# Android面试知识总结：

# Android的最基本的知识点：

# Android的四大组件：

# Activity：

用来跟用户来进行交互的界面，使用过程中要注意的就是他的生命周期，

onCreate的时候只是加载布局文件，进行控件的初始化，这个时候用户还看不到界面，只有在onStart以后，用户才能看的到界面，但是还不能跟界面进行交互。也就是这一步界面的初始化都已经完成，当onResume执行以后，界面可以接收用户的交互，点击，触摸等手势操作。

OnPause, onStop的调用是在当界面从新边的不可见的时候调用，（点home键，点返回键，只要是界面边的不可见，那么就肯定会调用它）

而在销毁一个Activity的时候会依次调用 onPause，onStop，onDestry。

使用好Activity的什么周期，结合Fragment的生命周期可以实现比较好的用户体验的功能。

当Activity界面上弹出一个Dialog的时候，不会对Activity造成影响。

# Service：

Service顾名思义，就是一个服务，他是Android中一个运行在后台的组件，

Service其实也是运行在主线程中，我们可以通过Thread.*currentThread*().getName()

获取到当前线程的名称，可以发现是在主线程中，但是Service一般用来做一些后台的，耗时的操作，这个是通过在他里边开线程进行实现的。

Android 还提供了IntentService，这个Service中会自己开线程，也就是操作都是在子线程中。

关于Service使用的场景很多，但是常见的是：音乐播放器，做轮询，做心跳，做下载管理器等等。

使用方式有俩种：

直接startService，这个是通过Intent来进行直接启动。

Intent intent = **new** Intent(**this**,Service.**class**);  
startService(intent);

还有就是绑定service，通过

@Override  
**public boolean** bindService(Intent service, ServiceConnection conn,  
 **int** flags) {  
 **return mBase**.bindService(service, conn, flags);  
}

这俩种方式的区别就是startService启动了service以后，交互的实现比较复杂，

比如音乐播放器的后台播放，想实现暂停的功能，他的播放是在Service中，那么暂停的时候是没有办法控制的，但是用bindService的话，就可以通过ServiceConnection

对象来获取service对象，就可以调用service中的方法，然后进行控制。

# BroadCast：

# ContentProvider：

# Android的基础控件：

# ListView：

# GridView：

# ScrollView：

# ProgressBar：

# ToolBar:

# ImageView:

# TextView

# Button:

# [Gallery的使用](http://www.cnblogs.com/menlsh/archive/2013/02/26/2934434.html)：

　在Android中，画廊控件Gallery用来显示图片列表，可以用手指直接拖动图片左右移动。Gallery只能水平显示一行，且Gallery列表中的图片会根据不同的拖动情况向左或向右移动，直到显示到最后一个图片为止。

使用模式：

适配器模式：

Gallery对象，适配器对象。

监听： onItemClickListener。

# ViewPager：

# WebView：

# 等。。。。

# 3、Android 的基础布局：

# LinearLayout：

# RelativeLayout：

# TabLayout：

# FrameLayout：

# AbsoluteLayout：

# Android Fragment的使用以及生命周期。

# Fragment与Fragment之间的交互

# Fragment 与Activity的交互

# Android Handler的使用场景以及源码实现流程：

# Android的Activity的启动模式分析以及代码中动态设置启动模式

# Android 事件处理，事件拦截：

# 数据库的使用：

# Android Dialog使用，以及源码实现流程

# Android Toast 使用以及源码实现流程（Android Toast自定义）

# Android 网络访问：

# Android socket编程

# Android 多线程：

# Android 图片上传，文件上传，文件下载，断点续传：

# Android 图片压缩

# Android 动画：

# 帧动画：

# 补间动画：

# 属性动画：

属性动画：ValueAnimation

# OpenGl使用实现3d效果

# Android传感器：

**Android的传感器开发**

1.1 开发传感器应用

开发传感器的步骤如下：

调用Context的getSystemService(Context.SENSOR\_SERVICE)方法获取SensorManager对象。

调用SensorManager的getDefaultSensor(int type)方法来获取指定类型的传感器。

一般在Activity的onResume()方法中调用SensorManager的registerListener()为指定传感器注册监听器即可。程序可以通过实现监听器即可获取传感器传回来的数据。

SersorManager提供的注册传感器的方法为registerListener(SensorListener listener, Sensor sensor, int rate)该方法中三个参数说明如下：

