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）

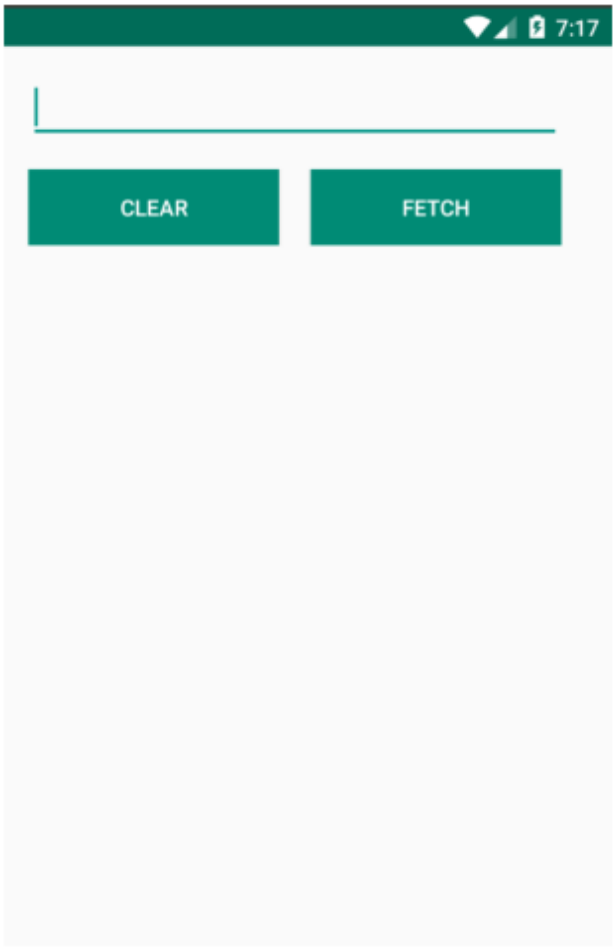
课程名称：移动应用开发　　任课教师：

年级	15 级	专业（方向）	软件工程（移动信息工程）
学号	15352223	姓名	刘朝开
电话	15626489095	Email	1209354383@qq.com
开始日期	2017.9.30	完成日期	2017.10.1

