四大基本组件

Activity：应用程序的入口，用户界面需要继承Activity基类。其主要实现界面显示、接收事件。对于Activity之间的页面跳转和数据传输可以通过Intent。

Activity的生命周期包括七个回调方法：

onCreate：Activity第一次被创建时调用，方法仅被调用一次。

onStart：启动Activity被调用，由不可见到到可见时被调用。

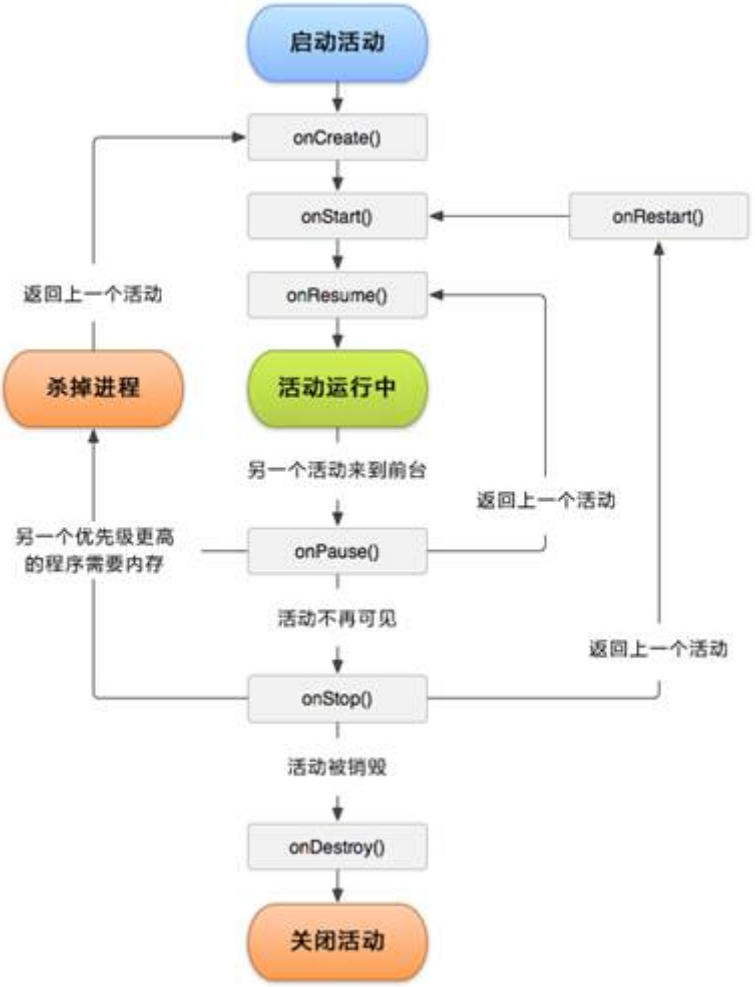
onResume：恢复Activity时被调用，onStart执行时，一定会执行该方法，并且此时Activity位于栈顶。

onPause：暂停Activity被调用，启动新的Activity。

onStop：停止Activity被调用，且在Activity完全不可见的情况下被调用。

onRestart：重新恢复Activity被调用，在onStop停止之后调用。

onDestroy：销毁Activity时被调用，且在程序停止时被调用。该方法与onCreate类似，仅调用一次。



Activity的四种状态：

活动状态：Activity位于栈顶，用户可见。
暂停状态：Activity不再位于栈顶，但用户可见。如Activity上出现对话框。
停止状态：Activity不位于栈顶，且用户不可见。
销毁状态：Activity被移除栈区。

Activity的启动模式：
Standard：默认模式，系统始终会Activity创建一个新的实例，并把Activity添加到栈顶，而不在乎Activity是否存在于栈中或栈顶。
singleTop：当Activity位于栈顶时，系统不会创建新的Activity实例，而是直接复用已有的Activity实例。
singleTask：若Activity已经存在于栈中，系统不会创建新的Activity实例。只有Activity不存在，系统才会创建Activity的实例。
singleInstance：当Activity已经存在于任何栈中，系统不会创建Activity的实例，只有Activity不存在与任何栈中，系统才创建Activity的实例。（前三个属于在一个栈中，而第四个可以跨栈）。