listener:监听传感器事件的监听器

sensor：传感器对象

rate：指定获取传感器数据的频率

rate可以获取传感器数据的频率，支持如下几个频率值：

SENSOR\_DELAY\_FASTEST：最快，延迟最小。

SENSOR\_DELAY\_GAME：适合游戏的频率。

SENSOR\_DELAY\_NORMAL：正常频率

SENSOR\_DELAY\_UI：适合普通用户界面的频率。

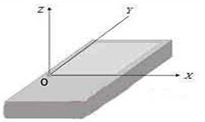
例：加速度传感器：

AccelerometerTest.java

[?](http://www.2cto.com/kf/201501/367225.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | public class AccelerometerTest extends Activity      implements SensorEventListener  {      // 定义系统的Sensor管理器      SensorManager sensorManager;      EditText etTxt1;        @Override      public void onCreate(Bundle savedInstanceState)      {          super.onCreate(savedInstanceState);          setContentView(R.layout.main);          // 获取程序界面上的文本框组件          etTxt1 = (EditText) findViewById(R.id.txt1);          // 获取系统的传感器管理服务          sensorManager = (SensorManager) getSystemService(              Context.SENSOR\_SERVICE);      }        @Override      protected void onResume()      {          super.onResume();          // 为系统的加速度传感器注册监听器          sensorManager.registerListener(this,              sensorManager.getDefaultSensor(Sensor.TYPE\_ACCELEROMETER),              SensorManager.SENSOR\_DELAY\_GAME);      }        @Override      protected void onStop()      {          // 取消注册          sensorManager.unregisterListener(this);          super.onStop();      }        // 以下是实现SensorEventListener接口必须实现的方法      // 当传感器的值发生改变时回调该方法      @Override      public void onSensorChanged(SensorEvent event)      {          float[] values = event.values;          StringBuilder sb = new StringBuilder();          sb.append(X方向上的加速度：);          sb.append(values[0]);          sb.append(  Y方向上的加速度：);          sb.append(values[1]);          sb.append(  Z方向上的加速度：);          sb.append(values[2]);          etTxt1.setText(sb.toString());      }        // 当传感器精度改变时回调该方法。      @Override      public void onAccuracyChanged(Sensor sensor, int accuracy)      {      }  } |

需要指出的是，传感器的坐标系统与屏幕坐标系统不同，传感器坐标系统的X轴沿屏幕向右；Y轴则沿屏幕向上，Z轴在垂直屏幕向上。



当拿着手机横向左右移动时，可能产生X轴上的加速度；拿着手机前后移动时，可能产生Y轴上的加速度；当拿着手机竖向上下移动时，可能产生Z轴上的加速度。

1.2 下载和安装SensorSimulator

SensorSimulator是传感器的模拟工具，安装这个模拟工具之后，开发者就可以在Android模拟器上开发、调试传感器应用。

SensorSimulator，由PC端程序和手机端程序组成，当两端的程序运行并建立连接之后，用户可以通过PC端的程序来改变手机的传感数据。

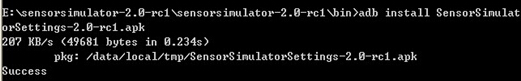
下载和安装SensorSimulator步骤如下：

登录到http://code.google.com/p/openintents/wiki/SensorSimulator站点或FTP上，下载SensorSimulator的最新版本。

下载SensorSimulator工具后，下载完成后得到一个sensorsimulator-2.0-rc1.zip压缩包。解压该文件，得到如下文件结构。



安装SensorSimulator的手机端程序。通过命令窗口进入到上面文件的bin目录下，输入如下命令来安装SensorSimulatorSettings-2.0-rc1.apk文件。adb install SensorSimulatorSettings-2.0-rc1.apk



运行SensorSimulator的PC端程序，通过命令窗口进入到上面文件的bin目录下，并在窗口内执行如下命令：java –jar sensorsimulator-2.0-rc1.jar。运行该程序出现如下界面。

运行SensorSimulator的手机端程序。

在SensorSimulator的手机端程序中填写SensorSimulator的PC端程序的监听IP地址、监听端口。

切换到SensorSimulator的Testting Tab页，单击该Tab里的Connect按钮，SensorSimulator手机端和PC端连接。

1.3 利用SensorSimulator开发传感器应用

通过使用SensorSimulator，接下来就可以在Android模拟器中开发、调试传感器应用了。不过使用SensorSimulator开发传感器应用与开发真实的传感器应用略有区别。

Android应用必须通过引用外部JAR包的形式来引用SensorSimulator的lib目录下的sensorsimulator-2.0-rc1.jar包。

在应用项目上右键单击选择“Build Path” à “Add External Archives…”，找到sensorsimulator-2.0-rc1.jar所在位置，将其添加到项目中。

应用程序编程使用SensorManagerSimulator代替了原有的SensorManager。

应用程序获取SensorManagerSimulator之后，需要调用connectSimulator()方法连接模拟器。

应用程序编程时所用的Sensor、SensorEvent、 SensorEventListener等不再是Android提供的类，而是由SensorSimulator提供的类。

