第 21 章 隐式intent

在Android系统中，使用隐式intent可以启动其他应用的activity。在显式intent中，我们指定要启动的activity类，操作系统会负责启动它。在隐式intent中，我们描述清楚要完成的任务，操作系统会找到合适的应用，并在其中启动相应的activity。

在CriminalIntent应用中，我们将使用隐式intent实现从联系人列表中选取`Crime`嫌疑人，然后发送文字陋习报告给他。实际使用时，用户从设备上安装的任意联系人应用中选取联系人，然后再从操作系统提供的信息发送应用列表中，选取目标应用将陋习报告发送出去。

图21-1　打开联系人和消息发送应用

从开发者角度来看，使用隐式intent利用其他应用完成常见任务，远比自己编写代码从头实现要容易得多。从用户角度来看，他们也乐意在当前应用中协同使用自己熟悉或喜爱的应用。

创建使用隐式intent前，需先在CriminalIntent应用中完成以下设置准备工作：

- 在CrimeFragment的布局上添加Choose Suspect和Send Crime Report按钮；
- 在Crime类中添加保存嫌疑人姓名的mSuspect变量；
- 使用格式化的字符串资源创建陋习报告模板。

21.1　添加按钮组件

首先，我们需要在CrimeFragment布局中添加两个投诉用按钮。添加按钮前，先来添加显示在按钮上的字符串资源，如代码清单21-1所示。

代码清单21-1　为按钮添加字符串（strings.xml）

```
    <string name="take">Take!</string>
    <string name="crime_suspect_text">Choose Suspect</string>
    <string name="crime_report_text">Send Crime Report</string>
</resources>
```

然后，在layout/fragment_crime.xml布局文件中，参照图21-2，添加两个按钮组件。注意，在布局定义示意图中，为集中注意力在新增组件的内容定义上，我们隐藏了第一个线形布局及其全部子元素。

