## 新建项目

在build.gradle(module)中添加arcgis的maven仓库地址。arcgis没有将相关的包上传在公共仓库上，所以必须添加arcgis自己的maven仓库地址才能下载到对应的包。



在build.gradle(app)中添加依赖，本项目使用了两个额外包，分别是arcgis的android包以及一个网络请求包。



在AndroidManifest.xml文件中添加网络权限请求。

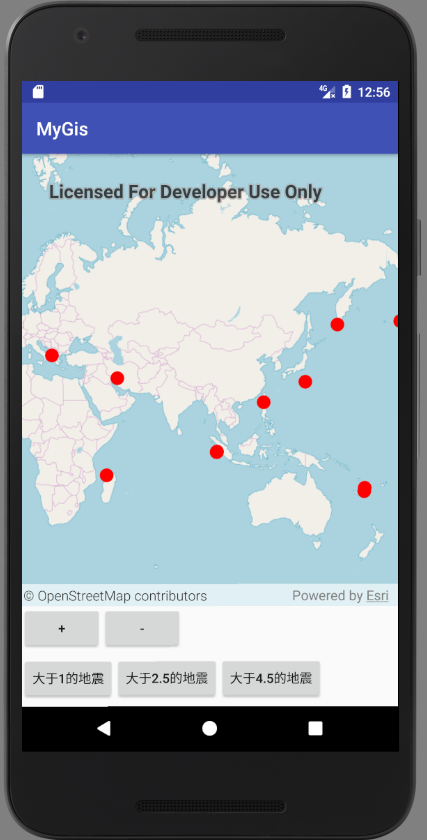


## 界面布局

在xml文件中添加mapview控件。



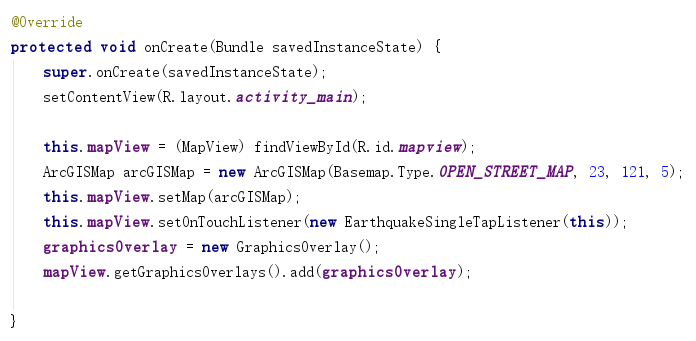
这个控件是arcgis android包的控件用于地图的显示。其它的按钮控件则是android本身的普通控件，添加并设置好相应的属性即可（参照代码）。



## 逻辑代码

### 初始化

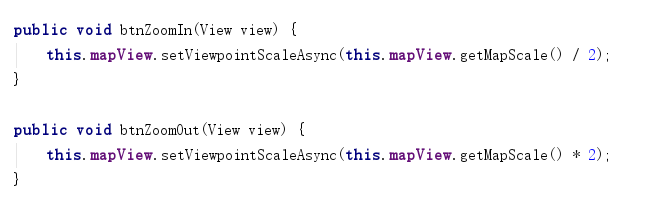
在activity的OnCreate函数完成必要的初始化工作，此处和arcgis相关的部分是，从xml文件获得地图视图对象，为地图视图设置地图（osm）并指定初始的中心位置和缩放比例，为地图视图设置触摸事件监听器，为地图视图添加一个graphic层(矢量)。



### 缩放事件响应函数

arcgis android的默认事件监听器实现了通过手势来实现缩放操作，模拟器上不便于使用手势操作，所以添加了两个按钮来进行缩放操作。

arcgis android在mapview上没有直接提供修改level of detail的函数，而是通过修改比例尺来进行缩放。

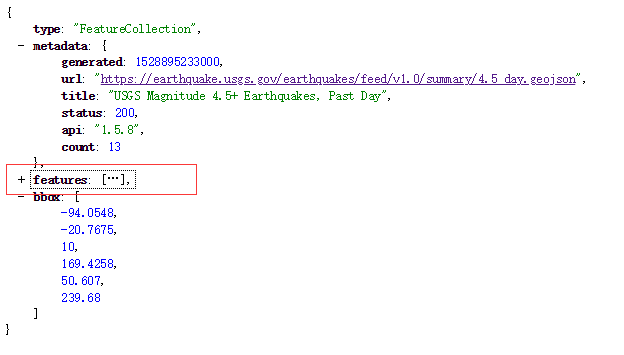


### 添加地震数据响应函数

usgs以json接口的形式提供了地震数据。这个app使用了近一天内级别大于1的地震，级别大于2.5的地震，级别大于4.5的地震这3个数据接口，它的api如下：



其中eqType可以是1.0，2.5或者4.5。api的返回格式为json格式



#### Android网络请求

为了避免阻塞UI线程，导致app失去响应（造成卡死的状态），通常所有网络请求需要在其它线程进行。手动的来创建和管理线程，并且处理网络请求返回的流数据是很繁琐和易错的工作，所以通常会借助网络请求包来实现网络请求。