应用程序需要访问网络的权限。

例：利用传感模拟工具开发加速度传感器：

AccelSimulatorTest.java

[?](http://www.2cto.com/kf/201501/367225.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | import org.openintents.sensorsimulator.hardware.SensorManagerSimulator;  import org.openintents.sensorsimulator.hardware.Sensor;  import org.openintents.sensorsimulator.hardware.SensorEvent;  import org.openintents.sensorsimulator.hardware.SensorEventListener;    import android.app.Activity;  import android.hardware.SensorManager;  import android.os.Bundle;  import android.widget.EditText;    public class AccelSimulatorTest extends Activity      implements SensorEventListener  {      // 定义模拟器的Sensor管理器      private SensorManagerSimulator mSensorManager;      // 定义界面上的文本框组件      EditText etTxt1;        @Override      public void onCreate(Bundle savedInstanceState)      {          super.onCreate(savedInstanceState);          setContentView(R.layout.main);          // 获取程序界面的文本框组件          etTxt1 = (EditText) findViewById(R.id.txt1);          // 获取传感器模拟器的传感器管理服务          mSensorManager = SensorManagerSimulator.getSystemService(              this, SENSOR\_SERVICE);          // 连接传感器模拟器          mSensorManager.connectSimulator();      }        @Override      protected void onResume()      {          super.onResume();          // 为系统的加速度传感器注册监听器          mSensorManager.registerListener(this,              mSensorManager.getDefaultSensor(Sensor.TYPE\_ACCELEROMETER),              SensorManager.SENSOR\_DELAY\_GAME);      }        @Override      protected void onStop()      {          // 取消注册          mSensorManager.unregisterListener(this);          super.onStop();      }        // 以下是实现SensorEventListener接口必须实现的方法      // 当传感器的值发生改变时回调该方法      @Override      public void onSensorChanged(SensorEvent event)      {          float[] values = event.values;          StringBuilder sb = new StringBuilder();          sb.append(X方向上的加速度：);          sb.append(values[0]);          sb.append(  Y方向上的加速度：);          sb.append(values[1]);          sb.append(  Z方向上的加速度：);          sb.append(values[2]);          etTxt1.setText(sb.toString());      }        @Override      // 当传感器精度改变时回调该方法。      public void onAccuracyChanged(Sensor sensor, int accuracy)      {      }  } |

[?](http://www.2cto.com/kf/201501/367225.html)

|  |  |
| --- | --- |
| 1  2 | <!-- 通过模拟器调试需要访问网络 -->      <uses-permission android:name="android.permission.INTERNET/"></uses-permission> |

**Android的常用传感器**

2.1加速度传感器Accelerometer

加速度传感器主要感应手机的运动，在注册了传感器监听器后加速度传感器主要捕获3个参数values[0]、values[1]、values[2]。

values[0]：空间坐标系中x轴方向上的加速度减去重力加速度减去中立加速度在x轴上的分量。

values[1]：空间坐标系中x轴方向上的加速度减去重力加速度减去中立加速度在y轴上的分量。

values[2]：空间坐标系中x轴方向上的加速度减去重力加速度减去中立加速度在z轴上的分量。

上述3个数据的单位均为米每二次方秒。

距离说明：

当手机平放到桌面静止时，加速度为重力加速度g，通过0减去-g（重力加速度g方向为z轴反方向，故为负值）得到values[2]为g。

如果把手机水平方向右推，此时手机x方向上的加速度为正，即values[0]为正。

当把手机以a米每二次方秒的加速度竖值向上举时，values[2]的返回值为(a+g)米每二次方秒，通过a减去-g得到。

2.2 方向传感器Orientation

方向传感器主要感应手机方位的变化，其每次读取的都是静态的状态值，在注册了传感器监听器后方向传感器主要捕获3个参数values[0]、values[1]、values[2]，关于三个角度的说明如下：

第一个角度：表示手机顶部朝向与正北方向的夹角。当手机绕着Z轴旋转时，该角度值发生改变。

第二个角度：表示手机顶部或尾部翘起的角度，当手机绕着X轴倾斜时，该角度值发生变化。

第三个角度：表示手机左侧或右侧翘起的角度。当手机绕着Y轴倾斜时，该角度值发生变化。

2.3磁场传感器Magnetic Field

磁场传感器主要用于感应周围的磁感应强度。即使周围没有任何直接的磁场，手机设备也始终会处于地球磁场中。随着手机状态设备摆放状态的改变，周围磁场在手机的X、Y、Z方向上的会发生改变。

