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）

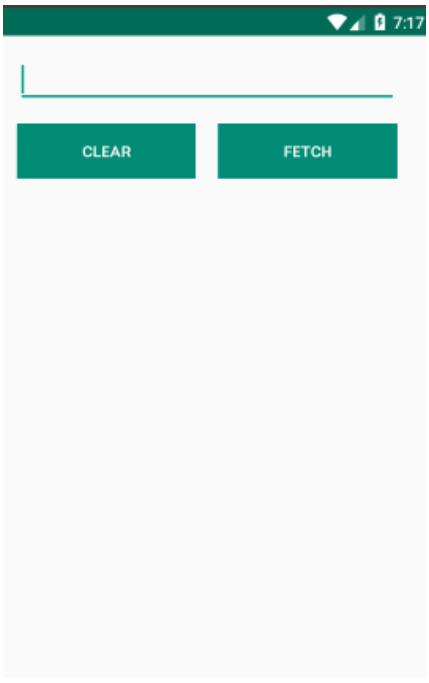
课程名称：移动应用开发　　任课教师： 郑贵峰

年级	15 级	专业（方向）	软件工程（移动信息工程） 互联网方向
学号	15352211	姓名	林苗
电话	13763360840	Email	554562948@qq.com
开始日期	2017.12.23	完成日期	2017.12.24

