

目录

目录

一、网络基础知识

- 1.1 网络分层模型
- 1.2 HTTP协议
- 1.3 RESTful API
 - 1.3.1 API
 - 1.3.2 RESTful API
- 1.4 数据传输格式
 - 1.4.1 JSON解析器: gson
 - 1.4.2 下划线命名 vs 驼峰命名
- 1.5 实用工具
 - 1.5.1 JSON相关
 - 1.5.2 抓包工具 Charles
 - 1.5.3 抓包工具 Postman-轻松创建请求

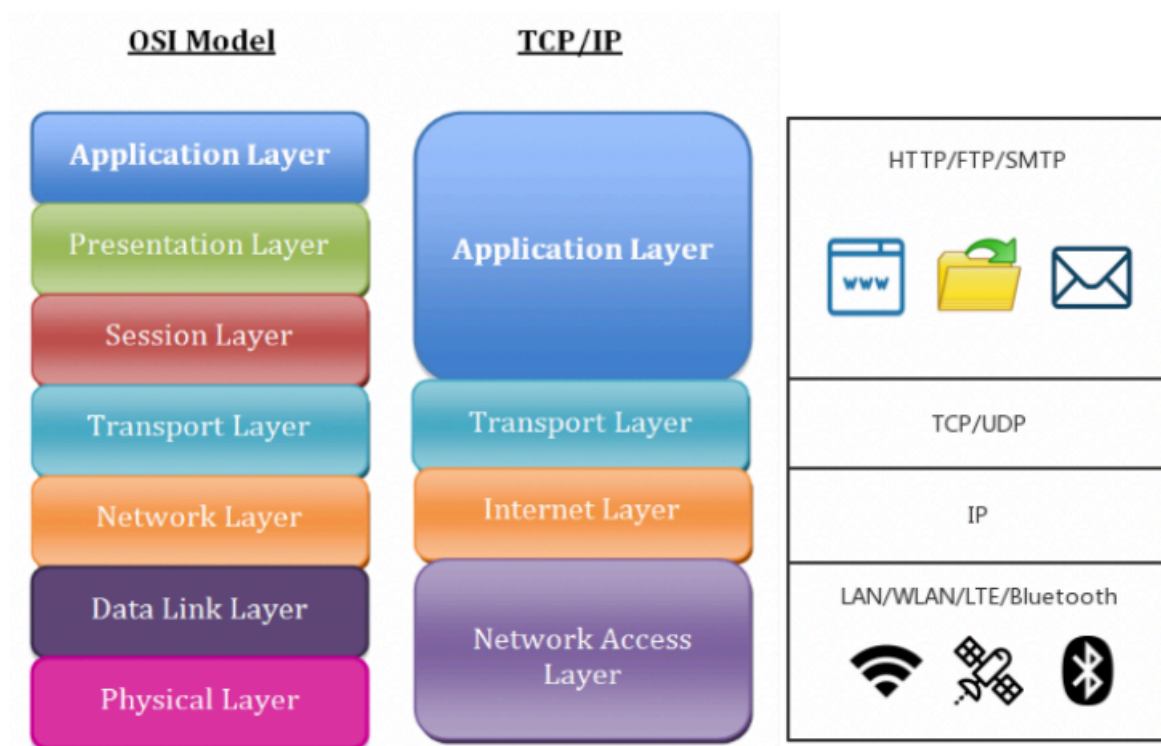
二、Android网络通信基础实现

- 2.1 添加网络权限
- 2.2 获取网络中的数据
- 2.3 使用WebView打开网页

三、进阶实现: Retrofit(自学)

一、网络基础知识

1.1 网络分层模型



1. 应用层**Application Layer**: 互联网上的各种应用

- 1. 如: 看网页、发邮件
- 2. 如: HTTP/FTP/SMTP协议

2. 传输层**Transport Layer**

1.2 HTTP协议

1. HTTP是一个client-server协议，只能由client主动发起请求，server进行响应

1. 用户发出请求，服务端进行相应

2. 一个HTTP请求一定要包含Method和URL

1. **Method**: 要做什么，如GET、PUT、HEAD、POST、DELETE、TRACE、OPTIONS、CONNETC

2. **URL**:

3. Request格式:

Http Request 格式 (访问https://www.zju.edu.cn/main.html)

```
:method GET
:authority www.zju.edu.cn
:scheme https
:path /main.htm
cache-control max-age=0
sec-ch-ua "Chromium";v="104", " Not A;Brand";v="99", "Google Chrome";v="104"
sec-ch-ua-mobile ?0
sec-ch-ua-platform "macOS"
upgrade-insecure-requests 1
user-agent Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
accept text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;...
sec-fetch-site same-origin
sec-fetch-mode navigate
sec-fetch-user ?1
sec-fetch-dest document
accept-encoding gzip, deflate, br
accept-language zh-CN,zh;q=0.9,en;q=0.8
cookie JSESSIONID=683C6313F9F6582F1AF16FA24738FDB8
cookie Hm_lvt_fe30bbc1ee45421ec1679d1b8d8f8453=1661170367
cookie Hm_lvt_fe30bbc1ee45421ec1679d1b8d8f8453=1661170913
```

4. Response格式:

协议版本 状态码 状态码的原因短语

HTTP/1.1 200 OK

响应首部字段

```
Date: Tue, 10 Jul 2012 06:50:15 GMT
Content-Length: 362
Content-Type: text/html

<html>
...
```

5. HTTP状态码:

	类别	原因短语
1XX	Informational (信息性状态码)	接收的请求正在处理
2XX	Success (成功状态码)	请求正常处理完毕
3XX	Redirection (重定向状态码)	需要进行附加操作以完成请求
4XX	Client Error (客户端错误状态码)	服务器无法处理请求
5XX	Server Error (服务器错误状态码)	服务器处理请求出错

1. **4XX**: 服务器正常，但客户端出错

2. **404**: 服务器正常，但服务不存在

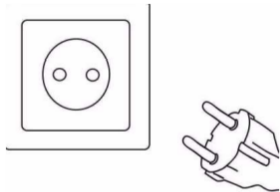
3. **5XX**: 服务器不正常，无法处理请求(请求过多)

1.3 RESTful API

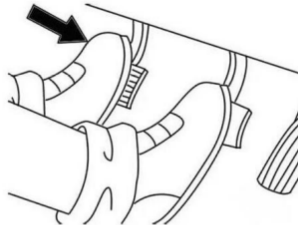
1.3.1 API

API: Application Programming Interface 应用程序接口

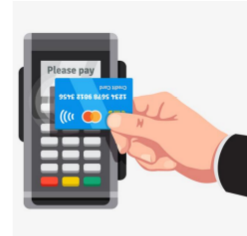
1. **用户端**使用固定的方式发起请求
2. **服务端**提供服务，响应该请求
3. 使用一个固定的方式，保证不同模块之间进行通信，提高兼容性
4. 略去不同模块之间的区别，找到共同的部分



插座是插头与交流电交互的 API



油门是司机与发动机交互的 API



NFC是磁卡与读卡机交互的 API

前后端通信中的 API 通常指的是一个 url，比如 <https://api.github.com/>

1.3.2 RESTful API

RESTful API: Resource Epresentational State Transfer (资源)表现层状态转换

1. API 是面向资源的，资源表达的形式可以是json或者xml，它的url中不包含动词，而是通过HTTP动词表达想要的动作
2. 资源：一段文本、一张图片、一首歌曲
3. 表现形式：json、xml
4. 状态变化：通过HTTP动词实现
5. 目的：看 URL 就知道要什么，看HTTP method 就知道干什么

RESTful 只是一种规范，并不是标准

SOAP Web API

GET http://127.0.0.1/user/select/1 根据用户id查询用户数据
POST http://127.0.0.1/user/save 新增用户
GET/POST http://127.0.0.1/user/delete 删除用户信息



不在 url 里面做具体动作的描述，而是通过 HTTP 请求方式表达意图

RESTful Web API

GET http://127.0.0.1/user/1 据用户id查询用户数据
POST http://127.0.0.1/user 新增用户
DELETE http://127.0.0.1/user 删除用户信息

GET 用来获取资源
POST 用来新建资源（也可以用于更新资源）
PUT 用来更新资源
DELETE 用来删除资源

1.4 数据传输格式





后端与前端/客户端需要约定数据传输的格式，以json为例

1. 每一组数据都是键值对
2. {}括起的数据：单个值
3. []括起的数据：数组值

```
{
  "name" : "中国",
  "provinces" : [{
    "name" : "黑龙江",
    "cities" : {
      "city" : ["哈尔滨", "大庆"]
    }
  }, {
    "name" : "广东",
    "cities" : {
      "city" : ["广州", "深圳", "珠海"]
    }
  }
}]
}
```

