中山大学移动信息工程学院本科生实验报告

（2017 年秋季学期）

课程名称：移动应用开发　　任课教师： 郑贵锋

年级	大三	专业（方向）	移动互联网
学号	15352408	姓名	张镓伟
电话	13531810182	Email	709075442@qq.com
开始日期	2017.12.18	完成日期	2017.12.20

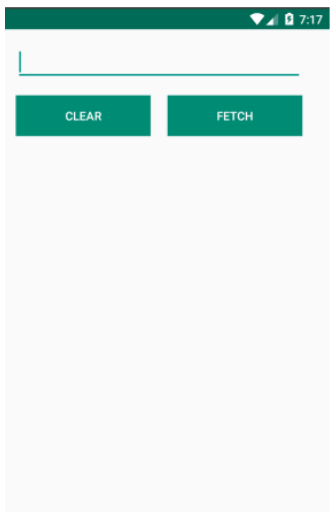
一、 实验题目

Retrofit+Rxjava+OkHttp 实现网络请求

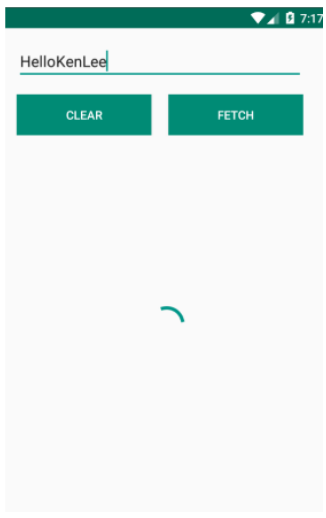
二、 实验目的

1. 学习 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。

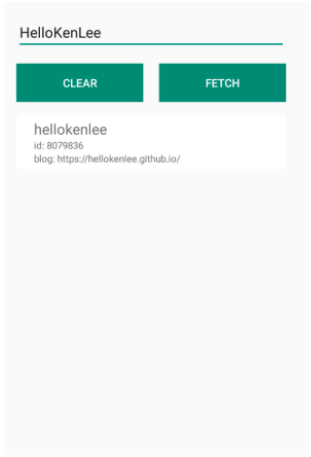
三、 实验内容



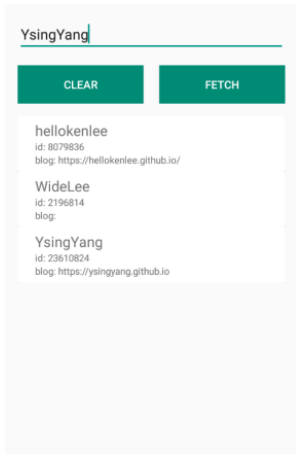
主界面



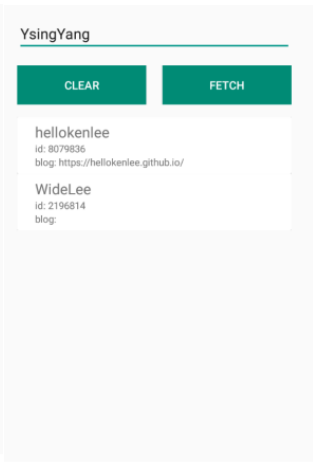
搜索用户



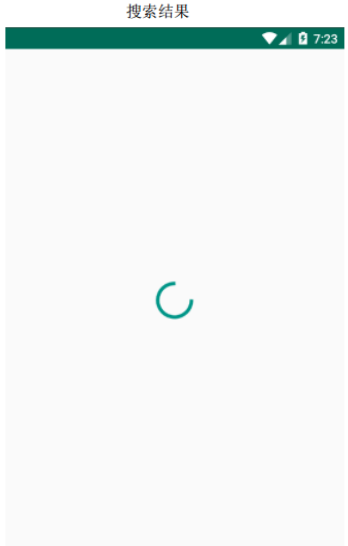
搜索结果



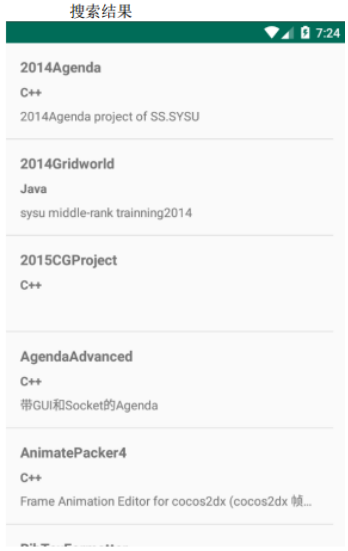
搜索结果



长按删除



点击进入个人详情页面

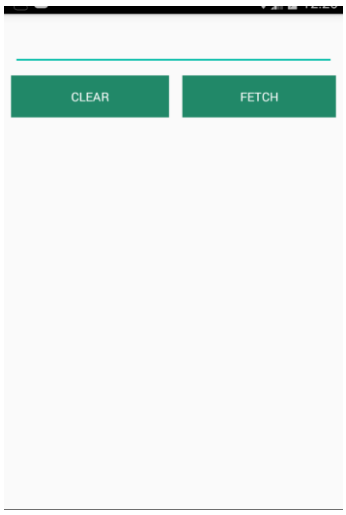


详情信息获取显示

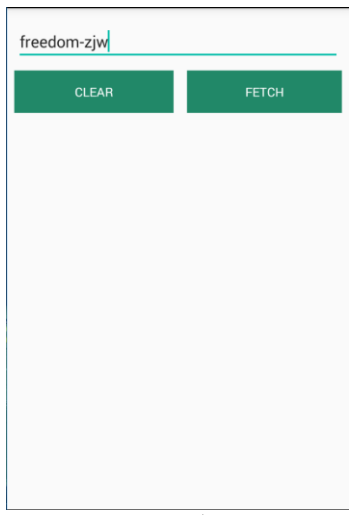
对于 User Model, 显示 id, login, blog
对于 Repository Model, 显示 name, description,language
(特别注意， 如果 description 对于 1 行要用省略号代替)

