
#RxJava +retrofit2实现安卓中网络操作~

> 在安卓中想实现网络操作有多种方式，可能许多没有经历过团队开发的安卓工程师，经常使用到的是第三方的云后台，但是其实它们的底层使用的也一定是安卓中网络的通信框架，例如：volley，nohttp，okhttp等。

> 今天我们要介绍的retrofit2底层就是用的okhttp的通信方式，下面简单介绍一下为什么写这篇文章吧，在开发团队项目以前，我也和我小伙伴交流过，我说咱们一直采用第三方的云后台，你说咱们怎么和真正的服务器做联系啊，我们那个时候异口同声的说不知道，后来在我做的第一个团队项目“黑大盒子”的时候，我就学习了okHttp，那个时候感觉很好用啊，代码也挺简洁的，直至后来在一次，和我学长的交流过程中了解了可以用RxJava +retrofit2实现网络的通信，那个时候我真的是下了很大的功夫研究，结果是毫无头绪，因为那个时候，我对观察者模式，和rxjava的系统的思想就是非常的不够的。在后来我学习了观察者模式，又一直在找rxjava的资料使我对这种网络通信有了一定的了解。

## 使用RxJava +retrofit2实现网络通信的优势

- 请求时间和返回时间短（性能上的优势）
- 代码简介，已经把封装实现到了极致

> 毫不夸张的说，如果公司没有自己的网络操作的框架，采用这种方式一定是最佳选择之一

## RxJava的简介

链接：https://github.com/ReactiveX/RxJava RxJava采用的思想便是观察者模式，可以异步实现实现我们的需求，简单的说，RxJava并不是轮询去检测被观察的对象，而是当被观察的对象有任何举动的时候都会告诉我们，我们便可以根据这个消息决定要做什么处理。

## retrofit2的简介

链接：https://github.com/square/retrofit retrofit这个库的功能非常的强大，它可以直接向Gson添加依赖，解析我们返回的json数据非常的轻松，而且我们实现网络操作也非常的便捷，非常轻松的就可以实现在工作线程请求网络操作，在主线程实现对网络操作结果的处理。

> 在这里我要非常正式的声明一下，不是笔者懒，用两句话，就把这么传奇的两个第三方开源库就给介绍完了，而且本文主要的目

> 的是教大家怎么用它们实现网络操作，我接下来的文章会非常认真的介绍，观察者模式、RxJava、retrofit，这三方面的知识。
> ps：我始终认为，在你学习一个编程上的知识的时候，你最先要做的不是弄明白它，而是要用明白它，然后跟着代码的逻辑走一遍，看看它是怎么实现的，然后可以看看人家的官方文档，或者大神们的博客学习一下，最后我们还可以阅读以下源码，这样学起来可能会轻松，也会比较高效。

# RxJava +retrofit2的使用！！！

第一步：在build.gradle中添加依赖

```
// RxJava Android 支持
compile 'io.reactivex:rxjava:1.1.6'
compile 'io.reactivex:rxandroid:1.2.1'
// Retrofit 网络支持
compile 'com.squareup.retrofit2:retrofit:2.1.0'compile 'com.squareup.retrofit2:converter-gson:2.1.0'
// Gson 适配器
compile 'com.squareup.retrofit2:adapter-rxjava:2.1.0'
```

第二步：添加必要的权限

```
<uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_CONFIGURATION"/>
    <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
```

第三步：定义一个接口，实现网络操作 这个类，是在我们主URL后面链上的字段，然后泛型的是要返回数据我要将它解析成什么样子，里面的参数就是post请求需要携带的参数了。

```
/**
 * Created by lin_sir on 2016/6/29.网络协议
 */

public interface Api {

    @FormUrlEncoded
    @POST("getCode")
    Observable<ResponseModel_nolist> getCode(@Field("tel") String tel);

    @FormUrlEncoded
    @POST("register")
    Observable<ResponseModel_nolist> register(@Field("tel") String tel, @Field("password") String pas

}
```