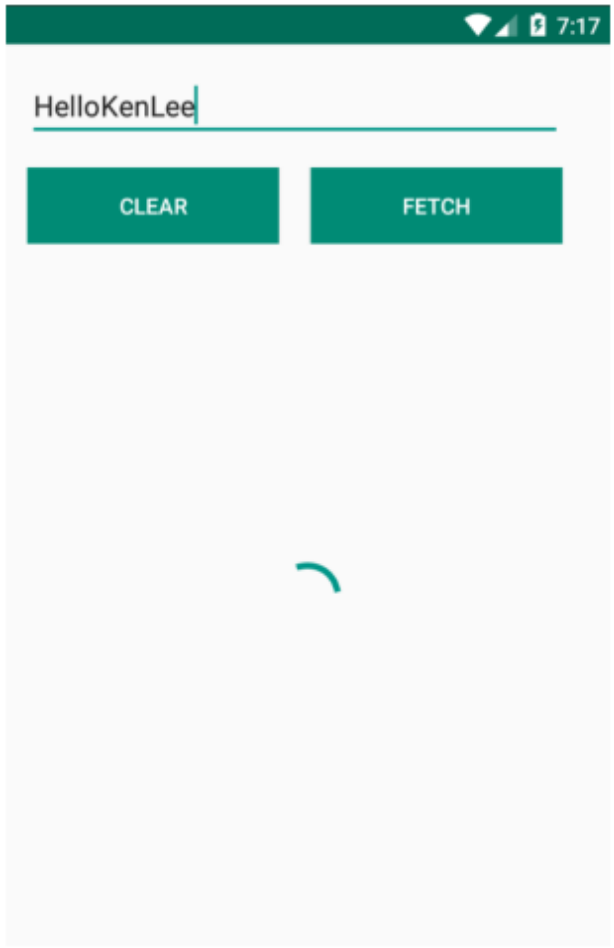
一、 实验题目

Retrofit+RxJava+OkHttp 实现 网络请求

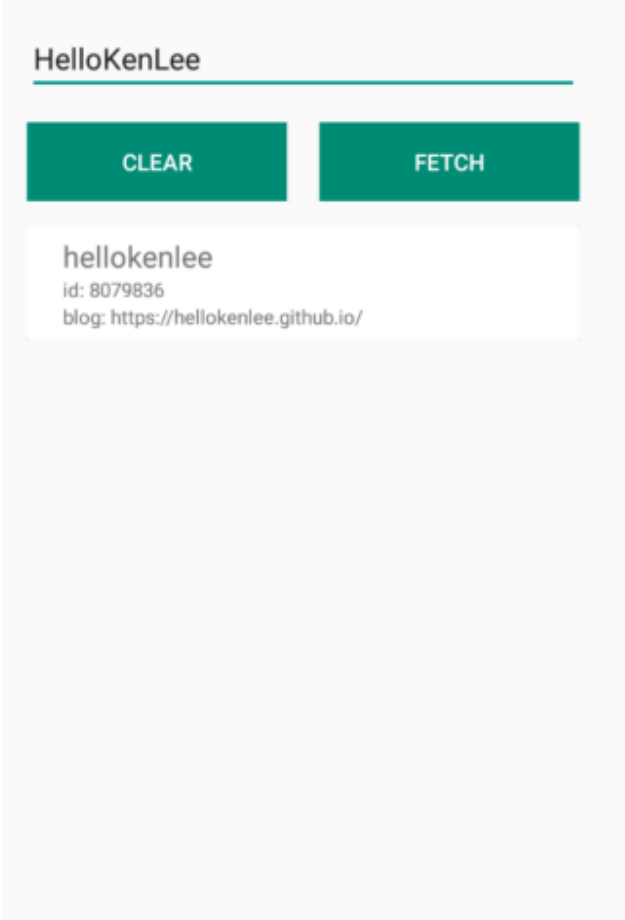
二、 实现内容



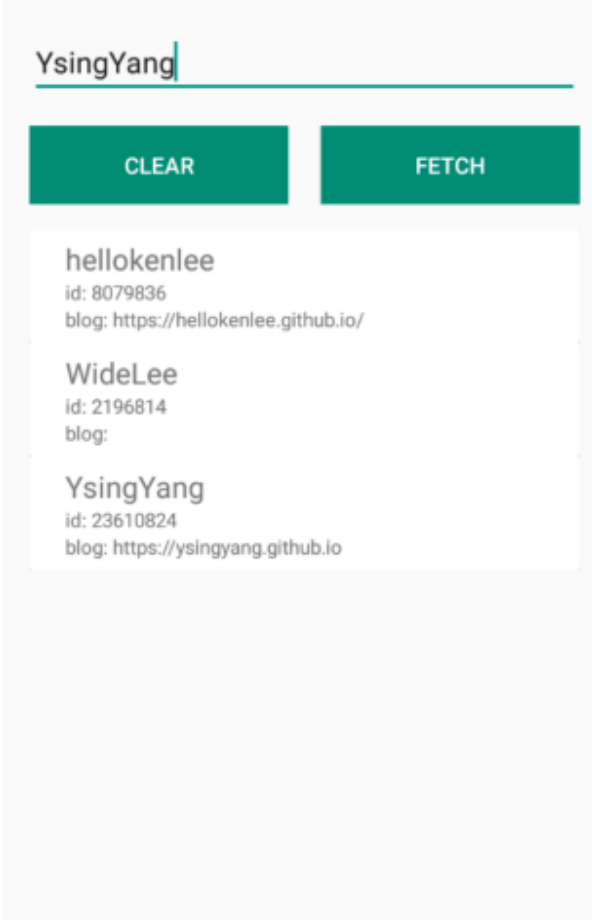
主界面



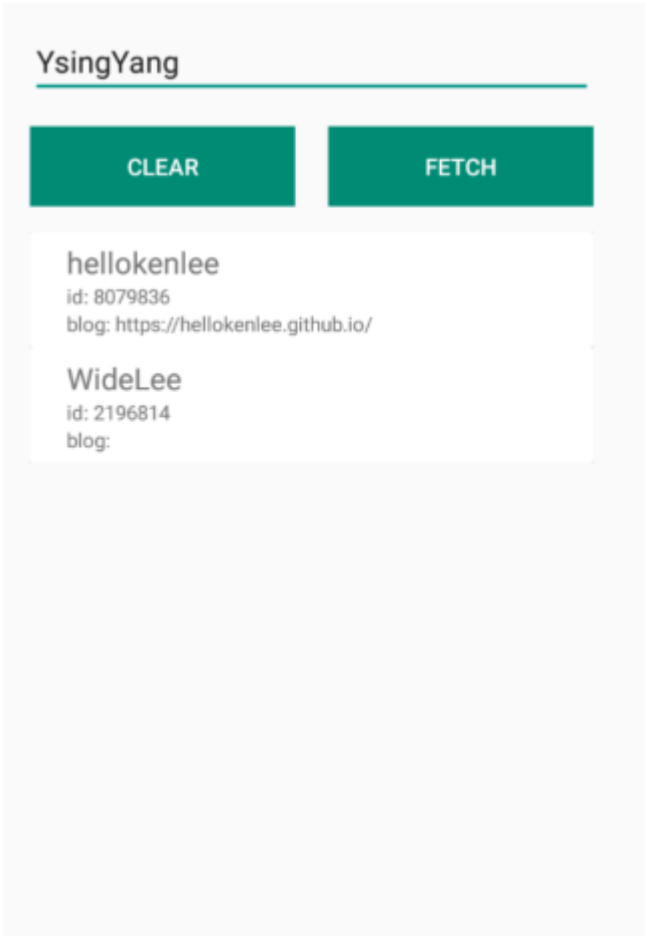
搜索用户


HelloKenLee
CLEAR
FETCH
hellokenlee