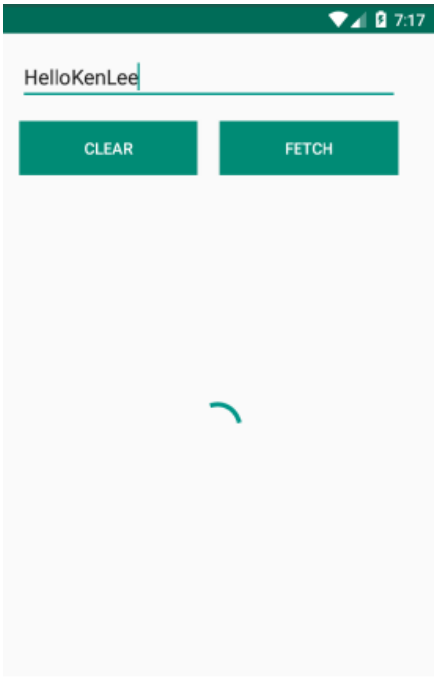
一、 实验题目

实验九
Retrofit+RxJava+OkHttp 实现网络请求

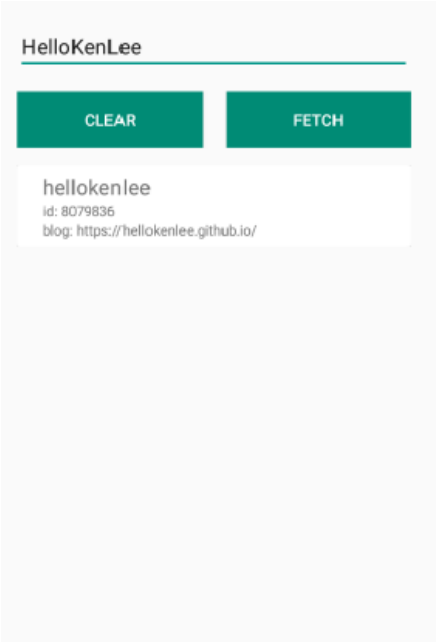
二、 实现内容



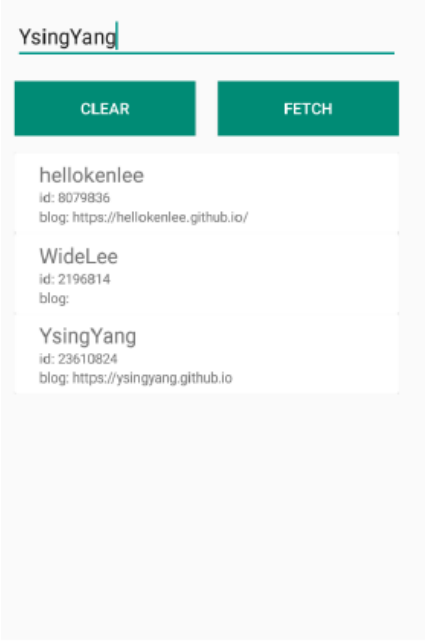
主界面



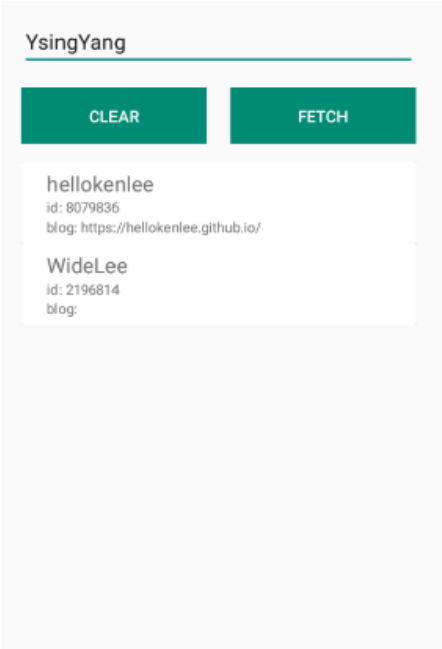
搜索用户



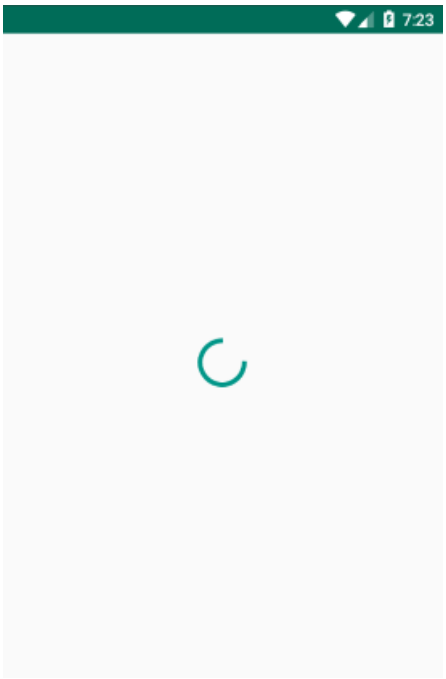
搜索结果



