

4 广场模块

这一章要完成广场模块的基本功能实现，有一点难度，主要包含以下内容：

- ListView
- RecyclerView
- 异步加载
- 数据缓存
- AsyncTask

4.1 界面布局设计

(1) 整体框架设计

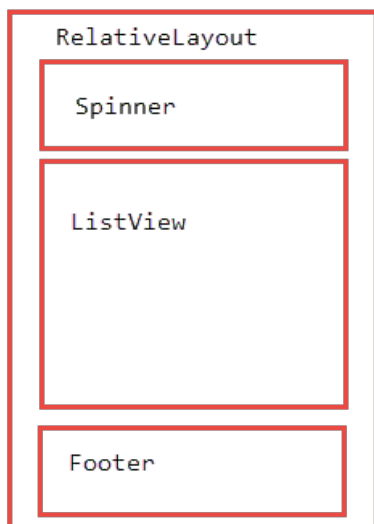
首先分析一下广场界面的原型设计，设计布局时需要和其他的原型放在一起观察。



显然可以把界面分为上中下三部分，其中底部的菜单栏和个人信息界面是相同的，所以可以单独提取出来复用。

在 `java/包名/ui/activity` 下新建一个 `Activity` 作为广场界面，取名为 `SquareActivity`，布局文件是 `res/layout/activity_square.xml`。

整体的框架设计如下图所示：

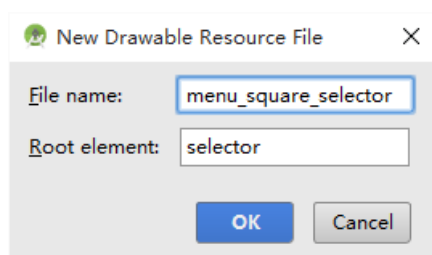


这里的布局为相对布局，最上方是一个 `Spinner` 下拉列表框控件，中间是 `ListView` 展示主要内容，底部的 `Footer` 并不是一个控件名称，而是需要提取出来的复用的菜单布局。

(2) 底部菜单的实现

底部的三个菜单选项互斥，即一次只能选中一个，并且每个选项选中与未选中时样式有区别，我们先来实现样式的选择功能。

光标定位到 `res/drawable` 目录，右键选择“New Drawable resource file”，如图命名为 `menu_square_selector`，新建广场菜单的样式选择器。



打开新建的 `menu_square_selector.xml` 文件，添加如下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_checked="true" android:drawable="@drawable/menu_square_selected"/>
    <item android:state_checked="false" android:drawable="@drawable/menu_square"/>
</selector>
```

`state_checked` 属性的值为 `true` 就是选中，为 `false` 就是未选中，这样我们就给广场菜单添加了选中与未选中的两种状态，类似地添加另外两个菜单的选择样式。

`menu_publish_selector.xml` 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_checked="true" android:drawable="@drawable/menu_publish_hover"/>
    <item android:state_checked="false" android:drawable="@drawable/menu_publish"/>
</selector>
```

`menu_my_selector.xml` 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_checked="true" android:drawable="@drawable/menu_my_selected"/>
    <item android:state_checked="false" android:drawable="@drawable/menu_my"/>
</selector>
```

这样三个菜单的图片样式我们就设置好了，此外图片下方的文字也同样有两种状态，所以再新建一个菜单文本的选择器：`menu_text_color_selector.xml` 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_checked="true" android:color="@color/colorOrange"/>
    <item android:state_checked="false" android:color="@color/colorGrey"/>
</selector>
```

完成以上工作后，我们来把底部菜单整合起来。新建一个布局文件：`res/layout/footer.xml`，打开 `footer.xml` 文件，编写代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_alignParentBottom="true">

    <RadioGroup
        android:id="@+id/menu_radio_group"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:measureWithLargestChild="true"
        android:orientation="horizontal">

        <RadioButton
            android:id="@+id/menu_square_radio_button"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:gravity="center"
            android:button="@null"
            android:drawableTop="@drawable/menu_square_selector"
            android:text="@string/menu_text_square"
            android:textColor="@drawable/menu_text_color_selector"/>

        <RadioButton
            android:id="@+id/menu_publish_radio_button"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:gravity="center"
            android:button="@null"
            android:drawableTop="@drawable/menu_publish_selector"/>

        <RadioButton
            android:id="@+id/menu_my_radio_button"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:button="@null"
            android:gravity="center"
            android:drawableTop="@drawable/menu_my_selector"
            android:text="@string/menu_text_my"
            android:textColor="@drawable/menu_text_color_selector"/>

    </RadioGroup>
</LinearLayout>
```

这里使用单选按钮 `RadioButton` 实现菜单按钮的互斥，把互斥的按钮放在同一个 `RadioGroup` 中，就可以实现一次只选其中一项了。

`android:button="@null"` 的作用是不显示单选按钮前面的圆圈。

`android:drawableTop="@drawable/menu_square_selector"` 表明图片显示在文字的上方，其中 `menu_square_selector` 就是之前设置的样式选择器。

文字内容还是定义在 `res/values/strings.xml` 文件中：

```
<resources>
    ....
    <string name="menu_text_square">广场</string>
    <string name="menu_text_my">我的</string>
</resources>
```

（3）顶部下拉列表的实现

下拉列表这里使用Spinner来实现，首先创建列表数据内容，可以直接在 `res/values/strings.xml` 里定义，小编这里又新建了一个文件 `res/values/arrays.xml` 用来存放数组数据。打开 `arrays.xml`，编写代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="select_distance_array">
        <item>附近100米</item>
        <item>附近500米</item>
        <item>附近1000米</item>
    </string-array>
</resources>
```

列表的背景颜色这里先不改，因为列表右侧的小三角箭头也是定义在背景里的，等小编找到图标资源，在下一个版本里更改。

（4）整合页面布局

打开 `res/layout/activity_square.xml` 文件，把界面的整体框架搭建起来：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.geekband.demo.moran.ui.activity.SquareActivity">

    <Spinner
        android:id="@+id/distance_spinner_square"
        android:layout_width="match_parent"
        android:layout_height="42dp"
        android:entries="@array/select_distance_array"
        android:spinnerMode="dropdown"
        android:textAlignment="center"/>

    <ListView
        android:id="@+id/square_list_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/distance_spinner_square"
        android:layout_marginTop="6dp"
        android:layout_marginBottom="44dp"/>

    <include layout="@layout/footer"/>
</RelativeLayout>
```

Spinner 控件的 `entries` 属性就是列表数据源。

`<include layout="@layout/footer"/>` 这句代码就把之前定义的底部菜单布局加入了进来，其他页面也需要相同的底部菜单的话，只有在布局里加上这一行就可以实现复用了。

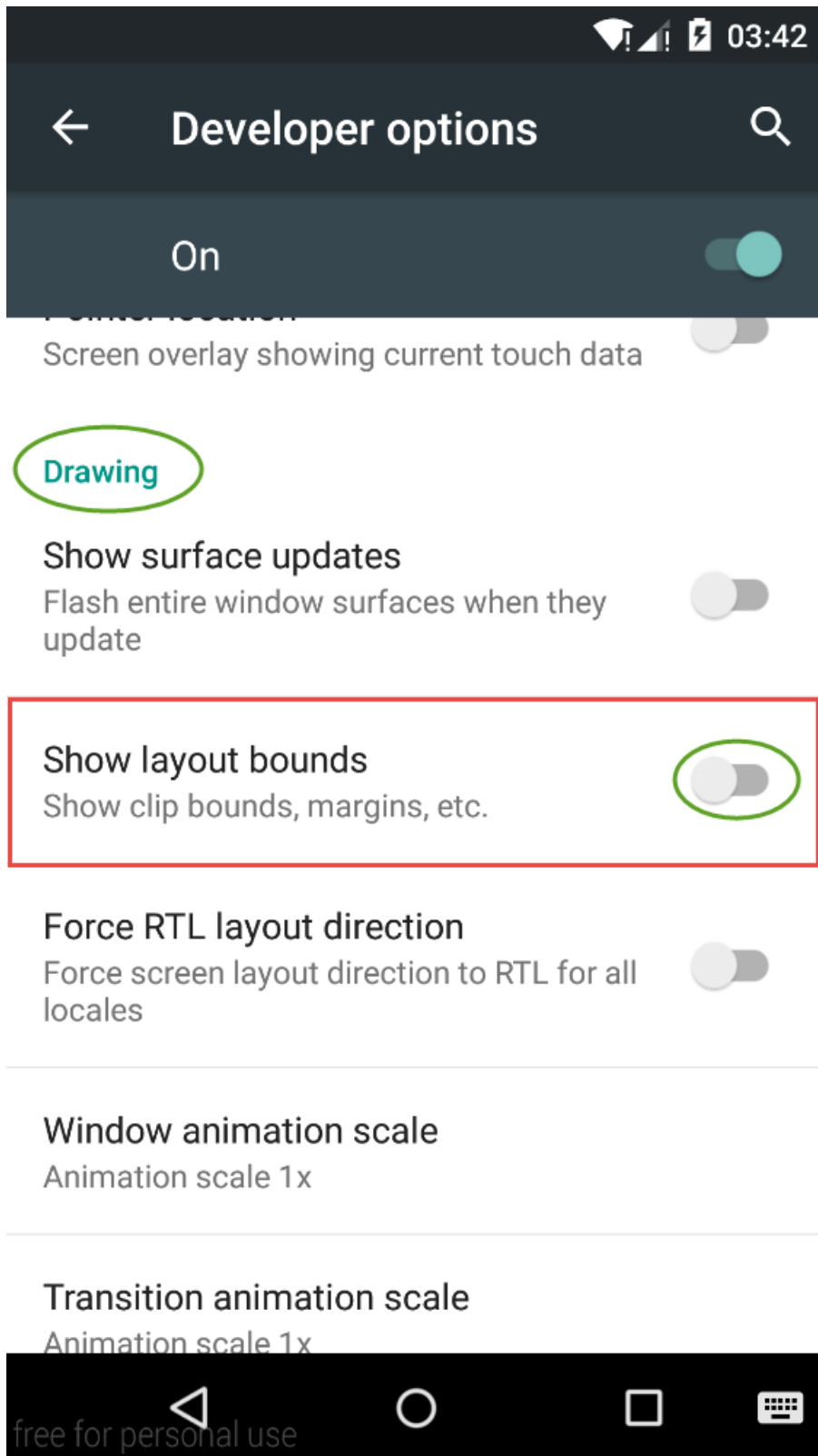
现在可以先部署到模拟器上运行一下了。记得在应用清单里把当前 `activity` 注册为启动项，打开 `AndroidManifest.xml` 作如下定义：

