第 29 章

后台服务 29

目前为止，我们的所有应用代码都离不开activity，也就是说它们都有着一个或多个可见的用户界面。

如果应用不需要用户界面，会怎样呢？如果任务既不用看也不用想，如播放音乐或在RSS feed上检查新博文的推送，会怎么样呢？对于这些场景，我们需要一个service（服务）。

本章，我们将为PhotoGallery应用添加一项新功能，允许用户在后台查询新的搜索结果。一旦有了新的搜索结果，用户即可在状态栏接收到通知消息。

29.1 创建 IntentService

首先来创建服务。本章，我们将使用IntentService。IntentService并不是Android提供的唯一服务，但却可能是最常用的。创建一个名为PollService的IntentService子类，它将是我们用于查询搜索结果的服务。

可以看到，在PollService.java中，Eclipse已自动添加了onHandleIntent(Intent)存根方法。添加一行日志记录语句，完成onHandleIntent(Intent)方法，然后添加一个日志标签和一个默认的构造方法，如代码清单29-1所示。

代码清单29-1 创建PollService（PollService.java）

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

    public PollService() {
        super(TAG);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        Log.i(TAG, "Received an intent: " + intent);
    }
}
```

这里实现的是最基本的IntentService。它能做什么呢？实际上，它有点类似于activity。IntentService也是一个context（Service是Context的子类），并能够响应intent（从onHandle-Intent(Intent)方法即可看出）。

服务的intent又称作命令（command）。每一条命令都要求服务完成某项具体的任务。根据服务的种类不同，服务执行命令的方式也不尽相同，如图29-1所示。

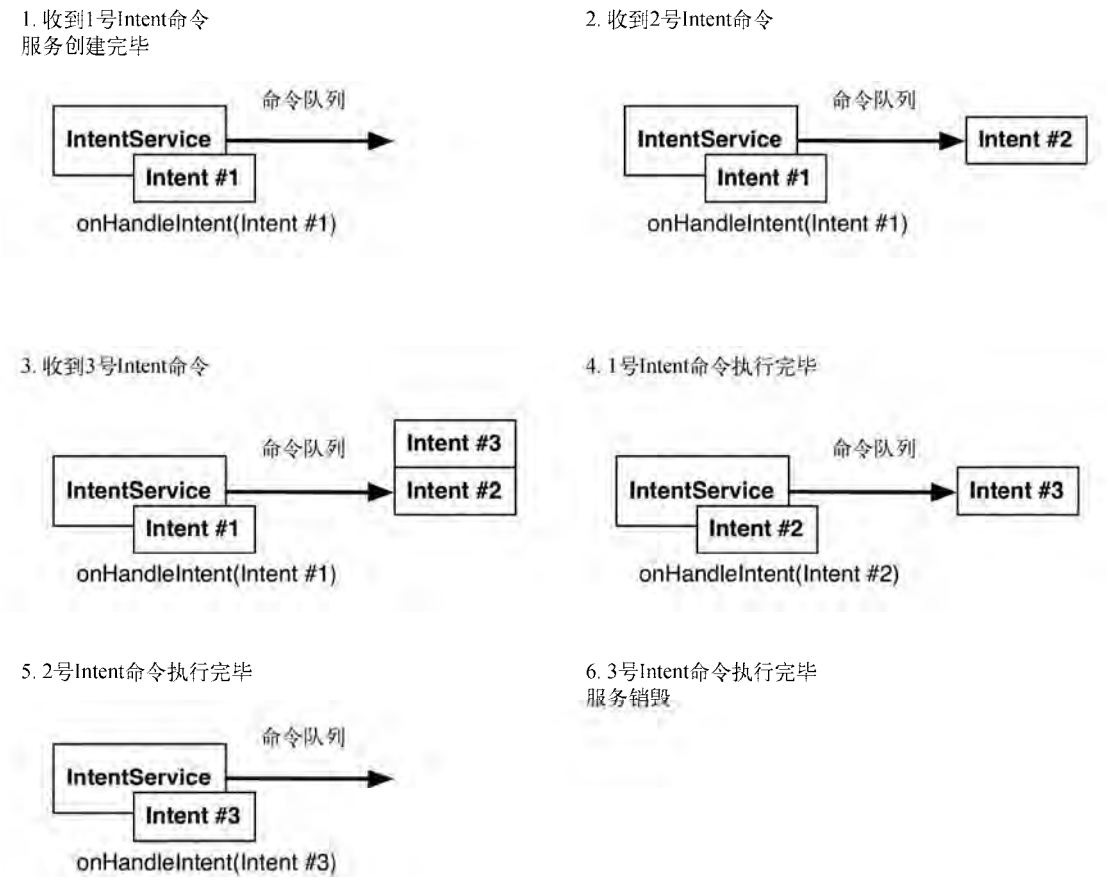

图29-1　IntentService执行命令的方式

IntentService逐个执行命令队列里的命令。接收到首个命令时，IntentService即完成启动，并触发一个后台线程，然后将命令放入队列。

随后，IntentService继续按顺序执行每一条命令，并同时为每一条命令在后台线程上调用onHandleIntent(Intent)方法。新进命令总是放置在队列尾部。最后，执行完队列中全部命令后，服务也随即停止并被销毁。

以上描述仅适用于IntentService。本章后续部分将介绍更多服务以及它们执行命令的方式。

学习了解了IntentService工作方式，大家可能已经猜到服务能够响应intent。没错！既然服务类似于activity，能够响应intent，我们就必须在AndroidManifest.xml中声明它。因此，添加一个用于PollService的元素节点定义，如代码清单29-2所示。

代码清单29-2 在manifest配置文件中添加服务（AndroidManifest.xml）

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

  ...

  <application
    ... >
    <activity
      android:name=".PhotoGalleryActivity"
      ... >
      ...
    </activity>
    <service android:name=".PollService" />
  </application>

</manifest>
```