第四步：实现刚才泛型的类 为了让大家使用起来毫无难度，我就在这里把这个简单的类也粘出来了

```
/**
 * Created by lin_sir on 2016/7/7.结果中不带有list的网络返回结果
 */
public class ResponseModel_nolist {

    private int code;
    private String msg;

    private obj obj;

    public int getCode() {
        return code;
    }

    public void setCode(int code) {
        this.code = code;
    }

    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
        this.msg = msg;
    }

    public com.example.lin_sir_one.tripbuyer.model.obj getObj() {
        return obj;
    }

    public void setObj(com.example.lin_sir_one.tripbuyer.model.obj obj) {
        this.obj = obj;
    }
}
```

第五步：实现一个BASE_URL,以后我们的网络操作走的url都是在这些url后面加字段 当然我打星号的，都是我们公司后台暴露出来的接口啦，虽然我们做了负载均衡，也做了防止别人攻击的处理啦，但是在这里暴露出来好像也还是不太好~

```
/* Created by lin_sir on 2016/6/29.全局常量定义
 */
public class Constant {

    public static final String BASE_URL = "http://xxx.xxx.xxx.xxx:8080/lxms-user/member/api/";

    public static final String BASE_BUY_URL = "http://xxx.xxx.xxx.xxx:8080//lxms-user/buyer/api/";

    public static final String BASE_SELL_URL = "http://xxx.xxx.xxx.xxx:8080/lxms-user/seller/api/";

}
```

第六步：实现一个Apiservice，这个类的作用就是实现我们的网络操作的直接方式~

```
/**
 * Created by lin_sir on 2016/7/7.调用api接口,获取验证码和注册采用这个接口
 */
public class ApiService {

    private Api mApi;
    private Context mContext;
    private static ApiService mInstance;

//-------- 存在内存泄漏的写法,如果传入 Activity 的 Context,会导致 Activity 无法被回收-------------------
//    private ApiService(Context mContext) {
//        this.mContext = mContext;
//        mApi = RetrofitClient.getClient(mContext).create(Api.class);
//    }
//
//    public static ApiService getInstance(Context mContext) {
//        if (mInstance == null) {
//            mInstance = new ApiService(mContext);
//        }
//        return mInstance;
//    }
//-----------------------------------------------------------------------------------------------

    private ApiService() {
        this.mContext = BaseApplication.get().getAppContext();
        mApi = RetrofitClient.getClient(mContext).create(Api.class);
```

```
    }

    public static ApiService getInstance() {
        if (mInstance == null) {
            mInstance = new ApiService();
        }
        return mInstance;
    }

    /**
     * 获取验证码
     */
    public void getCode(HttpResultListener<Boolean> listener, final String tel) {
        mApi.getCode(tel)
                .map(new HttpResultFuncNoList())
                .map(new Func1<String, Boolean>() {
                    @Override
                    public Boolean call(String s) {
                        if (s.equals("ok")) {
                            return true;
                        } else {
                            return false;
                        }
                    }
                })
                .subscribeOn(Schedulers.io())//在工作线程请求网络
                .observeOn(AndroidSchedulers.mainThread())//在主线程处理结果
                .subscribe(new HttpResultSubscriber<>(listener));
    }



    private static class HttpResultSubscriber<T> extends Subscriber<T> {

        private HttpResultListener<T> mListener;

        public HttpResultSubscriber(HttpResultListener<T> listener) {
            mListener = listener;
        }


        @Override
        public void onCompleted() {

        }

        @Override
        public void onError(Throwable e) {
            if (mListener != null) {
                mListener.onError(e);
            }
        }

        @Override
        public void onNext(T t) {
            if (mListener != null) {
                mListener.onSuccess(t);
            }
        }
    }

    /**
     * 对返回结果做统一处理,只有当结果码为 100 时,才返回正常,否则返回错误,不带list的
     */
    private class HttpResultFuncNoList implements Func1<ResponseModel_nolist, String> {

        @Override
        public String call(ResponseModel_nolist responseModel) {

            if (responseModel.getCode() == NetworkException.REQUEST_OK) {
                Log.i("lin", "---lin--->  目前没发生错误:   " + responseModel.getCode());
                return responseModel.getMsg();
            } else {
                Log.i("lin", "---lin--->  错误代码:   " + responseModel.getCode());
                throw new NetworkException(responseModel.getCode());
            }
        }
    }

}
```