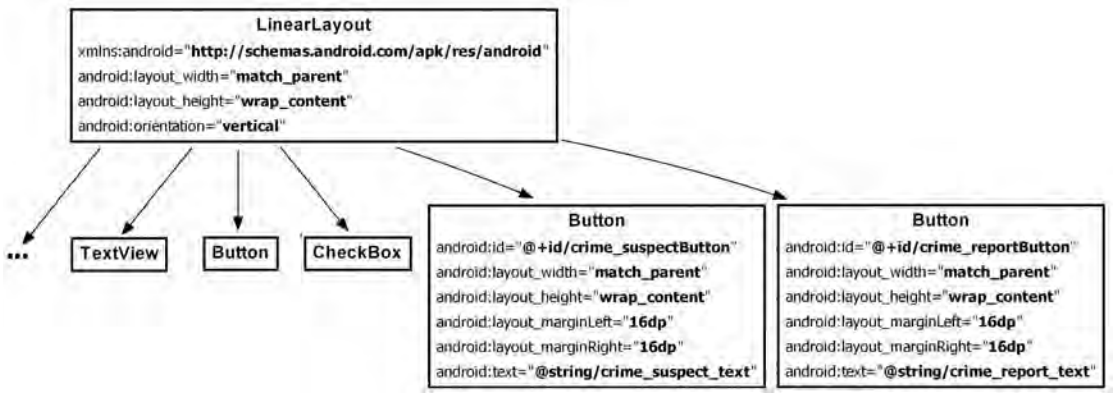

图21-2　添加嫌疑人选取和陋习报告发送按钮（layout/fragment_crime.xml）

在水平模式布局中，需要将新增按钮作为子元素添加到一个新的水平线性布局中，并放在包含日期按钮和“Solved?”单选框的线性布局下方。

如图21-3所示，在比较小的设备屏幕上，新添加的按钮无法完整的显示。为解决这个问题，需要将整个布局定义放在一个ScrollView中。

图21-4展示了更新后的布局定义。现在，因为ScrollView是整个布局的根元素，记得不要忘了将命名空间定义从原来的根元素移至ScrollView根元素。

现在，可以预览更新后的布局了，当然，也可以直接运行CriminalIntent应用，确认新增加的按钮是否显示正常。

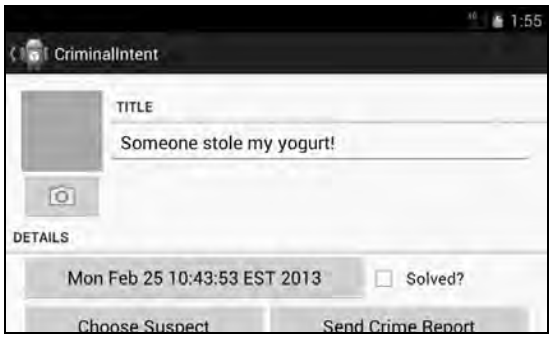

图21-3 水平模式下，新增按钮没有完全显示

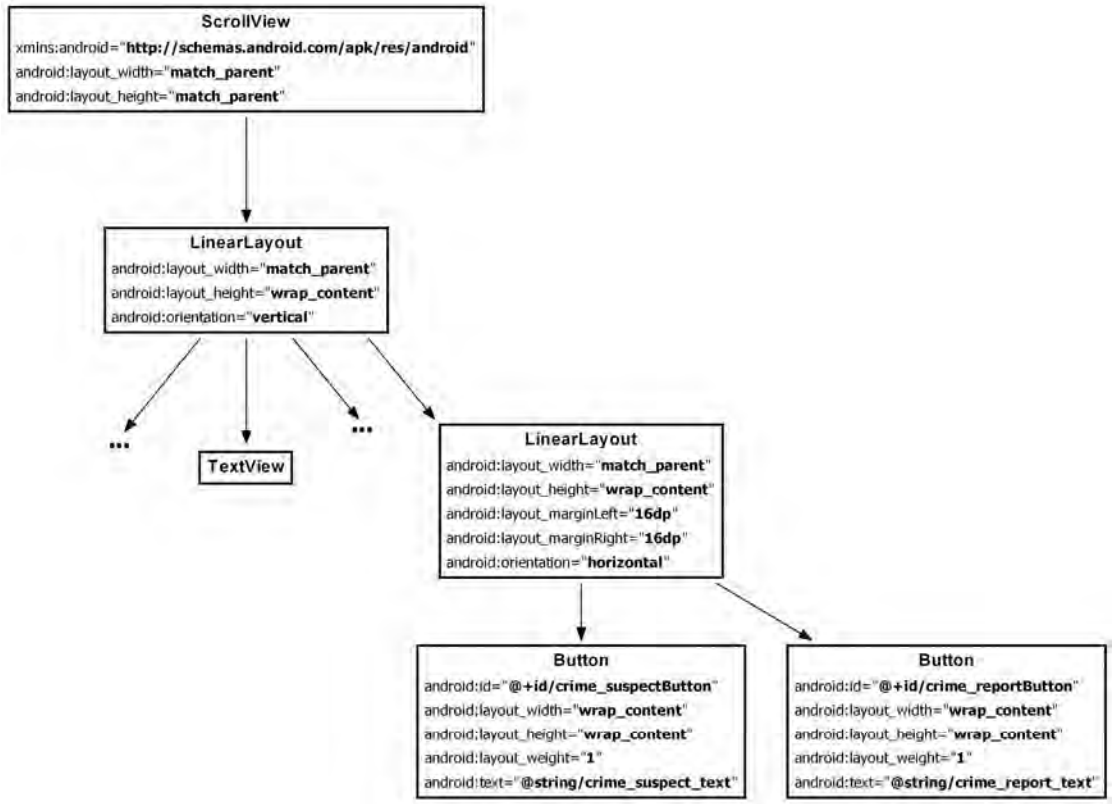

图21-4 添加嫌疑人选取和陋习报告发送按钮（layout-land/fragment_crime.xml）

21.2 添加嫌疑人信息至模型层

接下来，返回到Crime.java中，新增JSON常量以及存储嫌疑人姓名的`mSuspect`成员变量各一

个。然后修改JSON方法，将嫌疑人信息序列化为JSON格式（或从JSON格式还原嫌疑人信息）并添加相应的存取方法，如代码清单21-2所示。

代码清单21-2　添加嫌疑人成员变量（Crime.java）

```
public class Crime {
    ...
    private static final String JSON_PHOTO = "photo";
    private static final String JSON_SUSPECT = "suspect";

    ...
    private Photo mPhoto;
    private String mSuspect;

    public Crime(JSONObject json) throws JSONException {
        mId = UUID.fromString(json.getString(JSON_ID));
        ...
        if (json.has(JSON_PHOTO))
            mPhoto = new Photo(json.getJSONObject(JSON_PHOTO));
        if (json.has(JSON_SUSPECT))
            mSuspect = json.getString(JSON_SUSPECT);
    }

    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        ...
        if (mPhoto != null)
            json.put(JSON_PHOTO, mPhoto.toJSON());
        json.put(JSON_SUSPECT, mSuspect);
        return json;
    }

    public void setPhoto(Photo p) {
        mPhoto = p;
    }

    public String getSuspect() {
        return mSuspect;
    }
    public void setSuspect(String suspect) {
        mSuspect = suspect;
    }

}
```

