# Intent 和 Intent 过滤器

本文内容



另请参阅



`Intent` 是一个消息传递对象，您可以使用它从其他应用组件请求操作。尽管 Intent 可以通过多种方式促进组件之间的通信，但其基本用例主要包括以下三个：

- **启动 Activity**：

  `Activity` 表示应用中的一个屏幕。通过将 `Intent` 传递给 `startActivity()`，您可以启动新的 `Activity` 实例。`Intent` 描述了要启动的 Activity，并携带了任何必要的数据。

  如果您希望在 Activity 完成后收到结果，请调用 `startActivityForResult()`。在 Activity 的 `onActivityResult()` 回调中，您的 Activity 将结果作为单独的 `Intent` 对象接收。如需了解详细信息，请参阅 Activity 指南。

- **启动服务**：

  `Service` 是一个不使用用户界面而在后台执行操作的组件。通过将 `Intent` 传递给 `startService()`，您可以启动服务执行一次性操作（例如，下载文件）。`Intent` 描述了要启动的服务，并携带了任何必要的数据。

  如果服务旨在使用客户端-服务器接口，则通过将 `Intent` 传递给 `bindService()`，您可以从其他组件绑定到此服务。如需了解详细信息，请参阅服务指南。

- **传递广播**：

  广播是任何应用均可接收的消息。系统将针对系统事件（例如：系统启动或设备开始充电时）传递各种广播。通过将 `Intent` 传递给 `sendBroadcast()`、`sendOrderedBroadcast()` 或 `sendStickyBroadcast()`，您可以将广播传递给其他应用。

## Intent 类型

Intent 分为两种类型：

- **显式 Intent**：按名称（完全限定类名）指定要启动的组件。 通常，您会在自己的应用中使用显式 Intent 来启动组件，这是因为您知道要启动的 Activity 或服务的类名。例如，启动新 Activity 以响应用户操作，或者启动服务以在后台下载文件。

- **隐式 Intent** ：不会指定特定的组件，而是声明要执行的常规操作，从而允许其他应用中的组件来处理它。 例如，如需在地图上向用户显示位置，则可以使用隐式 Intent，请求另一具有此功能的应用在地图上显示指定的位置。

创建显式 Intent 启动 Activity 或服务时，系统将立即启动 `Intent` 对象中指定的应用组件。

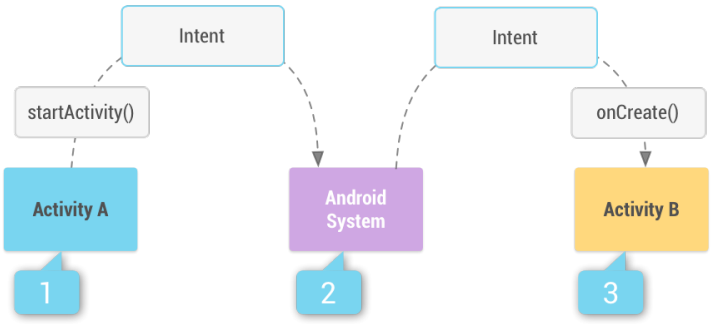


**图 1.** 隐式 Intent 如何通过系统传递以启动其他 Activity 的图解：**[1]** Activity A 创建包含操作描述的 `Intent`，并将其传递给 `startActivity()`。**[2]** Android 系统搜索所有应用中与 Intent 匹配的 Intent 过滤器。 找到匹配项之后，**[3]** 该系统通过调用匹配 Activity（Activity B）的 `onCreate()` 方法并将其传递给 `Intent`，以此启动匹配 Activity。

创建隐式 Intent 时，Android 系统通过将 Intent 的内容与在设备上其他应用的清单文件中声明的 Intent 过滤器进行比较，从而找到要启动的相应组件。 如果 Intent 与 Intent 过滤器匹配，则系统将启动该组件，并向其传递 `Intent` 对象。 如果多个 Intent 过滤器兼容，则系统会显示一个对话框，支持用户选取要使用的应用。

Intent 过滤器是应用清单文件中的一个表达式，它指定该组件要接收的 Intent 类型。 例如，通过为 Activity 声明 Intent 过滤器，您可以使其他应用能够直接使用某一特定类型的 Intent 启动 Activity。同样，如果您没有为 Activity 声明任何 Intent 过滤器，则 Activity 只能通过显式 Intent 启动。

> **注意：**为了确保应用的安全性，启动 `Service` 时，请始终使用显式 Intent，且不要为服务声明 Intent 过滤器。使用隐式 Intent 启动服务存在安全隐患，因为您无法确定哪些服务将响应 Intent，且用户无法看到哪些服务已启动。从 Android 5.0（API 级别 21）开始，如果使用隐式 Intent 调用 `bindService()`，系统会引发异常。

# 构建 Intent

`Intent` 对象携带了 Android 系统用来确定要启动哪个组件的信息（例如，准确的组件名称或应当接收该 Intent 的组件类别），以及收件人组件为了正确执行操作而使用的信息（例如，要采取的操作以及要处理的数据）。

