第 30 章 broadcast Intent

Android设备中，各种事件一直频繁地发生着。WIFI信号时有时无，各种软件包获得安装，电话不时呼入，短信频繁接收。

许多系统组件都需知道某些事件的发生。为满足这样的需求，Android提供了broadcast intent组件。broadcast intent的工作原理类似之前学过的intent，唯一不同的是broadcast intent可同时被多个组件接收，如图30-1所示。broadcast receiver负责接收各种broadcast intent。

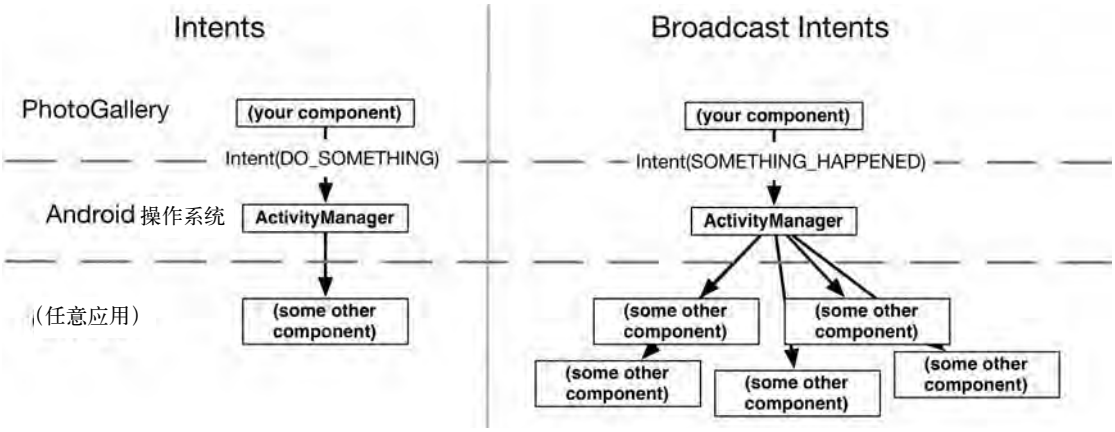

图30-1　普通intent与broadcast intent

本章，我们将学习如何监听系统发送的broadcast intent，以及如何在运行的应用中动态的发送与接收它们。首先，我们来监听一个宣告设备已启动完毕的broadcast，然后学习发送和接收我们自己的broadcast intent。

30.1　随设备重启而重启的定时器

PhotoGallery应用的后台定时器虽然可以正常工作，但还不够完美。如果用户重启了手机设备，定时器就会停止运行。

设备重启后，那些持续运行的应用通常也需要重启。通过监听具有BOOT_COMPLETED操作的broadcast intent，可得知设备是否已完成启动。

30.1.1　配置文件中的 broadcast receiver

现在，我们来编写一个broadcast receiver。首先以android.content.BroadcastReceiver为父类，创建一个StartupReceiver新类。Eclipse会自动实现一个onReceive(Context,Intent)抽象方法。输入代码清单30-1所示代码完成类的创建。

代码清单30-1　第一个broadcast receiver（StartupReceiver.java）

```
package com.bignerdranch.android.photogallery;

...

public class StartupReceiver extends BroadcastReceiver {
    private static final String TAG = "StartupReceiver";

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.i(TAG, "Received broadcast intent: " + intent.getAction());
    }
}
```

这就是我们创建的broadcast receiver。与服务和activity一样，broadcast receiver是接收intent的组件，必须在系统中登记后才能使用。说到登记，不要总以为是指在配置文件中进行登记。不过，当前broadcast receiver的确是在manifest配置文件中登记的。

登记关联receiver类似前面对服务或activity的处理。我们使用receiver标签并包含相应的intent-filter在其中。StartupReceiver将会监听BOOT_COMPLETED操作。而该操作也需要配置使用权限。因此，我们还需要添加一个相应的uses-permission标签。

打开AndroidManifest.xml配置文件，参照代码清单30-2关联登记StartupReceiver。

代码清单30-2　在manifest文件中添加receiver（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.photogallery"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />

  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
  <application
    ... >
    <activity
      ... >
      ...
    </activity>
    <service android:name=".PollService" />
    <receiver android:name=".StartupReceiver">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
```

