RxJava 概念全面介绍与上手建议

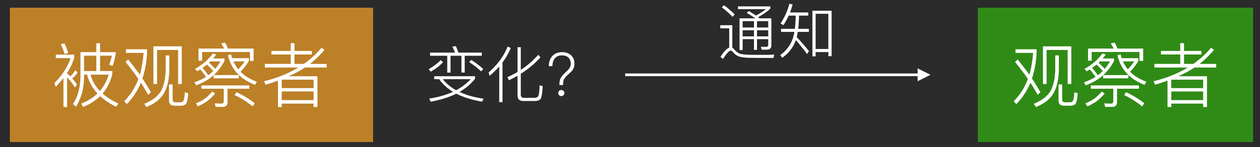
扔物线（朱凯） @ hencoder.com

RxJava

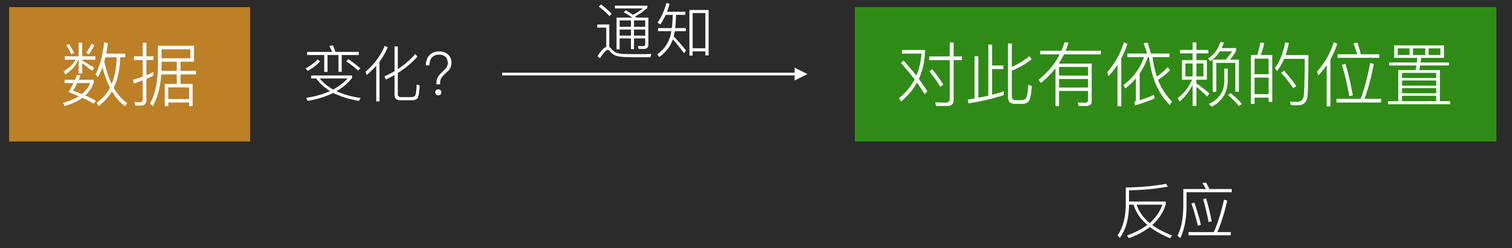
- Rx：Reactive Extension
  - Reactive -> Reactive Programming -> 观察者模式
  - Extension

Reactive Programming vs 观察者模式

- 观察者模式：观察者对被观察者监听，当有变化时立即通知



- Reactive Programming：当数据发生变化时，所有对它有依赖的位置立即相应地发生反应



RxJava

- Rx：Reactive Extension
  - Reactive -> Reactive Programming -> 观察者模式
  - Extension
    - 不仅支持事件序列，还支持数据流
    - 通过操作符对事件对象进行中间处理
    - 便捷的线程操作

事件序列 / 数据流

- 事件：不可预知、动态的 -> 离散
  - 用户点击
  - 服务器推送
  - HTTP / HTTPS 网络请求
- 数据流：现成的、静态的 -> 连续
  - 处理静态数据（字符串、Bitmap）
  - 逐行读取本地文件

用户点击

```
RxView.clicks(view)
        .subscribe(new Consumer<Object>() {
            @Override
            public void accept(@NonNull Object o) {
                // 处理点击事件
                ......
            }
        });
```

事件序列 / 数据流

- 事件：不可预知、动态的 -> 离散
  - 用户点击
  - 服务器推送
  - HTTP / HTTPS 网络请求
- 数据流：现成的、静态的 -> 连续
  - 处理静态数据（字符串、Bitmap）
  - 逐行读取本地文件

服务器推送

```
pushObservable
        .subscribe(new Consumer<Push>() {
            @Override
            public void accept(Push push) {
                // 显示推送
                showPush(push);
            }
        });
```

事件序列 / 数据流

- 事件：不可预知、动态的 -> 离散
  - 用户点击
  - 服务器推送
  - HTTP / HTTPS 网络请求
- 数据流：现成的、静态的 -> 连续
  - 处理静态数据（字符串、Bitmap）
  - 逐行读取本地文件

HTTP / HTTPS

```
api.getItems()
        .subscribe(new Consumer<List<Images>>() {
            @Override
            public void accept(List<Images> images) {
                showImages(images);
            }
        });
```

事件序列 / 数据流

- 事件：不可预知、动态的 -> 离散
  - 用户点击
  - 服务器推送
  - HTTP / HTTPS 网络请求
- 数据流：现成的、静态的 -> 连续
  - 处理静态数据（字符串、Bitmap）
  - 逐行读取本地文件

处理静态数据

```
Bitmap[] bitmaps = ...;

Observable.fromArray(bitmaps)
        .subscribe(new Consumer<Bitmap>() {
            @Override
            public void accept(Bitmap bitmap) {
                // 加水印
                addWatermark(bitmap);
            }
        });
```

事件序列 / 数据流

- 事件：不可预知、动态的 -> 离散
  - 用户点击
  - 服务器推送
  - HTTP / HTTPS 网络请求
- 数据流：现成的、静态的 -> 连续
  - 处理静态数据
  - 逐行读取本地文件

RxJava

- Rx：Reactive Extension
  - Reactive -> Reactive Programming -> 观察者模式
  - Extension
    - 不仅支持事件序列，还支持数据流
    - 通过操作符对事件对象进行中间处理
    - 便捷的线程操作

操作符

```
api.getItems(100, 1)
        .subscribe(new Consumer<List<Item>>() {
            @Override
            public void accept(List<Item> items) {
                // 存入数据库
                Database.getInstance().save(items);
                // 显示图片
                showImages(items);
            }
        });
```

