# ANDROID

## Android Studio

### 发布

buid -》generate sigend APK

签名类型选择V1

### 快捷键

Alt + Enter 解析未导入包的类或未处理异常的代码段，将代码自动加入

Ctrl + Shift + 空格 自动提示生成代码

Ctrl+B 跳转到函数定义（ctrl+鼠标左键）

F8 下一步

shift+F8 下一个断点

右键菜单“Column Selection Mode”就可以开启竖选模式

### 错误处理

========解决adb not responding if youd like to retry...错误

netstat -ano | findstr "5037"

通过线程id，在任务管理器关闭相应的程序

========编译联网问题

在环境变量中增加\_JAVA\_OPTIONS 然后它的值为-Djava.net.preferIPv4Stack=true,之后打开AS就会更新gradle

========More than one file was found with OS independent....

build.gradle中：

android {

.......

packagingOptions {

exclude 'META-INF/DEPENDENCIES'

exclude 'META-INF/NOTICE'

exclude 'META-INF/LICENSE'

exclude 'META-INF/LICENSE.txt'

exclude 'META-INF/NOTICE.txt'

}

}

此时编译报错无法删除某个文件，则手动删除

========控件不显示

修改styles.xml

将Theme改成Base.Theme

设计界面点击魔法棒

### 文件分类

Android目录下app

|  |  |  |
| --- | --- | --- |
| manifests | 工程的配置，包含所有activity的定义 | |
| java | 存放代码文件，通过右键创建activity可以在这里创建出类（同时会在manifests中添加定义，layout中加入xml） | |
| Res | drawable | 存放图片，可以使用复制，选中文件夹进行粘贴 |
| Layout | Activity文件，每个activity对应一个content\_\*\*。按钮design和Text可以切换成xml和可视化显示 |
| Values | 其中定义的值，可以通过name访问。如**"@string/ttt"**  可以访问string标签中的ttt字符串 |

Java文件夹下对应activity的java代码，res/layout下xml对应界面（xml可以直接编辑，也可以使用可视化）

## controls

### 布局

#### ConstraintLayout

**android.support.constraint.ConstraintLayout**

可以在编辑界面任意拖拽控件

控件四个边上的圆点可以拖拽到其他位置生成约束

拖拽控件中间可以移动位置

拖拽控件四个边可以改变控件大小

选中控件后，点击约束点可删除该条约束

通过编辑界面的按钮（Infer Constraint）可以自动设置控件在界面的相对位置

#### LinearLayout

不能在编辑界面任意拖拽控件

自动线性水平排列:

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="horizontal">

#### FrameLayout

不能在编辑界面任意拖拽控件

子元素总是以屏幕的左上角层叠在一起

### Fragment

不包含view，没有findViewById函数

需要在onCreateView中通过view调用findViewById函数

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View vv = inflater.inflate(R.layout.\*\*\*\*, container, false);

｝

### 消息框

import android.widget.Toast;

String str="xx";

Toast.makeText(getApplicationContext(), str,Toast.LENGTH\_SHORT).show();

### 按钮

activity\_\*\*\*.xml中

修改对应控件的id：

**android:id="@+id/btnStart"**

修改按钮显示文字（引用字符串资源）:

**android:text="@string/btn\_start"**

按钮响应1：

**android:onClick="ClickStart"**

**public void** ClickStart(View view) {  
 Toast.*makeText*(**this**,**"\*\*\*\*"**,Toast.***LENGTH\_LONG***).show();  
}

按钮响应2：

传入View.OnClickListener接口对象

btn.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View v) {

System.out.println(".......");

}

});

内部类的函数可以直接访问外部类的成员变量

### 背景图片

#### xml定义方式

xml标签中

android:background="@drawable/a"

表示链接到drawable系列文件夹下的a图像（图像格式任意）

#### 代码设置方式

**btn**.setImageDrawable(getResources().getDrawable(R.drawable.a));

### 去掉标题

androidManifest.xml文件，Application页面最下层Application Nodes，选择好当前的类，进入Theme，选择system Resources"，选Theme.NoTitleBar。

