移动应用程序开发实验报告

lab9: Retrofit + RxJava + OkHttp 实现网络请求

姓名	学号	年级	专业	电话	邮箱	日期
王若曦	15352319	15级	软件工程	15354222552	1511330303@qq.com	2017.12.16-2017.12.17

实验目的

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

实验题目

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)

实验过程

整体思路分析：

> 本次实验通过网络访问调用GitHub的API返回对应的值。所以要用到Retrofit实现网络请求。过程分为三步：1. 定义Model类，即根据API返回值的数据格式定义数据类型，这里需要定义一个GitHub类和一个Repos类，分别表示用户和库；2. 定义相应的访问接口，提供响应的URL，返回类型和参数即可。这里要注意的是GitHub类型和Repositories类型返回的类型不同，repositories返回的是一个List，所以接口也要定义为List； 3. 构造Retrofit对象并设置相应的URL，然后调用即可获取网络资源。然后根据输入进行网络访问GitHub，取回其库和库中的数据信息。
>
> 布局方面主界面列表使用RecyclerView，item使用CardView，库界面使用listview即可。具体使用方法在后面详述。

定义Model类

首先我们要定义从GithHub取回数据后使用的数据类型。根据API查看获取到的数据格式，由于我们只需要显示少量部分内容，所以只定义用到的部分即可。所以对于GitHub和Repos类的定义为：

```
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;
    }
}

public class Repos {
    String name;
    String description;
    String language;

    public String getName() {
        return name;
    }

    public String getDescription() {
        return description;
    }

    public String getLanguage() {
        return language;
    }
}
```

定义访问接口

利用retrofit进行访问，需要为其提供相应接口interface，其定义要提供相应的URL，返回类型和参数。这里要分别为用户和库定义接口。

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

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

利用传进的参数作为path进行访问搜索，其中Observable为RxJava中的类型。可以理解为返回一个定义好的GitHub类型的数据，而repositories返回的是一个List，所以要定义为List。同样以传入的user作为参数path进行访问。

构造Retrofit对象

`OkHttp`

Retrofit是基于OkHttp封装的，所以可以自己配置响应的OkHttp对象。这里创建OkHttp对象并返回。

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

OkHttp 只负责发起网络请求，维护网络连接等操作，而 Retrofit 帮我们将网络传输的数据转换为可用的 model 对象，并且提供简单的数据处理方式。所以这部分按照实验文档写即可。

`Retrofit对象`

然后构造Retrofit对象实现网络访问。这里使用retrofit创建响应访问API接口，即可通过创建好的实例进行访问操作。

```
    private static Retrofit createRetrofit(String baseUrl) {
        return new Retrofit.Builder()
                .baseUrl(baseUrl)
                .addConverterFactory(GsonConverterFactory.create())
                .addCallAdapterFactory(RxJavaCallAdapterFactory.create())
                .client(createOkHttp())
                .build();
    }
```

最后通过get方法调用，实现访问。参数为对应 的URL。

```
    public static Retrofit getRetrofit(String baseUrl) {
        return createRetrofit(baseUrl);
    }
```

实现网络访问

定义好Retrofit的网络访问之后，在主界面进行输入及访问，主界面为搜索用户访问，点击item跳转到库列表。

主界面中的两个button，一个用于清空，一个用于点击获取输入框内容然后网络访问搜索；对于搜索出来的item，点击跳转到对应的库列表界面。所以两个界面主要实现监听器的事件处理。

清空只需要对输入框进行settext为空，然后同时将adpter清空，刷新列表。

```
        clear.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                searchInput.setText("");
                cardAdp.clear();
            }
        });
```

搜索时，首先获取输入框内容，并将progressbar置为可见，即加载图标旋转起来。然后调用定义的接口，创建service，并传入GitHub的API的URL，创建实例。

```
        fetch.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String input = searchInput.getText().toString();
                progressBar.setVisibility(View.VISIBLE);
                GithubService service = ServiceFactory
                        .getRetrofit("https://api.github.com")
                        .create(GithubService.class);
                service.getUser(input)
                        .subscribeOn(Schedulers.newThread())
                        .observeOn(AndroidSchedulers.mainThread())
                        .subscribe(new Subscriber<Github>() {
                            @Override
                            public void onCompleted() {
                                progressBar.setVisibility(View.INVISIBLE);
                            }

                            @Override
                            public void onError(Throwable e) {
                                Toast.makeText(MainActivity.this, e.getMessage() + "\n没有找到用
户", Toast.LENGTH_SHORT).show();
                                progressBar.setVisibility(View.INVISIBLE);
                            }

                            @Override
                            public void onNext(Github github) {
                                cardAdp.addItem(github);
                            }
                        });
            }
        });
```