四、 课堂实验结果

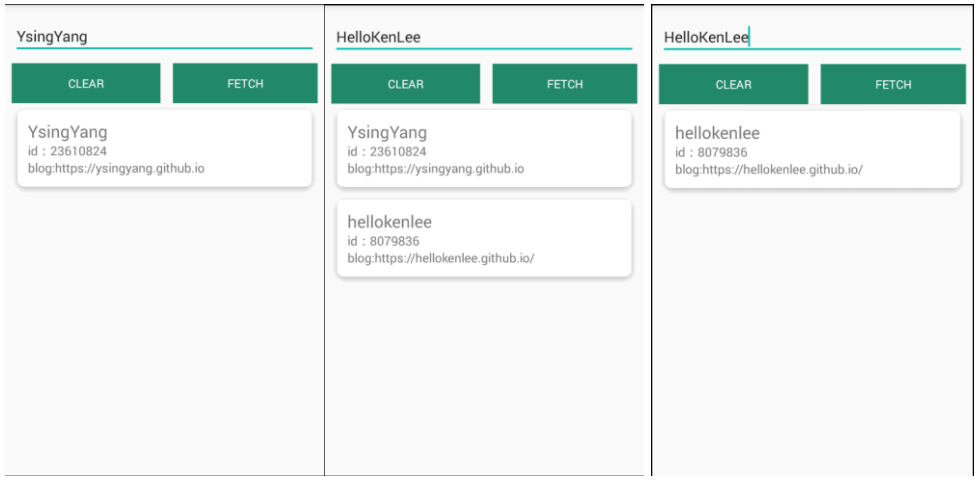
（1） 实验截图



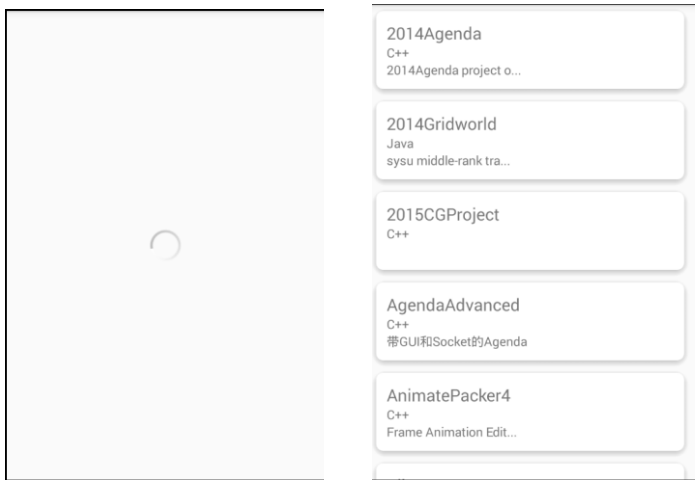
主界面



搜索用户



搜索结果　　搜索结果　　长按删除条目。