### 透明度

\*\*.getBackground().setAlpha(0);

0代表全透明

### 可见性

setVisibility():

View.VISIBLE:0 可见

View.INVISIBLE:4 不可见，但还占着原来的空间

View.GONE:8 不可见，不占用原来的布局空间

### 编辑框

控件名：Plain Text

修改id为edit

导入包

import android.widget.EditText;

变量

private EditText editText=null;

OnCreate中

editText=(EditText)super.findViewById(R.id.edit);

使用时

String str=editText.getText().toString();

editText.setText("xx");//设置初始显示

### 编辑框样式

新建样式文件editsharp.xml

*<?***xml version="1.0" encoding="utf-8"***?>* <**shape xmlns:android="http://schemas.android.com/apk/res/android"  
 android:shape="rectangle"** >  
  
 <**padding  
 android:bottom="7dp"  
 android:left="7dp"  
 android:right="7dp"  
 android:top="7dp"** />  
 *<!-- 设置圆角矩形 -->* <**corners android:radius="3dp"** />  
  
 <**stroke  
 android:width="5px"  
 android:color="#1199EF"** />  
  
 <**solid android:color="#CCCCCC"** />  
  
 </**shape**>

控件中设置：

**android:background="@drawable/editsharp"**

### ImageView

<**ImageView  
 android:background="@drawable/img\_11"  
 tools:srcCompat="@tools:sample/backgrounds/scenic"** />

ImageView **backImg** = (ImageView)**super**.findViewById(R.id.***imgback***);

**backImg**.setBackground(getResources().getDrawable(R.drawable.***img\_1***));

## frame

### Application

#### Activity包含

application标签中，如果某个Activity包含：

<**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
 <**category android:name="android.intent.category.LAUNCHER"** />  
</**intent-filter**>

则这个Activity类就是app启动时加载的

如果一个程序有多个Activity，且每个都包含以上标签，则会创建多个启动图标，每个图标启动对应的Activity。

<**activity android:name=".aty2" android:label="aty2"**>

Label可以设置启动图标的说明文字

#### App类使用

通过app类共享数据

**public class App** **extends** Application {  
 **public** String **strDat**;  
}

manifests中**application**标签增加属性**android:name=".App"**

此时程序启动加载不使用默认的Application类， 使用App类。

可以通过((App)getApplicationContext()).**strDat** 在Activity之间共享数据

App类中的onCreate()函数在每次Activity启动时都会先进行调用

### Activity

#### 创建

鼠标右键工程文件夹可以new一个activity，自动生成对应的java文件和xml

1.Res/layout文件夹右键new-》layout resource file创建myaty.xml文件，对应页面

2.java文件夹中创建Aty类extends Activity，重写onCreate函数，setContentView(R.layout.myaty);

3.Manifests文件夹中xml进行注册：<**activity android:name=".Aty"**></**activity**>

其中.Aty表示和manifest中package属性中文件夹位置拼凑出的Aty对应java文件的路径

setContentView是每个Activity类构造函数中需要调用的，用来绑定界面xml。传入的参数也可以是控件（控件也是Activity）。

#### 启动

##### 类名启动

假设在MainActivity类中启动一个aty

Intent in = **new** Intent(MainActivity.**this**,aty.**class**);  
startActivity(in); //启动Activity

##### 字符串启动

<**activity android:name=".Myaty" android:exported="true"**>  
 <**intent-filter**>  
 <**category android:name="android.intent.category.DEFAULT"**></**category**>  
 <**action android:name=" AtyName "**></**action**>  
 </**intent-filter**>

可以通过字符串跨程序启动

**android:exported="true"** 默认为ture，表示可以被其他程序启动

<**action android:name="AtyName"**></**action**> AtyName表示Activity标志字符串

=============启动

**try** {  
 startActivity(**new** Intent(**"AtyName "**));  
}**catch** (Exception e){}

=============名称相同

如果两个Activity字符串名称相同，则打开时会进行选择询问

<**action android:name="AtyName"**></**action**> 后面加入：

<**data android:scheme="sss"**></**data**>

startActivity( **new** Intent(**"AtyName "**,Uri.*parse*(**"sss://"**)));