21.3　使用格式化字符串

最后一项准备任务是创建可填充的陋习报告模板。应用运行前，我们无法获知具体陋习细节。因此，必须使用带有占位符（可在应用运行时替换）的格式化字符串。下面是将要使用的格式化字符串：

```
<string name="crime_report">%1$s! The crime was discovered on %2$s. %3$s, and %4$s
```

%1$s、%2$s等特殊字符串即为占位符，它们接受字符串参数。在代码中，我们将调用getString(...)方法，并传入格式化字符串资源ID以及四个其他字符串参数（与要替

换的占位符顺序一致）。

首先，在strings.xml中，添加代码清单21-3所示的字符串资源。

代码清单21-3 添加字符串资源（strings.xml）

```
    <string name="crime_suspect_text">Choose Suspect</string>
    <string name="crime_report_text">Send Crime Report</string>
    <string name="crime_report">%1$s!
      The crime was discovered on %2$s. %3$s, and %4$s
    </string>
    <string name="crime_report_solved">The case is solved</string>
    <string name="crime_report_unsolved">The case is not solved</string>
    <string name="crime_report_no_suspect">There is no suspect.</string>
    <string name="crime_report_suspect">The suspect is %s.</string>
    <string name="crime_report_subject">CriminalIntent Crime Report</string>
    <string name="send_report">Send crime report via</string>

</resources>
```

然后，在CrimeFragment.java中，添加`getCrimeReport()`方法创建四段字符串信息，并返回拼接完整的陋习报告文本信息，如代码清单21-4所示。

代码清单21-4 新增`getCrimeReport()`方法（CrimeFragment.java）

```
private String getCrimeReport() {
    String solvedString = null;
    if (mCrime.isSolved()) {
        solvedString = getString(R.string.crime_report_solved);
    } else {
        solvedString = getString(R.string.crime_report_unsolved);
    }

    String dateFormat = "EEE, MMM dd";
    String dateString = DateFormat.format(dateFormat, mCrime.getDate()).toString();

    String suspect = mCrime.getSuspect();
    if (suspect == null) {
        suspect = getString(R.string.crime_report_no_suspect);
    } else {
        suspect = getString(R.string.crime_report_suspect, suspect);
    }

    String report = getString(R.string.crime_report,
        mCrime.getTitle(), dateString, solvedString, suspect);

    return report;
}
```

至此，准备工作全部完成了，接下来学习如何使用隐式intent。

21.4 使用隐式 intent

`Intent`对象可以向操作系统描述我们需要处理的任务。使用显式intent，我们需明确地告诉操作系统要启动的activity类名。下面是之前创建过的显式intent：

```
Intent i = new Intent(getActivity(), CrimeCameraActivity.class);
startActivity(i);
```

而使用隐式intent，只需向操作系统描述清楚我们的工作意图。操作系统会去启动那些对外宣称能够胜任工作任务的activity。如果操作系统找到多个符合的activity，用户将会看到一个可选应用列表，然后就看用户如何选择了。

21.4.1　典型隐式intent的组成

下面是一个隐式intent的主要组成部分，可以用来定义我们的工作任务。

(1) 要执行的操作。

通常以Intent类中的常量来表示。例如，要访问查看某个URL，可以使用Intent.ACTION_VIEW；要发送邮件，可以使用Intent.ACTION_SEND。

(2) 要访问数据的位置。

这可能是设备以外的资源，如某个网页的URL，也可能是指向某个文件的URI，或者是指向ContentProvider中某条记录的某个内容URI（content URI）。

(3) 操作涉及的数据类型。

这指的是MIME形式的数据类型，如text/html或audio/mpeg3。如果一个intent包含某类数据的位置，那么通常可以从中推测出数据的类型。

(4) 可选类别。

如果操作用于描述具体要做什么，那么类别通常用来描述我们是何时、何地或者说如何使用某个activity的。Android的android.intent.category.LAUNCHER类别表明，activity应该显示在顶级应用启动器中。而android.intent.category.INFO类别表明，虽然activity向用户显示了包信息，但它不应该显示在启动器中。

