第 23 章

# 23 深入学习intent和任务



本章将使用隐式intent，创建一个启动器应用来替换Android默认的启动器应用。借此应用的实施，我们将深入理解intent、intent过滤器以及Android应用之间是如何交互的。

## 23.1 创建 NerdLauncher 项目

使用与创建CriminalIntent应用相同的设置，创建一个新项目（New→Android Application Project），如图23-1所示。项目名称处填入NerdLauncher，包名处填入`com.bignerdranch.android.nerdlauncher`。

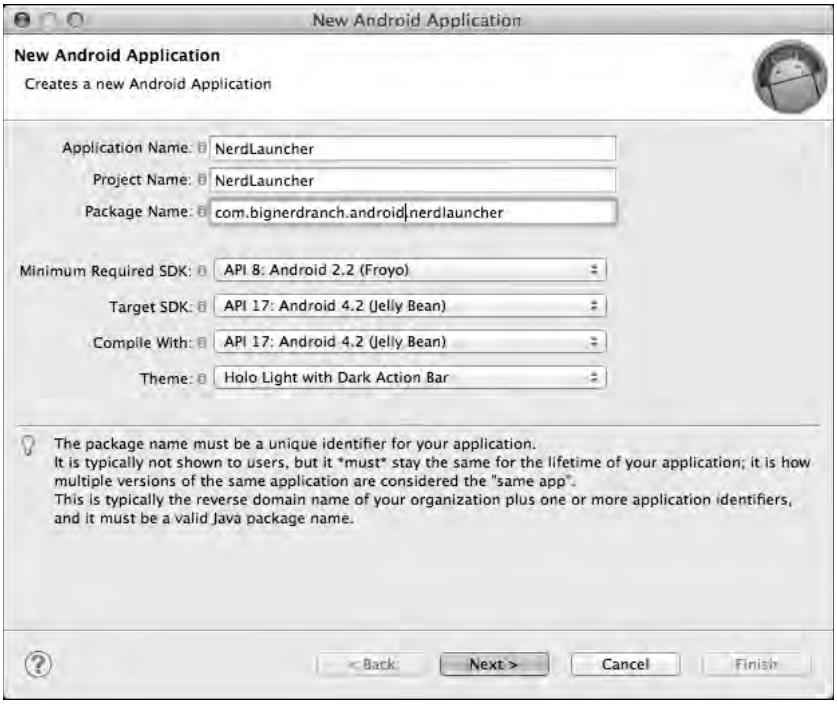


图23-1　创建NerdLauncher项目

单击Next按钮，在接下来的新建项目设置页面，不勾选创建自定义启动图标选项，选择创建一个名为NerdLauncherActivity的全新activity，最后单击Finish按钮完成项目的创建。

NerdLauncherActivity将会继承SingleFragmentActivity类，因此，首先需要将它添加到当前项目中。在包浏览器中，找到CriminalIntent应用包里的SingleFragmentActivity.java文件。将其复制到com.bignerdranch.android.nerdlauncher包中。复制文件时，Eclipse会自动更新包引用。

另外我们还需要activity_fragment.xml布局。因此，再将CriminalIntent项目中的res/layout/activity_fragment.xml文件复制到NerdLauncher项目的res/layout目录中。

NerdLauncher将以列表的形式显示设备上的应用。用户点击任意列表项将启动相应的应用。以下是该应用涉及的对象。

NerdLauncherFragment是ListFragment的子类，它的视图默认为ListFragment自带的ListView视图。

以android.support.v4.app.ListFragment为父类，创建一个名为NerdLauncherFragment的新类。暂时先不理会新建的空类。

打开NerdLauncherActivity.java文件，修改NerdLauncherActivity的超类为SingleFragmentActivity类。然后删除默认的模板代码，并覆盖createFragment()方法返回一个NerdLauncherFragment，如代码清单23-1所示。

**代码清单23-1　另一个SingleFragmentActivity（NerdLauncherActivity.java）**

```
public class NerdLauncherActivity extends Activity SingleFragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_nerd_launcher);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_nerd_launcher, menu);
        return true;
    }

    @Override
    public Fragment createFragment() {
        return new NerdLauncherFragment();
    }
}
```

## 23.2　解析隐式 intent

NerdLaucher应用会以列表的形式向用户展示设备上的应用。要实现该功能，它将发送一个所有应用的主activity都会响应的隐式intent。该隐式intent将包括一个MAIN操作和一个LAUNCHER类别。我们已在以前的项目里见过这种intent过滤器：