启动时就会自动筛选 //后面是参数

##### 网页启动Activity

<**activity android:name=".Myaty" android:label="aaa"**>  
 <**intent-filter**>  
 <**category android:name="android.intent.category.BROWSABLE"**/>  
 <**category android:name="android.intent.category.DEFAULT"** />  
  
 <**action android:name= "android.intent.action.VIEW"** />  
 <**data android:scheme="app"**></**data**>  
 </**intent-filter**>  
</**activity**>

网页标签

<a href="app://">open</a>

类中可以获取网页传递的启动参数

Uri uri = getIntent().getData();

##### 打开网页

startActivity(new Intent(Intent.ACTION\_VIEW,Uri.parse("https://www.baidu.com/")));

#### 传递参数

##### 传递单个参数

启动activity之前加入

in.putExtra(**"dat"**,**"hello"**);

activity中oncreate中获取

Intent i = getIntent();  
String info = i.getStringExtra(**"dat"**);

##### 传递数据包

启动activity之前加入

in.putExtra(**"name"**,**"aa"**);  
in.putExtra(**"id"**,1);

或者

Bundle b = **new** Bundle();  
b.putString(**"name"**,**"aa"**);  
b.putInt(**"id"**,1);

in.putExtras(b);

//可以是in.putExtra(**"dat"**,b); 通过Bundle b = i.getBundleExtra(**"dat"**);获取

activity中oncreate中获取

Intent i = getIntent();  
Bundle b = i.getExtras();  
  
String name = b.getString(**"name"**);  
**int** id = b.getInt(**"id"**);  
**int** age = b.getInt(**"age"**,20);*//默认值20*

##### 传递自定义数据1

使用java的序列化，操作简单，速度慢

**public class** UserDat **implements** Serializable {  
 String **name** = **"aa"**;  
 UserDat(String name)  
 {  
 **this**.**name** = name;  
 }  
}

加入数据

in.putExtra(**"user"**,**new** UserDat(**"xxx"**));

获取数据

UserDat dat = (UserDat) i.getSerializableExtra(**"user"**);

##### 传递自定义数据2

使用andorid接口，操作复杂，速度快

**public class** UserDat **implements** Parcelable {  
 String **name** = **"aa"**;  
 UserDat(String name) {  
 **this**.**name** = name;  
 }

//可以通过Alt + Enter自动实现（直接选择类名，Add Parcelable implement可以之直接全部自动实现）  
 @Override  
 **public int** describeContents() {  
 **return** 0;  
 }  
 @Override  
 **public void** writeToParcel(Parcel dest, **int** flags) {  
 dest.writeString(**name**);  
 }  
  
  
  
 **protected** UserDat(Parcel in) {  
 **name** = in.readString();  
 }  
 **public static final** Creator<UserDat> ***CREATOR*** = **new** Creator<UserDat>() {  
 @Override  
 **public** UserDat createFromParcel(Parcel in) {  
 **return new** UserDat(in);  
 }  
  
 @Override  
 **public** UserDat[] newArray(**int** size) {  
 **return new** UserDat[size];  
 }  
 };  
  
  
}

传递数据

in.putExtra(**"user"**,**new** UserDat(**"aaa"**));

接收数据

UserDat dat = (UserDat) i.getParcelableExtra(**"user"**);

#### 接收activity返回值

##### 启动activity

Intent in = **new** Intent(MainActivity.**this**,aty.**class**);  
startActivityForResult(in,0);//0代表请求码requestCode

实现虚函数用于接收返回值

@Override  
**protected void** onActivityResult(**int** requestCode, **int** resultCode, Intent data) {  
 **super**.onActivityResult(requestCode, resultCode, data);  
  
 String info = data.getStringExtra(**"dat"**);  
}

##### Activity中退出并设置返回值

Intent in = **new** Intent();  
in.putExtra(**"dat"**,**"xxxxx"**);  
setResult(1,in);//1为返回码resultCode  
finish();*//关闭当前activity*