```
        </intent-filter>
      </receiver>
    </application>

</manifest>
```

与activity和服务不同，在配置文件中声明的broadcast receiver几乎总是需要声明intent filter。broadcast intent就是为发送信息给多个监听者而生的，但显式intent只有一个receiver。因此显式broadcast intent很少见。

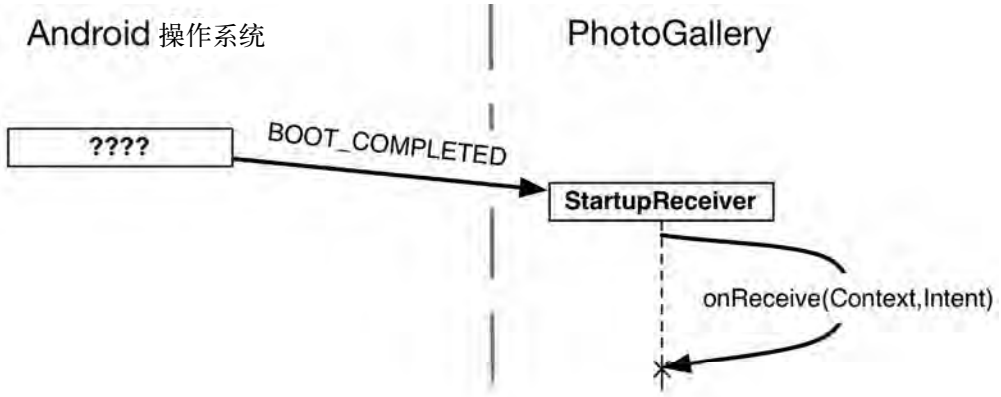

图30-2 接收BOOT_COMPLETED

在配置文件中完成声明后，即使应用当前并未运行，只要有匹配的broadcast intent的发来，broadcast receiver就会接收。一收到intent，broadcast receiver的onReceive(Context,Intent)方法即开始运行，然后broadcast receiver就会被销毁。

运行PhotoGallery应用。重启设备或模拟器并切换到DDMS视图。可在LogCat中看到表明receiver运行的日志。但如果在设备标签页查看设备，则可能看不到任何PhotoGallery的进程。这是因为进程在运行broadcast receiver后，随即就消亡了。

30.1.2 如何使用 receiver

broadcast receiver的存在如此短暂，因此它的作用有限。例如，我们无法使用任何异步API或登记任何监听器，因为onReceive(Context,Intent)方法刚运行完，receiver就不存在了。onReceive(Context,Intent)方法同样运行在主线程上，因此不能在该方法内做一些耗时的重度任务，如网络连接或数据的永久存储等。

然而，这并不代表receiver一无用处。对于轻型任务代码的运行而言，receiver非常有用。系统重启后，定时运行的定时器也需进行重置。显然，使用broadcast receiver处理这种小任务再合适不过了。

receiver需要知道定时器的启停状态。在PollService类中添加一个preference常量，用于存储状态信息，如代码清单30-3所示。

代码清单30-3　添加定时器状态preference（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    private static final int POLL_INTERVAL = 1000 * 60 * 5; // 5 minutes
    public static final String PREF_IS_ALARM_ON = "isAlarmOn";

    public PollService() {
        super(TAG);
    }

    ...

    public static void setServiceAlarm(Context context, boolean isOn) {
        Intent i = new Intent(context, PollService.class);
        PendingIntent pi = PendingIntent.getService(
                context, 0, i, 0);

        AlarmManager alarmManager = (AlarmManager)
                context.getSystemService(Context.ALARM_SERVICE);

        if (isOn) {
            alarmManager.setRepeating(AlarmManager.RTC,
                    System.currentTimeMillis(), POLL_INTERVAL, pi);
        } else {
            alarmManager.cancel(pi);
            pi.cancel();
        }

        PreferenceManager.getDefaultSharedPreferences(context)
            .edit()
            .putBoolean(PollService.PREF_IS_ALARM_ON, isOn)
            .commit();
    }
```