android的官方文档（https://developer.android.com/training/volley/

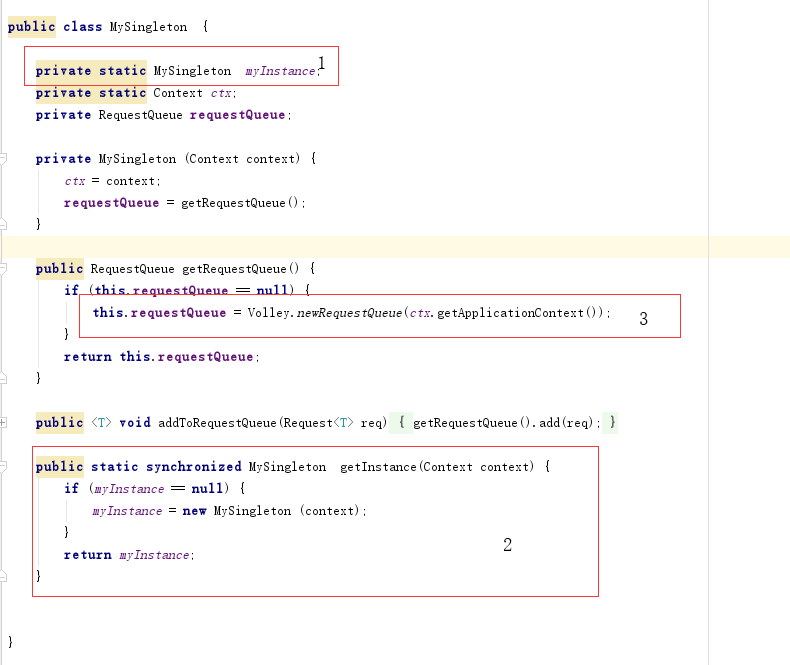
）介绍了一个volley库来使得网络请求更加的简单和快速。

volley的主要缺点是它需要在内存中完成所有数据的解析，所以不适合文件下载之类的操作。

一般来说，我们使用volly来创建一个RequestQueue(请求队列)，然后把Request(请求对象)传递给请求队列。请求队列会管理多个工作线程，对于传给它的请求对象， 它会自动的完成缓存读取（如果有缓存），发送网络请求，解析相应结果，缓存结果，将结果传回给主线程等操作。

我们也可以中途取消一个请求。

一般来说在一个应用内的很多地方都会需要进行网络请求，如果整个应用只使用一个”请求队列”app的性能会更好（而不是重复创建和销毁请求队列）。使用“单例模式”是一种很好的方法。要确保在单例中请求队列的参数是Application的context，这样才能保证请求队列的生命期和app一致。



单例的实现方法。

1. 类内部有一个静态成员，是这个类的对象

2. 在获得单例的函数中，直接返回静态成员。（第一次调用时，调用构造函数完成静态成员初始化）

3. 类内部有一个请求队列对象，请求队列使用application的context进行初始化。

volley提供了几个标准的请求对象。

StringRequest: 从指定的url处获得string

JsonObjectRequest：从指定的url处获得json

JsonArrayRequest：从指定的url处获得json array

String url = "http://my-json-feed";  
  
JsonObjectRequest jsonObjectRequest = new JsonObjectRequest  
        (Request.Method.GET, url, null, new Response.Listener<JSONObject>() {  
  
    @Override  
    public void onResponse(JSONObject response) {  
        mTextView.setText("Response: " + response.toString());  
    }  
}, new Response.ErrorListener() {  
  
    @Override  
    public void onErrorResponse(VolleyError error) {  
        // TODO: Handle error  
  
    }  
});  
  
// Access the RequestQueue through your singleton class.  
MySingleton.getInstance(this).addToRequestQueue(jsonObjectRequest);

实例化一个对应的请求对象，并将请求对象添加到请求队列中即完成了请求。请求对象需要指定，请求的url，请求的http 方法，请求的参数，请求成功后的处理函数，请求出错后的处理函数。(注意，此处是异步的，线程并不会阻塞在此等待内容返回)

#### json解析创建graphic对象

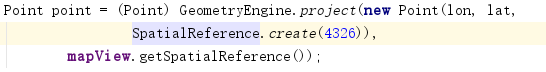
JSONArray featureArray = response.getJSONArray(**"features"**);

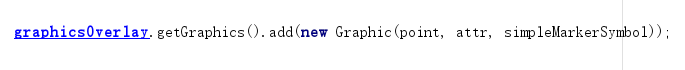
featureArray.getJSONObject(i).getJSONObject(**"geometry"**);

featureArray.getJSONObject(i).getJSONObject(**"properties"**);

String place = properties.getString(**"place"**).toString()