id: 8079836
blog: https://hellokenlee.github.io/

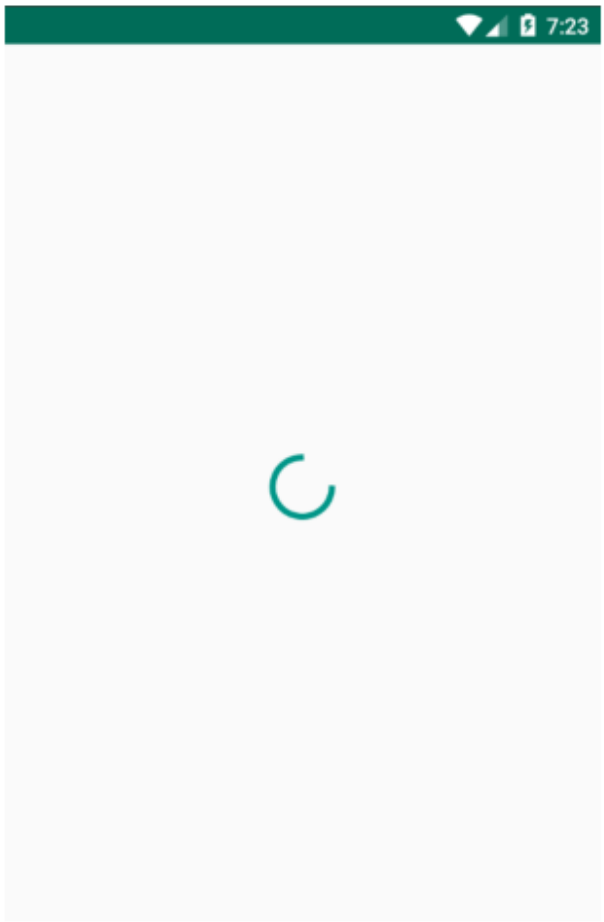
搜索结果


YsingYang
CLEAR
FETCH
hellokenlee
id: 8079836
blog: https://hellokenlee.github.io/
WideLee
id: 2196814
blog:
YsingYang
id: 23610824
blog: https://ysingyang.github.io