磁场传感器传感器会返回三个数据，三个数据分别代表周围磁场分解到X、Y、Z三个方向上的磁场分量。磁场数据的单位是微特斯拉(uT)。

2.4光传感器Light

光传感器用于感应周围的光强，注册监听器后只捕获一个参数：values[0]。该参数代表周围的光照强度，单位为勒克斯（lux）。

2.5温度传感器Temperature

温度传感器用于获取手机设备所处环境的温度。温度传感器会返回一个数据，该数据代表手机设备周围的温度，单位是摄氏度。

2.6压力传感器 Pressure

压力传感器用于获取手机设备所处环境的压力的大小。压力传感器会返回一个数据，代表手机设备周围的压力大小。

例：传感器应用：



SensorSimulatorTest.java

[?](http://www.2cto.com/kf/201501/367225.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145 | public class SensorSimulatorTest extends Activity      implements SensorEventListener  {      // // 定义真机的Sensor管理器      // private SensorManager mSensorManager;      // 定义模拟器的Sensor管理器      private SensorManagerSimulator mSensorManager;        EditText etOrientation;      EditText etMagnetic;      EditText etTemerature;      EditText etLight;      EditText etPressure;        @Override      public void onCreate(Bundle savedInstanceState)      {          super.onCreate(savedInstanceState);          setContentView(R.layout.main);          // 获取界面上的EditText组件          etOrientation = (EditText) findViewById(R.id.etOrientation);          etMagnetic = (EditText) findViewById(R.id.etMagnetic);          etTemerature = (EditText) findViewById(R.id.etTemerature);          etLight = (EditText) findViewById(R.id.etLight);          etPressure = (EditText) findViewById(R.id.etPressure);          // 获取真机的传感器管理服务          // mSensorManager = (SensorManager)getSystemService(SENSOR\_SERVICE);          // 获取传感器模拟器的传感器管理服务          mSensorManager = SensorManagerSimulator.getSystemService(this,              SENSOR\_SERVICE);          // 连接传感器模拟器          mSensorManager.connectSimulator();      }        @Override      protected void onResume()      {          super.onResume();          // 为系统的方向传感器注册监听器          mSensorManager.registerListener(this,              mSensorManager.getDefaultSensor(Sensor.TYPE\_ORIENTATION),              SensorManager.SENSOR\_DELAY\_GAME);          // 为系统的磁场传感器注册监听器          mSensorManager.registerListener(this,              mSensorManager.getDefaultSensor(Sensor.TYPE\_MAGNETIC\_FIELD),              SensorManager.SENSOR\_DELAY\_GAME);          // 为系统的温度传感器注册监听器          mSensorManager.registerListener(this,              mSensorManager.getDefaultSensor(Sensor.TYPE\_TEMPERATURE),              SensorManager.SENSOR\_DELAY\_GAME);          // 为系统的光传感器注册监听器          mSensorManager.registerListener(this,              mSensorManager.getDefaultSensor(Sensor.TYPE\_LIGHT),              SensorManager.SENSOR\_DELAY\_GAME);          // 为系统的压力传感器注册监听器          mSensorManager.registerListener(this,              mSensorManager.getDefaultSensor(Sensor.TYPE\_PRESSURE),              SensorManager.SENSOR\_DELAY\_GAME);      }        @Override      protected void onStop()      {          // 程序退出时取消注册传感器监听器          mSensorManager.unregisterListener(this);          super.onStop();      }        @Override      protected void onPause()      {          // 程序暂停时取消注册传感器监听器          mSensorManager.unregisterListener(this);          super.onPause();      }        // 以下是实现SensorEventListener接口必须实现的方法      @Override      // 当传感器精度改变时回调该方法。      public void onAccuracyChanged(Sensor sensor, int accuracy)      {      }        @Override      public void onSensorChanged(SensorEvent event)      {          float[] values = event.values;          // // 真机上获取触发event的传感器类型          // int sensorType = event.sensor.getType();          // 模拟器上获取触发event的传感器类型          int sensorType = event.type;          StringBuilder sb = null;          // 判断是哪个传感器发生改变          switch (sensorType)          {              // 方向传感器              case Sensor.TYPE\_ORIENTATION:                  sb = new StringBuilder();                  sb.append(绕Z轴转过的角度：);                  sb.append(values[0]);                  sb.append(  绕X轴转过的角度：);                  sb.append(values[1]);                  sb.append(  绕Y轴转过的角度：);                  sb.append(values[2]);                  etOrientation.setText(sb.toString());                  break;              // 磁场传感器              case Sensor.TYPE\_MAGNETIC\_FIELD:                  sb = new StringBuilder();                  sb.append(X方向上的角度：);                  sb.append(values[0]);                  sb.append(  Y方向上的角度：);                  sb.append(values[1]);                  sb.append(  Z方向上的角度：);                  sb.append(values[2]);                  etMagnetic.setText(sb.toString());                  break;              // 温度传感器              case Sensor.TYPE\_TEMPERATURE:                  sb = new StringBuilder();                  sb.append(当前温度为：);                  sb.append(values[0]);                  etTemerature.setText(sb.toString());                  break;              // 光传感器              case Sensor.TYPE\_LIGHT:                  sb = new StringBuilder();                  sb.append(当前光的强度为：);                  sb.append(values[0]);                  etLight.setText(sb.toString());                  break;              // 压力传感器              case Sensor.TYPE\_PRESSURE:                  sb = new StringBuilder();                  sb.append(当前压力为：);                  sb.append(values[0]);                  etPressure.setText(sb.toString());                  break;          }      }  } |

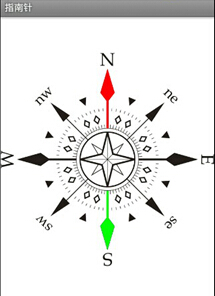
传感器应用案例

对传感器的支持是Android系统的特性之一，通过使用传感器可以开发出各种有趣的应用，我们通过方向传感器来开发指南针。

开发指南针的思路比较简单：程序先准备一张指南针图片，该图片上方向指针指向北方。接下来开发一个检测方向的传感器，程序检测到手机顶部绕Z轴转过多少度，让指南针图片反向转过多少度即可。