一个用来查看某个网址的简单隐式intent会包括一个Intent.ACTION_VIEW操作，以及某个具体URL网址的uri数据。

基于以上信息，操作系统将启动适用应用的适用activity（如果有多个适用应用可选，用户可自行如何选择）。

通过配置文件中的intent过滤器设置，activity会对外宣称自己是适合处理ACTION_VIEW的activity。如果是开发一款浏览器应用，为响应ACTION_VIEW操作，需要在activity声明中包含以下intent过滤器：

```
<activity
  android:name=".BrowserActivity"
  android:label="@string/app_name" >
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http" android:host="www.bignerdranch.com" />
  </intent-filter>
</activity>
```

DEFAULT类别必须明确地在intent过滤器中进行设置。intent过滤器中的action元素告诉操作系统，activity能够处理指定的任务，DEFAULT类别告诉操作系统，activity愿意处理某项任务。

DEFAULT类别实际隐含添加到了几乎每一个隐式intent中。（唯一的例外是LAUNCHER类别，第23章会用到它。）

如同显式intent，隐式intent也可以包含extra信息。不过，操作系统在寻找适用的activity时，它不会使用任何附加在隐式intent上的extra。

注意，隐式intent的操作和数据部分也可以与显式intent联合起来使用。这相当于要求特定activity去处理特定任务。

21.4.2 发送陋习报告

在CriminalIntent应用中，通过创建一个发送陋习报告的隐式intent，我们来看看隐式intent是如何工作的。陋习报告是由字符串组成的文本信息，我们的任务是发送一段文字信息，因此，隐式intent的操作是ACTION_SEND。它不指向任何数据，也不包含任何类别，但会指定数据类型为text/plain。

在CrimeFragment.onCreateView(...)方法中，首先以资源ID引用Send Crime Report按钮并为其设置一个监听器。然后在监听器接口实现中，创建一个隐式intent并传入startActivity(Intent)方法，如代码清单21-5所示。

代码清单21-5 发送陋习报告（CrimeFragment.java）

```
        ...

        Button reportButton = (Button)v.findViewById(R.id.crime_reportButton);
        reportButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Intent i = new Intent(Intent.ACTION_SEND);
                i.setType("text/plain");
                i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
                i.putExtra(Intent.EXTRA_SUBJECT,
                    getString(R.string.crime_report_subject));
                startActivity(i);
            }
        });

        return v;
    }
```

以上代码使用了一个接受字符串参数的Intent构造方法，我们传入的是一个定义操作的常量。取决于要创建的隐式intent类别，也可以使用一些其他形式的构造方法。其他全部intent构造方法及其使用说明，可以查阅Intent参考文档。因为没有接受数据类型的构造方法可用，所以必须专门设置它。

报告文本以及报告主题字符串是作为extra附加到intent上的。注意，这些extra信息使用了Intent类中定义的常量。因此，任何响应该intent的activity都知道这些常量，自然也知道如何使用它们关联的值。

运行CriminalIntent应用并点击Send Crime Report按钮。因为刚创建的intent会匹配设备上的许多activity，我们很可能看到一个长长的候选activity列表，如图21-5所示。

图21-5　愿意发送陋习报告的全部activity

从列表中作出选择后，可以看到，陋习报告信息都加载到了所选应用中，接下来，只需填入地址，点击发送即可。

然而，有时我们可能看不到候选activity列表。出现这种情况通常有两个原因，要么是针对某个隐式intent设置了默认响应应用，要么是设备上只有唯一一个activity可以响应隐式intent。

通常，对于某项操作，最好使用用户的默认应用。不过，在CriminalIntent应用中，针对`ACTION_SEND`操作，应该总是将选择权交给用户。要知道，也许今天用户想低调处理问题，只采取邮件的形式发送陋习报告，而明天很可能就改变主意了，他或她更希望通过Twitter公开抨击那些公共场所的陋习。

使用隐式intent启动activity时，也可以创建一个每次都显示的activity选择器。和以前一样创建一个隐式intent后，调用以下`Intent`方法并传入创建的隐式intent以及用作选择器标题的字符串：

```
public static Intent createChooser(Intent target, String title)
```

