我们在使用Android手机打电话时，有时可能会需要对来去电通话自动录音，本文就详细讲解实现Android来去电通话自动录音的方法，大家按照文中的方法编写程序就可以完成此功能。

****来去电自动录音的关键在于如何监听手机电话状态的转变：****

       1）来电的状态的转换如下（红色标记是我们要用到的状态）

       空闲（IDEL）——> 响铃（RINGING）——> 接听（ACTIVE）——> 挂断（经历DISCONNECTING——DISCONNECTED）——> 空闲（IDEL）

       或者  空闲（IDEL）——> 响铃（RINGING）——> 拒接 ——> 空闲（IDEL）

       2）去电状态的转换如下

       空闲（IDEL）——> 拨号 （DIALING）——> （对方）响铃（ALERTING） ——> 建立连接（ACTIVE）—— 挂断（经历DISCONNECTING——DISCONNECTED）——> 空闲（IDEL）

       或者 空闲（IDEL）——> 拨号 （DIALING）——> （对方）响铃（ALERTING）——> 挂断/对方拒接 ——> 空闲（IDEL）

       下面就分别就来电和去电这两种状态分析并实现。

****1、先进行来电的分析和实现。****

       相对去电来说，来电状态的转换检测要简单些。android api 中的PhoneStateListener 类提供了相应的方法，但我们需要覆盖其中的 onCallStateChanged(int state, String incomingNumber) 方法即可实现来电状态的检测，并在此基础上添加录音功能即可。其中 state 参数就是各种电话状态，到时我们将它跟下面我们要用到的状态进行比较，若是电话处在我们想要的状态上，则进行一系列操作，否则就不管他。想要获取这些状态，还需要另一个电话相关类，那就是 TelephonyManager， 该类 提供了一些电话状态，其中我们要用到的是：TelephonyManager.CALL\_STATE\_IDLE（空闲）、TelephonyManager.CALL\_STATE\_OFFHOOK（摘机）和 TelephonyManager.CALL\_STATE\_RINGING（来电响铃）这三个状态。判别这三种状态，可以继承 android.telephony.PhoneStateListener 类，实现上面提到的 onCallStateChanged(int state, String incomingNumber) 方法，请看如下代码：

**Java代码**

1. **public** **class** TelListener **extends** PhoneStateListener {
3. @Override
4. **public** **void** onCallStateChanged(**int** state, String incomingNumber) {
5. **super**.onCallStateChanged(state, incomingNumber);
7. **switch** (state) {
8. **case** TelephonyManager.CALL\_STATE\_IDLE: // 空闲状态，即无来电也无去电
9. Log.i("TelephoneState", "IDLE");
10. //此处添加一系列功能代码
11. **break**;
12. **case** TelephonyManager.CALL\_STATE\_RINGING: // 来电响铃
13. Log.i("TelephoneState", "RINGING");
14. //此处添加一系列功能代码
15. **break**;
16. **case** TelephonyManager.CALL\_STATE\_OFFHOOK: // 摘机，即接通
17. Log.i("TelephoneState", "OFFHOOK");
18. //此处添加一系列功能代码
19. **break**;
20. }
22. Log.i("TelephoneState", String.valueOf(incomingNumber));
23. }
25. }

       有了以上来电状态监听代码还不足以实现监听功能，还需要在我们的一个[Activity](http://www.jizhuomi.com/android/catalog.asp?tags=Activity" \t "http://www.jizhuomi.com/android/example/_blank)或者[Service](http://www.jizhuomi.com/android/catalog.asp?tags=Service" \t "http://www.jizhuomi.com/android/example/_blank)中实现监听，方法很简单，代码如下：

**Java代码**

1. /\*\*
2. \* 在activity 或者 service中加入如下代码，以实现来电状态监听
3. \*/
4. TelephonyManager telMgr = (TelephonyManager)context.getSystemService(
5. Context.TELEPHONY\_SERVICE);
6. telMgr.listen(**new** TelListener(), PhoneStateListener.LISTEN\_CALL\_STATE);

       这样就实现了来电状态监听功能，但要能够在设备中跑起来，这还不够，它还需要两个获取手机电话状态的权限：

**XML/HTML代码**

1. **<uses-permission** android:name="android.permission.READ\_PHONE\_STATE" **/>**
2. **<uses-permission** android:name="android.permission.PROCESS\_OUTGOING\_CALLS" **/>**