Activity的之间的跳转问题：
1. 显性Intent

```
case R.id.bt_secButton:
Intent intent=new Intent(FirstActivity.this,ScrollingActivity.class);
startActivity(intent);
break;
case R.id.bt_thirButton:
Bundle bundle=new Bundle();
bundle.putCharSequence("T","TTTTTTTTT");
bundle.putCharSequence("A","SSSSSSSSS");
Intent intentThir=new Intent(FirstActivity.this,ThirdActivity.class);
//intentThir.putExtra("T","你好");
//intentThir.putExtra("A","中国");
intentThir.putExtras(bundle);
```

```
startActivity(intentThir);
break;
```

2. 隐形Intent

```
<activity
   android:name=".ScrollingActivity">
   <intent-filter>
  <action android:name="com.example.android.testintent.ACTION_START" />
  <category android:name="android.intent.category.DEFAULT"/>
  </intent-filter>
</activity>

Intent intent=new Intent("com.example.android.testintent.ACTION_START");
startActivity(intent);
```

或：

```
<activity
android:name=".ScrollingActivity"
android:label="@string/title_activity_scrolling"
android:theme="@style/AppTheme.NoActionBar">
<intent-filter>
<action android:name="com.example.android.testintent.ACTION_START" />
<category android:name="android.intent.category.DEFAULT" />
<category android:name="com.example.MY_CATEGORY" />
</intent-filter>
</activity>


Intent  intent=new Intent("com.example.android.testintent.ACTION_START");
intent.addCategory("com.example.MY_CATEGORY");
```

startActivity(intent);

Intent:

1、Action：该activity可以执行的动作

该标识用来说明这个activity可以执行哪些动作，所以当隐式intent传递过来action时，如果跟这里<intent-filter>所列出的任意一个匹配的话，就说明这个activity是可以完成这个intent的意图的，可以将它激活！！！！

常用的Action如下所示：

ACTION_CALL activity 启动一个电话.

ACTION_EDIT activity 显示用户编辑的数据.

ACTION_MAIN activity 作为Task中第一个Activity启动

ACTION_SYNC activity 同步手机与数据服务器上的数据.

ACTION_BATTERY_LOW broadcast receiver 电池电量过低警告.

ACTION_HEADSET_PLUG broadcast receiver 插拔耳机警告

ACTION_SCREEN_ON broadcast receiver 屏幕变亮警告.

ACTION_TIMEZONE_CHANGED broadcast receiver 改变时区警告.

两条原则：

一条<intent-filter>元素至少应该包含一个<action>，否则任何Intent请求都不能和该<intent-filter>匹配。

如果Intent请求的Action和<intent-filter>中个任意一条<action>匹配，那么该Intent就可以激活该activity(前提是除了action的其它项也要通过)。

两条注意：

如果Intent请求或<intent-filter>中没有说明具体的Action类型，那么会出现下面两种情况。

如果<intent-filter>中没有包含任何Action类型，那么无论什么Intent请求都无法和这条<intent-filter>匹配。

反之，如果Intent请求中没有设定Action类型，那么只要<intent-filter>中包含有Action类型，这个Intent请求就将顺利地通过<intent-filter>的行为测试。

2、Category：于指定当前动作（Action）被执行的环境
即这个activity在哪个环境中才能被激活。不属于这个环境的，不能被激活。

常用的Category属性如下所示：
CATEGORY_DEFAULT：Android系统中默认的执行方式，按照普通Activity的执行方式执行。表示所有intent都可以激活它
CATEGORY_HOME：设置该组件为Home Activity。
CATEGORY_PREFERENCE：设置该组件为Preference。
CATEGORY_LAUNCHER：设置该组件为在当前应用程序启动器中优先级最高的Activity，通常为入口ACTION_MAIN配合使用。
CATEGORY_BROWSABLE：设置该组件可以使用浏览器启动。表示该activity只能用来浏览网页。
CATEGORY_GADGET：设置该组件可以内嵌到另外的Activity中。

