第十七章. Android广播机制

`android`

学习目标：

- BroadcastReceiver概述
- BroadcastReceiver的创建
- BroadcastReceiver的两种注册方法和使用范围
- 异步广播的发送方式
- 有序广播的发送和拦截
- 系统级提醒AlarmManager的使用方式

学员作业
系统广播

课程内容：

BroadcastReceiver简介

BroadcastReceiver是用于接收广播的组件用于组件与组件之间进行通信，可以跨应用程序传递.如操作系统电池电量低会发送一个广播，这样我们的程序里面可以去监听这个广播，可以关闭程序里面比较耗电的功能,提示用户注意保存进度什么的,还如其它安装新应用等,还有普通应用程序,例如启动特定线程,文件下载完毕等。

Android中的广播机制设计的非常出色，很多事情原本需要开发者亲自操作的，现在只需等待广播告知自己就可以了，大大减少了开发的工作量和开发周期。

- 发送频率低的可以使用
- 数据量小在可使用

BroadcastReceiver创建

要创建自己的BroadcastReceiver对象，我们需要继承`android.content.BroadcastReceiver`，并实现其`onReceive`方法。
在`onReceive`方法内，我们可以获取随广播而来的Intent中的数据，这非常重要，就像无线电一样，包含很多有用的信息。
在创建完我们的BroadcastReceiver之后，还不能够使它进入工作状态，我们需要为它注册一个指定的广播地址。没有注册广播地址的BroadcastReceiver就像一个缺少选台按钮的收音机，虽然功能俱备，但也无法收到电台的信号

BroadcastReceiver注册

静态注册

静态注册是在AndroidManifest.xml文件中配置的

```
1.          <receiver
    android:name="com.hua.bcreceiver.MyBroadcastReceiver" >
2.              <intent-filter>
3.                  <action
    android:name="android.intent.action.MY_BROADCAST" />
4.                  <category
    android:name="android.intent.category.DEFAULT" />
5.              </intent-filter>
6.          </receiver>
```

动态注册

动态注册需要在代码中动态的指定广播地址并注册

> 注意，`registerReceiver`是`android.content.ContextWrapper`类中的方法，Activity和Service都继承了ContextWrapper，所以可以直接调用。在实际应用中，我们在Activity或Service中注册了一个BroadcastReceiver，当这个Activity或Service被销毁时如果没有解除注册，系统会报一个异常，提示我们是否忘记解除注册了。所以，记得在特定的地方执行解除注册操作

注册

```
1.          MyBroadcastReceiver mbcr = new MyBroadcastReceiver();
2.          IntentFilter filter = new IntentFilter();
3.          filter.addAction(MyBroadcastReceiver.MY_BC_FIRST);
4.          registerReceiver(mbcr, filter);// 注册
```

解除注册

```
1.  @Override
2.      protected void onDestroy() {
3.          unregisterReceiver(mbcr);
4.          mbcr = null;
5.          super.onDestroy();
6.      }
```

发送广播

问题来了：
如果有多个接收者都注册了相同的广播地址，又会是什么情况呢，能同时接收到同一条广播吗，相互之间会不会有干扰呢？这就涉及到普通广播和有序广播的概念了。

普通广播

普通广播对于多个接收者来说是完全异步的，通常每个接收者都无需等待即可以接收到广播，接收者相互之间不会有影响。对于这种广播，接收者无法终止广播，即无法阻止其他接收者的接收动作。

有序广播

有序广播比较特殊，它每次只发送到优先级较高的接收者那里，然后由优先级高的接受者再传播到优先级低的接收者那里，优先级高的接收者有能力终止这个广播
可以使用intent-filter里面的android: priority="1000"去解决

例子

例子：获取短信广播，并读取数据

```
1.  public class SMSReceiver extends BroadcastReceiver {
2.
3.      private static final String action =
    "android.provider.Telephony.SMS_RECEIVED";
4.
5.      @Override
6.      public void onReceive(Context context, Intent intent) {
7.          if (intent.getAction().equals(action)) {
8.              Object[] pduses = (Object[]) intent.getExtras().get("pdus")
    ;
9.              String mobile = "";
10.             String content = "";
11.             String time = "";
```

```
            System.out.println(pduses.length);
            for (Object pdus : pduses) {
                byte[] pdusmessage = (byte[]) pdus;
                SmsMessage sms = SmsMessage.createFromPdu(pdusmessage);
                mobile = sms.getOriginatingAddress();// 发送短信的手机号码
                content = sms.getMessageBody(); // 短信内容
                Date date = new Date(sms.getTimestampMillis());// 发送日期
                SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                time = format.format(date); // 得到发送时间
            }
            Toast.makeText(context, context.getResources().getString(R.string.message), Toast.LENGTH_LONG).show();
            String show = "发送者:" + mobile + "   内容:" + content + "时间:" + time;
            Toast.makeText(context, show, Toast.LENGTH_LONG).show();
        }
    }
}
```