       这样的话就可以跑起来了。

       说到这，我想如果你可以实现录音功能的话，在此基础上实现来电自动录音就应该没什么问题了，不过请容我简单罗嗦几句。既然是来电，那么要想录音的话，那么应该就是在监听到 TelephonyManager.CALL\_STATE\_OFFHOOK 的状态时开启录音机开始录音， 在监听到TelephonyManager.CALL\_STATE\_IDLE 的状态时关闭录音机停止录音。这样，来电录音功能就完成了，不要忘记录音功能同样需要权限：

**XML/HTML代码**

1. **<uses-permission** android:name="android.permission.RECORD\_AUDIO"**/>**
3. <!-- 要存储文件或者创建文件夹的话还需要以下两个权限 -->
4. **<uses-permission** android:name="android.permission.MOUNT\_UNMOUNT\_FILESYSTEMS"**/>**
5. **<uses-permission** android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"**/>**

****2、介绍完了来电自动录音，下面就来介绍去电自动录音的实现方法。****

       上面说过，相比来电状态的监听，去电的要麻烦些，甚至这种方法不是通用的，这个主要是因为android api 中没有提供去电状态监听的相应类和方法（也许我刚接触，没有找到）。刚开始网上搜索了一通也没有找到对应的解决方法，大多是 来电监听的，也就是上面的方法。不过中途发现一篇博文（后来就搜不到了），记得是查询系统日志的方式，从中找到去电过程中的各个状态的关键词。无奈之中，最终妥协了此方法。

       我的（联想A65上的）去电日志内容如下：

       过滤关键词为 mforeground

**Java代码**

1. 01-06 16:29:54.225: D/InCallScreen(251): onPhoneStateChanged: mForegroundCall.getState() : DIALING
2. 01-06 16:29:54.245: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mForegroundCall.getState() : DIALING
3. 01-06 16:29:54.631: D/InCallScreen(251): onPhoneStateChanged: mForegroundCall.getState() : DIALING
4. 01-06 16:29:54.645: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mForegroundCall.getState() : DIALING
5. 01-06 16:29:54.742: D/InCallScreen(251): onPhoneStateChanged: mForegroundCall.getState() : DIALING
6. 01-06 16:29:54.766: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mForegroundCall.getState() : DIALING
7. 01-06 16:29:54.873: D/InCallScreen(251): onPhoneStateChanged: mForegroundCall.getState() : DIALING
8. 01-06 16:29:54.877: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mForegroundCall.getState() : DIALING
9. 01-06 16:29:55.108: D/InCallScreen(251): onPhoneStateChanged: mForegroundCall.getState() : DIALING
10. 01-06 16:29:55.125: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mForegroundCall.getState() : DIALING
11. 01-06 16:29:57.030: D/InCallScreen(251): onPhoneStateChanged: mForegroundCall.getState() : ACTIVE
12. 01-06 16:29:57.155: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mForegroundCall.getState() : ACTIVE
13. 01-06 16:29:57.480: D/InCallScreen(251): onPhoneStateChanged: mForegroundCall.getState() : ACTIVE
14. 01-06 16:29:57.598: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mForegroundCall.getState() : ACTIVE
15. 01-06 16:29:59.319: D/InCallScreen(251): onPhoneStateChanged: mForegroundCall.getState() : DISCONNECTING
16. 01-06 16:29:59.373: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mForegroundCall.getState() : DISCONNECTING
17. 01-06 16:30:00.392: D/InCallScreen(251): - onDisconnect: currentlyIdle:**true** ; mForegroundCall.getState():DISCONNECTED
18. 01-06 16:30:00.399: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): - onDisconnect: currentlyIdle:**true** ; mForegroundCall.getState():DISCONNECTED
19. 01-06 16:30:01.042: D/InCallScreen(251): onPhoneStateChanged: mForegroundCall.getState() : IDLE
20. 01-06 16:30:01.070: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mForegroundCall.getState() : IDLE
21. 01-06 16:30:01.558: D/InCallScreen(251): onPhoneStateChanged: mForegroundCall.getState() : IDLE
22. 01-06 16:30:01.572: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mForegroundCall.getState() : IDLE