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

在NerdLauncherFragment.java中，覆盖onCreate(Bundle)方法创建一个隐式intent。然后，从PackageManager中获取匹配intent的activity列表。当前，先以日志记录下PackageManager返回的activity数，如代码清单23-2所示。

**代码清单23-2 向PackageManager查询activity数（NerdLauncherFragment.java）**

```
public class NerdLauncherFragment extends ListFragment {
    private static final String TAG = "NerdLauncherFragment";

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

        Intent startupIntent = new Intent(Intent.ACTION_MAIN);
        startupIntent.addCategory(Intent.CATEGORY_LAUNCHER);

        PackageManager pm = getActivity().getPackageManager();
        List<ResolveInfo> activities = pm.queryIntentActivities(startupIntent, 0);

        Log.i(TAG, "I've found " + activities.size() + " activities.");
    }
}
```

运行NerdLauncher应用，在LogCat日志窗口查看PackageManager返回的activity数目。

在前面的CriminalIntent应用中，为使用隐式intent触发activity选择器来发送crime报告，我们采取了这样的实现方式：创建一个隐式intent并将其封装在一个选择器intent中，然后调用startActivity(Intent)方法。具体实现代码如下：

```
Intent i = new Intent(Intent.ACTION_SEND);
... // Create and put intent extras
i = Intent.createChooser(i, getString(R.string.send_report));
startActivity(i);
```

有没有感到不解，为什么这里没有使用类似的处理方式？简单的解释是，MAIN/LAUNCHER intent过滤器可能无法与通过startActivity(...)方法发送的MAIN/LAUNCHER隐式intent相匹配。

实际上，调用startActivity(Intent)方法意味着“启动与发送的隐式intent相匹配的默认activity”，而不是想当然的“启动与发送的隐式intent相匹配的activity”。调用startActivity(Intent)方法（或startActivityForResult(...)方法）发送隐式intent时，操作系统会悄然地将Intent.CATEGORY_DEFAULT类别添加给目标intent。

因而，如果希望一个intent过滤器能够与通过startActivity(...)方法发送的隐式intent相匹配，那么必须在对应的intent过滤器中包含DEFAULT类别。

定义了MAIN/LAUNCHER intent过滤器的activity是应用的主要入口点。它只关心作为应用主要入口点要处理的工作。它通常不关心自己是否属于默认的主要入口点，因此，它不必包含CATEGORY_DEFAULT类别。

前面说过，MAIN/LAUNCHER intent过滤器并不一定包含CATEGORY_DEFAULT类别，因此，是否可以与通过 startActivity(...)方法发送的隐式intent匹配，谁也说不准。所以，我们转而使用intent直接向PackageManager查询具有MAIN/LAUNCHER intent过滤器的activity。

接下来，我们需要将查询到的activity标签显示在NerdLauncherFragment的ListView视图中。activity的标签即用户可以识别的显示名称。既然查询到的activity都是启动activity，标签名通常也就是应用名。

在PackageManager返回的ResolveInfo对象中，可以获取activity的标签和其他一些元数据。

首先，添加如下代码对PackageManager返回的ResolveInfo对象按标签（使用ResolveInfo.loadLabel(...)方法）的字母顺序进行排序，如代码清单23-3所示。

**代码清单23-3　按字母顺序对activity进行排序（NerdLauncherFragment.java）**

```
...

Log.i("NerdLauncher", "I've found " + activities.size() + " activities.");

Collections.sort(activities, new Comparator<ResolveInfo>() {
    public int compare(ResolveInfo a, ResolveInfo b) {
        PackageManager pm = getActivity().getPackageManager();
        return String.CASE_INSENSITIVE_ORDER.compare(
            a.loadLabel(pm).toString(),
            b.loadLabel(pm).toString());
    }
});
```

然后，为创建显示activity标签名的简单列表项视图，还需创建一个ArrayAdapter并设置给ListView，如代码清单23-4所示。

**代码清单23-4　创建一个适配器（NerdLauncherFragment.java）**

```
...

Collections.sort(activities, new Comparator<ResolveInfo></ResolveInfo>() {

    ...

    }
});

ArrayAdapter<ResolveInfo> adapter = new ArrayAdapter<ResolveInfo>(
        getActivity(), android.R.layout.simple_list_item_1, activities) {
    public View getView(int pos, View convertView, ViewGroup parent) {
        PackageManager pm = getActivity().getPackageManager();
        View v = super.getView(pos, convertView, parent);
        // Documentation says that simple_list_item_1 is a TextView,
        // so cast it so that you can set its text value
        TextView tv = (TextView)v;
        ResolveInfo ri = getItem(pos);
        tv.setText(ri.loadLabel(pm));
        return v;
    }
};

setListAdapter(adapter);
```

