RxJava

By 李侦跃

微博&知乎：@hi大头鬼hi

RxJava是什么？

- 扩展的观察者模式
- 异步数据流处理

观察者模式

- Observable发出事件
- Subscriber订阅事件

```
bus.post(new AnswerEvent(42));

@Subscribe
public void onAnswer(AnswerEvent event) {

}
```

```
Observable observable = Observable.create(new
Observable.OnSubscribe<String>() {
    @Override
    public void call(Subscriber<? super String>
subscriber) {
        subscriber.onNext("a");
        subscriber.onCompleted();
    }
});


observable.subscribe(new Subscriber() {
    …
});
```

观察者模式的不足

- 不知道事件何时结束
- 缺少错误通知机制

RxJava的改进

- onComplete方法通知Subscriber事件结束
- onError方法通知Subscriber出错

```
Observable.just("hello")
    .subscribe(new Subscriber<String>() {
        @Override
        public void onCompleted() {
        }

        @Override
        public void onError(Throwable e) {
        }

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

unsubscribe

```
Subscription subscription =
observable.subscribe(System.out::println)
```

```
subscription.unsubscribe();
```

异步数据流处理

- 指定被观察者生产数据所在的线程
- 指定订阅者接收数据所在的线程
- 强大的数据处理功能

```
Observable.just("hello")
        //指定被观察者在新的线程中生产数据
        .subscribeOn(Schedulers.newThread())
        //指定观察者在UI主线程接收数据
        .observeOn(AndroidSchedulers.mainThread())
        //因为上面指定了在UI线程接收数据，
        //所以这里可以做更新UI的事情
        .subscribe(System.out::println);
```

Scheduler

默认情况下RxJava中生产者和订阅者都是在当前线程下运行

Scheduler就是方便切换生产者和订阅者执行的线程

RxJava默认提供了一些内置的Scheduler，方便针对不同的任务选择

Schedulers

- Schedulers.immediate()
- Schedulers.trampoline()
- Schedulers.newThread()
- Schedulers.computation()
- Schedulers.io()
- AndroidSchedulers.mainThread()

数据处理-Operator

- 方便的对数据进行各种变换处理
- 内置丰富的operators
- 自定义operator

```java
Observable.just(1, 2, 3)
        .map(new Func1<Integer, String>() {
            @Override
            public String call(Integer integer) {
                return integer.toString();
            }
        })
        .subscribe(new Action1<String>() {
            @Override
            public void call(String s) {
                System.out.println(s);
            }
        });
```

```java
Observable.just(1, 2, 3)
        .map(integer -> integer.toString())
        .subscribe(System.out::println);
```

需求

- 服务端返回一段字符串数组，每个字符串都是数字
- 将数组每个元素转换成数字
- 过滤掉小于1的元素
- 去重
- 取前三个元素
- 累加求和

```
Observable
    .just("1", "2", "2", "3", "4", "5")
    .map(Integer::parseInt)
    .filter(s -> s > 1)
    .distinct()
    .take(3)
    .reduce((integer, integer2) ->
            integer.intValue() + integer2.intValue(
    )
    .subscribe(System.out::println);//9
```

operators

- 创建Observable create just
- 变换Observable map flatMap
- 过滤Observable filter first last
- 合并Observable merge zip
- 错误处理 catch
- 条件过滤 all skipUtil takeWhile
- 聚集函数 average reduce count

自定义operator

```
Observable.just("a")
    .lift(subscriber -> {
        return new Subscriber<String>() {
            @Override
            public void onCompleted() {
                subscriber.onCompleted();
            }

            @Override
            public void onError(Throwable e) {
                subscriber.onError(e);
            }

            @Override
            public void onNext(String s) {
                subscriber.onNext(1);
            }
        };
})
.map(i -> i)
.subscribe(System.out::println);//1
```

Android中的应用

- Retrofit
- RxAndroid
- RxBinding
- rx-preferences
- sqlbrite
- RxLifecycle

Retrofit