```
<application>
    ....
    <activity
        android:name=".ui.activity.SquareActivity"
        android:label="@string/title_activity_square"
        android:theme="@style/AppTheme.NoActionBar" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

点击运行，大家可以测试一下列表框和菜单按钮的选中状态是否有效。

中间的ListView应该是一块空白，因为我们没有给它内容，打开模拟器的开发者模式（激活方式和真机一样，选择“设置→关于手机→版本号”，连续点击版本号，激活后再进入手机设置就可以看到开发者选项了），如图，在 **Drawing**（绘图）节点下，选择 **Show layout bounds**（显示布局边界），再返回应用界面就可以看到ListView了。



4.2 使用ListView

(1) 获取控件

打开 `SquareActivity`，添加如下代码：

```

public class SquareActivity extends AppCompatActivity {
    //声明成员
    private Spinner mSpinner;
    private ListView mListView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_square);

        //获取控件
        mSpinner=(Spinner) findViewById(R.id.distance_spinner_square);
        mListView=(ListView) findViewById(R.id.square_list_view);
    }
}

```

（2）定义数据模型

列表视图ListView在Android开发中非常非常重要，大家可以打开手机中的常用应用，看一看有没有列表视图的影子，我们这里先拿出一节来介绍ListView的用法。

要使用ListView，需要提供ListView里每一条项目的布局以及数据。

先来准备数据，我们先在本地做一些练习，在列表里显示姓名和年龄。

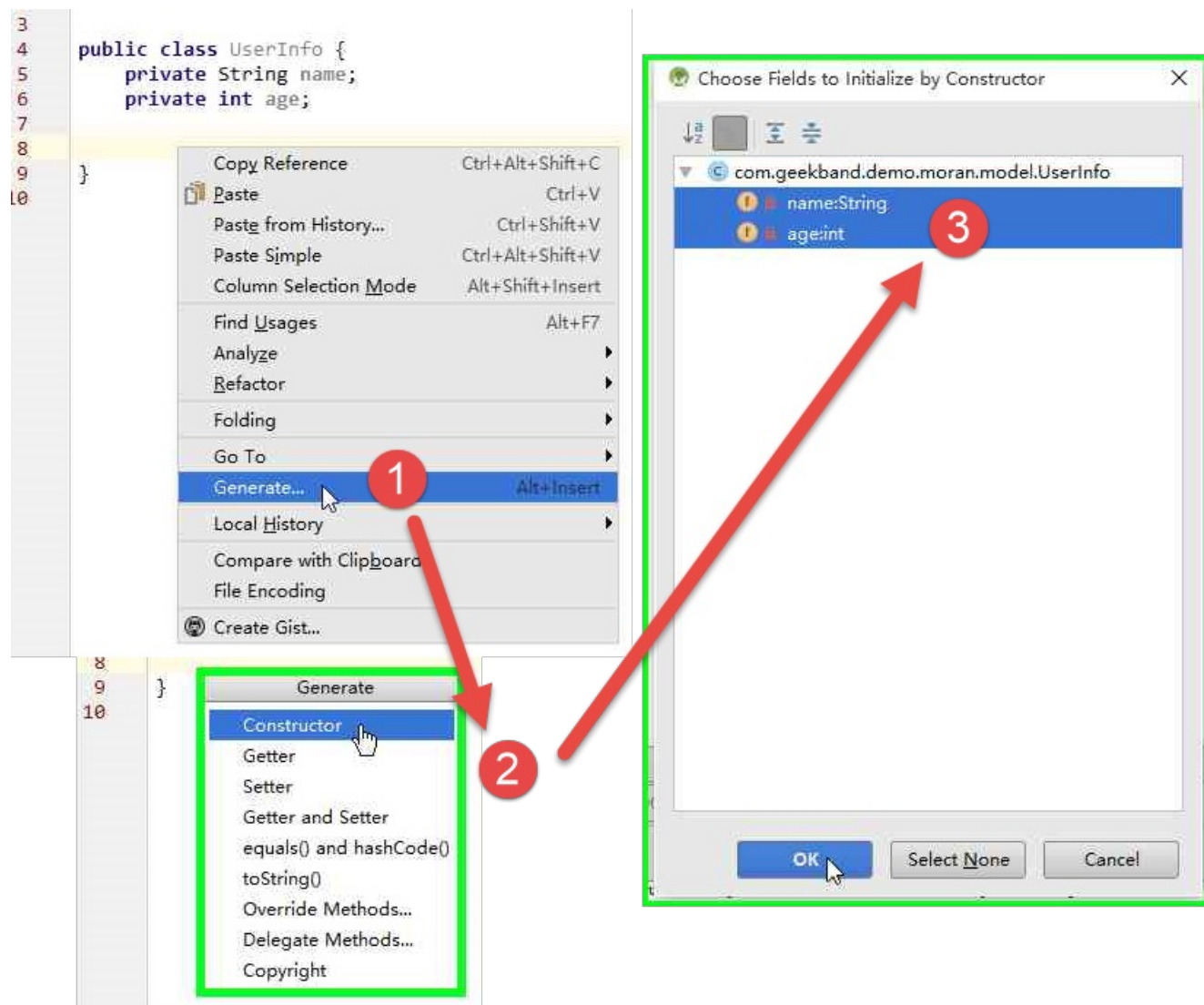
光标定位到 `java/包名/model`，右键选择“New|Java Class”，取名为 `UserInfo`，在其中添加数据字段：

```

public class UserInfo {
    private String name;
    private int age;
}

```

然后，有一个小技巧，光标放在 `private int age;` 的下一行，右键选择“Generate...”，弹出窗口中选择第一个“Constructor”，在新弹窗中按住 `Shift` 或 `Ctrl` 键，把两项都选中，再点击OK按钮，构造器的代码就自动生成了。



同样地，右键选择“Generate...|Getter and Setter”，然后选中所有项目，点击OK，取值和赋值方法也自动生成了。

最后得到的代码如下：

```
public class UserInfo {  
    private String name;  
    private int age;  
  
    public UserInfo(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int age) {  
        this.age = age;  
    }  
}
```

这些自动生成的代码也可以手写，其实对Java语法不熟悉的也推荐手写一遍，熟悉写这些代码的“手感”。

(3) 定义项目布局

再来定义 `ListView` 中每一项内容的布局，在 `res/layout` 目录下新建布局文件 `item_list_view_square.xml`，在其中编写代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/user_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <TextView
        android:id="@+id/user_age"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

(4) 使用Adapter

有了数据和布局，我们就可以使用适配器 `Adapter` 来给 `ListView` 填充数据，在 `java/应用包名/ui/adapters` 上右键，选择“New|Java Class”，取名为 `UserInfoAdapter`，打开文件修改代码，让其继承于 `BaseAdapter`：

```
public class UserInfoAdapter extends BaseAdapter{

}
```

这时候，AS会提示有错误，光标放在 `BaseAdapter` 或标红代码的任意位置，使用快捷键组合 `Alt (Option) + Enter`，在弹出的对话框中选择 `Implement Methods`，或者使用刚刚介绍的方法，在class内部右键，选择“Generate...|Implement Methods...”，选中所有方法，点击OK，就自动生成了 `BaseAdapter` 的方法实现，结果如下图所示：

```
public class UserInfoAdapter extends BaseAdapter{
    //获取列表项目数量
    @Override
    public int getCount() {
        return 0;
    }

    //获取列表指定项目
    @Override
    public Object getItem(int position) {
        return null;
    }

    //获取列表项目位置
    @Override
    public long getItemId(int position) {
        return 0;
    }

    //获取列表项目视图
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        return null;
    }
}
```

(5) 实现Adapter的方法

自动生成的代码并不能满足我们的需求，打开 `UserInfoAdapter`，实现其构造函数以及其他方法：

```

public class UserInfoAdapter extends BaseAdapter{

    private Context mContext;
    private LayoutInflater mLayoutInflater;
    private List<UserInfo> mUserInfos = new ArrayList<>();

    public UserInfoAdapter(Context mContext, List<UserInfo> userInfos) {
        this.mContext = mContext;
        this.mLayoutInflater = (LayoutInflater) mContext.
            getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        this.mUserInfos = userInfos;
    }
    ....
}

```

（上面的 `getSystemService` 换行是为了导出文档能看全）

上面的代码定义了该适配器的构造方法，其中 `Context` 为要传入的上下文，也就是哪个 `Activity` 使用这个适配器，`LayoutInflater` 用来填充布局，`List<UserInfo>` 为传入的数据。接着实现其他方法：

```

//获取列表项目数量
@Override
public int getCount() {
    return mUserInfos.size();
}

//获取列表指定项目
@Override
public Object getItem(int position) {
    return mUserInfos.get(position);
}

//获取列表项目位置
@Override
public long getItemId(int position) {
    return position;
}

//获取列表项目视图
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    //返回视图
    convertView = mLayoutInflater.inflate(R.layout.item_list_view_square, null);

    //获取控件
    TextView nameTextView = (TextView) convertView.findViewById(R.id.user_name);
    TextView ageTextView = (TextView) convertView.findViewById(R.id.user_age);

    //绑定数据
    nameTextView.setText(mUserInfos.get(position).getName());
    ageTextView.setText(String.valueOf(mUserInfos.get(position).getAge()));

    return convertView;
}

```

上面的代码重写了基类适配器的方法，前面三个方法都很好理解，这里讲一下第四个 `获取列表项目视图` 的方法。

首先通过 `mLayoutInflater` 读取前面定义的 `res/layout/item_list_view_square.xml` 列表项目布局，返回给 `convertView` 视图。

然后在返回的视图里找到具体的视图控件，即获取对这些控件的引用。

接着给视图控件赋值，其中 `ageTextView.setText()` 方法中，传入的年龄需要转换一下，因为我们定义的 `UserInfo` 数据模型中，年龄字段是 `int` 类型，这样的话 `TextView` 的 `setText()` 方法会误将传入的参数当作是一个资源的 `id`。