然后，在PhotoGalleryFragment类中，添加启动服务的代码，如代码清单29-3所示。

代码清单29-3 添加服务启动代码（PhotoGalleryFragment.java）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenu(true);

    updateItems();

    Intent i = new Intent(getActivity(), PollService.class);
    getActivity().startService(i);

    mThumbnailThread = new ThumbnailDownloader<ImageView>(new Handler());
    ...
}
```

运行应用，在LogCat窗口查看返回结果，可看到类似图29-2所示的画面。

```
D 12-01 19:45… jdwp          Got wake-up signal, bailing out of select
D 12-01 19:45… dalvikvm      Debugger has detached; object registry had 1 entries
I 12-01 19:45… PhotoFetcher  Fetching URL: http://api.flickr.com/services/rest/?method=flickr.phot
                             bbafa75bf6d2cb5af54f937bb70&extras=url_s
I 12-01 19:45… PollService   Received an intent: Intent { cmp=com.bignerdranch.android.photogaller
```

图29-2 服务的第一步

29.2 服务的作用

是不是觉得查看LogCat日志很乏味？确实是！但我们刚添加的代码着实令人兴奋！为什么？利用它可以完成什么？

再次回到我们的假想之地，在那里，我们不再是应用开发者，而是与闪电侠一起工作的鞋店工作人员。

鞋店内有两处地方可以工作：与客户打交道的前台，以及不与客户接触的后台。根据零售店

的规模，工作后台可大可小。

目前为止，我们的所有代码都在activity中运行。activity就是Android应用的前台。所有运行代码都专注于提供良好的用户视觉体验。

而服务就是Android应用的后台。用户无需关心后台发生的一切。即使前台关闭，activity长时间停止运行，后台服务依然可以持续不断地执行工作任务。

好了，关于鞋店的假想可以告一段落了。有什么服务可以完成，但activity却做不到的事情吗？有！在用户离开当前应用去别处时，服务依然可以在后台运行。

后台网络连接的安全

服务将在后台查询Flickr网站。为保证后台网络连接的安全性，我们需进一步完善代码。Android为用户提供了关闭后台应用网络连接的功能。对于非常耗电的应用而言，这项功能可极大地改善手机的续航能力。

然而，这也意味着在后台连接网络时，需使用`ConnectivityManager`确认网络连接是否可用。因为Android API经常随版本的升级而变化，因此这里需要两处检查。首先，需确认`ConnectivityManager.getBackgroundDataSetting()`方法的返回值为`true`，其次再确认`ConnectivityManager.getActiveNetworkInfo()`方法的返回结果不为空。

参照代码清单29-4添加相应的检查代码。完成后，我们来讲解一下具体实现细节。

代码清单29-4　检查后台网络的可用性（PollService.java）