1.4.1 JSON解析器：gson

gson ←-----→ jackson

 gson-2.8.5.jar	242 KB
 jackson-annotations-2.11.3.jar	68 KB
 jackson-core-2.11.3.jar	351 KB
 jackson-databind-2.11.3.jar	1.4 MB

轻量级

(7.6倍体积之差)

```
// 将对象转化成Json
public <T> toJson(Object src);
// 从Json中提取对象
public <T> fromJson(String json, Class<T> classOfT);
public <T> fromJson(Reader json, Type typeOfT);
```

```
Test.java
5 public class Test {
6     private static final String json =
7         "{" +
8             "\"company_name\": \"bytedance\"," +
9             "\"age\": 8," +
10            "\"social_code\": 911101085923662400," +
11            "\"is_global\": true," +
12            "\"office_area\": [\"shanghai\", \"beijing\", \"shenzhen\", \"chengdu\"]," +
13            "\"other_info\": {" +
14                "\"job\": \"https://job.bytedance.com/\", \"ceo\": \"zhangyiming\"" +
15            "}" +
16        "}" +
17    };
18
19     public static void main(String[] args) {
20         Company company = new Gson().fromJson(json, Company.class);
21         System.out.println(company.getCompany_name()); // bytedance
22     }
23 }
```

```
Company.java
5 public class Company {
6     private String company_name;
7     private int age;
8     private long social_code;
9     private boolean is_global;
10    private Set<String> office_area;
11    private OtherInfo other_info;
12
13    public static class OtherInfo {
14        private String job;
15        private String ceo;
16    }
17
18    public String getCompany_name() {
19        return company_name;
20    }
21 }
```

Run: sample [Test.main()] ×

✓ sample [Test.main()]: successful at 2020/11/3, 10:40 上午 9 s 958 ms > Task :Test.main() bytedance

1.4.2 下划线命名 vs 驼峰命名

```
public class Company {
    private String company_name;
    private int age;
    private long social_code;
    private boolean is_global;
    private Set<String> office_area;
    private OtherInfo other_info;

    public static class OtherInfo {
        private String job;
        private String ceo;
    }

    public String getCompany_name() {
        return company_name;
    }
}
```

```
public class Company {
    @SerializedName("company_name")
    private String companyName;
    @SerializedName("age")
    private int age;
    @SerializedName("social_code")
    private long socialCode;
    @SerializedName("is_global")
    private boolean isGlobal;
    @SerializedName("office_area")
    private Set<String> officeArea;
    @SerializedName("other_info")
    private OtherInfo other_info;

    public static class OtherInfo {
        @SerializedName("job")
        private String job;
        @SerializedName("ceo")
        private String ceo;
    }

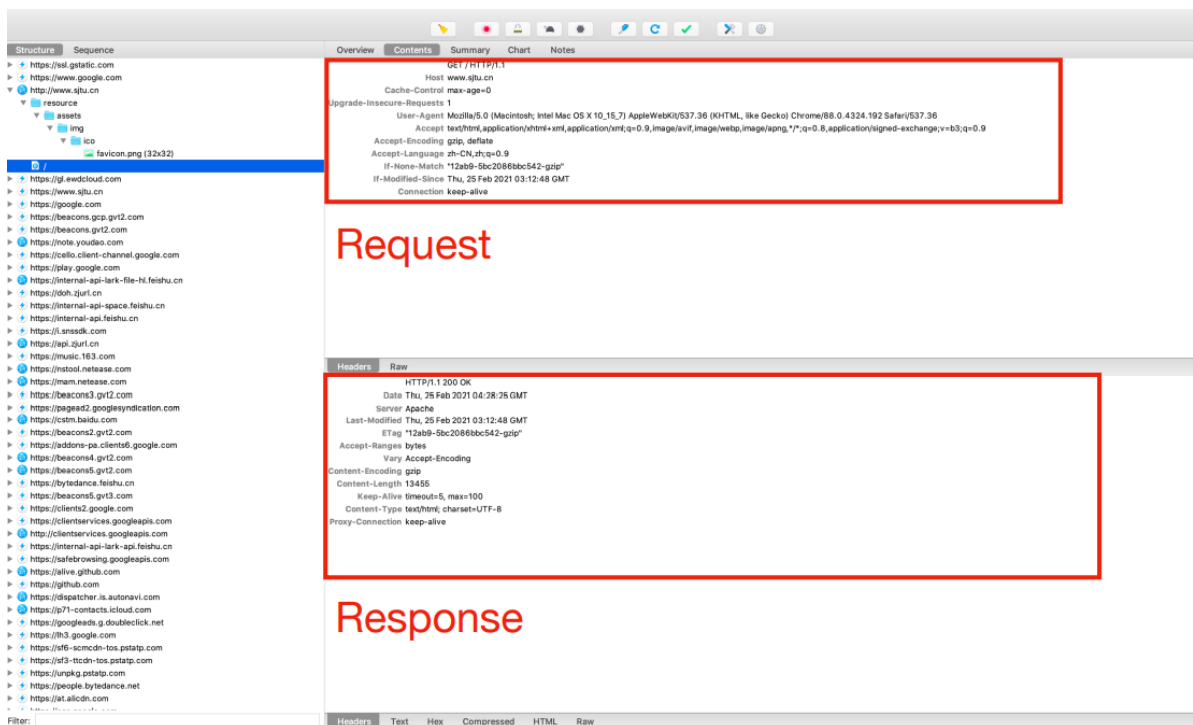
    public String getCompanyName() {
        return companyName;
    }
}
```