最后，我们再把 `convertView` 返回给调用这个方法的对象。

（6）向 `ListView` 绑定数据

打开 `SquareActivity`，先构造一批数据：

```
public class SquareActivity extends AppCompatActivity {
    //声明成员
    private Spinner mSpinner;
    private ListView mListView;
    private List<UserInfo> mUserInfos;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_square);

        //获取控件
        mSpinner = (Spinner) findViewById(R.id.distance_spinner_square);
        mListView = (ListView) findViewById(R.id.square_list_view);

        //构造一批数据
        mUserInfos = new ArrayList<>();
        mUserInfos.add(new UserInfo("小明", 20));
        mUserInfos.add(new UserInfo("小王", 23));
        mUserInfos.add(new UserInfo("小红", 18));
        mUserInfos.add(new UserInfo("小明", 20));
        mUserInfos.add(new UserInfo("小王", 23));
        mUserInfos.add(new UserInfo("小红", 18));
        mUserInfos.add(new UserInfo("小明", 20));
        mUserInfos.add(new UserInfo("小王", 23));
        mUserInfos.add(new UserInfo("小红", 18));
        mUserInfos.add(new UserInfo("小明", 20));
        mUserInfos.add(new UserInfo("小王", 23));
        mUserInfos.add(new UserInfo("小红", 18));
        mUserInfos.add(new UserInfo("小明", 20));
        mUserInfos.add(new UserInfo("小王", 23));
        mUserInfos.add(new UserInfo("小红", 18));
        mUserInfos.add(new UserInfo("小明", 20));
        mUserInfos.add(new UserInfo("小王", 23));
        mUserInfos.add(new UserInfo("小红", 18));

    }
}
```

然后，创建填充 `UserInfo` 的适配器，传入上下文与用户信息数据：

```
.....
mUserInfos.add(new UserInfo("小红", 18));
//创建填充用户信息的适配器对象
UserInfoAdapter adapter = new UserInfoAdapter(SquareActivity.this, mUserInfos);
```

最后，给 `ListView` 设置适配器，绑定数据：

```
//创建填充用户信息的适配器对象
UserInfoAdapter adapter = new UserInfoAdapter(SquareActivity.this, mUserInfos);
//绑定数据
mListView.setAdapter(adapter);
```

现在，在模拟器上运行一下吧。

(7) 初步优化

`ListView` 的基本用法介绍完了，我们再回过头来看看 `Adapter` 的代码，打开 `java/包名/ui/adapters/UserInfoAdapter`，注意观察下面的代码：

```
//获取列表项目视图
@Override
public View getView(int position, View convertView, ViewGroup parent) {

    //返回视图
    convertView = mLayoutInflater.inflate(R.layout.item_list_view_square, null);

    //获取控件
    TextView nameTextView = (TextView) convertView.findViewById(R.id.user_name);
    TextView ageTextView = (TextView) convertView.findViewById(R.id.user_age);

    //绑定数据
    nameTextView.setText(mUserInfos.get(position).getName());
    ageTextView.setText(String.valueOf(mUserInfos.get(position).getAge()));

    return convertView;
}
```

上面的代码是每一个列表项目的数据填充方法，可以发现，每一个项目都需要解析布局文件，而这里所有项目的布局都是一样的，我们只需要解析一次就可以了，修改返回视图的代码：

```
//返回视图
if(convertView == null){
    convertView = mLayoutInflater.inflate(R.layout.item_list_view_square, null);
}
```

只有读取第一个项目的时候，才创建视图对象。

除此之外，得到视图后还需要在视图上寻找控件，并为找到的控件创建接收对象，这个过程也是比较损耗性能的。

在 `UserInfoAdapter` 的内部新建一个 `ViewHolder` 内部类：

```
//获取列表项目视图
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ....
}

//接收控件
class ViewHolder {
    TextView nameTextView;
    TextView ageTextView;
}
```

再把之前的代码改造如下：

```
//获取列表项目视图
@Override
public View getView(int position, View convertView, ViewGroup parent) {

    //返回视图
    ViewHolder viewHolder;
    if (convertView == null) {
        convertView = mLayoutInflater.inflate(R.layout.item_list_view_square, null);
        viewHolder = new ViewHolder();
        //获取控件
        viewHolder.nameTextView = (TextView) convertView.findViewById(R.id.user_name);
        viewHolder.ageTextView = (TextView) convertView.findViewById(R.id.user_age);

        convertView.setTag(viewHolder);
    } else {
        viewHolder = (ViewHolder) convertView.getTag();
    }

    //绑定数据
    viewHolder.nameTextView.setText(mUserInfos.get(position).getName());
    viewHolder.ageTextView.setText(String.valueOf(mUserInfos.get(position).getAge()));

    return convertView;
}

//接收控件
class ViewHolder {
    TextView nameTextView;
    TextView ageTextView;
}
```

先在外部声明 `viewHolder`，但是不实例化，只在第一次读取时实例化，然后用 `setTag()` 给视图关联额外的信息，这里用来存储 `viewHolder` 中的数据，以后要使用时再用 `getTag()` 方法返回 `viewHolder`。

这样一来，解析布局、寻找控件、创建接收对象就只有第一次读取时执行，以后的列表项目都可以直接套用了。

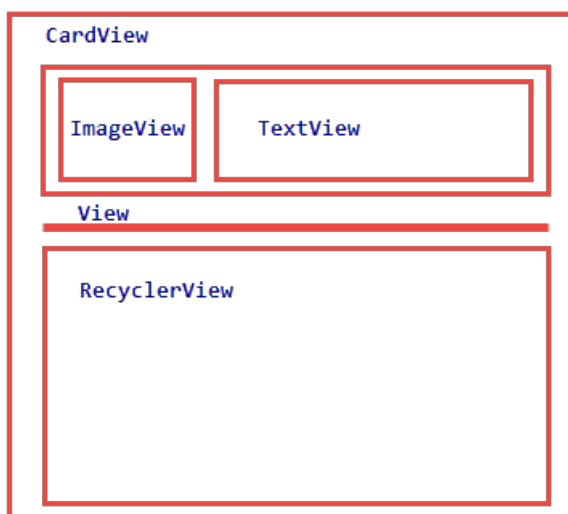
4.3 使用 RecyclerView

原型中每个列表项目是横向滚动的，我们可以用 `HorizontalScrollView` 来实现，这里使用的是 `RecyclerView`。

`RecyclerView` 是 `ListView` 的高级视图，是随 Android 5.0 发布的新组件，可以用来动态地展示一组数据。

(1) 列表项目布局设计

列表项目的布局设计如下图所示：



其中的布局标签省略掉了，在 `res/layout` 目录下新建布局文件 `item_node_square.xml`，在其中编写代码如下：

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView
    android:id="@+id/cv"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="160dp"
    android:layout_margin="8dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal">

            <ImageView
                android:id="@+id/location_image"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:src="@drawable/square_location"/>

            <TextView
                android:id="@+id/address_text"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:textSize="10sp"/>
        </LinearLayout>

        <View
            android:layout_width="match_parent"
            android:layout_height="1dp"
            android:background="@android:color/darker_gray"/>

        <android.support.v7.widget.RecyclerView
            android:id="@+id/recycler_view"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginTop="10dp"
            android:scrollbars="horizontal"/>
    </LinearLayout>
</android.support.v7.widget.CardView>

```

在输入 `CardView` 和 `RecyclerView` 时可能会报找不到依赖项的错误，因为它们是 `support.v7` 包中的组件，使用下面的两种方法导入依赖包：

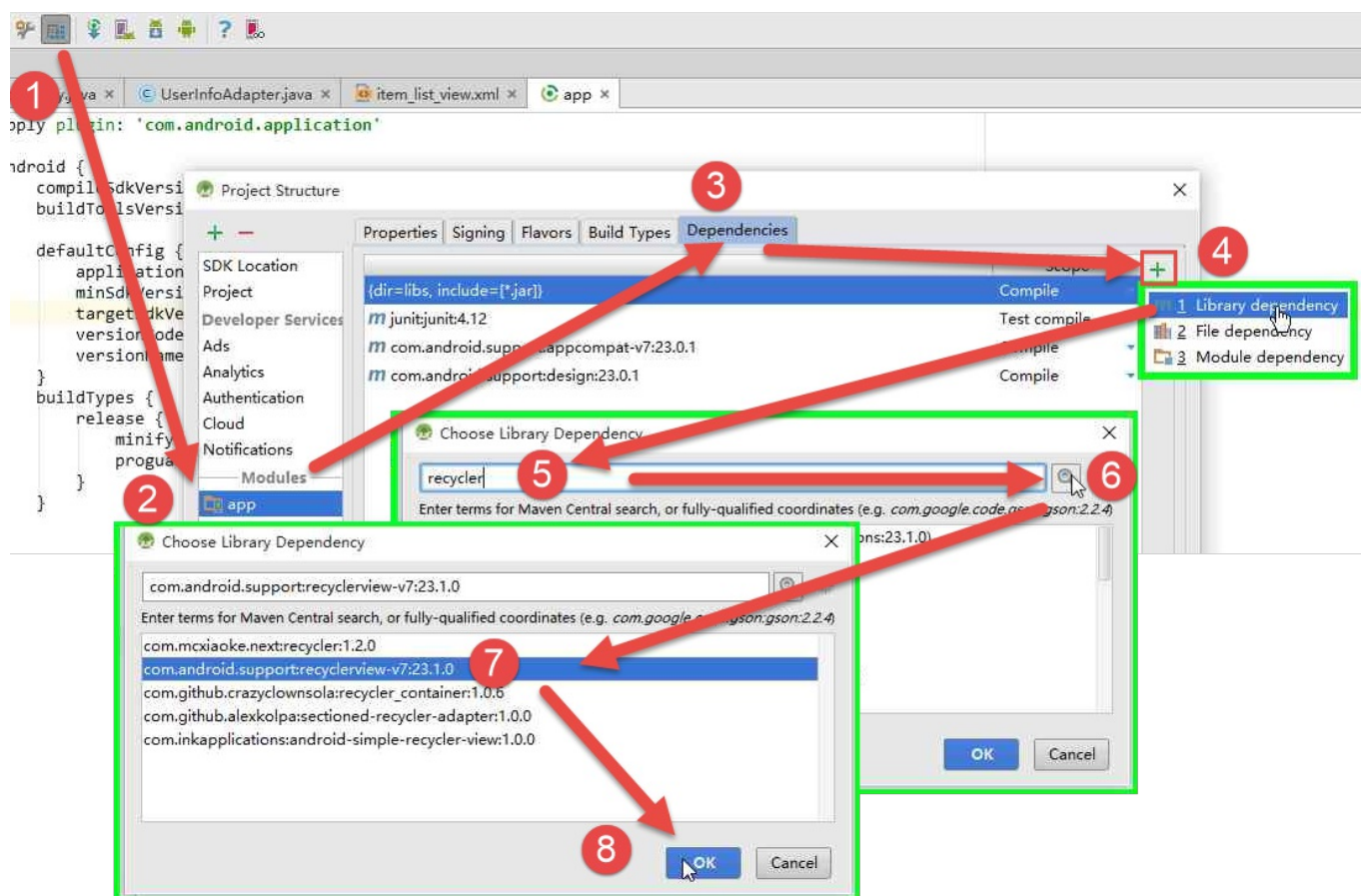
方法一：打开 `app/build.gradle` 文件，在 `dependencies` 中填入下图红框中的代码：

