中山大学数据科学与计算机学院本科生实验报告

（2017 年秋季学期）

课程名称：手机应用平台开发　　任课教师： 刘宁

年级	15 级	专业（方向）	软件工程数媒方向
学号	15331016	姓名	陈桂燕
电话	13719346314	Email	735594896@qq.com
开始日期	2017/12/21	完成日期	2017/12/22

一、 实验题目：实现网络请求

【目的】

1. 学习使用 Retrofit 实现网络请求

2. 学习 RxJava 中 Observable 的使用

3. 复习同步异步概念

二、 实现内容

利用 Github 的 API 接口，要求实现一个搜索引擎，能对该 API 里的 user 进行搜索并可点击进入 repos 详情界面。搜索结果由 recyclerView+cardView 展现，并利用 progressBar 向用户反馈网络请求状态。Repos 详情界面利用 listView 展示。

三、 实验过程

1. 前提说明

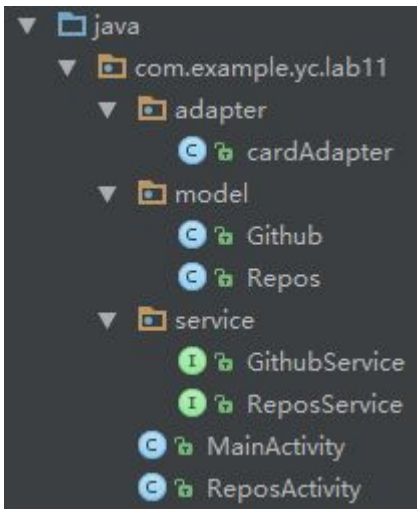
AndroidManifest 里的权限请求：

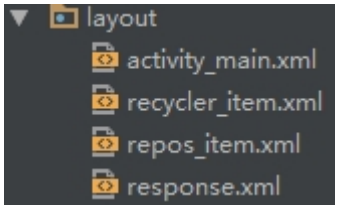
```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Gradle 新添加的依赖：

```
compile 'com.android.support:recyclerview-v7:25.2.0'
compile 'com.android.support:cardview-v7:25.2.0'
compile 'com.squareup.retrofit2:retrofit:2.0.2'
compile 'com.squareup.retrofit2:converter-gson:2.0.2'
compile 'com.squareup.okhttp3:okhttp:3.2.0'
compile 'io.reactivex:rxjava:1.0.+'
compile 'io.reactivex:rxandroid:0.23.+'
compile 'com.squareup.retrofit2:adapter-rxjava:2.0.2'
```

文件目录：





2. 界面设计

首先，在 res/value/colors 里新增以下颜色：

```
    <color name="colorBasic">#FF008B75</color>
```

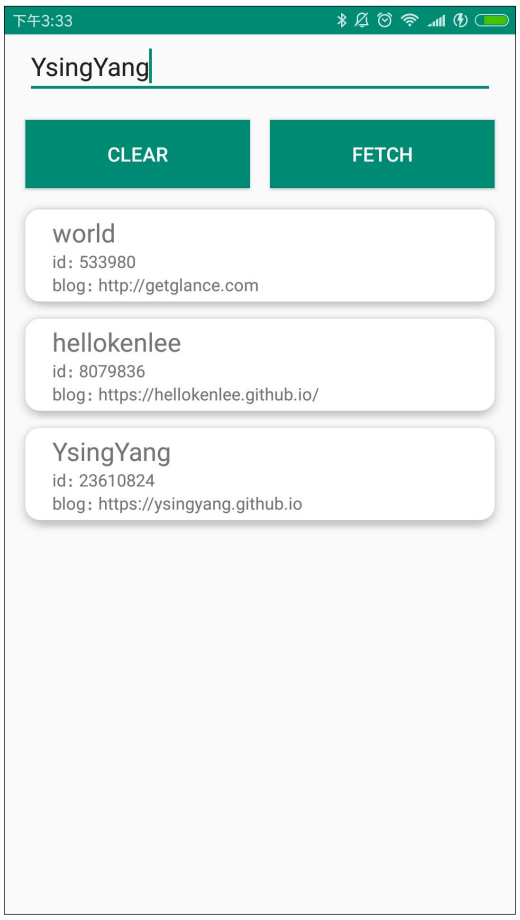
然后，在 res/value/style 修改如下，实现 app 的主题颜色皆为绿色，并且除去标题栏。

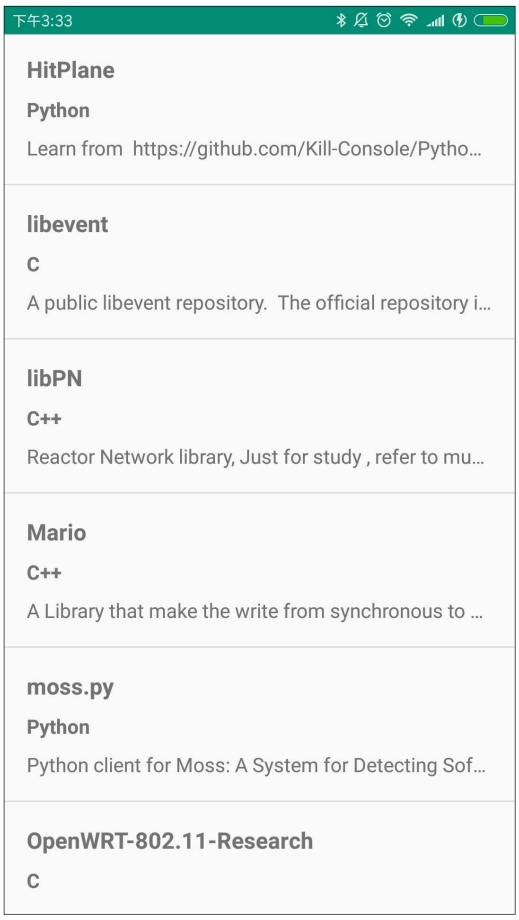
```
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorBasic</item>
    <item name="colorPrimaryDark">@color/colorBasic</item>
    <item name="colorAccent">@color/colorBasic</item>
</style>
```

在搜索主界面 activity_main.xml 里添加 EditView 和两个 button 控件后 ,再添加一个位于中心的 progressBar，先设其为不可见，开始网络请求时可见，请求结束后又设置不可见。

