 在Android中并没有定义MediaButtonReceive这个广播类，MediaButtonReceive只是作为一种通俗的命名方式来响应

   插入耳机后，点击耳机上的按钮(名称：MEDIA\_BUTTON)接受该广播事件的类。所有该MEDIA\_BUTTON的按下我们就简称

   为MEDIA\_BUTTON广播吧。

           顾名思义：它显然是一个广播接收器类(BroadbcastReceiver)，那么它就具备了BroadbcastReceiver类的使用方式，

   但是，因为它需要通过AudioManager对象注册，所以它有着自己的独特之处(否则我也不会单独拿出来分析，- -)，后面我们

   会慢慢的讲解。

        点击MEDIA\_BUTTON发送的Intent Action 为：

**ACTION\_MEDIA\_BUTTON  =**"android.intent.action.MEDIA\_BUTTON"

        Intent 附加值为(Extra)点击MEDIA\_BUTTON的按键码 ：

                        //获得KeyEvent对象

                        KeyEvent keyEvent = (KeyEvent)intent.getParcelableExtra(Intent.*EXTRA\_KEY\_EVENT*);

                        //获得Action

                        String intentAction = intent.getAction() ;

AudioManager对象注册MEDIA\_BUTTON广播的方法原型为：

   public void**registerMediaButtonEventReceiver**([ComponentName](file:///F:\android-sdk-windows\docs\reference\android\content\ComponentName.html)eventReceiver)

          Register a component to be the**sole receiver**of MEDIA\_BUTTON intents

     Parameters：

                eventReceiver  ： identifier of a BroadcastReceiver that will receive the media button intent. This broadcast receiver

                                   must be declared in the application manifest.

   从注释可知以下两点：

      1、 在AudioManager对象注册一个MediaoButtonRecevie，使它成为MEDIA\_BUTTON的**唯一接收器**(这很重要，

          我们会放在后面讲解)**也就是说只有我能收到，其他的都收不到这个广播了，否则的话大家都收到会照成一定的混乱；**

      2、   该广播必须在AndroidManifest.xml文件中进行声明，否则就监听不到该MEDIA\_BUTTON广播了。

下面我们就简单的写一个MediaButtonReceiver类，并且在AndroidManifest.xml定义

1、  自定义的MediaButtonReceiver 广播类

**[java]** [view plaincopyprint?](http://blog.csdn.net/qinjuning/article/details/6938436)

1. **package** com.qin.mediabutton;
3. **import** android.content.BroadcastReceiver;
4. **import** android.content.Context;
5. **import** android.content.Intent;
6. **import** android.util.Log;
7. **import** android.view.KeyEvent;
9. **public** **class** MediaButtonReceiver **extends** BroadcastReceiver {
10. **private** **static** String TAG = "MediaButtonReceiver";
11. @Override
12. **public** **void** onReceive(Context context, Intent intent) {
13. // 获得Action
14. String intentAction = intent.getAction();
15. // 获得KeyEvent对象
16. KeyEvent keyEvent = (KeyEvent) intent.getParcelableExtra(Intent.EXTRA\_KEY\_EVENT);
18. Log.i(TAG, "Action ---->" + intentAction + "  KeyEvent----->"+ keyEvent.toString());
20. **if** (Intent.ACTION\_MEDIA\_BUTTON.equals(intentAction)) {
21. // 获得按键字节码
22. **int** keyCode = keyEvent.getKeyCode();
23. // 按下 / 松开 按钮
24. **int** keyAction = keyEvent.getAction();
25. // 获得事件的时间
26. **long** downtime = keyEvent.getEventTime();
28. // 获取按键码 keyCode
29. StringBuilder sb = **new** StringBuilder();
30. // 这些都是可能的按键码 ， 打印出来用户按下的键
31. **if** (KeyEvent.KEYCODE\_MEDIA\_NEXT == keyCode) {
32. sb.append("KEYCODE\_MEDIA\_NEXT");
33. }
34. // 说明：当我们按下MEDIA\_BUTTON中间按钮时，实际出发的是 KEYCODE\_HEADSETHOOK 而不是
35. // KEYCODE\_MEDIA\_PLAY\_PAUSE
36. **if** (KeyEvent.KEYCODE\_MEDIA\_PLAY\_PAUSE == keyCode) {
37. sb.append("KEYCODE\_MEDIA\_PLAY\_PAUSE");
38. }
39. **if** (KeyEvent.KEYCODE\_HEADSETHOOK == keyCode) {
40. sb.append("KEYCODE\_HEADSETHOOK");
41. }
42. **if** (KeyEvent.KEYCODE\_MEDIA\_PREVIOUS == keyCode) {
43. sb.append("KEYCODE\_MEDIA\_PREVIOUS");
44. }
45. **if** (KeyEvent.KEYCODE\_MEDIA\_STOP == keyCode) {
46. sb.append("KEYCODE\_MEDIA\_STOP");
47. }
48. // 输出点击的按键码
49. Log.i(TAG, sb.toString());
50. }
51. }
52. }