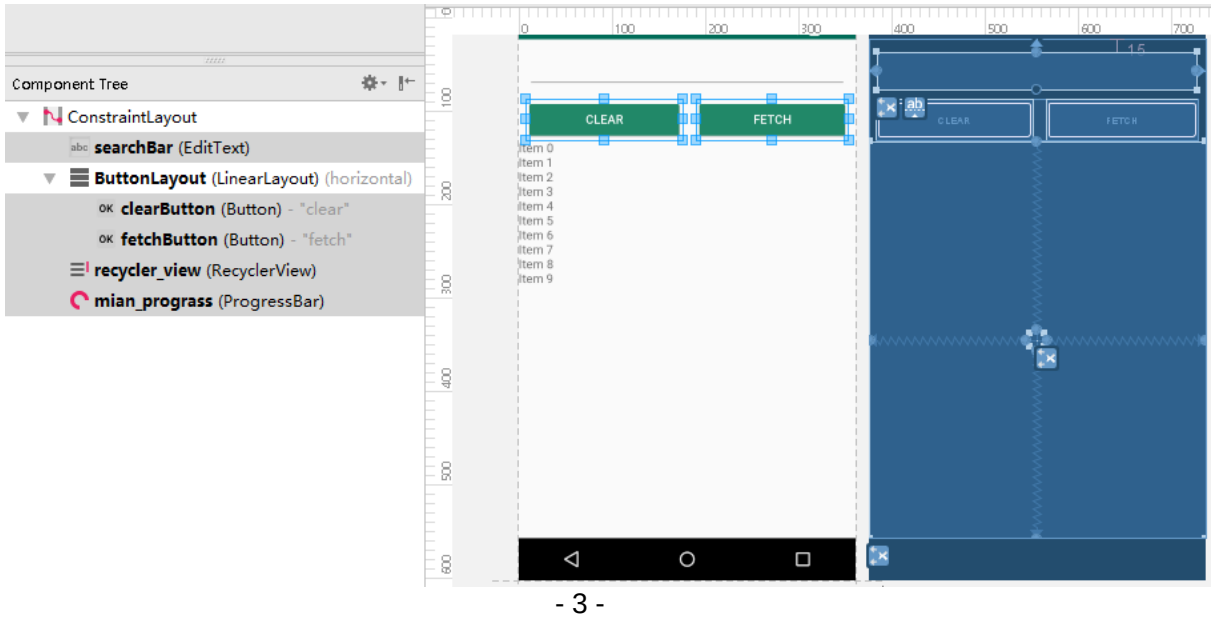


点击进入个人详情页　　详情信息获取显示

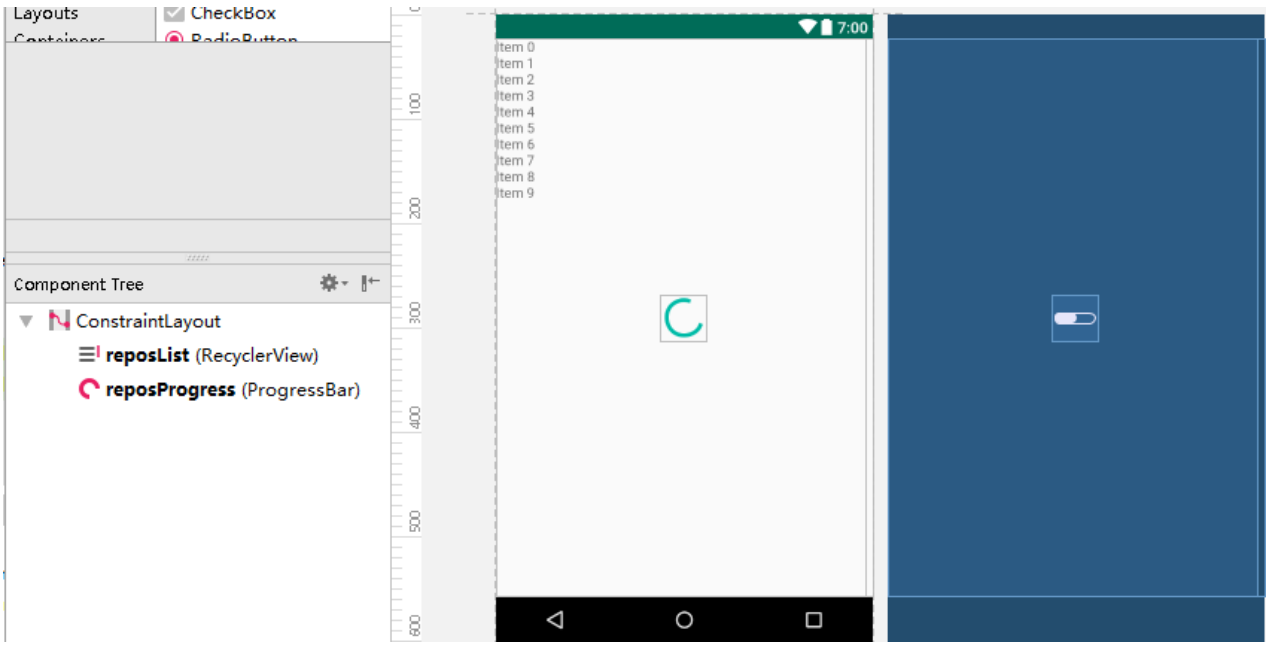
（2） 实验步骤以及关键代码

1.布局部分

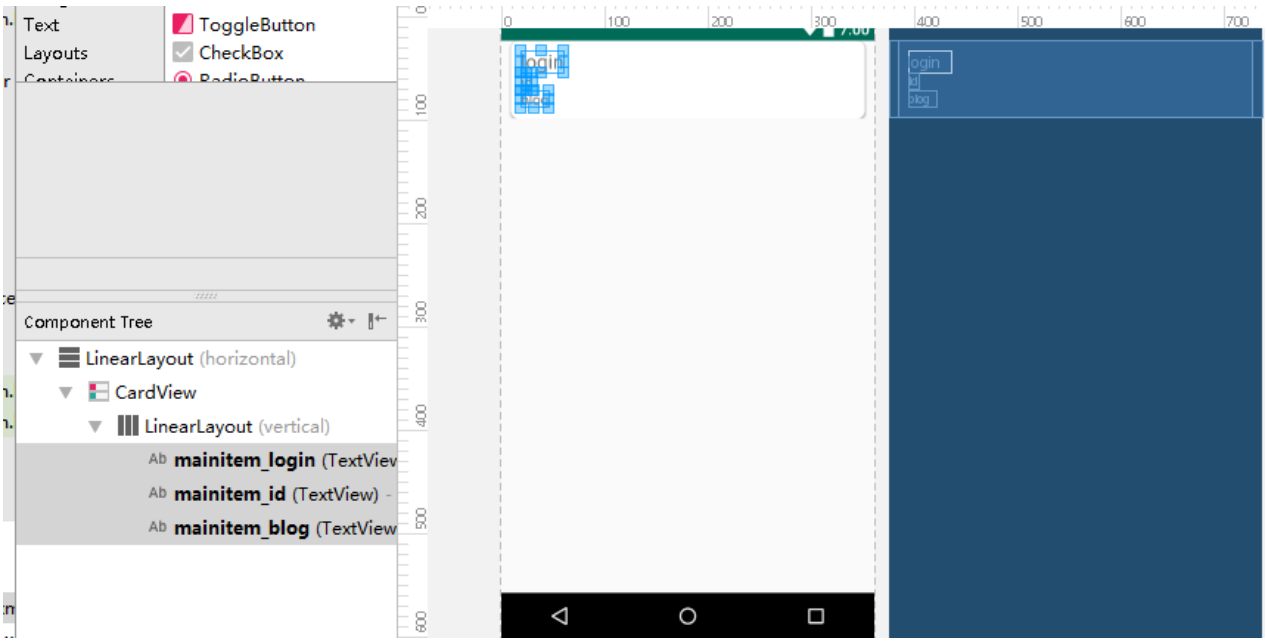
主界面布局:采用约束布局，从上到下，第一个是 EditText 类型的搜索栏 searchBar，然后是一个水平线性布局，包含 clear 按钮和 fetch 按钮，剩下的位置是 RecyclerView 用来展示搜索结果，最后是一个固定在屏幕中间的 ProgressBar（默认不可见）。



