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	**2015**	专业（方向）	**互联网**
学号	**15352194**	姓名	**梁杰鑫**
电话	**15113959962**	Email	**Alcanderian@gmail.com**
开始日期	**2017.12.20**	完成日期	**2017.12.27**

一、 实验题目

利用 retrofit 和 rxjava 实现一个异步请求 Github API 并且展示结果的 api，要请求的 API 有 user API 和 repository API。

二、 实现内容

第一个页面有一个输入框和两个按钮，一个 list。

输入框输入用户名

第一个按钮按下之后，输入框内容被清空

第二个按钮触发 user API 的请求，参数为输入框的内容

List 装载已经请求的 user 信息，短按进入 repo 页面，长按删除

第二个页面只有一个 list

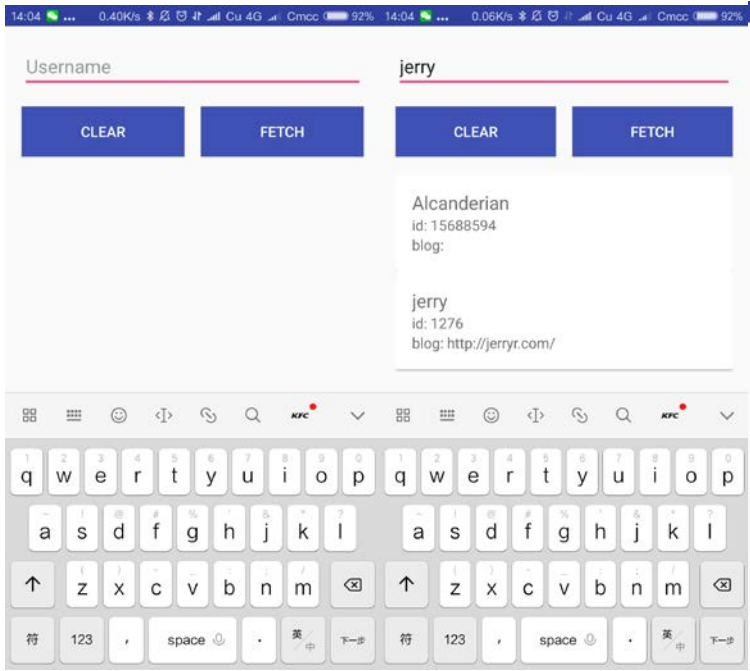
List 装载 repository API 的结果

对于 User Model, 显示 id, login, blog

对于 Repository Model, 显示 name, description,language

课堂实验结果

（1） 实验截图

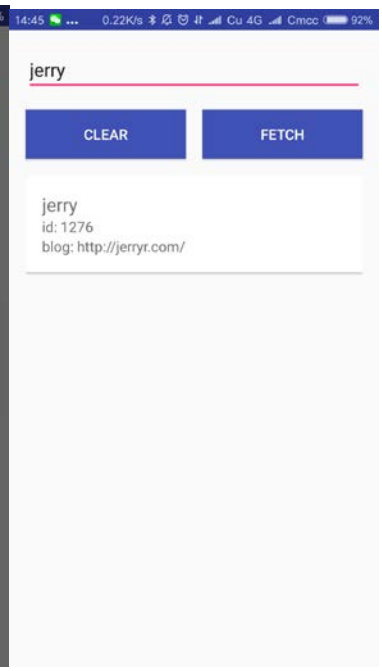


初始页面

fetch 了几个用户

alcanderian 的 repo

确认删除



删除结果

(2) 实验步骤以及关键代码

- 关于 RETROFIT 的运作模式和 RXJAVA 与 RETROFIT 的关系。

RETROFIT 是一个 http 请求的高级封装，基于 okhttp 封装的接口库。

运作模式：

根据用户自定义的 interface 以及 interface 中的提示（@GET,@POST 请求类型以及 @Path,@Param 参数类型）构造一个 observable 请求体。

通过 okhttp 等底层库进行 http 请求。

请求结束后调用 gson-converter 将返回的字符串转换为相应的 model。

Observer 收到转换之后的 model 进行处理。

- 实现

首先定义 model

```
public class Repo {
    public String name;
    public String language;
    public String description;
}

public class User {
    public String login;
    public Integer id;
    public String blog;
}
```

再定义 inferface

```
public interface GithubService {
    @GET("/users/{user}/repos")
    Observable<ArrayList<Repo>> getRepos(@Path("user") String user);

    @GET("/users/{user}")
    Observable<User> getUser(@Path("user") String user);
}
```