```
@GET("/story/{id}")
Observable<NewsDetail> getNewsDetailObservable(@Path("id") long id);
```

RxAndroid

目前只有一个功能

AndroidSchedulers.mainThread()

RxBinding

使用RxJava对Android UI控件进行了封装

```
RxView.clicks(findViewById(R.id.btn_throttle))
    .throttleFirst(1, TimeUnit.SECONDS)
    .subscribe(aVoid -> {
        System.out.println("click");
    });
```

RxPreference

使用RxJava封装SharePreference

```
SharedPreferences preferences =
    PreferenceManager.getDefaultSharedPreferences(context);

RxSharedPreferences rxPreferences =
    RxSharedPreferences.create(preferences);

Preference<String> username =
    rxPreferences.getString(“username");

username.asObservable()
    .subscribe(new Action1<String>() {
        @Override public void call(String username) {
        Log.d(TAG, "Username: " + username);
    }
}
```

RxBinding结合RxPreference

```
SharedPreferences preferences =
    PreferenceManager.getDefaultSharedPreferences(this);

RxSharedPreferences rxPreferences =
    RxSharedPreferences.create(preferences);

Preference<Boolean> checked =
    rxPreferences.getBoolean("checked", true);

CheckBox checkBox = (CheckBox) findViewById(R.id.cb_test);
RxCompoundButton.checkedChanges(checkBox)
        .subscribe(checked.asAction());

checked.asObservable()
    .subscribe(aBoolean -> {
        System.out.println("-----------------checked: " +
aBoolean);
    });
```

RxLifecycle

根据Activity或者Fragment的声明周期，在指定的事件中结束数据流

```
myObservable
    .compose(RxLifecycle.bindUntilActivityEvent(lifecycle,
ActivityEvent.DESTROY))
    .subscribe();
```

```
myObservable
    .compose(RxLifecycle.bindActivity(lifecycle))
    .subscribe();
```

RxJava适用场景

- 出现多层嵌套回调(Callback hell)
- 复杂的数据处理
- 响应式UI
- 复杂的线程切换

多层嵌套回调

```
getToken(new Callback<String>() {
    @Override
    public void success(String token) {
        getUser(userId, new Callback<User>() {
            @Override
            public void success(User user) {
                userView.setUser(user);
            }

            @Override
            public void failure(RetrofitError error) {
                // Error handling
            }
        };
    }

    @Override
    public void failure(RetrofitError error) {
        // Error handling
    }
});
```

```
getToken("username", "password")
    .flatMap(token -> getUser(token))
    .subscribe(user -> {
        System.out.println("user: " + user.toJson());
    });
```

复杂的数据处理

```
Observable
        .just("1", "2", "2", "3", "4", "5")
        .map(Integer::parseInt)
        .filter(s -> s > 1)
        .distinct()
        .take(3)
        .reduce((integer, integer2) ->
                integer.intValue() + integer2.intValue())
        .subscribe(System.out::println);//9
```

响应式UI

```
CheckBox checkBox = (CheckBox)
findViewById(R.id.cb_test);

RxCompoundButton
      .checkedChanges(checkBox)
      .subscribe(checked.asAction());
```

复杂的线程切换

```
Observable<String> observable1 =
    createObservable1()
    .subscribeOn(Schedulers.newThread());

Observable<String> observable2 =
    createObservable2()
    .subscribeOn(Schedulers.io());

Observable
    .concat(observable1, observable2)
    .subscribeOn(Schedulers.computation())
    .subscribe(System.out::println);
```

RxJava缺点

- 库体积稍大（900K）
- 入门不太容易（函数式）
- 大量使用匿名对象，容易造成内存泄露（及时unsubsc
  者使用RxBinding库）
- Java6中不能适用lambda简化代码（Retrolambda & Gr
  插件）

参考

- 官方网站：http://reactivex.io/
- 扔物线的：给 Android 开发者的 RxJava 详解（http://gank.io/post/560e15be2dca930e00da1083）
- 深入浅出RxJava系列 （http://blog.csdn.net/lzyzsd/article/details/41833541）

应用案例

- Square
- SoundCloud
- Flipboard
- 天天动听
- 薄荷

Thanks

Q&A