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：

年级	15352272	专业（方向）	软件工程（移动信息工程）
学号	15352272	姓名	彭国栋
电话	15626066058	Email	578291308@qq.com
开始日期		完成日期	

一、 实验题目

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

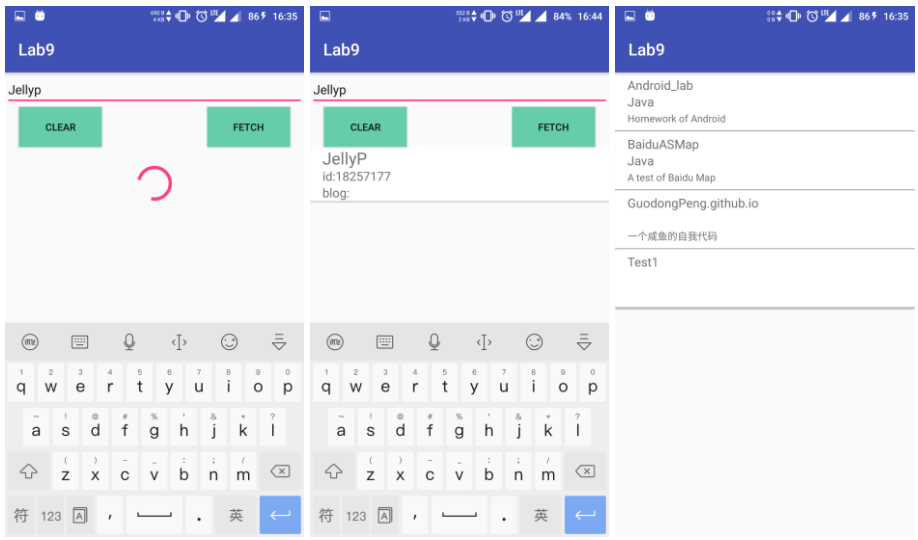
二、 实现内容

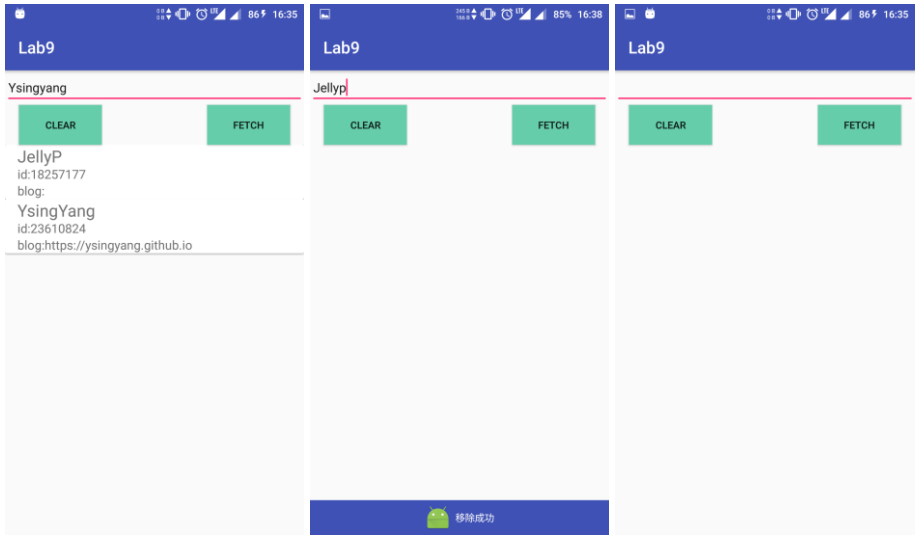
能够实现对 GitHub 上面的用户进行抓取，然后读取用户信息，点击用户后得到其仓库的内容。