注意：
如果该activity想要通过隐式intent方式激活，那么不能没有任何category设置，至少包含一个android.intent.category.DEFAULT

3、Data 执行时要操作的数据
在目标<data/>标签中包含了以下几种子元素，他们定义了url的匹配规则：
android:scheme 匹配url中的前缀，除了“http”、“https”、“tel”...之外，我们可以定义自己的前缀
android:host 匹配url中的主机名部分，如“google.com”，如果定义为“*”则表示任意主机名
android:port 匹配url中的端口
android:path 匹配url中的路径

在XML中声明可以操作的data域应该是这样的：

```
<activity android:name=".TargetActivity">
<intent-filter>
    <action android:name="com.scott.intent.action.TARGET"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:scheme="scott" android:host="com.scott.intent.data"
android:port="7788" android:path="/target"/>
</intent-filter>
</activity>
```

注意：

这个标识比较特殊，它定义了执行此Activity时所需要的数据，也就是说，这些数据是必须的！所有如果其它条件都足以激活该Activity，但intent却没有传进来指定类型的Data时，就不能激活该activity！

广播Broadcast

BroadcastReceiver：接收系统的广播消息，实现系统中组件间的通信。

标准广播：所有广播接收器都会在同一时刻接收到广播消息，属于异步执行。

有序广播：同一时刻仅有一个接收器接收到广播消息，属于同步执行。

动态注册

onReceive 是当广播到来时，就会得到执行。

```
    private IntentFilter intentFilter;

    private NetworkChangeReceiver networkChangeReceiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        intentFilter = new IntentFilter();
        intentFilter.addAction("android.net.conn.CONNECTIVITY_CHANGE");
        networkChangeReceiver = new NetworkChangeReceiver();
        registerReceiver(networkChangeReceiver, intentFilter);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(networkChangeReceiver);
    }
```

静态注册

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    ......
    <receiver android:name=".BootCompleteReceiver" >
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
```

自定义广播

1. 继承BroadcastReceiver，重写onReceive方法。
2. 在xml中填写注册信息。

```
    ......
    <receiver android:name=".MyBroadcastReceiver">
        <intent-filter>
            <action android:name="com.example.broadcasttest. MY_BROADCAST"/>
        </intent-filter>
    </receiver>
```

abortBroadcast：终止广播。

本地广播：只能够在应用程序内部传递，并且广播接收器只能接收本应用发出的广播信息，避免了应用之间的传递，导致不安全。
本地广播主要使用LocalBroadcastManager对广播进行管理，

1. 获得实例

localBroadcastManager = LocalBroadcastManager.getInstance(this);

2. 发送广播

localBroadcastManager.sendBroadcast(intent); // 发送本地广播

3. 注册本地广播

intentFilter = new IntentFilter();
intentFilter.addAction("com.example.broadcasttest.LOCAL_BROADCAST");
localReceiver = new LocalReceiver();
localBroadcastManager.registerReceiver(localReceiver, intentFilter);

接收者Provider

ContentProvider：提供数据查询接口，允许其他应用程序公开访问或查询数据。实现数据共享。

实现不同应用程序之间的数据共享。
ContentResolver使用insert、update、delete和query，与SQLiteDatab的用法相似。

Uri主要有权限（authority）和路径（path）两部分组成。
Authority：主要以程序包名的方式命名。如com.exampler.app.provider，红色是包名，整体就是authority。
Path：主要对同一应用的不同表做区分，常添加在authority的后面。/table
如：　content://com.example.app.provider/table
com.example.app 是程序包名；

com.example.app.provider 是authority

MIME类型：

1. 必须以vnd开头
2. 如果内容URI以路径结尾，则后接：android.cursor.dir/，如果内容URI以id结尾，则后接：android.Cursor.item/
3. 最后接：vnd.<authority>.<path>
4.

所以content://com.exampler.app.provider/table1的MIME为：
vnd.android.cursor.dir/vnd.com.exampler.app.provider.table1
所以content://com.exampler.app.provider/table1/1的MIME为：
vnd.android.cursor.item/vnd.com.exampler.app.provider.table1

Service：没有用户界面，后台运行的，具有独立生命周期。
Service 不能操作UI的更新，可以利用“消息处理机制”

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

    tv_txt = (TextView) findViewById(R.id.tv_txt);
    bt_bt = (Button) findViewById(R.id.bt_bt);

    bt_bt.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Message message = new Message();
            message.what = 0x123;
            handler.sendMessage(message);
```

```
        }
    });
}

private Handler handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        super.handleMessage(msg);

        switch (msg.what) {
            case 0x123:
                tv_txt.setText("Nice to me you");
        }
    }
};
```

