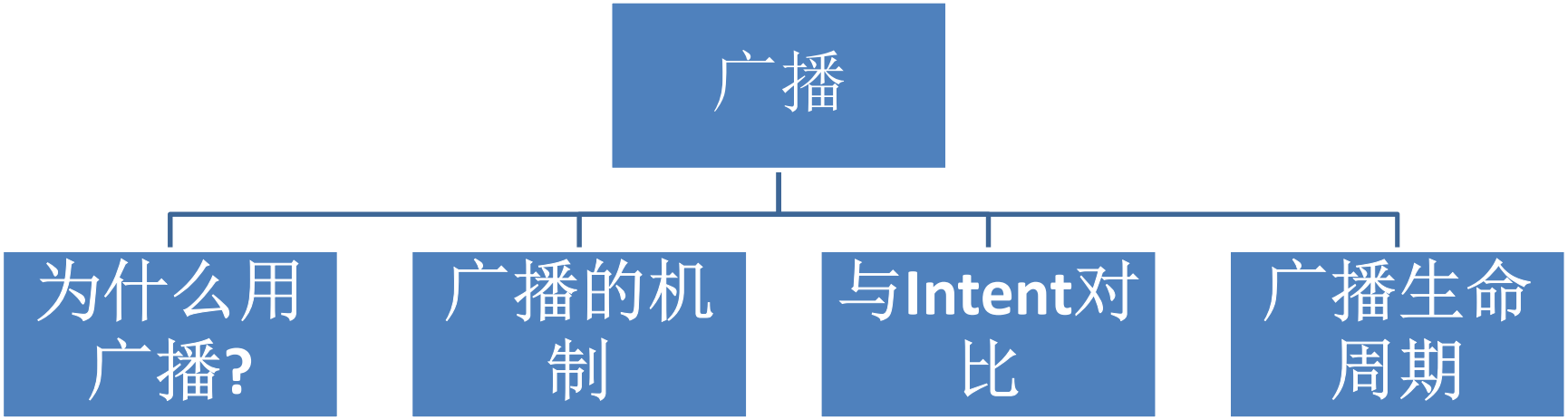
# #8 Broadcast的使用

# Broadcast




# Broadcast（广播）有何用？（1）

- 场景一

  如果你在等快递，你是每天24小时守候在快递网点等待你的包裹到来，还是你该干嘛干嘛去，等到包裹到了有人打电话通知你？

- 场景二

  我们需要程序在手机来电或接收短信时，显示电话号码，号码归属地，或者号码在我们黑名单中的话自动挂断或删除短信，要如何实现？

# Broadcast（广播）有何用？（2）

- 场景一相信答案很明确
- 场景二
  - 在Symbian、Windows Mobile中，应用若需要等待一个来电或短信，来实现显示归属地之类的功能，必须让自己的应用保证开机启动、潜伏在后台运行、监控相关事件。
  - 在Android平台中，考虑到广泛存在这类需求，在框架中设计了BroadcastReceiver。当发生这类事件时，系统会自动唤醒负责接收对应事件的Receiver，处理完事件后，Receiver就马上退出，这对手机有限的资源来说是一种极好的解决方案

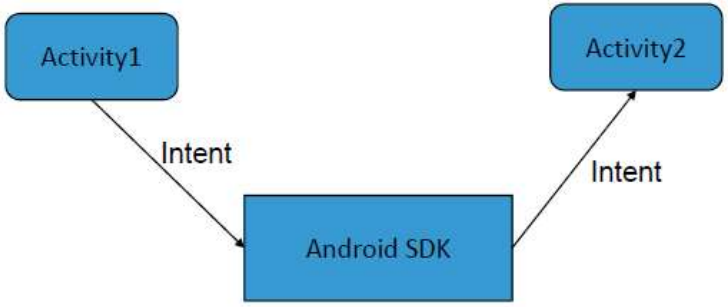
# Broadcast Receiver 简介

- 发送与接受广播类似于发布订阅者。
- Broadcast可以用作跨应用程序和正常用户流之外的消息传递系统。
- Broadcast组件提供了一种把Intent作为一个消息广播出去，由所有对其感兴趣的程序对其作出反应的机制。

# 还记得什么是Intent吗？

- Intent作为一种操作系统级别的消息传递机制，能够在不同的进程之间传递结构化消息。

- 不同Activity之间的通讯，属于不同UI线程之间的通讯，如果要在不同的Activity之间传递消息，则需要用到Intent。在Intent中告诉操作系统要选择哪个目标Activity进行实例化，并传递消息。