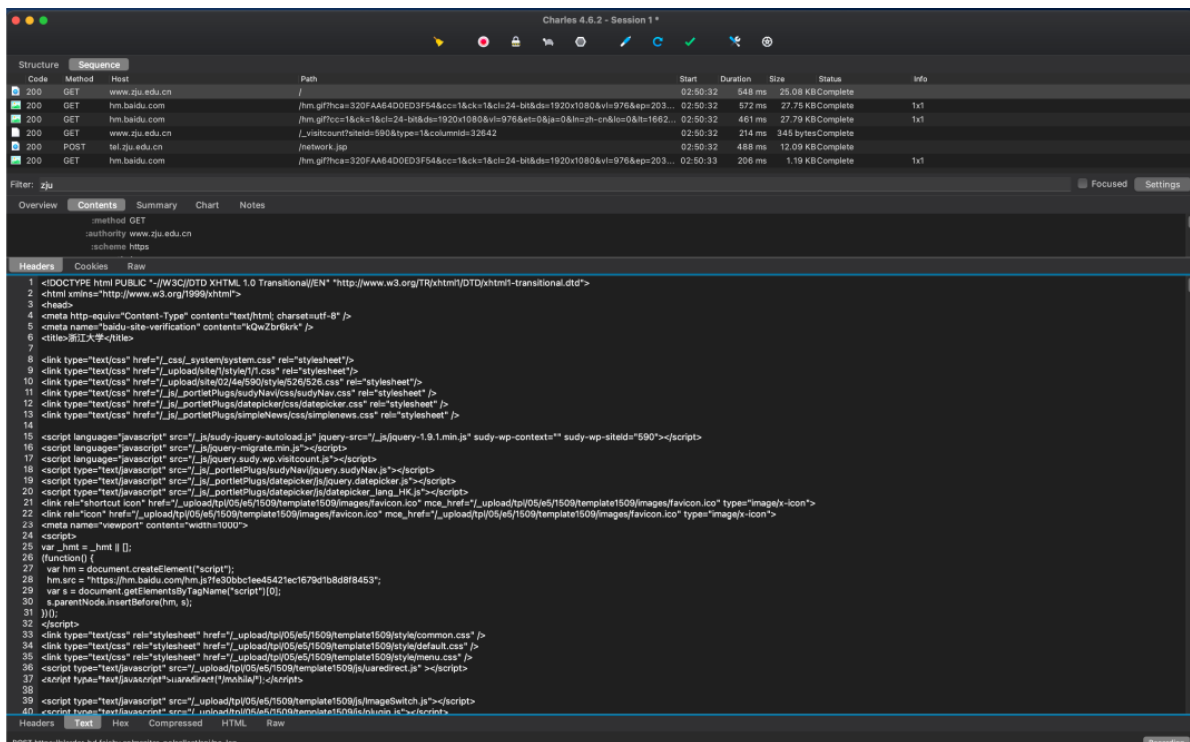
1.5 实用工具

1.5.1 JSON相关

1. [JSON 在线辅助网站](#)：转 JavaBean；合法性校验；压缩；优化预览
2. GsonFormatPlus：IDEA 插件，JSON 转 JavaBean
3. JSON：维基百科，了解 JSON 的来龙去脉

