RxBinding系列之RxView(一)

2017-11-09 | Android | 91

前言

自从用上RxJava后彻底迷上Rx系列响应式编程，从本篇开始一起来学习一下Rx套餐之一的RxBinding。RxBinding是什么？它是一组开源库，来自大神Jake Wharton之手，可将Android中各类UI控件的动作事件转换为RxJava中的数据流。也就是说使用RxBinding，我们就可以以RxJava的形式来处理UI事件。本篇主要讲解其中RxView的相关View事件如何绑定。

RxBinding中主要包含RxView、RxTextView、RxAdapterView、RxCompoundButton等等。由于全写一起篇幅太长，我就一篇讲解一种了。本系列主要讲解以上常用的4种，详细内容可飞机到：

Jake Wharton的RxBinding

依赖

本系列围绕我自己编写的RxBindingDemo来进行讲解，项目中主要使用ButterKnife做View注入，RxBinding做事件绑定。RxBindingDemo代码在文末有给地址。

注意：RxBinding包中包含RxJava的内容，所以就无需再添加RxJava的依赖了。

```
dependencies {
    ......
    compile 'com.jakewharton.rxbinding2:rxbinding:2.0.0'
    compile 'com.jakewharton:butterknife:8.8.1'
    annotationProcessor 'com.jakewharton:butterknife-compiler:8.8.1'
```

```
}
```

BaseActivity

相信学习过RxJava的码友一定记得，使用RxJava实例化的Disposable需在不用时及时销毁。

由于每个Activity中都写一套add与clear的方法会造成代码冗余，所以我将它们封装到BaseActivity中方便统一对Disposable进行管理，以及ButterKnife的统一绑定与解绑。

献上代码：

```
public abstract class BaseActivity extends AppCompatActivity {
    public CompositeDisposable mCompositeDisposable;
    private Unbinder mUnbinder;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(getLayoutId());
        mUnbinder = ButterKnife.bind(this);
        mCompositeDisposable = new CompositeDisposable();
        onViewCreated(savedInstanceState);
    }

    /**
     * 添加订阅
     */
    public void addDisposable(Disposable mDisposable) {
        if (mCompositeDisposable == null) {
            mCompositeDisposable = new CompositeDisposable();
        }
        mCompositeDisposable.add(mDisposable);
    }

    /**
     * 取消所有订阅
     */
    public void clearDisposable() {
        if (mCompositeDisposable != null) {
            mCompositeDisposable.clear();
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        clearDisposable();
```

```
        mUnbinder.unbind();
    }

    protected abstract int getLayoutId();

    protected abstract void onViewCreated(Bundle savedInstanceState);
}
```

click点击事件

clicks

`RxView.clicks(View view)`，通过源码可发现其内部封装了View.OnClickListener点击监听，调用clicks方法返回一个Observable对象，每当点击这个View的时候，该Observable对象就会发射一个事件，随即调用onNext()方法，Observable对应的观察者就可以通过onNext()回调响应此次点击事件。使用RxBinding还可做到点击防抖的效果。来看代码：

```
addDisposable(RxView.clicks(btnClick)
        .throttleFirst(2, TimeUnit.SECONDS)
        .subscribe(o -> {
            Log.e("rx_binding_test", "clicks:点击了按钮：两秒内防抖");
        }));
```

`throttleFirst(long windowDuration, TimeUnit unit)`，设置一定时间内只响应首次(throttleFirst)或者末次(throttleLast)的点击事件。windowDuration为防抖时间，unit为时间单位。调用这个方法便可防止短时间内对View的重复点击，本例中设置的防抖时间为2s。从代码看来是不是方便又简洁呢，以往实现防抖还得添加各种标记，忒麻烦。

longClicks

`RxView.longClicks(View view)`，内部封装了View.OnLongClickListener长按监听，原理同上。

```
addDisposable(RxView.longClicks(btnClick)
        .subscribe(o -> {
            Log.e("rx_binding_test", "longClicks:长点击了按钮");
        }));
```