# Broadcast广播机制（1）

- **Android**广播机制包含三个基本要素：

1. Broadcast（广播）---发送广播；
2. BroadcastReceiver（广播接收器）---接收广播；
3. Intent（意图）---保存广播信息的媒介

# Broadcast广播机制（2）

- Broadcast是一种广泛运用的在应用程序之间传输信息的机制。

- BroadcastReceiver是对发送出来的Broadcast进行过滤，接收并响应的一类组件。

# Broadcast广播机制（3）

- 广播的详细过程

  发送广播：在需要发送信息的地方，把要发送的信息和用于过滤的信息(如Action、Category)装入一个Intent对象，调用*Context.sendBroadcast()* 方法把Intent对象以广播方式发送出去。

  接收广播：当Intent发送以后，所有已经注册的BroadcastReceiver会检查注册时的IntentFilter是否与发送的Intent相匹配，若匹配则就会调用BroadcastReceiver的*void onReceive(Context curContext, Intent broadcastMsg)*方法。

# BroadcastReceiver开发

- **BroadcastReceiver**开发过程如下

1.开发BroadcastReceiver子类，重写onReceive()

2. 注册BroadcastReceiver对象

3. 将要广播的消息封装到Intent中，调用方法发送出去

4. 通过IntentFilter对象过滤Intent，处理信息名称匹配的广播

# Broadcast--最简单的例程(1)

- 在A中发出一个自定义的广播:

```
Button mBtnTest =(Button) findViewById(R.id.btnTest);
mBtnTest.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent=new Intent("TheStringUsedAsFilter");
        sendBroadcast(intent);
    }
})
```

- 在A与B中实现自定义的接收广播类: MyReceiver.java

```
public class MyReceiver extends BroadcastReceiver{
        @Override
        public void onReceive(Context context,Intent intent)
        {
         Toast.makeText(context,"A Receiver",Toast.LENGTH_LONG).show();
         //Toast.makeText(context,"B Receiver",Toast.LENGTH_LONG).show();
        }
}
```

# Broadcast--最简单的例程(2)

- 配置A、B的文件**AndroidManifest.xml:**

用<Receiver>标签注册一个BroadcastReceiver，还需要有一个字符串作为filter，通过filter选择接收广播的类。

```
<receiver android:name=".MyBroadCast1"
      android:exported="true">
    <intent-filter>
        <action android:name="TheStringUsedAsFilter"/>
    </intent-filter>
</receiver>
```

➡receiver默认不接受外部广播，B接收A的广播要设置exported属性

➡要捕捉的信息名称为TheStringUsedAsFilter

# Broadcast--最简单的例程(3)

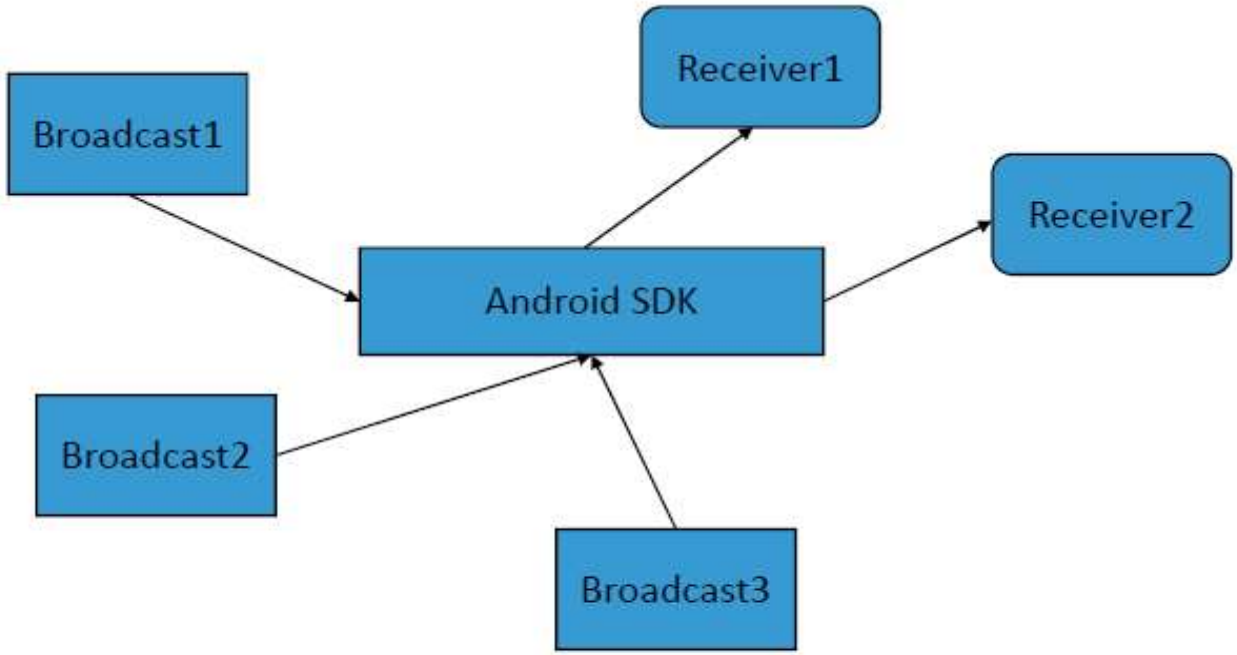
整体流程如下：

1. 在AndroidManifest.xml配置文件中，用<Receiver>标签注册一个BroadcastReceiver，还需要有一个字符串作为过滤filter，通过filter选择接收广播的类。