  2、  在AndroidManifest.xml声明我们定义的广播类。

**[java]** [view plaincopyprint?](http://blog.csdn.net/qinjuning/article/details/6938436)

1. <receiver android:name="MediaButtonReceiver">
2. <intent-filter >
3. <action android:name="android.intent.action.MEDIA\_BUTTON"></action>
4. </intent-filter>
5. </receiver>

         在模拟器上，我们可以手动构造MEDA\_BUTTON的广播，并且将它发送出去(后面会介绍)。

         如果有真机测试的话，按下MEDIA\_BUTTON是可以接受到MEDIA\_BUTTON广播的，如果没有接受到，请关闭所有应用

   程序，在观察效果。

  继续我们的下一步分析:

         前面我们说明通过**registerMediaButtonEventReceiver**(eventReceiver)方法注册时**，**使它成为MEDIA\_BUTTON的

**唯一 接收器**。这个唯一是怎么实现的呢？ 我们在源码中，一步步追本溯源，相信一定可以找到答案，知道这“唯一“是

    怎么来的。

第一步、   为AudioManager注册一个MediaButtonReceiver() ;

**[java]** [view plaincopyprint?](http://blog.csdn.net/qinjuning/article/details/6938436)

1. //获得AudioManager对象
2. AudioManager mAudioManager =(AudioManager)getSystemService(Context.AUDIO\_SERVICE);
3. //构造一个ComponentName，指向MediaoButtonReceiver类
4. //下面为了叙述方便，我直接使用ComponentName类来替代MediaoButtonReceiver类
5. ComponentName  mbCN = **new** ComponentName(getPackageName(),MediaButtonReceiver.**class**.getName());
6. //注册一个MedioButtonReceiver广播监听
7. mAudioManager.registerMediaButtonEventReceiver(mbCN);
8. //取消注册的方法
9. mAudioManager.unregisterMediaButtonEventReceiver(mbCN);

        MediaButtonReceiver就是我们用来接收MEDIA\_BUTTON的广播类，下面为了叙述方便和直观上得体验，我直接使用

**ComponentName类来替代真正的MediaoButtonReceiver广播类。**

   说明 接下来分析的文件路径全部在   frameworks/base/media/java/android/media/ 下

 第二步、 进入AudioManager.java进行查看 ，发现如下方法：

**[java]** [view plaincopyprint?](http://blog.csdn.net/qinjuning/article/details/6938436)

1. //注册的方法为：
2. **public** **void** registerMediaButtonEventReceiver(ComponentName eventReceiver) {
3. //TODO enforce the rule about the receiver being declared in the manifest
4. //我们继续查看getService()方法，看看IAudioService类到底是什么？
5. IAudioService service = getService();
6. **try** {
7. //只是简单的调用了service的方法来完成注册，继续跟踪
8. service.registerMediaButtonEventReceiver(eventReceiver);
10. } **catch** (RemoteException e) {
11. Log.e(TAG, "Dead object in registerMediaButtonEventReceiver"+e);
12. }
13. }
14. //取消注册的方法为
15. **public** **void** unregisterMediaButtonEventReceiver(ComponentName eventReceiver) {
16. IAudioService service = getService();
17. **try** {
18. //只是简单的调用了service的方法来取消注册，，继续跟踪
19. service.unregisterMediaButtonEventReceiver(eventReceiver);
20. } **catch** (RemoteException e) {
21. Log.e(TAG, "Dead object in unregisterMediaButtonEventReceiver"+e);
22. }
23. }