三、 课堂实验结果

（1） 实验截图

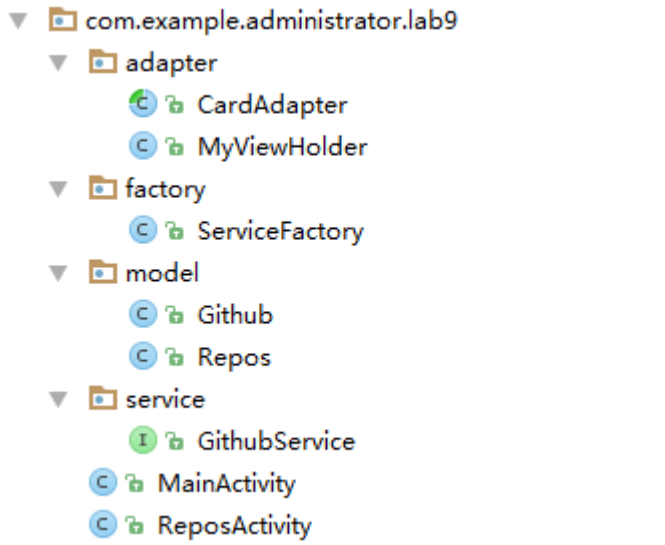
下面分别是：fetch，fetch 成功，点击进入列表，多个用户列表，删除列表的用户，clear 的操作。





（2） 实验步骤以及关键代码

这次实验所涉及到的函数主要有以下几个：



为了要能访问网络，首先需要赋予权限：

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

接下来是各个类和接口的介绍：

1、CardAdapter 和 MyViewHolder，采用了之前 lab3 的方式来实现：

·MyViewHolder

```
public class MyViewHolder extends RecyclerView.ViewHolder {
    private SparseArray<View> mViews;
    private View mListView;
    public MyViewHolder(Context context, View itemView, ViewGroup parent)
    {//默认构造函数

        super(itemView);
        mListView=itemView;
        mViews=new SparseArray<View>();
    }

    public static MyViewHolder get(Context context,ViewGroup parent,int layoutid)
    {
        View itemView= LayoutInflater.from(context).inflate(layoutid,parent
                ,false);//inflate可以将一个xml中定义的布局控件找出来
        MyViewHolder holder=new MyViewHolder(context,itemView,parent);
        return holder;
    }
    public <T extends View> T getView(int viewId)
    {
        View view = mViews.get(viewId);
        if(view==null)
        {
            view=mListView.findViewById(viewId);
            mViews.put(viewId,view);
        }
        return (T) view;
    }
```

·CardAdapter

```
public abstract class CardAdapter<T> extends RecyclerView.Adapter<MyViewHolder> {

    protected Context mContext;
    protected int mLayoutId;
    protected List<T> mDatas;
    private OnItemClickListener mOnItemClickListener=null;
    public CardAdapter(Context context,int layoutId,List<T> datas)
    {//构造函数
        mContext=context;
        mLayoutId=layoutId;
        mDatas=datas;
    }
    @Override
    public MyViewHolder onCreateViewHolder(final ViewGroup parent, int viewType)
    {//创建ViewHolder
        MyViewHolder viewHolder=MyViewHolder.get(mContext,parent,mLayoutId);
        return viewHolder;
    }
@Override
public void onBindViewHolder(final MyViewHolder holder,int position)
{//这个方法主要用于适配渲染数据到View中。方法提供给你了一个viewHolder，而不是原来的convertView。
    convert(holder,mDatas.get(position));
    if(mOnItemClickListener!=null)
    {
        holder.itemView.setOnClickListener((v) -> {
                mOnItemClickListener.onClick(holder.getAdapterPosition());
        });
        holder.itemView.setOnLongClickListener((v) -> {
                mOnItemClickListener.onLongClick(holder.getAdapterPosition());
                return false;
        });
    }
}
```

```
public abstract void convert(MyViewHolder viewHolder,T data);//需要重写的abstract函数convert
@Override
public int getItemCount() { return mDatas.size(); }
public void removeItem(int position)//删除该位置的数据
{
    mDatas.remove(position);
    notifyItemRemoved(position);
}

public interface OnItemClickListener
{
    void onClick(int position);
    void onLongClick(int position);
}

public void setOnItemClickListener(OnItemClickListener onItemClickListener)
{
    this.mOnItemClickListener=onItemClickListener;
}
```