```

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.0.1'
    compile 'com.android.support:design:23.0.1'
    compile 'com.android.support:recyclerview-v7:23.1.0'
    compile 'com.android.support:cardview-v7:23.1.0'
}

```

方法二：如下图所示，在 `Project Structure` 中，按照图示步骤搜索添加。



小编的SDK编译版本选择的是23，所以 `RecyclerView` 和 `CardView` 的版本号都是23。大家的SDK是22的话，就选择版本号22的 `RecyclerView` 和 `CardView` 就好。

上面步骤完成后，可能需要重新编译一下工程，才能使设置生效。

(2) 创建数据模型

简单分析一下我们需要哪些数据，观察一下4.3步骤(1)中的布局设计。

首先，`CardView` 中上半部分，在位置图标右侧的地址数据是我们需要获取的。其次，下半部分的 `RecyclerView` 中，每一个项目包含一张图片和一条评论。一个地点对应了多个图片，而图片和评论在广场页面是一对一的。我们可以像下面一样，创建两个数据模型：

光标定位到 `java/包名/model` 目录，右键选择“New|Java Class”，新建数据模型取名为 `ImageItem`，编写代码如下：

```

public class ImageItem {
    private int imageId;
    private String comment;

    public ImageItem(int imageId, String comment) {
        this.imageId = imageId;
        this.comment = comment;
    }

    public int getImageId() {
        return imageId;
    }

    public void setImageId(int imageId) {
        this.imageId = imageId;
    }

    public String getComment() {
        return comment;
    }

    public void setComment(String comment) {
        this.comment = comment;
    }
}

```

类似地，新建模型“Node”，编写代码如下：

```

public class Node {
    private String address;
    private List<ImageItem> images;

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public List<ImageItem> getImages() {
        return images;
    }

    public void setImages(List<ImageItem> images) {
        this.images = images;
    }
}

```

其中，`ImageItem` 是每一对图片和评论的数据模型，而 `Node` 的 `images` 成员正好是 `ImageItem` 的集合，这样我们就把对应关系设置好了。

注意，`Node` 类中小编并没有设置构造器，这是为后面的创建数据时使用方便（如果不设置类的构造器（方法），则默认会有一个参数为空的构造器，大家也可以显示地添加一个空的构造器）。

（3）定义项目布局

光标定位到 `res/layout`，右键选择“New|Layout resource file”，新建列表项目的布局文件 `item_image`：


```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:layout_margin="8dp"
    card_view:cardElevation="0dp"
    card_view:cardPreventCornerOverlap="false">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <ImageView
            android:id="@+id/photo"
            android:layout_width="139dp"
            android:layout_height="96dp"/>

        <TextView
            android:id="@+id/comment"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_below="@+id/photo"
            android:layout_centerInParent="true"
            android:gravity="center"/>
    </RelativeLayout>
</android.support.v7.widget.CardView>
```

(4) 创建Adapter

为 RecyclerView 创建适配器：光标定位在 `java/包名/ui/adpter`，右键选择“New|Java Class”，为新建的适配器取名 `ImageItemAdapter`，首先修改代码如下：

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

}
```

与 `ListView` 不同，这次我们让它继承 `RecyclerView.Adapter`，同时使用自定义的 `ViewHolder`。实现其方法（自动生成代码的方法同上）：

```
public class ImageItemAdapter extends RecyclerView.Adapter<ImageItemAdapter.ViewHolder> {
    //根据布局文件创建新视图
    @Override
    public ImageItemAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        return null;
    }
    //替换视图控件的内容
    @Override
    public void onBindViewHolder(ImageItemAdapter.ViewHolder holder, int position) {

    }
    //返回数据集包含项目的数量
    @Override
    public int getItemCount() {
        return 0;
    }
}
```

然后，再添加自定义的 `ViewHolder`：

```

public class ImageItemAdapter extends RecyclerView.Adapter<ImageItemAdapter.ViewHolder> {
    //根据布局文件创建新视图
    @Override
    public ImageItemAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        return null;
    }
    //替换视图控件的内容
    @Override
    public void onBindViewHolder(ImageItemAdapter.ViewHolder holder, int position) {

    }
    //返回数据集包含项目的数量
    @Override
    public int getItemCount() {
        return 0;
    }

    //自定义ViewHolder
    public static class ViewHolder extends RecyclerView.ViewHolder {
        public ViewHolder(View itemView) {
            super(itemView);
        }
    }
}

```

先把Adapter放一边，考虑如同提供数据。

(5) 改造外层列表

光标定位到 `java/包名/ui/adapter`，新建一个名为 `NodeAdapter` 的适配器，作为外层的竖向列表，把原来 `UserInfo` 中的代码全部粘贴到其中，然后修改如下：

```

public class NodeAdapter extends BaseAdapter {
    private Context mContext;
    private LayoutInflater mLayoutInflater;
    private List<Node> mNodes = new ArrayList<>();

    public NodeAdapter(Context context, List<Node> nodes) {
        mContext = context;
        mLayoutInflater = (LayoutInflater) mContext.
            getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        mNodes = nodes;
    }

    //获取列表项目数量
    @Override
    public int getCount() {
        return mNodes.size();
    }

    //获取列表指定项目
    @Override
    public Object getItem(int position) {
        return mNodes.get(position);
    }

    //获取列表项目位置
    @Override
    public long getItemId(int position) {
        return position;
    }

    //获取列表项目视图
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        //返回视图
        ViewHolder viewHolder;
        if (convertView == null) {
            convertView = mLayoutInflater.inflate(R.layout.item_node_square, null);
            viewHolder = new ViewHolder();
            //获取控件
            viewHolder.addressTextView = (TextView) convertView.findViewById(R.id.address_text);
            viewHolder.imageRecyclerView = (RecyclerView) convertView.findViewById(R.id.recycler_view);

            convertView.setTag(viewHolder);
        } else {
            viewHolder = (ViewHolder) convertView.getTag();
        }

        //绑定数据
        viewHolder.addressTextView.setText(mNodes.get(position).getAddress());
        // TODO: 填充内层列表

        return convertView;
    }

    //接收控件
    class ViewHolder {
        TextView addressTextView;
        RecyclerView imageRecyclerView;
    }
}

```

把传入的数据修改为 `mNodes`，读取的视图文件改为本节步骤（1）中定义的 `item_node_square` 布局。

对应地，修改 `SquareActivity` 中的代码如下：

```

public class SquareActivity extends AppCompatActivity {
    //声明成员
    private Spinner mSpinner;
    private ListView mListView;
    private List<Node> mNodes;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_square);

        //获取控件
        mSpinner = (Spinner) findViewById(R.id.distance_spinner_square);
        mListView = (ListView) findViewById(R.id.square_list_view);

        //创建填充用户信息的适配器对象
        NodeAdapter adapter = new NodeAdapter(SquareActivity.this, mNodes);

        //绑定数据
        mListView.setAdapter(adapter);
    }
}

```

外层列表每个节点的地址数据的填充方法已经设置好了，但是其他内容不能直接设置，因为返回的数据是一个列表，需要再用一个适配器来填充内层的列表，就是我们上面定义的 `ImageItemAdapter`。

（6）填充内层列表

把上一步代码中的 `TODO：填充内层列表` 替换为如下代码：

```

//绑定数据
viewHolder.addressTextView.setText(mNodes.get(position).getAddress());
//创建布局管理器
LinearLayoutManager layoutManager = new LinearLayoutManager(mContext);
//设置布局方向为水平
layoutManager.setOrientation(LinearLayoutManager.HORIZONTAL);
//关联布局
viewHolder.imageRecyclerView.setLayoutManager(layoutManager);

//创建数据适配器
ImageItemAdapter itemAdapter = new ImageItemAdapter(mNodes.get(position).getImages());
viewHolder.imageRecyclerView.setAdapter(itemAdapter);

```

使用 `RecyclerView` 时，需要定义一个布局管理器，我们把这个布局管理器设置为水平方向的线性布局。

然后给 `RecyclerView` 设置数据适配器，只需要传入 `mNodes` 中的 `images` 集合成员就可以了。

再次打开 `ImageItemAdapter` 适配器，现在来实现数据的绑定，首先创建一个 `ImageItem` 的集合成员，接收传入的数据：

```

public class ImageItemAdapter extends RecyclerView.Adapter<ImageItemAdapter.ViewHolder> {
    //创建接收数据的成员
    private List<ImageItem> mImageItems;
    //构造方法
    public ImageItemAdapter(List<ImageItem> imageItems) {
        mImageItems = imageItems;
    }
    .....
}

```

接着实现其填充方法：

```

//根据布局文件创建新视图
@Override
public ImageItemAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_image, parent,
        false);
    ViewHolder holder = new ViewHolder(view);
    return holder;
}

//替换视图控件的内容
@Override
public void onBindViewHolder(ImageItemAdapter.ViewHolder holder, int position) {
    holder.mPhoto.setImageResource(mImageItems.get(position).getImageId());
    holder.mComment.setText(mImageItems.get(position).getComment());
}

//返回数据集包含项目的数量
@Override
public int getItemCount() {
    return mImageItems == null ? 0 : mImageItems.size();
}

//自定义ViewHolder
public static class ViewHolder extends RecyclerView.ViewHolder {
    public ImageView mPhoto;
    public TextView mComment;

    public ViewHolder(View itemView) {
        super(itemView);
        mPhoto = (ImageView) itemView.findViewById(R.id.photo);
        mComment = (TextView) itemView.findViewById(R.id.comment);
    }
}

```

上面是参照官方样例编写的实现方法，`onCreateViewHolder()` 方法中读取 `item_image.xml` 返回列表项目的布局，然后使用 `ViewHolder` 来接收返回的视图。

这个 `ViewHolder` 是自定义的视图，因为 `RecyclerView` 本身有实现 `ViewHolder`，所以自定义的 `ViewHolder` 需要继承 `RecyclerView.ViewHolder`，然后通过资源id找到对应控件。

在 `onBindViewHolder()` 方法中实现数据的绑定，这几个方法和 `ListView` 中相似，可以对照着看。

(7) 初始化数据

回到 `SquareActivity`，我们创建一些数据，数据的初始化单独封装在 `initData()` 方法中

```

public class SquareActivity extends AppCompatActivity {
    //声明成员
    private Spinner mSpinner;
    private ListView mListView;
    private List<Node> mNodes;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_square);

        //获取控件
        mSpinner = (Spinner) findViewById(R.id.distance_spinner_square);
        mListView = (ListView) findViewById(R.id.square_list_view);

        //初始化数据
        initData();

        //创建填充用户信息的适配器对象
        NodeAdapter adapter = new NodeAdapter(SquareActivity.this, mNodes);

        //绑定数据
        mListView.setAdapter(adapter);
    }
}