该应用中只要在界面中添加一张图片，并让图片总是反向转过方向传感器反回的第一个角度即可。

例：指南针：



Compass.java

[?](http://www.2cto.com/kf/201501/367225.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85 | public class Compass extends Activity      implements SensorEventListener  {      // 定义显示指南针的图片      ImageView znzImage;      // 记录指南针图片转过的角度      float currentDegree = 0f;      // 定义模拟器的Sensor管理器  //  private SensorManagerSimulator mSensorManager;         // 定义真机的Sensor管理器      SensorManager mSensorManager;      @Override      public void onCreate(Bundle savedInstanceState)      {          super.onCreate(savedInstanceState);          setContentView(R.layout.main);          // 获取界面中显示指南针的图片          znzImage = (ImageView) findViewById(R.id.znzImage);             // 获取真机的传感器管理服务           mSensorManager = (SensorManager)getSystemService(SENSOR\_SERVICE);  //      // 获取传感器模拟器的传感器管理服务  //      mSensorManager = SensorManagerSimulator.getSystemService(this,  //          SENSOR\_SERVICE);  //      // 连接传感器模拟器  //      mSensorManager.connectSimulator();      }        @Override      protected void onResume()      {          super.onResume();          // 为系统的方向传感器注册监听器          mSensorManager.registerListener(this,              mSensorManager.getDefaultSensor(Sensor.TYPE\_ORIENTATION),              SensorManager.SENSOR\_DELAY\_GAME);      }        @Override      protected void onPause()      {          // 取消注册          mSensorManager.unregisterListener(this);          super.onPause();      }        @Override      protected void onStop()      {          // 取消注册          mSensorManager.unregisterListener(this);          super.onStop();      }        @Override      public void onSensorChanged(SensorEvent event)      {          // 真机上获取触发event的传感器类型          int sensorType = event.sensor.getType();  //      // 模拟器上获取触发event的传感器类型  //      int sensorType = event.type;          switch (sensorType)          {              case Sensor.TYPE\_ORIENTATION:                  // 获取绕Z轴转过的角度。                  float degree = event.values[0];                  // 创建旋转动画（反向转过degree度）                  RotateAnimation ra = new RotateAnimation(currentDegree,                      -degree, Animation.RELATIVE\_TO\_SELF, 0.5f,                      Animation.RELATIVE\_TO\_SELF, 0.5f);                  // 设置动画的持续时间                  ra.setDuration(200);                  // 运行动画                  znzImage.startAnimation(ra);                  currentDegree = -degree;                  break;          }      }        @Override      public void onAccuracyChanged(Sensor sensor, int accuracy)      {      }  } |
|  |  | |

例：水平仪：

Gradienter.java

[?](http://www.2cto.com/kf/201501/367225.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147 | public class Gradienter extends Activity implements SensorEventListener  {      // 定义水平仪的仪表盘      MyView show;      // 定义水平仪能处理的最大倾斜角，超过该角度，气泡将直接在位于边界。      int MAX\_ANGLE = 30;      // // 定义真机的Sensor管理器      // SensorManager mSensorManager;      // 定义模拟器的Sensor管理器      SensorManagerSimulator mSensorManager;        @Override      public void onCreate(Bundle savedInstanceState)      {          super.onCreate(savedInstanceState);          setContentView(R.layout.main);          // 获取水平仪的主组件          show = (MyView) findViewById(R.id.show);          // 获取真机的传感器管理服务          // mSensorManager = (SensorManager)getSystemService(SENSOR\_SERVICE);          // 获取传感器模拟器的传感器管理服务          mSensorManager = SensorManagerSimulator.getSystemService(this,              SENSOR\_SERVICE);          // 连接传感器模拟器          mSensorManager.connectSimulator();      }        @Override      public void onResume()      {          super.onResume();          // 为系统的方向传感器注册监听器          mSensorManager.registerListener(this,              mSensorManager.getDefaultSensor(Sensor.TYPE\_ORIENTATION),              SensorManager.SENSOR\_DELAY\_GAME);      }        @Override      protected void onPause()      {          // 取消注册          mSensorManager.unregisterListener(this);          super.onPause();      }        @Override      protected void onStop()      {          // 取消注册          mSensorManager.unregisterListener(this);          super.onStop();      }        @Override      public void onAccuracyChanged(Sensor sensor, int accuracy)      {      }        @Override      public void onSensorChanged(SensorEvent event)      {          float[] values = event.values;          // // 真机上获取触发event的传感器类型          // int sensorType = event.sensor.getType();          // 模拟器上获取触发event的传感器类型          int sensorType = event.type;          switch (sensorType)          {              case Sensor.TYPE\_ORIENTATION:                  // 获取与Y轴的夹角                  float yAngle = values[1];                  // 获取与Z轴的夹角                  float zAngle = values[2];                  // 气泡位于中间时（水平仪完全水平），气泡的X、Y座标                  int x = (show.back.getWidth() - show.bubble.getWidth()) / 2;                  int y = (show.back.getHeight() - show.bubble.getHeight()) / 2;                  // 如果与Z轴的倾斜角还在最大角度之内                  if (Math.abs(zAngle) <= MAX\_ANGLE)                  {                      // 根据与Z轴的倾斜角度计算X座标的变化值（倾斜角度越大，X座标变化越大）                      int deltaX = (int) ((show.back.getWidth() - show.bubble                          .getWidth()) / 2 \* zAngle / MAX\_ANGLE);                      x += deltaX;                  }                  // 如果与Z轴的倾斜角已经大于MAX\_ANGLE，气泡应到最左边                  else if (zAngle > MAX\_ANGLE)                  {                      x = 0;                  }                  // 如果与Z轴的倾斜角已经小于负的MAX\_ANGLE，气泡应到最右边                  else                  {                      x = show.back.getWidth() - show.bubble.getWidth();                  }                  // 如果与Y轴的倾斜角还在最大角度之内                  if (Math.abs(yAngle) <= MAX\_ANGLE)                  {                      // 根据与Y轴的倾斜角度计算Y座标的变化值（倾斜角度越大，Y座标变化越大）                      int deltaY = (int) ((show.back.getHeight() - show.bubble                          .getHeight()) / 2 \* yAngle / MAX\_ANGLE);                      y += deltaY;                  }                  // 如果与Y轴的倾斜角已经大于MAX\_ANGLE，气泡应到最下边                  else if (yAngle > MAX\_ANGLE)                  {                      y = show.back.getHeight() - show.bubble.getHeight();                  }                  // 如果与Y轴的倾斜角已经小于负的MAX\_ANGLE，气泡应到最右边                  else                  {                      y = 0;                  }                  // 如果计算出来的X、Y座标还位于水平仪的仪表盘内，更新水平仪的气泡座标                  if (isContain(x, y))                  {                      show.bubbleX = x;                      show.bubbleY = y;                  }                  // 通知系统重回MyView组件                  show.postInvalidate();                  break;          }      }        // 计算x、y点的气泡是否处于水平仪的仪表盘内      private boolean isContain(int x, int y)      {          // 计算气泡的圆心座标X、Y          int bubbleCx = x + show.bubble.getWidth() / 2;          int bubbleCy = y + show.bubble.getWidth() / 2;          // 计算水平仪仪表盘的圆心座标X、Y          int backCx = show.back.getWidth() / 2;          int backCy = show.back.getWidth() / 2;          // 计算气泡的圆心与水平仪仪表盘的圆心之间的距离。          double distance = Math.sqrt((bubbleCx - backCx) \* (bubbleCx - backCx)              + (bubbleCy - backCy) \* (bubbleCy - backCy));          // 若两个圆心的距离小于它们的半径差，即可认为处于该点的气泡依然位于仪表盘内          if (distance < (show.back.getWidth() - show.bubble.getWidth()) / 2)          {              return true;          }          else          {              return false;          }      }  } |