2. TestActivity.java中将filter字符串放入intent中，再通过广播发出去，等待系统接收。

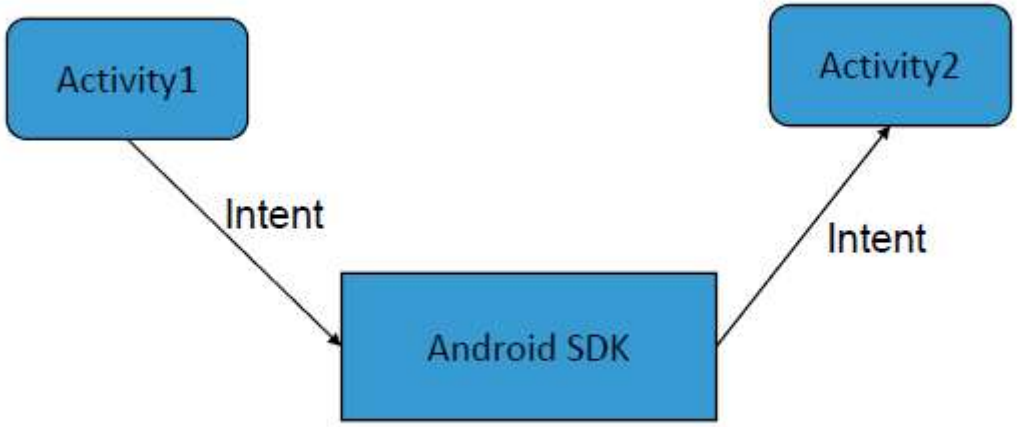
3. 系统通过xml文件，查找到对应的filter，映射到对应的BroadcastReceiver类。

# Broadcast机制图示




# Broadcast机制

- Broadcast机制与Intent消息机制的图是一致的。



# Broadcast机制与Intent机制

- 通过Intent实现Activity间的跳转有两种方式：
  1. 显式Intent：即在构造Intent对象时就指定接收者；
  2. 隐式Intent：即Intent的发送者在构造Intent对象时，并不知道也不关心接收者是谁，只通过filter来选择接收者，有利于降低发送者和接收者之间的耦合。
- 而Broadcast的实现机制，与通过隐式的Intent启动Activity的方式是类似的

# Broadcast的生命周期（1）

- 广播接收器仅在它执行这个方法时处于活跃状态。当onReceive()返回后，它即为失活状态。
- 拥有一个活跃状态的广播接收器的进程被保护起来而不会被杀死，但仅拥有失活状态组件的进程则会在其它进程需要它所占有的内存的时候随时被杀掉。所以，如果响应一个广播信息需要很长的一段时间，我们一般会将其纳入一个衍生的线程中去完成，而不是在主线程内完成它，从而保证用户交互过程的流畅。

# Broadcast的生命周期（2）

- 当系统通过filter字符串找到匹配的Receiver，onReceive方法就会被执行。
- onReceive 方法必须在5秒内执行完毕退出，否则会导致FC（Force Close强制关闭）。

# Broadcast的应用

- 通常来说，BroadcastReceiver用来更新content、启动service、更新UI或者通过notificationmanager在状态栏中提示。

- 5秒的限制保证了主要处理任务不会、也不应该在BroadcastReceiver中完成。

  如果需要执行大量任务，可以在BroadcastReceiver中启用Service去处理。

# 系统广播

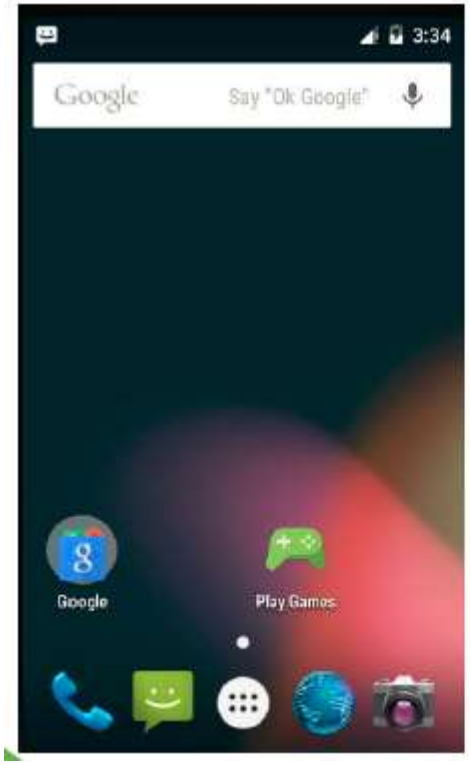
- 在Android操作系统中，有许多与手机相关的事件会对系统发送广播信息。当系统发出广播后，就会搜索是否注册了负责处理该广播的BroadcastReceiver。