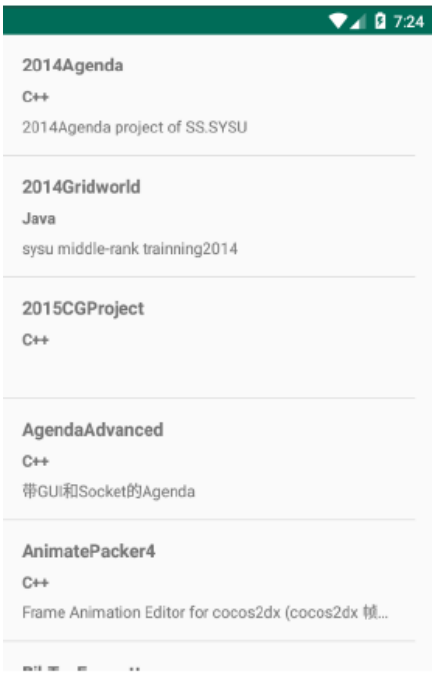
历史搜索结果



长按删除



点击进入个人详情页面

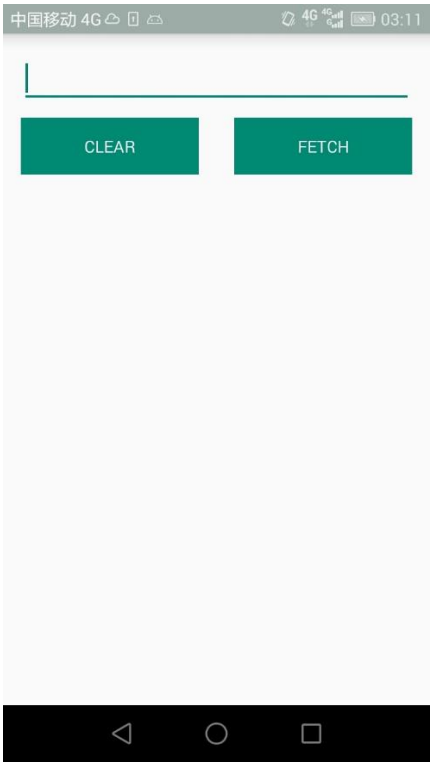


详情页面

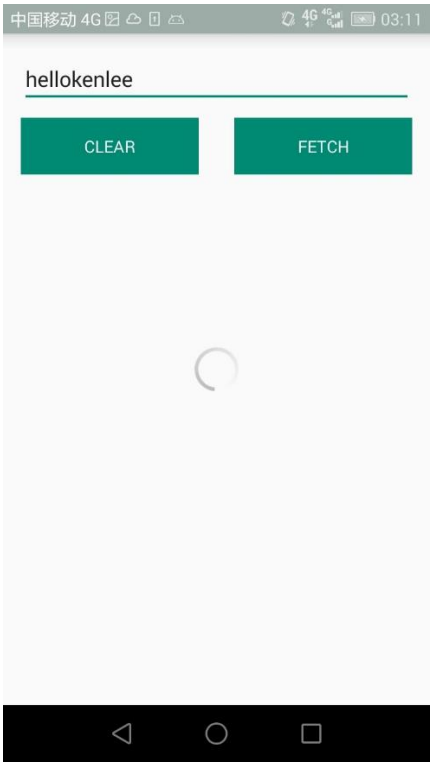
对于 User Model, 显示 id, login, blog
对于 Repository Model, 显示 name, description,language
(特别注意， 如果 description 对于 1 行要用省略号代替)