第七步：实现网络操作的回调接口

```
/**
 * Created by lin_sir on 2016/7/7.网络操作的回调接口
 */
public interface HttpResultListener<T> {

    void onSuccess(T t);

    void onError(Throwable e);

}
```

第八步：实现retrifit2客户端 在这个客户端里面，我们不仅实现了Gson的适配器，也实现了Rxjava的适配器，还为我们的操作添加了主URL就可以了。

```
public class RetrofitClient {

    /**
     * 采用base_url
     */
    public static Retrofit getClient(Context context) {
        return new Retrofit.Builder()
                .baseUrl(Constant.BASE_URL)
                //.client(httpClient(context))
                .addConverterFactory(GsonConverterFactory.create())//Gson 适配器
                .addCallAdapterFactory(RxJavaCallAdapterFactory.create())// RxJava 适配器
                .build();
    }
}
```

其实我们的网络操作就已经实现完了，如果我们代码习惯比较好的话，可以把所有的服务器返回的状态吗放在一起进行统一的处理。

统一处理后台返回状态码的代码：

```java
/**
 * Created by tc on 6/21/16. 网络操作错误
 */
public class NetworkException extends RuntimeException {

    public static final int REQUEST_OK = 100;
    public static final int REQUEST_FAIL = 101;
    public static final int METHOD_NOT_ALLOWED = 102;
    public static final int PARAMETER_ERROR = 103;
    public static final int UID_OR_PWD_ERROR = 104;
    public static final int SERVER_INTERNAL_ERROR = 105;
    public static final int REQUEST_TIMEOUT = 106;
    public static final int CONNECTION_ERROR = 107;
    public static final int VERIFY_EXPIRED = 108;
    public static final int NO_DATA = 109;


    public NetworkException(int resultCode) {
        this(getNetworkExceptionMessage(resultCode));
    }

    public NetworkException(String detailMessage) {
        super(detailMessage);
    }

    /**
     * 将结果码转换成对应的文本信息
     */
    private static String getNetworkExceptionMessage(int code) {
        String message = "";
        switch (code) {
            case REQUEST_OK:
                message = "请求成功";
                break;
            case REQUEST_FAIL:
                message = "请求失败";
                break;

            case METHOD_NOT_ALLOWED:
                message = "请求方式不允许";
                break;
            case PARAMETER_ERROR:
                message = "用户不存在";
                break;
            case UID_OR_PWD_ERROR:
                message = "用户名或密码错误";
                break;
            case SERVER_INTERNAL_ERROR:
                message = "服务器内部错误";
                break;
            case REQUEST_TIMEOUT:
                message = "请求超时";
                break;
            case CONNECTION_ERROR:
                message = "连接错误";
                break;
            case VERIFY_EXPIRED:
                message = "验证过期";
                break;
            case NO_DATA:
                message = "没有数据";
                break;
            case 110:
                message = "该用户已存在";
                break;
            default:
                message = "未知错误";
        }
        return message;
    }
}
```

好了，我们已经可以使用RxJava +retrofit2实现网络操作了：

```
HttpResultListener<List<JourneyModel>> listener = new HttpResultListener<List<JourneyModel>>() {
            @Override
            public void onSuccess(List<JourneyModel> journeyModels) {
                KLog.d("----lin---->  成功");
            }

            @Override
            public void onError(Throwable e) {
                KLog.d("----lin---->  失败" + e.toString());
            }
        };
        ApiService6.getInstance().route2(listener, "1");
```

到这里我们就已经把网络通信彻底的研究了一遍啦，虽然看起来稍微有一点点小复杂，但是我们要想的是，整个工程的网络通信，我这点代码就已经都写完了啊，以后用起来可就非常的方便啦。

在这里，我们对RxJava，Retrofit2已经有了一个了解了，初步我们已经会使用了这些知识，在接下来的文章里，我不会再这么详细的介绍它们的使用，而是要介绍它们的实现的原理，和它们思想上的一些东西。

> 声明：以上内容为linSir本人原创，这些知识都是我的小手下tc孜孜不倦教给我的，哈哈哈，他的个人网站是：www.classtc.com，里面也有许多和编程有关的知识~如果大家，对这些知识有什么疑问，可以留言，我会第一时间回复的。