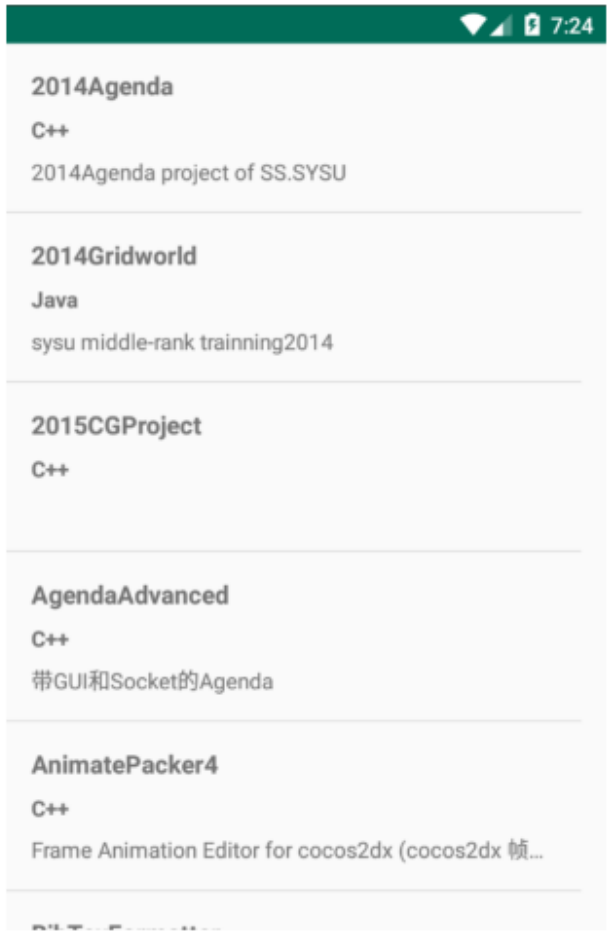
搜索结果


YsingYang
CLEAR
FETCH
hellokenlee
id: 8079836
blog: https://hellokenlee.github.io/
WideLee
id: 2196814
blog:

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

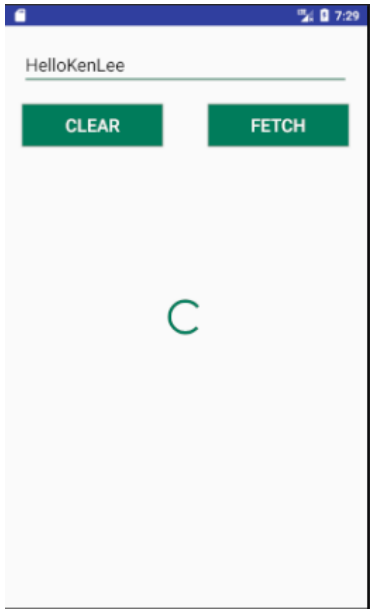
对于 Repository Model, 显示 name, description,language

(特别注意， 如果 description 对于 1 行要用省略号代替)