```
@Override
public void onHandleIntent(Intent intent) {
    ConnectivityManager cm = (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);
    @SuppressWarnings("deprecation")
    boolean isNetworkAvailable = cm.getBackgroundDataSetting() &&
        cm.getActiveNetworkInfo() != null;
    if (!isNetworkAvailable) return;

    Log.i(TAG, "Received an intent: " + intent);
}
```

为什么需要两处代码检查呢？在Android旧版本系统中，应检查`getBackgroundDataSetting()`方法的返回结果，如果返回结果为`false`，表示不允许使用后台数据，那我们也就解脱了。当然，如果不去检查，也能随意使用后台数据。但这样做很可能会出问题（电量耗光或应用运行缓慢）。既然代码检查不费吹灰之力，有什么理由不去做呢？。

而在Android 4.0（Ice Cream Sandwich）中，后台数据设置直接会禁用网络。这也是为什么需要检查`getActiveNetworkInfo()`方法是否返回空值的原因。如果返回为空，则网络不可用。对用户来说，这是好事，因为这意味着后台数据设置总是按用户的预期行事。当然，对开发者来说，还有一些额外的工作要做。

要使用`getActiveNetworkInfo()`方法，还需获取`ACCESS_NETWORK_STATE`权限，如代码清单29-5所示。

代码清单29-5 获取网络状态权限（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" />
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    ...
  </application>

</manifest>
```

29.3 查找最新返回结果

后台服务会一直查看最新的返回结果，因此它需知道最近一次获取的结果。使用SharedPreferences保存结果值再合适不过了。在FlickrFetchr类中添加另一个常量，用于存储最近一次获取图片的ID，如代码清单29-6所示。

代码清单29-6 添加最近获取图片ID的preference常量（FlickrFetchr.java）

```
public class FlickrFetchr {
    public static final String TAG = "PhotoFetcher";

    public static final String PREF_SEARCH_QUERY = "searchQuery";
    public static final String PREF_LAST_RESULT_ID = "lastResultId";

    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "xxx";
    ...
```

接下来更新服务代码，以下为需要处理的任务：

- 从默认SharedPreferences中获取当前查询结果以及上一次结果ID；
- 使用FlickrFetchr类获取最新结果集；
- 如果有结果返回，抓取结果的第一条；
- 检查确认是否不同于上一次结果ID；
- 将第一条结果保存回SharedPreferences。

返回PollService.java，添加以上任务的实现代码。代码清单29-7中的代码很长，但其中的代码大家都很熟悉，这里就不再赘述了。

代码清单29-7　检查最新返回结果（PollService.java）

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

        if (!isNetworkAvailable) return;

        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
        String query = prefs.getString(FlickrFetchr.PREF_SEARCH_QUERY, null);
        String lastResultId = prefs.getString(FlickrFetchr.PREF_LAST_RESULT_ID, null);

        ArrayList<GalleryItem> items;
        if (query != null) {
            items = new FlickrFetchr().search(query);
        } else {
            items = new FlickrFetchr().fetchItems();
        }

        if (items.size() == 0)
            return;

        String resultId = items.get(0).getId();

        if (!resultId.equals(lastResultId)) {
            Log.i(TAG, "Got a new result: " + resultId);
        } else {
            Log.i(TAG, "Got an old result: " + resultId);
        }