User 详情页布局：这个页面比较简单，就是一个 constrainLayout 包含一个 RecyclerView 用来展示 user 的每一个 repository 信息。然后是一个固定在屏幕中央的 ProgressBar，用来在等待搜索结果时候显示等待状态。

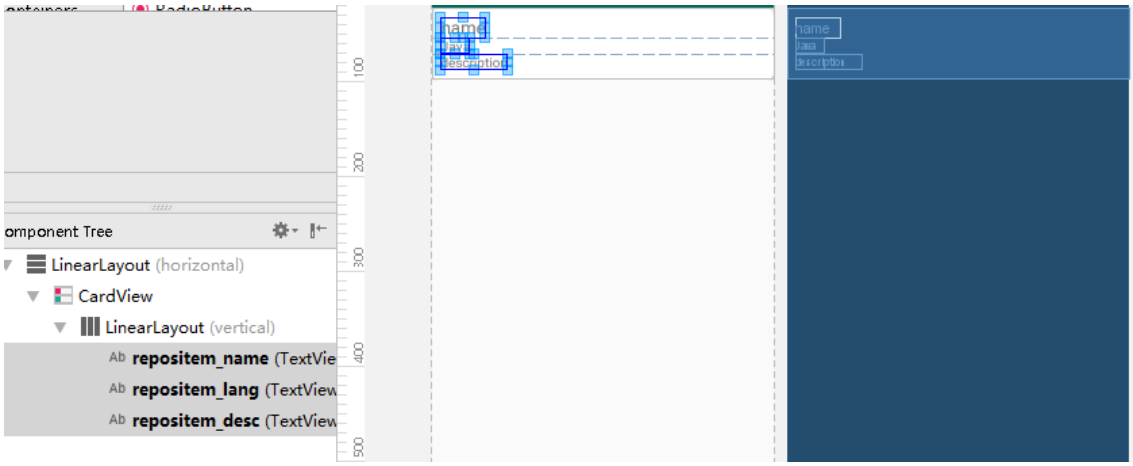


用户信息 item 布局：用一个 linearLayout，包含一个 cardview，cardview 是这次实验新用到的布局控件，它经常被作为容易用于一个 ListView 或一个 RecyclerView 的 item 布局中，作用是可以添加圆角和阴影，使每一项 item 更有层次感，容易被用户区分。这里我们用 cardview 显示一个用户的信息，用户的信息使用一个垂直线性布局布在 cardview 里面，信息包括登录名 login，id，以及 blog



Repository 界面的 item 布局：这个与前面的 user 信息的 item 布局是一样的，只是显示的内容改

为了该用户每一个 repository 的信息：name、language 以及 decription



1.java 部分

首先我们需要理清一下代码逻辑，我们这次的实验是根据需要搜索的内容，向 github 发起 http 请求，将获得 json 信息用 gson 库解析了之后显示出来。显示信息的时候使用了 recyclerview 显示每一项搜索结果，由于 recyclerview 是照搬 lab3 的，这次不再累述，主要写跟 http 请求相关的部分。

这次实验要求使用 Retrofit+Rxjava+OkHttp 实现网络请求，那么我们首先要搞清楚这三者是什么，关系是什么。OkHttp 是一个实现网络请求框架，它可以实现常用的 get、post 请求等，但是需要我们自己编写实现代码。而 Retrofit 是对 OkHttp 的封装，它可以不需要我们写 get、post 方法具体的实现，而是通过提供注解的构建方式自动生成我们定义的接口的实现类。Rxjava 则是处理异步操作的类，在这里我们希望自定义接收的数据存储类型，通过 Rxjava 的被观察者模式 observable 实现接收数据，对数据进行处理的操作。

A.首先我们定义接收的数据类型，即 model 类，这里，我定义了 Github 和 Repos 类，分别用于存储用户信息，以及用户的 repository 信息类（只需要定义变量和实现 get 方法）。

```
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; }
}
```

Github.java

```
public class Repos {
    private String name;
    private String language;
    private String description;
    private String html_url;

    public String getName() {return name;}
    public String getLanguage() { return language; }
    public String getDescription() { return description; }
    public String getHtml_url() {return html_url;}
}
```