例子：读电量信息

如果我们阅读软件，可能是全屏阅读，这个时候用户就看不到剩余的电量，我们就可以为他们提供电量的信息。要想做到这一点，我们需要接收一条电量变化的广播，然后获取百分比信息

```
public class BatteryChangedReceiver extends BroadcastReceiver {
    private int BatteryN; // 目前电量
    private int BatteryV; // 电池电压
    private double BatteryT; // 电池温度
    private String BatteryStatus; // 电池状态
    private String BatteryTemp; // 电池使用情况
    private TextView tv;

    public BatteryChangedReceiver(TextView tv) {
        this.tv = tv;
    }

    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        /*
         * 如果捕捉到的action是ACTION_BATTERY_CHANGED,  就运行onBatteryInfoReceiver()
         */
```

```
18.          if (Intent.ACTION_BATTERY_CHANGED.equals(action)) {
19.              BatteryN = intent.getIntExtra("level", 0); // 目前电量
20.              BatteryV = intent.getIntExtra("voltage", 0); // 电池电压
21.              BatteryT = intent.getIntExtra("temperature", 0); // 电池温度
22.
23.              switch (intent.getIntExtra("status",
    BatteryManager.BATTERY_STATUS_UNKNOWN)) {
24.              case BatteryManager.BATTERY_STATUS_CHARGING:
25.                  BatteryStatus = "充电状态";
26.                  break;
27.              case BatteryManager.BATTERY_STATUS_DISCHARGING:
28.                  BatteryStatus = "放电状态";
29.                  break;
30.              case BatteryManager.BATTERY_STATUS_NOT_CHARGING:
31.                  BatteryStatus = "未充电";
32.                  break;
33.              case BatteryManager.BATTERY_STATUS_FULL:
34.                  BatteryStatus = "充满电";
35.                  break;
36.              case BatteryManager.BATTERY_STATUS_UNKNOWN:
37.                  BatteryStatus = "未知道状态";
38.                  break;
39.              }
40.
41.              switch (intent.getIntExtra("health",
    BatteryManager.BATTERY_HEALTH_UNKNOWN)) {
42.              case BatteryManager.BATTERY_HEALTH_UNKNOWN:
43.                  BatteryTemp = "未知错误";
44.                  break;
45.              case BatteryManager.BATTERY_HEALTH_GOOD:
46.                  BatteryTemp = "状态良好";
47.                  break;
48.              case BatteryManager.BATTERY_HEALTH_DEAD:
49.                  BatteryTemp = "电池没有电";
50.                  break;
51.              case BatteryManager.BATTERY_HEALTH_OVER_VOLTAGE:
52.                  BatteryTemp = "电池电压过高";
53.                  break;
54.              case BatteryManager.BATTERY_HEALTH_OVERHEAT:
55.                  BatteryTemp = "电池过热";
56.                  break;
57.              }
58.              tv.setText("目前电量为" + BatteryN + "% --- " + BatteryStatus
    + "\n" + "电压为" + BatteryV + "mV --- "
59.                      + BatteryTemp + "\n" + "温度为" + (BatteryT * 0.1) +
```

```
"℃");
60.             }
61.         }
```

例子：监控网络状态

在某些场合，比如用户浏览网络信息时，网络突然断开，我们要及时地提醒用户网络已断开。要实现这个功能，我们可以接收网络状态改变这样一条广播，当由连接状态变为断开状态时，系统就会发送一条广播，我们接收到之后，再通过网络的状态做出相应的操作

```
1.  public class NetworkStateReceiver extends BroadcastReceiver {
2.      private static final String TAG = "NetworkStateReceiver";
3.      @Override
4.      public void onReceive(Context context, Intent intent) {
5.          Log.i(TAG, "network state changed.");
6.          if (!isNetworkAvailable(context)) {
7.              Toast.makeText(context, "network disconnected!", 0).show();
8.          }
9.      }
10.
11.     /**
12.      * 网络是否可用
13.      *
14.      * @param context
15.      * @return
16.      */
17.     public static boolean isNetworkAvailable(Context context) {
18.         ConnectivityManager mgr = (ConnectivityManager)context.getSyste
    mService(Context.CONNECTIVITY_SERVICE);
19.         NetworkInfo[] info = mgr.getAllNetworkInfo();
20.         if (info != null) {
21.             for (int i = 0; i < info.length; i++) {
22.                 if (info[i].getState() == NetworkInfo.State.CONNECTED)
    {
23.                     return true;
24.                 }
25.             }
26.         }
27.         return false;
28.     }
29. }
```