然后，将`createChooser(...)`方法返回的intent传入`startActivity(...)`方法。

在CrimeFragment.java中，创建一个选择器显示响应隐式intent的全部activity，如代码清单21-6所示。

代码清单21-6 使用选择器（CrimeFragment.java）

```
public void onClick(View v) {
    Intent i = new Intent(Intent.ACTION_SEND);
    i.setType("text/plain");
    i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
    i.putExtra(Intent.EXTRA_SUBJECT,
        getString(R.string.crime_report_subject));
    i = Intent.createChooser(i, getString(R.string.send_report));
    startActivity(i);
}
```

运行CriminalIntent应用并点击Send Crime Report按钮。可以看到，只要有多个activity可以处理隐式intent，我们都会得到一个候选activity列表，如图21-6所示。

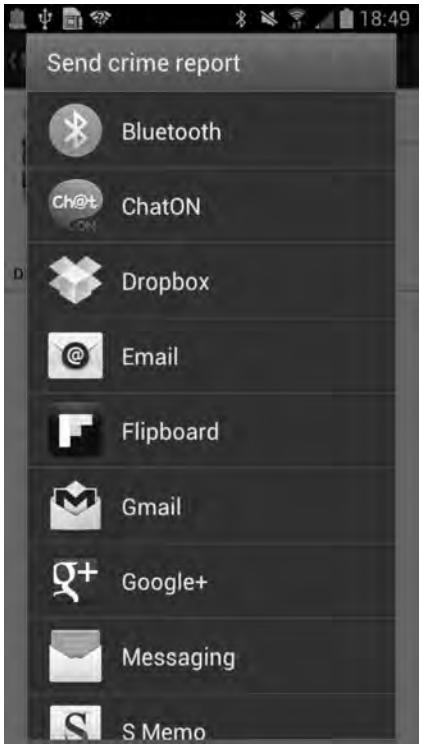

图21-6 通过选择器选择应用发送文本信息

21.4.3 获取联系人信息

现在，创建另一个隐式intent，实现让用户从联系人应用里选择嫌疑人。新建的隐式intent将由操作以及数据获取位置组成。操作为`Intent.ACTION_PICK`。联系人数据获取位置为`ContactsContract.Contacts.CONTENT_URI`。简言之，我们是请求Android协助从联系人数据库里获取某个具体联系人。

因为要获取启动activity的返回结果，所以我们调用`startActivityForResult(...)`方法并传入intent和请求码。在CrimeFragment.java中，新增请求码常量和按钮成员变量，如代码清单21-7所示。

代码清单21-7　添加suspect按钮成员变量（CrimeFragment.java）

```
...
private static final int REQUEST_PHOTO = 1;
private static final int REQUEST_CONTACT = 2;

...

private ImageButton mPhotoButton;
private Button mSuspectButton;

...
```

在`onCreateView(...)`方法的末尾，引用新增按钮并为其设置监听器。在监听器接口实现中，创建一个隐式intent并传入`startActivityForResult(...)`方法。最后，如果`Crime`有与之关联的联系人，那么就将其姓名显示在按钮上，如代码清单21-8所示。

代码清单21-8　发送隐式intent（CrimeFragment.java）

```
        ...

        mSuspectButton = (Button)v.findViewById(R.id.crime_suspectButton);
        mSuspectButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
               Intent i = new Intent(Intent.ACTION_PICK,
                    ContactsContract.Contacts.CONTENT_URI);
               startActivityForResult(i, REQUEST_CONTACT);
            }
        });

        if (mCrime.getSuspect() != null) {
            mSuspectButton.setText(mCrime.getSuspect());
        }

        return v;
    }
```

运行CriminalIntent应用并点击Choose Suspect按钮。我们会看到一个类似图21-7所示的联系人列表。

注意，如果设备上安装了其他联系人应用，那么应用界面可能会有所不同。这里，我们又一次受益于隐式intent。可以看到，从当前应用中调用联系人应用时，我们无需知道应用的名字。因此，用户可以安装任何喜爱的联系人应用，操作系统会负责找到并启动它。

1. 从联系人列表中获取联系人数据

现在，我们需要从联系人应用中获取返回结果。很多应用都在共享联系人信息，因此，Android提供了一个深度定制的API用于处理联系人信息，这主要是通过`ContentProvider`类来实现的。该类的实例封装了联系人数据库并提供给其他应用使用。我们可以通过一个`ContentResolver`访问`ContentProvider`。

