赵彦军
做一只快乐的程序猿！！

RxJava 和 RxAndroid 一 (基础)

1、RxJava 项目地址

https://github.com/ReactiveX/RxJava

2、RxAndroid 项目地址

https://github.com/ReactiveX/RxAndroid

3、RxJava 和 RxAndroid 的关系

RxAndroid是RxJava的一个针对Android平台的扩展，主要用于 Android 开发

4、RxJava和EventBus的区别?

https://www.zhihu.com/question/32179258/answer/54989242

5、RxAndroid的使用方法

compile 'io.reactivex:rxandroid:1.2.0'

6、如何查看RxAndroid最新版本?

http://search.maven.org/#search%7Cga%7C1%7Ca%3A%22rxandroid%22

7、RxAndroid具体使用方法

http://gank.io/post/560e15be2dca930e00da1083#toc_14

http://blog.csdn.net/theone10211024/article/details/50435325

http://huxian99.github.io/tags/RxJava/

https://github.com/mcxiaoke/RxDocs

8、创建观察者

```
1  package lib.com.myapplication;
2
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5
6  import rx.Observer;
7  import rx.Subscriber;
8
9  public class Main2Activity extends AppCompatActivity {
10
11     @Override
```

昵称：赵彦军
园龄：3年2个月
粉丝：89
关注：10
+加关注

我的标签

android(30)
赵彦军(26)
zhaoyanjun(25)
java(10)
IO流(8)
android studio(7)
markdown(6)
rxandroid(5)
rxbinding(5)
rxjava(5)
更多

随笔分类(349)

Android(169)
Android Studio(100)
Android 错误(4)
Android 动画
android 框架(6)
CVS (SVN 和 Git)(19)
IOS(7)
IOS 错误(1)
Java(23)
python(1)
产品与设计(4)
生活(3)
网站与服务器(10)
运营与测试(2)

随笔档案(210)

2017年3月 (2)
2017年2月 (9)
2017年1月 (2)
2016年12月 (2)
2016年11月 (6)
2016年10月 (7)
2016年9月 (7)
2016年8月 (7)
2016年7月 (4)
2016年6月 (9)
2016年5月 (8)
2016年4月 (11)
2016年3月 (20)
2016年2月 (8)
2016年1月 (8)
2015年12月 (5)
2015年11月 (6)
2015年10月 (6)
2015年9月 (5)
2015年8月 (12)
2015年7月 (15)
2015年6月 (31)
2015年5月 (19)
2015年4月 (1)

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);

        //创建观察者 2 种方法
        Observer<String> observer = new Observer<String>() {
            @Override
            public void onCompleted() {

            }

            @Override
            public void onError(Throwable e) {

            }

            @Override
            public void onNext(String s) {

            }
        } ;

        // Subscriber 继承 Observer ，对Observer类做了扩展
        Subscriber<String> subscriber = new Subscriber<String>() {
            @Override
            public void onCompleted() {

            }

            @Override
            public void onError(Throwable e) {

            }

            @Override
            public void onNext(String s) {

            }

        } ;


    }
}
```

- 从上文可以看到，Subscriber继承Observer, 只是 Subscriber对Observer做了一些扩展。Subscriber的使用和Observer完全一样。
- Subscriber 多了一个 onStart 方法
- `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，具体可以在后面的文中看到。

```
// Subscriber 继承 Observer ，对Observer类做了扩展
       Subscriber<String> subscriber = new Subscriber<String>() {
           @Override
           public void onCompleted() {

           }

           @Override
           public void onError(Throwable e) {
```

```
            }

            @Override
            public void onNext(String s) {

            }

            @Override
            public void onStart() {
                super.onStart();
            }
        } ;
```

9、创建被观察者

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

 //just方式  最多支持10个数据
 Observable<String> observable1 = Observable.just( "aa" , "bb" , "cc") ;
 // 将会依次调用:
 // onNext("aa");
 // onNext("bb");
 // onNext("cc");
 // onCompleted();

 //from方式
 //1:集合
 List<String> list = new ArrayList<>() ;
 list.add( "aa" ) ;
 list.add( "bb" ) ;
 list.add( "cc" ) ;

 Observable<String> observable2 = Observable.from( list ) ;

 //2: 数组
 String[] words = { "aa", "bb", "cc" };
 Observable<String> observable3 = Observable.from( words ) ;
```

- Call()方法：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。

10、订阅

由于观察者可以由两种方式被创建，所以订阅的方式也有两种

```
observable.subscribe( observer ) ;
observable.subscribe( subscriber ) ;
```

- Observable.subscribe(Subscriber) 的内部实现是这样的（仅核心代码）：

```
// 注意：这不是 subscribe() 的源码，而是将源码中与性能、兼容性、扩展性有关的代码剔除后的核心代码
// 如果需要看源码，可以去 RxJava 的 GitHub 仓库下载。
```

```
public Subscription subscribe(Subscriber subscriber) {
    subscriber.onStart();
    onSubscribe.call(subscriber);
    return subscriber;
}
```