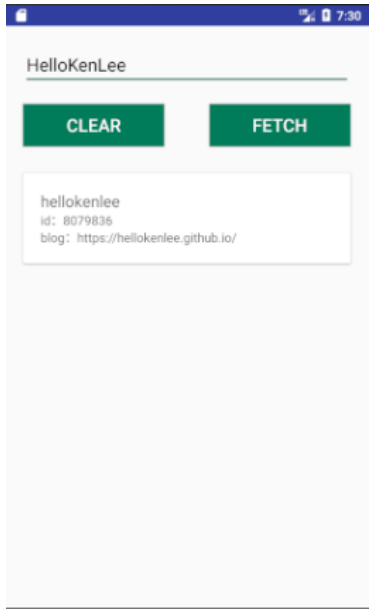
三、 课堂实验结果

（1） 实验截图

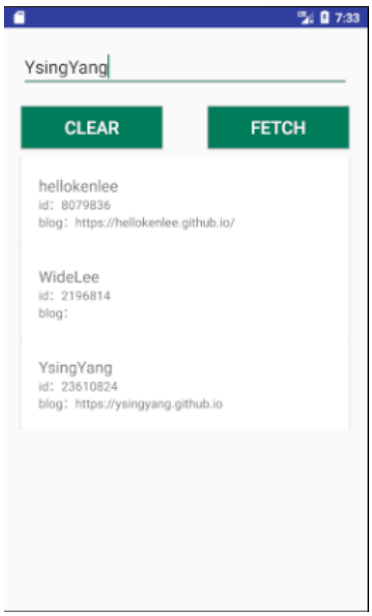
主界面



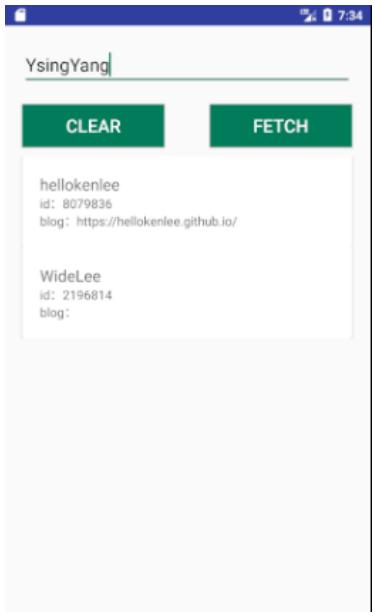
搜索用户



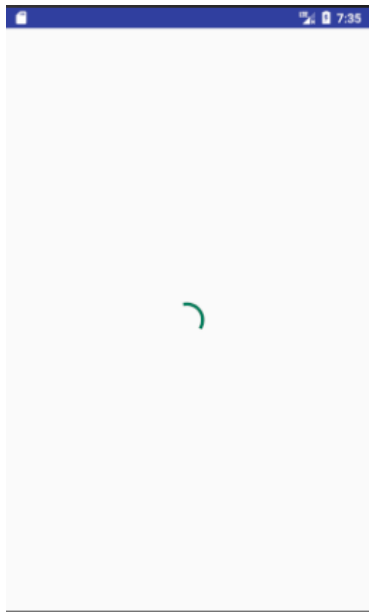
搜索结果



搜索结果



长按删除



点击进入个人详情页面

详情信息获取显示

（2） 实验步骤以及关键代码

步骤 1：添加依赖，设置布局。

本次实验涉及到的库比较多，包括 RecyclerView,CardView, Retrofit 和 OkHttp, RxJava 的一些依赖。截图如下：

```
compile 'com.android.support:recyclerview-v7:26.1.0'
compile 'com.android.support:cardview-v7:26.1.0'
implementation 'io.reactivex:rxjava:1.0+'
implementation 'io.reactivex:rxandroid:0.23+'
implementation 'com.squareup.retrofit2:retrofit:2.1.0'
implementation 'com.squareup.retrofit2:adapter-rxjava:2.1.0'
implementation 'com.squareup.retrofit2:converter-gson:2.1.0'
implementation 'com.android.support:design:26.1.0'
```

主要需要设计的有四个布局：activity_main.xml,设计主界面的布局；activity_repos.xml 设计详情信息界面布局。useritem.xml 设计显示用户 id, login, blog 的条目，repositoryitem.xml 显示用户的 name, description,language 的条目。

其中，用户 id, login, blog 的条目在 activity_main.xml 用 recyclerview 实现，在 useritem.xml 中用 cardview 实现包裹起来，ProgressBar 设置为 gone，当搜索时才出现，搜索结束后再设置为 gone。name, description,language 的条目在 activity_repos.xml 用 listview 实现。

在 color.xml 中修改 colorAccent 使得 EditText 和 ProgressBar 显示为绿色。

```
<color name="colorAccent">#007C5B</color>
```

在 style.xml 中修改 parent, 去除标题栏。

```
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
```

