

# 实验九

## Retrofit+RxJava+OkHttp 实现网络请求

这次实验请确保调试机器是联网状态！

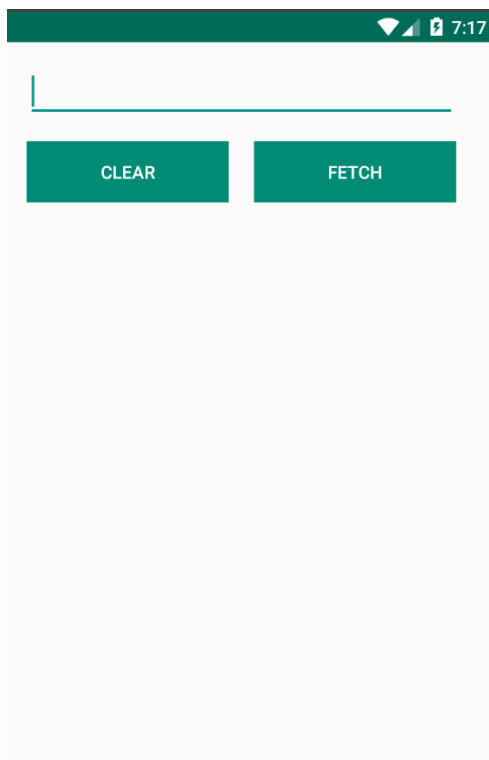
### 1. 实验目的

学习使用 Retrofit 实现网络请求

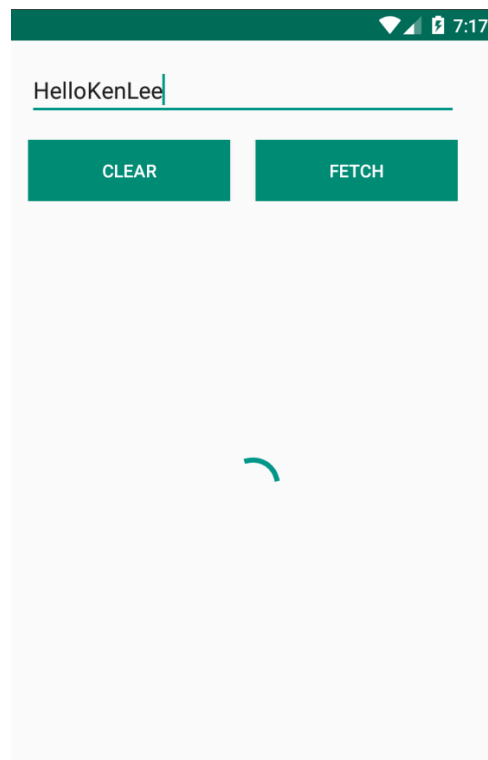
学习 RxJava 中 Observable 的使用

复习同步异步概念

### 2. 实验内容



主界面



搜索用户

HelloKenLee

CLEAR

FETCH

hellokenlee

id: 8079836

blog: <https://hellokenlee.github.io/>

搜索结果

YsingYang

CLEAR

FETCH

hellokenlee

id: 8079836

blog: <https://hellokenlee.github.io/>

WideLee

id: 2196814

blog:

YsingYang

id: 23610824

blog: <https://ysingyang.github.io>

搜索结果

YsingYang

CLEAR

FETCH

hellokenlee

id: 8079836

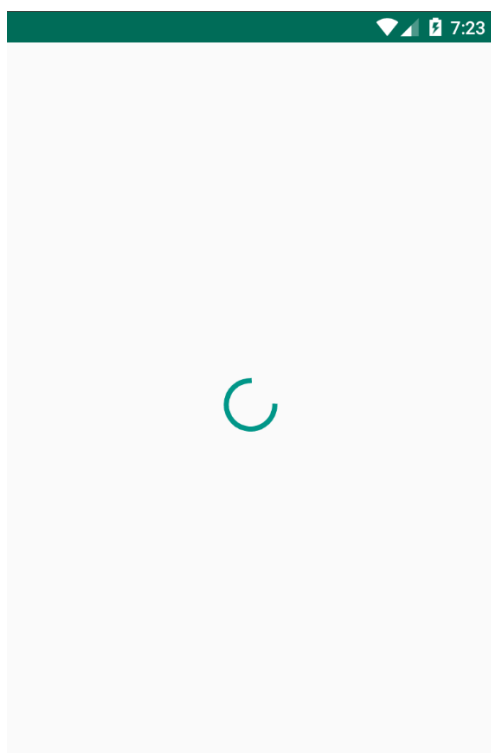
blog: <https://hellokenlee.github.io/>

WideLee

id: 2196814

blog:

长按删除



点击进入个人详情页面



详情信息获取显示

对于 User Model, 显示 `id`, `login`, `blog`

对于 Repository Model, 显示 `name`, `description`,  
`language`

(特别注意, 如果 `description` 对于 1 行要用省略号代替)

## 3. 基础知识

### 3.1. 搜索相应的 Github API

API 地址 : <https://developer.github.com/v3/>

User API 地址 :

