



[home](#) | [javascript](#) [php](#) [python](#) [java](#) [mysql](#) [ios](#) [android](#) [node.js](#) [html5](#) [linux](#) [c++](#) [css3](#) [git](#) [golang](#) [ruby](#) [vim](#) [docker](#)

## RxJava操作符在android中的使用场景详解（一）

[okhttp](#) [rxandroid](#) [retrofit](#) [rxjava](#) [android](#)

wangxinarhat 2016年04月19日发布

转载请注明出处：<http://www.wangxinarhat.com/2016/04/19/2016-04-19-rxjava-android-operate1/>

最近学习了RxJava在android中的使用，关于RxJava是啥，为什么要用RxJava，好在哪，这里就不叙述了，如果想要了解请移步[官方文档](#)、[大神文章](#)。

这里只讲解一下RxJava中的操作符在项目中的具体的使用场景。

因为学习了有20个操作符，可能一篇文章过于臃肿，所以打算写成系列文章，本文中所有操作符的使用，都写在了一个demo中，已上传至[github](#)

## 场景一：RxJava基本使用

配合Retrofit请求网络数据，如果你对Retrofit不熟悉就先看[Retrofit官网](#)，实现步骤如下：

1. 先是build.gradle的配置

```
compile 'io.reactivex:rxandroid:1.1.0'
compile 'io.reactivex:rxjava:1.1.0'
compile 'com.squareup.retrofit2:retrofit:2.0.0-beta3'
compile 'com.squareup.retrofit2:adapter-rxjava:2.0.0-beta3'
compile 'com.squareup.retrofit2:converter-gson:2.0.0-beta3'
compile 'com.jakewharton:butterknife:7.0.1'
```

也就是说本文是基于RxJava1.1.0和Retrofit 2.0.0-beta3来进行的。添加rxandroid是因为rxjava中的线程问题。

2. 基本网络请求使用准备

我们使用<http://zhuangbi.info/search?q=param>测试连接，返回的是json格式，代码就不贴了。接下来我们要创建一个接口取名为ZhuangbiApi，代码如下：

```
public interface ZhuangbiApi {
    @GET("search")
    Observable<List<ImageInfoBean>> search(@Query("q") String query);
}
```

Retrofit、Gson、RxJava结合使用，建立网络请求类：

```
public static ZhuangbiApi getZhuangbiApi() {
    if (zhuangbiApi == null) {
        Retrofit retrofit = new Retrofit.Builder()
            .client(okHttpClient)
            .baseUrl("http://zhuangbi.info/")
            .addConverterFactory(gsonConverterFactory)
            .addCallAdapterFactory(rxJavaCallAdapterFactory)
            .build();
        zhuangbiApi = retrofit.create(ZhuangbiApi.class);
    }
    return zhuangbiApi;
}
```

3. 具体使用 将要查询的关键词传进去，使用上面建立的网络请求类请求数据，并在订阅者的回调方法中，进行网络请求结果的处理

```
private void search(String key) {
    subscription = Network.getZhuangbiApi()
        .search(key)
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(getObserver());
}

private Observer<? super List<ImageInfoBean>> getObserver() {
    if (null == observer) {
        observer = new Observer<List<ImageInfoBean>>() {
            @Override
            public void onComplete() {
            }
        };
    }
}
```

```

    }

    @Override
    public void onError(Throwable e) {
        swipeRefreshLayout.setRefreshing(false);
        Toast.makeText(getActivity(), R.string.loading_failed, Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onNext(List<ImageInfoBean> images) {
        swipeRefreshLayout.setRefreshing(false);
        adapter.setImages(images);
    }
};
}

return observer;
}

```

4. 详解 search方法中传入的key是要查询的关键词，getObserver()是获取订阅者对象，并在其回调方法中根据返回结果，做相应处理：

- 其中onNext方法返回了数据，这样我们能够在onNext里面处理数据相关的逻辑；
- onError方法中处理错误，同时也可以停止ProgressDialog等；
- onComplete只调用一次结束本次请求操作，也可以停止ProgressDialog；

## 场景二：Map操作符的使用（变换）

对Observable发射的每一项数据应用一个函数，执行变换为指定类型的操作，然后再发射。

有些服务端的接口设计，会在返回的数据外层包裹一些额外信息，这些信息对于调试很有用，但本地显示是用不到的。使用 map() 可以把外层的格式剥掉，只留下我们只关心的部分，具体实现步骤如下：

### 1. 网络请求使用准备

我们使用<http://gank.io/api/>测试连接

接下来我们要创建一个接口取名为GankApi，代码如下：

```

public interface GankApi {

    @GET("data/福利/{number}/{page}")
    Observable<BeautyResult> getBeauties(@Path("number") int number, @Path("page") int page);

}