- 因此，只要了解系统的广播类型，就可以实现很多手机自动服务功能。例如，收到新短信进行提示、手机来电时自动拒接等等。

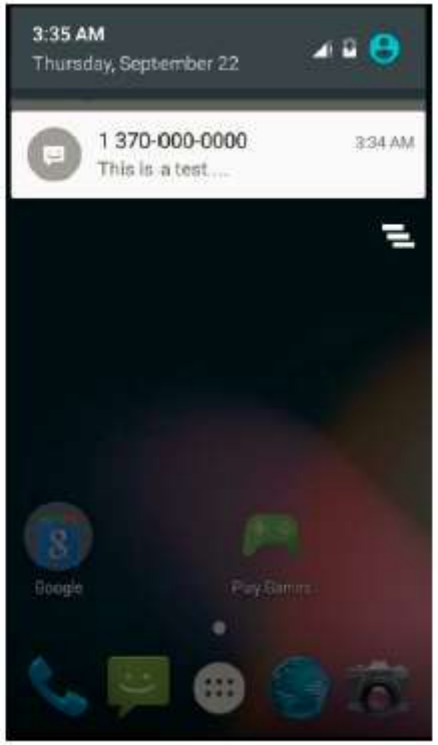
# 系统Broadcast的IntentFilter（部分）

ACTION_TIME_TICK //当前时间改变，每分钟都发送，不能通过组件声明来接收，只有通过Context.registerReceiver()方法来注册
ACTION_TIME_CHANGED//时间被设置
ACTION_TIMEZONE_CHANGED //时间区改变
ACTION_BOOT_COMPLETED //系统完成启动后，一次广播
ACTION_PACKAGE_ADDED //一个新应用包已经安装在设备上，数据包括包名（最新安装的包程序不能接收到这个广播）
ACTION_PACKAGE_CHANGED //一个已存在的应用程序包已经改变，包括包名
ACTION_PACKAGE_REMOVED //一个已存在的应用程序包已经从设备上移除，包括包名（正在被安装的包程序不能接收到这个广播）
ACTION_PACKAGE_RESTARTED //用户重新开始一个包，包的所有进程将被杀死，所有与其联系的运行时间状态应该被移除，包括包名（重新开始包程序不能接收到这个广播）
ACTION_PACKAGE_DATA_CLEARED //用户已经清楚一个包的数据，包括包名（清除包程序不能接收到这个广播）
ACTION_BATTERY_CHANGED //电池的充电状态、电荷级别改变，不能通过组建声明接收这个广播，只有通过Context.registerReceiver()注册
ACTION_UID_REMOVED //一个用户ID已经从系统中移除

# 系统广播例子--新短信提示




收到新短信，状态栏中有提示




拉下状态栏


点击查看详情

# 新短信提示--AndroidManifest.xml（1）

- 在AndroidManifest.xml里面，注册一个Receiver，并添加intent-filter节点，在节点下添加action，告诉系统这个Receiver用于处理哪些Broadcast（这里可以有多个action）

```
<receiver
    android:name=".SmsReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

# 新短信提示--AndroidManifest.xml（2）

- 定义一个继承BroadcastReceiver的子类SmsReceiver,对系统“收到短信”这个广播进行处理。

- 配置文件中
  - receiver名为”.SmsReceiver”
  - action名为”android.provider.Telephony.SMS_RECEIVED”
    (查看系统广播intent大全)

# 新短信提示--SmsReceiver.java（1）

- 定义一个继承BroadcastReceiver的子类
- 重写onReceive方法

```
public class SmsReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {

    }
}
```

# 新短信提示--SmsReceiver.java（2）

- 如果一个Receiver可以处理多个广播，则需要在代码中对各个Action进行判断，分别处理。

```
@Override
public void onReceive(Context context,Intent intent)
{
    if(intent.getAction().equals(mAction))
    {
        /*这里对Action进行判断，例如如果我们要对接受信息这个广播事件进行
        * 处理的话，则mAction="android.provider.Telephony.SMS.Received"
        */
    }
}
```

- 其中intent变量是由onReceive方法参数传入的，所以我们也可以看出，系统广播其实也是通过sendBroadcast(intent)发出的。
- 系统广播“收信息”中，还将短信内容附在Bundle中，随着intent传入到Receiver。

# 新短信提示--SmsReceiver.java（3）

- 获取短信