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

看到前面各项讨论任务的对应实现代码了吗？干的不错。现在，运行PhotoGallery应用，可看到应用首先获取了最新结果。如选择搜索查询，则随后启动应用时，很可能会看到和上次同样的结果。

29.4　使用 AlarmManager 延迟运行服务

为保证服务在后台的切实可用，当没有activity在运行时，需通过某种方式在后台执行一些任务。比如说，设置一个5分钟间隔的定时器。

一种方式是调用Handler的sendMessageDelayed(...)或者postDelayed(...)方法。但如果用户离开当前应用，进程就会停止，Handler消息也会随之消亡，因此该解决方案并不可靠。

因此，我们应转而使用AlarmManager。AlarmManager是可以发送Intent的系统服务。

如何告诉AlarmManager发送什么样的intent呢？使用PendingIntent。我们可以使用PendingIntent打包intent：“我想启动PollService服务。”然后，将其发送给系统中的其他部件，如AlarmManager。

在PollService类中，实现一个启停定时器的setServiceAlarm(Context,boolean)方法，如代码清单29-8所示。该方法是一个静态方法。这样，可使定时器代码和与之相关的代码都放置在PollService类中，但同时又允许其他系统部件调用它。要知道，我们通常会从前端的fragment或其他控制层代码中启停定时器。

代码清单29-8 添加定时方法（PollService.java）

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

    private static final int POLL_INTERVAL = 1000 * 15; // 15 seconds

    public PollService() {
        super(TAG);
    }

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

    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();
        }
    }
}
```

以上代码中，首先是通过调用PendingIntent.getService(...)方法，创建一个用来启动PollService的PendingIntent。PendingIntent.getService(...)方法打包了一个Context.startService(Intent)方法的调用。它有四个参数：一个用来发送intent的Context、一个区分PendingIntent来源的请求代码，待发送的Intent对象以及一组用来决定如何创建PendingIntent的标志符。（稍后将使用其中的一个。）

接下来，需要设置或取消定时器。设置定时器可调用AlarmManager.setRepeating(...)方法。该方法同样具有四个参数：一个描述定时器时间基准的常量（稍后详述）、定时器运行的开始时间、定时器循环的时间间隔以及一个到时要发送的PendingIntent。

取消定时器可调用AlarmManager.cancel(PendingIntent)方法。通常，也需同步取消PendingIntent。稍后，我们将介绍取消PendingIntent的操作是如何有助于我们跟踪定时器状态的。

添加一些快速测试代码，从PhotoGalleryFragment中启动PollService服务，如代码清单29-9所示。

代码清单29-9　添加定时器启动代码（PhotoGalleryFragment.java）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenu(true);

    updateItems();

    Intent i = new Intent(getActivity(), PollService.class);
    getActivity().startService(i);
    PollService.setServiceAlarm(getActivity(), true);

    mThumbnailThread = new ThumbnailDownloader<ImageView>(new Handler());
    ...
}
```

完成以上代码添加后，运行PhotoGallery应用。然后，立即点按后退键并退出应用。

注意观察LogCat日志窗口。可看到PollService服务运行了一次，随后以15秒为间隔再次运行。这正是AlarmManager设计实现的功能。即使进程停止了，AlarmManager依然会不断发送intent，以反复启动PollService服务。（这种后台服务行为有时非常恼人。为彻底清除它，可能需要卸载相关应用。）

29.4.1　PendingIntent

我们来进一步了解前面提及的PendingIntent。PendingIntent是一种token对象。调用PendingIntent.getService(...)方法获取PendingIntent时，我们告诉操作系统："请记住，我需要使用startService(Intent)方法发送这个intent。"随后，调用PendingIntent对象的send()方法时，操作系统会按照我们的要求发送原来封装的intent。

