链滴

RxJava2 中的基本类型

作者：flowaters

原文链接：https://ld246.com/article/1529996063458

来源网站：链滴

背景

RxJava2中的类型，除了Observable和Flowable外，还有Single、Completable、Maybe三种类型
这些类型有什么区别呢?

简介

类型	含义
io.reactivex.Observable	0..N flows, supporting Reactive-Streams and backpressure
io.reactivex.Flowable	0..N flows, no backpressure
io.reactivex.Single	a flow of exactly 1 item or an error
io.reactivex.Completable	a flow without items but only a completion or error signal
io.reactivex.Maybe	a flow with no items, exactly one item or an error

示例

下面的示例，来演示如何逐步简化代码。

示例一: Flowable

先来一个最基础的Flowable示例。

```
import java.io.IOException;

import org.apache.http.client.fluent.Request;
import org.reactivestreams.Subscription;

import io.reactivex.BackpressureStrategy;
import io.reactivex.Flowable;
import io.reactivex.FlowableEmitter;
import io.reactivex.FlowableOnSubscribe;
import io.reactivex.FlowableSubscriber;

public class HelloFlowable {
    public static void main(String[] args) {

        Flowable.create(new FlowableOnSubscribe<String>() {

            @Override
            public void subscribe(FlowableEmitter<String> emitter) throws Exception {
                // 模拟无限的上流数据
                for (int i = 1;; ++i) {
                    emitter.onNext("http://www.abeffect.com/" + i);
                }
            }
        }, BackpressureStrategy.BUFFER).subscribe(new FlowableSubscriber<String>() {
```

```
            @Override
            public void onComplete() {

            }

            @Override
            public void onError(Throwable t) {

            }

            @Override
            public void onNext(String s) {
                try {
                    System.out.println(Request.Get(s).execute().returnResponse().getStatusLine().getS
atusCode());
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            @Override
            public void onSubscribe(Subscription s) {
                s.request(Long.MAX_VALUE);
            }
        });
    }
}
```

示例二: Consumer

如果在FlowableSubscriber中只关心onNext方法的话，可以使用Consumer

```
import java.io.IOException;

import org.apache.http.client.fluent.Request;

import io.reactivex.BackpressureStrategy;
import io.reactivex.Flowable;
import io.reactivex.FlowableEmitter;
import io.reactivex.FlowableOnSubscribe;
import io.reactivex.functions.Consumer;

public class HelloConsumer {
    public static void main(String[] args) {

        Flowable.create(new FlowableOnSubscribe<String>() {

            @Override
            public void subscribe(FlowableEmitter<String> emitter) throws Exception {
                // 模拟无限的上流数据
                for (int i = 1;; ++i) {
                    emitter.onNext("http://www.abeffect.com/" + i);
                }
```

```
            }
        }, BackpressureStrategy.BUFFER).subscribe(new Consumer<String>() {

            @Override
            public void accept(String s) throws Exception {
                try {
                    System.out.println(Request.Get(s).execute().returnResponse().getStatusLine().getS
atusCode());
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

从subscribe()中，可以看出其可用的方法有:

- subscribe(Consumer<? super T>)
- subscribe(Consumer<? super T>, Consumer<? super Throwable>)
- subscribe(Consumer<? super T>, Consumer<? super Throwable>, Action)
- subscribe(Consumer<? super T>, Consumer<? super Throwable>, Action, Consumer<? supe Subscription>)
- subscribe(Subscriber<? super T>)
- subscribe(FlowableSubscriber<? super T>)

可以根据需要，逐步添加onNext, onError, onComplete, onSubscribe参数。

同时，可以支持双参数或者多参数类型

- BiConsumer<T1, T2>: 双参数类型
- Consumer<Obejct[]>: 多参数类型

Observable和Flowable的对比, 来自[1]

	Observable	Flowable
元素个数	不超过1000个	超过10+个
开销	开销低	开销高

示例三: Single

如果只有一个onNext事件时，即唯一事件(流)，则可以用Single。

```
import io.reactivex.Single;
import io.reactivex.SingleEmitter;
import io.reactivex.SingleObserver;
import io.reactivex.SingleOnSubscribe;
import io.reactivex.disposables.Disposable;
```