当 repo 的描述超过一行时，需要显示末尾为省略号，在对应 TextView 中配置如下属性：

```
android:singleLine="true"
android:ellipsize="end"/>
```

步骤 2：定义 Model 类

Model 类用于封装所请求的网络数据。本实验需要请求两种不同的网络数据，分别为 github 用户的个人信息和该用户的仓库信息。因此定义了两个类如下：

Github 类：用于保存用户的个人信息，包括登陆名，博客地址和 id。并设置相应的 get 函数。

```
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;}
}
```

Repos 类：用于保存用户的个人信息，包括名字，语言，描述。并设置相应的 get 函数。

```
public class Repos {
    private String name;
    private String language;
    private String description;
    public String getName() { return name; }
    public String getLanguage() { return language; }
    public String getDescription() { return description; }
}
```

步骤 3：实现接口函数

给 retrofit 对象提供相应的接口，指定提交方法(get)，数据对应的 URL，函数传入的参数和返回类型。

```
public interface GithubService {
    @GET("/users/{user}")
    Observable<Github> getUser(@Path("user") String user);
    //返回一个 Github 类型

    @GET("/users/{user}/repos")
    Observable<List<Repos>> getRepos(@Path("user") String user);
    //返回一个 List<Repos> 类型
}
```

步骤 4：实现工厂类。

工厂类通过设置好需要使用到的“产品”，本实验为 Retrofit 对象，可以方便的批量生产，即每次需要新建对象时可直接调用里面的函数生成配置好的对象，不必每次都执行相同的代码进行新建对象，省去了配置的过程，从而简化了代码。

在以下代码中，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。

```
public class ServiceFactory {
    //静态成员不属于对象，而属于类。不创建对象也可调用。
    private static OkHttpClient createOkHttp()//创建一个 OkHttpClent 并进行简单配置
    {
        return new OkHttpClient.Builder()
                .connectTimeout( timeout: 10, TimeUnit.SECONDS)//连接超时10s
                .readTimeout( timeout: 30,TimeUnit.SECONDS)//读超时30s
                .writeTimeout( timeout: 10, TimeUnit.SECONDS)//写超时10s
                .build();
    }

    public static Retrofit createRetrofit(String baseUrl)
    {
        return new Retrofit.Builder()
                .baseUrl(baseUrl)//设置baseUrl
                .addConverterFactory(GsonConverterFactory.create())//添加GSONConverter,数据转换 adapter
                .addCallAdapterFactory(RxJavaCallAdapterFactory.create())// rxJava 调用 adapter
                .client(createOkHttp())// client
                .build();
    }
}
```

步骤 5：设置适配器。

类构造函数：传入上下文 context，Github 对象的数组。

静态函数 ViewHolder 继承于 RecyclerView.ViewHolder,用于对控件的实例进行缓存，可以提高运行效率。

```
public CardAdapter (Context context, List<Github> githubList)
{
    this.context = context;
    this.githubList = githubList;
    inflater = LayoutInflater.from(context);
    mItemClickListener = null;
}
//定义内部类继承于RecyclerView.ViewHolder
static class ViewHolder extends RecyclerView.ViewHolder
{
    TextView Login;
    TextView Id;
    TextView Blog;
    private ViewHolder(View view)
    {
        super(view);
        Login = (TextView) view.findViewById(R.id.login);
        Id = (TextView) view.findViewById(R.id.id);
        Blog = (TextView) view.findViewById(R.id.blog);
    }
}
```

三个必须重载的函数：

```
@Override//填充onCreateViewHolder方法返回的holder中的控件
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View view = inflater.inflate(R.layout.useritem, parent, attachToRoot: false);
    return new ViewHolder(view);
}
@Override
public void onBindViewHolder(final CardAdapter.ViewHolder holder, final int position) {
    final Github github = githubList.get(position);
    holder.Login.setText(github.getLogin());
    String id = "id: "+github.getId();
    holder.Id.setText(id);
    String blog = "blog: "+github.getBlog();
    holder.Blog.setText(blog);
    //item的点击事件
    if(mItemClickListener != null) {
        holder.itemView.setOnClickListener((view) -> {
                int position = holder.getLayoutPosition();
                mItemClickListener.onItemClick(position);
        });
        holder.itemView.setOnLongClickListener((view) -> {
                int position = holder.getLayoutPosition();
                mItemClickListener.onItemLongClick(position);
                return true;
        });
    }
}
@Override//返回数组的大小
public int getItemCount() { return githubList.size(); }
```