```
/*接收由Intent传来的数据*/
Bundle bundle = intent.getExtras();
/*判断Intent是否有资料*/
if (bundle != null) {
    //bundle通过key=pdus获取短信内容
    //这里的pdus是一个Object数组，因为当短信长度超出限制时会被自动拆分成多条短信
    Object[] pdus = (Object[]) bundle.get("pdus");
    //根据Object数组长度创建SmsMessage数组
    final SmsMessage[] messages = new SmsMessage[pdus.length];
    for (int i = 0; i < pdus.length; i++) {
        //通过SmsMessage的静态方法createFromPdu通过myOBJpdus创建SmsMessage对象
        messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
    }
}
```

# 新短信提示--SmsReceiver.java（4）

- 获取短信（API 23之后）可以发现，上面的代码createFromPdu上有一条横线，即：

'createFromPdu(byte[])' is deprecated less... (Ctrl+F1)

This inspection reports where deprecated code is used in the specified inspection scope.

- API 23之后的参考代码如下：

```
Object [] pdus=(Object[]) bundle.get("pdus");
String format=intent.getStringExtra("format");
SmsMessage[] messages=new SmsMessage[pdus.length];
for(int i=0;i<messages.length;i++) {
    byte[] sms=(byte[]) pdus[i];
    messages[i]=SmsMessage.createFromPdu(sms,format);
}
```

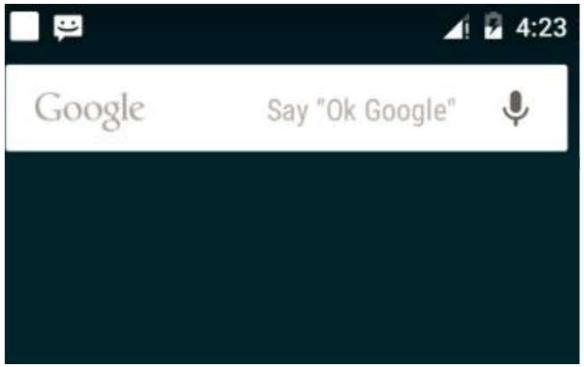
# 新短信提示--SmsReceiver.java（5）

- *SmsMessage*对象可以通过*getDisplayOriginatingAddress()*,*getDisplayMessageBody()*方法分别获得发送人手机号和短信内容。

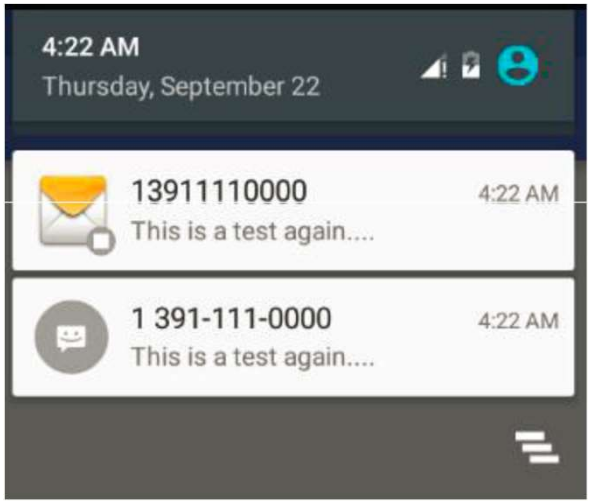
- 注意: 如果长信息被拆成几个短信息时，发信人手机号会被保存在*SmsMessage*数组的最后一个元素中.

# 新短信提示– Notification（1）

- 如何在状态栏显示提示信息？--Notification




图一




图二

# 新短信提示– Notification（2）

- Notification可以提供持久的通知，位于手机最上层的状态通知栏中。用手指按下状态栏，并从手机上方向下滑动，就可以打开状态栏查看提示消息。开发Notification主要涉及以下3个类：
  1. Notification.Builder:用于动态的设置Notification的一些属性。
  2. NotificationManager:负责将Notification在状态显示出来和取消。
  3. Notification： 设置Notification的相关属性。

在API14以后，setLatestEventInfo()函数被废弃，因此要想新建通知栏消息，就需要借助Notification的内部类Builder，该构造方法接收一个参数Context。

# 新短信提示– Notification（3）

- Notification 具体代码

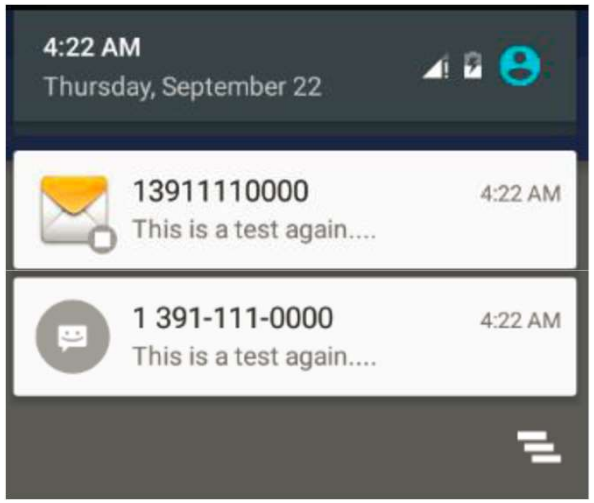
```
//获取状态通知栏管理
 NotificationManager manager=(NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
 //实例化通知栏构造器 Notification.Builder
 Notification.Builder builder=new Notification.Builder(context);
 //对builder进行配置
 builder.setContentText(sender)    //设置通知栏标题:发件人
         .setContentText(msg)      //设置通知栏显示内容
         .setTicker("您有一条新短信")    //通知首次出现在通知栏,带上升动画效果的
         .setPriority(Notification.PRIORITY_DEFAULT)       //设置通知优先级
         .setLargeIcon(bm)                                 //设置大ICON
         .setWhen(System.currentTimeMillis())              //通知产生的时间，会在通知信息里显
示，一般是系统获取到的时间
          .setSmallIcon(R.drawable.icon)
          .setAutoCancel(false)                          //设置这个标志当用户单击面板就可以将
通知取消