1.5.2 抓包工具 Charles

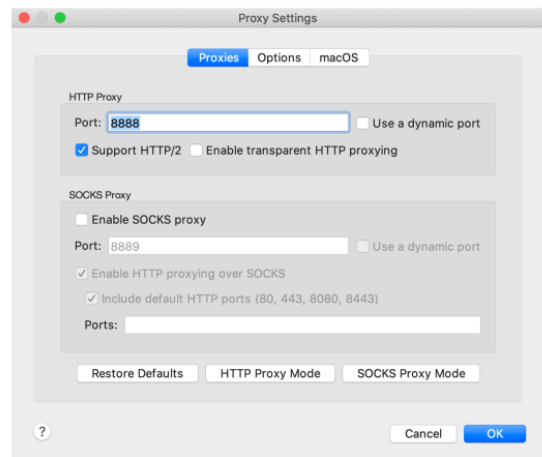




实用工具

抓包工具——Charles

- 1、将手机和电脑连到同一个局域网
- 2、设置Charles代理端口
- 3、将手机wifi的代理服务器设到电脑上
- 4、如果要抓取https包，手机要安装证书



1.5.3 抓包工具 Postman-轻松创建请求

https://api.github.com/users/JakeWharton/repos?page=0&per_page=10

GET https://api.github.com/users/JakeWharton/repos?page=0&per_page=10 Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	page	0			
<input checked="" type="checkbox"/>	per_page	10			
	Key	Value	Description		

Body Cookies Headers (24) Test Results 200 OK 1152 ms 56.43 KB Save Response

Pretty Raw Preview Visualize JSON

```
2 {
3   "id": 3070104,
4   "node_id": "MDEwO1JlcG9zaXRvcnkzMdcwMTA0",
5   "name": "abs.io",
6   "full_name": "JakeWharton/abs.io",
7   "private": false,
8   "owner": {
9     "login": "JakeWharton",
10    "id": 66577,
11    "node_id": "MDQ6VXNlcjY2NTc3",
12    "avatar_url": "https://avatars.githubusercontent.com/u/66577?v=4",
13    "gravatar_id": "",
14    "url": "https://api.github.com/users/JakeWharton",
```

二、Android网络通信基础实现

2.1 添加网络权限

在AndroidManifest.xml中，添加网络权限uses-permission

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    . . . . .
</manifest>
```

2.2 获取网络中的数据

1. 新建一个线程HandlerThread，用于执行与网络有关的任务
2. 新建一个任务Handler，用于从url中获取数据
 1. 在run()方法中
 2. 调用自定义的getDataFromNetwork()方法，获取数据
 3. 然后根据得到的数据，调用自定义的showDataFromNetwork()方法，刷新界面

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    // 初始化界面的相关操作
    super.onCreate(savedInstanceState);
    setContentView(R.layout.lab5_main_layout);
    networkResult = findViewById(R.id.Lab5_NetworkResult);

    // 执行从网络中获取数据的操作
```



```

runGetDataFromNetwork("https://www.wanandroid.com/article/list/0/json/");
}

// 新建线程，执行从网络中获取数据的操作
private void runGetDataFromNetwork(String urlString){
    // 新建一个线程，用于执行与网络有关的任务
    HandlerThread handlerThread = new HandlerThread("Networking");
    handlerThread.start();

    // 新建一个Handler，用于从url中获取数据
    Handler networkingHandler = new Handler(handlerThread.getLooper());
    Runnable networkingRunnable = new Runnable() {
        @Override
        public void run() {
            Log.d(TAG, "run networking Runnable: 从url中获取数据");
            // 调用getDataFromNetwork()方法，获取数据
            ArrayList<Lab5_NetworkData> result = getDataFromNetwork(urlString);

            // 根据得到的数据，刷新界面
            if(result != null && !result.isEmpty()){
                Log.d(TAG, "run networking Runnable: 从url中获取数据为:" + "not
null");
                showDataFromNetwork(result);
            }else{
                Log.d(TAG, "run networking Runnable: 从url中获取数据为:" + "null");
            }
        }
    };
    networkingHandler.post(networkingRunnable);
}