       过滤关键词  mbackground

**Java代码**

1. 01-06 16:29:54.226: D/InCallScreen(251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
2. 01-06 16:29:54.256: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
3. 01-06 16:29:54.638: D/InCallScreen(251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
4. 01-06 16:29:54.652: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
5. 01-06 16:29:54.743: D/InCallScreen(251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
6. 01-06 16:29:54.770: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
7. 01-06 16:29:54.875: D/InCallScreen(251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
8. 01-06 16:29:54.882: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
9. 01-06 16:29:55.109: D/InCallScreen(251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
10. 01-06 16:29:55.142: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
11. 01-06 16:29:57.031: D/InCallScreen(251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
12. 01-06 16:29:57.160: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
13. 01-06 16:29:57.481: D/InCallScreen(251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
14. 01-06 16:29:57.622: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
15. 01-06 16:29:59.319: D/InCallScreen(251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
16. 01-06 16:29:59.373: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
17. 01-06 16:30:01.042: D/InCallScreen(251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
18. 01-06 16:30:01.070: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
19. 01-06 16:30:01.559: D/InCallScreen(251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE
20. 01-06 16:30:01.573: V/LogInfo OutGoing Call(2492): D/InCallScreen(  251): onPhoneStateChanged: mBackgroundCall.getState() : IDLE

       从上面的日志可以看到，每一行的末尾的大写英文词就是去电的状态，状态说明如下：

       DIALING 拨号，对方还未响铃  
       ACTIVE   对方接通，通话建立  
       DISCONNECTING 通话断开时  
       DISCONNECTED  通话已断开，可以认为是挂机了

       由于我拨打的是10010，没有响铃过程（电脑自动接通的够快），还少了一个状态，状态是ALERTING ，这个就是对方正在响铃的状态。

       有了这几个去电状态就好办了，现在我们要做的就是读取系统日志，然后找到这些状态，提取的关键词就是上面提到的 mforeground（前台通话状态） 和 mbackground （后台通话状态）（可能不一样的设备生成的不一样，根据自己具体设备设置，这里只提取前台的），如果读取的这一行日志中 包含 mforground ，再看看是否包含上面的状态的单词。既然说的如此，那么看看读取系统日志的代码吧。