<https://developer.github.com/v3/users/>

## 3.2. 界面设计

### RecyclerView

在主界面中使用到了 RecyclerView，具体使用可以参考实验三的实验文档。对于 RecyclerView 中的每一项 item 学习使用 **CardView**

需要添加的依赖如：`compile 'com.android.support:cardview-v7:26.+'`

CardView 的布局设计与正常布局设计类似，实验中不涉及太多 CardView 的属性，只需要添加 TextView 即可，更多属性可以参考：

<http://seniorzhai.github.io/2015/01/06/CardView%E7%9A%84%E4%BD%BF%E7%94%A8/>

### ProgressBar

因为网络请求需要一定的时间，这时候就需要将等待状态反馈给用户，在这次实验中可以学习基本 **ProgressBar** 的使用，最简单的可以直接设置其 **visibility** 属性来实现相应的效果

### ListItem

安卓官网上有许多 Component 的设计样例，对于 ListView 的 Item 也有相应的参考设计，具体可以参考

<https://material.io/guidelines/components/lists.html#lists-specs>

### 3.3. Retrofit

在本次实验中，会学习使用 Retrofit 实现网络请求。实现一个 API 的访问非常简单， 可以将 Retrofit 实现的网络请求分为以下步骤：

1. 定义 Model 类
2. 定义相应的访问接口(interface)
3. 构造 Retrofit 对象并设置相应的 URL 后， 调用即可获取到网络资源

#### Model

定义相应的 Model 类前， 可以先通过 API 查看获取到的数据格式， 如访问

<https://api.github.com/users/HelloKenLee>， 可以获取用户名为 HelloKenLee 的相应信息。

```
{
  "login": "hellokenlee",
  "id": 8079836,
  "avatar_url": "https://avatars1.githubusercontent.com/u/8079836?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/hellokenlee",
  "html_url": "https://github.com/hellokenlee",
  "followers_url": "https://api.github.com/users/hellokenlee/followers",
  "following_url": "https://api.github.com/users/hellokenlee/following{/other_user}",
  "gists_url": "https://api.github.com/users/hellokenlee/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/hellokenlee/starred{/owner}/{repo}",
  "subscriptions_url": "https://api.github.com/users/hellokenlee/subscriptions",
  "organizations_url": "https://api.github.com/users/hellokenlee/orgs",
  "repos_url": "https://api.github.com/users/hellokenlee/repos",
  "events_url": "https://api.github.com/users/hellokenlee/events{/privacy}",
  "received_events_url": "https://api.github.com/users/hellokenlee/received_events",
  "type": "User",
  "site_admin": false,
  "name": "KenLee",
  "company": null,
  "blog": "https://hellokenlee.github.io/",
  "location": "Guangzhou, China",
  "email": null,
  "hireable": null,
  "bio": "A CS student studying in Sun Yat-Sen University.",
  "public_repos": 24,
  "public_gists": 0,
  "followers": 12,
  "following": 7,
  "created_at": "2014-07-06T09:00:05Z",
  "updated_at": "2017-11-20T11:55:05Z"
}
```

我们并不需要显示上面的所有信息， 本次实验只需要获取 **login**, **id**, **blog** 这些信息即可。根据需求，就可以定义相应的 model class，如下所示

```
public class Github {
    private String login;
    private String blog;
    private int id;

    public String getLogin() { return login; }

    public String getBlog() { return blog; }

    public int getId() {
        return id;
    }
}
```

对于下一个 Activity (ReposActivity)，获取用户所有 **repositories** 信息所需要定义的 model 也与上述类似

## API Interface

实验需要给 retrofit 对象提供相应的 Interface，

Interface 的定义非常简单， 提供相应的 URL，返回类型与参数即可。

如访问用户信息的访问接口可以定义为

```
public interface GithubService {  
    @GET("/users/{user}")  
    Observable<Github> getUser(@Path("user") String user);  
}
```

其中 Observable 为 RxJava 中的类型。暂时可以理解为这个接口会返回一个 Github 类型（即 Model 中定义好的类）  
需要注意的是访问用户 repositories 返回的是一个 List，  
所以对于定义相应接口前先用 API 查看下返回类型。

## GsonConverter

在了解相应的 API interface 与 model 后， 需要了解下 **GsonConverter**， 因为我们从 API 获取到的是一个 JSON 格式的数据， 因此需要使用到 GsonConverter 将其“转换”为所需要的 model， 使用 GsonConverter 需要添加相应的依赖， 如

```
compile 'com.squareup.retrofit2:converter-gson:2.1.0'
```

相应的使用会在 Retrofit 构造时说明

## OkHttp

Retrofit 也是基于 OkHttp 的封装， 所以可以自己配置相应的 OkHttp 对象

```
private static OkHttpClient createOkHttp() {
    OkHttpClient okHttpClient = new OkHttpClient.Builder()
        .connectTimeout(10, TimeUnit.SECONDS) // 连接超时
        .readTimeout(30, TimeUnit.SECONDS) // 读超时
        .writeTimeout(10, TimeUnit.SECONDS) // 写超时
        .build();
    return okHttpClient;
}
```