```
public class HelloSingle {
    public static void main(String[] args) {
        Single.create(new SingleOnSubscribe<String>() {

            @Override
            public void subscribe(SingleEmitter<String> singleEmitter) throws Exception {
                singleEmitter.onSuccess("Hello Single");
            }
        }).subscribe(new SingleObserver<String>() {

            @Override
            public void onError(Throwable t) {
            }

            @Override
            public void onSubscribe(Disposable d) {
            }

            @Override
            public void onSuccess(String s) {
                System.out.println(s);
            }
        });
    }
}
```

结果

```
Hello Single
```

BiConsumer

这边也可以直接用BiConsumer

```
import io.reactivex.Single;
import io.reactivex.SingleEmitter;
import io.reactivex.SingleOnSubscribe;
import io.reactivex.functions.BiConsumer;

public class HelloSingle2 {
    public static void main(String[] args) {
        Single.create(new SingleOnSubscribe<String>() {

            @Override
            public void subscribe(SingleEmitter<String> singleEmitter) throws Exception {
                singleEmitter.onSuccess("Hello Single");
            }
        }).subscribe(new BiConsumer<String, Throwable>() {

            @Override
            public void accept(String s, Throwable t) throws Exception {
                System.out.println(s);
            }
```

```
        });
    }
}
```

结果

```
Hello Single
```

Consumer

```
import io.reactivex.Single;
import io.reactivex.SingleEmitter;
import io.reactivex.SingleOnSubscribe;
import io.reactivex.functions.Consumer;

public class HelloSingle2 {
    public static void main(String[] args) {
        Single.create(new SingleOnSubscribe<String>() {

            @Override
            public void subscribe(SingleEmitter<String> singleEmitter) throws Exception {
                singleEmitter.onSuccess("Hello Single");
            }
        }).subscribe(new Consumer<String>() {

            @Override
            public void accept(String s) throws Exception {
                System.out.println(s);
            }
        });
    }
}
```

结果

```
Hello Single
```

类型转换

Single类型也可以转换成其它类型

- Single.toFlowable()
- Single.toObservable()
- Single.toMaybe()

示例四: Completable

如果也不关心onNext事件，只关心onComplete和onError，则直接使用Completable。

```
import io.reactivex.Completable;
import io.reactivex.CompletableEmitter;
import io.reactivex.CompletableObserver;
```

```
import io.reactivex.CompletableOnSubscribe;
import io.reactivex.disposables.Disposable;

public class HelloCompletable {
    public static void main(String[] args) {
        Completable.create(new CompletableOnSubscribe() {

            @Override
            public void subscribe(CompletableEmitter e) throws Exception {
                e.onComplete();
            }
        }).subscribe(new CompletableObserver() {

            @Override
            public void onSubscribe(Disposable d) {
            }

            @Override
            public void onError(Throwable t) {
            }

            @Override
            public void onComplete() {
                System.out.println("Complete");
            }
        });
    }
}
```

示例五: Maybe

Single是发送一个事件, Completable是不发送任何事件。

如果不知道会不会发送事件呢？即有可能发送一个事件，也有可能不发送任何事件。

那这种情况对应的就是Maybe。

发送事件时，会响应onSuccess，不发送事件时，会响应onComplete，二者只有一个会响应。

```
import io.reactivex.Maybe;
import io.reactivex.MaybeEmitter;
import io.reactivex.MaybeObserver;
import io.reactivex.MaybeOnSubscribe;
import io.reactivex.disposables.Disposable;

public class HelloMaybe {
    public static void main(String[] args) {
        Maybe.create(new MaybeOnSubscribe<String>() {

            @Override
            public void subscribe(MaybeEmitter<String> e) throws Exception {
                e.onSuccess("Success");
                // 第二条是不会执行的
```

```
                e.onComplete();
            }
        }).subscribe(new MaybeObserver<String>() {

            @Override
            public void onComplete() {
                System.out.println("Complete");
            }

            @Override
            public void onError(Throwable t) {
            }

            @Override
            public void onSubscribe(Disposable d) {
            }

            @Override
            public void onSuccess(String s) {
                System.out.println(s);
            }
        });
    }
}
```

结果

```
Success
```

参考

1. RxJava 2.x 使用详解(一) 快速入门: 此文有一个系列，推荐阅读.