  找到getService()方法，其实现为:

**[java]** [view plaincopyprint?](http://blog.csdn.net/qinjuning/article/details/6938436)

1. //看看它到底是什么
2. **private** **static** IAudioService getService()
3. {
4. // 单例模式，大家懂得
5. **if** (sService != **null**) {
6. **return** sService;
7. }
8. //了解Binder机制 以及AIDL文件的使用，就明白了这不过是通过AIDL文件定义的Java层Binder机制
9. //b为IBinder基类接口
10. IBinder b = ServiceManager.getService(Context.AUDIO\_SERVICE);
11. //强制转换后,sService不过是一个客户端对象，IAudioService就是aidl文件定义的接口了
12. sService = IAudioService.Stub.asInterface(b);
13. **return** sService;
14. }
15. //sService对象的声明
16. **private** **static** IAudioService sService; //单例模式，不足为奇了

         我们知道了AudiaoManager只不过是一个傀儡，所有的方法都是由IAudioService 对象去实现的，通过它的构造方式，

  可以知道它应该是有AIDL文件形成的Binder机制， sService只是客户端对象，那么它的服务端对象在什么地方呢？

  也就是继承了IAudioService.Stub桩的类。

第三步、接下来我们需要找到该IAudioService.aidl文件和真正的服务端对象

   IAudioService.aidl定义如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/qinjuning/article/details/6938436)

1. **package** android.media;
3. **import** android.content.ComponentName;
4. **import** android.media.IAudioFocusDispatcher;
5. /\*\*
6. \* {@hide}
7. \*/
8. **interface** IAudioService {
10. **void** adjustVolume(**int** direction, **int** flags);
11. **void** adjustSuggestedStreamVolume(**int** direction, **int** suggestedStreamType, **int** flags);
12. **void** adjustStreamVolume(**int** streamType, **int** direction, **int** flags);
13. **void** setStreamVolume(**int** streamType, **int** index, **int** flags);
14. **void** setStreamSolo(**int** streamType, **boolean** state, IBinder cb);
15. **void** setStreamMute(**int** streamType, **boolean** state, IBinder cb);
16. **int** getStreamVolume(**int** streamType);
17. **int** getStreamMaxVolume(**int** streamType);
18. **void** setRingerMode(**int** ringerMode);
19. **int** getRingerMode();
20. **void** setVibrateSetting(**int** vibrateType, **int** vibrateSetting);
21. **int** getVibrateSetting(**int** vibrateType);
22. **boolean** shouldVibrate(**int** vibrateType);
23. **void** setMode(**int** mode, IBinder cb);
24. **int** getMode();
25. oneway **void** playSoundEffect(**int** effectType);
26. oneway **void** playSoundEffectVolume(**int** effectType, **float** volume);
27. **boolean** loadSoundEffects();
28. oneway **void** unloadSoundEffects();
29. oneway **void** reloadAudioSettings();
30. **void** setSpeakerphoneOn(**boolean** on);
31. **boolean** isSpeakerphoneOn();
32. **void** setBluetoothScoOn(**boolean** on);
33. **boolean** isBluetoothScoOn();
34. **int** requestAudioFocus(**int** mainStreamType, **int** durationHint, IBinder cb, IAudioFocusDispatcher l, String clientId);
35. **int** abandonAudioFocus(IAudioFocusDispatcher l, String clientId);
36. **void** unregisterAudioFocusClient(String clientId);
37. **void** registerMediaButtonEventReceiver(in ComponentName eventReceiver);   //这个方法是我们需要弄懂的
38. **void** unregisterMediaButtonEventReceiver(in ComponentName eventReceiver);  //这个方法也是是我们需要弄懂的
39. **void** startBluetoothSco(IBinder cb);
40. **void** stopBluetoothSco(IBinder cb);
41. }