PendingIntent真正精妙的地方在于，将PendingIntenttoken交给其他应用使用时，它是代表当前应用发送token对象的。另外，PendingIntent本身存在于操作系统而不是token里，因此实际上是我们在控制着它。如果不顾及别人感受的话，也可以在交给别人一个PendingIntent对象后，立即撤销它，让send()方法啥也做不了。

如果使用同一个intent请求PendingIntent两次，得到的PendingIntent仍会是同一个。我们可借此测试某个PendingIntent是否已存在，或撤销已发出的PendingIntent。

29.4.2　使用PendingIntent管理定时器

一个PendingIntent只能登记一个定时器。这也是isOn值为false时，setServiceAlarm(Context,boolean)方法的工作原理：首先调用AlarmManager.cancel(PendingIntent)方法撤销PendingIntent的定时器，然后撤销PendingIntent。

既然撤销定时器也随即撤消了PendingIntent，可通过检查PendingIntent是否存在，确认定时器激活与否。具体代码实现时，传入PendingIntent.FLAG_NO_CREATE标志给PendingIntent.getService(...)方法即可。该标志表示如果PendingIntent不存在，则返回null值，而不是创建它。

添加一个名为isServiceAlarmOn(Context)的方法，并传入PendingIntent.FLAG_NO_CREATE标志，以判断定时器的启停状态，如代码清单29-10所示。

代码清单29-10 添加isServiceAlarmOn()方法（PollService.java）

```
public static void setServiceAlarm(Context context, boolean isOn) {
    ...
}

public static boolean isServiceAlarmOn(Context context) {
    Intent i = new Intent(context, PollService.class);
    PendingIntent pi = PendingIntent.getService(
            context, 0, i, PendingIntent.FLAG_NO_CREATE);
    return pi != null;
}
```

这里的PendingIntent仅用于设置定时器，因此PendingIntent空值表示定时器还未设置。

29.5 控制定时器

既然可以开关定时器并判定其启停状态，接下来我们通过图形界面对其进行开关控制。首先添加另一菜单项到menu/fragment_photo_gallery.xml，如代码清单29-11所示。

代码清单29-11 添加服务开关（menu/fragment_photo_gallery.xml）

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/menu_item_search"
    android:title="@string/search"
    android:icon="@android:drawable/ic_menu_search"
    android:showAsAction="ifRoom"
    android:actionViewClass="android.widget.SearchView"
    />
  <item android:id="@+id/menu_item_clear"
    android:title="@string/clear_search"
    android:icon="@android:drawable/ic_menu_close_clear_cancel"
    android:showAsAction="ifRoom"
    />
  <item android:id="@+id/menu_item_toggle_polling"
    android:title="@string/start_polling"
    android:showAsAction="ifRoom"
    />
</menu>
```

然后添加一些字符串资源，一个用于启动polling，一个用于停止polling，如代码清单29-12所示。（后续还需要其他一些字符串资源，如显示在状态栏的通知信息，因此现在也一并完成添加。）

代码清单29-12　添加polling字符串资源（res/values/strings.xml）

```
<resources>

    ...
    <string name="search">Search</string>
    <string name="clear_search">Clear Search</string>
    <string name="start_polling">Poll for new pictures</string>
    <string name="stop_polling">Stop polling</string>
    <string name="new_pictures_title">New PhotoGallery Pictures</string>
    <string name="new_pictures_text">You have new pictures in PhotoGallery.</string>