设置 item 点击事件的接口类和监听方法。（在 onBindViewHolder 中实现了对应的点击事件）

```
//定义item的回调接口类
public interface OnItemClickListener
{
    void onItemClick(int position);
    void onItemLongClick(int position);
}
//定义一个设置点击监听的方法
public void setOnItemClickListener(OnItemClickListener onItemClickListener){
    this.mItemClickListener = onItemClickListener;//设置监听器
}
```

步骤 6：MainActivity 的逻辑实现。

1．CLEAR 按钮的点击事件。将数组清空，并更新 recyclerview 列表。

```
case R.id.clear:
    githubList.clear();
    cardAdapter.notifyDataSetChanged();
    break;
```

2．FETCH 点击事件。通过 EditText 获得对应的用户，使用 Retrofit 对象进行网络请求，将得到的数据存入 githubList 数组，并更新 recyclerview 列表。

```
case R.id.fetch:
    progressBar.setVisibility(View.VISIBLE);
    String Name = Search.getText().toString();
    ServiceFactory.createRetrofit( baseUrl: "https://api.github.com/")
            .create(GithubService.class)
            .getUser(Name)
            .observeOn(AndroidSchedulers.mainThread())
            .subscribeOn(Schedulers.newThread())
            .subscribe(new Subscriber<Github>() {
                @Override//请求结束时调用的回调函数
                public void onCompleted() { progressBar.setVisibility(View.GONE); }

                @Override//请求出现错误时调用的函数
                public void onError(Throwable e) {
                    Toast.makeText( context: MainActivity.this, text: "用户不存在", Toast.LENGTH_SHORT).show();
                    progressBar.setVisibility(View.GONE);
                    e.printStackTrace();
                }

                @Override//收到每一次数据时调用的函数
                public void onNext(Github github) {
                    githubList.add(github);
                    cardAdapter.notifyDataSetChanged();
                }
            });
        break;
```

3.Item 的点击事件：点击跳转到 ReposActivity 并传入登陆名。长按删除对应的 item，并更新 recyclerview 列表。

```
cardAdapter.setOnItemClickListener(new CardAdapter.OnItemClickListener() {
    @Override
    public void onItemClick(int position) {
        Intent intent = new Intent( packageContext: MainActivity.this, ReposActivity.class);
        intent.putExtra( name: "login",githubList.get(position).getLogin());
        startActivity(intent);
    }

    @Override
    public void onItemLongClick(int position) {
        githubList.remove(position);
        cardAdapter.notifyDataSetChanged();
    }
});
```

步骤 7：ReposActivity 的逻辑实现。

通过传入的登陆名，使用 Retrofit 请求对应用户的 repos 列表数据。并将其传送到 initListData 函数中，对适配器进行装载。

```
ServiceFactory.createRetrofit( baseUrl: "https://api.github.com/")//指定baseUrl,(由于是静态函数，可以用类直接调用)
        .create(GithubService.class)//“使用”我们指定的 Interface,到该URL的指定的“下级目录”去请求数据。
        .getRepos(Name)//指定下级目录的接口为"/users/"+Name+"/repos"
        .observeOn(AndroidSchedulers.mainThread())//observeOn()用于指定的是接收事件的线程，此处为主线程，用于操作UI
        .subscribeOn(Schedulers.newThread())//subscribeOn()用于指定发送事件的线程,此处为子线程，用于做耗时操作
        .subscribe(new Subscriber<List<Repos>>()//指定了接收的对象
        {
            @Override
            public void onCompleted() {
                progressBar.setVisibility(View.GONE);
                listView.setVisibility(View.VISIBLE);
            }

            @Override
            public void onError(Throwable e) {
                progressBar.setVisibility(View.GONE);
                listView.setVisibility(View.VISIBLE);
                Toast.makeText( context: ReposActivity.this, text: "出错!", Toast.LENGTH_SHORT).show();
            }

            @Override
            public void onNext(List<Repos> reposList) {//在这里接收数据
                initListData(reposList);
            }
        });
```

