# Android四大组件：

**Activity、广播接收者、内容提供者、服务**

# 广播的概念：

* 现实：电台通过发送广播发布消息，买个收音机，就能收听，得知这条消息
* Android：系统运行时，会产生很多事件，比如：电量改变、收发短信、拨打电话、屏幕解锁、开机。
  + 当其中某个事件产生时，系统就会发送广播，应用程序只要使用广播接收者，接收这个广播，

就知道系统产生了什么事件，从而执行相应的代码。

* 发广播也是用意图来发的

# 定义广播接收者：

## 定义一个类，继承：android.content.BroadcastReceiver

* BroadcastReceiver# onReceive(Context context, Intent intent)：接收到广播时调用

## 在清单文件中配置广播接收者，并且指定要接收的广播类型，一个广播接收者可以接收多个类型的广播

广播接收者的<intent-filter>也需要跟广播的<intent-filter>匹配，例如：其中一个类型的广播的<intent-filter>是有一个data子节点，那么广播接收者的<intent-filter>也需要跟广播的data子节点匹配

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"utf-8"*?>  <manifest xmlns:android=*"http://schemas.android.com/apk/res/android"*  package=*"com.example.test9"*  android:versionCode=*"1"*  android:versionName=*"1.0"* >  <uses-sdk  android:minSdkVersion=*"10"*  android:targetSdkVersion=*"17"* />  <application  android:allowBackup=*"true"*  android:icon=*"@drawable/ic\_launcher"*  android:label=*"@string/app\_name"*  android:theme=*"@style/AppTheme"* >  <activity android:name=*"com.example.test9.ContentActivity"*></activity>    <receiver android:name=*"com.example.test9.MyBroadcastReceiver"*>  <intent-filter android:priority=*"1000"*>  <action android:name=*"android.intent.action.NEW\_OUTGOING\_CALL"*/>  <action android:name=*"* *android.provider.Telephony.SMS\_RECEIVED "* />  </intent-filter>  </receiver>  </application>  </manifest> |

**注意：收听打电话广播需要权限：android.permission.PROCESS\_OUTGOING\_CALLS**

# 系统中各种类型广播的接收者

## 广播是怎么把数据存进去的，我们就怎么把数据从广播中取出来

## 打电话广播的action（有序广播）：*android.intent.action.N EW\_OUTGOING\_CALL*

* + 打电话广播有个特性：会携带拨打出去的那个电话号码
  + 通过BroadcastReceiver# getResultData()得到广播中携带的数据，也就是拨打出去的那个电话号码
  + 接收打电话广播需要权限：**android.permission.PROCESS\_OUTGOING\_CALLS**

## 收短信广播的action（有序广播）：*android.provider.Telephony.SMS\_RECEIVED*

* + 被谷歌官方屏蔽掉了，不建议人使用，因为会被窃取信息。所以eclipse提示不出来，需要死记
  + 收短信广播需要权限：**android.permission.RECEIVE\_SMS**
  + 收短信广播中也携带有信息的具体内容，不过信息的内容可能会比较多，所以是用Intent来携带的。
    - 这样我们要获得信息的内容，就需要通过意图了

|  |
| --- |
| **public** **class** SmsBroadcastReceiver **extends** BroadcastReceiver {  @Override  **public** **void** onReceive(Context context, Intent intent) {  //拿到短信内容，短信内容封装在Intent中  Bundle bundle = intent.getExtras();  //取出多条短信，广播中有多少条，接收多少条  //以pdus为键，取出一个object数组，数组中每一个元素，都是一条短信  //pdu是数据单元，在android中表示短信的意思  Object[] objects = (Object[]) bundle.get("pdus");    //把所有短信取出来  **for** (Object object : objects) {  //Android中用android.telephony.SmsMessage来封装短信  //通过pdu来构造短信，这里pdu就是object，需要把object强转为字节数组  SmsMessage sms = SmsMessage.*createFromPdu*((**byte**[]) object);  //拿到发送者的地址，就是拿到哪个号码发过来的  System.*out*.println(sms.getOriginatingAddress());  //拿到短信的内容  System.*out*.println(sms.getMessageBody());  }  }  } |