```

//初始化数据

```
private void initData() {
    mNodes = new ArrayList<>();
    Node node = new Node();
    node.setAddress("地点一");
    List<ImageItem> imageItems = new ArrayList<>();
    imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
    imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
    imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
    imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
    imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
    imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
    imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
    imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
    imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
    node.setImages(imageItems);
    mNodes.add(node);

    node = new Node();
    node.setAddress("地点二");
    imageItems = new ArrayList<>();
    imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
    imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
    imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
    node.setImages(imageItems);
    mNodes.add(node);

    node = new Node();
    node.setAddress("地点三");
    imageItems = new ArrayList<>();
    imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
    imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
    imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
    imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
    imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
    imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
    imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
    node.setImages(imageItems);
    mNodes.add(node);

    node = new Node();
    node.setAddress("地点四");
    imageItems = new ArrayList<>();
    imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
    imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
    imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
    imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
    node.setImages(imageItems);
    mNodes.add(node);

    node = new Node();
    node.setAddress("地点五");
    imageItems = new ArrayList<>();
    imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
    imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
    imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
    imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
    imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
    imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
    imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
    imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
    imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
    imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
    imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
    imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
    node.setImages(imageItems);
    mNodes.add(node);

    node = new Node();
    node.setAddress("地点六");
    imageItems = new ArrayList<>();
    imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
```

```

imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
node.setImages(imageItems);
mNodes.add(node);

node = new Node();
node.setAddress("地点七");
imageItems = new ArrayList<>();
imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
node.setImages(imageItems);
mNodes.add(node);

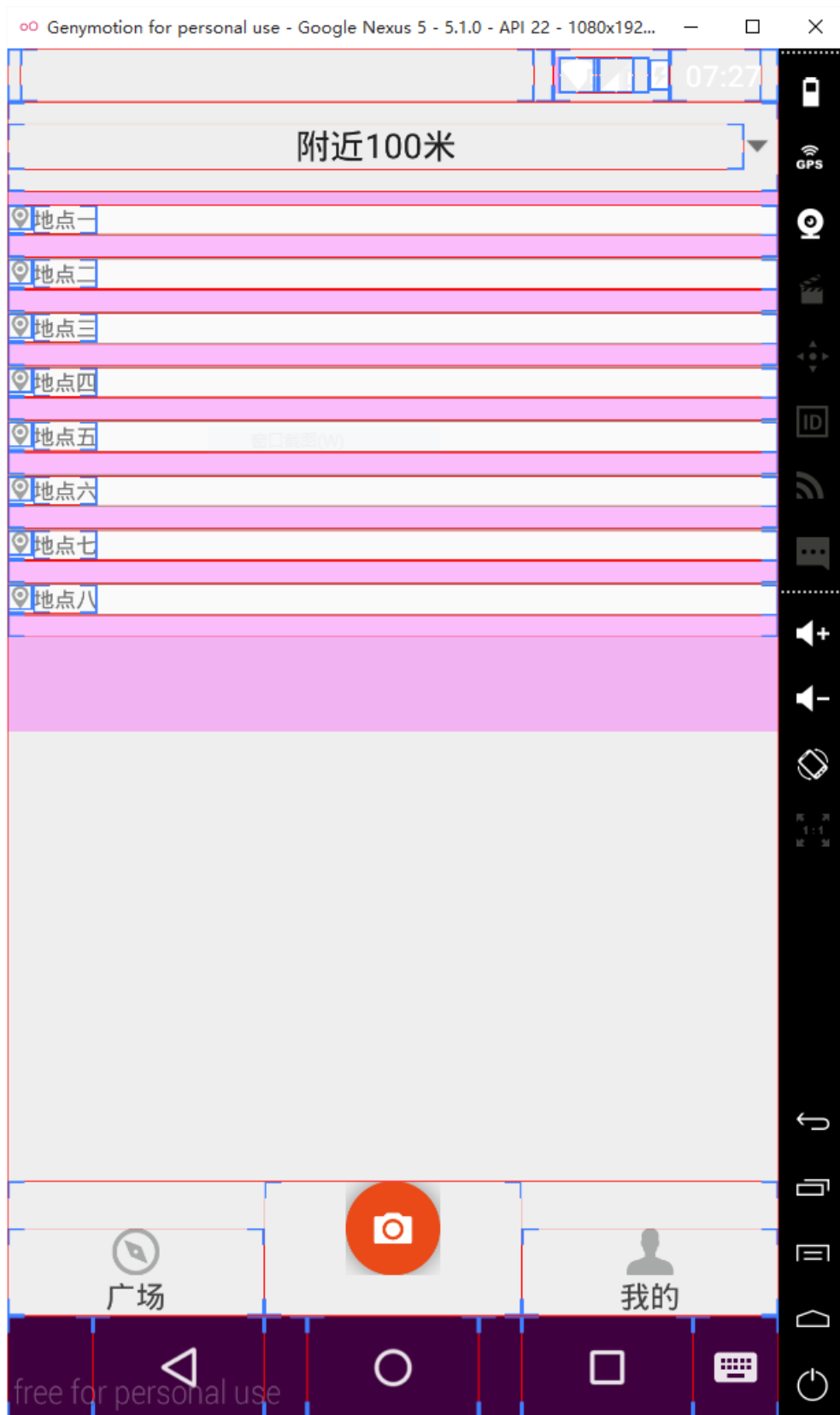
node = new Node();
node.setAddress("地点八");
imageItems = new ArrayList<>();
imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
imageItems.add(new ImageItem(R.drawable.steak, "真好吃!"));
imageItems.add(new ImageItem(R.drawable.crabs, "好吃你就多吃点!"));
imageItems.add(new ImageItem(R.drawable.ribs, "不能浪费!"));
node.setImages(imageItems);
mNodes.add(node);
}
}

```

小编自己找了三张图片，每张的像素大概在 200*100 左右，放在 `res/drawable` 目录下，名字分别为 `steak`、`crabs`、`ribs`，大家可以自己找一些图片，取自己喜欢的名字，添加上评论内容。

首先创建一个列表集合，集合中每个元素就是我们自己定义的 `Node` 对象，它包含一个地点成员以及一个 `ImageItem` 元素的列表集合，先把 `ImageItem` 的集合添加到每一个节点对象中，再把节点对象都添加进 `mNode` 对象中，`mNode` 最终构成我们的数据源。

好，现在把程序部署到模拟器上，小编运行之后会发现结果是这样：



打开绘图边界，看不到 `RecyclerView`，小编最初以为里层列表没有读取出来，反复找了好长时间代码有没有问题。后来发现，有可能是 `RecyclerView` 的高度为0，所以看起来好像没有内容。

打开里层列表的项目布局文件 `item_node_square.xml`，给 `RecyclerView` 添加两个属性：

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/recycler_view"
    android:minWidth="200dp"
    android:minHeight="120dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="10dp"
    android:scrollbars="horizontal"/>
```


添加两行代码:`android:minWidth="200dp"` 和 `android:minHeight="120dp"` , 给 `RecyclerView` 设置最小宽度和最小高度, 再次运行, 神奇的事情发生了, 图片和评论都正常显示出来了:



单独使用 `RecyclerView` 时并没有这个问题, 但是嵌套后就出现了。 `RecyclerView` 的宽高是由本节步骤 (6) 中设置的 `LinearLayoutManager` 控制的, 那么回到 `NodeAdapter` 中, 我们自定义一个 `LinearLayoutManager` :

```

public class NodeAdapter extends BaseAdapter {
    ....
    //获取列表项目视图
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        //返回视图
        ViewHolder viewHolder;
        if (convertView == null) {
            convertView = mLayoutInflater.inflate(R.layout.item_node_square, null);
            viewHolder = new ViewHolder();
            //获取控件
            viewHolder.addressTextView = (TextView) convertView.findViewById(R.id.address_text);
            viewHolder.imageRecyclerView = (RecyclerView) convertView.findViewById(R.id.recycler_view);

            convertView.setTag(viewHolder);
        } else {
            viewHolder = (ViewHolder) convertView.getTag();
        }

        //绑定数据
        viewHolder.addressTextView.setText(mNodes.get(position).getAddress());
        //创建布局管理器
        MyLayoutManager layoutManager = new MyLayoutManager(mContext);
        //设置布局方向为水平
        layoutManager.setOrientation(LinearLayoutManager.HORIZONTAL);
        //关联布局
        viewHolder.imageRecyclerView.setLayoutManager(layoutManager);

        //创建数据适配器
        ImageItemAdapter itemAdapter = new ImageItemAdapter(mNodes.get(position).getImages());
        viewHolder.imageRecyclerView.setAdapter(itemAdapter);

        return convertView;
    }

    //接收控件
    class ViewHolder {
        TextView addressTextView;
        RecyclerView imageRecyclerView;
    }

    //自定义视图管理器
    class MyLayoutManager extends LinearLayoutManager{

        public MyLayoutManager(Context context) {
            super(context);
        }

        @Override
        public void onMeasure(RecyclerView.Recycler recycler,
                             RecyclerView.State state, int widthSpec, int heightSpec) {
            View view = recycler.getViewForPosition(0);
            if (view != null) {
                measureChild(view, widthSpec, heightSpec);
                int measuredWidth = View.MeasureSpec.getSize(widthSpec);
                int measuredHeight = view.getMeasuredHeight();
                setMeasuredDimension(measuredWidth, measuredHeight);
            }
        }
    }
}

```

先把 `android:minWidth="200dp"` 和 `android:minHeight="120dp"` 代码去掉，观察一下运行结果。

然后在 `NodeAdapter` 中自定义 `MyLayoutManager` 视图管理器，让它集成 `LinearLayoutManager`，然后管理布局文件时，就不是创建 `LinearLayoutManager` 了，而是 `MyLayoutManager layoutManager = new MyLayoutManager(mContext);`。

再次运行，得到我们想要的结果。

4.4 异步加载

现在我们来请求网络资源，通过阅读API文档，我们知道应该要使用 获取地理位置列表 里的请求参数，其中 `user_id` 以及 `token` 是其他模块也需要的，在登录之后可以直接把值存起来。

(1) 保存验证信息

首先通过登录来获得 `user_id` 和 `token`，我们先在应用程序清单 `AndroidManifest.xml` 中把登录界面 `SignInActivity` 设为启动项：

```
<application >
.....
<activity
    android:name=".ui.activity.SignInActivity"
    android:label="@string/title_activity_sign_in"
    android:theme="@style/AppTheme.NoActionBar" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
.....
</application>
```

完成登录模块的网络编程部分，其他部分代码大家可以参照注册模块改写：