```
<ProgressBar
    android:id="@+id/progress" android:visibility="gone"/>
```

主界面下方的搜索结果用 recyclerView+cardView 呈现，点击 item 进去的 repos 详情界面用 listView 实现，效果如下：





先设置 recyclerView 每个 item 的样式，利用 cardView 来设置其外壳样式：

```
<?xml version='1.0' encoding='utf-8' ?>
    <android.support.v7.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:card_view="http://schemas.android.com/apk/res-auto"
```

cardView 的一些属性设置如下：

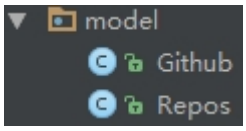
```
card_view:cardCornerRadius="10dp"
card_view:cardElevation="5dp"
card_view:contentPadding="4dp"
android:foreground="?attr/selectableItemBackground">
```

然后再用 linearView 来设置 item 里 TextView 的格式即可。值得注意的是，recyclerView 要单独创建 Adapter+viewholder，并且要自己实现“点击+长按”的函数接口，具体看实验代码，这次我不是按照之前实验三的文档，而是模仿书籍《第一行代码》里的相关教程来创建的。

至于 listView，也是先设计每个 item 的布局样式，在每个 item 所展示的三个 TextView 控件信息里，前两个设置 textStyle=” bold” 起到加粗作用，第三个要设置只能显示一行文字，未能显示的文字则以省略号结尾呈现。

```
android:maxLines="1"
android:ellipsize="end"
```

3. Retrofit 实现网络请求



首先定义两个 model 数据结构类。

Github 里包含的 String 信息分别是 login、id、blog；

Repos 里包含的 String 信息分别是 name、language、description。

然后再定义相应的两个访问接口：

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

```
public interface ReposService {
    @GET("/users/{user}/repos")
    Call<List<Repos>> listRepos(@Path("user") String user);
}
```

在主界面对应的 MainActivity.java 里，每次点击 fetch 按钮，都发生如下事件：

先利用 ConnectivityManager 的 getActiveNetworkInfo()来判断当前手机是否连网，若判断为是，便再发送网络请求。

```
fetch.setOnClickListener((view) -> {
        //将等待状态反馈给用户
        progress.setVisibility(View.VISIBLE);

        /**先判断是否有可用网络
         *使用ConnectivityManager获取手机所有连接管理对象
         *使用manager获取NetworkInfo对象
         *最后判断当前网络状态是否为连接状态即可。
         */
        ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo networkInfo = cm.getActiveNetworkInfo();
        if ((networkInfo == null) || !networkInfo.isConnected()) {
            progress.setVisibility(View.GONE);
            Toast.makeText(MainActivity.this, "当前网络不可用", Toast.LENGTH_SHORT).show();
            return;
        } else {//当前网络可用,进行数据获取
            //使用Retrofit实现网络请求
            retrofit();
        }
});
```

使用 Retrofit 发送网络请求，总体结构如下：

构造 Retrofit 对象并设置相应的 URL 后，调用即可获取到网络资源。

```
void retrofit() {
    //初始化OkHttpClient
    OkHttpClient client = new OkHttpClient();
    //创建Retrofit
    Retrofit retrofit = new Retrofit.Builder()
            .client(client)//设置OKHttpClient
            .baseUrl("https://api.github.com") // 设置网络请求的Url地址
            .addConverterFactory(GsonConverterFactory.create()) // 设置数据解析器
            .build();
    //获取GitHub的API
    GithubService gitHubAPI = retrofit.create(GithubService.class);

    //异步调用
    Call<Github> userCall = gitHubAPI.getUser(search.getText().toString());
    userCall.enqueue(new Callback<Github>() {...});
}
```

对于网络资源的获取，我做了以下的调用处理：

若获取到了所要的 model 结构则新增 recyclerView。其余情况再做相应的反馈。