三、 课堂实验结果

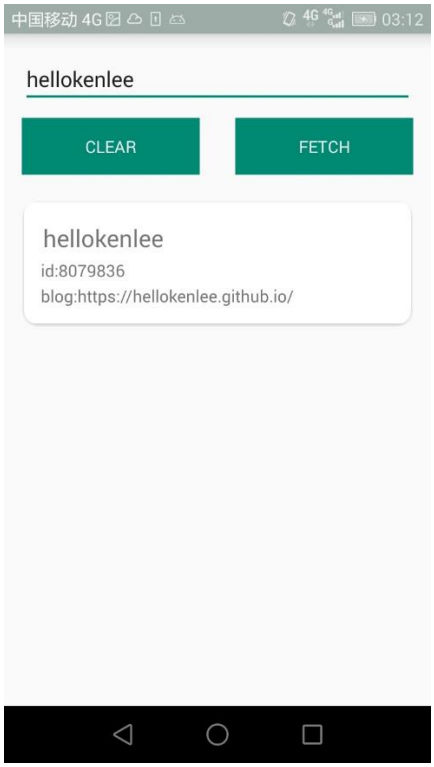
（1） 实验截图



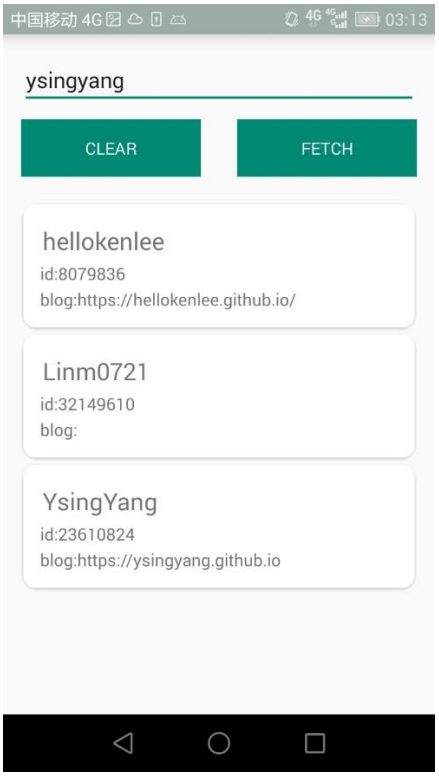
初始界面



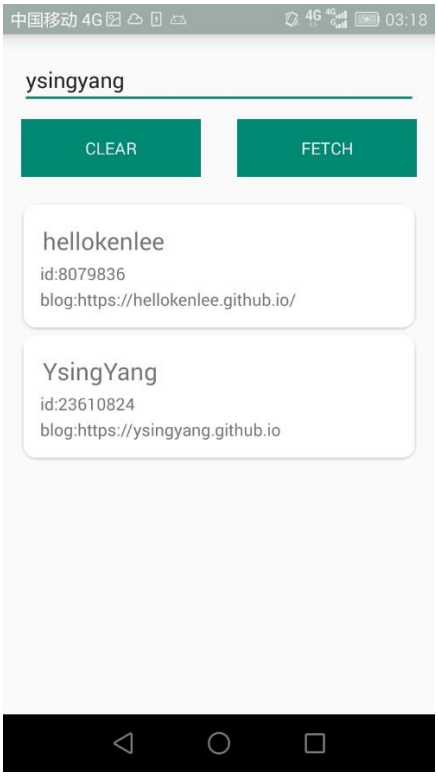
搜索时有进度条



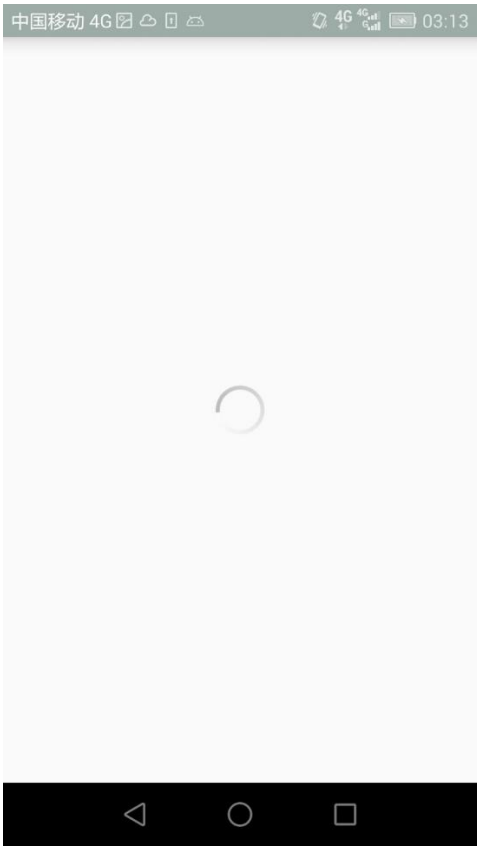
搜索结果



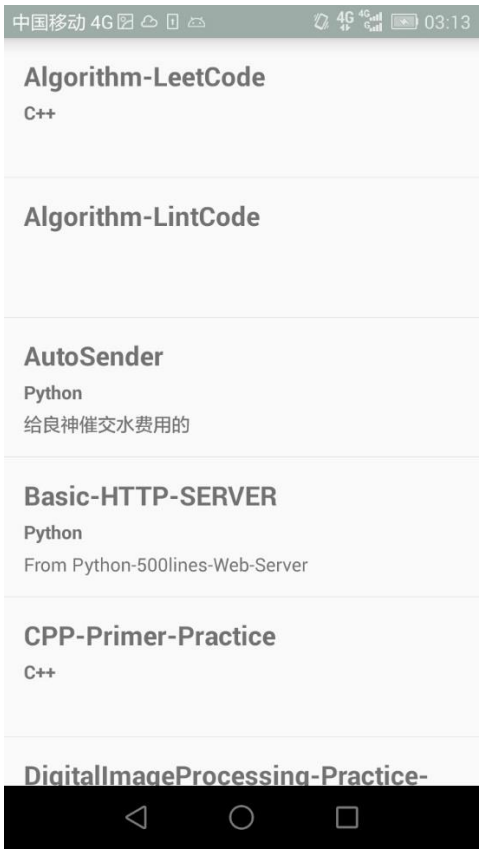
历史搜索结果



长按实现删除



单击进入个人主页，有滑动条



主页内容显示

（2） 实验步骤及关键代码

1） 添加依赖

本次实验用到的依赖较多，因此一次性添加所有依赖较为方便：

```
compile 'com.android.support:appcompat-v7:25.3.1'
compile 'com.android.support.constraint:constraint-layout:1.0.2'
compile 'com.android.support:recyclerview-v7:25.4.0'
compile 'com.android.support:cardview-v7:25.4.0'
compile 'com.android.support:design:25.4.0'
compile 'com.android.support:percent:25.4.0'
compile 'jp.wasabeef:recyclerview-animators:2.2.7'
compile 'com.android.support:support-core-utils:25.4.0'
compile 'com.squareup.retrofit2:converter-gson:2.1.0'
compile 'io.reactivex:rxjava:1.0.+'
compile 'io.reactivex:rxandroid:0.23.+'
compile 'com.squareup.retrofit2:retrofit:2.1.0'
compile 'com.squareup.retrofit2:adapter-rxjava:2.1.0'
testCompile 'junit:junit:4.12'
```

在 build.gradle 中：

```
allprojects {
    repositories {
        jcenter()
        //lab3 add start
        maven{
            url "https://maven.google.com"
        }
        //lab 3 add end
    }
}
```