```
.....
protected void onCreate() {
.....
    //设置点击事件监听器
    mSignIn.setOnClickListener(listener);
    mSignUp.setOnClickListener(listener);
    //获取对全局变量的引用
    mAppContext = (ApplicationContext) getApplication();
}
```

在 `onCreate` 方法中给两个按钮传入相同的监听器，监听器的定义如下：

```
.....
private Button mSignIn;
private Button mSignUp;
//使用全局变量
private ApplicationContext mAppContext;
private static final String mPath = "/user/login";

private View.OnClickListener listener = new View.OnClickListener() {
    @Override
    public void onClick(View v) { //传入被点击对象
        switch (v.getId()) { //获取该对象的Id
            case R.id.sign_in_button:
                SignIn(); //尝试登录
                break;
            case R.id.sign_up_button:
                SignUp(); //转到注册
                break;
            default:
                break;
        }
    }
};
```

其中 `SignUp()` 为转到注册页面：

```
private View.OnClickListener listener = new View.OnClickListener() {  
    .....  
};  
private void SignUp() {  
    Intent intent = new Intent(SignInActivity.this, SignUpActivity.class);  
    startActivity(intent);  
}
```

在 `SignIn()` 中完成本地验证后向服务器提交请求:

```

private void SignUp() {}
public void SignIn() {
    //重置错误提示
    mEmail.setError(null);
    mPassword.setError(null);

    //获取登录数据
    final String email = mEmail.getText().toString().trim();
    final String password = mPassword.getText().toString().trim();

    //初始化获取焦点的控件
    boolean isValid = true;
    View focusView = null;

    //密码验证
    if (TextUtils.isEmpty(password)) {
        mPassword.setError(getString(R.string.error_empty_password));
        focusView = mPassword;
        isValid = false;
    } else if (StringUtil.isPasswordValid(password) == false) {
        mPassword.setError(getString(R.string.error_length_password));
        focusView = mPassword;
        isValid = false;
    }

    //邮箱验证
    if (TextUtils.isEmpty(email)) {
        mEmail.setError(getString(R.string.error_empty_email));
        focusView = mEmail;
        isValid = false;
    } else if (StringUtil.isEmail(email) == false) {
        mEmail.setError(getString(R.string.error_pattern_email));
        focusView = mEmail;
        isValid = false;
    }

    if (isValid == false) {
        focusView.requestFocus();
    } else if (NetworkStatus.isNetworkConnected(getApplicationContext())) {

        new Thread() {
            @Override
            public void run() {
                try {
                    //获取密码的md5值，并截取前20个字符
                    String pwd = StringUtil.getMD5(password).substring(0, 20);
                    String gbid = "GeekBand-A150010"; //换为自己的学号，学号前面没有"GeekBand"前缀
                    JSONObject user = new JSONObject();
                    user.put("email", email);
                    user.put("password", pwd);
                    user.put("gbid", gbid);

                    String url = mAppContext.getUrl(mPath);
                    doPostRequest(url, user);
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
        }.start();

    } else {

        Message msg = Message.obtain();
        msg.what = UNCONNECTED;
        msg.obj = getString(R.string.unavailable_network_connection);
        mHandler.sendMessage(msg);
    }
}

```

提交服务器前先使用之前封装的 `isNetworkConnected` 方法检查网络连接状态，若为连接状态，则打开一个新线程，把我们要提交的数据传递给封装的请求方法中：

```

public void SignIn() {}
private void doPostRequest(String path, JSONObject data) {
    try {
        //取得输出实体,请求时以字节流的方式传输
        byte[] entity = data.toString().getBytes("UTF-8");
        URL url = new URL(path);
        //实例化一个HTTP连接对象
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setConnectTimeout(5000); //定义超时时间,最好不要超过10秒
        connection.setRequestMethod("POST"); //使用POST方式发送请求
        connection.setUseCaches(false); //设置不允许使用缓存
        connection.setDoOutput(true); //允许对外输出
        connection.setDoInput(true); //允许对内输入
        connection.setRequestProperty("Content-Type", "application/json"); //提交格式为json
        connection.setRequestProperty("Content-Length", String.valueOf(entity.length));
        OutputStream outputStream = connection.getOutputStream(); //获得输出流
        outputStream.write(entity);
        outputStream.close();

        int responseCode = connection.getResponseCode();
        if (responseCode == 200) {
            // 处理成功的请求
            InputStream inputStream = connection.getInputStream();
            byte[] is = StreamUtil.readInputStream(inputStream);
            String json = new String(is);

            Message msg = Message.obtain();
            msg.what = SUCCESS;
            msg.obj = json;
            mHandler.sendMessage(msg);
        } else {
            // 处理失败的请求
            // 添加处理逻辑
            Message msg = Message.obtain();
            msg.what = ERROR;
            mHandler.sendMessage(msg);
        }
    } catch (UnsupportedEncodingException e) { //编码方面的异常
        e.printStackTrace();
    } catch (MalformedURLException e) { //路径方面的异常
        e.printStackTrace();
    } catch (ProtocolException e) { //协议方面的异常
        e.printStackTrace();
    } catch (IOException e) { //输入输出方面的异常
        e.printStackTrace();
    } catch (Exception e) { //其他方面的异常
        e.printStackTrace();
    }
}
}

```

之前也说过,由于服务器端对无效请求不能返回数据,所以无效的请求不做相应处理。

如果登录成功,我们先显示一下全部的数据,然后再对其进行解析。

而由于当前操作是在子线程中进行,要在页面中显示提示信息必须要在主线程中执行,所以我们使用 `Handler` 技术,其定义如下:

```

private static final int UNCONNECTED = -1;
private static final int SUCCESS = 1;
private static final int ERROR = 0;
//定义本类成员变量
private autoCompleteTextView mEmail;
private EditText mPassword;
private Button mSignIn;
private Button mSignUp;
//使用全局变量
private ApplicationContext mAppContext;
private static final String mPath = "/user/login";

private Handler mHandler;
{
    mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case UNCONNECTED:
                    Toast.makeText(getApplicationContext(), msg.obj.toString(), Toast
                        .LENGTH_SHORT).show();
                    break;
                case SUCCESS:
                    Toast.makeText(getApplicationContext(), msg.obj.toString(), Toast
                        .LENGTH_LONG).show();
                    break;
                case ERROR:
                    Toast.makeText(getApplicationContext(), "用户名或密码错误", Toast.LENGTH_LONG)
                        .show();
                    break;
                default:
                    break;
            }
        }
    };
};

```

好，现在把应用部署到模拟器上运行一下，如果上面的代码遗漏的地方，请大家参考之前的登录和注册模块补齐。
登录成功后，页面显示类似下面的信息：

```

{
  "status": 1,
  "data": {
    "user_id": "2",
    "user_name": "testProject3",
    "token": "cec5a0a5e7a1951c87099b96d16851bf6f9f0a21",
    "avatar": "",
    "project_id": "1",
    "last_login": "2015-08-30 17:35:49",
    "login_times": 3
  },
  "message": "Login success"
}

```

我们需要把 `user_id` 和 `token` 中的数据保存起来。

首先解析出这两个变量，修改 `doPostRequest` 中对应代码如下：


```

if (responseCode == 200) {
    // 处理成功的请求
    InputStream inputStream = connection.getInputStream();
    byte[] is = StreamUtil.readInputStream(inputStream);
    String json = new String(is);

    //解析返回的JSON对象
    JSONObject jsonObject = new JSONObject(json);
    JSONObject myData = jsonObject.getJSONObject("data");
    int myId = myData.getInt("user_id");
    String myToken = myData.getString("token");

    Message msg = Message.obtain();
    msg.what = SUCCESS;
    mHandler.sendMessage(msg);
}

```

因为我们的数据是被嵌套在里层的 `JSON` 对象中，所以创建了两个 `JSONObject`，使用 `JSONObject` 的 `getInt()` 和 `getString()` 方法，通过参数名称获取对应的值。

得到这两个变量后，把它们存到哪儿呢？

这两个变量在其他模块也需要被用到，所以应该定义为全局的变量。还记得之前介绍过的 `SharedPreferences` 工具吗，我们可以把这两个变量加密后存到 `SharedPreferences` 中（小编后来查了一下，在 `Application` 组件中存储数据的方式并不可取）。

我们先来获取 `SharedPreferences` 接口，并通过 `SharedPreferences.Editor` 对象编辑数据。

```

//解析返回的JSON对象
JSONObject jsonObject = new JSONObject(json);
JSONObject myData = jsonObject.getJSONObject("data");
int myId = myData.getInt("user_id");
String myToken = myData.getString("token");

//获取SharedPreferences接口
SharedPreferences sp = getSharedPreferences("moran", MODE_PRIVATE);
//获取编辑器对象
SharedPreferences.Editor editor = sp.edit();
.....

```

`user_id` 和 `token` 保存时不应该直接使用明文，我们需要将其加密后再存储，使用时再解密出来。由于前面介绍的 `MD5` 算法不能被解密，所以小编又找了一个 `AES` 加密解密工具类。

光标定位到 `java/包名/util`，右键新建一个类，命名为 `AESEncryptor`，其代码如下：

```

/**
 * AES加密器
 * @author Eric_Ni
 *
 */
public class AESEncryptor {
    /**
     * AES加密
     */
    public static String encrypt(String seed, String cleartext) throws Exception {
        byte[] rawKey = getRawKey(seed.getBytes());
        byte[] result = encrypt(rawKey, cleartext.getBytes());
        return toHex(result);
    }

    /**
     * AES解密
     */
    public static String decrypt(String seed, String encrypted) throws Exception {
        byte[] rawKey = getRawKey(seed.getBytes());
        byte[] enc = toByte(encrypted);
        byte[] result = decrypt(rawKey, enc);
        return new String(result);
    }
}

```