* 收短信类型的广播接收者，在4.0之后有个改动，如果该收短信广播接收者所在的进程没有被启动过，就不会生效。所以想要生效，那么收短信广播接收者就需要启动一次
* 如果短息过多，就会被分为几条短息发送过来，那么广播中就会有多条短信。可以在收短信广播接收者中对广播中的多条信息进行拼接，形成一条短信

## 多媒体(sd卡)就绪广播的action（未知）：*android.intent.action.MEDIA\_MOUNTED*

* 多媒体就绪广播的<intent-filter>有data子节点，所以该广播的接收者配置时也需要一模一样的data子节点，有data字节点就说明数据使用Intent的data携带的
* <data android:scheme=*"file"*/>

## 多媒体(sd卡)卸载广播的action（未知）：*android.intent.action.MEDIA\_REMOVED*

* 多媒体卸载广播的<intent-filter>有data子节点，所以该广播的接收者配置时也需要一模一样的data子节点，有data字节点就说明数据使用Intent的data携带的
* <data android:scheme=*"file"*/>

## 多媒体（sd卡）未挂载广播的action（未知）：*android.intent.action.MEDIA\_UNMOUNTED*

* 多媒体未挂载广播的<intent-filter>有data子节点，所以该广播的接收者配置时也需要一模一样的data子节点，有data字节点就说明数据使用Intent的data携带的
* <data android:scheme=*"file"*/>

## 开机广播的action（未知）：*android.intent.action.BOOT\_COMPLETED*

* 接收开机广播需要权限：**android.permission.RECEIVE\_BOOT\_COMPLETED**

## 应用安装广播的action（未知）：*android.intent.action.PACKAGE\_ADDED*

* 应用安装广播的<intent-filter>有data子节点，所以该广播的接收者配置时也需要一模一样的data子节点，有data字节点就说明数据使用Intent的data携带的
* <data android:scheme=*"package"*/>
* 广播中携带有被安装的应用的包名，通过Intent#getData()得到

## 应用更新广播的action（未知）：*android.intent.action.PACKAGE\_REPLACED*

* 应用更新广播的<intent-filter>有data子节点，所以该广播的接收者配置时也需要一模一样的data子节点，有data字节点就说明数据使用Intent的data携带的
* <data android:scheme=*"package"*/>
* 广播中携带有被更新的应用的包名，通过Intent#getData()得到

## 应用卸载广播的action（未知）：*android.intent.action.PACKAGE\_REMOVED*

* 应用卸载广播的<intent-filter>有data子节点，所以该广播的接收者配置时也需要一模一样的data子节点，有data字节点就说明数据使用Intent的data携带的
* <data android:scheme=*"package"*/>
* 广播中携带有被卸载的应用的包名，通过Intent#getData()得到

## 电量过低广播的action（未知）：*ACTION\_BATTERY\_LOW*

## 电量从低恢复到正常的aciton（未知）：*ACTION\_BATTERY\_OKAY*

# Ip拨号器

## 原理：当拨号盘应用拨打电话时，会发出打电话广播，里面携带着要拨打的号码。

这个广播会传递给打电话应用，打电话应用会从广播中获取到要拨打的号码，并拨打出去。

而我们做的就是，使用广播接收者，在拨号盘应用把广播传到打电话应用之前，中途截取广播，并拿到广播中携带的数据进行修改，

然后在把数据放回广播中。最后，打电话应用接收到的广播中的数据，已经是我们修改

## 实现过程：

* 1. 设置布局：

|  |
| --- |
| <LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*  xmlns:tools=*"http://schemas.android.com/tools"*  android:layout\_width=*"match\_parent"*  android:layout\_height=*"match\_parent"*  tools:context=*".MainActivity"*  android:orientation=*"vertical"* >  <EditText  android:id=*"@+id/et\_ip"*  android:layout\_width=*"match\_parent"*  android:layout\_height=*"wrap\_content"*  android:hint=*"输入ip"*/>  <Button  android:layout\_width=*"wrap\_content"*  android:layout\_height=*"wrap\_content"*  android:text=*"保存"*  android:onClick=*"click"*/>  </LinearLayout> |