Retrofit 会根据@path 中的提醒用 user 参数替换 GET 参数中的{user}字段

利用工厂设计模式（不这不是工厂设计模式），构造一个工厂类。这个类用于生成 okhttp 实体和 retrofit 实体

```
public class ServiceFactory {
    public static final String URL = "https://api.github.com";

    public static OkHttpClient getOkHttpClient() {
        return new OkHttpClient.Builder()
                .connectTimeout(10, TimeUnit.SECONDS)
                .readTimeout(30, TimeUnit.SECONDS)
                .writeTimeout(10, TimeUnit.SECONDS)
                .build();
    }

    public static Retrofit getRetrofit(String baseUrl) {
        return new Retrofit.Builder()
                .baseUrl(baseUrl)
                .addConverterFactory(GsonConverterFactory.create())
                .addCallAdapterFactory(RxJavaCallAdapterFactory.create())
                .client(getOkHttpClient())
                .build();
    }
}
```

- User API

在调用 service 的时候，需要构造一个订阅者（观察者的高级封装），订阅者的定义如下

```
private void fetch(String username) {
    service.getUser(username)
            .subscribeOn(Schedulers.newThread())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(new Subscriber<User>() {
                @Override
                public void onStart() {
                    super.onStart();
                    btnFetch.setEnabled(false);
                    pb.setVisibility(View.VISIBLE);
                }

                @Override
                public void onCompleted() {
                    btnFetch.setEnabled(true);
                    pb.setVisibility(View.GONE);
                }
```

在 onStart 和 onComplete 中调用 progressBar 的 setVisibilty 显示或者隐藏进度条

```
                @Override
                public void onError(Throwable e) {
                    e.printStackTrace();
                    Toast.makeText(MainActivity.this, "用户不存在", Toast.LENGTH_SHORT).show();
                    btnFetch.setEnabled(true);
                    pb.setVisibility(View.GONE);
                }

                @Override
                public void onNext(final User user) {
                    caUserInfo.data.add(new HashMap<String, Object>() {{
                        put("login", user.login);
                        put("id", "id: " + user.id.toString());
                        put("blog", "blog: " + user.blog);
                    }});
                    caUserInfo.notifyDataSetChanged();
                }
            });
}
```

请求错误的时候隐藏进度条并 toast

请求成功之后将数据放入 list 中并更新 recycleView

- Repo API 的获取

RepoAPI 的请求方法和 userAPI 大同小异，主要区别在与获取到数据后的处理，返回的结果是一个

```
        @Override
        public void onNext(final ArrayList<Repo> list) {
            for (final Repo item : list) {
                repos.add(new HashMap<String, Object>() {{
                    put("name", item.name);
                    put("language", item.language == null ? "unknown" : item.language);
                    put("description", item.description == null ? "null" : item.description);
                }});
            }
            caRepos.notifyDataSetChanged();
        }
```

- RecyclerView

RecyclerView 在 shoplist 实验中使用过，在那一次实验中我们构造的是一个通用的 recyclerViewAdapter，所以我们只需要重写其中的 convertData 函数就好

定义

```
ublic abstract class CommonAdapter extends RecyclerView.Adapter<CommonAdapter.ViewHolder> {
    public ArrayList<HashMap<String, Object>> data;
    public Context context;
    public Integer layoutId;
    public OnItemClickListener onItemClickListener;
```