#### 属性

获取当前Activity的id和名称

**int** taskId = getTaskId();  
String curActivity = toString();

#### 启动方式

Manifest.xml中可以配置Activity启动方式，默认是**standard**

<**activity android:name=".MainActivity" android:launchMode="standard"**>

##### Standard

堆栈方式，新建的Activity向堆栈中放入（创建新实例），当点击后退时，退出当前Activity，显示之前的一个Activity。

taskId相同，不同堆栈位置不同。

##### singleTop

如果当前Activity在栈顶，如当前Activity启动自己，则不会创建新的实例

##### singleTask

一个任务堆栈只包含Activity一个实例，可以包含多个Activity

##### singleInstance

一个任务堆栈只包含一个Activity一个实例

#### 运行流程

onCreate()方法是启动Activity地默认调用的方法

setContentView(R.layout.***activity\_main***);

设置启动的activity

运行流程包括的函数：

onCreate onStart onResume onPause onStop onDestroy onRestart

启动时：onCreate onStart onResume

点击最小化或进入后台时：onPause onStop

再次进入：onRestart onStart onResume

退出activity：onPause onStop onDestroy

在一个activity1中启动另一个activity2时，先执行activity1的onPause ，再执行activity2的onCreate onStart onResume，当activity2完全呈现时，再执行activity1的onStop

activity2退出时，先执行onPause，当activity1执行完onRestart onStart onResume之后，activity2再执行onStop onDestroy

### Servece

#### 特性

程序在退出Activity界面时在系统运行任务中可以看到，如果停止服务，则退出界面时系统运行任务中看不到。

工程文件夹右键new –》Serve自动创建Service类，以及在manifests中加入service标签。与Activity相同。

onCreate和onDestroy会在server被创建和被停止时调用

由startService启动则只能用stopService停止

由bindService启动则只能用unbindService停止

如果Service已经创建，则调用函数时只执行onStartCommand（MyServer中）或者onServiceConnected（Activity中）

#### 启动停止

启动：startService(**new** Intent(MainActivity.**this**,MyService.**class**));

停止：stopService(**new** Intent(MainActivity.**this**,MyService.**class**));

onStartCommand 函数在startService每次调用时执行

#### 绑定

bindService(**new** Intent(MainActivity.**this**, MyService.**class**), MainActivity.**this**, Context.***BIND\_AUTO\_CREATE***);

实现ServiceConnection接口，onServiceConnected在第一次bindService时调用

onServiceConnected(ComponentName name, IBinder service)

MyService类中：

**public** IBinder onBind(Intent intent) {  
 **return new** Binder();  
}

解除绑定：unbindService(MainActivity.**this**);

如果server不存在或已经解绑，则unbindService会抛出异常

====================================

可以通过IBinder接口实现外部直接操作Myserver（在Myserver类中创建一个类MyBinder，MyBinder直接通过Myserver.this访问。外部通过将IBinder转换为Myserver进行访问）

=========MyService类中

**public** IBinder onBind(Intent intent){ **return new** MyBind();}  
**public class** MyBind **extends** Binder  
{  
 **public void** Fun(){ MyService.**this**.Fun();}  
}  
**void** Fun(){System.***out***.println(**"fun"**);}

=========Activity中

**public void** onServiceConnected(ComponentName name, IBinder service) {  
 MyService.MyBind bind = (MyService.MyBind) service;  
 bind.Fun();  
}

#### 跨应用的Servece

##### 启动参数

通过包名和类名地址，可以跨应用启动服务

com.\*\*\*是包名（manifest 中package属性）

MyService是类名

Intent intent = **new** Intent();  
intent.setComponent(**new** ComponentName(**"com.\*\*\* "**,**"com.\*\*\*.MyService"**));

##### 启动

startService(intent);

##### 绑定

生成或加入AIDL文件后先进行重新编译

New AIDL文件 自动生成IMyAidlInterface文件

MyServer中

**public** IBinder onBind(Intent intent) {  
 **return new** IMyAidlInterface.Stub() {  
 @Override  
 **public void** basicTypes(**int** anInt, **long** aLong, **boolean** aBoolean, **float** aFloat, **double** aDouble, String aString) **throws** RemoteException {  
  