draw绘制事件

`RxView.draws(View view)`，内部封装了OnDrawListener绘制监听。

```
//点击btnDraw调用viewCanvas的绘制
addDisposable(RxView.clicks(btnDraw)
        .throttleFirst(2, TimeUnit.SECONDS)
        .subscribe(o -> {
            //此处可模拟让viewCanvas绘制
            //viewCanvas.getViewTreeObserver().dispatchOnDraw();
        }));

//当viewCanvas绘制时触发
addDisposable(RxView.draws(viewCanvas)
        .subscribe(o -> {
            Log.e("rx_binding_test", "draws:viewCanvas绘制了");
        }));
```

drag拖拽事件

`RxView.drags(View view)`，内部封装了OnDragListener拖拽监听。

```
//当btnDraw被拖拽时触发
addDisposable(RxView.drags(btnDraw)
        .subscribe(o -> {
            Log.e("rx_binding_test", "drags:btnDraw被拖拽了");
        }));
```

layoutChange布局改变事件

`RxView.layoutChanges(View view)`，内部封装了OnLayoutChangeListener布局改变监听。

```
//点击btnChange改变btn_layout的布局,防抖2s
addDisposable(RxView.clicks(btnChange)
        .throttleFirst(2, TimeUnit.SECONDS)
        .subscribe(o -> btnLayout.layout(btnLayout.getLeft() - 20,
                btnLayout.getTop(), btnLayout.getRight() - 20, btnLayout.getBottom())
        ));

//btn_layout布局改变时触发
addDisposable(RxView.layoutChanges(btnLayout)
        .subscribe(o -> {
            Log.e("rx_binding_test", "layoutChanges:btnLayout布局改变了");
        }));
```

scrollChange滑动事件

`RxView.scrollChangeEvents(View view)`，内部封装了OnScrollChangeListener滑动监听。

```
//点击btnScroll模拟让btnScrollLayout滑动
addDisposable(RxView.clicks(btnScroll)
        .throttleFirst(2, TimeUnit.SECONDS)
        .subscribe(o -> {
            x += 10;
            if (x == 100) {
                x = 0;
            }
            btnScrollLayout.scrollTo(x, 0);
        }));

//btnScrollLayout滑动时触发
addDisposable(RxView.scrollChangeEvents(btnScrollLayout)
        .subscribe(event -> {
            Log.e("rx_binding_test", "scrollChangeEvents:btnScrollLayout滑动了:" + ever
        }));
```

View操作

RxView中还封装了一些常用的例如`setVisibility()`、`setClickable()`等View操作。使用起来也很简单，如下：

```
addDisposable(RxView.clicks(btnClick)
        .throttleFirst(2, TimeUnit.SECONDS)
        .subscribe(o -> {
            RxView.visibility(btnClick).accept(true);
            RxView.clickable(btnClick).accept(true);
            RxView.enabled(btnClick).accept(true);
        }));
```

这种操作方法单独使用是需要进行try-catch的，但还记得RxJava2系列中我们学习到，RxJava2的Action与Function的回调方法中都默认throws Exception，RxBinding也是如此。所以在观察者中调用就无需try-catch了。

更多的操作可在RxView的源码中查到，基本上View中有的，J大神都编写了，膜拜大神。

取消订阅

最后别忘了在Activity销毁时对创建的Disposable取消订阅。Demo中的Act都以BaseActivity为基类，所以就无需再调用取消订阅了，BaseActivity已经将这些工作做好了。

总结

RxBinding使用起来非常简单，RxView中还有attaches、detaches，focusChanges，globalLayouts，hovers，touches等等就不一一演示了，码友们可自行尝试。

进阶中的码猿一枚，写的不对的地方欢迎大神们留言指正，有什么疑惑或者建议也可以在我Github上RxBindingDemo项目Issues中提出，我会及时回复。

附上Demo的地址：

RxBindingDemo

RxBinding

RxJava2系列实践之倒计时功能(三)

RxBinding系列之RxTextView(二)

正在载入来必力