 //绑定intent，点击图标能够进入某activity
 Intent mInent=new Intent(context,MainActivity.class);
 PendingIntent mPendingIntent=PendingIntent.getActivity(context,0,mInent,0);
 builder.setContentIntent(mPendingIntent);
 //绑定Notification，发送通知请求
 Notification notify=builder.build();
 manager.notify(0,notify);
```

# 新短信提示– Notification（4）

- 点击notification，就可以跳转到我们intent中指定的activity。

```
//绑定intent，点击图标能够进入某activity
 Intent mInent=new Intent(context,MainActivity.class);
 PendingIntent mPendingIntent=PendingIntent.getActivity(context,0,mInent,0);
 builder.setContentIntent(mPendingIntent);
```




- 如果我们需要在目标activity显示短信内容等信息时，则需要在解析短信的时候，将短信内容存入bundle，再赋给intent，并在目标activity中解析 bundle。

# 新短信提示– PendingIntent(1)

- pendingIntent是一种特殊的Intent。主要的区别在于Intent的执行立刻的，而pendingIntent的执行不是立刻的。
  - getActivity(Context context, int requestCode, Intent intent, int flags)方法从系统取得一个用于启动一个Activity的PendingIntent对象。
  - getService(Context context, int requestCode, Intent intent, int flags)方法从系统取得一个用于启动一个Service的PendingIntent对象。
  - getBroadcast(Context context, int requestCode, Intent intent, int flags)方法从系统取得一个用于向BroadcastReceiver的Intent广播的PendingIntent对象

# 新短信提示– PendingIntent(2)

- flags的取值有四个：

1. FLAG_ONE_SHOT：获取的PendingIntent只能使用一次。
2. FLAG_NO_CREATE：利用FLAG_NO_CREAT获取的PendingIntent，若描述的Intent不存在则返回NULL值。
3. FLAG_CANCEL_CURRENT：如果描述的PendingIntent已经存在，则在产生新的Intent之前会先取消掉当前的。
4. FLAG_UPDATE_CURRENT：能够新new一个 Intent。

# 新短信提示--MainActivity.java

- 在这个activity中，我们用两个TextView，一个用于显示发信人，一个用于显示短信内容。

- 在onCreate中，通过代码获取我们从Notification传过来的数据，进行解析，然后显示。