然后，设备重启后，StartupReceiver可使用它打开定时器，如代码清单30-4所示。

代码清单30-4　设备重启后启动定时器（StartupReceiver.java）

```
@Override
public void onReceive(Context context, Intent intent) {
    Log.i(TAG, "Received broadcast intent: " + intent.getAction());

    SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);
    boolean isOn = prefs.getBoolean(PollService.PREF_IS_ALARM_ON, false);
    PollService.setServiceAlarm(context, isOn);
}
```

再次运行PhotoGallery应用。这次，设备重启后，后台结果检查服务也应该得到了重启。

30.2　过滤前台通知消息

解决了设备重启问题后，再来看看PhotoGallery应用的另一缺陷。应用的通知消息虽然工作良好，但在打开应用后，我们依然会收到通知消息。

同样，我们可以利用broadcast intent来解决这个问题。但它将会以一种完全不同的方式进行工作。

30.2.1 发送 broadcast intent

首先来处理问题修正方案中最容易的部分：发送自己的broadcast intent。要发送broadcast intent，只需创建一个intent，并传入sendBroadcast(Intent)方法即可。这里，需要通过sendBroadcast(Intent)方法广播我们定义的操作（action），因此还需要定义一个操作常量。在PollService类中，完成清单30-5所示代码的输入。

代码清单30-5 发送broadcast intent（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    private static final int POLL_INTERVAL = 1000 * 60 * 5; // 5 minutes
    public static final String PREF_IS_ALARM_ON = "isAlarmOn";

    public static final String ACTION_SHOW_NOTIFICATION =
        "com.bignerdranch.android.photogallery.SHOW_NOTIFICATION";

    public PollService() {
        super(TAG);
    }

    @Override
    public void onHandleIntent(Intent intent) {
        ...

        if (!resultId.equals(lastResultId)) {
            ...

            NotificationManager notificationManager = (NotificationManager)
                getSystemService(NOTIFICATION_SERVICE);

            notificationManager.notify(0, notification);

            sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION));
        }

        prefs.edit()
            .putString(FlickrFetchr.PREF_LAST_RESULT_ID, resultId)
            .commit();
    }
```

30.2.2 动态 broadcast receiver

完成intent的发送后，接下来的任务是接收broadcast intent。可以编写一个类似StartupReceiver，并在配置文件中登记的broadcast receiver来接收intent。但这里该方法行不通。我们需要在PhotoGalleryFragment存在的时候接收intent。而在配置文件中声明的独立receiver则很难做到这一点。因为该receiver在不断接收intent的同时，还需要另一种方法来知晓PhotoGalleryFragment的存在状态。

使用动态broadcast receiver可解决该问题。动态broadcast receiver是在代码中，而不是在配置文件中完成登记声明的。要在代码中登记receiver，可调用`registerReceiver(BroadcastReceiver, IntentFilter)`方法;取消登记时,则调用`unregisterReceiver(BroadcastReceiver)`方法。如同一个按钮点击监听器，receiver本身通常被定义为一个内部类实例。不过，在`registerReceiver(...)`和`unregisterReceiver(...)`方法中，我们需要同一个实例，因此需要将receiver赋值给一个实例变量。

以`Fragment`为超类，新建一个`VisibleFragment`抽象类，如代码清单30-6所示。该类是一个隐藏前台通知的通用fragment。（第31章，我们将学习编写一个类似的fragment。）

代码清单30-6　`VisibleFragment`自己的receiver（VisibleFragment.java）

```
package com.bignerdranch.android.photogallery;

...

public abstract class VisibleFragment extends Fragment {
    public static final String TAG = "VisibleFragment";

    private BroadcastReceiver mOnShowNotification = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(getActivity(),
                    "Got a broadcast:" + intent.getAction(),
                    Toast.LENGTH_LONG)
                .show();
        }
    };

    @Override
    public void onResume() {
        super.onResume();
        IntentFilter filter = new IntentFilter(PollService.ACTION_SHOW_NOTIFICATION);
        getActivity().registerReceiver(mOnShowNotification, filter);
    }

    @Override
    public void onPause() {
        super.onPause();
        getActivity().unregisterReceiver(mOnShowNotification);
    }
}
```

注意，要传入一个`IntentFilter`，必须先以代码的方式创建它。这里创建的`IntentFilter`同以下在XML文件定义的filter是一样的：