2、ServiceFactory 是定义了一个封装了 OkHttp 和 Retrofit 的类，获取 api 的信息。

```
public class ServiceFactory {
    public static OkHttpClient createOkHttp()
    {
        OkHttpClient okHttpClient=new OkHttpClient.Builder().connectTimeout(10, TimeUnit.SECONDS)
                                .readTimeout(30,TimeUnit.SECONDS)
                                .writeTimeout(10,TimeUnit.SECONDS)
                                .build();
        return okHttpClient;
    }
    public static Retrofit createRetrofit(String baseUrl)
    {
        return new Retrofit.Builder()
                .baseUrl(baseUrl)
                .addConverterFactory(GsonConverterFactory.create())//添加Gson
                .addCallAdapterFactory(RxJavaCallAdapterFactory.create())//异步调用
                .client(createOkHttp())
                .build();
    }

}
```

3、Github 类和 Repos 类较简单，如下所示

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

    public String getLogin() { return login; }

    public String getBlog() { return blog; }

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

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

    public String getDescription() {
        return description;
    }

    public String getLanguage() { return language; }

    public String getName() { return name; }
}
```

4、接口 GithubService，用于获得用户信息，使用 get 方法。

```
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);
}
```

5、MainActivity，主要由 4 个函数组成：

（1）、初始化函数 init()

```
/**
 * 初始化
 */
public void init()
{
    search_edit=(EditText)findViewById(R.id.search_user);
    clear_button=(Button)findViewById(R.id.clear_button);
    recyclerView=(RecyclerView)findViewById(R.id.recycler_view);
    progressBar=(ProgressBar)findViewById(R.id.activity_progressbar);
    fetch_button=(Button)findViewById(R.id.fetch_button);
    githubList=new ArrayList<>();
}
```

（2）、设置 RecyclerView 的显示的函数 RecyclerView()：

```
recyclerView.setLayoutManager(new LinearLayoutManager(this));
githubCardAdapter=new CardAdapter<Github>(this,R.layout.user_item,githubList) {
    @Override
    public void convert(MyViewHolder viewHolder, Github data) {
        TextView login=viewHolder.getView(R.id.item_login);
        TextView id=viewHolder.getView(R.id.item_id);
        TextView blog=viewHolder.getView(R.id.item_blog);
        login.setText(data.getLogin());
        id.setText("id:"+data.getId());
        blog.setText("blog:"+data.getBlog());
    }
};
githubCardAdapter.setOnItemClickListener(new CardAdapter.OnItemClickListener() {
    @Override
    public void onClick(int position) {
        Intent intent=new Intent(MainActivity.this,ReposActivity.class);
        Bundle bundle=new Bundle();
        bundle.putString("login",githubList.get(position).getLogin());
        intent.putExtras(bundle);
        startActivity(intent);
        //startActivityForResult(intent,MAIN2REPOS);
    }
    @Override
    public void onLongClick(int position) {
        githubCardAdapter.removeItem(position);
        githubCardAdapter.notifyDataSetChanged();
        Toast.makeText(getApplicationContext(),"移除成功",Toast.LENGTH_SHORT).show();
    }
});
recyclerView.setAdapter(githubCardAdapter);
```

（3）设置 fetch 按钮的点击事件，setFetch_button()函数，用于绑定 fetch 按钮的事件，点击后用来获得 api 网站上面的输入框内用户的信息。

```
public void setFetch_button()
{
    fetch_button.setOnClickListener((v) → {
            Retrofit retrofit = ServiceFactory.createRetrofit("https://api.github.com");
            GithubService service = retrofit.create(GithubService.class);
            String User = search_edit.getText().toString();
            recyclerView.setVisibility(View.INVISIBLE);
            progressBar.setVisibility(View.VISIBLE);
            service.getUser(User)
                    .subscribeOn(Schedulers.newThread())
                    .observeOn(AndroidSchedulers.mainThread())
                    .subscribe(new Subscriber<Github>() {
                        @Override
                        public void onCompleted() {
                            progressBar.setVisibility(View.INVISIBLE);
                            recyclerView.setVisibility(View.VISIBLE);