运行NerdLauncher应用。我们将看到一个显示了activity标签的ListView视图，如图23-2所示。

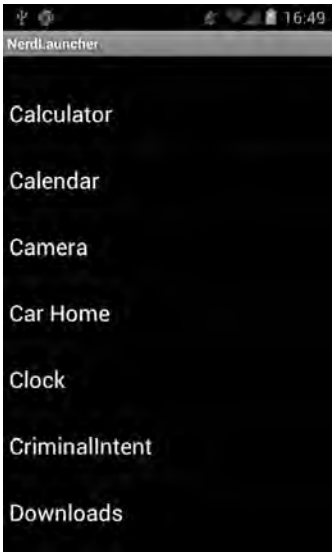


图23-2 设备上的全部activity

## 23.3 在运行时创建显式 intent

上一节，我们使用隐式intent获取匹配的activity并实现以列表的形式展示。接下来就是实现用户点击任一列表项时，启动该列表项的activity。我们将使用显式intent来启动activity。

要创建显式intent，还需从ResolveInfo对象中获取更多数据信息。特别是需要知道activity的包名与类名。这些信息可以从ResolveInfo对象的ActivityInfo中获取。（从ResolveInfo类中还可以获取其他哪些信息，具体请查阅该类的参考文档。）

在NerdLauncherFragment.java中，覆盖onListItemClick(...)方法，取得列表项的ActivityInfo对象。然后，使用ActivityInfo对象中的数据信息，创建一个显式intent并启动目标activity，如代码清单23-5所示。

**代码清单23-5 实现onListItemClick(...)方法（NerdLauncherFragment.java）**

```
@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    ResolveInfo resolveInfo = (ResolveInfo)l.getAdapter().getItem(position);
    ActivityInfo activityInfo = resolveInfo.activityInfo;

    if (activityInfo == null) return;

    Intent i = new Intent(Intent.ACTION_MAIN);
    i.setClassName(activityInfo.applicationInfo.packageName, activityInfo.name);

    startActivity(i);
}
```

从以上代码可以看到，作为显式intent的一部分，我们还发送了ACTION_MAIN操作。发送的intent是否包含操作，对于大多数应用来说没有什么差别。不过，有些应用的启动行为可能会有所不同。取决于不同的启动要求，同样的activity可能会显示不同的用户界面。开发人员最好能明确启动意图，以便让activity完成它应该完成的任务。

在代码清单23-5中，使用获取的包名和类名创建一个显式intent时，我们使用了以下Intent方法：

```
public Intent setClassName(String packageName, String className)
```

这不同于以往创建显式intent的方式。在这之前，我们都是使用一个接受Context和Class对象的Intent构造方法：

```
public Intent(Context packageContext, Class<?> cls)
```

该构造方法使用传入的参数来获取Intent需要的ComponentName。ComponentName由包名和类名共同组成。传入Activity和Class创建Intent时，构造方法会通过Activity类自行确定全路径包名。

也可以自己通过包名和类名创建一个ComponentName，然后再使用下面的Intent方法创建一个显式intent：

```
public Intent setComponent(ComponentName component)
```

然而，setClassName(...)方法能够自动创建组件名，所以使用该方法需要的实现代码相对最少。

运行NerdLauncher应用并尝试启动一些应用。

## 23.4　任务与后退栈

在每一个运行的应用中，Android都使用任务来跟踪用户的状态。任务是用户比较关心的activity栈。栈底部的activity通常称为基activity。栈顶的activity是用户可以看到的activity。用户点击后退键时，栈顶activity会弹出栈外。如果当前屏幕上显示的是基activity，点击后退键，系统将退回主屏幕。

不影响各个任务的状态，任务管理器可以让我们在任务间切换。例如，如果启动进入联系人应用，然后切换到Twitter应用查看信息，这时，我们将启动两个任务。如果再切换回联系人应用，我们在两项任务中所处的状态位置会被保存下来。

