Andoid中系统在运行的过程中, 会产生很多事件, 那么某些事件产生时, 例如：电量改变、收发短信、拨打电话、屏幕解锁、开机、系统都会发送广播, 只要应用程序收到这条广播. 就知道系统发生了相应的事件, 从而执行相应的代码。 使用广播接受者(广播接受者, 是Android四大组件之一), 就可以收听广播.

Android自动启动Service

背景知识：当Android启动时，会发出一个系统广播，内容为ACTION_BOOT_COMPLETED，它的字符串常量表示为android.intent.action.BOOT_COMPLETED。只要在程序中“捕捉”到这个消息，再启动之即可。记住，Android框架说：Don't call me, I'll call you back。我们要做的是做好接收这个消息的准备，而实现的手段就是实现一个BroadcastReceiver。

```
/*
 * BroadcastReceiver 创建一个广播接受者. 注意在应用程序中注册相应的权限
 * <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
 * 否则当前应用程序不会被触发
 * */
public class BootBroadcastReceiver extends BroadcastReceiver {

 @Override
 public void onReceive(Context context, Intent intent) {
  Toast.makeText(context, "在这里运行一些代码", Toast.LENGTH_LONG).show();
 }
}
```

在 AndroidManifest.xml 的配置中配置权限：

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

最后在AndroidManifest.xml中配置广播接收器：

```
<receiver android:name="com.example.demo01.BootBroadcastReceiver" >
      <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
      </intent-filter>
</receiver>
```

注意事项：

- 测试的时候最好拿真实机测试(在安装的时候会显示自动运行权限提示框), 正常如果安装成功, 会在系统的自动启动软件中查看到当前消息

小案例 IP拨号器

定义一个广播接受者必须继承BroadcastReceiver, 用来接收广播信息

```
public class CallReceiver extends BroadcastReceiver {
 @Override // 接受到广播时就会被调用
 public void onReceive(Context context, Intent intent) {
  Log.i("jxy","接收到了");
 }
```

```
6.    }
```

配置一个广播接受者,广播接受者是没有界面的,因此也不需要布局文件

```
1.    <receiver android:name="com.example.service.CallReceiver">
2.          <intent-filter>
3.          <!-- 3: 指定广播接受者接收打广播类型(打电话广播),记得添加权限 -->
4.          <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
5.          </intent-filter>
6.    </receiver>
```

拨打电话涉及到隐私, 因此需要开启相应的用户权限

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

在原来用户的号码上面添加17951(即使广播接受者进程被关闭. 当系统发出的广播中action跟改广播的接受者匹配, 系统会自动重启广播接受者进程. 主要是为了节省内存, 因为用户并不会24小时打电话)

```
1.    public class CallReceiver extends BroadcastReceiver {
2.     @Override // 接受到广播时就会被调用
3.     public void onReceive(Context context, Intent intent) {
4.      Log.i("jxy", "接收到了");
5.      // 拨号器到call页面不是正常的activity跳转,而是通过广播的形式发送,发送的过程中被我们
      截取而已
6.      String number = getResultData();
7.      // 吧新的号码重新翻入广播中
8.      setResultData("17951" + number);
9.     }
10.   }
11.
```

发自定义无序广播(用的非常少, 只需要了解即可)

```
1.    // intent 发送自定义广播,注意自己定义Action字符串
2.    Intent intent=new Intent();
3.    intent.setAction("cn.jxy.action");
4.    sendBroadcast(intent);
```

开启方式：
startService：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一毛钱关系都没有了
bingService：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. 服务也会死亡

开启和关闭服务：在Android四大组件中：Activity、BroadCase、Service三大组件启动的时候都需要使用Intent

```
1.    public void click(View v) {
2.     // intent 发送自定义广播,注意自己定义Action字符串
3.     Intent intent = new Intent(this, MyService.class);
4.     startService(intent);
5.    }
```

```
6.
7.  public void click2(View v) {
8.   // intent 发送自定义广播,注意自己定义Action字符串
9.   Intent intent = new Intent(this, MyService.class);
10.  stopService(intent);
11.  }
```

自己定义一个服务, 注意必须要继承Service类

```
1.  public class MyService extends Service {
2.
3.   @Override
4.   public IBinder onBind(Intent intent) {
5.    return null;
6.   }
7.
8.  }
```

讲解Service的生命周期：

```
1.  @Override
2.  public void onCreate() {
3.   super.onCreate();
4.   Log.i("jxy","第一次创建服务的时候执行");
5.  }
6.
7.  @Override
8.  public void onDestroy() {
9.   // TODO Auto-generated method stub
10.  super.onDestroy();
11.  Log.i("jxy","销毁服务的时候执行");
12.  }
13.
14.  @Override
15.  public int onStartCommand(Intent intent, int flags, int startId) {
16.   // TODO Auto-generated method stub
17.   Log.i("jxy","按多次仅仅启动一次服务");
18.   return super.onStartCommand(intent, flags, startId);
19.  }
```