       真正的服务端对象就是继承了 IAudioService.Stub 桩的类，AudioService就是该服务端对象，其实AudioManager的

  所有操作都是由AudioService来实现的，它才是真正的老大。

第五步、   AudioService.java

**[java]** [view plaincopyprint?](http://blog.csdn.net/qinjuning/article/details/6938436)

1. //AudioService类
2. **public** **class** AudioService **extends** IAudioService.Stub {
3. //.....
4. //仅仅列出我们需要的方法
5. //这儿才是真正的注册MediaButtonReceiver的方法
6. **public** **void** registerMediaButtonEventReceiver(ComponentName eventReceiver) {
7. Log.i(TAG, "  Remote Control   registerMediaButtonEventReceiver() for " + eventReceiver);
9. **synchronized**(mRCStack) {
10. //调用它去实现注册ComponentName
11. pushMediaButtonReceiver(eventReceiver);
12. }
13. }
15. //在查看pushMediaButtonReceiver()方法  先理解一下两个知识点，很重要的。
16. //RemoteControlStackEntry内部类不过是对ComponentName类的进一步封装（感觉没必要在加一层进行封装了）
17. **private** **static** **class** RemoteControlStackEntry {
18. **public** ComponentName mReceiverComponent;// 属性
19. //TODO implement registration expiration?
20. //public int mRegistrationTime;
22. **public** RemoteControlStackEntry() {
23. }
25. **public** RemoteControlStackEntry(ComponentName r) {
26. mReceiverComponent = r;// 构造函数赋值给mReceiverComponent对象
27. }
28. }
30. //采用了栈存储结构（先进后出）来保存所有RemoteControlStackEntry对象，也就是保存了ComponentName对象
31. **private** Stack<RemoteControlStackEntry> mRCStack = **new** Stack<RemoteControlStackEntry>();
33. //回到pushMediaButtonReceiver()查看，这下该拨开云雾了吧，继续学习
34. **private** **void** pushMediaButtonReceiver(ComponentName newReceiver) {
35. // already at top of stack?
36. //采用了一个栈(前面我们介绍的知识点)来保存所有注册的ComponentName对象
37. //如果当前栈不为空并且栈顶的对象与新注册的ComponentName对象一样，不做任何事，直接返回
38. **if** (!mRCStack.empty() && mRCStack.peek().mReceiverComponent.equals(newReceiver)) {
39. **return**;
40. }
41. //获得mRCStack栈的迭代器
42. Iterator<RemoteControlStackEntry> stackIterator = mRCStack.iterator();
43. //循环
44. **while**(stackIterator.hasNext()) {
45. RemoteControlStackEntry rcse = (RemoteControlStackEntry)stackIterator.next();
46. //如果当前栈内保存该新注册的ComponentName对象，将它移除，跳出循环
47. **if**(rcse.mReceiverComponent.equals(newReceiver)) {
48. mRCStack.remove(rcse);
49. **break**;
50. }
51. }
52. //将新注册的ComponentName对象放入栈顶
53. mRCStack.push(**new** RemoteControlStackEntry(newReceiver));
54. }
55. }