有时，我们需要在当前任务中启动activity。而有时又需要在新任务中启动activity，如图23-3所示。

默认情况下，新activity都在当前任务中启动。在CriminalIntent应用中，无论什么时候启动新activity，它都会被添加到当前任务中。即使要启动的activity不属于CriminalIntent应用，它同样也是在当前任务中启动。启动activity发送crime报告就是这样的一个例子。在当前任务中启动activity的好处是，用户可以在任务内而不是在应用层级间导航返回。

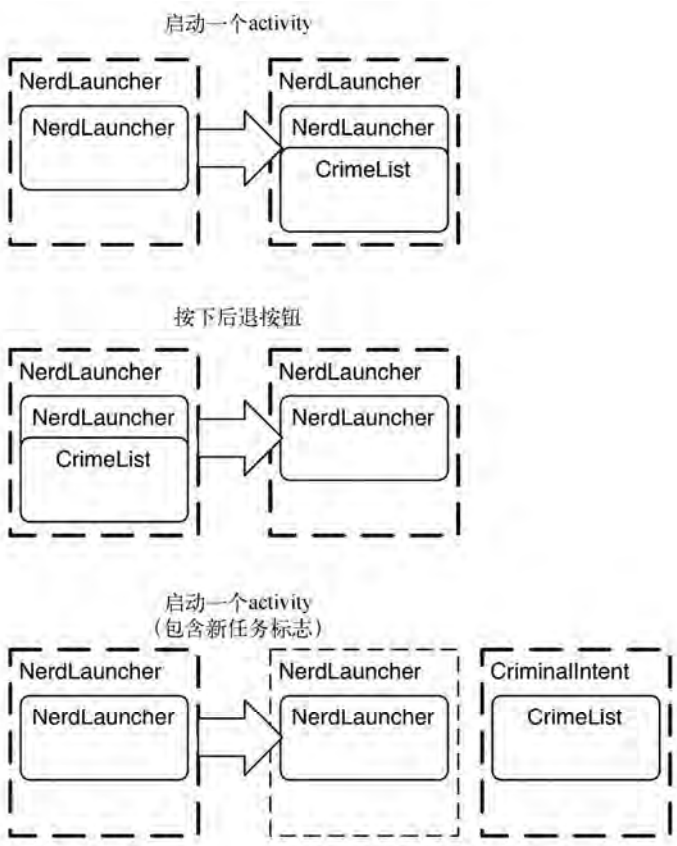


图23-3 任务与后退栈

当前，从NerdLauncher应用启动的任何activity都会添加到NerdLauncher的任务中。可以通过在NerdLauncher应用中启动CriminalIntent应用的activity并启动任务管理器进行验证。（点击设备的“最近应用”按键可以启动任务管理器，如果设备没有该按键，可以长按Home键。）在任务管理器中，我们不会看到任何CriminalIntent应用activity。实际上，启动的`CrimeListActivity`已被添加到NerdLauncher任务中了。如果点击NerdLauncher任务，我们将回到任务管理器启动之前的CriminalIntent应用的用户界面，如图23-4所示。

我们需要NerdLauncher在新任务中启动activity。这样，用户可以在运行的应用间自由切换。为启动新activity时启动新任务，需要为intent添加一个标志，如代码清单23-6所示。

**代码清单23-6 添加新任务标志给intent（NerdLauncherFragment.java）**

```
Intent i = new Intent(Intent.ACTION_MAIN);
i.setClassName(activityInfo.applicationInfo.packageName, activityInfo.name);
i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

startActivity(i);
```


图23-4　CriminalIntent不在自身任务里

运行NerdLauncher应用并启动CriminalIntent。这次，如果启动任务管理器，就会看到CriminalIntent应用处于一个单独的任务中，如图23-5所示。


图23-5　CriminalIntent应用处于独立的任务中

如果从NerdLauncher应用中再次启动CriminalIntent应用，不会看到有第二个`CriminalIntent`任务创建。`FLAG_ACTIVITY_NEW_TASK`标志控制着每个activity仅创建一个任务。`CrimeListActivity`已经有了一个运行的任务，因此，代替创建全新的任务，Android会自动切换到原来的任务中。

## 23.5 使用 NerdLauncher 应用作为设备主屏幕

没人愿意通过启动一个应用来启动其他应用。因此，以替换Android主界面（home screen）的方式使用NerdLauncher应用会更合适一些。打开NerdLauncher项目的配置文件AndroidManifest.xml，对照代码清单23-7在intent主过滤器添加以下节点定义。



**代码清单23-7 修改NerdLauncher应用的类别（AndroidManifest.xml）**

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
  <category android:name="android.intent.category.HOME" />
  <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