* 1. 在MainActivity中，把用户设置的ip短号保存起来
  2. 定义广播接收者，配置广播接收者，设置接收的广播类型为打电话广播：*android.intent.action.NEW\_OUTGOING\_CALL。*

因为我们定义的广播接收者的级别，比打电话应用的广播接收者的级别高，所以会比打电话应用先收到打电话广播

在广播接收者中，把广播中携带的号码数据修改，再重新设置回广播中。最后广播被打电话应用接收

|  |
| --- |
| **package** com.example.ipcall;  **import** android.content.BroadcastReceiver;  **import** android.content.Context;  **import** android.content.Intent;  **import** android.content.SharedPreferences;  **public** **class** CallBroadcastReceiver **extends** BroadcastReceiver {  //接收到广播时调用  @Override  **public** **void** onReceive(Context context, Intent intent) {  System.*out*.println("接收");    //打电话广播会携带拨打出去的号码，获取到携带的号码  String number = getResultData();    //在从SharedPreference中取出保存的ip段号  SharedPreferences sp = context.getSharedPreferences("ip", Context.*MODE\_PRIVATE*);  String ip = sp.getString("ip", "");  number = ip + number;    //在重新设置回广播中  setResultData(number);  }  } |

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"utf-8"*?>  <manifest xmlns:android=*"http://schemas.android.com/apk/res/android"*  package=*"com.example.ipcall"*  android:versionCode=*"1"*  android:versionName=*"1.0"* >  <uses-sdk  android:minSdkVersion=*"10"*  android:targetSdkVersion=*"17"* />  <uses-permission android:name=*"android.permission.PROCESS\_OUTGOING\_CALLS"*/>  <application  android:allowBackup=*"true"*  android:icon=*"@drawable/ic\_launcher"*  android:label=*"@string/app\_name"*  android:theme=*"@style/AppTheme"* >  <activity  android:name=*"com.example.ipcall.MainActivity"*  android:label=*"@string/app\_name"* >  <intent-filter>  <action android:name=*"android.intent.action.MAIN"* />  <category android:name=*"android.intent.category.LAUNCHER"* />  </intent-filter>  </activity>    <receiver android:name=*"com.example.ipcall.CallBroadcastReceiver"*>  <intent-filter>  <action android:name=*"android.intent.action.NEW\_OUTGOING\_CALL"*/>  </intent-filter>  </receiver>  </application>  </manifest> |

* 1. 接收打电话广播需要权限：**android.permission.PROCESS\_OUTGOING\_CALLS**

# 短信防火墙

## 原理：

在系统的短信应用收到收短信广播之前，我们自定义的收短信广播接收者就优先收到了短信，并且获取短信中的发送者号码，如果不符合条件，就屏蔽掉这条收短信广播，不被其他收短信广播接收者收到这条广播

## 实现过程：

1. 定义广播接收者，在清单文件中配置其接收的广播类型为收短信广播，并且设置优先级（比系统的收短信广播接收者高即可）
2. 收短信广播携带的短信量可能会比较大，所以是通过Intent来携带短信内容的
3. 在BroadcastReceiver# onReceive(Context context, Intent intent)中，通过Intent来获取广播接收者中的所有短信
4. 判断短信的发送者是否不是黑名单中的号码，如果是黑名单中的号码，就屏蔽掉这条收短信广播，不让其他收短信广播接收者接收到这条广播
5. 通过BroadcastReceiver# abortBroadcast()来屏蔽广播，阻止其他广播接收者接收这条广播

# SD卡状态监控

定义一个广播接收者，配置其接收的类型为多媒体就绪、卸载、未挂载这三个广播

在BroadcastReceiver# onReceive(Context context, Intent intent)中，通过Intent#getAction()来获取广播接收者中所接收到的广播类型