 }  
 };  
}

绑定操作

bindService(intent,MainActivity.**this**, Context.***BIND\_AUTO\_CREATE***);

================= 通信

将其他程序的AIDL文件原样拷贝到当前工程

1.New-》Folder-》AIDL Folder

2.为AIDL Folder中创建包，包名和其他程序的AIDL文件包名一样

3.将AIDL文件拷贝

IMyAidlInterface bind = IMyAidlInterface.Stub.*asInterface*(iBinder);

## function

### 线程中操作UI

Handler **handerGetDat** = **new** Handler() {  
 @Override  
 **public void** handleMessage(Message msg) {  
 Bundle bd = msg.getData();  
 String str = bd.getString(**"dat"**);

**tv**.setText(str);  
 **super**.handleMessage(msg);  
 }  
};

线程中调用

Message msg = **new** Message();

String str = **"abc"** ;  
Bundle bd = **new** Bundle();  
bd.putString(**"dat"**,str);

msg.setData(bd);  
**handerGetDat**.sendMessage(msg);

### 网络

#### 权限配置

AndroidMenifest.xml中，manifest标签内添加权限。

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

网络连接必须在线程中

#### http支持

在 res 下新增一个目录(目录名可以叫xml)，然后创建一个名为：network\_security\_config.xml

<?xml version="1.0" encoding="utf-8"?>

<network-security-config>

<base-config cleartextTrafficPermitted="true" />

</network-security-config>

在APP的AndroidManifest.xml文件下的application标签增加以下属性

<application

...

android:networkSecurityConfig="@xml/network\_security\_config"

... />

#### http请求

**public static** String sendHttpRequest(String address) {  
 HttpURLConnection connection = **null**;  
 **try** {  
 URL url = **new** URL(address);  
 connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod(**"POST"**);//只能发出post  
 connection.setConnectTimeout(8000);  
 connection.setReadTimeout(8000);  
 connection.setDoInput(**true**);  
 connection.setDoOutput(**true**);  
 InputStream in = connection.getInputStream();  
 BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(in));  
 StringBuilder builder = **new** StringBuilder();  
 String line;  
 **while** ((line = reader.readLine()) != **null**) {  
 builder.append(line);*// 一行行的读取内容并追加到builder中去* }  
 **return** builder.toString();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return** e.getMessage();  
 } **finally** {  
 **if** (connection != **null**) {  
 connection.disconnect();*//连接不为空就关闭连接* }  
 }  
}

**new** Thread(){  
 @Override  
 **public void** run() {  
 *sendHttpRequest*(**"http://192.168.1.3:8080/"**);  
 }  
}.start();

### 振动

<**uses-permission android:name="android.permission.VIBRATE"**/>

Vibrator vibrator = (Vibrator)**this**.getSystemService(**this**.***VIBRATOR\_SERVICE***);  
vibrator.vibrate(20);

### 去除标题标题栏

styles.xml

修改：

<**style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar"**>  
 <**item name="windowNoTitle"**>true</**item**>  
</**style**>

### 地图

#### 调用

##### key值获取

cmd命令行中，cd .android

输入keytool -list -v -keystore debug.keystore，  
输入密钥库口令，通常是：android

然后就可以获得SHA1安全码:

12:3B:38:F4:AB:44:6D:C9:5C:8F:97:32:DA:F3:93:44:38:07:88:5C

工程中build.gradle(Module:app)内的applicationId

**"com.example.jyl.myapplication"**

得到高德key：4bac2f198d6aa1c7d537c15bc698e48f

##### 加入开发包

将jar和so的文件夹全部放入libs目录

app/build.gradle文件中：

android {  
 compileSdkVersion 28  
 \*\*\*\*\*。。。。。

sourceSets{  
 main{  
 jniLibs.srcDirs = [**'libs'**]  
 }  
 }  
}

右键jar，add as library

##### 配置AndroidManifest.xml

<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.jyl.myapplication"**>  
  
 <**application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme"**>  
 <**activity android:name=".MainActivity"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
  