更多详情可以参考该网页

<http://cdevlab.top/article/e57450c6/>

## Retrofit 对象

构造 Retrofit 对象实现网络访问。

```
private static Retrofit createRetrofit(String baseUrl) {
    return new Retrofit.Builder()
        .baseUrl(baseUrl) //设置baseUrl
        .addConverterFactory(GsonConverterFactory.create()) //添加GSON Converter
        .addCallAdapterFactory(RxJavaCallAdapterFactory.create()) //RxJavaCall Adapter
        .client(createOkHttp()) //okHttp
        .build();
}
```

使用 retrofit 创建相应访问 API 接口， 即可通过创建好的实例进行访问操作， 具体的 Retrofit 介绍可以查看课堂 PPT

## RxJava Observable

由于本次实验不是直接使用 Retrofit 的 Call 对象， 而是加入 RxJava 的 Observable 对象， 参考样例中使用的是 RxJava1， 与 RxJava2 有部分区别， 所以使用前， 明确添加上的依赖是 RxJava1 还是 RxJava2 的。

通过 retrofit 创建好相应的访问实例后， 调用相应的访问 API 的方法。例子如下所示。



```

service.getUser(User)
    .subscribeOn(Schedulers.newThread()) //新建线程进行网络访问
    .observeOn(AndroidSchedulers.mainThread()) //在主线程处理请求结束
    .subscribe(new Subscriber<Github>() {

        @Override
        public final void onCompleted() {
            System.out.println("完成传输");
            removeWait();
        }

        @Override
        public void onError(Throwable e) {
            Toast.makeText(MainActivity.this, e.hashCode() + "请确认你搜索的用户存在", Toast.LENGTH_SHORT).show();
            Log.e("Github-Demo", e.getMessage());
            removeWait();
        }

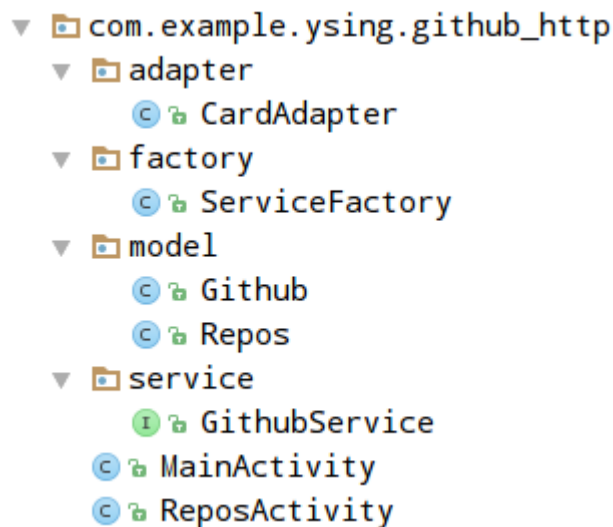
        @Override
        public void onNext(Github github) {
            cardAdapter.addData(github);
        }
    });

```

onCompleted 函数为请求结束时调用的回调函数，onNext 表示收到每一次数据时调用的函数。onError 表示请求出现错误时调用的函数。

熟悉以上知识点后，就可以完成这次实验了。这次实验只涉及到了 GET 请求，有兴趣的同学可以自己尝试使用 POST 等其他请求。

## 4. 项目结构与参考依赖



```
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:26.+'
    compile 'com.android.support:recyclerview-v7:26.+'
    compile 'com.android.support:cardview-v7:26.+'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    testCompile 'junit:junit:4.12'
    compile 'io.reactivex:rxjava:1.0.+'
    compile 'io.reactivex:rxandroid:0.23.+'
    compile 'com.squareup.retrofit2:retrofit:2.1.0'
    compile 'com.squareup.retrofit2:adapter-rxjava:2.1.0'
    compile 'com.squareup.retrofit2:converter-gson:2.1.0'
    compile 'com.android.support:design:26.0.0-alpha1'
}
```

## 5. 检查内容

1. Progress 正确显示
2. 正确获取访问结果
3. 点击用户获取用户 Repositories

## 6. 注意事项

1. 确保调试设备联网
2. 对于 description, 若多于一行用省略号代替

## 7. 参考网页

1. RxJava
  - 1.1 [初学者 RxJava1 教程](#)
  - 1.2 [RxJava Github](#)
2. Retrofit
  - 2.1 [Retrofit 开发](#)
  - 2.2 [Retrofit 改进](#)
  - 2.3 [Retrofit 官方文档](#)
3. OkHttp
  - 3.1 [OkHttp 官方文档](#)
4. CardView
  - 4.1 [CardView 设计](#)
5. List Design
  - 5.1 [官方 Design 示例](#)

## 8. 提交说明

1. Deadline: 下一次实验课前晚上 23:59;
2. 提交作业地址 : <ftp://edin.sysu.edu.cn>
3. 文件命名及格式要求: 学号\_姓名\_labX.zip (姓名中文拼音即可)
4. 目录结构

```
15331111_huashen_lab1 --  
|  
-- lab1实验报告.pdf  
|  
-- lab1_code (包含项目代码文件)
```

提交之前先 clean !