**Java代码**

1. **package** com.sdvdxl.phonerecorder;
3. **import** java.io.BufferedReader;
4. **import** java.io.IOException;
5. **import** java.io.InputStream;
6. **import** java.io.InputStreamReader;
8. **import** com.sdvdxl.outgoingcall.OutgoingCallState;
10. **import** android.content.Context;
11. **import** android.content.Intent;
12. **import** android.util.Log;
14. /\*\*
15. \*
16. \* @author sdvdxl
17. \*  找到 日志中的
18. \*  onPhoneStateChanged: mForegroundCall.getState() 这个是前台呼叫状态
19. \*  mBackgroundCall.getState() 后台电话
20. \*  若 是 DIALING 则是正在拨号，等待建立连接，但对方还没有响铃，
21. \*  ALERTING 呼叫成功，即对方正在响铃，
22. \*  若是 ACTIVE 则已经接通
23. \*  若是 DISCONNECTED 则本号码呼叫已经挂断
24. \*  若是 IDLE 则是处于 空闲状态
25. \*
26. \*/
27. **public** **class** ReadLog **extends** Thread {
28. **private** Context ctx;
29. **private** **int** logCount;
31. **private** **static** **final** String TAG = "LogInfo OutGoing Call";
33. /\*\*
34. \*  前后台电话
35. \* @author sdvdxl
36. \*
37. \*/
38. **private** **static** **class** CallViewState {
39. **public** **static** **final** String FORE\_GROUND\_CALL\_STATE = "mForeground";
40. }
42. /\*\*
43. \* 呼叫状态
44. \* @author sdvdxl
45. \*
46. \*/
47. **private** **static** **class** CallState {
48. **public** **static** **final** String DIALING = "DIALING";
49. **public** **static** **final** String ALERTING = "ALERTING";
50. **public** **static** **final** String ACTIVE = "ACTIVE";
51. **public** **static** **final** String IDLE = "IDLE";
52. **public** **static** **final** String DISCONNECTED = "DISCONNECTED";
53. }
55. **public** ReadLog(Context ctx) {
56. **this**.ctx = ctx;
57. }
59. /\*\*
60. \* 读取Log流
61. \* 取得呼出状态的log
62. \* 从而得到转换状态
63. \*/
64. @Override
65. **public** **void** run() {
66. Log.d(TAG, "开始读取日志记录");
68. String[] catchParams = {"logcat", "InCallScreen \*:s"};
69. String[] clearParams = {"logcat", "-c"};
71. **try** {
72. Process process=Runtime.getRuntime().exec(catchParams);
73. InputStream is = process.getInputStream();
74. BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(is));
76. String line = **null**;
77. **while** ((line=reader.readLine())!=**null**) {
78. logCount++;
79. //输出所有
80. Log.v(TAG, line);
82. //日志超过512条就清理
83. **if** (logCount>512) {
84. //清理日志
85. Runtime.getRuntime().exec(clearParams)
86. .destroy();//销毁进程，释放资源
87. logCount = 0;
88. Log.v(TAG, "-----------清理日志---------------");
89. }
91. /\*---------------------------------前台呼叫-----------------------\*/
92. //空闲
93. **if** (line.contains(ReadLog.CallViewState.FORE\_GROUND\_CALL\_STATE)
94. && line.contains(ReadLog.CallState.IDLE)) {
95. Log.d(TAG, ReadLog.CallState.IDLE);
96. }
98. //正在拨号，等待建立连接，即已拨号，但对方还没有响铃，
99. **if** (line.contains(ReadLog.CallViewState.FORE\_GROUND\_CALL\_STATE)
100. && line.contains(ReadLog.CallState.DIALING)) {
101. Log.d(TAG, ReadLog.CallState.DIALING);
102. }
104. //呼叫对方 正在响铃
105. **if** (line.contains(ReadLog.CallViewState.FORE\_GROUND\_CALL\_STATE)
106. && line.contains(ReadLog.CallState.ALERTING)) {
107. Log.d(TAG, ReadLog.CallState.ALERTING);
108. }
110. //已接通，通话建立
111. **if** (line.contains(ReadLog.CallViewState.FORE\_GROUND\_CALL\_STATE)
112. && line.contains(ReadLog.CallState.ACTIVE)) {
113. Log.d(TAG, ReadLog.CallState.ACTIVE);
114. }
116. //断开连接，即挂机
117. **if** (line.contains(ReadLog.CallViewState.FORE\_GROUND\_CALL\_STATE)
118. && line.contains(ReadLog.CallState.DISCONNECTED)) {
119. Log.d(TAG, ReadLog.CallState.DISCONNECTED);
120. }
122. } //END while
124. } **catch** (IOException e) {
125. e.printStackTrace();
126. } //END try-catch
127. } //END run
128. } //END class ReadLog

       以上代码中，之所以用线程，是为了防止读取日志过程中阻滞主方法的其他方法的执行，影响到程序捕捉对应的电话状态。

       好了，捕捉到了去电过程中各个状态的转变，那么，如何通知给程序呢，我采用的方法是捕获后立马给系统发送广播，然后程序进行广播接收，接收后再处理录音事件。要发送广播，就要发送一个唯一的广播，为此，建立如下类：

**Java代码**

1. **package** com.sdvdxl.outgoingcall;
3. **import** com.sdvdxl.phonerecorder.ReadLog;
5. **import** android.content.Context;
6. **import** android.util.Log;
8. **public** **class** OutgoingCallState {
9. Context ctx;
10. **public** OutgoingCallState(Context ctx) {
11. **this**.ctx = ctx;
12. }
14. /\*\*
15. \* 前台呼叫状态
16. \* @author sdvdxl
17. \*
18. \*/
19. **public** **static** **final** **class** ForeGroundCallState {
20. **public** **static** **final** String DIALING =
21. "com.sdvdxl.phonerecorder.FORE\_GROUND\_DIALING";
22. **public** **static** **final** String ALERTING =
23. "com.sdvdxl.phonerecorder.FORE\_GROUND\_ALERTING";
24. **public** **static** **final** String ACTIVE =
25. "com.sdvdxl.phonerecorder.FORE\_GROUND\_ACTIVE";
26. **public** **static** **final** String IDLE =
27. "com.sdvdxl.phonerecorder.FORE\_GROUND\_IDLE";
28. **public** **static** **final** String DISCONNECTED =
29. "com.sdvdxl.phonerecorder.FORE\_GROUND\_DISCONNECTED";
30. }
32. /\*\*
33. \* 开始监听呼出状态的转变，
34. \* 并在对应状态发送广播
35. \*/
36. **public** **void** startListen() {
37. **new** ReadLog(ctx).start();
38. Log.d("Recorder", "开始监听呼出状态的转变，并在对应状态发送广播");
39. }
41. }