MyView.java

[?](http://www.2cto.com/kf/201501/367225.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | public class MyView extends View  {      // 定义水平仪仪表盘图片      Bitmap back;      // 定义水平仪中的气泡图标      Bitmap bubble;      // 定义水平仪中气泡 的X、Y座标      int bubbleX, bubbleY;        public MyView(Context context, AttributeSet attrs)      {          super(context, attrs);          // 加载水平仪图片和气泡图片          back = BitmapFactory.decodeResource(getResources()              , R.drawable.back);          bubble = BitmapFactory              .decodeResource(getResources(), R.drawable.bubble);      }        @Override      protected void onDraw(Canvas canvas)      {          super.onDraw(canvas);          // 绘制水平仪表盘图片          canvas.drawBitmap(back, 0, 0, null);          // 根据气泡座标绘制气泡          canvas.drawBitmap(bubble, bubbleX, bubbleY, null);      }  } |

# Android 通知栏功能

通知栏主要是为了及时的通知用户消息，配合推送来进行使用是最常见的情况，也可以对下载，音乐播放器等的进行使用，比如酷我音乐通知栏可以切换播放歌曲，暂停等功能。

使用方式：

# Android 推送

Android 的推送是为了实现当产品有新的重大更新，或是有新的优惠或是活动，等的时候及时 的通知。让用户提高对于我们产品的粘性。提高活跃度。

现在使用最多的几个推送：

百度

极光推送

友盟

等

使用方式也是按接入文档来进行。

# Android 地图开发：

关于地图开发：

百度地图开发：

高德地图开发：

开发流程基本相同，最新的版本的使用更加简单，按流程接入就好，难点在于实现自定义的遮罩，已经浮层，路径绘制，对导航进行自定义出来等。

很多人说高德地图比百度地图更加精确，这个可以自己测试，如果要精确定位检测是否打开GPS.

# 20:、Android 第三广告平台：

# Android 即时通讯：

即时通讯就是实现及时应答的功能，通过Tcp，udp，或socket进行连接，进行实时应答的功能。

腾讯的微信，QQ，陌陌，等等就是实现这样的效果。

关于这个功能的实现有很多方式：

1. XMPP协议实现
2. Mqtt协议实现
3. Socket实现

# Android 视频开发：

# Android 短信开发：

# Android 音频开发

# Android蓝牙开发

# Android 无线网络开发：

# Android 联系人开发：

# Android各种第三方框架的使用：

# Android listView的优化，内存优化，Android代码的重构

# Android 各种错位问题的处理

# Android 下载管理器的书写

# Android 手机适配

# Android 6.0权限的管理

# Android 材料设计语言，已经v7中新添加的控件等

# Java的一些常用的设计模式

# 单例模式

# 工厂模式

# 适配器模式

# 观察者模式

# Build（建造者）模式

# 模板模式

# 等等。。

# Java的基础算法：

# 二分查找，冒泡排序，等等。。。

# Android jni +ndk 开发

Android的jni开发通俗的讲就是java native interface ，java本地接口调用，就是我们做本地方法的实现。

为什么要做本地方法的实现呢？

# Android aidl开发，实现进程间通信

关于Android Aidl开发，Aidl就是Android interface definition language (接口定义语言)