```

Retrofit、Gson、RxJava结合使用，建立网络请求类：

```

public static GankApi getGankApi() {
    if (gankApi == null) {
        Retrofit retrofit = new Retrofit.Builder()
            .client(okHttpClient)
            .baseUrl("http://gank.io/api/")
            .addConverterFactory(gsonConverterFactory)
            .addCallAdapterFactory(rxJavaCallAdapterFactory)
            .build();
        gankApi = retrofit.create(GankApi.class);
    }
    return gankApi;
}

```

### 2. 数据转换

返回数据就不贴了，有兴趣可以请求接口看一下。

接口返回的数据包含了一些额外的信息，但是我们只需要返回数据中的list部分，所以创建一个类，来实现数据转换的功能，代码如下：

```

public class BeautyResult2Beautise implements Func1<BeautyResult, List<ImageInfoBean>> {

    public static BeautyResult2Beautise newInstance() {
        return new BeautyResult2Beautise();
    }

    /**
     * 将接口返回的BeautyResult数据中的list部分提取出来，返回集合List<ImageInfoBean>
     * @param beautyResult
     * @return
     */
    @Override
    public List<ImageInfoBean> call(BeautyResult beautyResult) {

        List<ImageInfoBean> imageInfoBeanList = new ArrayList<>(beautyResult.results.size());

        for (ImageInfoBean bean : beautyResult.results) {
            ImageInfoBean imageInfoBean = new ImageInfoBean();

```

```

        imageInfoBean.description = bean.desc;

        imageInfoBean.image_url = bean.url;

        imageInfoBeanList.add(imageInfoBean);
    }
}

```

### 3. 操作符的使用 加载数据

```

/**
 * 加载数据的方法
 * @param page
 */
private void loadPage(int page) {
    mSwipeRefreshLayout.setRefreshing(true);
    unsubscribe();
    subscription = Network.getGankApi()
        .getBeauties(8, page)
        .map(BeautyResult2Beautise.newInstance())
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(getObserver());
}

```

订阅者

```

/**
 * 获取订阅者
 * @return
 */
private Observer<? super List<ImageInfoBean>> getObserver() {
    if (null == observer) {
        observer = new Observer<List<ImageInfoBean>>() {
            @Override
            public void onComplete() {
                mSwipeRefreshLayout.setRefreshing(false);
            }

            @Override
            public void onError(Throwable e) {
                mSwipeRefreshLayout.setRefreshing(false);

                Toast.makeText(getActivity(), R.string.loading_failed, Toast.LENGTH_SHORT).show();
            }

            @Override
            public void onNext(List<ImageInfoBean> images) {
                adapter.setImages(images);
            }
        };
    }

    return observer;
}

```

### 4. 详解

Map操作符对Observable发射的每一项数据应用一个函数，执行变换操作，然后返回一个发射这些结果的Observable。

本例中，接口返回的数据格式是：

```

public class BeautyResult {

    public boolean error;

    public List<ImageInfoBean> results;

}

```

但是我们只关心list部分的数据，所以进行转换操作，这样订阅者回调方法中拿到的数据直接使用就好了

## 场景三：Zip操作符的使用（结合）

通过一个函数将多个Observables的发射物结合到一起，基于这个函数的结果为每个结合体发射单个数据项，具体实现步骤如下：

1. 网络请求装备 网络请求Api，以及请求类，还是使用场景一、二中的创建好的。
2. 请求数据，并结合，代码如下：

```

/**
 * 请求两个接口，对返回的数据进行结合
 */
private void load() {
    swipeRefreshLayout.setRefreshing(true);
    subscription = Observable.zip(Network.getGankApi().getBeauties(188, 1).map(BeautyResult2Beautise.newInstance()),
        Network.getZhuangbiApi().search("装逼"),

```

```
new Func2<List<ImageInfoBean>, List<ImageInfoBean>, List<ImageInfoBean>>() {
    @Override
    public List<ImageInfoBean> call(List<ImageInfoBean> imageInfoBeen, List<ImageInfoBean> imageInfoBeen2) {

        int num = imageInfoBeen.size() < imageInfoBeen2.size() ? imageInfoBeen.size() : imageInfoBeen2.size();
        List<ImageInfoBean> list = new ArrayList<>();
        for (int i = 0; i < num; i++) {

            list.add(imageInfoBeen.get(i));
            list.add(imageInfoBeen2.get(i));

        }

        return list;
    }
}).subscribeOn(Schedulers.io())
.observeOn(AndroidSchedulers.mainThread())
.subscribe(getObserver());
}
```