在 AndroidManifest.xml 中：

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

2) 编写布局文件

这次需要两个界面，但用户搜索列表 RecyclerView 和主页库列表 ListView 都需要实现子项，因此共 4 个 xml，分别是：主页面 activity_main.xml、个人主页 activity_repo.xml、用户搜索列表子项 item.xml、库列表子项 repoitem.xml，布局不难编写，这里就不一一贴代码了。

搜索列表的子项用 CardView，可以有圆角等设置，和 LinearLayout 等其它布局一样，里面可以存放 TextView 等：

```
<android.support.v7.widget.CardView
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    card_view:cardCornerRadius="10dp"
    card_view:cardPreventCornerOverlap="true"
    card_view:cardUseCompatPadding="true"
    card_view:contentPadding="10dp">
```

值得注意的是，主页面中，要求文档中的输入框的线是绿色的，因此需要在 styles.xml 中自定义一个 style：

```
<style name="myEdit" parent="Theme.AppCompat.Light">
    <item name="colorControlNormal">#008B75</item>
    <item name="colorControlActivated">#008B75</item>
</style>
```

在 EditText 中引用：

```
<EditText
    android:id="@+id/search"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:theme="@style/myEdit"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

进度条 ProcessBar：

```
<ProgressBar
    android:id="@+id/pg"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    />
```

在个人主页中，要求 description 用一行显示，超出用省略号表示，在 TextView 中加上这两行关键代码：

```
<TextView
    android:id="@+id/description"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="2014Agenda project of SS.SYSU"
    android:layout_marginTop="5dp"
    android:maxLines="1"
    android:ellipsize="end"
    />
```

3) Retrofit 定义 Model 类

通过 API 查看获取到的数据格式，编写 Github 类和 Repo 类：

```
public class Github {

    private String login;
    private int id;
    private String  blog;

    public String  getLogin() {return login;}
    public int getId() {return id;}
    public String getBlog() {return blog;}

}
```

```
public class Repo {

    private String name;
    private String language;
    private String description;

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

}
```

4) 为 Retrofit 对象提供相应的 Interface

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

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

注意其中的 Observable 的类型为 rx

```
import rx.Observable;
```

一开始 import 成 database 类型的，结果 subscribeOn 报错。

5) 构造 Retrofit 对象实现网络访问（ServiceFactory 类）

Retrofit 也是基于 OkHttp 的封装，可以自己配置 OkHttp 对象

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

构造 Retrofit 对象实现网络访问：

```
public static Retrofit createRetrofit(String baseUrl) {
    return new Retrofit.Builder()
            .baseUrl(baseUrl)
            .addConverterFactory(GsonConverterFactory.create()) //添加GSon Converter
            .addCallAdapterFactory(RxJavaCallAdapterFactory.create()) //RxJavaCall Adapter
            .client(createOkHttp())
            .build();
}
```

6) 自定义 RecyclerView 的 Adapter

```
public class CardAdapter extends RecyclerView.Adapter<CardAdapter.ViewHolder>{

    private OnRecyclerViewItemClickListener mItemClickListener = null;
    private List<Github> GihubList = new ArrayList<>();

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

    public void setOnItemClickListener(OnRecyclerViewItemClickListener onItemClickListener){
        this.mItemClickListener = onItemClickListener;
    }
```

```
//添加子项
public void addItem(Github github) {
    GihubList.add(github);
    notifyDataSetChanged();
}
//得到相应位置的子项
public Github getItem(int position) { return GihubList.get(position); }
//移除相应位置的子项
public void removeItem(int position) {
    GihubList.remove(position);
    notifyItemRemoved(position);
}
//移除所有子项
public void removeAllItem() {
    GihubList.clear();
    notifyDataSetChanged();
}

//获得Item的总数
@Override
public int getItemCount() { return GihubList.size(); }
//创建Item视图
@Override
public CardAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item, parent, false);
    final ViewHolder holder = new ViewHolder(view);
    holder.itemView.setOnClickListener((v) -> {
            if(mItemClickListener != null)
                mItemClickListener.onClick(holder.getAdapterPosition());
    });
    holder.itemView.setOnLongClickListener((v) -> {
            if(mItemClickListener != null)
                mItemClickListener.onLongClick(holder.getAdapterPosition());
            return false;
    });
    return holder;
}
```