</resources>
```

删除前面用于启动定时器的快速测试代码，添加用于菜单项的实现代码，如代码清单29-13所示。

代码清单29-13　菜单项切换实现（PhotoGalleryFragment.java）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenu(true);

    updateItems();

    PollService.setServiceAlarm(getActivity(), true);

    mThumbnailThread = new ThumbnailDownloader<ImageView>(new Handler());
    ...
}

...

@Override
@TargetApi(11)
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_search:
            ...
        case R.id.menu_item_clear:
            ...

            updateItems();
            return true;
        case R.id.menu_item_toggle_polling:
            boolean shouldStartAlarm = !PollService.isServiceAlarmOn(getActivity());
            PollService.setServiceAlarm(getActivity(), shouldStartAlarm);

            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

完成代码添加后，应该就可以启停定时器了。

如何更新菜单项呢？我们将在下一节给出答案。

更新选项菜单项

通常，开发选项菜单，只需完成菜单的用户界面即可。然而，有时还需更新选项菜单，以反映应用的状态。

当前，即使是旧式选项菜单，也不会在每次使用时重新实例化生成。如需更新某个选项菜单项的内容，我们应在onPrepareOptionsMenu(Menu)方法中添加实现代码。除了菜单的首次创建外，每次菜单需要配置都会调用该方法。

添加onPrepareOptionsMenu(Menu)方法及其实现代码，检查定时器的开关状态，然后相应更新menu_item_toggle_polling的标题文字，提供反馈信息供用户查看，如代码清单29-14所示。

代码清单29-14 添加onPrepareOptionsMenu(Menu)方法（PhotoGalleryFragment.java）

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    ...
}

@Override
public void onPrepareOptionsMenu(Menu menu) {
    super.onPrepareOptionsMenu(menu);

    MenuItem toggleItem = menu.findItem(R.id.menu_item_toggle_polling);
    if (PollService.isServiceAlarmOn(getActivity())) {
        toggleItem.setTitle(R.string.stop_polling);
    } else {
        toggleItem.setTitle(R.string.start_polling);
    }
}
```

在3.0以前版本的设备中，每次显示菜单时都会调用该方法，这保证了菜单项总能显示正确的文字信息。如需亲自验证，可在3.0以前版本的模拟器上运行PhotoGallery应用。

而在3.0以后版本的设备中，以上做法就不行了。操作栏无法自动更新自己，因此，需通过Activity.invalidateOptionsMenu()方法回调onPrepareOptionsMenu(Menu)方法并刷新菜单项。