3. 详解 请求GankApi中的数据使用map操作符进行转换，取出自己想要的list数据，然后结合ZhuangbiApi中的数据，形成新的数据集合，填充到view。

Zip操作符使用函数按顺序结合多个Observables发射的数据项，然后它发射这个函数返回的结果，它只发射与数据项最少的那个Observable一样多的数据。

一般app中同一个界面有时需要同时访问不同接口，然后将结果糅合后转为统一的格式后输出（例如将第三方广告 API 的广告夹杂进自家平台返回的数据 List 中）。这种并行的异步处理比较麻烦，不过用了 zip() 之后就会简单明了。

上一个效果图：



可以看出，recyclerView中使用了一个数据集合，但左侧的一列展示的是GankApi中的数据，右侧一列展示的是ZhuangbiApi 中的数据。

### 场景四：CombineLatest操作符的使用（结合）

结合多个Observable发射的最近数据项，当原始Observables的任何一个发射了一条数据时，CombineLatest使用一个函数结合它们最近发射的数据，然后发射这个函数的返回值，具体实现步骤如下：

## 1. 使用场景，用一个简单明了的图片来表示吧



## 2. 上图简单演示了CombineLatest的使用场景，看代码吧：

```
/**
 * 将3个EditText的事件进行结合
 */
private void combineLatestEvent() {
    usernameObservable = RxTextView.textChanges(mUsername).skip(1);
    emailObservable = RxTextView.textChanges(mEmail).skip(1);
    passwordObservable = RxTextView.textChanges(mPassword).skip(1);

    subscription = Observable.combineLatest(usernameObservable, emailObservable, passwordObservable,
        new Func3<CharSequence, CharSequence, CharSequence, Boolean>() {
            @Override
            public Boolean call(CharSequence userName, CharSequence email, CharSequence password) {
                boolean isUserNameValid = !TextUtils.isEmpty(userName) && (userName.toString().length() > 2 && userNa

                if (!isUserNameValid) {
                    mUsername.setError("用户名无效");
                }

                boolean isEmailValid = !TextUtils.isEmpty(email) && Patterns.EMAIL_ADDRESS.matcher(email).matches();

                if (!isEmailValid) {
                    mEmail.setError("邮箱无效");
                }

                boolean isPasswordValid = !TextUtils.isEmpty(password) && (password.toString().length() > 6 && passwo

/**
 * 获取订阅者
 * @return
 */
private Observer<Boolean> getObserver() {
    return new Observer<Boolean>() {
        @Override
        public void onCompleted() {

        }

        @Override
        public void onError(Throwable e) {

        }

        @Override
        public void onNext(Boolean aBoolean) {
            //更改注册按钮是否可用的状态
            mButton.setEnabled(aBoolean);
        }
    };
}
```

```
    }  
    };  
}
```

3. 详解 CombineLatest操作符行为类似于zip，但是只有当原始的Observable中的每一个都发射了一条数据时zip才发射数据。CombineLatest则在原始的Observable中任意一个发射了数据时发射一条数据。

当原始Observables的任何一个发射了一条数据时，CombineLatest使用一个函数结合它们最近发射的数据，然后发射这个函数的返回值。

本例中，含用户名、邮箱、密码、注册按钮的注册页面的场景非常常见，当然可以使用普通的处理方式能够达成，注册按钮的是否可用更改的效果，以及输入是否合法的及时提示。

但是使用RxJava的方式，代码明显简洁、易懂。

## 小结

虽然，上面四个使用场景主要介绍四个操作符的使用，但其实demo中穿插了不少其他操作符的使用，想要详细了解的话，代码在[这里](#)。

暂时先写到这里，后面会把其他自己学会的的操作符，写成系列文章。如有兴趣，请关注我的[github](#)。

2016年04月19日发布 更多

2 推荐

收藏

### 你可能感兴趣的文章


- [Android入门及效率开发](#) 12 收藏，1k 浏览
- [2016年最值得学习的五大开源项目](#) 14 收藏，1.6k 浏览
- [Rxjava实践-把混乱的WORKFLOW撸成串吧](#) 338 浏览

### 评论 默认排序

文明社会，理性评论

发布评论

广告



wangxinarh...

372 声望

发布于专栏

wangxinarhat

The world as I see it

3 人关注

关注作者


关注专栏

### 系列文章

[RxJava操作符在android中的使用场景详解（二）](#) 8 收藏，2k 浏览


### 相关收藏夹

换一组



Android

5 个条目 | 0 人关注



Android