注册一下这个接收者的信息

```
<receiver android:name="com.hua.bcreceiver.NetworkStateReceiver" >
            <intent-filter>
                <action
android:name="android.net.conn.CONNECTIVITY_CHANGE" />

                <category
android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </receiver>
```

因为在`isNetworkAvailable`方法中我们使用到了网络状态相关的API，所以需要声明相关的权限才行

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

AlarmManager类

AlarmManager概述

AlarmManager的使用机制有的称呼为全局定时器，有的称呼为闹钟。通过对它的使用，个人觉得叫全局定时器比较合适，其实它的作用和Timer有点相似。都有两种相似的用法：（1）在指定时长后执行某项操作（2）周期性的执行某项操作
AlarmManager对象配合Intent使用，可以定时的开启一个Activity,发送一个BroadCast,或者开启一个Service。

公有方法

```
oid cancel(PendingIntent operation)
取消AlarmManager的定时服务。

void set(int type, long triggerAtTime, PendingIntent operation)
设置在triggerAtTime时间启动由operation参数指定的组件。（该方法用于设置一次性闹钟）

void setInexactRepeating(int type, long triggerAtTime, long interval,
PendingIntent operation)
```

```
设置一个非精确的周期性任务。

void setRepeating(int type, long triggerAtTime, long interval,
PendingIntent operation)
设置一个周期性执行的定时服务。

void setTime(long millis)
设置系统“墙”时钟。需要android.permission.SET_TIME.权限。

void setTimeZone(String timeZone)
设置系统的默认时区。需要android.permission.SET_TIME_ZONE.权限。
```

常用方法说明：

AlarmManager的常用方法有三个：

```
1,set(int type,long startTime,PendingIntent pi)//该方法用于设置一次性闹钟。
第一个参数int type指定定时服务的类型，该参数接受值见下`Type类型表`。

第二个参数表示闹钟执行时间。

第三个参数PendingIntent pi表示闹钟响应动作

2,setRepeating(int type,long startTime,long intervalTime,PendingIntent
pi)
设置一个周期性执行的定时服务。第一个参数表示闹钟类型，第二个参数表示闹钟首次执行时间
，第三个参数表示闹钟两次执行的间隔时间，第三个参数表示闹钟响应动作

3,setInexactRepeating(int type,long startTime,long
intervalTime,PendingIntent pi)
该方法也用于设置重复闹钟，与第二个方法相似，不过其两个闹钟执行的间隔时间不是固定的而
已。它相对而言更省电（power-efficient）一些，因为系统可能会将几个差不多的闹钟合并
为一个来执行，减少设备的唤醒次数。第三个参数intervalTime为闹钟间隔，内置的几个变量
见`intervalTime常量表`,第四个参数表示动作。

```

Type类型

```
    ELAPSED_REALTIME： 在指定的延时过后，发送广播，但不唤醒设备（闹钟在睡眠状态
下不可用）。如果在系统休眠时闹钟触发，它将不会被传递，直到下一次设备唤醒。
    ELAPSED_REALTIME_WAKEUP： 在指定的延时过后，发送广播，并唤醒设备（即使关机
也会执行operation所对应的组件）。延时是要把系统启动的时
```

```
间SystemClock.elapsedRealtime()算进去的，具体用法看代码。
    RTC： 指定当系统调用System.currentTimeMillis()方法返回的值与triggerAtTime相等时启动operation所对应的设备（在指定的时刻，发送广播，但不唤醒设备）。如果在系统休眠时闹钟触发，它将不会被传递，直到下一次设备唤醒（闹钟在睡眠状态下不可用）。
    RTC_WAKEUP： 指定当系统调用System.currentTimeMillis()方法返回的值与triggerAtTime相等时启动operation所对应的设备（在指定的时刻，发送广播，并唤醒设备）。即使系统关机也会执行operation所对应的组件。
```

intervalTime常量

```
INTERVAL_DAY：                设置闹钟，间隔一天
INTERVAL_HALF_DAY：           设置闹钟，间隔半天
INTERVAL_FIFTEEN_MINUTES：    设置闹钟，间隔15分钟
INTERVAL_HALF_HOUR：          设置闹钟，间隔半个小时
INTERVAL_HOUR：               设置闹钟，间隔一个小时
```