在onOptionsItemSelected(MenuItem)方法中添加代码清单29-15所示代码，实现3.0以后版本设备的操作栏更新。

代码清单29-15 失效选项菜单（PhotoGalleryFragment.java）

```
@Override
@TargetApi(11)
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        ...
        case R.id.menu_item_toggle_polling:
            boolean shouldStartAlarm = !PollService.isServiceAlarmOn(getActivity());
            PollService.setServiceAlarm(getActivity(), shouldStartAlarm);

            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB)
                getActivity().invalidateOptionsMenu();
```

```
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

完成代码添加后，在新的4.2系统版本模拟器上，代码应该也可正常运作。

不过，我们还有一处需要完善。

29.6　通知信息

我们的服务已在后台运行并执行指定任务。不过用户对此毫不知情，因此价值不大。

如果服务需要与用户进行信息沟通，通知信息（notification）永远是个不错的选择。通知信息是指显示在通知抽屉上的消息条目，用户可向下滑动屏幕读取。

为发送通知信息，首先需创建一个Notification对象。与第12章的AlertDialog类似，Notification需使用构造对象完成创建。Notification应至少具备：

- 首次显示通知信息时，在状态栏上显示的ticker text；
- ticker text消失后，在状态栏上显示的图标；
- 代表通知信息自身，在通知抽屉中显示的视图；
- 用户点击抽屉中的通知信息，触发PendingIntent。

完成Notification对象的创建后，可调用NotificationManager系统服务的notify(int, Notification)方法发送它。

添加代码清单29-16中的实现代码，创建Notification对象并调用NotificationManager.notify(int, Notification)方法，实现让PollService通知新结果信息给用户。

代码清单29-16　添加通知信息（PollService.java）

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

    String resultId = items.get(0).getId();

    if (!resultId.equals(lastResultId)) {
        Log.i(TAG, "Got a new result: " + resultId);

        Resources r = getResources();
        PendingIntent pi = PendingIntent
            .getActivity(this, 0, new Intent(this, PhotoGalleryActivity.class), 0);

        Notification notification = new NotificationCompat.Builder(this)
            .setTicker(r.getString(R.string.new_pictures_title))
            .setSmallIcon(android.R.drawable.ic_menu_report_image)
            .setContentTitle(r.getString(R.string.new_pictures_title))
            .setContentText(r.getString(R.string.new_pictures_text))
            .setContentIntent(pi)
            .setAutoCancel(true)
            .build();
```

```
            NotificationManager notificationManager = (NotificationManager)
                getSystemService(NOTIFICATION_SERVICE);

            notificationManager.notify(0, notification);
        }

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

我们来从上至下逐行解读新增代码。首先，调用setTicker(CharSequence)和setSmallIcon(int)方法，配置ticker text和小图标。

然后配置Notification在下拉抽屉中的外观。虽然可以定制Notification视图的显示外观和样式，但使用带有图标、标题以及文字显示区域的标准视图要相对更容易些。图标的值来自于setSmallIcon(int)方法，而设置标题和显示文字，我们需分别调用setContentTitle(CharSequence)和setContentText(CharSequence)方法。

接下来，须指定用户点击Notification消息时所触发的动作行为。与AlarmManager类似，这里通过使用PendingIntent来完成指定任务。用户在下拉抽屉中点击Notification消息时，传入setContentIntent(PendingIntent)方法的PendingIntent将会被发送。调用setAutoCancel(true)方法可调整上述行为。使用setAutoCancel(true)设置方法，用户点击Notification消息后，也可将该消息从消息抽屉中删除。

最后，调用NotificationManager.notify(...)方法。传入的整数参数是通知消息的标识符，在整个应用中该值应该是唯一的。如使用同一ID发送两条消息，则第二条消息会替换掉第一条消息。这也是进度条或其他动态视觉效果的实现方式。

本章任务到此结束。我们实现了一个完整的后台服务。确认应用能够正常工作，然后将定时器常量调整为结果更明显的时间，如代码清单29-17所示。

代码清单29-17 调整定时器常量（PollService.java）

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

    public static final int POLL_INTERVAL = 1000 * 15; // 15 seconds
    public static final int POLL_INTERVAL = 1000 * 60 * 5; // 5 minutes

    public PollService() {
        super(TAG);
    }
```

29.7 深入学习：服务细节内容

对于大多数服务任务，推荐使用IntentService。但IntentService模式不一定适合所有架构，因此我们需进一步了解并掌握服务，以便自己实现相关服务。做好接受信息轰炸的心理准备。接下来，我们将学习大量有关服务使用的详细内容与复杂细节。

29.7.1　服务的能与不能

与activity一样，服务是一个提供了生命周期回调方法的应用组件。而且，这些回调方法同样也会在主UI线程上运行。

初始创建的服务不会在后台线程上运行任何代码。这也是我们推荐使用IntentService的最主要原因。大多重要服务都需要某种后台线程，而IntentService已提供了一套标准实现代码。

下面我们来看看服务有哪些生命周期回调方法。

29.7.2　服务的生命周期

通过startService(Intent)方法启动的服务，其生命周期很简单，并具有四种生命周期回调方法。

- onCreate(...)方法。服务创建时调用。
- onStartCommand(Intent,int,int)方法。每次组件通过startService(Intent)方法启动服务时调用一次。两个整数参数，一个是一组标识符，一个是启动ID。标识符用来表示当前intent发送究竟是一次重新发送，还是一次从没成功过的发送。每次调用onStartCommand(Intent,int,int)方法，启动ID都会不同。因此，启动ID也可用于区分不同的命令。
- onDestroy()方法。服务不再需要时调用。通常是在服务停止后。

还有一个问题：服务是如何停止的？根据所开发服务的具体类型，有多种方式可以停止服务。服务的类型由onStartCommand(...)方法的返回值决定，可能的服务类型有Service.START_NOT_STICKY、START_REDELIVER_INTENT和START_STICKY等。

29.7.3　non-sticky服务

IntentService是一种non-sticky服务。non-sticky服务在服务自己认为已完成任务时停止。为获得non-sticky服务，应返回START_NOT_STICKY或START_REDELIVER_INTENT。