**小结一下：**

**栈(mRCStack)维护了所有CompoentName对象，对每个CompoentName对象，保证它有且仅有一个，**

**新注册的CompoentName对象永远处于栈顶**

 我们看下取消注册的方法：

**[java]** [view plaincopyprint?](http://blog.csdn.net/qinjuning/article/details/6938436)

1. //我们看下取消注册的方法
2. /\*\* see AudioManager.unregisterMediaButtonEventReceiver(ComponentName eventReceiver) \*/
3. **public** **void** unregisterMediaButtonEventReceiver(ComponentName eventReceiver) {
4. Log.i(TAG, "  Remote Control   unregisterMediaButtonEventReceiver() for " + eventReceiver);
6. **synchronized**(mRCStack) {
7. //调用removeMediaButtonReceiver方法去实现
8. removeMediaButtonReceiver(eventReceiver);
9. }
10. }
12. **private** **void** removeMediaButtonReceiver(ComponentName newReceiver) {
13. Iterator<RemoteControlStackEntry> stackIterator = mRCStack.iterator();
14. **while**(stackIterator.hasNext()) {
15. //获得mRCStack栈的迭代器
16. RemoteControlStackEntry rcse = (RemoteControlStackEntry)stackIterator.next();
17. //如果存在该对象，则移除，跳出循环
18. **if**(rcse.mReceiverComponent.equals(newReceiver)) {
19. mRCStack.remove(rcse);
20. **break**;
21. }
22. }
23. }

          通过对前面的学习，我们知道了AudioManager内部利用一个栈来管理包括加入和移除ComponentName对象，

    新的疑问来了？这个MEDIA\_BUTTON广播是如何分发的呢 ？

         其实，AudioService.java文件中也存在这么一个MediaoButtonReceiver的广播类，它为系统广播接收器，即用来接收

  系统的MEDIA\_BUTTON广播，当它接收到了这个MEDIA\_BUTTON广播   ，它会对这个广播进行进一步处理，这个处理过程

   就是我们需要的弄清楚。

MediaButtonBroadcastReceiver 内部类如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/qinjuning/article/details/6938436)

1. **private** **class** MediaButtonBroadcastReceiver **extends** BroadcastReceiver {
2. @Override
3. **public** **void** onReceive(Context context, Intent intent) {
4. //获得action ，系统MEDIA\_BUTTON广播来了
5. String action = intent.getAction();
6. //action不正确 直接返回
7. **if** (!Intent.ACTION\_MEDIA\_BUTTON.equals(action)) {
8. **return**;
9. }
10. //获得KeyEvent对象
11. KeyEvent event = (KeyEvent) intent.getParcelableExtra(Intent.EXTRA\_KEY\_EVENT);
12. **if** (event != **null**) {
13. // if in a call or ringing, do not break the current phone app behavior
14. // TODO modify this to let the phone app specifically get the RC focus
15. //      add modify the phone app to take advantage of the new API
16. //来电或通话中，不做处理直接返回
17. **if** ((getMode() == AudioSystem.MODE\_IN\_CALL) ||(getMode() == AudioSystem.MODE\_RINGTONE)) {
18. **return**;
19. }
20. **synchronized**(mRCStack) {
21. //栈不为空
22. **if** (!mRCStack.empty()) {
23. // create a new intent specifically aimed at the current registered listener
24. //构造一个Intent对象 ，并且赋予Action和KeyEvent
25. Intent targetedIntent = **new** Intent(Intent.ACTION\_MEDIA\_BUTTON);
26. targetedIntent.putExtras(intent.getExtras());
27. //指定该处理Intent的对象为栈顶ComponentName对象的广播类
28. targetedIntent.setComponent(mRCStack.peek().mReceiverComponent);
29. // trap the current broadcast
30. // 终止系统广播
31. abortBroadcast();
32. //Log.v(TAG, " Sending intent" + targetedIntent);
33. //手动发送该广播至目标对象去处理，该广播不再是系统发送的了
34. context.sendBroadcast(targetedIntent, **null**);
35. }
36. //假设栈为空，那么所有定义在AndroidManifest.xml的监听MEDIA\_BUTTON的广播都会处理，
37. //在此过程中如果有任何应用程注册了registerMediaButton 该广播也会立即终止
38. }
39. }
40. }
41. }