                        }
                        @Override
                        public void onError(Throwable e) {
                            Toast.makeText(getApplicationContext(),"fetch wrong",Toast.LENGTH_SHORT).show();
                            progressBar.setVisibility(View.INVISIBLE);
                            recyclerView.setVisibility(View.VISIBLE);

                        }
                        @Override
                        public void onNext(Github github) {
                            githubList.add(github);
                            githubCardAdapter.notifyDataSetChanged();
                        }
                    });
```

（4）设置 clear 按钮的点击事件，setClear_button()函数，用来点击后清除输入框和 recyclerView 的用户列表。

```
/**
 * 清除输入框和用户列表
 */
public void setClear_button()
{
    clear_button.setOnClickListener((v) → {
            search_edit.setText("");
            githubList.clear();
            githubCardAdapter.notifyDataSetChanged();
            Toast.makeText(getApplicationContext(),"清除成功",Toast.LENGTH_SHORT).show();

    });
}
```

6、ReposActivity 类，和 MainActivity 差不多，只是不需要进行点击事件而是直接进行获取数据并在 RecyclerView 中显示。

（1）初始化函数 init()

```
/**
 * 初始化
 */
public void init()
{
    reposList=new ArrayList<>();
    progressBar=(ProgressBar)findViewById(R.id.repos_progressbar);
    recyclerView=(RecyclerView)findViewById(R.id.repos_recycler_view);

}
```

（2）用于读取某个用户的仓库数据的函数，getData()，由于前面接口什么的都比较完善，所以只需要直接简单调用即可，使用方法类似前面的获取用户数据一样。只是这里的用户可以有多个仓库，所以返回的是个 List。

```
public void getData()
{
    progressBar.setVisibility(View.VISIBLE);
    Retrofit retrofit= ServiceFactory.createRetrofit("https://api.github.com");
    GithubService service=retrofit.create(GithubService.class);
    String User=getIntent().getExtras().get("login").toString();
    service.getRepos(User)
            .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) {
                    Toast.makeText(getApplicationContext(),"get_error",Toast.LENGTH_SHORT).show();
                }

                @Override
                public void onNext(List<Repos> reposes) {
                    for(Repos item:reposes)
                    {
                        reposList.add(item);

                    }
                    reposCardAdapter.notifyDataSetChanged();
                }

            }
);
```

（3）用于设置 RecyclerView 的显示的函数 RecyclerView()，这里不需要重写点击事件，因为并不需要设置。

```
public void RecyclerView()
{
    recyclerView.setLayoutManager(new LinearLayoutManager(ReposActivity.this));
    reposCardAdapter=new CardAdapter<Repos>(this,R.layout.repos_item,reposList) {
        @Override
        public void convert(MyViewHolder viewHolder, Repos data) {
            TextView name=viewHolder.getView(R.id.item_name);
            TextView language=viewHolder.getView(R.id.item_language);
            TextView description=viewHolder.getView(R.id.item_description);
            name.setText(data.getName());
            language.setText(data.getLanguage());
            description.setText(data.getDescription());
        }
    };
    recyclerView.setAdapter(reposCardAdapter);
}
```

（3） 实验遇到困难以及解决思路

在使用 retrofit 的时候，一开始调用 subscribe 函数出错，检查了一下和咨询了同学之后，发现是依赖加错了。重写修改依赖即可。

在查找用户查找了第一个用户之后发现再查找用户的话不会显示 progressbar，原因是查找了第一个用户用户显示的位置挡住了 progressbar 的显示，通过在函数中设置当显示 progressbar 的时候不显示 recyclerView 即可。

四、 实验思考及感想

在实验过程中，还是很多思考的，首先就是为什么要使用 cardview 而不用和以前一样的布局来实现呢，而且在实现的过程中并没有感受到明显的区别。后面通过查阅资料的方式，发现 Cardview 的特殊点就是有 rounded corner（圆角）和 shadow（阴影），这个就是它的特殊之处，通过这个布局能够直接来设置背景的圆角而不用自己写 shape。其次就是网络的使用了，在之前的 lab 里面基本都是本地的操作，所有的数据也是放在本地，而这次实验则通过使用 GitHub 上面的 api 来实现了简单的网络操作，感觉对于大作业还是很有帮助的。