```

1. 获取数据：getDataFromNetwork(String urlString)方法

1. 将传入的urlString转化为URL
 2. 使用URLConnection建立连接
 1. 创建实例connection
 3. 从connection中获取数据
 1. 先获取为InputStream
 2. 然后转化为BufferedReader
 3. 最后，通过BufferedReader.readLine()方法，将其中的数据转化为JSONObject
 4. 解析JSON：从JSONObject中，读取所有data.datas[].title/link
 1. 根据目标JSON文件的结构，反复调用getJSONObject()、getJSONArray()方法
 2. 如果当前到达了最低层，则调用getString()方法，获取String类型的数据
 3. 将数据存放入一个ArrayList中
 5. 将读取到的ArrayList返回
2. 注意要使用try，防止操作出现异常

```

// 从网络中获取数据
// 返回一个List，存放解析出的Json数据
private ArrayList<Lab5_NetworkData> getDataFromNetwork(String urlString){
    Log.d(TAG, "getDataFromNetwork: 从网络中获取数据，返回List");
    ArrayList<Lab5_NetworkData> result = new ArrayList<Lab5_NetworkData>();
    // 通过try，防止获取数据出现问题
    try{

```

```

// 将urlString转化为URL
URL url = new URL(urlString);

// 使用URLConnection建立连接
URLConnection connection = (URLConnection) url.openConnection();
connection.setConnectTimeout(6000);
connection.connect();

// 获取数据，并将其转化为JSONObject
InputStream inputStream = connection.getInputStream();
BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream, StandardCharsets.UTF_8));
JSONObject resultJson = new JSONObject(reader.readLine());

// 从JSONObject中，读取所有data.datas[].title/link
JSONObject data = resultJson.getJSONObject("data");
JSONArray datas = data.getJSONArray("datas");
for(int index = 0; index < datas.length(); index++){
    JSONObject item = datas.getJSONObject(index);
    String title = item.getString("title");
    String link = item.getString("link");
    result.add(new Lab5_NetworkData(title, link));
}

// 关闭BufferedReader，InputStream，取消URLConnection的连接
reader.close();
inputStream.close();
connection.disconnect();
return result;
} catch (Exception e) {
    e.printStackTrace();
}
return null;
}

```

2.3 使用WebView打开网页

1. 新建一个Activity，其中有WebView控件

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".Lab5_WebViewActivity">

    <WebView
        android:id="@+id/Lab5_WebView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>

```

2. 切换至WebViewActivity时，将urlString作为参数，添加进intent中

```
// 使用单个从网络中获取的数据
private void useDataFromNetwork(Lab5_NetworkData data){
    // 使用WebView Activity打开网页
    String urlString = data.getLink();
    Intent intent = new Intent(this, Lab5_WebViewActivity.class);
    intent.putExtra("url", urlString);
    startActivity(intent);
}
```

3. 在WebViewActivity初始化时，获取urlString参数，打开网页

1. 由于现在的网页包含的内容过多，因此要进行一系列设置，才能正常读取页面的数据

```
public class Lab5_WebViewActivity extends AppCompatActivity {
    private webView webView;
    private webSettings webSettings;
    private String urlString;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.lab5_webview_layout);

        // 获取WebView控件，传入的url字符串，webSettings属性
        webView = findViewById(R.id.Lab5_WebView);
        urlString = getIntent().getStringExtra("url");
        webSettings = webView.getSettings();