```
<intent-filter>
    <action android:name="com.bignerdranch.android.photogallery.SHOW_NOTIFICATION" />
</intent-filter>
```

任何使用XML定义的`IntentFilter`，均可以代码的方式完成定义。要配置以代码方式创建的`IntentFilter`，直接调用`addCategory(String)`、`addAction(String)`和`addDataPath(String)`等方法。

使用完后，动态登记的broadcast receiver必须能够自我清除。通常，如果在启动生命周期方法中登记了receiver，则需在相应的停止方法中调用Context.unregisterReceiver(BroadcastReceiver)方法。因此，这里，我们在onResume()方法里登记，而在onPause()方法里撤销登记。同样地，如在onActivityCreated(...)方法里登记，则应在onActivityDestroyed()里撤销登记。

（顺便要说的是，我们应注意保留fragment中的onCreate(...)和onDestroy()方法的运用。设备发生旋转时，onCreate(...)和onDestroy()方法中的getActivity()方法会返回不同的值。因此，如想在Fragment.onCreate(Bundle)和Fragment.onDestroy()方法中实现登记或撤销登记，应使用getActivity().getApplicationContext()方法。）

修改PhotoGalleryFragment，调整其父类为VisibleFragment，如代码清单30-7所示。

代码清单30-7 设置fragment为可见（PhotoGalleryFragment.java）

```
~~public class PhotoGalleryFragment extends Fragment {~~
public class PhotoGalleryFragment extends VisibleFragment {
    GridView mGridView;
    ArrayList<GalleryItem> mItems;
    ThumbnailDownloader<ImageView> mThumbnailThread;
```

运行PhotoGallery应用。多次开关后台结果检查服务，可看到toast提示消息以及顶部状态栏显示的通知信息，如图30-3所示。

图30-3 验证broadcast的存在

30.2.3　使用私有权限

使用动态broadcast receiver存在一个问题，即系统中的任何应用均可监听并触发我们的receiver。通常情况下，我们肯定不希望发生这样的事情。

不要担心，有多种方式可用于阻止未授权的应用闯入我们的私人领域。如果receiver声明在manifest配置文件里，且仅限应用内部使用，则可在receiver标签上添加一个`android:exported="false"`属性。这样，系统中的其他应用就再也无法接触到该receiver。另外，也可创建自己的使用权限。这通常通过在AndroidManifest.xml中添加一个`permission`标签来完成。

在AndroidManifest.xml配置文件中，添加代码清单30-8所示代码，声明并获取属于自己的使用权限。

代码清单30-8　添加私有权限（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.photogallery"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />

  <permission android:name="com.bignerdranch.android.photogallery.PRIVATE"
    android:protectionLevel="signature" />

  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
  <uses-permission android:name="com.bignerdranch.android.photogallery.PRIVATE" />

  <application
    ... >
    ...
  </application>

</manifest>
```

以上代码中，使用protection level签名，我们定义了自己的定制权限。稍后，我们将学习到更多有关保护级别的内容。如同前面用过的intent操作、类别以及系统权限，权限本身只是一行简单的字符串。即使是自定义的权限，也必须在使用前获取它，这是规则。

注意代码中的加灰常量，这样的字符串需要在三个地方出现，并且必须保证完全一致。因此，最好使用复制粘贴功能，而不是手动输入。

接下来，为使用权限，在代码中定义一个对应常量，然后将其传入`sendBroadcast(...)`方法，如代码清单30-9所示。

代码清单30-9　发送带有权限的broadcast（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";
```

```
    private static final int POLL_INTERVAL = 1000 * 60 * 5; // 5 minutes
    public static final String PREF_IS_ALARM_ON = "isAlarmOn";

    public static final String ACTION_SHOW_NOTIFICATION =
        "com.bignerdranch.android.photogallery.SHOW_NOTIFICATION";

    public static final String PERM_PRIVATE =
        "com.bignerdranch.android.photogallery.PRIVATE";

    public PollService() {
        super(TAG);
    }

    @Override
    public void onHandleIntent(Intent intent) {
        ...

        if (!resultId.equals(lastResultId)) {
            ...