```
Intent from=this.getIntent();
Bundle bundle=from.getExtras();
```

# 新短信提示----AndroidManifest.xml

- 最后，由于程序使用到系统的收短信功能，所以我们需要在配置文件中，添加用户使用权限。

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

同样是API版本问题，出于安全角度考虑，在API 23之后的用户权限使用需要进行动态权限申请。

# 调试程序

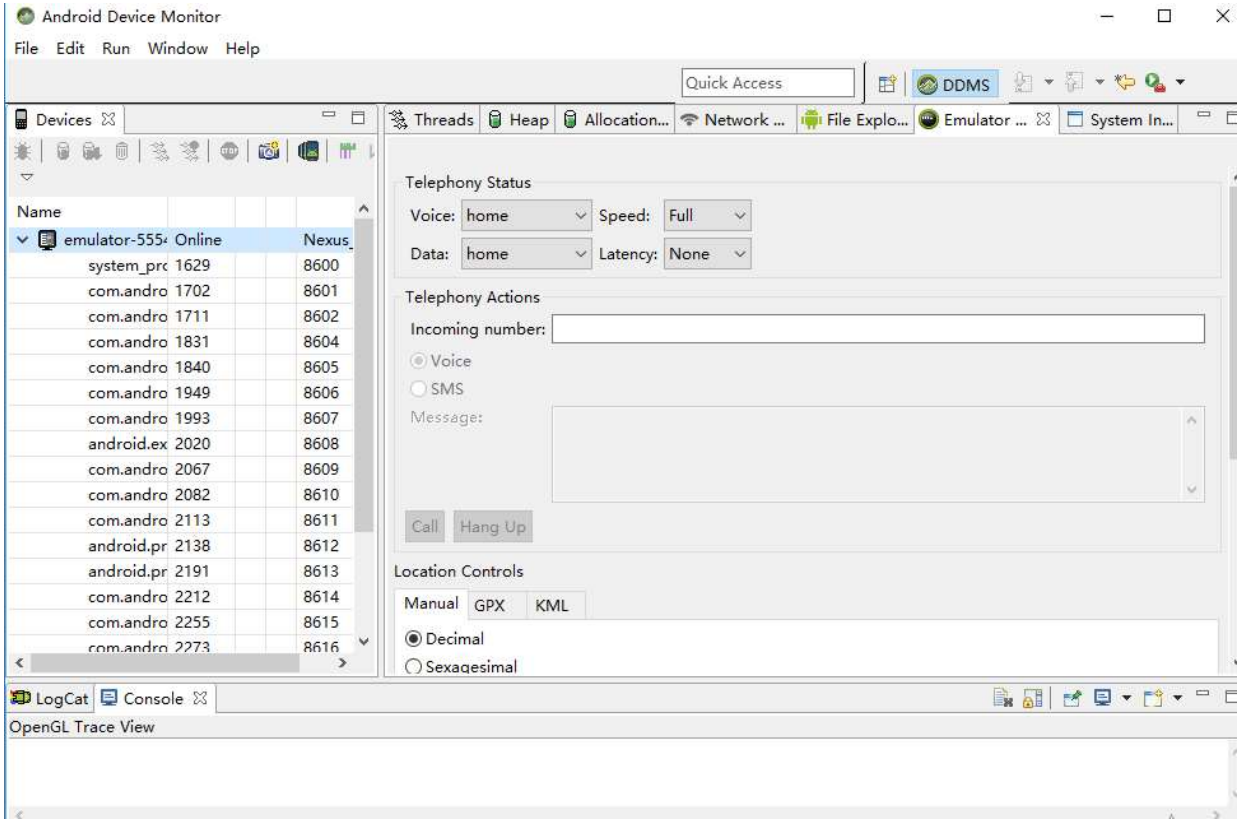
- 程序写完之后，你就可以将apk装到手机上，测试收短信的时候程序是否运行正常。

- 但是如果没有android手机怎么办？或者别人没话费发短信给你调试程序怎么办？

- 不怕，有DDMS……

# 通过DDMS往模拟器发送短信（1）

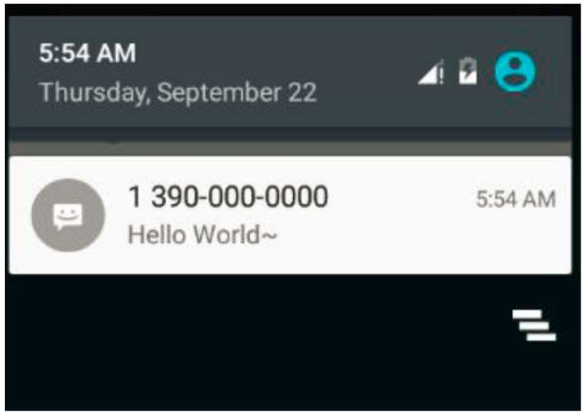
- DDMS 的全称是Dalvik Debug Monitor Service，它为我们提供例如：为测试设备截屏，针对特定的进程查看正在运行的线程以及堆信息、Logcat、广播状态信息、模拟电话呼叫、接收SMS、虚拟地理坐标等等。



# 通过DDMS往模拟器发送短信（2）

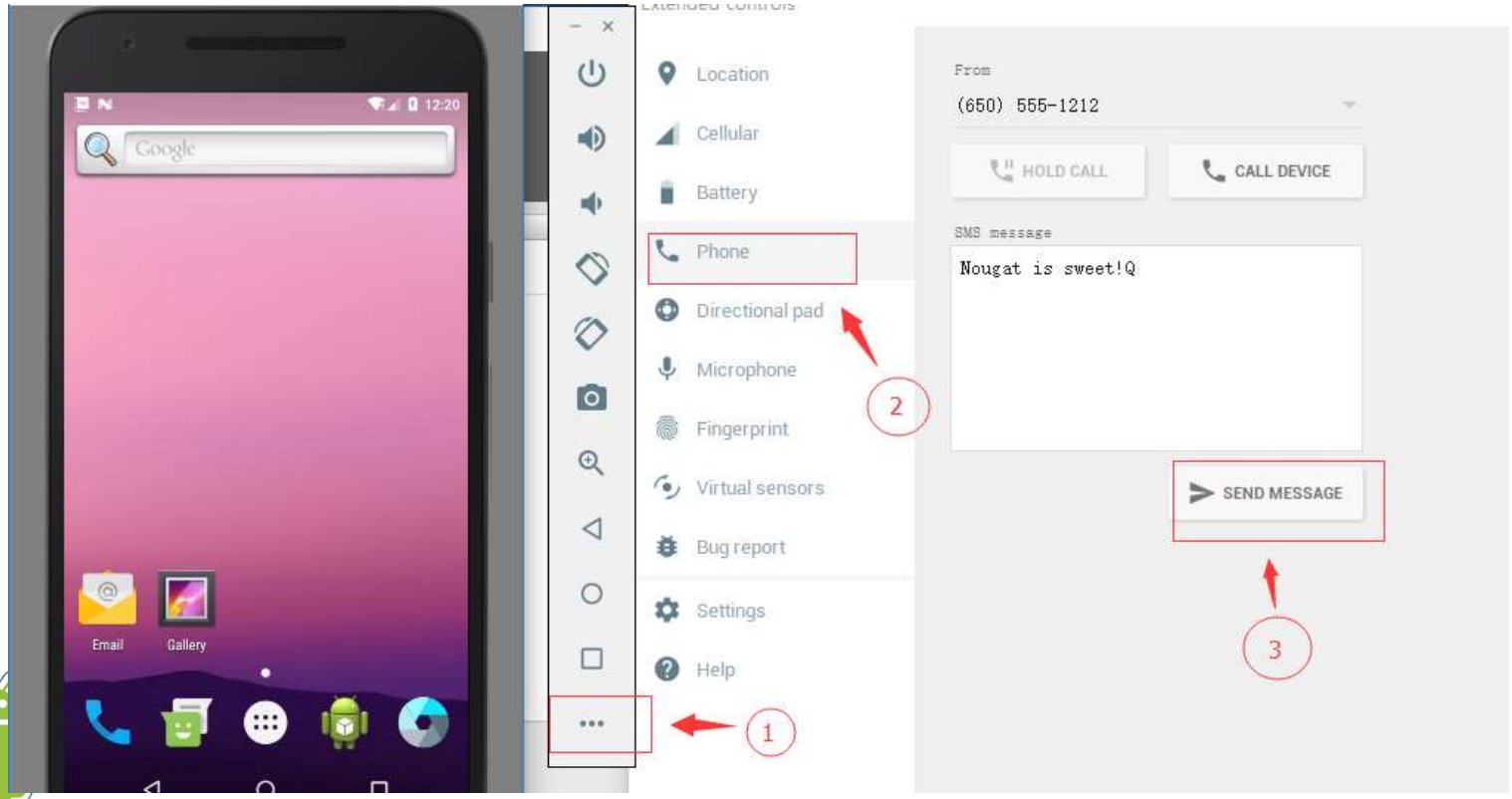
- 我们可以通过DDMS的file explorer查看手机rom里面的数据库文件

- 这次我们需要通过Emulator Control中的Telephony Actions往模拟器发送短信

# 通过DDMS往模拟器发送短信（3）

- 在Android Studio中，也可以直接点击模拟器下的Extended controls，选择phone部分发送短信。



# 如何注销Receiver

- 当手机收到短信的时候Receiver就实例化，执行完onReceive函数之后Receiver对象就被注销。
- 那我们如何注销这个Receiver呢？
  - 很遗憾，这个Receiver无法注销(Why?)
  - 如何注销Receiver 唯一地关闭方法，就是进入手机设置，把我们安装的这个apk给删除掉

# 如何注销Receiver

- 在AndroidManifest.xml注册Receiver，是一种静态注册，所以无法注销