**总结一下MEDIA\_BUTTON广播：**

         AudioManager也就是AudioService服务端对象内部会利用一个栈来管理所有ComponentName对象，所有对象有且仅有一个，

   新注册的ComponentName总是会位于栈顶。

         当系统发送MEDIA\_BUTTON，系统MediaButtonBroadcastReceiver 监听到系统广播，它会做如下处理：

**1、 如果栈为空，则所有注册了该Action的广播都会接受到，因为它是由系统发送的。  
                 2、 如果栈不为空，那么只有栈顶的那个广播能接受到MEDIA\_BUTTON的广播，手动发送了MEDIA\_BUTTON**

**广播，并且指定了目标对象(栈顶对象)去处理该MEDIA\_BUTTON 。**

 下面分析一下KeyEvent对象里的KeyCode按键，可能的按键码有：

       1、KeyEvent.KEYCODE\_MEDIA\_NEXT  
       2、KeyEvent.KEYCODE\_HEADSETHOOK   
       3、KeyEvent.KEYCODE\_MEDIA\_PLAY\_PAUSE(已废除，等同于KEYCODE\_HEADSETHOOK)  
       4、KeyEvent.KEYCODE\_MEDIA\_PREVIOUS  
       5、KeyEvent.KEYCODE\_MEDIA\_STOP   
      
    PS : 在我的真机测试中，按下MEDIA\_BUTTON只有KEYCODE\_HEADSETHOOK可以打印出来了。

下面给出一个小DEMO检验一下我们之前所做的一切，看看MEDIA\_BUTTON是如何处理分发广播的。

   编写两个MediaButtonReceiver类用来监听MEDIA\_BUTTON广播：

**1 、China\_MBReceiver.java**

**[java]** [view plaincopyprint?](http://blog.csdn.net/qinjuning/article/details/6938436)

1. **package** com.qin.mediabutton;
3. **import** android.content.BroadcastReceiver;
4. **import** android.content.Context;
5. **import** android.content.Intent;
6. **import** android.util.Log;
7. **import** android.view.KeyEvent;
9. **public** **class** China\_MBReceiver **extends** BroadcastReceiver  {
11. **private** **static** String TAG = "China\_MBReceiver" ;
12. @Override
13. **public** **void** onReceive(Context context, Intent intent) {
14. //获得Action
15. String intentAction = intent.getAction() ;
16. //获得KeyEvent对象
17. KeyEvent keyEvent = (KeyEvent)intent.getParcelableExtra(Intent.EXTRA\_KEY\_EVENT);
19. Log.i(TAG, "Action ---->"+intentAction + "  KeyEvent----->"+keyEvent.toString());
21. **if**(Intent.ACTION\_MEDIA\_BUTTON.equals(intentAction)){
22. //获得按键字节码
23. **int** keyCode = keyEvent.getKeyCode() ;
24. //按下 / 松开 按钮
25. **int** keyAction = keyEvent.getAction() ;
26. //获得事件的时间
27. **long** downtime = keyEvent.getEventTime();
29. //获取按键码 keyCode
30. StringBuilder sb = **new** StringBuilder();
31. //这些都是可能的按键码 ， 打印出来用户按下的键
32. **if**(KeyEvent.KEYCODE\_MEDIA\_NEXT == keyCode){
33. sb.append("KEYCODE\_MEDIA\_NEXT");
34. }
35. //说明：当我们按下MEDIA\_BUTTON中间按钮时，实际出发的是 KEYCODE\_HEADSETHOOK 而不是 KEYCODE\_MEDIA\_PLAY\_PAUSE
36. **if**(KeyEvent.KEYCODE\_MEDIA\_PLAY\_PAUSE ==keyCode){
37. sb.append("KEYCODE\_MEDIA\_PLAY\_PAUSE");
38. }
39. **if**(KeyEvent.KEYCODE\_HEADSETHOOK == keyCode){
40. sb.append("KEYCODE\_HEADSETHOOK");
41. }
42. **if**(KeyEvent.KEYCODE\_MEDIA\_PREVIOUS ==keyCode){
43. sb.append("KEYCODE\_MEDIA\_PREVIOUS");
44. }
45. **if**(KeyEvent.KEYCODE\_MEDIA\_STOP ==keyCode){
46. sb.append("KEYCODE\_MEDIA\_STOP");
47. }
48. //输出点击的按键码
49. Log.i(TAG, sb.toString());
51. }
53. }
55. }