采用bind与解绑的方式创建Service与销毁服

```
1.
    1.  public void click3(View v) {
    2.   bindService(it, conn, BIND_AUTO_CREATE);
    3.  }
    4.
    5.  public void click4(View v) {
    6.   unbindService(conn);
    7.  }
    8.
    9.  class MyServiceConnection implements ServiceConnection {
    10.  @Override
    11.  public void onServiceConnected(ComponentName name, IBinder service) {
    12.   Log.i("jxy","onServiceConnected");
```

```
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
     Log.i("jxy","onServiceDisconnected");
    }
   }
```

触发的生命周期为：注意,此时按返回键服务也会被销毁掉(生命周期与Activity相同)

```
@Override
public void onCreate() {
 super.onCreate();
 Log.i("jxy","onCreate");
}

@Override
public IBinder onBind(Intent intent) {
 Log.i("jxy","onBind");
 return null;
}

@Override
public void onDestroy() {
 Log.i("jxy","onDestroy");
 super.onDestroy();
}

@Override
public boolean onUnbind(Intent intent) {
 Log.i("jxy","onUnbind");
 return super.onUnbind(intent);
}
```

广播+服务完成录音功能

预备技能点：

1. getSystemService使用
2. 广播+服务的机制
3. 同一个格式有不同的编码

getSystemService是Android很重要的一个API，它是Activity的一个方法，根据传入的NAME来取得对应的Object，然后转换成相应的服务对象。以下介绍系统相应的服务

传入的Name	返回的对象	说明
WINDOW_SERVICE	WindowManager	管理打开的窗口程序
LAYOUT_INFLATER_SERVICE	LayoutInflater	取得xml里定义的view
ACTIVITY_SERVICE	ActivityManager	管理应用程序的系统状态
POWER_SERVICE	PowerManger	电源的服务
ALARM_SERVICE	AlarmManager	闹钟的服务
NOTIFICATION_SERVICE	NotificationManager	状态栏的服务
KEYGUARD_SERVICE	KeyguardManager	键盘锁的服务
LOCATION_SERVICE	LocationManager	位置的服务，如GPS
SEARCH_SERVICE	SearchManager	搜索的服务
VEBRATOR_SERVICE	Vebrator	手机震动的服务
CONNECTIVITY_SERVICE	Connectivity	网络连接的服务
WIFI_SERVICE	WifiManager	Wi-Fi服务
TELEPHONY_SERVICE	TeleponyManager	电话服务

第一步：创建一个服务并且部署到xml中：

```
/*
 * 电话的状态：
 *   1：空闲状态
 *   2：响铃状态
 *   3：摘机状态，如果挂断则从新回到空闲状态
 * */
public class RecordService extends Service {

 private MediaRecorder recorder;

 @Override
 public IBinder onBind(Intent intent) {
  return null;
 }

 @Override
 public void onCreate() {
  super.onCreate();
  Log.i("jxy", "第一次创建服务的时候执行");
 // 系统提供了很多服务用来完成一些基本的功能,电源、布局.....
 TelephonyManager tm = (TelephonyManager) getSystemService(TELEPHONY_SERVICE);
 // 监听电话状态：events:决定PhoneStateListener用来决定监听哪些事件的.
 tm.listen(new CallListener(), PhoneStateListener.LISTEN_CALL_STATE);
}
```

第二步：创建一个电话监听类：PhoneStateListener此功能必须要用户允许

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

第三步：根据电话的状态,来实现相应的操作：

```
class CallListener extends PhoneStateListener {

 @Override
 public void onCallStateChanged(int state, String incomingNumber) {
  // TODO Auto-generated method stub
  super.onCallStateChanged(state, incomingNumber);
  switch (state) {
  case TelephonyManager.CALL_STATE_IDLE:
   Log.i("jxy", "空闲状态");
   if (recorder != null) {
    recorder.stop();
    recorder.release();
    recorder = null;
   }
   break;
  case TelephonyManager.CALL_STATE_RINGING:
   Log.i("jxy", "响铃状态");
   if (recorder == null) {
    recorder = new MediaRecorder();
    // 设置音频的来源，由于法律的问题：只能录制自己的声音,不能录制别人的声音的 (VOICE_CALL),
    // 通过百度有解决方案
    recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
    // 设置音频的格式，注意一个格式可能会有多个编码，因此相同的格式有些播放器能播放有些不能,因为编码问题
    recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    // 此处可以修改成按日期存储
    recorder.setOutputFile(RecordService.this.getFilesDir()
      + "/luyin.3gp");
    // 设置编码，一个格式可以设置多个编码
    recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
    try {
     recorder.prepare();
    } catch (Exception e) {
     e.printStackTrace();
    }
   }
   break;
  case TelephonyManager.CALL_STATE_OFFHOOK:
   Log.i("jxy", "摘机状态");
   // 开始录音
   if (recorder != null) {
    recorder.start();
   }
   break;
  }
 }
```