- 动态注册：用户进入程序，按下注册按钮，注册一个Receiver，短信一来，接收并处理广播。如果用户想注销，则按下注销按钮，就可以把Receiver注销，手机收到短信再也不会自动接收广播，除非用户重新注册。

# 动态注册/注销Receiver

- 动态地在代码中先定义并设置好一个IntentFilter对象，然后在需要注册的地方调Context.registerReceiver()方法，如果取消时就调用Context.unregisterReceiver()方法。

- 如果用动态方式注册BroadcastReceiver的Context对象被销毁时，BroadcastReceiver也就自动取消注册了。

# 动态注册/注销Receiver实例

```
mBtnRegister.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        BroadcastReceiver br=new SmsReceiver();
        IntentFilter filter =new IntentFilter();
        filter.addAction(SmsReceiver.mAction);
        registerReceiver(br,filter);
    }
});
mBtnUnRegister.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        BroadcastReceiver br=new SmsReceiver();
        unregisterReceiver(br);
    }
});
```

# 更多的手机自动服务（1）

- android.intent.action.BATTERY_CHANGED
  - 当电量改变时会触发的广播

- android.intent.action.TIME_TIC
  - 每分钟时间跳一次会触发的广播（注意，该广播的Receiver无法在配置文件中注册，只能在代码中动态注册）

# 更多的手机自动服务（2）

- android.intent.action.CAMERA_BUTTON
  - 按下相机键会触发的广播（注意，该广播只在homescreen下才会触发，且必须没有相机程序运行）

- 还有例如蓝牙设备的状态改变、低电量、运营商改变等等广播，这些广播的使用主要靠经验的积累。

# Questions?