根据接收到的不同广播类型，做相应操作

# 手机流氓软件

## 原理：

* activity不会被关掉。屏蔽返回键，返回键是调用生命周期的onBackPressed()方法，屏蔽掉其super.onBackPressed()即可，

使home键失效，在安全卫士中会讲解。开机自启动

## 实现过程：

1. 创建广播接收者，配置其接收的广播类型为开机广播
2. 接收到广播时，在BroadcastReceiver#onReceive(Context context, Intent intent)中，启动流氓软件的activity
3. 因为广播接收者运行是不需要依赖activity的，所以没有任务栈。而启动的activity需要任务栈存放，所以要为Intent设置一个flag，告诉它创建一个新的任务栈存放流氓软件的activity

# 应用状态监控

## 实现过程：

* 广播接收者接收的类型配置为，接收应用安装、卸载、更新广播
* 通过Intent#getAction()得到接收到的广播的类型，做相应操作
* 通过Intent#getData()得到广播中携带的应用的包名

# 自定义广播

* 广播也是用Intent发的，所以需要定义一个Intent，设置Action，然后使用Activity#SendBroadcast(Intent)发送广播
* 自定义广播时不需要在清单文件中配置，定义自定义广播的接收者才需要

## 发送自定义广播

## 接收自定义广播

# 广播的两种类型

## 无序广播

* 所有跟广播的intent匹配的广播接收者都可以收到该广播，并且是没有先后顺序（同时收到）

## 有序广播

* 所有跟广播的intent匹配的广播接收者都可以收到该广播，但是会按照广播接收者的优先级来决定接收的先后顺序
* 在清单文件中定义优先级，取值为-1000~1000
* 最终接收者(也是一个广播接收者)：所有广播接收者都接收到广播之后，它才接收，并且一定会接收，即使被BroadCastReceiver#abortBroadCast()拦截了，也能收到。它不需要在清单文件中配置
* BroadCastReceiver#abortBroadCast()：阻止其他接收者接收这条广播，类似拦截，只有有序广播可以被拦截
* BroadCastReceiver#setResultData(String):修改广播中的内容，只有有序广播才可以修改

# 定义有序广播并发送、接收

## 实现过程

* + 定义Intent，设置action，使用ContextWrapper#sendOrderedBroadcast(intent, receiverPermission, resultReceiver, scheduler, initialCode, initialData, initialExtras)发送有序广播
    1. Intent：意图对象
    2. receiverPermission：接收此广播所需要的权限
    3. resultReceiver：最终接收者对象
    4. scheduler：
    5. initialCode：和请求码类似，做标记用
    6. initialData：携带数据用，携带一些比较简单的数据。如果数据比较复杂，可使用Intent携带
    7. initialExtras：这是一个Bundle，也是用来携带数据的，携带比较复杂的数据
  + 定义多个广播接收者，配置其接收的广播类型为我们自定义的广播类型。设置他们的优先级