```

private static byte[] getRawKey(byte[] seed) throws Exception {
    KeyGenerator kgen = KeyGenerator.getInstance("AES");
    SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
    sr.setSeed(seed);
    kgen.init(128, sr); // 192 and 256 bits may not be available
    SecretKey skey = kgen.generateKey();
    byte[] raw = skey.getEncoded();
    return raw;
}

private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
    SecretKeySpec keySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, keySpec);
    byte[] encrypted = cipher.doFinal(clear);
    return encrypted;
}

private static byte[] decrypt(byte[] raw, byte[] encrypted) throws Exception {
    SecretKeySpec keySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.DECRYPT_MODE, keySpec);
    byte[] decrypted = cipher.doFinal(encrypted);
    return decrypted;
}

public static String toHex(String txt) {
    return toHex(txt.getBytes());
}

public static String fromHex(String hex) {
    return new String(toByte(hex));
}

public static byte[] toByte(String hexString) {
    int len = hexString.length()/2;
    byte[] result = new byte[len];
    for (int i = 0; i < len; i++)
        result[i] = Integer.valueOf(hexString.substring(2*i, 2*i+2), 16).byteValue();
    return result;
}

public static String toHex(byte[] buf) {
    if (buf == null)
        return "";
    StringBuffer result = new StringBuffer(2*buf.length);
    for (int i = 0; i < buf.length; i++) {
        appendHex(result, buf[i]);
    }
    return result.toString();
}

private final static String HEX = "0123456789ABCDEF";
private static void appendHex(StringBuffer sb, byte b) {
    sb.append(HEX.charAt((b>>4)&0x0f)).append(HEX.charAt(b&0x0f));
}
}

```

先来使用加密方法，保存数据：

```

//获取SharedPreferences接口
SharedPreferences sp = getSharedPreferences("moran", MODE_PRIVATE);
//获取编辑器对象
SharedPreferences.Editor editor = sp.edit();
editor.putString("user_id", AESEncryptor.encrypt(getString(R.string.random_seed),
    String.valueOf(myId)));
editor.putString("token", AESEncryptor.encrypt(getString(R.string.random_seed),
    myToken));
editor.commit(); //提交保存

```

加密方法 `encrypt()` 中第一个参数是一个字符串类型的随机数种子，解密时需要填入相同的随机数种子，大家可以填写任意字符串，比如小编这里在 `res/values/strings.xml` 中定义的是：

```
<resources>
.....
<string name="random_seed">geekband</string>
</resources>
```

由于 `AESCryptor` 的加密方法只提供了加密字符串的方法，只能传入字符串参数，所以需要把得到的 `user_id` 转换成字符串。

最后，不要忘了执行 `commit()` 方法，执行这一句代码时，数据才真正写入到 `SharedPreferences` 文件中。

现在，修改 `Handler` 中请求成功的逻辑，如果登录成功，则跳转到广场界面：

```
mHandler = new Handler() {
.....
case SUCCESS:
    Intent intent = new Intent(SignInActivity.this, SquareActivity.class);
    startActivity(intent);
    break;
.....
};
```

部署到模拟器运行一下吧，如果登录成功，则可以按注册模块的 Lab Manual 中的方式，查看保存的内容。

(2) 读取保存内容

我们先来解密出保存的内容，看看是否有问题，回到 `SquareActivity`，首先获取 `SharedPreferences` 接口：

```
.....
private List<Node> mNodes;
//声明SharedPreferences
private SharedPreferences mSharedPref;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_square);

    //获取控件
    mSpinner = (Spinner) findViewById(R.id.distance_spinner_square);
    mListView = (ListView) findViewById(R.id.square_list_view);

    //获取SharedPreferences接口
    mSharedPref = getSharedPreferences("moran", MODE_PRIVATE);
    .....
}
```

使用 `getString()` 方法，通过参数名称取得保存的对应值：

```
//获取SharedPreferences接口
mSharedPref = getSharedPreferences("moran", MODE_PRIVATE);
String user_id = mSharedPref.getString("user_id", "");
String token = mSharedPref.getString("token", "");
```

为处理方便，`user_id` 使用的是 `String` 类型。

尝试解密：

```

.....
String token = mSharedPreferences.getString("token", "");

try {
    user_id = AESEncryptor.decrypt(getString(R.string.random_seed), user_id);
} catch (Exception e) {
    e.printStackTrace();
}

try {
    token = AESEncryptor.decrypt(getString(R.string.random_seed), token);
} catch (Exception e) {
    e.printStackTrace();
}

```

使用相同的 随机数种子，得到解密后的字符串，先把这些字符串在界面上显示出来：

```

.....
try {
    token = AESEncryptor.decrypt(getString(R.string.random_seed), token);
} catch (Exception e) {
    e.printStackTrace();
}

Toast.makeText(SquareActivity.this, user_id + " , " + token, Toast.LENGTH_LONG)
    .show();

//初始化数据
initData();

```

在模拟器上运行，输入用户名和密码，观察界面上显示出的信息。

如果一切顺利的话，那么在逗号前显示的应该是看起来像整数的一串数字，而不应该是一长串意义不明的字符。

但小编的模拟器返回的就是这样一长串字符，查阅相关资料发现，原来加密工具类中的SHA1PRNG强随机种子算法，在Android 4.2以上版本中调用方法稍有不同，回到 AESEncryptor 工具类中，找到 getRawKey() 方法，修改如下：

```

private static byte[] getRawKey(byte[] seed) throws Exception {
    KeyGenerator kgen = KeyGenerator.getInstance("AES");
    SecureRandom sr = null;
    if( Build.VERSION.SDK_INT >= 17 ){
        sr = SecureRandom.getInstance("SHA1PRNG","Crypto");
    }
    else {
        sr = SecureRandom.getInstance("SHA1PRNG");
    }
    sr.setSeed(seed);
    kgen.init(128, sr); // 192 and 256 bits may not be available
    SecretKey skey = kgen.generateKey();
    byte[] raw = skey.getEncoded();
    return raw;
}

```

即修改原来这一行代码 `SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");`，Android 4.2对应的SDK版本号为17，所以通过上述逻辑，区分不同系列版本的算法调用方式。

再次在模拟器上运行，小编的模拟器上已经可以成功显示出解密后的内容了。

现在，可以多次返回尝试登录，可以发现，token 的内容每次都不一样，只要重新登录后，服务器端就会返回新的 token，token 的验证机制还是比较安全的。

鉴于这两项信息会被频繁调用，我们把读取的逻辑简单封装在一个业务类中（大家也可以封装为更通用的工具类），光标定位到 java/包名，右键选择“New|Package”，输入子包名为“engine”（这里命名是为了与“service”作区分），添加代码如下：

```

public class TokenEngine {

    /**
     * 读取验证信息，并解密
     * @param context
     * @return
     */
    public static TokenInfo getTokenInfo(Context context) {

        SharedPreferences sp = context.getSharedPreferences("moran", Context.MODE_PRIVATE);
        String user_id = sp.getString("user_id", "");
        String token = sp.getString("token", "");

        try {
            user_id = AESEncryptor.decrypt(context.getString(R.string.random_seed), user_id);
        } catch (Exception e) {
            e.printStackTrace();
        }

        try {
            token = AESEncryptor.decrypt(context.getString(R.string.random_seed), token);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return new TokenInfo(Integer.parseInt(user_id), token);
    }

    /**
     * 构造验证信息对象
     */
    public static class TokenInfo {
        public int user_id;
        public String token;

        public TokenInfo(int user_id, String token) {
            this.user_id = user_id;
            this.token = token;
        }
    }
}

```

把之前的代码粘贴进来，稍作改动，方法和对象模型都使用 `static` 修饰，这样就不用实例化外部对象。

同样地，尝试使用一下，把 `SquareActivity` 中的代码修改如下：

```

public class SquareActivity extends AppCompatActivity {
    //声明成员
    private Spinner mSpinner;
    private ListView mListView;
    private List<Node> mNodes;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_square);

        //获取控件
        mSpinner = (Spinner) findViewById(R.id.distance_spinner_square);
        mListView = (ListView) findViewById(R.id.square_list_view);

        //获取上一次登录保存的验证信息
        TokenEngine.TokenInfo tokenInfo = TokenEngine.getTokenInfo(SquareActivity.this);

        Toast.makeText(SquareActivity.this, String.valueOf(tokenInfo.user_id) + " , " + tokenInfo.token, Toast.LENGTH_LONG).show();

        //初始化数据
        initData();
        .....
    }
}

```

部署到模拟器，输入用户名和密码，查看显示信息是否正确。

如果显示的 `user_id` 显示的是一个数字，那么我们封装的业务类就能正常读取和解密验证信息了。

接下来，把 `SquareActivity` 中的代码整理如下：

```
public class SquareActivity extends AppCompatActivity {
    //声明成员
    private Spinner mSpinner;
    private ListView mListView;
    private List<Node> mNodes;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_square);

        //获取控件
        mSpinner = (Spinner) findViewById(R.id.distance_spinner_square);
        mListView = (ListView) findViewById(R.id.square_list_view);

        //获取上次登录保存的验证信息
        TokenEngine.TokenInfo tokenInfo = TokenEngine.getTokenInfo(SquareActivity.this);

        //初始化数据
        initData();

        //创建填充用户信息的适配器对象
        NodeAdapter adapter = new NodeAdapter(SquareActivity.this, mNodes);

        //绑定数据
        mListView.setAdapter(adapter);
    }
    //初始化数据
    private void initData() {.....}
}
```

之前封装的 `初始化数据` 方法是本地测试用的，现在可以把它删掉了，然后再把需要用到的网络路径引用进来，代码如下：

```

public class SquareActivity extends AppCompatActivity {
    //声明成员
    private Spinner mSpinner;
    private ListView mListView;
    private List<Node> mNodes;

    private ApplicationContext mAppContext;
    private String mPath = "/node/list";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_square);

        //获取控件
        mSpinner = (Spinner) findViewById(R.id.distance_spinner_square);
        mListView = (ListView) findViewById(R.id.square_list_view);

        //获取上次登录保存的验证信息
        TokenEngine.TokenInfo tokenInfo = TokenEngine.getTokenInfo(SquareActivity.this);

        //获取网络路径
        mAppContext = (ApplicationContext) getApplication();
        String path = mAppContext.getUrl(mPath);

        // TODO: 请求网络资源

        //创建填充用户信息的适配器对象
        NodeAdapter adapter = new NodeAdapter(SquareActivity.this, mNodes);

        //绑定数据
        mListView.setAdapter(adapter);
    }
}

```

整理好之后，我们进入下一个步骤。

(3) 请求网络资源

每次发送网络请求时，首先检查网络连接状态，在上一步中 <TODO: 请求网络资源> 部分的添加代码如下：

```

// 请求网络资源
if (NetworkStatus.isNetworkConnected(SquareActivity.this)) {

} else {
    Toast.makeText(SquareActivity.this, R.string.unavailable_network_connection, Toast
        .LENGTH_SHORT).show();
}

```

网络操作要放在非UI线程中进行，所以新开一个线程：

```

// 请求网络资源
if (NetworkStatus.isNetworkConnected(SquareActivity.this)) {
    new Thread() {
        @Override
        public void run() {
            //初始化数据
            NodeEngine.getNodes(path, tokenInfo);
        }
    }.start();
} else {
    Toast.makeText(SquareActivity.this, R.string.unavailable_network_connection, Toast
        .LENGTH_SHORT).show();
}

```

上面的代码在新开线程时，使用的是匿名内部类的方法，在这个内部类里面还有一个 `getNodes` 方法，它的两个参数是外部变量，一个内部类里要访问外部变量，要么变量是全局的，要么变量声明时加上 `final` 关键字，这样变量的值就不可更改，在下面两行代码前加上 `final` 修饰：