 <**category android:name="android.intent.category.LAUNCHER"** />  
 </**intent-filter**>  
 </**activity**>  
 <**meta-data android:name="com.amap.api.v2.apikey" android:value="4bac2f198d6aa1c7d537c15bc698e48f"**>  
 //value=开发者申请的key  
 </**meta-data**>  
 </**application**>  
  
  
 *<!--允许程序打开网络套接字-->* <**uses-permission android:name="android.permission.INTERNET"** />  
 *<!--允许程序设置内置sd卡的写权限-->* <**uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"** />  
 *<!--允许程序获取网络状态-->* <**uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"** />  
 *<!--允许程序访问WiFi网络信息-->* <**uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE"** />  
 *<!--允许程序读写手机状态和身份-->* <**uses-permission android:name="android.permission.READ\_PHONE\_STATE"** />  
 *<!--允许程序访问CellID或WiFi热点来获取粗略的位置-->* <**uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION"** />  
  
</**manifest**>

##### 配置界面容器

activity对应的xml中直接加入

<**com.amap.api.maps.MapView  
 android:id="@+id/map"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"**/>

##### 初始化代码

**import** com.amap.api.maps.MapView;  
  
**public class** MainActivity **extends** AppCompatActivity {  
  
 MapView **mMapView** = **null**;  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
 *//获取地图控件引用* **mMapView** = (MapView) findViewById(R.id.***map***);  
 *//在activity执行onCreate时执行mMapView.onCreate(savedInstanceState)，创建地图* **mMapView**.onCreate(savedInstanceState);  
 }  
 @Override  
 **protected void** onDestroy() {  
 **super**.onDestroy();  
 *//在activity执行onDestroy时执行mMapView.onDestroy()，销毁地图* **mMapView**.onDestroy();  
 }  
 @Override  
 **protected void** onResume() {  
 **super**.onResume();  
 *//在activity执行onResume时执行mMapView.onResume ()，重新绘制加载地图* **mMapView**.onResume();  
 }  
 @Override  
 **protected void** onPause() {  
 **super**.onPause();  
 *//在activity执行onPause时执行mMapView.onPause ()，暂停地图的绘制* **mMapView**.onPause();  
 }  
 @Override  
 **protected void** onSaveInstanceState(Bundle outState) {  
 **super**.onSaveInstanceState(outState);  
 *//在activity执行onSaveInstanceState时执行mMapView.onSaveInstanceState (outState)，保存地图当前的状态* **mMapView**.onSaveInstanceState(outState);  
 }  
}

#### 操作

##### 获取控制器

在mapView初始化后可以获取控制器：

//初始化地图控制器对象

AMap aMap = null; if (aMap == null) { aMap = mMapView.getMap(); }

##### 设置标记点

LatLng latLng = **new** LatLng(39.906901,116.397972);  
**final** Marker marker = aMap.addMarker(**new** MarkerOptions().position(latLng).title(**"北京"**).snippet(**"DefaultMarker"**));

##### 标记点窗口的显示和隐藏

======实现地图点击和标记点点击接口

**implements** AMap.OnMapClickListener, AMap.OnMarkerClickListener

**private** Marker **curShowWindowMarker**;  
**public boolean** onMarkerClick(Marker marker) {  
 **curShowWindowMarker** = marker;  
 marker.showInfoWindow();  
 **return true**;  
}  
**public void** onMapClick(LatLng latLng) {  
 **if**(**curShowWindowMarker**!=**null**){  
 **curShowWindowMarker**.hideInfoWindow();  
 }  
}

======设置监听

**aMap**.setOnMapClickListener(**this**);  
**aMap**.setOnMarkerClickListener(**this**);

##### 设置标记点图标

**aMap**.addMarker(**new** MarkerOptions().\*\*\*.icon(BitmapDescriptorFactory.*fromResource*(R.drawable.xx)));

##### 设置地图缩放大小

**aMap**.moveCamera(CameraUpdateFactory.*zoomTo*(5));