主要是为了实现进程间通信，或者是远程通信。

关于这个功能再Android的系统里边就有很多的实现。

有开发过蓝牙或者WIFI应用的朋友肯定都知道，要去操作它必须先获得一个管理类，比如WIFI的管理类是WifiManager，通过getSystemService(Context.WIFI\_SERVICE)就可以得到wifi的管理权限，这个提供了很多的方法可以让用户去操作它，比如打开wifi可以调用setWifiEnabled(true)方法。那这个Manager到底做了什么工作呢？是怎样实现打开wifi的呢？其实这个Manager只是一个管理类，真正干活的另有其人，是一个叫WifiService的系统服务。在Android系统中有很多的Manager，wifi的管理类叫WifiManager，蓝牙的管理类叫BluetoothManager，但是，只要有xxxManager.java，就会有Ixxx.aidl，并且有xxxService.java。这个aidl类就是实现Manager和Service通信的桥梁。

对于Android手机来说，当开机以后整个手机的系统其实就是一个大大的应用，我们平常写的应用都运行在这个应用上边，系统中不管是蓝牙，通讯录，等等都是他里边的应用，我们要使用它里边的功能就等于是在实现手机上应用之间的通信。

文章地址：<http://blog.csdn.net/u012808234/article/details/51312491>

# Android xml解析方式及比较

在Android中，常见的XML解析器分别为SAX解析器、DOM解析器和PULL解析器，下面，详细介绍。

**SAX解析器：**

SAX(Simple API for XML)解析器是一种基于事件的解析器，它的核心是事件处理模式，主要是围绕着事件源以及事件处理器来工作的。当事件源产生事件后，调用事件处理器相应的处理方法，一个事件就可以得到处理。在事件源调用事件处理器中特定方法的时候，还要传递给事件处理器相应事件的状态信息，这样事件处理器才能够根据提供的事件信息来决定自己的行为。

SAX解析器的优点是解析速度快，占用内存少。非常适合在Android移动设备中使用。

**DOM解析器：**

DOM是基于树形结构的的节点或信息片段的集合，允许开发人员使用DOM API遍历XML树、检索所需数据。分析该结构通常需要加载整个文档和构造树形结构，然后才可以检索和更新节点信息。

由于DOM在内存中以树形结构存放，因此检索和更新效率会更高。但是对于特别大的文档，解析和加载整个文档将会很耗资源。

**PULL解析器：**

PULL解析器的运行方式和SAX类似，都是基于事件的模式。不同的是，在PULL解析过程中，我们需要自己获取产生的事件然后做相应的操作，而不像SAX那样由处理器触发一种事件的方法，执行我们的代码。PULL解析器小巧轻便，解析速度快，简单易用，非常适合在Android移动设备中使用，Android系统内部在解析各种XML时也是用PULL解析器。

可以发现每个解析方式都是有他的优点以及确定，具体使用还需要视情况而定，也可以使用第三方的解析框架。

# Android代码混淆

Java 是一种跨平台的、解释型语言，Java 源代码编译成中间”字节码”存储于 class 文件中。由于跨平台的需要，Java 字节码中包括了很多源代码信息，如变量名、方法名，并且通过这些名称来访问变量和方法，这些符号带有许多语义信息，很容易被反编译成 Java 源代码。为了防止这种现象，我们可以使用 Java 混淆器对 Java 字节码进行混淆。  
  
混淆就是对发布出去的程序进行重新组织和处理，使得处理后的代码与处理前代码完成相同的功能，而混淆后的代码很难被反编译，即使反编译成功也很难得出程序的真正语义。被混淆过的程序代码，仍然遵照原来的档案格式和指令集，执行结果也与混淆前一样，只是混淆器将代码中的所有变量、函数、类的名称变为简短的英文字母代号，在缺乏相应的函数名和程序注释的况下，即使被反编译，也将难以阅读。同时混淆是不可逆的，在混淆的过程中一些不影响正常运行的信息将永久丢失，这些信息的丢失使程序变得更加难以理解。  
  
混淆器的作用不仅仅是保护代码，它也有精简编译后程序大小的作用。由于以上介绍的缩短变量和函数名以及丢失部分信息的原因， 编译后 jar 文件体积大约能减少25% ，这对当前费用较贵的无线网络传输是有一定意义的。

# Android 热更新，插件化apk，热修复的原理及使用，第三方热修复的框架

关于这个知识点，上边的功能其实基于的原理都是相同的。

热更新：是为了实现应用可以实时更新，而不需要不停的发布新的线上版本。当应用有小的修改的时候，通过这个技术来实现功能的替换或是修改调整。

插件化Apk：关于插件，已经在各大平台上出现过很多，eclipse插件、chrome插件、3dmax插件，所有这些插件大概都为了在一个主程序中实现比较通用的功能，把业务相关或者让可以让用户自定义扩展的功能不附加在主程序中，主程序可在运行时安装和卸载。

在android如何实现插件也已经被广泛传播，实现的原理都是实现一套插件接口，把插件实现编成apk或者dex，然后在运行时使用DexClassLoader动态加载进来。