图21-7 包含嫌疑人的联系人列表

前面，activity是以需返回结果的方式启动的，所以我们会调用`onActivityResult(...)`方法来接收一个intent。该intent包括了数据URI。该数据URI是一个指向用户所选联系人的定位符。

在CrimeFragment.java中，将以下代码添加到`onActivityResult(...)`实现方法中，如代码清单21-9所示。

代码清单21-9 获取联系人姓名（CrimeFragment.java）

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;
    if (requestCode == REQUEST_DATE) {
        ...
    } else if (requestCode == REQUEST_PHOTO) {
        ...
    } else if (requestCode == REQUEST_CONTACT) {
        Uri contactUri = data.getData();

        // Specify which fields you want your query to return
        // values for.
        String[] queryFields = new String[] {
            ContactsContract.Contacts.DISPLAY_NAME
        };
        // Perform your query - the contactUri is like a "where"
        // clause here
        Cursor c = getActivity().getContentResolver()
            .query(contactUri, queryFields, null, null, null);
```

```
            // Double-check that you actually got results
            if (c.getCount() == 0) {
                c.close();
                return;
            }

            // Pull out the first column of the first row of data -
            // that is your suspect's name.
            c.moveToFirst();
            String suspect = c.getString(0);
            mCrime.setSuspect(suspect);
            mSuspectButton.setText(suspect);
            c.close();
        }
    }
```

代码清单21-9创建了一条查询语句，要求返回全部联系人的显示名字。然后查询联系人数据库，获得一个可用的Cursor。因为已经知道Cursor只包含一条记录，所以将Cursor移动到第一条记录并获取它的字符串形式。该字符串即为嫌疑人的姓名。然后，使用它设置Crime嫌疑人，并显示在Choose Suspect按钮上。

（联系人数据库本身是一个比较复杂的主题，这里不会展开讨论。如果需要了解更多信息，可以阅读Contacts Provider API指南：http://developer.android.com/guide/topics/providers/contacts-provider.html。）

2. 联系人信息使用权限

我们是如何获得读取联系人数据库的权限呢？实际上，这是联系人应用将其权限临时拓展给了我们。联系人应用具有使用联系人数据库的全部权限。联系人应用返回包含在intent中的URI数据给父activity时，它会添加一个Intent.FLAG_GRANT_READ_URI_PERMISSION标志。该标志向Android示意，CriminalIntent应用中的父activity可以使用联系人数据一次。这种方式工作的很好，因为，我们不需要访问整个联系人数据库，只需访问数据库中的一条联系人信息。

21.4.4　检查可以响应的activity

本章创建的两个隐式intent总是会得到响应。因为，任何Android设备都会自带一个Email应用和一个联系人应用。但是如果某个设备上没有任何与目标隐式intent相匹配的activity，会出现什么情况呢？答案是，如果操作系统找不到匹配的activity，那么应用会立即崩溃。

解决办法是首先通过操作系统中的PackageManager类进行自检。具体代码实现如下:

```
PackageManager pm = getPackageManager();
List<ResolveInfo> activities = pm.queryIntentActivities(yourIntent, 0);
boolean isIntentSafe = activities.size() > 0;
```

将intent传入PackageManager类的queryIntentActivities(...)方法中，该方法会返回一个对象列表。这些对象包含响应传入intent的activity的元数据信息。我们只需确认对象列表中至少包含一个有效项。也就是说，设备上，至少有一个activity可以响应我们的intent。

我们可以在onCreateView(...)方法中运行以上检查，对于设备不能响应的特色intent，直

接禁用相应的功能。

21.5 挑战练习：又一个隐式 intent

代替发送陋习报告，愤怒的用户可能更倾向于直接责问陋习嫌疑人。新增一个按钮，直接拨打已知姓名的陋习嫌疑人的电话。

首先需要来自联系人数据库的手机号码。然后，使用电话URI，创建一个隐式intent：

```
Uri number = Uri.parse("tel:5551234");
```

电话相关的intent操作通常有两种：Intent.ACTION_DIAL和Intent.ACTION_CALL。ACTION_CALL直接调出手机应用并立即拨打来自intent的电话号码；而ACTION_DIAL则拨好号码，然后等待用户发起通话。

我们推荐使用ACTION_DIAL操作。ACTION_CALL使用有限制并且需要特定的使用权限。点击Call按钮前，如果用户有机会冷静下来，他们应该会很欢迎这种贴心的设计。