       程序需要读取系统日志权限：

**XML/HTML代码**

1. **<uses-permission** android:name="android.permission.READ\_LOGS"**/>**

       然后，在读取日志的类中检测到去电各个状态的地方发送一个广播，那么，读取日志的完整代码如下：

**Java代码**

1. **package** com.sdvdxl.phonerecorder;
3. **import** java.io.BufferedReader;
4. **import** java.io.IOException;
5. **import** java.io.InputStream;
6. **import** java.io.InputStreamReader;
8. **import** com.sdvdxl.outgoingcall.OutgoingCallState;
10. **import** android.content.Context;
11. **import** android.content.Intent;
12. **import** android.util.Log;
14. /\*\*
15. \*
16. \* @author mrloong
17. \*  找到 日志中的
18. \*  onPhoneStateChanged: mForegroundCall.getState() 这个是前台呼叫状态
19. \*  mBackgroundCall.getState() 后台电话
20. \*  若 是 DIALING 则是正在拨号，等待建立连接，但对方还没有响铃，
21. \*  ALERTING 呼叫成功，即对方正在响铃，
22. \*  若是 ACTIVE 则已经接通
23. \*  若是 DISCONNECTED 则本号码呼叫已经挂断
24. \*  若是 IDLE 则是处于 空闲状态
25. \*
26. \*/
27. **public** **class** ReadLog **extends** Thread {
28. **private** Context ctx;
29. **private** **int** logCount;
31. **private** **static** **final** String TAG = "LogInfo OutGoing Call";
33. /\*\*
34. \*  前后台电话
35. \* @author sdvdxl
36. \*
37. \*/
38. **private** **static** **class** CallViewState {
39. **public** **static** **final** String FORE\_GROUND\_CALL\_STATE = "mForeground";
40. }
42. /\*\*
43. \* 呼叫状态
44. \* @author sdvdxl
45. \*
46. \*/
47. **private** **static** **class** CallState {
48. **public** **static** **final** String DIALING = "DIALING";
49. **public** **static** **final** String ALERTING = "ALERTING";
50. **public** **static** **final** String ACTIVE = "ACTIVE";
51. **public** **static** **final** String IDLE = "IDLE";
52. **public** **static** **final** String DISCONNECTED = "DISCONNECTED";
53. }
55. **public** ReadLog(Context ctx) {
56. **this**.ctx = ctx;
57. }
59. /\*\*
60. \* 读取Log流
61. \* 取得呼出状态的log
62. \* 从而得到转换状态
63. \*/
64. @Override
65. **public** **void** run() {
66. Log.d(TAG, "开始读取日志记录");
68. String[] catchParams = {"logcat", "InCallScreen \*:s"};
69. String[] clearParams = {"logcat", "-c"};
71. **try** {
72. Process process=Runtime.getRuntime().exec(catchParams);
73. InputStream is = process.getInputStream();
74. BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(is));
76. String line = **null**;
77. **while** ((line=reader.readLine())!=**null**) {
78. logCount++;
79. //输出所有
80. Log.v(TAG, line);
82. //日志超过512条就清理
83. **if** (logCount>512) {
84. //清理日志
85. Runtime.getRuntime().exec(clearParams)
86. .destroy();//销毁进程，释放资源
87. logCount = 0;
88. Log.v(TAG, "-----------清理日志---------------");
89. }
91. /\*---------------------------------前台呼叫-----------------------\*/
92. //空闲
93. **if** (line.contains(ReadLog.CallViewState.FORE\_GROUND\_CALL\_STATE)
94. && line.contains(ReadLog.CallState.IDLE)) {
95. Log.d(TAG, ReadLog.CallState.IDLE);
96. }
98. //正在拨号，等待建立连接，即已拨号，但对方还没有响铃，
99. **if** (line.contains(ReadLog.CallViewState.FORE\_GROUND\_CALL\_STATE)
100. && line.contains(ReadLog.CallState.DIALING)) {
101. //发送广播
102. Intent dialingIntent = **new** Intent();
103. dialingIntent.setAction(OutgoingCallState.ForeGroundCallState.DIALING);
104. ctx.sendBroadcast(dialingIntent);
106. Log.d(TAG, ReadLog.CallState.DIALING);
107. }
109. //呼叫对方 正在响铃
110. **if** (line.contains(ReadLog.CallViewState.FORE\_GROUND\_CALL\_STATE)
111. && line.contains(ReadLog.CallState.ALERTING)) {
112. //发送广播
113. Intent dialingIntent = **new** Intent();
114. dialingIntent.setAction(OutgoingCallState.ForeGroundCallState.ALERTING);
115. ctx.sendBroadcast(dialingIntent);
117. Log.d(TAG, ReadLog.CallState.ALERTING);
118. }
120. //已接通，通话建立
121. **if** (line.contains(ReadLog.CallViewState.FORE\_GROUND\_CALL\_STATE)
122. && line.contains(ReadLog.CallState.ACTIVE)) {
123. //发送广播
124. Intent dialingIntent = **new** Intent();
125. dialingIntent.setAction(OutgoingCallState.ForeGroundCallState.ACTIVE);
126. ctx.sendBroadcast(dialingIntent);
128. Log.d(TAG, ReadLog.CallState.ACTIVE);
129. }
131. //断开连接，即挂机
132. **if** (line.contains(ReadLog.CallViewState.FORE\_GROUND\_CALL\_STATE)
133. && line.contains(ReadLog.CallState.DISCONNECTED)) {
134. //发送广播
135. Intent dialingIntent = **new** Intent();
136. dialingIntent.setAction(OutgoingCallState.ForeGroundCallState.DISCONNECTED);
137. ctx.sendBroadcast(dialingIntent);
139. Log.d(TAG, ReadLog.CallState.DISCONNECTED);
140. }
142. } //END while
144. } **catch** (IOException e) {
145. e.printStackTrace();
146. } //END try-catch
147. } //END run
148. } //END class ReadLog