```
userCall.enqueue(new Callback<Github>() {
    @Override
    public void onResponse(Call<Github> call, Response<Github> response) {
        Github body = response.body();
        progress.setVisibility(View.GONE);
        if (body == null) {
            Toast.makeText(MainActivity.this, "该用户不存在", Toast.LENGTH_SHORT).show();
        } else {
            //用户存在，添加item
            userList.add(new Github(body.getLogin(), "id: "+body.getId(), "blog: "+body.getBlog()));
            adapter.notifyDataSetChanged();
        }
    }
    @Override public void onFailure(Call<Github> call, Throwable t) {
        progress.setVisibility(View.GONE);
        //判断是否被cancel掉了
        if (call.isCanceled()) {
            Toast.makeText(MainActivity.this, "the call is canceled", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(MainActivity.this, t.toString(), Toast.LENGTH_SHORT).show();
        }
    }
});
```

主界面 MainActivity.java 的网络获取便是这样子。

另外，点击 clear 按钮需清除所有的 recyclerView 列表：

```
clear.setOnClickListener((view) → {
        userList.clear();
        adapter.notifyDataSetChanged();
});
```

recyclerView 的 item 需要实现长按删除，点击进入 repos 详情界面，这里我是将用户 login 传递给下一界面 ReposActivity.java。

```
adapter.setOnClickListener(new cardAdapter.OnItemClickListener() {
    @Override
    public void onClick(int position) {
        //单击事件,进入详情界面,将login传递过去
        Intent it = new Intent(MainActivity.this, ReposActivity.class);
        it.putExtra("login", userList.get(position).getLogin());
        startActivity(it);
    }

    @Override
    public void onLongClick(int position) {
        //长按事件,删除该item
        userList.remove(position);
        adapter.notifyDataSetChanged();
    }
});
```

若来到第二个界面 ReposActivity.java，首先需获取上一界面所传递的用户的 login：

```
//获取传递的参数
Intent intent = getIntent();
String login = intent.getStringExtra("login");
```

构建 listView 的 Adapter：

```
simpleAdapter = new SimpleAdapter(ReposActivity.this, data, R.layout.repos_item,
        new String[]{"name", "language", "description"}, new int[]{R.id.name, R.id.language, R.id.description});
listView.setAdapter(simpleAdapter);
```

构造 Retrofit 对象：

```
//初始化OkHttpClient
OkHttpClient client = new OkHttpClient();
//创建Retrofit
Retrofit retrofit = new Retrofit.Builder()
        .client(client)//设置OKHttpClient
        .baseUrl("https://api.github.com") // 设置网络请求的Url地址
        .addConverterFactory(GsonConverterFactory.create()) // 设置数据解析器
        .build();
//获取API
ReposService ReposAPI = retrofit.create(ReposService.class);
```

利用上一界面所传递的 login 构造相应的 url，然后进行网络请求，若能获取相应的资源则进行处理。由于得到的是一个 list 变量，则先获取其长度，再循环将其一个个加入该 activity 里创建的 List<Map<String, String>> data 里，最后再更新 adapter 即可呈现所要的界面。

```
Call<List<Repos>> userCall = ReposAPI.listRepos(login);
userCall.enqueue(new Callback<List<Repos>>() {
    @Override
    public void onResponse(Call<List<Repos>> call, Response<List<Repos>> response) {
        progress.setVisibility(View.GONE);
        List<Repos> body = response.body();
        if (body == null) {
            Toast.makeText(ReposActivity.this, "该用户不存在", Toast.LENGTH_SHORT).show();
        } else {
            //用户存在，添加item
            int size = body.size();
            for (int i = 0; i < size; i++) {
                Map<String, String> map = new HashMap<>();
                map.put("name", body.get(i).getName());
                map.put("language", body.get(i).getLanguage());
                map.put("description", body.get(i).getDescription());
                data.add(map);
            }
            simpleAdapter.notifyDataSetChanged();
        }
    }
    @Override
    public void onFailure(Call<List<Repos>> call, Throwable t) {
        progress.setVisibility(View.GONE);
        //判断是否被cancel掉了
        if (call.isCanceled()) {
            Toast.makeText(ReposActivity.this, "the call is canceled", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(ReposActivity.this, "the call is not canceled", Toast.LENGTH_SHORT).show();
        }
```

四、 实验思考及感想

先说下 recyclerView，虽然之前实验已经接触过了，但是这次重新使用还是觉得上手稍稍有点难度，特别是自己要重新创建“点击/长按”事件的接口。

然后关于 retrofit 实现网络请求，自己看了几遍实验文档还是搞不大懂，后来还是网上找了份教程，认真看了一遍然后跟着做，庆幸的是很快就成功了。

至于 repos 获取的信息，最开始我还烦恼怎么将其一组一组获取然后添至 listView 里，网上也找不到资料 = =！后来在吃饭的时候突然想到这不就是 java 里一个简单的集合数组嘛，利用 size()获取长度然后循环就可以了……感觉自己之前想太复杂了，如果抱着“应该会挺难”的心态，那么往往会忘记那些简单的基础内容。