Repos.java

B.(GithubService.java)创建一个 api 服务接口，因为需要给 retrofit 对象提供相应 interface。Interface 的定义非常简单， 提供相应的 URL，返回类型与参数即可。这里注意它通过注解声明网络请求方法和参数，如下面@GET(url) 就代表使用 get 方法请求 url 路径下的内容，如果 url 中有动态部分，比如下图的 user（需用大括号括起来表示动态变量），那么使用注解 @Path 去动态修改它的值。下面的 Observable<Github> getUser()表示返回的类型是一个 Github 类型，并且获取 Github 类中的 User 信息，加上 Observable 是为了在其他地方对该数据使用 Rxjava 的异步处理功能。

```
public interface GithubService { //创建api 服务接口 创建接口类，使用注解声明网络请求的方法和相关参数
    //请求路径，请求路径中可以包含参数，并在参数中使用 @PATH 注解来动态改变路径，
    //比如路径 api/v1/user/{id}，使用注解 @PATH("id") Long id 即可改变路径中 {id} 请求时的值。
    @GET("/users/{user}")
    Observable<Github> getUser(@Path("user") String user);

    @GET("/users/{user}/repos")
    Observable<List<Repos>> getUserRepos(@Path("user") String user);
}
```

C.(ServiceFactory.java)创建一个服务工厂类，这个类完成两个功能，一是创建和配置一个 okHttpclient 对象，第二是创建和配置一个 Retrofit 对象，在创建 Retrofit 对象中，配置 CallAdapterFactory 为 RxJavaCallAdapterFactory.create() 用来将返回的数据转换为 RxJava 的 Observable 对象。配置 ConverterFactory 为 GsonConverterFactory.create() 用来使用 GSON 将网络数据序列化为可用对象。

```
public class ServiceFactory {
    public static Retrofit createRetrofit(String baseUrl) {
        return new Retrofit.Builder()    //创建一个新的Retrofit对象
                .baseUrl(baseUrl) //设置baseURL，后续访问都是这个url目录下的
                .addConverterFactory(GsonConverterFactory.create()) //添加GSON Converter
                .addCallAdapterFactory(RxJavaCallAdapterFactory.create()) //RxJavaCall Adapter
                .client(createOkHttp()) //okHttp
                .build();
    }

    public static OkHttpClient createOkHttp() {  //自己配置相应的okHttp
        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;
    }
}
```

D.(MainActivity.java) 使用 retrofit 初始化我们创建的接口 GithubService，获取到 GithubService 实例之后就可以使用该实例发起请求.

```
Retrofit GithubRetrofit = ServiceFactory.createRetrofit("https://api.github.com");
githubservice = GithubRetrofit.create(GithubService.class);
```

E.(MainActivity.java),点击 fetch 按钮，调用之前创建好的 GithubService 实例发起请求。onCompleted 函数为请求结束时调用的回调函数，这里是将 Prograss 设为不可见， onNext

表示收到每一次数据时调用的函数，此处我们的操作是将新接收的数据加进 recyclerview 的 adapter 中使其得以显示给我们看。 onError 表示请求出现错误时调用的函数，这里我们抛出错误信息。

```
//点击fetch按钮，根据searchbar输入的内容搜索对应user的信息
fetchButton.setOnClickListener((v) -> {
        String User = searchBar.getText().toString();
        waitProgress.setVisibility(View.VISIBLE); //将Progress设为可见
        githubservice.getUser(User)  //用getUser method 访问对应url
                .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果
                .subscribe(new Subscriber<Github>() {
                    @Override
                    public void onCompleted() {
                        //请求结束时调用的回调函数，这里是完成了请求就让等待的Progress不可见
                        waitProgress.setVisibility(View.GONE);
                    }

                    @Override
                    public void onError(Throwable e) {
                        //有错位的时候抛出提示信息
                        Toast.makeText(MainActivity.this, e.getMessage()+"确认你搜索的用户存在", Toast.LENGTH_LONG).show();
                        waitProgress.setVisibility(View.GONE);
                    }

                    @Override
                    public void onNext(Github github) {
                        //onNext 函数是收到一次数据时调用的函数，收到的是Github类型的函数
                        //处理操作是将新数据加入到adapter中
                        cardAdapter.addData(github);
                    }
                });
});
```