通过添加HOME和DEFAULT类别定义，NerdLauncher应用的activity会成为可选的主界面。点击Home键，可以看到，在弹出的界面选择对话框中，NerdLauncher变成了主界面可选项。

（如果已设置NerdLauncher应用为主界面，然后需要恢复到系统默认设置，可以选择Settings→Applications→Manage Applications菜单项，找到NerdLauncher应用并清除它的Launch by default选项。）

## 23.6 挑战练习：应用图标与任务重排

本章使用了ResolveInfo.loadLabel(...)方法，在启动器应用中显示了各个activity的名称。ResolveInfo类还提供了另一个名为loadIcon()的方法。可以使用该方法为每一个应用加载显示图标。你要接受的挑战就是，为NerdLauncher应用中显示的所有应用添加对应的图标。

加载显示图标是不是没有挑战性？那么，还可以尝试添加另一个activity给NerdLauncher应用，以实现在不同的运行任务间切换。要完成任务，需要使用ActivityManager系统服务，该系统服务提供了当前运行的activity、任务以及应用的有用信息。不像PackageManager类，Activity类并没有提供类似于getActivityManager()的便利方法来获取ActivityManager系统服务。

你应该使用Activity.ACTIVITY_SERVICE常量调用Activity.getSystemService()方法，来获取ActivityManager。然后调用ActivityManager的getRunningTasks()方法，得到按运行时间由近及早排序的运行任务列表。再调用moveTaskToFront()方法实现将任意任务切换到前台。最后，要提醒的是，关于任务间的切换，还需要在配置文件中增加其他权限配置。具体使用信息，请查阅Android参考文档。

## 23.7 进程与任务

对象需要内存和虚拟机的支持才能存在。进程是操作系统创建的供应用对象生存以及应用运行的地方。

进程通常占用由操作系统管理着的系统资源，如内存、网络端口以及打开的文件等。进程还拥有至少一个（可能多个）执行线程。在Android系统中，进程总会有一个运行的Dalvik虚拟机。

尽管存在着未知的异常情况，但总的来说，Android世界里的每个应用组件都仅与一个进程相关联。应用伴随着自己的进程一起完成创建，该进程同时也是应用中所有组件的默认进程。

虽然，组件可以指派给不同的进程，但我们推荐使用默认进程。如果确实需要在不同进程中运行应用组件，通常也可以借助多线程来达到目的。相比多进程的使用，Android多线程的使用更加简单。

每一个activity实例都仅存在于一个进程和一个任务中。这也是进程与任务的唯一类似的地方。任务只包含activity，这些activity通常来自于不同应用。而进程则包含了应用的全部运行代码和对象。

进程与任务很容易让人混淆，主要原因在于它们不仅在概念上有某种重叠，而且通常都是以它们所属应用的名称被人提及的。例如，从NerdLauncher启动器中启动CriminalIntent应用时，操作系统创建了一个CriminalIntent进程以及一个以`CrimeListActivity`为基activity的新任务。在任务管理器中，我们可以看到标签为CriminalIntent的任务。

activity赖以生存的任务和进程有可能会不同。例如，在CriminalIntent应用中启动联系人应用选择嫌疑人时，虽然联系人activity是在CriminalIntent任务中启动的，但它是在联系人应用的进程中运行的，如图23-6所示。

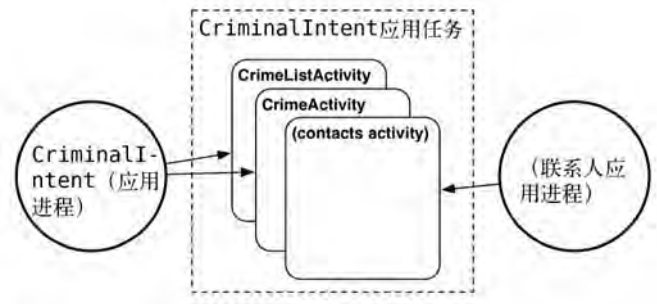


图23-6　任务与进程

这也意味着，用户点击后退键在不同activity间导航时，他或她可能还没意识到他们正在进程间切换。

本章我们创建了任务并实现了任务间的切换。有没有想过终止任务或替换Android默认的任务管理器呢？很遗憾，Android没有提供任何处理它们的方法。虽然长按Home键可以硬链接到默认的任务管理器，但我们没有办法终止任务。不过，我们可以终止进程。Google Play商店中那些宣称自己是任务终止器的应用，实际上都是进程终止器。