通过调用stopSelf()或stopSelf(int)方法，我们告诉Android任务已完成。stopSelf()是一个无条件方法。不管onStartCommand(...)方法调用多少次，该方法总是会成功停止服务。

IntentService使用的是stopSelf(int)方法。该方法需要来自于onStartCommand(...)方法的启动ID。只有在接收到最新启动ID后，该方法才会停止服务。（这也是IntentService工作的后台实现部分。）

返回START_NOT_STICKY和START_REDELIVER_INTENT有什么不同呢？区别就在于，如果系统需要在服务完成任务之前关闭它，则服务的具体表现会有所不同。START_NOT_STICKY型服务会被关闭。而START_REDELIVER_INTENT型服务，则会在可用资源不再吃紧时，尝试再次启动服务。

根据操作于应用的重要程度，在START_NOT_STICKY和START_REDELIVER_INTENT之间做出选择。如服务并不重要，则选择START_NOT_STICKY。PhotoGallery应用中，服务根据定时器的设定重复运行。如发生方法调用失败，也不会产生严重后果，因此应选择START_NOT_STICKY，同时，它也是IntentService的默认行为。我们也可调用IntentService.setIntentRedelivery(true)方法，切换使用START_REDELIVER_INTENT。

29.7.4 sticky服务

sticky服务会持续运行，直到外部组件调用Context.stopService(Intent)方法让它停止为止。为启动sticky服务，应返回START_STICKY。

sticky服务启动后会持续运行，直到某个组件调用Context.stopService(Intent)方法为止。如因某种原因需终止服务，可传入一个null intent给onStartCommand(...)方法，实现服务的重启。

sticky服务适用于长时间运行的服务，如音乐播放器这种启动后一直保持运行状态，直到用户主动停止的服务。即使是这样，也应考虑一种使用non-sticky服务的替代架构方案。sticky服务的管理很不方便，因为判断服务是否已启动会比较困难。

29.7.5 绑定服务

除以上各类型服务外，也可使用bindService(Intent,ServiceConnection,int)方法绑定一个服务。通过服务绑定可连接到一个服务并直接调用它的方法。ServiceConnection是代表服务绑定的一个对象，并负责接收全部绑定回调方法。

在fragment中，绑定代码示例如下：

```
private ServiceConnection mServiceConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className,
            IBinder service) {
        // Used to communicate with the service
        MyBinder binder = (MyBinder)service;
    }

    public void onServiceDisconnected(ComponentName className) {
    }
};

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Intent i = new Intent(c, MyService.class);
    c.bindService(i, mServiceConnection, 0);
}

@Override
public void onDestroy() {
    super.onDestroy();
    getActivity().getApplicationContext().unbindService(mServiceConnection);
}
```

对服务来说，绑定引入了另外两个生命周期回调方法：

- onBind(Intent)方法。绑定服务时调用，返回来自ServiceConnection.onServiceConnected(ComponentName,IBinder)方法的IBinder对象。
- onUnbind(Intent)方法。服务绑定终止时调用。

本地服务绑定

MyBinder是一种怎样的对象呢？如果服务是一个本地服务，MyBinde就可能是本地进程中一个简单的Java对象。通常，MyBinde用于提供一个句柄，以便直接调用服务方法：

```
private class MyBinder extends IBinder {
    public MyService getService() {
        return MyService.this;
    }
}

@Override
public void onBind(Intent intent) {
    return new MyBinder();
}
```

这种模式看上去让人激动。这是Android系统中唯一一处支持组件间直接对话的地方。不过，我们并不推荐此种模式。服务是高效的单例，与仅使用一个单例相比，使用此种模式则显现不出优势。

远程服务绑定

绑定更适用于远程服务，因为它们赋予了其他进程的应用调用服务方法的能力。创建远程绑定服务属于高级主题，不在本书讨论范畴之内。请查阅AIDL或Messager类，了解更多相关内容。