = BehaviorSubject.create (); // BehaviorSubject
- Subject <T, R> subject = PublishSubject.create (); // PublishSubject
- Subject <T, R> subject = ReplaySubject.create (); // ReplaySubject
- mySafeSubject = SerializedSubject (unSafeSubject); // unsafeSubject safeSubject -



`Subject` . .

Examples

RxJava `Subject` `Observable` `Observer` . `Observable` `Observer` Observable .

```
Subject<String, String> subject = PublishSubject.create();
subject.subscribe(System.out::print);
subject.onNext("Hello, World!");
```

"Hello, World!". `Subjects` .

1. `PublishSubject` `Subject` .

   ```
   Subject<String, String> subject = PublishSubject.create();
       |      |        |        |                  |
    subject<input, output>   name    = default publish subject
   ```

2. `Observer` .

   ```
   subject.subscribe(System.out::print);
   ```

   `Subject` .

3. `onNext` `Observable` .

   ```
   subject.onNext("Hello, World!");
   ```

   , `Subject` , `Subject` .

`Subject` (RxJava) .

- AsyncSubject
-
- PublishSubject
- ReplaySubject

`Subject` `SerializedSubject` . `Subject` *Observable Contract* ( ) .

:

- Dave Sexton

PublishSubject

`PublishSubject` `Observable` `Observer` .

`PublishSubject` :

```
Observable<Long> clock = Observable.interval(500, TimeUnit.MILLISECONDS);
Subject<Long, Long> subjectLong = PublishSubject.create();

clock.subscribe(subjectLong);

System.out.println("sub1 subscribing...");
subjectLong.subscribe(l -> System.out.println("sub1 -> " + l));
Thread.sleep(3000);
System.out.println("sub2 subscribing...");
subjectLong.subscribe(l -> System.out.println("sub2 -> " + l));
Thread.sleep(5000);
```

:

```
sub1 subscribing...
sub1 -> 0
sub1 -> 1
sub2 subscribing...
sub1 -> 2
sub2 -> 2
sub1 -> 3
sub2 -> 3
```

`PublishSubject` `Observable` 500 (Long) . `PublishSubject` (`clock`) (`sub1` `sub2`) .

`PublishSubject` . .

```
createClock(); // 3 lines moved for brevity. same as above example

Thread.sleep(5000); // introduces a delay before first subscribe

sub1andsub2(); // 6 lines moved for brevity. same as above example
```

:

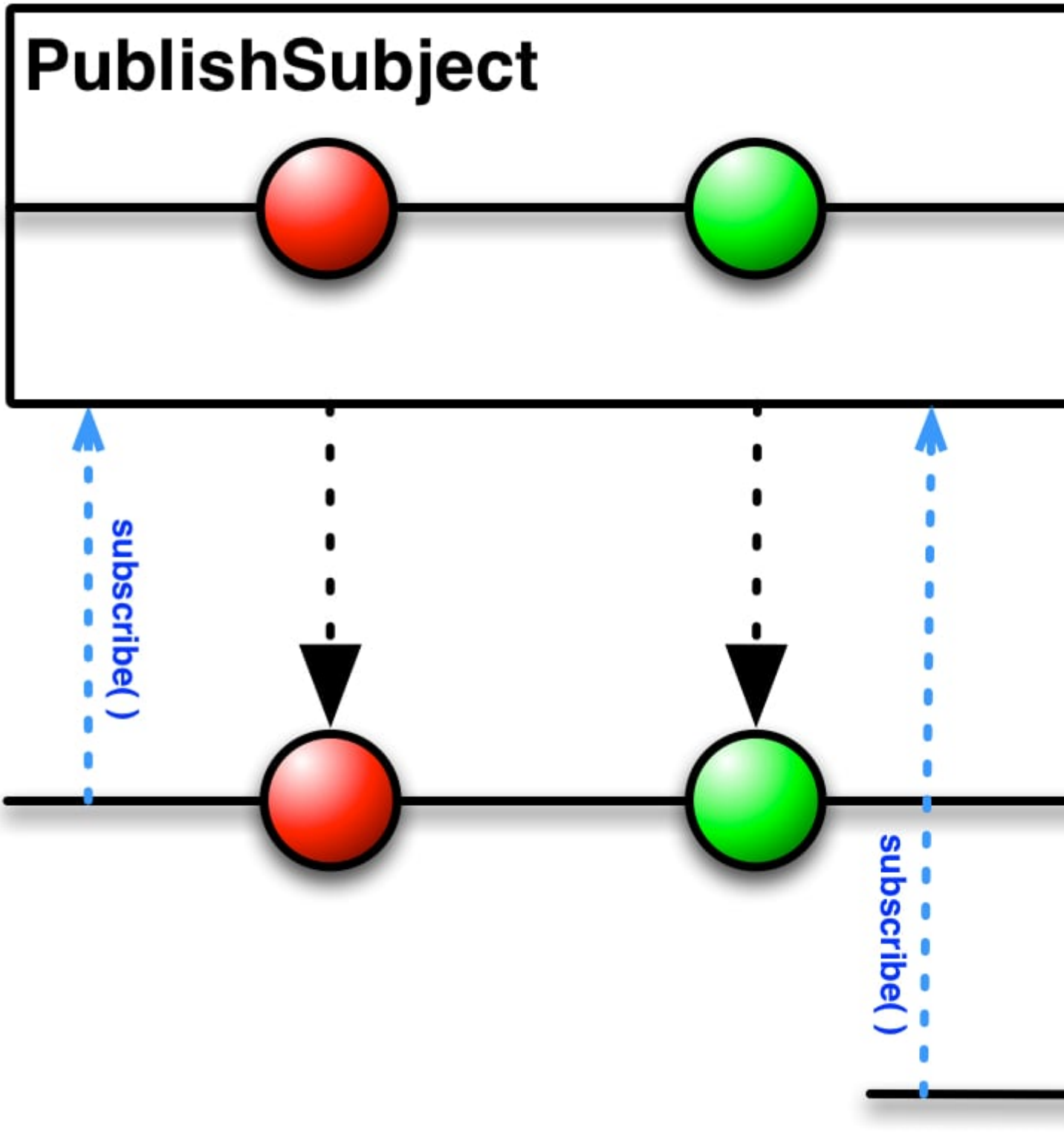
```
sub1 subscribing...
sub1 -> 10
sub1 -> 11
sub2 subscribing...
sub1 -> 12
sub2 -> 12
sub1 -> 13
```

```
sub2 -> 13
```

sub1 10 . 5 . . PublishSubject Hot Observable .