       发送了广播，那么就要有接收者，定义接收者如下（关于录音机的代码可以先忽略）：

**Java代码**

1. **package** com.sdvdxl.phonerecorder;
3. **import** android.content.BroadcastReceiver;
4. **import** android.content.Context;
5. **import** android.content.Intent;
6. **import** android.util.Log;
8. **import** com.sdvdxl.outgoingcall.OutgoingCallState;
10. **public** **class** OutgoingCallReciver **extends** BroadcastReceiver {
11. **static** **final** String TAG = "Recorder";
12. **private** MyRecorder recorder;
14. **public** OutgoingCallReciver() {
15. recorder = **new** MyRecorder();
16. }
18. **public**  OutgoingCallReciver (MyRecorder recorder) {
19. **this**.recorder = recorder;
20. }
22. @Override
23. **public** **void** onReceive(Context ctx, Intent intent) {
24. String phoneState = intent.getAction();
26. **if** (phoneState.equals(Intent.ACTION\_NEW\_OUTGOING\_CALL)) {
27. String phoneNum = intent.getStringExtra(Intent.EXTRA\_PHONE\_NUMBER);//拨出号码
28. recorder.setPhoneNumber(phoneNum);
29. recorder.setIsCommingNumber(**false**);
30. Log.d(TAG, "设置为去电状态");
31. Log.d(TAG, "去电状态 呼叫：" + phoneNum);
32. }
34. **if** (phoneState.equals(OutgoingCallState.ForeGroundCallState.DIALING)) {
35. Log.d(TAG, "正在拨号...");
36. }
38. **if** (phoneState.equals(OutgoingCallState.ForeGroundCallState.ALERTING)) {
39. Log.d(TAG, "正在呼叫...");
40. }
42. **if** (phoneState.equals(OutgoingCallState.ForeGroundCallState.ACTIVE)) {
43. **if** (!recorder.isCommingNumber() && !recorder.isStarted()) {
44. Log.d(TAG, "去电已接通 启动录音机");
45. recorder.start();
47. }
48. }
50. **if** (phoneState.equals(OutgoingCallState.ForeGroundCallState.DISCONNECTED)) {
51. **if** (!recorder.isCommingNumber() && recorder.isStarted()) {
52. Log.d(TAG, "已挂断 关闭录音机");
53. recorder.stop();
54. }
55. }
56. }
58. }