其中，initListData 定义如下：通过设置适配器从而将用户仓库信息显示出来。

```
void initListData(List<Repos> reposList)
{
    List<Map<String, String>> repoListData = new ArrayList<>();
    for (Repos repo : reposList) {
        Map<String, String> temp = new LinkedHashMap<>();
        temp.put( k: "name", repo.getName());
        temp.put( k: "language", repo.getLanguage());
        temp.put( k: "intro", repo.getDescription());
        repoListData.add(temp);
    }
    SimpleAdapter simpleAdapter = new SimpleAdapter( context: this,repoListData,
            R.layout.repositoryitem,
            new String[]{"name", "language", "intro"},
            new int[]{R.id.rname, R.id.language, R.id.intro});
    listView.setAdapter(simpleAdapter);
}
```

（3） 实验遇到困难以及解决思路

困难：

1 对工厂类的理解出现错误。

2 不会使用 RxJava。

3 SimpleAdapter 的使用。

解决思路：

1 上网查找相应的例子。工厂类顾名思义，就是用来生产产品的。通过设置好需要使用到的“产品”，本实验为 Retrofit 对象，可以方便的批量生产，即每次需要新建对象时可直接调用里面的函数生成配置好的对象，不必每次都执行相同的代码进行新建对象，省去了配置的过程，从而简化了代码。

2RxJava 的基本工作原理：Observable 对应着被观察者，Observer 对应着观察者，subscribe()对应着它们之间的连接。Observable 通过发射器 ObservableEmitter 发出不同的事

件，主要分为三类，next 事件、complete 事件和 error 事件。在 subscribe 中通过重载 new Subscriber 内部的三个函数 onCompleted()，onError(Throwable e)和 onNext()来实现，其中 onComplete()函数中接收传输结束事件，即收到该事件后不再接收 Next 事件（除非再次发出请求）。onError(Throwable e)函数在请求出现错误时调用，收到该事件后不再接收 Next 事件。onNext()函数用于接收请求的数据，本实验中为 Github 对象和 List<Repos>对象。

3 SimpleAdapter 的参数说明：

第一个参数表示访问整个 android 应用程序接口，基本上所有的组件都需要

第二个参数表示生成一个 Map(String ,Object)列表选项

第三个参数表示界面布局的 id 表示该文件作为列表项的组件，本实验为 R.layout.repositoryitem。

第四个参数表示该 Map 对象的哪些 key 对应 value 来生成列表项。

第五个参数表示来填充的组件 Map 对象 key 对应的资源，依次填充组件顺序有对应关系。

其中 Map(String ,Object)列表选项定义与赋值如下，

```
List<Map<String, String>> repoListData = new ArrayList<>();
for (Repos repo : reposList) {
    Map<String, String> temp = new LinkedHashMap<>();
    temp.put( k: "name", repo.getName());
    temp.put( k: "language", repo.getLanguage());
    temp.put( k: "intro", repo.getDescription());
    repoListData.add(temp);
}
```

四、 课后实验结果

五、 实验思考及感想

这次实验学习的新知识比较多，学习使用 Retrofit 实现网络请求，学习 RxJava 中 Observable 的使用，复习同步异步概念很多知识一开始都很陌生，上网查阅了很多的博客，一点一点地学习 java 的注解，接口类，工厂类的作用，Retrofit 的工作原理和使用方法等。通过这次实验，学习了网络请求的过程。感觉收获良多。

作业要求：

1. 命名要求: 学号_姓名_实验编号，例如 15330000_林 XX_lab1。
2. 实验报告提交格式为 pdf。
3. 实验内容不允许抄袭，我们要进行代码相似度对比。如发现抄袭，按 0 分处理。