解决异步消息处理机制：
Message：线程之间传递消息；
Handler：发送和处理消息；
MessageQueue：存放所有Handler消息；
Looper： 取出消息，传递到handlerMessage()。

系统框架
系统主要建立在Linux系统上，内核为2.6版本。Android系统的体系架构主要由四层组成（从低到高）：
Linux内核：提供安全、内存管理、进程管理等核心系统服务。
Android系统运行库（函数库与运行库）：函数库主要包括C/C++库，主要由应用程序框架来调用函数库，应用程序不能直接调用该库。运行库主要包括Dalvik虚拟机和Android核心库，虚拟机主要负责运行Android应用程序，核心库提供了Java语言核心库所能使用的绝大多数功能。

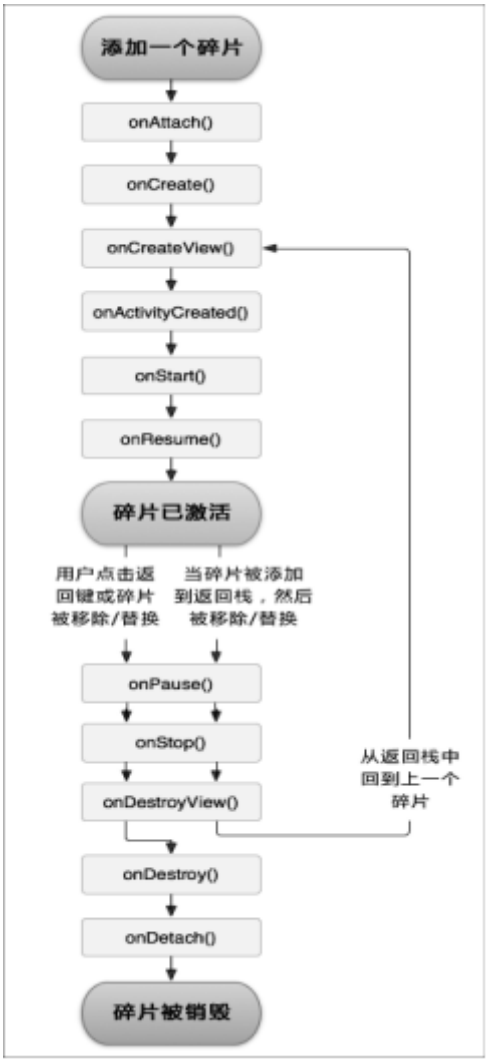
应用程序框架：提供给应用程序开发大量的API，应用程序主要调用框架的函数，来实现一些主要的功能。

应用程序：主要实现手机的各种功能。

碎片

碎片（fragment）是一种嵌入在活动当中的UI片段。

生命周期：



onAttach(): 当碎片和活动建立关联的时候调用;

onCreateView(): 为碎片创建视图时调用;

onActivityCreated(): 确保与碎片相关联的活动一定已经创建完毕的时候调用

onDestroyView(): 当与碎片关联的视图被移除的时候调用

onDetach(): 当碎片和活动解除关联的时候调用

动态添加碎片

1. 开启事务

```
AnotherRightFragment fragment = new AnotherRightFragment();
FragmentManager fragmentManager = getFragmentManager();

FragmentTransaction transaction = fragmentManager.
beginTransaction();
```