            NotificationManager notificationManager = (NotificationManager)
                getSystemService(NOTIFICATION_SERVICE);

            notificationManager.notify(0, notification);

            sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION));
            sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION), PERM_PRIVATE);
        }

        prefs.edit()
            .putString(FlickrFetchr.PREF_LAST_RESULT_ID, resultId)
            .commit();
    }
```

要使用权限，须将其作为参数传入sendBroadcast(...)方法。这里，调用sendBroadcast(...)方法时指定了接收权限，任何应用必须使用同样的权限才能接收我们发送的intent。

要怎么保护我们的broadcast receiver呢？其他应用可通过创建自己的broadcast intent来触发broadcast receiver。同样，在registerReceiver(...)方法中传入自定义权限即可解决该问题，如代码清单30-10所示。

代码清单30-10 broadcast receiver的使用权限（VisibleFragment.java）

```
    @Override
    public void onResume() {
        super.onResume();
        IntentFilter filter = new IntentFilter(PollService.ACTION_SHOW_NOTIFICATION);
        getActivity().registerReceiver(mOnShowNotification, filter);
        getActivity().registerReceiver(mOnShowNotification, filter,
            PollService.PERM_PRIVATE, null);
    }
```

现在，只有我们的应用才能够触发目标receiver。

深入学习protection level

自定义权限必须指定android:protectionLevel属性值。Android根据protectionLevel属性值确定自定义权限的使用方式。PhotoGallery应用中，我们使用的是signature protectionLevel。

signature安全级别表明，如果其他应用需要使用我们的自定义权限，则必须使用和当前应用相同的key做签名认证。对于仅限应用内部使用的权限，我们通常会选择signature安全级别。既然其他开发者并没有相同的key值，自然也就无法接触到权限保护的东西。此外，有了自己的key，将来还可用于我们开发的其他应用中。protectionLevel的可选值如表30-1所示。

表30-1　protectionLevel的可选值

可　选　值	用法描述
normal	用于阻止应用执行危险操作，如访问个人隐私数据、联网传送数据等。应用安装前，用户可看到相应的安全级别，但无需他们主动授权。android.permission.RECEIVE_BOOT_COMPLETED使用该安全级别。同样，手机振动也使用该安全级别。虽然这些安全级别没有危险，但最好让用户知晓可能带来的影响
dangerous	normal安全级别控制以外的任何危险操作，如访问个人隐私数据、通过网络接口收发数据、使用可监视用户的硬件功能等。总之，包括一切可能为用户带来麻烦的行为。网络使用权限、相机使用权限以及联系人信息使用权限都属于危险操作。需要dangerous权限级别时，Android会明确要求用户授权
signature	如果应用签署了与声明应用一致的权限证书，则该权限由系统授予。否则，系统则作相应的拒绝。权限授予时，系统不会通知用户。它通常适用于应用内部。只要拥有证书，则只有签署了同样证书的应用才能拥有该权限，因此可自由控制权限的使用。这里，我们使用它阻止其他应用监听到应用发出的broadcast。不过如有需要，可定制开发能够监听它们的专有应用
signatureOrSystem	类似于signature授权级别。但该授权级别针对Android系统镜像中的所有包授权。该授权级别用于系统镜像内应用间的通信，因此用户通常无需关心

30.2.4　使用 ordered broadcast 接收结果

最后来接收broadcast intent。虽然已经发送了个人私有的broadcast，但目前还只是只发不收的单向通信，如图30-4所示

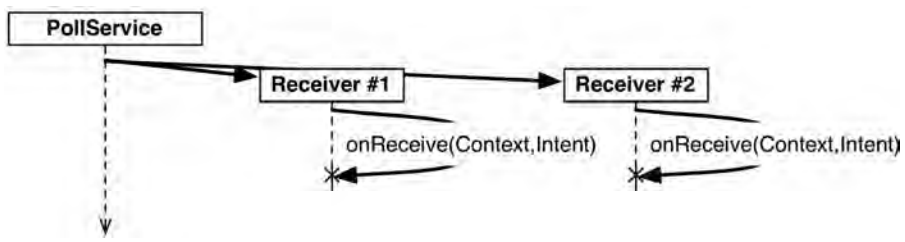

图30-4　常规broadcast intent

这是因为，从概念上讲，常规broadcast intent可同时被其他应用所接收。而现在，onReceive(...)方法是在主线程上调用的，所以实际上，receiver并没有同步并发运行。因而，指望它们会按照某种顺序依次运行，或知道它们什么时候全部结束运行也是不可能的。结果，这给broadcast receiver之间的通信，或intent发送者接收receiver的信息都带来了麻烦。

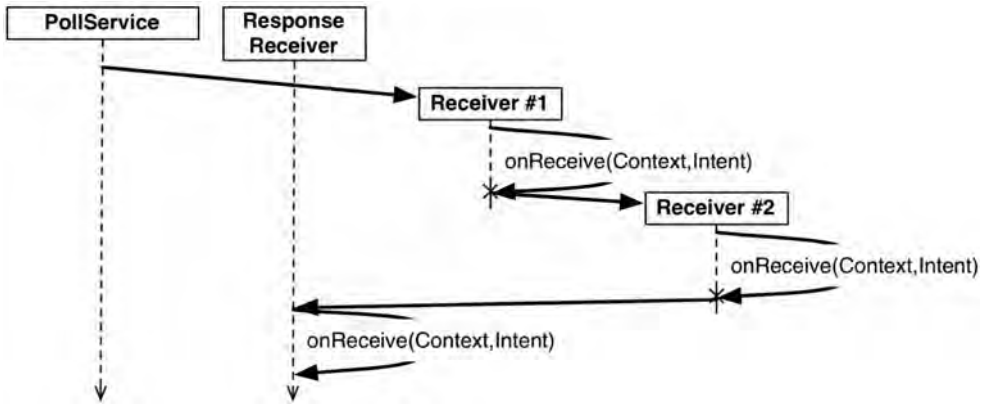

图30-5 有序broadcast intent

为解决问题，可使用有序broadcast intent实现双向通信。有序broadcast允许多个broadcast receiver依序处理broadcast intent。另外，通过传入一个名为result receiver的特别broadcast receiver，有序broadcast还可实现让broadcast的发送者接收broadcast接收者发送的返回结果。

从接收方来看，这看上去与常规broadcast没什么不同。然而，这里我们获得了一个特别工具：一套改变接收者返回值的方法。这里我们需要取消通知信息。可通过一个简单的整数结果码，将此需要告知信息发送者。使用`setResultCode(int)`方法，设置结果码为`Activity.RESULT_CANCELED`。

修改`VisibleFragment`类，将取消通知的信息发送给`SHOW_NOTIFICATION`的发送者，如代码清单30-11所示。

代码清单30-11 返回一个简单结果码（VisibleFragment.java）