然后通过retrofit创建好响应的访问实例后，调用相应的访问API的方法即可。这里要注意的是：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。所以在onCompleted中将progressbar置为不可见，即表示加载完成；若出现错误则表示搜索失败，没有该用户；在onNext中将搜索到的GitHub类型的数据add到cardview 的adpter中，用于列表显示。

然后是点击列表中的某一项时跳转到对应库列表界面，将login作为参数传过去即可。长按的话则调用removeitem方法直接删除，并弹出Toast信息。

```
        cardAdp.setOnItemClickListener(new CardAdapter.OnItemClickListener() {
            @Override
            public void onClick(int position) {
                Intent intent = new Intent(MainActivity.this, ReposActivity.class);
                intent.putExtra("login", cardAdp.getItem(position).getLogin());
                startActivity(intent);
            }

            @Override
            public void onLongClick(int position) {
                cardAdp.removeItem(position);
                Toast.makeText(MainActivity.this, "用户已删除", Toast.LENGTH_SHORT).show();
            }
        });
```

库列表界面更简单，直接将访问结果显示出来即可。访问方式和主界面一致。

区别在于由于repos返回的是一个list，所以在想listitem中添加时要利用循环一项一项地添加，调用定义好的方法。

```
        GithubService service = ServiceFactory
                .getRetrofit("https://api.github.com")
                .create(GithubService.class);
        service.getRepos(login)
                .subscribeOn(Schedulers.newThread())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(new Subscriber<List<Repos>>() {
                    @Override
                    public void onCompleted() {
                        progressBar.setVisibility(View.INVISIBLE);
                    }

                    @Override
                    public void onError(Throwable e) {
                        progressBar.setVisibility(View.INVISIBLE);
                        Toast.makeText(ReposActivity.this, e.getMessage() + "\n无法获得相关信息",
Toast.LENGTH_SHORT).show();
                    }

                    @Override
                    public void onNext(List<Repos> repos) {
                        for(int i = 0; i < repos.size(); i++){
                            Map<String, Object> tmp = new LinkedHashMap<>();
                            tmp.put("name", repos.get(i).getName());
                            tmp.put("language", repos.get(i).getLanguage());
                            tmp.put("detail", repos.get(i).getDescription());
                            listItem.add(tmp);
                        }
                        listAdp.notifyDataSetChanged();
                    }
                });
```

布局方面

布局方面在主界面使用recyclerview，按照第三次实验的方法自定义adpter和viewholder。具体定义方法不再赘述，参照第三次实验文档和代码即可。要注意的是在添加item后调用notifydatasetchanged方法，刷新列表，使其实时更新。

然后对于recyclerview的item使用cardview，其中定义需要用到的控件，如textview。需要注意添加依赖。

然后网络访问的加载过程通过progressbar表示，即一个旋转的圆圈。通过设置其visibility属性设置其是否可见，加载时可见，加载完成或不加载时不可见即可。

实验总结

实验过程中遇到的问题：

1. 网络访问的实现。因为之前没有做过网络访问的实验，所以对网络的访问不是很清楚。尤其是对各种方法，名词不理解，如retrofit, OkHttp等，感觉很迷。不过还好实验文档中的实现步骤很清晰，而且代码很全（最重要的）。所以按照实验文档来基本就可以实现了。做完之后才感觉对这个网络访问的过程有了一点了解。知道了其过程和需要的东西。
2. 布局的设置。本次实验再次使用recyclerview实现列表显示，但是由于要自定义适配器，所以以前的实验我基本都用listview。但是没办法只能按照实验三的方法进行自定义adpter和viewholder。然后item使用cardview，感觉也是一种布局方式，在里面添加要用到的控件即可， 如textview显示内容。可能是cardview更适合作为item显示。实现方法也比较简单。然后一点是整个app的调色，因为看到实验文档中的截图输入框的下边线也是绿的，但我的是蓝的，然后就找了找是在哪里定义的这个颜色。然后发现是在colors里，然后就改掉了一直感觉很丑了蓝色顶部栏。
3. API的访问。因为这次实验是访问GitHub，而且其API也给出来了，我们直接利用URL访问取回数据即可。但是如果想要访问其他网站的话不知道它们的API要去哪里找，比如微博或者ins。然后我搜了一下微博的API，不像github那么简单的给出来，有点麻烦，所以还没有尝试，有时间会试一下。

其实这次实验主要是依靠TA给的实验文档写的，如果什么都不给的话可能实现起来很费劲。不过做完之后感觉思路还是比较清晰的，只要熟悉了这些类的定义和方法的使用基本上可以实现的差不多。但是现在还不知道怎么找其他网站如微博或ins的API，怎么访问。看了一下微博的接口好像有点麻烦，以后有时间会试一下访问微博。