通过使用json的接口，将数据取出即可，具体可参照代码。

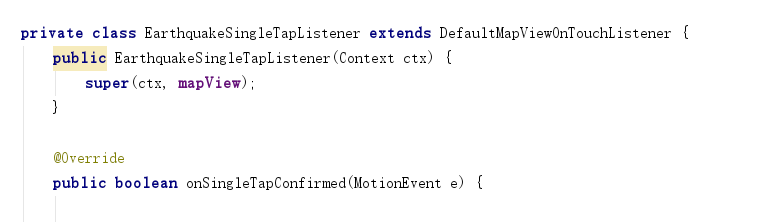
 



将json转化为graphic并添加到graphic层中。

**for** (**int** i = 0; i < featureArray.length(); i++) {  
 *// 获得几何信息* JSONObject geometry = featureArray.getJSONObject(i).getJSONObject(**"geometry"**);  
 **double** lon = Double.*parseDouble*(geometry.getJSONArray(**"coordinates"**).get(0).toString());  
 **double** lat = Double.*parseDouble*(geometry.getJSONArray(**"coordinates"**).get(1).toString());  
 *// 新建点，此处是两步，新建点和投影，api返回的数据是wgs84投影，需要转换为何底图一致的投影* Point point = (Point) GeometryEngine.*project*(**new** Point(lon, lat,  
 SpatialReference.*create*(4326)),  
 **mapView**.getSpatialReference());  
 *// 获得属性信息* JSONObject properties = featureArray.getJSONObject(i).getJSONObject(**"properties"**);  
  
 Map<String, Object> attr = **new** HashMap<String, Object>();  
  
 String place = properties.getString(**"place"**).toString();  
 attr.put(**"place"**, place);  
  
 **float** mag = Float.*valueOf*(properties.getString(**"mag"**).toString());  
 attr.put(**"mag"**, mag);  
  
 **float** size = getSizeFromMag(mag);  
 simpleMarkerSymbol.setSize(size);  
 *// 将属性信息中的unix时间戳转化为本地时间字符串* **long** time = Long.*valueOf*(properties.getString(**"time"**).toString());  
 Date date = **new** Date(time);  
 attr.put(**"date"**, date.toString());  
  
 attr.put(**"rms"**, properties.getString(**"rms"**).toString());  
 attr.put(**"gap"**, properties.getString(**"gap"**).toString());  
  
 *// 新建并添加graphic元素，参数为坐标点，属性字典，符号* **graphicsOverlay**.getGraphics().add(**new** Graphic(point, attr, simpleMarkerSymbol));  
  
 postGetData();  
  
}

#### 地图视图触摸监听器



继承DefaultMapViewOnTouchListener来实现自定义的监听器，因为默认的实现在DefaultMapViewOnTouchListener中，最好不要直接实现接口。

onSingleTapConfirmed响应单击事件。我们希望点击每个地震点以后可以弹出一个弹出框，显示地震的信息。

@Override  
**public boolean** onSingleTapConfirmed(MotionEvent e) {  
  
 **mMapView**.getCallout().dismiss(); *// 关闭弹窗* **for**(**int** i = 0; i < **graphicsOverlay**.getGraphics().size(); i++) {  
 **graphicsOverlay**.getGraphics().get(i).setSelected(**false**); *// 将所有元素设置为未选中* }  
  
 **final** android.graphics.Point screenPoint = **new** android.graphics.Point((**int**)e.getX(), (**int**)e.getY()); *// 获得屏幕点* **final** Point mapPoint = **mapView**.screenToLocation(screenPoint); *// 将屏幕点转为坐标点  
 // 标准的识别方法， 指定在graphics图层中，屏幕点周围10单位,识别最多10个元素* **final** ListenableFuture<IdentifyGraphicsOverlayResult> identifyFuture = **mapView**.identifyGraphicsOverlayAsync(**graphicsOverlay**, screenPoint, 10, **false**, 10);  
 *// 识别结果通过监听器函数获得* identifyFuture.addDoneListener(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **try** {  
 *// 获得识别结果* List<Graphic> identifiedGraphics = identifyFuture.get().getGraphics();  
  
 **for** (Graphic graphic : identifiedGraphics) {  
 *// 遍历识别结果， 由于是点击，我们只处理一个识别到的元素* graphic.setSelected(**true**); *// 选中元素* Map<String, Object> attrs = graphic.getAttributes(); *// 获得元素上的属性* String text = **""**;  
 **for** (**int** i = 0; i < attrs.size(); i++) {  
 text = text + attrs.keySet().toArray()[i] + **": "** + attrs.values().toArray()[i] + **"\n"**;  
 } *// 将属性拼接为字符串* TextView tv = **new** TextView(MainActivity.**this**); *// 新建文本框* tv.setText(text);  
 tv.setTextSize(20);  
  
 Callout mapCallout = **mMapView**.getCallout(); *// 新建弹出框* mapCallout.setContent(tv); *// 将文本框加到弹出框中* mapCallout.setLocation(mapPoint); *// 设置弹出框位置为点击处* mapCallout.show(); *// 显示弹出框* **break**;  
  
 }  
 } **catch** (Exception e){  
 }  
 }  
 });  
  
  
 **return true**;  
}