```
    @Override
    public void onBindViewHolder(final CardAdapter.ViewHolder holder, int position) {
        //绑定数据到正确的Item视图上
        Github github = GihubList.get(position);
        holder.login.setText(github.getLogin());
        holder.id.setText(String.valueOf(github.getId()));
        holder.blog.setText(github.getBlog());
    }

    static class ViewHolder extends RecyclerView.ViewHolder
    {
        TextView id, login, blog;

        ViewHolder(View view)
        {
            super(view);
            //通过id获取view
            login = (TextView)view.findViewById(R.id.name);
            id = (TextView)view.findViewById(R.id.id);
            blog = (TextView)view.findViewById(R.id.blog);
        }
    }
}
```

7) 实现主界面 MainActivity

定义成员变量：

```
private EditText SearchEdit; //编辑框
private Button ClearButton; //清除按钮
private Button FetchButton; //搜索按钮
private ProgressBar ProgBar; //进度条

private CardAdapter mAdapter; //适配器
private RecyclerView mRecyclerView; //RecyclerView列表
```

初始化函数：

```
void init() {
    SearchEdit = (EditText)findViewById(R.id.search);
    ClearButton = (Button)findViewById(R.id.clear);
    FetchButton = (Button)findViewById(R.id.fetch);
    ProgBar = (ProgressBar)findViewById(R.id.pg);

    mRecyclerView = (RecyclerView)findViewById(R.id.recyclerview);
    mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
    mAdapter = new CardAdapter() ;
    mRecyclerView.setAdapter(mAdapter);
}
```

在 onCreate 函数中，先设置进度条不可见：

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    init();//绑定id

    ProgBar.setVisibility(View.GONE);//设置进度不可见
```

清空按钮点击事件：

```
//清空所有用户github
ClearButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        mAdapter.removeAllItem();
    }
});
```

搜索按钮单击事件，先取得编辑框的名字，然后通过 Retrofit 创建相应的访问实例，将找到的结果存到适配器绑定的队列中：

```
//搜索
FetchButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        ProgBar.setVisibility(View.VISIBLE);//显示进度条
        String name = SearchEdit.getText().toString();//获取搜索的名字
        ServiceFactory.createRetrofit("https://api.github.com")
                .create(GithubService.class)
                .getUser(name)
                .subscribeOn(Schedulers.newThread())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(new Subscriber<Github>() {
                    //完成传输
                    public final void onCompleted() {
                        ProgBar.setVisibility(View.GONE);//进度条消失
                        System.out.println("完成传输");
                    }
                    //出错
                    public void onError(Throwable throwable) {
                        Toast.makeText(MainActivity.this, "请确认你搜索的用户存在", Toast.LENGTH_SHORT).show();
                        ProgBar.setVisibility(View.GONE);
                        Log.e("debug","error");
                        throwable.printStackTrace();
                    }

                    //找到结果
                    public void onNext(Github github) {
                        mAdapter.addItem(github);//将结果加入到适配器的队列
                    }
                });

    }
});
```

RecyclerView 单击与长按事件：

单击：将对应的名字传给跳转的页面；长按：获取下标直接删除适配器队列对应的元素。

```
mAdapter.setOnItemClickListener(new CardAdapter.OnRecyclerViewItemClickListener() {
    //单击跳转到库界面
    public void onClick(int position) {
        Intent intent = new Intent(MainActivity.this, RepoActivity.class);
        intent.putExtra("username", mAdapter.getItem(position).getLogin()); //传入用户名
        MainActivity.this.startActivity(intent);
    }
    //长按删除
    public void onLongClick(int position) {
        mAdapter.removeItem(position);
    }
});
```

8) 实现个人主页 RepoActivity

定义成员变量：

```
private ProgressBar progressBar; //进度条
private ListView lv; //ListView列表
private List<Map<String, String>> repoListData = new ArrayList<>(); //适配器绑定的队列
```

在 onCreate 函数中，绑定 id：

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_repo);

    progressBar = (ProgressBar)findViewById(R.id.pb);
    lv = (ListView)findViewById(R.id.listview);
```

获取得到的用户名，通过 Retrofit 创建相应的访问实例，同时匹配各种情况下进度条的显示与否

```
String userName = getIntent().getStringExtra("username");//获得用户名
ServiceFactory.createRetrofit("https://api.github.com")
        .create(GithubService.class)
        .getRepos(userName) //获得库
        .subscribeOn(Schedulers.newThread())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(new Subscriber<List<Repo>>() {
            //传输完成
            public void onCompleted() {
                progressBar.setVisibility(View.GONE); //隐藏进度条
                lv.setVisibility(View.VISIBLE);//显示listview结果
                System.out.println("完成传输");
            }
```