       其中有这么一段代码：

**Java代码**

1. String phoneState = intent.getAction();
3. **if** (phoneState.equals(Intent.ACTION\_NEW\_OUTGOING\_CALL)) {
4. String phoneNum = intent.getStringExtra(Intent.EXTRA\_PHONE\_NUMBER);//拨出号码
5. recorder.setPhoneNumber(phoneNum);
6. recorder.setIsCommingNumber(**false**);
7. Log.d(TAG, "设置为去电状态");
8. Log.d(TAG, "去电状态 呼叫：" + phoneNum);
9. }

       这里是接收系统发出的广播，用于接收去电广播。这样，就获得了去电状态。

****3、有了以上主要代码，可以说，来去电监听功能算是完成了，下面创建一个service来运行监听：****

**Java代码**

1. **package** com.sdvdxl.service;
3. **import** android.app.Service;
4. **import** android.content.Context;
5. **import** android.content.Intent;
6. **import** android.content.IntentFilter;
7. **import** android.os.IBinder;
8. **import** android.telephony.PhoneStateListener;
9. **import** android.telephony.TelephonyManager;
10. **import** android.util.Log;
11. **import** android.widget.Toast;
13. **import** com.sdvdxl.outgoingcall.OutgoingCallState;
14. **import** com.sdvdxl.phonerecorder.MyRecorder;
15. **import** com.sdvdxl.phonerecorder.OutgoingCallReciver;
16. **import** com.sdvdxl.phonerecorder.TelListener;
18. **public** **class** PhoneCallStateService **extends** Service {
19. **private** OutgoingCallState outgoingCallState;
20. **private** OutgoingCallReciver outgoingCallReciver;
21. **private** MyRecorder recorder;
23. @Override
24. **public** **void** onCreate() {
25. **super**.onCreate();
27. //------以下应放在onStartCommand中，但2.3.5以下版本不会因service重新启动而重新调用--------
28. //监听电话状态，如果是打入且接听 或者 打出 则开始自动录音
29. //通话结束，保存文件到外部存储器上
30. Log.d("Recorder", "正在监听中...");
31. recorder = **new** MyRecorder();
32. outgoingCallState = **new** OutgoingCallState(**this**);
33. outgoingCallReciver = **new** OutgoingCallReciver(recorder);
34. outgoingCallState.startListen();
35. Toast.makeText(**this**, "服务已启动", Toast.LENGTH\_LONG).show();
37. //去电
38. IntentFilter outgoingCallFilter = **new** IntentFilter();
39. outgoingCallFilter.addAction(OutgoingCallState.ForeGroundCallState.IDLE);
40. outgoingCallFilter.addAction(OutgoingCallState.ForeGroundCallState.DIALING);
41. outgoingCallFilter.addAction(OutgoingCallState.ForeGroundCallState.ALERTING);
42. outgoingCallFilter.addAction(OutgoingCallState.ForeGroundCallState.ACTIVE);
43. outgoingCallFilter.addAction(OutgoingCallState.ForeGroundCallState.DISCONNECTED);
45. outgoingCallFilter.addAction("android.intent.action.PHONE\_STATE");
46. outgoingCallFilter.addAction("android.intent.action.NEW\_OUTGOING\_CALL");
48. //注册接收者
49. registerReceiver(outgoingCallReciver, outgoingCallFilter);
51. //来电
52. TelephonyManager telmgr = (TelephonyManager)getSystemService(
53. Context.TELEPHONY\_SERVICE);
54. telmgr.listen(**new** TelListener(recorder), PhoneStateListener.LISTEN\_CALL\_STATE);