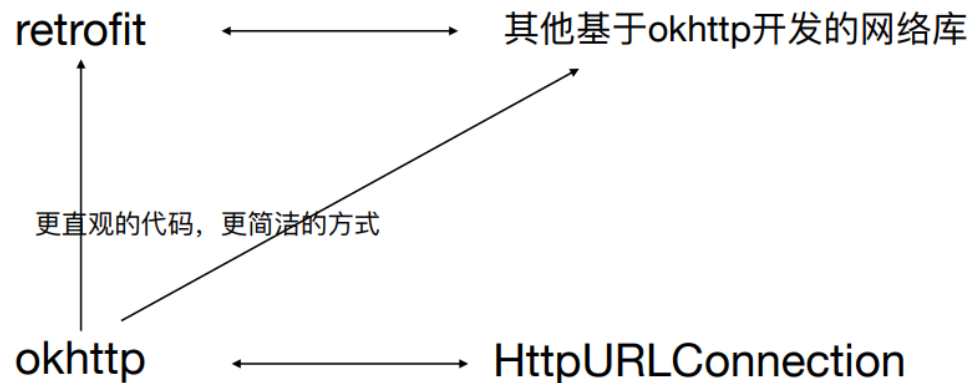
        // 设置WebView的属性
        webSettings.setJavaScriptEnabled(true);
        webSettings.setJavaScriptCanOpenWindowsAutomatically(true);
        webSettings.setSupportZoom(true);
        webSettings.setBuiltInZoomControls(true);
        webSettings.setLoadWithOverviewMode(true);
        webSettings.setAppCacheEnabled(true);
        webSettings.setDomStorageEnabled(true);

        // 使用WebView控件，打开对应网页
        webView.loadUrl(urlString);
    }
}
```

三、进阶实现： Retrofit(自学)

目前最广泛使用的 Android 网络框架：retrofit

持续维护：稳定性



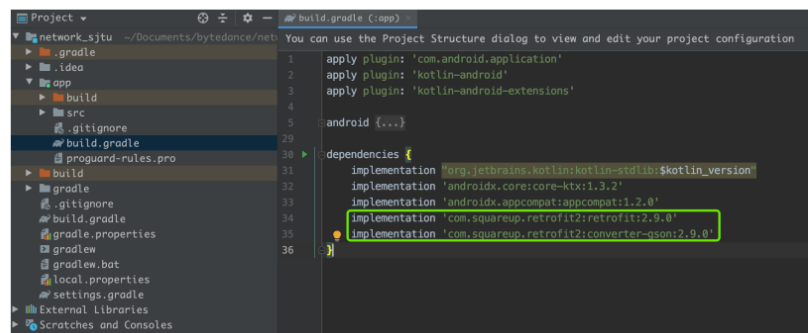
更好的生态环境：各种基于/扩展 okhttp 开发的库

更方便的异步操作：Android 自3.0开始需要在非 UI 线程进行网络请求

官方支持：Android 4.4开始 HttpURLConnection 底层用 okhttp 实现

其他优势：响应缓存等

添加 retrofit 依赖与 retrofit-gson 依赖



根据 json 创建 java 类

```
public class Repo {
    // 仓库名
    @SerializedName("name")
    private String name;

    //...

    public String getName() { return name; }
}
```

根据 api 信息（请求方法、URL、响应体结构等）创建 retrofit 所需要的接口方法

```
public interface GitHubService {

    /**
     * 假设我们现在传入的 username 参数为 JakeWharton，那么经过在程序运行时，经过 @Path 的匹配
     * 替换掉 {username} 之后就会变成 users/JakeWharton/repos
     */
    @GET("users/{username}/repos")
    Call<List<Repo>> getRepos(@Path("username") String userName,
                             @Query("page") int page,
                             @Query("per_page") int perPage);
}
```

@Path: named replacement in a URL path segment.

创建 retrofit 对象

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.github.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build();
```

GsonConverterFactory 是 retrofit 用来将 json 转换成 java 类的工具类，如果你想用 jackson 的，有 JacksonConverterFactory

用retrofit对象和接口类型动态创建接口服务对象

```
GitHubService service = retrofit.create(GitHubService.class);
```

调用接口服务对象的方法，获取数据

```
private void requestRetrofit(String userName) {
    GitHubService service = retrofit.create(GitHubService.class);
    Call<List<Repo>> repos = service.getRepos(userName, page, perPage: 10);
    repos.enqueue(new Callback<List<Repo>>() {
        @Override public void onResponse(final Call<List<Repo>> call, final Response<List<Repo>> response) {
            if (!response.isSuccessful()) {
                return;
            }
            final List<Repo> repoList = response.body();
            if (repoList == null || repoList.isEmpty()) {
                return;
            }
            page++;
            showRepos(repoList);
        }

        @Override public void onFailure(final Call<List<Repo>> call, final Throwable t) {
            t.printStackTrace();
        }
    });
}
```