操作符

```
api.getItems(100, 1)
        .doOnNext(new Consumer<List<Item>>() {
            @Override
            public void accept(List<Item> items) {
            }
        })
        .subscribe(new Consumer<List<Item>>() {
            @Override
            public void accept(List<Item> items) {
                // 存入数据库
                Database.getInstance().save(items);
                // 显示图片
                showImages(items);
            }
        });
```

操作符

```
api.getItems(100, 1)
        .doOnNext(new Consumer<List<Item>>() {
            @Override
            public void accept(List<Item> items) {
                // 存入数据库
                Database.getInstance().save(items);
            }
        })
        .subscribe(new Consumer<List<Item>>() {
            @Override
            public void accept(List<Item> items) {
                // 显示图片
                showImages(items);
            }
        });
```

操作符

- 对事件对象进行中间处理
- 优势：
  - 将操作拆散，简化代码逻辑 -> 代码对复杂逻辑的承受能力增强

操作符

```
api.getItems(100, 1)
        .doOnNext( items -> 存到数据库 )
        .subscribe(new Consumer<List<Item>>() {
            @Override
            public void accept(List<Item> items) {
                // 显示图片
                showImages(items);
            }
        });
```

操作符

```
api.getItems(100, 1)
        .doOnNext( items -> 存到数据库 )
        .map(new Function<List<Item>>, List<Image>>() {
            @Override
            public List<Image> apply(@NonNull List<Item> items) {
                return getImagesFromItems(items);
            }
        })
        .subscribe(new Consumer<List<Item>>() {
            @Override
            public void accept(List<Item> items) {
                // 显示图片
                showImages(items);
            }
        });
```

操作符

```
api.getItems(100, 1)
        .doOnNext( items -> 存到数据库 )
        .map(new Function<List<Item>>, List<Image>>() {
            @Override
            public List<Image> apply(@NonNull List<Item> items) {
                return getImagesFromItems(items);
            }
        })
        .subscribe(new Consumer<List<Image>>() {
            @Override
            public void accept(List<Image> images) {
                // 显示图片
                showImages(images);
            }
        });
```

操作符

```
RxView.clicks(view)
        .subscribe(new Consumer<Object>() {
            @Override
            public void accept(@NonNull Object o) {
                // 处理点击事件
                ......
            }
        });
```

操作符

```
RxView.clicks(view)
        .throttleFirst(1, TimeUnit.SECONDS) // 去抖动
        .subscribe(new Consumer<Object>() {
            @Override
            public void accept(@NonNull Object o) {
                // 处理点击事件
                ......
            }
        });
```

操作符

```
api.getToken()
        .flatMap(new Function<Token, Observable<List<Item>>>() {
            @Override
            public Observable<List<Items>> apply(Token token) {
                return api.getItems();
            }
        })
        ....
        .subscribe(....);
```

操作符

- 对事件对象进行中间处理
- 优势：
  - 将操作拆散，简化代码逻辑 -> 代码对复杂逻辑的承受能力增强
  - 把耗时操作放在后台线程，不卡界面

RxJava

- Rx：Reactive Extension
  - Reactive -> Reactive Programming -> 观察者模式
  - Extension
    - 不仅支持事件序列，还支持数据流
    - 通过操作符对事件对象进行中间处理
    - 便捷的线程操作

线程操作

- subscribeOn()
- observeOn()

线程操作

```
api.getItems()
        .doOnNext( items -> 存到数据库 )
        .map(items -> 转换成 images)
        .subscribe(....);
```

线程操作

```
api.getItems()
        .doOnNext( items -> 存到数据库 )
        .map(items -> 转换成 images)
        .subscribeOn(schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(....);
```

RxJava

- Rx：Reactive Extension
  - Reactive -> Reactive Programming -> 观察者模式
  - Extension
    - 不仅支持事件序列，还支持数据流
    - 通过操作符对事件对象进行中间处理
    - 便捷的线程操作

常见问题 / 常错问题解答

- Observable / Flowable? -> Backpressure?

Backpressure

- 听不懂的含义：上游生产速度大于下游处理速度，导致下游处理不及的状况，被称为 Backpressure <- 来自工程概念 back pressure（背压）
- 问题 1：听起来好像很常见？ -> 并不常见，确切说是非常罕见
- 问题 2：如果遇到该怎么办呢？ -> 丢弃 -> 「你在逗我？」
- 根本问题：什么时候用 Observable，什么时候用 Flowable?

Backpressure

- 听得懂的含义：对于可丢弃的事件，上游生产过快导致事件堆积，当堆积到超出 buffer 上限，就叫做 Backpressure 出现。
- 处理方案：
  1. 丢弃新事件（具体的丢弃方案可能有细分，但原则是丢弃）
  2. ~~不丢弃，继续堆积（忽略 Backpressure，等价于用 Observable）~~

Backpressure

- 适合支持 Backpressure 的情况：
  - 在线直播视频流
  - Log
  - 用户请求数超限丢弃（服务端）
- 关键点：事件的产生不可控、可丢弃

常见问题 / 常错问题解答

- Observable / Flowable? -> Backpressure?
- Reactive Programming? -> 数据的改变会立即传达到依赖它的位置
- Functional Programming? -> 一种用于计算的编程范式
- Functional Reactive Programming? -> ？ ？ ？

上手建议

- 先了解最基本概念：Observable、Observer、subscribe() (Flowable、Subscriber、subscribe())
- 先了解无 Backpressue 的：Observable、Observer
- 先选一个库来让上游自动生产（RxBinding、Retrofit）
- 一定要动手

问题?