**2 、England\_MBReceiver.java**同于China\_MBRreceiver**，打印**Log TAG**= "England\_MBReceiver"**

**3、在AndroidManifest.xml文件定义：**

**[java]** [view plaincopyprint?](http://blog.csdn.net/qinjuning/article/details/6938436)

1. <strong>  <receiver android:name=".China\_MBReceiver">
2. <intent-filter >
3. <action android:name="android.intent.action.MEDIA\_BUTTON"></action>
4. </intent-filter>
5. </receiver>
7. <receiver android:name=".Enaland\_MBReceiver">
8. <intent-filter >
9. <action android:name="android.intent.action.MEDIA\_BUTTON"></action>
10. </intent-filter>
11. </receiver></strong>

**4、MainActivity .java** 我们通过手动构造一个MEDIA\_BUTTON广播去查看我们的MediaButtonReceiver类的打印信息。

**[java]** [view plaincopyprint?](http://blog.csdn.net/qinjuning/article/details/6938436)

1. **package** com.qin.mediabutton;
3. **import** android.app.Activity;
4. **import** android.content.ComponentName;
5. **import** android.content.Context;
6. **import** android.content.Intent;
7. **import** android.media.AudioManager;
8. **import** android.os.Bundle;
9. **import** android.view.KeyEvent;
11. **public** **class** MainActivity **extends** Activity {
12. /\*\* Called when the activity is first created. \*/
13. @Override
14. **public** **void** onCreate(Bundle savedInstanceState) {
15. **super**.onCreate(savedInstanceState);
16. setContentView(R.layout.main);
18. //由于在模拟器上测试，我们手动发送一个MEDIA\_BUTTON的广播,有真机更好处理了
19. Intent mbIntent = **new** Intent(Intent.ACTION\_MEDIA\_BUTTON);
20. //构造一个KeyEvent对象
21. KeyEvent keyEvent = **new** KeyEvent (KeyEvent.ACTION\_DOWN,KeyEvent.KEYCODE\_HEADSETHOOK) ;
22. //作为附加值添加至mbIntent对象中
23. mbIntent.putExtra(Intent.EXTRA\_KEY\_EVENT, keyEvent);
25. //此时China\_MBReceiver和England\_MBReceiver都会接收到该广播
26. sendBroadcast(mbIntent);

29. AudioManager mAudioManager = (AudioManager)getSystemService(Context.AUDIO\_SERVICE);
30. //AudioManager注册一个MediaButton对象
31. ComponentName chinaCN = **new** ComponentName(getPackageName(),China\_MBReceiver.**class**.getName());
32. //只有China\_MBReceiver能够接收到了，它是出于栈顶的。
33. //不过，在模拟上检测不到这个效果，因为这个广播是我们发送的，流程不是我们在上面介绍的。
34. mAudioManager.registerMediaButtonEventReceiver(chinaCN);
35. //sendBroadcast(mbIntent,null);
36. }
37. //当一个Activity/Service死去时，我们需要取消这个MediaoButtonReceiver的注册，如下
38. **protected** **void** onDestroy(){
39. **super**.onDestroy() ;
40. AudioManager mAudioManager = (AudioManager)getSystemService(Context.AUDIO\_SERVICE);
41. ComponentName chinaCN = **new** ComponentName(getPackageName(),China\_MBReceiver.**class**.getName());
42. //取消注册
43. mAudioManager.unregisterMediaButtonEventReceiver(chinaCN);
44. }
45. }

       值得注意的一点时，当我们为一个应用程序注册了MediaoButtonReceiver时，在程序离开时，我们需要**取消**该

  MediaoButtonReceiver的注册，在onDestroy()调用unregisterMediaButtonEventReceiver()方法就OK，这样应用程序之间

  的交互就更具逻辑性了。