```
private BroadcastReceiver mOnShowNotification = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(getActivity(),
                "Got a broadcast:" + intent.getAction(),
                Toast.LENGTH_LONG)
            .show();
        // If we receive this, we're visible, so cancel
        // the notification
        Log.i(TAG, "canceling notification");
        setResultCode(Activity.RESULT_CANCELED);
    }
};
```

既然此处只需发送`YES`或`NO`标志，因此使用`int`结果码即可。如需返回更多复杂数据，可使用`setResultData(String)`或`setResultExtras(Bundle)`方法。如需设置所有三个参数值，可调用`setResult(int,String,Bundle)`方法。设定返回值后，每个后续接收者均可看到或修改返回值。

为让以上方法发挥作用，broadcast必须有序。在`PollService`类中，编写一个可发送有序

broadcast的新方法，如代码清单30-12所示。该方法打包一个Notification调用，然后作为一个broadcast发出。只要通知信息还没被撤消，可指定一个result receiver发出打包的Notification。

代码清单30-12　发送有序broadcast（PollService.java）

```
void showBackgroundNotification(int requestCode, Notification notification) {
    Intent i = new Intent(ACTION_SHOW_NOTIFICATION);
    i.putExtra("REQUEST_CODE", requestCode);
    i.putExtra("NOTIFICATION", notification);

    sendOrderedBroadcast(i, PERM_PRIVATE, null, null,
        Activity.RESULT_OK, null, null);
}
```

除了在sendBroadcast(Intent,String)方法中使用的参数外，Context.sendOrderedBroadcast(Intent,String,BroadcastReceiver,Handler,int,String,Bundle)方法还有另外五个参数，依次为：一个result receiver、一个支持result receiver运行的Handler、结果代码初始值、结果数据以及有序broadcast的结果附加内容。

result receiver比较特殊，只有在所有有序broadcast intent的接收者结束运行后，它才开始运行。虽然有时可使用result receiver接收broadcast和发送通知对象，但此处该方法行不通。目标broadcast intent通常是在PollService对象消亡之前发出的，也就是说broadcast receiver可能也被销毁了。

因此，最终的broadcast receiver需要保持独立运行。以BroadcastReceiver为父类，新建一个NotificationReceiver类。输入代码清单30-13所示的实现代码。

代码清单30-13　实现result receiver（NotificationReceiver.java）

