RxJava

1. 观察者模式

观察者模式面向的需求是：A 对象（观察者）对 B 对象（被观察者）的某种变化高度敏感，需要在 B 变化的一瞬间做出反应。

在程序中，观察者多是采用注册(**Register**)或者称为订阅(**Subscribe**)的方式，告诉被观察者：我需要你的某某状态，你要在它变化的时候通知我。Android 开发中一个比较典型的例子是点击监听器 `OnClickListener`。

OnClickListener 的模式大致如下图：

通过 `setOnClickListener()` 方法，`Button` 持有 `OnClickListener` 的引用；当用户点击时，`Button` 自动调用 `OnClickListener` 的 `onClick()` 方法。

把这张图中的概念抽象出来（`Button` -> 被观察者、`OnClickListener` -> 观察者、`setOnClickListener()` -> 订阅，`onClick()` -> 事件），就由专用的观察者模式（例如只用于监听控件点击）转变成了通用的观察者模式。如下图：

RxJava 作为一个工具库，使用的就是通用形式的观察者模式。

2. 四个基本概念

- `Observable` 可观察者，即被观察者
- `Observer` 观察者
- `subscribe` 订阅
- `事件`

  `Observable` 和 `Observer` 通过 `subscribe()` 方法实现订阅关系，从而 `Observable` 可以在需要的时候发出事件来通知 `Observer`。

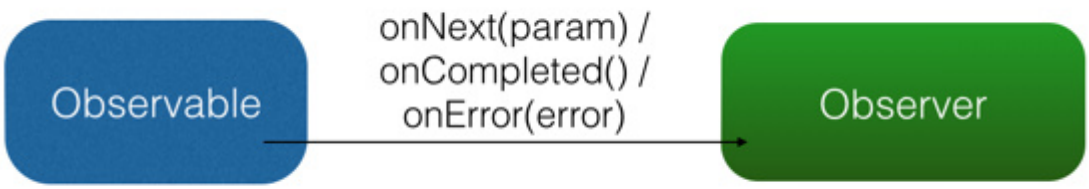
3. 三个重要回调

与传统观察者模式不同，RxJava 的事件回调方法除了普通事件 `onNext()`（相当于 `onClick()` / `onEvent()`）之外，还定义了两个特殊的事件：`onCompleted()` 和 `onError()`。

- onNext()：相当于onClick() 事件
- onCompleted：事件队列完结。RxJava 不仅把每个事件单独处理，还会把它们看做一个队列。RxJava 规定，当不会再有新的 `onNext()` 发出时，需要触发 `onCompleted()` 方法作为标志。
- onError：事件队列异常。在事件处理过程中出异常时，`onError()` 会被触发，同时队列自动终止，不允许再有事件发出。

> 在一个正确运行的事件序列中, `onCompleted()` 和 `onError()` 有且只有一个，并且是事件序列中的最后一个。需要注意的是，`onCompleted()` 和 `onError()` 二者也是互斥的，即在队列中调用了其中一个，就不应该再调用另一个。

RxJava 的观察者模式大致如下图：



4. 基本用法

1> 创建 Observer 监听器

Observer 即观察者，它决定事件触发的时候将有怎样的行为。 RxJava 中的 `Observer` 接口的实现方式：

```
Observer<String> observer = new Observer<String>() {
    @Override
    public void onNext(String s) {
        Log.d(tag, "Item: " + s);
    }

    @Override
    public void onCompleted() {
        Log.d(tag, "Completed!");
    }

    @Override
    public void onError(Throwable e) {
        Log.d(tag, "Error!");
    }
};
```

除了 `Observer` 接口之外，RxJava 还内置了一个实现了 `Observer` 的抽象类：`Subscriber`。 `Subscriber` 对 `Observer` 接口进行了一些扩展，但他们的基本使用方式是完全一样的：

```
Subscriber<String> subscriber = new Subscriber<String>() {
    @Override
    public void onNext(String s) {
        Log.d(tag, "Item: " + s);
    }

    @Override
    public void onCompleted() {
        Log.d(tag, "Completed!");
    }

    @Override
    public void onError(Throwable e) {
        Log.d(tag, "Error!");
    }
};
```

> 不仅基本使用方式一样，实质上，在 RxJava 的 subscribe 过程中，`Observer` 也总是会先被转换成一个 `Subscriber` 再使用。如果只是想使用基本功能，选择 `Observer` 和 `Subscriber` 是完全一样的。

它们的区别：

1. `onStart()`: 这是 `Subscriber` 增加的方法。它会在 subscribe 刚开始，而事件还未发送之前被调用，可以用于做一些准备工作，例如数据的清零或重置。这是一个可选方法，默认情况下它的实现为空。

   需要注意的是，如果对准备工作的线程有要求（例如弹出一个显示进度的对话框，这必须在主线程执行），`onStart()` 就不适用了。因为它总是在 subscribe 所发生的线程被调用，而不能指定线程。要在指定的线程来做准备工作，可以使用 `doOnSubscribe()` 方法。

2. `unsubscribe()`: 这是 `Subscriber` 所实现的另一个接口 `Subscription` 的方法，用于取消订阅。在这个方法被调用后，`Subscriber` 将不再接收事件。一般在这个方法调用前，可以使用 `isUnsubscribed()` 先判断一下状态。

`subscribe()` 之后， `Observable` 会持有 `Subscriber` 的引用，这个引用如果不能及时被释放，将有内存泄露的风险。

原则：要在不再使用的时候尽快在合适的地方（例如 `onPause()` `onStop()` 等方法中）调用 `unsubscribe()` 来解除引用关系，以避免内存泄露的发生。

2> 创建 Observable

Observable 即被观察者，它决定什么时候触发事件以及触发怎样的事件。 RxJava 使用 `create()` 方法来创建一个 Observable ，并为它定义事件触发规则：

```
Observable observable = Observable.create(new Observable.OnSubscribe<String>() {
    // 当 observable 被订阅的时候，call() 方法被触发
    @Override
    public void call(Subscriber<? super String> subscriber) {
        // 此处模拟触发的事件
        subscriber.onNext("Hello");
        subscriber.onNext("Hi");
        subscriber.onNext("Aloha");
        subscriber.onCompleted();
    }
});
```

这里传入了一个 `OnSubscribe` 对象作为参数。 `OnSubscribe` 会被存储在返回的 `Observable` 对象中，它的作用相当于一个计划表，当 `Observable` 被订阅的时候， `OnSubscribe` 的 `call()` 方法会自动被调用，事件序列就会依照设定依次触发（对于上面的代码，就是观察者 `Subscriber` 将会被调用三次 `onNext()` 和一次 `onCompleted()` ）。

这样，由被观察者调用了观察者的回调方法，就实现了由被观察者向观察者的事件传递，即观察者模式。

3> Subscribe (订阅)