## 发送

## 接收

# 服务的概念：

* 就是默默运行在后台的组件，可以理解为是没有前台的activity，适合用来运行不需要前台界面的代码，例如：下载，音乐播放
* 服务可以被手动关闭，不会重启，但是如果被自动关闭，内存充足就会重启
* 服务的开启方式有两种：
  + ContextWrapper# startService(Intent)：启动和关闭，这种方式是得不到中间对象的
    - 该方法启动的服务所在的进程属于服务进程
    - Activity一旦启动服务，服务就跟Activity一毛钱关系也没有了
  + ContextWrapper# bindService([Intent](eclipse-javadoc:%E2%98%82=BindService/D:%5C/Program%20Files%20%5C(x86)%5C/adt-bundle-windows-x86_64_20140101%5C/sdk%5C/platforms%5C/android-18%5C/android.jar%3Candroid.content(ContextWrapper.class%E2%98%83ContextWrapper~bindService~Landroid.content.Intent;~Landroid.content.ServiceConnection;~I%E2%98%82Intent) service, [ServiceConnection](eclipse-javadoc:%E2%98%82=BindService/D:%5C/Program%20Files%20%5C(x86)%5C/adt-bundle-windows-x86_64_20140101%5C/sdk%5C/platforms%5C/android-18%5C/android.jar%3Candroid.content(ContextWrapper.class%E2%98%83ContextWrapper~bindService~Landroid.content.Intent;~Landroid.content.ServiceConnection;~I%E2%98%82ServiceConnection) conn, int flags)：绑定和解绑，这个方式可以得到中间对象
    - 该方法启动的服务所在进程不属于服务进程
    - Activity与服务建立连接，Activity一旦死亡，服务也会死亡
* 服务的生命周期方法：
  + onBind(Intent intent):服务绑定时此方法调用，如果服务没有创建，会先调用onCreate()创建服务
  + onUnbind(Intent intent): 服务解绑时此方法调用，解绑之后会调用onDestroy()销毁服务
  + onCreate():服务创建时此方法调用
  + onStart(Intent intent, int startId): 服务启动时此方法调用
  + onDestroy():服务销毁时此方法调用

# 进程优先级：

1. 前台进程：拥有一个正在与用户交互的前台activity（Activity#onResume方法被调用）
2. 可见进程：拥有一个可见但是没有焦点的activity（Activity#onPause方法被调用）
3. 服务进程：拥有一个通过startService方法启动的服务，不到万不得已不会被回收，而且即便被回收（仅限自动关闭，而不是手动关闭），内存充足时也会被重启
4. 后台进程：拥有一个不可见的activity（Activity#onStop方法被调用了），很容易被回收
5. 空进程：没有运行任何activity或其他四大组件，很容易被回收

# 开启和关闭服务（第一种方式：ContextWrapper# startService(Intent)）

## 启动服务（显式启动）

* 创建一个类，继承android.app.Service
* 在清单文件中配置服务，跟activity的配置几乎一样。

|  |
| --- |
| <application  android:allowBackup=*"true"*  android:icon=*"@drawable/ic\_launcher"*  android:label=*"@string/app\_name"*  android:theme=*"@style/AppTheme"* >    <service android:name=*"com.example.test10.aService"*></service>  </application> |

* 服务的开启也有显式和隐式之分，如果是显式启动，就不需要配置<intent-filter>，如果是隐式启动（用来开启远程服务，在32天里面讲解），就要配置<intent-filter>
* 在布局文件中设置一个按钮，点击按钮显式启动服务，跟显式启动activity几乎一样
* 使用ContextWrapper# startService(Intent)，启动服务

## 关闭显式开启的服务

* 使用ContextWrapper# stopService(Intent)，关闭服务
* 使用代码关闭也属于用户手动关闭，所以服务不会被自动重启

## 该启动方式会调用的服务的生命周期方法

* onCreate：服务被创建时调用
* onStartCommand：服务被开启时调用，里面调用的是onstart方法，因为名字不合适，所以改了onStartCommand。实际上还是调用onstart方法
* onDestroy：服务被关闭时调用
* 重复的调用startService会导致onStartCommand被重复调用，但是onCreate只调用了一次，所以不会再创建一个服务

# 电话录音机

不是重点，知道就行。就算要用到录音的api，也是用第三方的，Android官方的不太好

* 电话状态：空闲、响铃、接听
* 录音机写在服务里，不会被系统杀死。
  + 创建一个服务，按下按钮开启服务
* 在服务中获取电话管理器TelephonyManager，设置监听电话状态，当响铃时就初始化录音机MediaRecorder
* 摘机时开始录音，空闲时停止录音
* 需要的权限：电话状态监听权限，写sd卡权限，录音权限

<uses-permission android:name=*"android.permission.READ\_PHONE\_STATE"*/>

<uses-permission android:name=*"android.permission.WRITE\_EXTERNAL\_STORAGE"*/>

<uses-permission android:name=*"android.permission.RECORD\_AUDIO"* />