插件化Apk就是为了减小Apk的大小，而把一个应用的不同功能模块打包成一个Apk文件。然后通过被安装的Apk来加载这几个Apk文件，插件化在运行流畅度上有比较大的问题，个人看法。使用过支付的人应该知道，支付宝就是使用插件化来加载的，可以发现小的版本支付宝基本上都不需要重新下载，只需要更新那几个插件包就可以实现替换。

那么关于这个热修复，以及插件化实现的原理，就是通过ClassLoader来加载你所修改或是更新好的插件，然后在应该中调用。

具体看下边资料：

插件原理即实现：<http://www.alloyteam.com/2014/04/android-cha-jian-yuan-li-pou-xi/>

热修复：http://www.tuicool.com/articles/367RBvY

# Android NFC功能开发

NFC手机相比普通手机来说，有以下3个附加功能：   
　　1.可以当成POS机来用，也就是“读取”模式    
　　2.可以当成一张卡来刷，也就是NFC技术最核心的移动支付功能   
　　3.可以像蓝牙、Wi-Fi一样做点对点通信

Near  Field Communication (NFC) 为一短距离无线通信技术，通常有效通讯距离为4厘米以内。NFC工作频率为13.65 兆赫兹，通信速率为106 kbit/秒到 848kbit/秒。

NFC通信总是由一个发起者 (initiator)和一个接受者(target)组成。通常initiator 主动发送电磁场(RF)可以为被动式接受者(passive target)提供电源。其工作的基本原理和收音机类似。正是由于被动式接受者可以通过发起者提供电源，因此target 可以有非常简单的形式,比如标签，卡，sticker 的形式。

NFC 也支持点到点的通信(peer to peer) 此时参与通信的双方都有电源支持。

和其它无线通信方式如Bluetooth相比，NFC 支持的通信带宽和距离要小的多，但是它成本低，如价格标签可能只有几分钱，也不需要配对，搜寻设备等，通信双方可以在靠近的瞬间完成交互。

在[**Android**](http://lib.csdn.net/base/15) NFC 应用中，Android手机通常是作为通信中的发起者，也就是作为NFC的读写器。Android手机也可以模拟作为NFC通信的接受者且从Android 2.3.3起也支持P2P通信。

# Android使用第三方字体，字体文件 (.TTF)

关于第三方字体,他是用来使应用看起来；更加符合用户的观看习惯或是突出产品的特性等等。比如要漫画里边字体可能是娃娃字体，或是火星文字体

看下图：



使用第三方字体也很简单，把字体文件下载以后，放到assets文件夹下，然后通过

Typeface typeface = Typeface.createFromAsset(“参数一AssetsManager”，“路径”)；

然后给TextView设置，或是给Paint设置都可以。

# 第三方登录，支付等等

第三方登录通常是使用现在使用范围最广的用户群的账号来进行登录，通过第三方认证登录以后在自己应用里边生成对应的账号，来实现账号绑定，以及减少因注册减少的用户量。

现在使用最多的有一下几个：

QQ

新浪

人人

支付宝

微信

等等吧。

要做第三方的登录只要根据官方的文档来进行接入就可以了。

关于第三方支付这一块，是为了提升支付的成功率：

第三方的支付现在分很多种：

支付宝

微信

QQ钱包

银联

充值卡

手机短信支付

等等。

相应的接入方式官方都有，有问题可以查查百度，很多人都做过了。

# Android 反射

　反射主要是指程序可以访问、检测和修改它本身状态或行为的一种能力。在计算机科学领域，反射是一类应用，它们能够自描述和自控制。这类应用通过某种机制来实现对自己行为的描述和检测，并能根据自身行为的状态和结果，调整或修改应用所描述行为的状态和相关的语义。

 　　在Java中的反射机制，被称为Reflection（大家看到这个单词，第一个想法应该就是去开发文档中搜一下了）。它允许运行中的Java程序对自身进行检查，并能直接操作程序的内部属性或方法。Reflection机制允许程序在正在执行的过程中，利用Reflection APIs取得任何已知名称的类的内部信息，包括：package、 type parameters、 superclass、 implemented interfaces、 inner classes、 outer classes、 fields、 constructors、 methods、 modifiers等，并可以在执行的过程中，动态生成Instances、变更fields内容或唤起methods。

　　好，了解这些，那我们就知道了，我们可以利用反射机制在Java程序中，动态的去调用一些protected甚至是private的方法或类，这样可以很大程度上满足我们的一些比较特殊需求。你当然会问，反射机制在[**Android**](http://lib.csdn.net/base/15)平台下有何用处呢？

　　我们在进行Android程序的开发时，为了方便调试程序，并快速定位程序的错误点，会从网上下载到对应版本的Android SDK的源码（这里给大家提供一个2.3.3版本的下载链接）。你会发现很多类或方法中经常加上了“@hide”注释标记，它的作用是使这个方法或类在生成SDK时不可见，那么我们的程序可能无法编译通过，而且在最终发布的时候，就可能存在一些问题。

　　那么，对于这个问题，第一种方法就是自己去掉Android源码中的"@hide"标记，然后重新编译生成一个SDK。另一种方法就是使用Java反射机制了，可以利用这种反射机制访问存在访问权限的方法或修改其域。