```
public class NotificationReceiver extends BroadcastReceiver {
    private static final String TAG = "NotificationReceiver";

    @Override
    public void onReceive(Context c, Intent i) {
        Log.i(TAG, "received result: " + getResultCode());
        if (getResultCode() != Activity.RESULT_OK)
            // A foreground activity cancelled the broadcast
            return;

        int requestCode = i.getIntExtra("REQUEST_CODE", 0);
        Notification notification = (Notification)
                i.getParcelableExtra("NOTIFICATION");

        NotificationManager notificationManager = (NotificationManager)
                c.getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(requestCode, notification);
    }
}
```

最后，登记新建的receiver。既然该receiver负责发送通知信息，并接收其他接收者返回的结果码，它的运行应该总是在最后。这就需要将receiver的优先级设置为最低。为保证receiver最后运行，设置其优先级值为–999（–1000及以下值属系统保留值）。

另外，既然NotificationReceiver仅限PhotoGallery应用内部使用，我们还需设置属性值为android:exported="false"，以保证其对外部应用不可见，如代码清单30-14所示。

代码清单30-14 登记notification receiver（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  ... >
  ...

  <application
    ... >
    ...
    <receiver android:name=".StartupReceiver">
      <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
      </intent-filter>
    </receiver>
    <receiver android:name=".NotificationReceiver"
      android:exported="false">
      <intent-filter
        android:priority="-999">
        <action
            android:name="com.bignerdranch.android.photogallery.SHOW_NOTIFICATION" />
      </intent-filter>
    </receiver>
  </application>

</manifest>
```

现在，代替NotificationManager，使用新方法发送通知信息，如代码清单30-15所示。

代码清单30-15 完成最后的代码修改（PollService.java）

```
@Override
public void onHandleIntent(Intent intent) {
    ...

    if (!resultId.equals(lastResultId)) {
        ...

        Notification notification = new NotificationCompat.Builder(this)
            ...
            .build();

        NotificationManager notificationManager = (NotificationManager)
            getSystemService(NOTIFICATION_SERVICE);

        notificationManager.notify(0, notification);

        sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION), PERM_PRIVATE);

        showBackgroundNotification(0, notification);
    }

    prefs.edit()
        .putString(FlickrFetchr.PREF_LAST_RESULT_ID, resultId)
        .commit();
}
```

运行PhotoGallery应用，多次切换后台polling状态。可以看到，通知信息不见了。为验证通知信息仍在后台运行，可再次将PollService.POLL_INTERVAL的时间间隔设置为5秒，以免等待长达5分钟的时间。

30.3　receiver 与长时运行任务

如不想受限于主线程的时间限制，并希望broadcast intent可触发一个长时运行任务，该怎么做呢？

有两种方式可以选择。

- 将任务交给服务去处理，然后再通过broadcast receiver启动服务。这也是我们推荐的方式。服务可以运行很久，直到完成需要处理的任务。同时服务可将请求放在队列中，然后依次进行处理，或按其自认为合适的方式管理全部任务请求。
- 使用BroadcastReceiver.goAsync()方法。该方法返回一个BroadcastReceiver.PendingResult对象，随后，我们可使用该对象提供结果。因此，可将PendingResult交给AsyncTask去执行长时运行的任务，然后再调用PendingResult的方法响应broadcast。

BroadcastReceiver.goAsync()方法有两处弊端。首先它不支持旧设备。其次，它不够灵活：我们仍需快速响应broadcast，并且与使用服务相比，没什么架构模式好选择。

当然，goAsync()方法并非一无是处：可通过该方法的调用，完成有序broadcast的结果设置。如果真的要使用它，应注意不要耗时过长。