1. 在subscribe() 中，首先会调用 onStart() 方法，这个方法前文已经介绍了，是可选的。接着会调用 call()方法，我们已经分析了在call()方法中会调用多次 onNext() ，最后调用 onCompleted().看到这里你就会突然明白原来subscribe() 方法其实相当于依次执行了：onStart() --> onNext（）--> onCompleted（）
2. 从这也可以看出，在 RxJava 中，`Observable` 并不是在创建的时候就立即开始发送事件，而是在它被订阅的时候，即当 `subscribe()` 方法执行的时候。
3. `Observer` 和 `Subscriber` 具有相同的角色，而且 `Observer` 在 `subscribe()` 过程中最终会被转换成 `Subscriber`对象
4. 将传入的 `Subscriber` 作为 `Subscription` 返回。这是为了方便 `unsubscribe()`.

11、RxBus

你是否听说过EventBus , 他是android 中的事件总线。用rxjava同样可以实现android的事件总线功能，也就是RxBus.

关于rxbus 的基本说明在这里 http://nerds.weddingpartyapp.com/tech/2014/12/24/implementing-an-event-bus-with-rxjava-rxbus/

然而这并没有什么卵用！

下面是RxBus的封装版

```
package lib.com.myapplication;
import android.support.annotation.NonNull;
import android.util.Log;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.concurrent.ConcurrentHashMap;
import rx.Observable;
import rx.subjects.PublishSubject;
import rx.subjects.Subject;

/**
 * Created by ${zyj} on 2016/5/6.
 */
public class RxBus {

    private static final String TAG = RxBus.class.getSimpleName();
    private static RxBus instance;
    public static boolean DEBUG = false;

    public static RxBus get() {
        if (instance == null) {
            synchronized (RxBus.class) {
                if (instance == null) {
                    instance = new RxBus();
                }
            }
        }
        return instance;
    }

    private RxBus() {
    }

```

```
    private ConcurrentHashMap<Object, List<Subject>> subjectMapper = new ConcurrentHas

    @SuppressWarnings("unchecked")
    public <T> Observable<T> register(@NonNull Object tag, @NonNull Class<T> clazz) {
        List<Subject> subjectList = subjectMapper.get(tag);
        if (null == subjectList) {
            subjectList = new ArrayList<>();
            subjectMapper.put(tag, subjectList);
        }

        Subject<T, T> subject;
        subjectList.add(subject = PublishSubject.create());
        if (DEBUG) Log.d(TAG, "[register]subjectMapper: " + subjectMapper);
        return subject;
    }

    public void unregister(@NonNull Object tag, @NonNull Observable observable) {
        List<Subject> subjects = subjectMapper.get(tag);
        if (null != subjects) {
            if ( observable != null  && subjects.contains( observable )){
                subjects.remove((Subject) observable);
            }

            if (isEmpty(subjects)) {
                subjectMapper.remove(tag);
            }
        }

        if (DEBUG) Log.d(TAG, "[unregister]subjectMapper: " + subjectMapper);
    }

    public void post(@NonNull Object content) {
        post( content.getClass().getName(), content);
    }

    @SuppressWarnings("unchecked")
    public void post(@NonNull Object tag, @NonNull Object content) {
        List<Subject> subjectList = subjectMapper.get(tag);

        if (!isEmpty(subjectList)) {
            for (Subject subject : subjectList) {
                subject.onNext(content);
            }
        }
        if (DEBUG) Log.d(TAG, "[send]subjectMapper: " + subjectMapper);
    }

    private boolean isEmpty(Collection collection) {
        return null == collection || collection.isEmpty();
    }
}
```

RxBus的使用

```
package lib.com.myapplication;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import rx.Observable;
import rx.functions.Action1;

public class Activity1 extends AppCompatActivity {

    String tag = "tag" ;
    Observable<String> ob ;

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity1);

        //创建被观察者
        ob = RxBus.get().register( tag , String.class ) ;
        //订阅观察事件
        ob.subscribe(new Action1<String>() {
            @Override
            public void call(String s) {
                System.out.println( "fff-- " + s  );
            }
        }) ;

        //发送内容
        RxBus.get().post(  tag , "我是内容" );
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        //取消订阅
        RxBus.get().unregister( tag , ob );
    }
}
```

- 在Activity销毁的时候，要取消订阅服务 。 否则 post() 次数会随着post()调用逐渐增加
- 除了上面的简单使用外，还可以使用 **Schedulers** 、**AndroidSchedulers** **进行线程切换**

RxJava 和 RxAndroid 二（操作符的使用）

分类: Android,Android Studio

标签: rxjava, rxandroid, rxjava操作符, markdown, 赵彦军, zhaoyanjun, android, rxbinding

赵彦军
关注 - 10
粉丝 - 89
+加关注

0 0

« 上一篇：Android studio 如何查看当前git 分支的4种方式
» 下一篇：Android studio .gitignore 文件的内容

posted @ 2016-02-01 15:32 赵彦军 阅读(7908) 评论(0) 编辑 收藏