2. Replace实现碎片切换

```
transaction.replace(R.id.right_layout, fragment);
transaction.commit()
```

碎片的返还栈

1.启动事务

```
AnotherRightFragment fragment = new AnotherRightFragment();
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction transaction = fragmentManager.
beginTransaction();
```

2. 碎片切换

```
transaction.replace(R.id.right_layout, fragment);
```

3. 将老碎片放入栈中, 准备返回

```
transaction.addToBackStack(null);
transaction.commit();
```

自定义碎片

1. 设计一个类似于activity的布局，作为碎片的显示页面。
2. 继承Fragment，重写onCreateView方法，并返回。

```
    View view=inflater.inflate(R.layout.fragment,null);
return view;
```

3. 在activity中加入

```
fragOne = new FragOne();
fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
fragmentTransaction.replace(R.id.ll_content, fragOne);
fragmentTransaction.commit();
```

持久化技术

针对手机不能显示/data/data文件夹，可以使用adb root和adb remount两个命令。

持久化: 将内存中的瞬时数据保存到存储设备中,保证数据在设备断电的情况下仍可以保存。

1. 文件存储：将数据内容不进行任何处理而保存到手机中。默认路径：/data/data/<packageName>/files/docu.txt。

openFIleOutput(String，Mode)，将数据存入到指定的文件中。

String：文件名；

Mode：操作模式MODE_PRIVATE（覆盖）MODE_APPEND（追加）

```
public void save(String data) {

    FileOutputStream fileOutputStream = null;
    BufferedWriter writer = null;
    try {
        fileOutputStream = openFileOutput("data", Context.MODE_PRIVATE);
        writer = new BufferedWriter(new OutputStreamWriter(fileOutputStream));
        writer.write(data);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (writer != null) {
```

```
            try {
                writer.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

public String load() {
    FileInputStream in = null;
    BufferedReader reader = null;
    StringBuilder builder = new StringBuilder();
    try {
        in = openFileInput("data");
        reader = new BufferedReader(new InputStreamReader(in));
        String line = "";

        while ((line = reader.readLine()) != null) {
            builder.append(line);
        }

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (reader != null) {
            try {
                reader.close();
            } catch (IOException e) {
                e.printStackTrace();
```

```
            }
        }
    }
    return builder.toString();
}
```

2. SharedPreferences存储：使用键值对来保存数据，类似于Map。默认路径: /data/data/<packageName>/shared_prefs/docus.xml。

- 获取Sharedpreferences对象
- 调用edit()方法来得到Sharedpreferences.Edit对象
- 向Sharedpreferences.Edit对象添加数据
- 使用commit()方法来提交数据，完成数据的存储。

```
SharedPreferences sharedPreferences = getSharedPreferences("data",
MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPreferences.edit();

editor.putString("name", "Tom");
editor.putInt("age", 29);
editor.putBoolean("married", false);

editor.commit();
```

读取数据

```
SharedPreferences pref=getSharedPreferences("data",MODE_PRIVATE);
String name=pref.getString("name","");
int age=pref.getInt("age",10);
boolean married=pref.getBoolean("married",false);
```

3. SQLite数据库存储：数据存放在sqlite数据库中。默认路径：/data/data/<packageName>/databases/

- 继承SQLiteOpenHelper抽象类，重写onCreate和onUpgrade
- 调用实例方法：getReadableDatabase或getWritableDatabase，创建数据库
- SQLiteDatabase 对象是数据库实例，用来操作执行语句。
- 构造方法用来创建数据库，SQLiteDatabase对表进行操作。

- 添加数据，insert

```
ContentValues values=new ContentValues();
values.put(“keyName”,”valueName”);
db.insert("tableName”, null, values);
values.clear();
```

- 更新数据，update

```
Update(“tableName”,values,”condition”,new String(){condition=values})
```