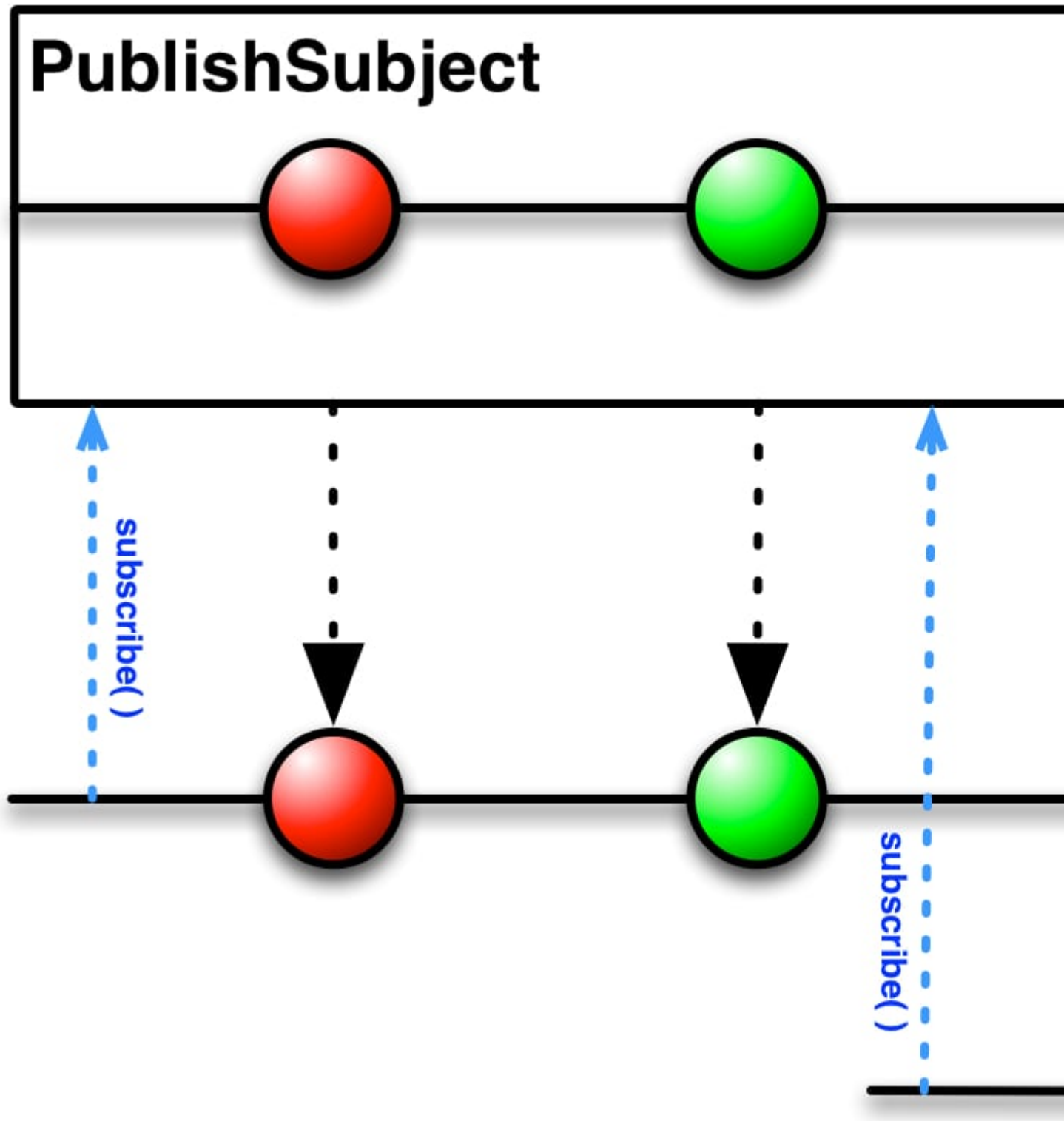
n PublishSubject ***n*** .

PublishSubject .



PublishSubject Observable onCompleted .

Observable PublishSubject Observable .



.

```
/* Dummy stock prices */
Observable<Integer> prices = Observable.just(11, 12, 14, 11, 10, 12, 15, 11, 10);
```

```
/* Your server */
PublishSubject<Integer> watcher = PublishSubject.create();
/* subscribe to listen to stock price changes and push to observers/clients */
prices.subscribe(watcher);

/* Client application */
stockWatcher = getWatcherInstance(); // gets subject
Subscription steve = stockWatcher.subscribe(i -> System.out.println("steve watching " + i));
Thread.sleep(1000);
System.out.println("steve stops watching");
steve.unsubscribe();
```

PublishSubject watcher .

:

- PublishSubject javadocs
- Thomas Nield ( )

: https://riptutorial.com/ko/rx-java/topic/3287/

S. No		Contributors
1	rx-java	Buttink, Community, dimsuz, Dmitry Avtonomov, Hans Wurst, hello_world, Omar Al Halabi, Saulius Next, Sneh Pandya, svarog, Tom
2	Android with RxJava	akarnokd, Athafoud, Daniele Segato, Eugen Martynov, Geng Jiawen, Sneh Pandya
3	RxJava2	P.J.Meisch
4	RxJava	LordRaydenMK
5		Buttink, Sir Celsius
6		akarnokd, Bartek Lipinski, Chris A, Cristian, dwursteisen, Niklas, Sebas LG
7		dwursteisen, Gal Dreiman
8		dwursteisen, hello_world, svarog, Vadeg
9		Aki K, dwursteisen, hello_world, JonesV
10		hello_world, mavHarsha