学员作业

尝试做一个应用，将每一个拨入和拨出电话记录到数据库，并记录日期时间，并可以随时查阅。

系统广播

```
//关闭或打开飞行模式时的广播
Intent.ACTION_AIRPLANE_M;

//充电状态，或者电池的电量发生变化;//电池的充电状态、电荷级别改变，不能通过组建声;
Intent.ACTION_BATTERY_CH;

//表示电池电量低
Intent.ACTION_BATTERY_LO;

//表示电池电量充足
Intent.ACTION_BATTERY_OK;

//关闭或打开飞行模式时的广播
Intent.ACTION_AIRPLANE_MODE_CHANGED;

```

```
16.  //充电状态，或者电池的电量发生变化//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
17.  Intent.ACTION_BATTERY_CHANGED;
18.
19.  //表示电池电量低
20.  Intent.ACTION_BATTERY_LOW;
21.
22.  //表示电池电量充足，即从电池电量低变化到饱满时会发出广播
23.  Intent.ACTION_BATTERY_OKAY;
24.
25.  //在系统启动完成后，这个动作被广播一次（只有一次）。
26.  Intent.ACTION_BOOT_COMPLETED;
27.
28.  //按下照相时的拍照按键(硬件按键)时发出的广播
29.  Intent.ACTION_CAMERA_BUTTON;
30.
31.  //当屏幕超时进行锁屏时,当用户按下电源按钮,长按或短按(不管有没跳出话框)，进行锁屏时，
     android系统都会广播此Action消息
32.  Intent.ACTION_CLOSE_SYSTEM_DIALOGS;
33.
34.  //设备当前设置被改变时发出的广播(包括的改变:界面语言，设备方向，等，请参考Configur
     ation.java)
35.  Intent.ACTION_CONFIGURATION_CHANGED;
36.
37.  //设备日期发生改变时会发出此广播
38.  Intent.ACTION_DATE_CHANGED;
39.
40.  //设备内存不足时发出的广播,此广播只能由系统使用，其它APP不可用
41.  Intent.ACTION_DEVICE_STORAGE_LOW;
42.
43.  //设备内存从不足到充足时发出的广播,此广播只能由系统使用，其它APP不可用
44.  Intent.ACTION_DEVICE_STORAGE_OK;
45.
46.  //发出此广播的地方
     frameworks\base\services\java\com\android\server\DockObserver.java
47.  Intent.ACTION_DOCK_EVENT;
48.
49.  //移动APP完成之后，发出的广播(移动是指:APP2SD)
50.  Intent.ACTION_EXTERNAL_APPLICATIONS_AVAILABLE;
51.
52.  //正在移动APP时，发出的广播(移动是指:APP2SD)
53.  Intent.ACTION_EXTERNAL_APPLICATIONS_UNAVAILABLE;
54.
55.  //Gtalk已建立连接时发出的广播
56.  Intent.ACTION_GTALK_SERVICE_CONNECTED;
```

```
57.
58.   //Gtalk已断开连接时发出的广播
59.   Intent.ACTION_GTALK_SERVICE_DISCONNECTED;
60.
61.   //在耳机口上插入耳机时发出的广播
62.   Intent.ACTION_HEADSET_PLUG;
63.
64.   //改变输入法时发出的广播
65.   Intent.ACTION_INPUT_METHOD_CHANGED;
66.
67.   //设备当前区域设置已更改时发出的广播
68.   Intent.ACTION_LOCALE_CHANGED;
69.
70.   //表示用户和包管理所承认的低内存状态通知应该开始。
71.   Intent.ACTION_MANAGE_PACKAGE_STORAGE;
72.
73.   //未正确移除SD卡(正确移除SD卡的方法:设置--SD卡和设备内存--卸载SD卡)，但已把SD卡取
      出来时发出的广播 ,扩展介质（扩展卡）已经从 SD 卡插槽拔出，但是挂载点 (mount
      point) 还没解除 (unmount)
74.   Intent.ACTION_MEDIA_BAD_REMOVAL;
75.
76.   //按下"Media Button" 按键时发出的广播,假如有"Media Button" 按键的话(硬件按键)
77.   Intent.ACTION_MEDIA_BUTTON;
78.
79.   //插入外部储存装置，比如SD卡时，系统会检验SD卡，此时发出的广播?