- 删除数据， delete

```
Delete(“tableName”,”condition”,new String(){condition=values});
```

- 查询数据 query

query()方法参数	对应 SQL 部分	描述
table	from table_name	指定查询的表名
columns	select column1, column2	指定查询的列名
selection	where column = value	指定 where 的约束条件
selectionArgs	-	为 where 中的占位符提供具体的值
groupBy	group by column	指定需要 group by 的列
having	having column = value	对 group by 后的结果进一步约束
orderBy	order by column1, column2	指定查询结果的排序方式

```
SQLiteDatabase db = dbHelper.getWritableDatabase();
// 查询Book表中所有的数据
Cursor cursor = db.query("Book", null, null, null, null, null, null);
if (cursor.moveToFirst()) {
while(cursor.moveToNext()) {
// 遍历Cursor对象,取出数据并打印
String name = cursor.getString(cursor.
getColumnIndex("name"));
String author = cursor.getString(cursor.
getColumnIndex("author"));
int pages = cursor.getInt(cursor.getColumnIndex
("pages"));
double price = cursor.getDouble(cursor.
getColumnIndex("price"));
```

添加数据的方法如下：

```
db.execSQL("insert into Book (name, author, pages, price) values(?, ?, ?, ?)",
            new String[] { "The Da Vinci Code", "Dan Brown", "454", "16.96" });
db.execSQL("insert into Book (name, author, pages, price) values(?, ?, ?, ?)",
            new String[] { "The Lost Symbol", "Dan Brown", "510", "19.95" });
```

更新数据的方法如下：

```
db.execSQL("update Book set price = ? where name = ?", new String[] { "10.99",
"The Da Vinci Code" });
```

删除数据的方法如下：

```
db.execSQL("delete from Book where pages > ?", new String[] { "500" });
```

查询数据的方法如下：

```
db.rawQuery("select * from Book", null);
```

可以看到，除了查询数据的时候调用的是 SQLiteDatabase 的 rawQuery()方法，其他的操作都是调用的 execSQL()方法。以上演示的几种方式，执行结果会和前面几小节中我们学习的 CRUD 操作的结果完全相同，选择使用哪一种方式就看你个人的喜好了。

通知的基本用法

1. 获取系统服务

```
NotificationManager nm = (NotificationManager)
```

```
getSystemService(Context.NOTIFICATION_SERVICE);
```

2. 创建notification对象

```
Notification notification = new Notification.Builder(getApplicationContext())
        .setContentTitle("title")
        .setContentText("text")
        .setSmallIcon(R.drawable.ic_launcher)
        .build();
```

3. 显示通知

```
notificationManage.notify(0, notification);
```

4. 取消通知

```
notificationManage.cancel(0);
```

高级技巧

1. 声音属性

```
Uri soundUri= Uri.fromFile(new File("/system/media/audio/basic_tone.ogg"));
notification.sound=soundUri;
```

2．震动属性

```
long[] vibrates={0,1000,1000,1000};//单列表示静止,双列表示震动
notification.vibrate=vibrates;
```

灯管属性

```
notification.ledARGB= Color.GREEN;  //灯光颜色
notification.ledOnMS=1000;         //灯亮时长
notification.ledOffMS=1000;         //灯灭时长
```

```
notification.flags=Notification.FLAG_SHOW_LIGHTS; //指定通知行为
```

短信接收：

```
Bundle bundle = intent.getExtras();
Object[] pdus = (Object[]) bundle.get("pdus");
SmsMessage[] message = new SmsMessage[pdus.length];

for(int i=0;i<message.length;i++){
    message[i]=SmsMessage.createFromPdu((byte[]) pdus[i]);
}

String address=message[0].getOriginatingAddress();
String fullMessage="";

for(SmsMessage message1:message){
    fullMessage+=message1.getMessageBody();
}
tv_sender.setText(address);
tv_content.setText(fullMessage);
```