F.(ReposActivity.java)，这里调用 retrofit 的实例去获取用户的 repository，直接写在 onCreate 计科，不需要特地通过某个控件去触发，因为我们切到这个页面已经知道要取哪个 user 的信息了，user 信息通过 intent 从 MainActivity 传递过来。这里的操作与 main 中不同的是，是对 Repos 型的数据操作，而 main 中是对 Github 型数据操作。

```
//调用retrofit创建好的访问实例去获取用户的repository信息
githubService.getUserRepos(login)
        .subscribeOn(Schedulers.newThread())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(new Subscriber<List<Repos>>() {
            @Override
            public void onCompleted() { reposProgress.setVisibility(View.GONE); }

            @Override
            public void onError(Throwable e) {
                Toast.makeText(ReposActivity.this, e.getMessage()+"确认你搜索的用户存在", Toast.LENGTH_LONG).show();
                reposProgress.setVisibility(View.GONE);
            }

            @Override
            public void onNext(List<Repos> repos) {
                //将数据添加进adapter以在recyclerview中显示
                for(int i = 0; i < repos.size(); i++) {
                    cardAdapter.addData(repos.get(i));
                }
            }
        });
```

G，至于 RecyclerView 以及其 Adapter 的编写是 lab3 的内容，直接拿过来稍微改一改就可以了，不是本次实验的重点，不再累述。

（3） 实验遇到困难以及解决思路

1. 一开始还对 Retrofit、okHttp 以及 Rxjava 有点懵，弄不清楚三者的关系，导致不知道如何下手编写代码，但是通过自己查询资料，弄清楚了三者的关系（如前文所述），那么代码的编写思路一下子就清晰了起来，后面编写代码基本就没有遇到其他障碍了。

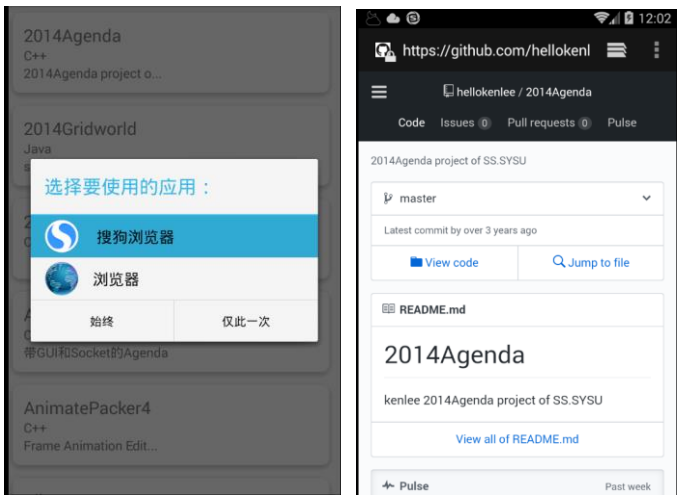
五、 课后实验结果

本次实验只要求获取信息并显示就可以了，但是我在最后添加了一个功能，就是在 ReposActivity 点击某个的 repository 的话，会通过浏览器打开相应的 github 仓库页面。实现方法很简单，通过使用 Uri 跳转就可以了。假如要跳转的页面的 url 是 a，那么只需要 `Uri uri = Uri.parse(a)` 这样添加一个 url，然后用 intent 和 startactivity 传递就可以了。具体如下：

```
cardAdapter.setOnItemClickListener(new CardAdapter.OnItemClickListener() {
    @Override
    public void onClick(int position) { //点击跳转到项目具体网页
        Intent intent = new Intent();
        intent.setAction("android.intent.action.VIEW");
        Uri content_url = Uri.parse(cardAdapter.getData(position, "html_url"));
        intent.setData(content_url);
        startActivity(intent);
    }

    @Override
    public boolean onLongClick(int position) { return false; }
});
```

效果：



六、 课后思考及感想

这次实验我们主要学习的是网络通信。这个实验一开始困扰我的是 Retrofit 和 okhttp 以及 Rxjava 之间的关系。所有的网络通信核心任务都是客户端与服务器进行数据交互，而在 Android 中，okhttp 可以完成网络的链接和请求，Retrofit 将 Okhttp 的操作封装起来的一个网络请求库，它简化了我们编写网络请求的代码，比如 get 方法只需要通过注解就可以生成我们接口的实现类，不用我们自己写如何实现。而 Rxjava 是一个实现异步操作库，这里我们使用

Rxjava 的被观察者模式（observable）去接收网络请求获得的数据，这样当一个新数据到来的时候，它会通知我们，然后我们可以通过重写提供好的方法进行异步处理数据。
这次实验的收获还是很大的，知道了如何进行网络请求，对后面大作业的帮助会很大。