```
            public void onError(Throwable throwable) {
                Toast.makeText(RepoActivity.this, "出错啦", Toast.LENGTH_SHORT).show();
                progressBar.setVisibility(View.GONE); //隐藏进度条
                lv.setVisibility(View.VISIBLE); //显示listview
                throwable.printStackTrace();
            }

            public void onNext(List<Repo> list) {

                repoListData.clear();
                for(int i=0; i<list.size(); i++){
                    Repo repo = list.get(i);
                    Map<String, String> temp = new LinkedHashMap<>();
                    temp.put("name", repo.getName());
                    temp.put("language", repo.getLanguage());
                    temp.put("description", repo.getDescription());
                    repoListData.add(temp);
                }
                renewAdapter();
            }
        });
```

其中更新适配器的函数定义如下：

```
private void renewAdapter() {

    SimpleAdapter mSimpleAdapter = new SimpleAdapter(this, repoListData,
            R.layout.repoitem,
            new String[]{"name", "language", "description"},
            new int[]{R.id.name, R.id.language, R.id.description});
    lv.setAdapter(mSimpleAdapter);
}
```

（3） 实验遇到的困难及解决思路

- 编辑框默认是灰色的，但是参考 apk 里面是绿色的，于是想把编辑框的底线也改成绿色的。

解决思路：在 styles.xml 中声明自定义样式：

```
<style name="myEdit" parent="Theme.AppCompat.Light">
    <item name="colorControlNormal">#008B75</item>
    <item name="colorControlActivated">#008B75</item>
</style>
```

在 EditText 中引用：

```
<EditText
    android:id="@+id/search"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:theme="@style/myEdit"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

- 在个人主页获取 description，用一行显示，超出部分用省略号代替。

解决思路：有博客给出解决方法：

```
android:singleLine="true"
android:maxEms="8"
android:ellipsize="end"
```

试了一下，发现可以实现超出部分用省略号代替，但是 singleLine 这个属性被划掉了，提示是过时了，因此照着提示改为：

```
android:maxLines="1"
android:ellipsize="end"
```

也可完美解决。其中 maxLines 表示最大行数不超过 1 行，而省略号则是在 ellipsize 这个属性中设置的。

- Retrofit 调用 subscribeOn 时报红。

解决思路：在群里看到好多人也是同样的问题，发现原来是 GithubService 中引入 Observable 的类型搞错了，应该引入 rx.Observable，而不是 database 的。

四、 实验思考及感想

这次需要用到的新知识不多，但是要自己自定义 RecyclerView 的适配器，虽然在 Lab3 的时候已经写过一次了，但那个时候是直接照着 ppt 打的，只是依样画葫芦将变量改一下，并没有很认真地去思考适配器和 ViewHolder 是什么。这次虽然也是参考的，不过这次多了一些自己的思考：适配器中有自己的队列，这个队列就是 RecyclerView 展现出来的。在适配器中可以定义添加子项、移除相应位置子项的函数，可以使用原本 Adapter 类的 notifyDataSetChanged 和 notifyItemRemoved 函数进行数据同步更新操作，在其它 activity 中可以直接调用新定义的函数，按需所取，实现增删改的操作。在适配器中，还需要重写点击函数。要获取点击的相应位置，就需要用到 ViewHolder，首先定义 ViewHolder 静态类，设置其成员变量以及绑定 UI 界面的 id，其次需要绑定数据到正确位置的 Item 视图上，然后再创建视图，通过 ViewHolder 来获取位置，实现正确位置的点击。

访问网络资源，这次用到的是 Retrofit，Retrofit 通过相应的 Interface 来获取得到用户信息，然后新建线程进行网络访问，用 observeOn 在主线程处理请求结果。subcribe 中的 onComplete、onError、onNext 函数分别用来处理完成网络请求、网络请求出错、网络请求返回结果时执行的函数，在里面可定义我们想要进行的操作，如当请求完成时，进度条消失，当收到结果时，加入到适配器的队列中等。

虽然这次真正体现在代码上的新知识不多，而且大部分 ppt 都有给，但是真正理解还是需要一定时间。但是能初步实现网络访问，了解其中的原理，还是觉得收获颇丰。