发送短信

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage(  et_to.getText().toString(), null,
et_msgInput.getText().toString(), pi, null);
```

传感器的开发：

1. 获取SensorManager实例

SensorManager sensormanager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);

2. 获取传感器类型

Sensor sensor=sensormanager.getDefaultSensor(Sensor.TYPE_LIGHT);

3. 注册

Sensormanager.registerListener(listener, sensor, SensorManager.SENSOR_??);

4. 创建SensorEventListener 监听器，并重写onSensorChanged方法。

5. 在onDestroy中解除注册

sensorManager.unregisterListener(listener);

颜色：#00000000　　前两位（一位）是透明值：00-FF（不透明）
　　#0000　　接着两位（一位）是红色值：00-FF（最红）
　　接下来两位（一位）是绿色：00-FF（最绿）
　　最后两位（一位）是蓝色：00-FF（最蓝）

anim 设计组件的动画效果

alpha　渐变透明度动画效果　AlphaAnimation
scale　渐变尺寸伸缩动画效果　ScaleAnimation
translate　画面转换位置移动动画效果　TranslateAnimation
rotate　画面转移旋转动画效果　RotateAnimation

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">

    <translate
        android:duration="500"
        android:fromXDelta="0"
        android:fromYDelta="1000"
        android:toXDelta="0"
        android:toYDelta="0" />

</set>
```

Shape主要定义各种各样的形状

主要标签：

solid：填充

android:color 填充颜色

gradient 渐变

android:startColor和android:endColor分别为起始和结束颜色，

android:angle是渐变角度，必须为45的整数倍。

另外渐变默认的模式为android:type="linear"，即线性渐变，

可以指定渐变为径向渐变，android:type="radial"，

径向渐变需要指定半径android:gradientRadius="50"。

stroke 描边

android:radius为角的弧度，值越大角越圆。

我们还可以把四个角设定成不同的角度，

同时设置五个属性，则Radius属性无效

android:Radius="20dp"　　设置四个角的半径

android:topLeftRadius="20dp"　　设置左上角的半径
android:topRightRadius="20dp"　　设置右上角的半径
android:bottomLeftRadius="20dp"　设置右下角的半径
android:bottomRightRadius="20dp　设置左下角的半径

padding 间隔，设置四个方向的间隔大小。

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
   android:shape="rectangle">

   <gradient
      android:angle="90"
      android:endColor="#EBEBEB"
      android:startColor="#EAEAEA" />

   <corners
      android:bottomLeftRadius="5dp"
      android:bottomRightRadius="5dp"
      android:topLeftRadius="0dp"
      android:topRightRadius="0dp"
      />
</shape>
```

自定义Dialog

1. 设计一个layout布局，与activity差不多，作为Dialog的页面。
2. 继承Dialog类，重写onCreate方法。

3. 在activity中，添加代码，显示dialog。

```
Dialog dialog = new MyDialog(MainActivity.this, R.style.MyDialog);
WindowManager windowManager =getWindowManager();
Display display = windowManager.getDefaultDisplay();
WindowManager.LayoutParams lp = dialog.getWindow().getAttributes();
lp.width = (int)(display.getWidth());
lp.height=(int)display.getHeight()-735;
dialog.getWindow().setAttributes(lp);
dialog.show();


View view = getLayoutInflater().inflate(R.layout.dialog, null);
Dialog dialog = new Dialog(this, R.style.transparentFragmeWindowStyle);
dialog.setContentView(view, new
ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
ViewGroup.LayoutParams.WRAP_CONTENT));

Window window = dialog.getWindow();
window.setWindowAnimations(R.style.main_menu_animstyle);

WindowManager.LayoutParams wl = window.getAttributes();

wl.x = 0;
wl.y = getWindowManager().getDefaultDisplay().getHeight();

wl.width = ViewGroup.LayoutParams.MATCH_PARENT;
wl.height = ViewGroup.LayoutParams.WRAP_CONTENT;

dialog.onWindowAttributesChanged(wl);
dialog.setCanceledOnTouchOutside(true);
```

```
dialog.show();
```