```
//获取上次登录保存的验证信息
final TokenEngine.TokenInfo tokenInfo = TokenEngine.getTokenInfo(SquareActivity.this);
.....
final String path = mAppContext.getUrl(mPath);
```

然后，后面绑定数据的代码先注释掉，小编带大家把前期的试探过程也了解一下：

```
//创建填充用户信息的适配器对象
//NodeAdapter adapter = new NodeAdapter(SquareActivity.this, mNodes);

//绑定数据
//mListView.setAdapter(adapter);
```

之前的网络访问都是直接写在 `Activity` 中，现在我们把操作单独封装在一个业务类里面，观察上面的 `初始化数据` 代码：

```
// 初始化数据
NodeEngine.getNodes(path, tokenInfo);
```

也就是说会有一个 `NodeEngine` 类，它的 `getNodes` 方法后面我们会用来返回给 `mNodes` 对象相应数据，返回数据类型为 `List<Node>`。

光标定位到 `java/包名/engine` 目录，右键选择 "New|Java Class"，新建一个类，取名为 `NodeEngine`，在其中添加一个静态方法 `getNodes`：

```
public class NodeEngine {

    public static List<Node> getNodes(String path) {
        return null;
    }

}
```

查看API文档，现在需要使用 `获取地理位置列表` 中提交的参数，我们需要以 `GET` 方式提交请求，提交的参数有 `token`、`user_id`、`distance`、`longitude`、`latitude`。

使用 `GET` 方式提交数据，我们需要把参数以键值对的形式拼接到 `URL` 中，即最后的网络路径需要拼接为如下形式：`http://moran.chinacloudapp.cn/moran/web/node/list?token={0}&user_id={1}&distance={2}&logitude{3}&latitude={4}`。

请求路径后加上一个 `?`，把请求路径和参数隔开，每个参数以 `Key=Value` 的形式表示，参数之间使用 `&` 符号连接。

在 `getNodes` 方法中添加代码如下：

```
public static List<Node> getNodes(String path, TokenEngine.TokenInfo tokenInfo) {

    Map<String,String> params = new HashMap<String,String>();

    return null;

}
```

新建一个 `Map` 对象，用来封装我们的参数，为了处理方便，参数的键和值均定义为 `String` 类型的，所以我们回过头去修改 `TokenEngine` 中的代码，把 `user_id` 的类型改为 `String`，解密后也不需要再转换成整数了：

```

public class TokenEngine {

    /**
     * 读取验证信息，并解密
     *
     * @param context
     * @return
     */
    public static TokenInfo getTokenInfo(Context context) {

        SharedPreferences sp = context.getSharedPreferences("moran", Context.MODE_PRIVATE);
        String user_id = sp.getString("user_id", "");
        String token = sp.getString("token", "");

        try {
            user_id = AESEncryptor.decrypt(context.getString(R.string.random_seed), user_id);
        } catch (Exception e) {
            e.printStackTrace();
        }

        try {
            token = AESEncryptor.decrypt(context.getString(R.string.random_seed), token);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return new TokenInfo(user_id, token);
    }

    /**
     * 构造验证信息对象
     */
    public static class TokenInfo {
        public String user_id;
        public String token;

        public TokenInfo(String user_id, String token) {
            this.user_id = user_id;
            this.token = token;
        }
    }
}

```

再回到 `NodeEngine`，继续封装提交的参数：

```

public class NodeEngine {

    public static List<Node> getNodes(String path, TokenEngine.TokenInfo tokenInfo) {
        Map<String, String> params = new HashMap<String, String>();
        params.put("token", tokenInfo.token);
        params.put("user_id", tokenInfo.user_id);
        params.put("distance", "500");
        params.put("longitude", "121.47749");
        params.put("latitude", "31.22516");

        return null;
    }
}

```

(4) 发送GET请求

接下来，就要发送 `GET` 请求了，把处理逻辑单独封装到一个 `doGETRequest` 方法中，修改 `getNodes` 的返回值：


```

public static List<Node> getNodes(String path, TokenEngine.TokenInfo tokenInfo) {
    Map<String, String> params = new HashMap<String, String>();
    params.put("token", tokenInfo.token);
    params.put("user_id", tokenInfo.user_id);
    params.put("distance", "500");
    params.put("longitude", "121.47749");
    params.put("latitude", "31.22516");

    return doGETRequest(path, params);
}

```

新建 `doGETRequest` 方法：

```

public static List<Node> getNodes(String path, TokenEngine.TokenInfo tokenInfo) {
    .....
    return doGETRequest(path, params);
}

private static List<Node> doGETRequest(String path, Map<String, String> params) {
    return null;
}

```

把请求路径拼接起来：

```

private static List<Node> doGETRequest(String path, Map<String, String> params) {
    StringBuilder urlBuilder = new StringBuilder(path);
    urlBuilder.append("?");
    for (Map.Entry<String, String> entry : params.entrySet()) {
        urlBuilder.append(entry.getKey()).append("=");
        urlBuilder.append(entry.getValue()).append("&");
    }
    urlBuilder.deleteCharAt(urlBuilder.length() - 1);
    return null;
}

```

新建一个 `StringBuilder`，利用 `append` 方法在路径后面附加字符串，先附加一个问号，然后参数集合的附加使用一个循环（小编将其理解为 `Java` 中的 `foreach` 循环），然后按照索引删除最后多出来的 `&` 符号。

使用最后的请求路径创建 `URL` 对象：

```

private static List<Node> doGETRequest(String path, Map<String, String> params) {
    .....
    urlBuilder.deleteCharAt(urlBuilder.length() - 1);
    URL url = new URL(urlBuilder.toString());
    return null;
}

```

这时候编译器会报一个网络路径方面的异常，通过之前学习其他模块的网络请求，我们知道，在进行网络访问时，有可能产生多种异常情况，所以我们先让方法对外声明抛出异常，在方法名后面添加 `throws Exception`：

```

private static List<Node> doGETRequest(String path, Map<String, String> params) throws Exception {
    .....
    URL url = new URL(urlBuilder.toString());
    return null;
}

```

然后，在调用该方法的时候，捕获这个异常，把 `getNodes` 中调用时的代码 `return doGETRequest(path, params);` 修改如下：

```

public static List<Node> getNodes(String path, TokenEngine.TokenInfo tokenInfo) {
    .....
    try {
        return doGETRequest(path, params);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}

```

继续完成GET请求：

```

private static List<Node> doGETRequest(String path, Map<String, String> params) throws Exception {
    .....
    URL url = new URL(urlBuilder.toString());
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setConnectTimeout(5000);
    conn.setRequestMethod("GET");
    if (conn.getResponseCode() == 200) {
        //处理成功的请求
        return parseJSON(conn.getInputStream());
    } else {
        // TODO: 处理失败的请求
    }

    return null;
}

```

如果请求成功，则需要解析返回的数据，我们把解析的操作也单独封装在一个方法中，不过，我们可以先在查看一下返回的数据，把解析方法先注释掉，然后在日志中打印出请求路径：

```

Log.i(TAG, String.valueOf(url));
if (conn.getResponseCode() == 200) {
    //处理成功的请求
    //return parseJSON(conn.getInputStream());
} else {
    // TODO: 处理失败的请求
}

```

其中，TAG 为创建的静态标签：

```

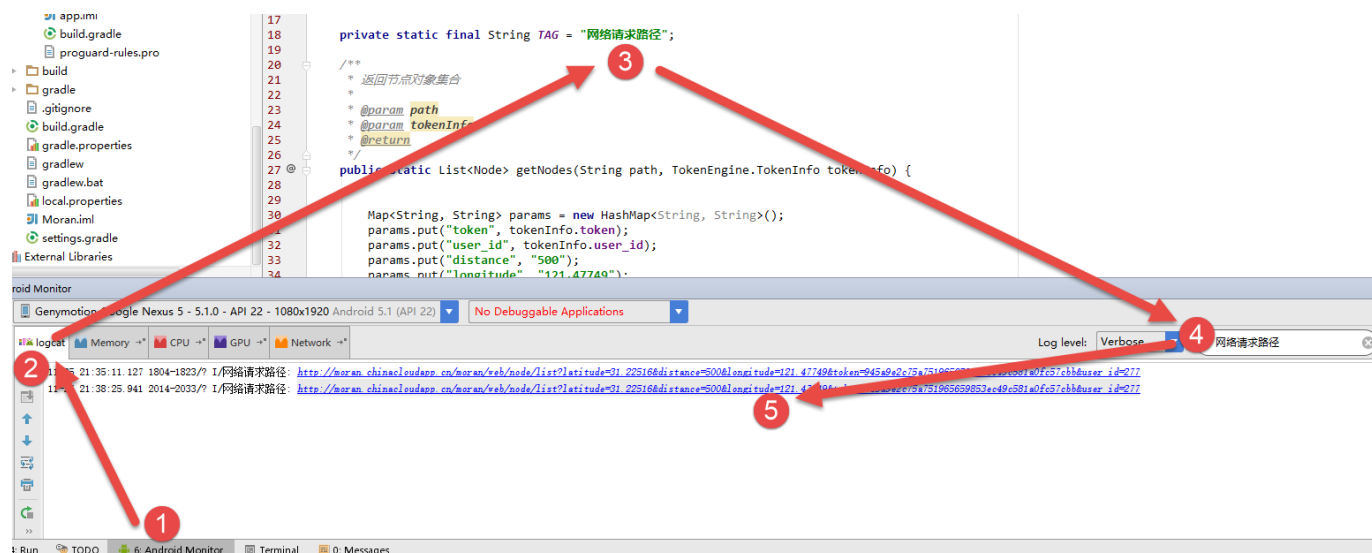
public class NodeEngine {

    private static final String TAG = "网络请求路径";

    public static List<Node> getNodes(String path, TokenEngine.TokenInfo tokenInfo) {
    }
    .....
}

```

部署到模拟器，输入用户名密码，点击登录，返回到Android Studio中，进行如下图所示操作：



点击最下方的 Android Monitor 面板，切换到 logcat 选项卡，复制定义的 TAG 标签，粘贴到过滤器中，点击删选出的网络路径，或直接复制路径到浏览器地址栏，就可以在浏览器窗口中看到返回的数据，小编昨天写代码时是正常的，到写文档时登录一直没跳转，后来发现是服务器的问题，现在问题已解决。

（4）解析返回数据

（未完待续.....）