57. }
59. @Override
60. **public** IBinder onBind(Intent intent) {
61. // TODO Auto-generated method stub
62. **return** **null**;
63. }
65. @Override
66. **public** **void** onDestroy() {
67. **super**.onDestroy();
68. unregisterReceiver(outgoingCallReciver);
69. Toast.makeText(
70. **this**, "已关闭电话监听服务", Toast.LENGTH\_LONG)
71. .show();
72. Log.d("Recorder", "已关闭电话监听服务");
73. }
75. @Override
76. **public** **int** onStartCommand(Intent intent, **int** flags, **int** startId) {
78. **return** START\_STICKY;
79. }
81. }

       注册以下service：

**XML/HTML代码**

1. **<service** android:name="com.sdvdxl.service.PhoneCallStateService" **/>**

       到此为止，来去电状态的监听功能算是完成了，剩下一个录音机，附上录音机代码如下：

**Java代码**

1. **package** com.sdvdxl.phonerecorder;
3. **import** java.io.File;
4. **import** java.io.IOException;
5. **import** java.text.SimpleDateFormat;
6. **import** java.util.Date;
8. **import** android.media.MediaRecorder;
9. **import** android.os.Environment;
10. **import** android.util.Log;
12. **public** **class** MyRecorder {
13. **private** String phoneNumber;
14. **private** MediaRecorder mrecorder;
15. **private** **boolean** started = **false**; //录音机是否已经启动
16. **private** **boolean** isCommingNumber = **false**;//是否是来电
17. **private** String TAG = "Recorder";

20. **public** MyRecorder(String phoneNumber) {
21. **this**.setPhoneNumber(phoneNumber);
22. }
24. **public** MyRecorder() {
25. }
27. **public** **void** start() {
28. started = **true**;
29. mrecorder = **new** MediaRecorder();
31. File recordPath = **new** File(
32. Environment.getExternalStorageDirectory()
33. , "/My record");
34. **if** (!recordPath.exists()) {
35. recordPath.mkdirs();
36. Log.d("recorder", "创建目录");
37. }
39. String callDir = "呼出";
40. **if** (isCommingNumber) {
41. callDir = "呼入";
42. }
43. String fileName = callDir + "-" + phoneNumber + "-"
44. + **new** SimpleDateFormat("yy-MM-dd\_HH-mm-ss")
45. .format(**new** Date(System.currentTimeMillis())) + ".mp3";//实际是3gp
46. File recordName = **new** File(recordPath, fileName);
48. **try** {
49. recordName.createNewFile();
50. Log.d("recorder", "创建文件" + recordName.getName());
51. } **catch** (IOException e) {
52. e.printStackTrace();
53. }
55. mrecorder.setAudioSource(MediaRecorder.AudioSource.DEFAULT);
56. mrecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
57. mrecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
59. mrecorder.setOutputFile(recordName.getAbsolutePath());
61. **try** {
62. mrecorder.prepare();
63. } **catch** (IllegalStateException e) {
64. e.printStackTrace();
65. } **catch** (IOException e) {
66. e.printStackTrace();
67. }
68. mrecorder.start();
69. started = **true**;
70. Log.d(TAG , "录音开始");
71. }
73. **public** **void** stop() {
74. **try** {
75. **if** (mrecorder!=**null**) {
76. mrecorder.stop();
77. mrecorder.release();
78. mrecorder = **null**;
79. }
80. started = **false**;
81. } **catch** (IllegalStateException e) {
82. e.printStackTrace();
83. }

86. Log.d(TAG , "录音结束");
87. }
89. **public** **void** pause() {
91. }
93. **public** String getPhoneNumber() {
94. **return** phoneNumber;
95. }
97. **public** **void** setPhoneNumber(String phoneNumber) {
98. **this**.phoneNumber = phoneNumber;
99. }
101. **public** **boolean** isStarted() {
102. **return** started;
103. }
105. **public** **void** setStarted(**boolean** hasStarted) {
106. **this**.started = hasStarted;
107. }
109. **public** **boolean** isCommingNumber() {
110. **return** isCommingNumber;
111. }
113. **public** **void** setIsCommingNumber(**boolean** isCommingNumber) {
114. **this**.isCommingNumber = isCommingNumber;
115. }
117. }

       到此，来去电通话自动录音的所有功能就完成了，大家可以自己试着编写并实现。