重写 convertData 来匹配数据

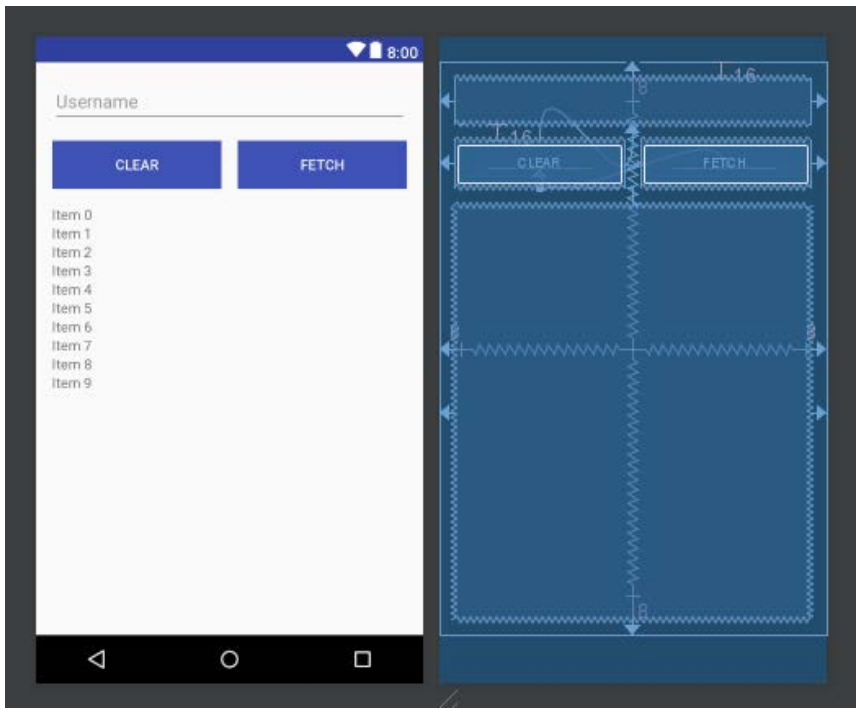
```
caUserInfo = new CommonAdapter(
        MainActivity.this,
        R.layout.user_card,
        new ArrayList<HashMap<String, Object>>()) {
    @Override
    public void convertData(ViewHolder holder, HashMap<String, Object> map) {
        ((TextView) holder.getView(R.id.txt_username)).setText((String) map.get("login"));
        ((TextView) holder.getView(R.id.txt_id)).setText((String) map.get("id"));
        ((TextView) holder.getView(R.id.txt_blog)).setText((String) map.get("blog"));
    }
};
```

在 onBindViewholder 中调用 convertData

```
@Override
public void onBindViewHolder(final ViewHolder holder, int position) {
    convertData(holder, data.get(position));
    if (onItemClickListener != null) {
        holder.view.setOnClickListener((v) -> {
                onItemClickListener.onItemClick(v, holder.getAdapterPosition());
        });
        holder.view.setOnLongClickListener((v) -> {
                return onItemClickListener.onItemLongClick(v, holder.getAdapterPosition());
        });
    }
}
```

- 布局

主界面

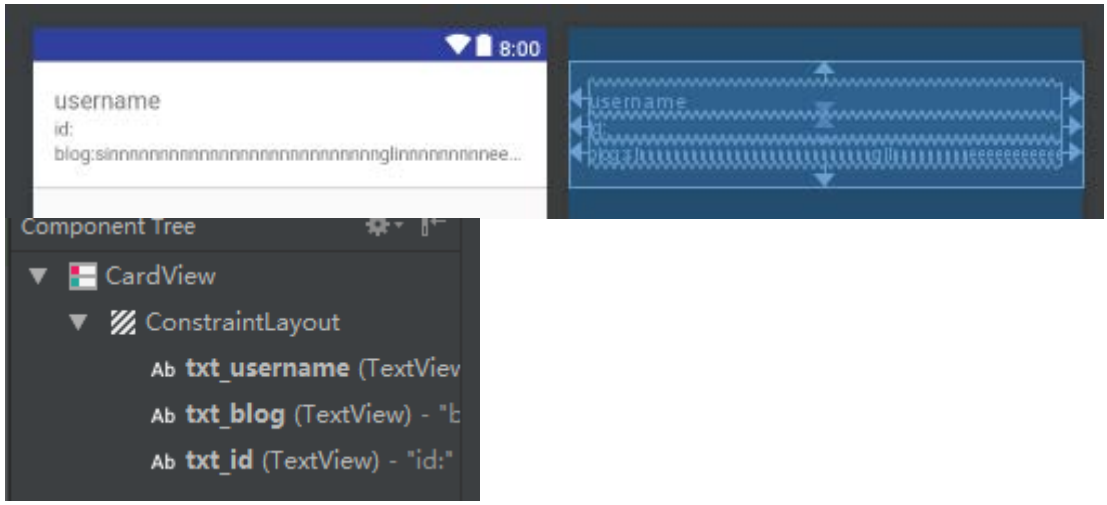


Repo 界面就是一个 recyclerView，这里不展示

repoItem 的布局

箭头处是 imageView

UserItem 的布局



下划线其实是 cardview 的阴影

三、 实验思考

这次实验中我们使用了高级封装的异步网络请求库，以及 gson。可以深刻地体会到 JAX-RS 标准以及 GSON 等解析库对于安卓 web 开发的帮助是巨大的，几乎彻底掩盖了 http 请求的底层逻辑。RESTFUL 风格服务器也是当前很流行的 web 服务器构造方式，我们的 project 决定采用这种方法构建项目。