80.   Intent.ACTION_MEDIA_CHECKING;
81.
82.   //已拔掉外部大容量储存设备发出的广播（比如SD卡，或移动硬盘）,不管有没有正确卸载都会
      发出此广播， 用户想要移除扩展介质（拔掉扩展卡）。
83.   Intent.ACTION_MEDIA_EJECT;
84.
85.   //插入SD卡并且已正确安装（识别）时发出的广播， 扩展介质被插入，而且已经被挂载。
86.   Intent.ACTION_MEDIA_MOUNTED;
87.
88.   //拓展介质存在，但使用不兼容FS（或为空）的路径安装点检查介质包含在Intent.mData领
      域。
89.   Intent.ACTION_MEDIA_NOFS;
90.
91.   //外部储存设备已被移除，不管有没正确卸载,都会发出此广播， 扩展介质被移除。
92.   Intent.ACTION_MEDIA_REMOVED;
93.
94.   //广播：已经扫描完介质的一个目录
95.   Intent.ACTION_MEDIA_SCANNER_FINISHED;
96.
97.   //请求媒体扫描仪扫描文件并将其添加到媒体数据库。
```

```
Intent.ACTION_MEDIA_SCANNER_SCAN_FILE;

//广播：开始扫描介质的一个目录
Intent.ACTION_MEDIA_SCANNER_STARTED;

// 广播：扩展介质的挂载被解除 (unmount)，因为它已经作为 USB 大容量存储被共享。
Intent.ACTION_MEDIA_SHARED;

Intent.ACTION_MEDIA_UNMOUNTABLE;//

// 广播：扩展介质存在，但是还没有被挂载 (mount)
Intent.ACTION_MEDIA_UNMOUNTED

Intent.ACTION_NEW_OUTGOING_CALL;

//成功的安装APK之后//广播：设备上新安装了一个应用程序包。//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）
Intent.ACTION_PACKAGE_ADDED;

//一个已存在的应用程序包已经改变，包括包名
Intent.ACTION_PACKAGE_CHANGED;

//清除一个应用程序的数据时发出的广播(在设置 - -应用管理 - -选中某个应用，之后点清除数据时?)//用户已经清除一个包的数据，包括包名（清除包程序不能接收到这个广播）
Intent.ACTION_PACKAGE_DATA_CLEARED;

//触发一个下载并且完成安装时发出的广播，比如在电子市场里下载应用？
Intent.ACTION_PACKAGE_INSTALL;

//成功的删除某个APK之后发出的广播，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）
Intent.ACTION_PACKAGE_REMOVED;

//替换一个现有的安装包时发出的广播（不管现在安装的APP比之前的新还是旧，都会发出此广播？）
Intent.ACTION_PACKAGE_REPLACED;

//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）
Intent.ACTION_PACKAGE_RESTARTED;

//插上外部电源时发出的广播
Intent.ACTION_POWER_CONNECTED;

//已断开外部电源连接时发出的广播
```

```
Intent.ACTION_POWER_DISCONNECTED;

Intent.ACTION_PROVIDER_CHANGED;//

//重启设备时的广播
Intent.ACTION_REBOOT;

//屏幕被关闭之后的广播
Intent.ACTION_SCREEN_OFF;

//屏幕被打开之后的广播
Intent.ACTION_SCREEN_ON;

//关闭系统时发出的广播
Intent.ACTION_SHUTDOWN;

//时区发生改变时发出的广播
Intent.ACTION_TIMEZONE_CHANGED;

//时间被设置时发出的广播
Intent.ACTION_TIME_CHANGED;

//广播：当前时间已经变化（正常的时间流逝）， 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
Intent.ACTION_TIME_TICK;

//一个用户ID已经从系统中移除发出的广播
Intent.ACTION_UID_REMOVED;

//设备已进入USB大容量储存状态时发出的广播？
Intent.ACTION_UMS_CONNECTED;

//设备已从USB大容量储存状态转为正常状态时发出的广播？
Intent.ACTION_UMS_DISCONNECTED;

Intent.ACTION_USER_PRESENT;//

//设备墙纸已改变时发出的广播
Intent.ACTION_WALLPAPER_CHANGED;
```