`Intent` 中包含的主要信息如下：

**组件名称**

要启动的组件名称。

这是可选项，但也是构建**显式** Intent 的一项重要信息，这意味着 Intent 应当仅传递给由组件名称定义的应用组件。 如果没有组件名称，则 Intent 是**隐式**的，且系统将根据其他 Intent 信息（例如，以下所述的操作、数据和类别）决定哪个组件应当接收 Intent。 因此，如需在应用中启动特定的组件，则应指定该组件的名称。

> **注：**启动 `Service` 时，您应**始终指定组件名称**。 否则，您无法确定哪项服务会响应 Intent，且用户无法看到哪项服务已启动。

`Intent` 的这一字段是一个 `ComponentName` 对象，您可以使用目标组件的完全限定类名指定此对象，其中包括应用的软件包名称。 例如， `com.example.ExampleActivity`。您可以使用 `setComponent()`、`setClass()`、`setClassName()` 或 `Intent` 构造函数设置组件名称。

**操作**

指定要执行的通用操作（例如，“查看”或“选取”）的字符串。

对于广播 Intent，这是指已发生且正在报告的操作。操作在很大程度上决定了其余 Intent 的构成，特别是数据和 extra 中包含的内容。

您可以指定自己的操作，供 Intent 在您的应用内使用（或者供其他应用在您的应用中调用组件）。但是，您通常应该使用由 `Intent` 类或其他框架类定义的操作常量。以下是一些用于启动 Activity 的常见操作：

`ACTION_VIEW`

如果您拥有一些某项 Activity 可向用户显示的信息（例如，要使用图库应用查看的照片；或者要使用地图应用查看的地址），请使用 Intent 将此操作与 `startActivity()` 结合使用。

`ACTION_SEND`

这也称为“共享”Intent。如果您拥有一些用户可通过其他应用（例如，电子邮件应用或社交共享应用）共享的数据，则应使用 Intent 将此操作与 `startActivity()` 结合使用。

有关更多定义通用操作的常量，请参阅 `Intent` 类参考文档。 其他操作在 Android 框架中的其他位置定义。例如，对于在系统的设置应用中打开特定屏幕的操作，将在 `Settings` 中定义。

您可以使用 `setAction()` 或 `Intent` 构造函数为 Intent 指定操作。

如果定义自己的操作，请确保将应用的软件包名称作为前缀。 例如：

```
static final String ACTION_TIMETRAVEL = "com.example.action.TIMETRAVEL";
```

**数据**

引用待操作数据和/或该数据 MIME 类型的 URI（`Uri` 对象）。提供的数据类型通常由 Intent 的操作决定。例如，如果操作是 `ACTION_EDIT`，则数据应包含待编辑文档的 URI。

创建 Intent 时，除了指定 URI 以外，指定数据类型（其 MIME 类型）往往也很重要。例如，能够显示图像的 Activity 可能无法播放音频文件，即便 URI 格式十分类似时也是如此。因此，指定数据的 MIME 类型有助于 Android 系统找到接收 Intent 的最佳组件。但有时，MIME 类型可以从 URI 中推断得出，特别当数据是 `content:` URI 时尤其如此。这表明数据位于设备中，且由 `ContentProvider` 控制，这使得数据 MIME 类型对系统可见。

要仅设置数据 URI，请调用 `setData()`。 要仅设置 MIME 类型，请调用 `setType()`。如有必要，您可以使用 `setDataAndType()` 同时显式设置二者。

> **注意：**若要同时设置 URI 和 MIME 类型，**请勿**调用 `setData()` 和 `setType()`，因为它们会互相抵消彼此的值。请始终使用 `setDataAndType()` 同时设置 URI 和 MIME 类型。

**类别**

一个包含应处理 Intent 组件类型的附加信息的字符串。 您可以将任意数量的类别描述放入一个 Intent 中，但大多数 Intent 均不需要类别。 以下是一些常见类别：

`CATEGORY_BROWSABLE`

目标 Activity 允许本身通过网络浏览器启动，以显示链接引用的数据，如图像或电子邮件。

`CATEGORY_LAUNCHER`

该 Activity 是任务的初始 Activity，在系统的应用启动器中列出。

有关类别的完整列表，请参阅 `Intent` 类描述。

您可以使用 `addCategory()` 指定类别。

以上列出的这些属性（组件名称、操作、数据和类别）表示 Intent 的既定特征。 通过读取这些属性，Android 系统能够解析应当启动哪个应用组件。

但是，Intent 也有可能会一些携带不影响其如何解析为应用组件的信息。 Intent 还可以提供：

**Extra**

携带完成请求操作所需的附加信息的键值对。正如某些操作使用特定类型的数据 URI 一样，有些操作也使用特定的 extra。

您可以使用各种 `putExtra()` 方法添加 extra 数据，每种方法均接受两个参数：键名和值。您还可以创建一个包含所有 extra 数据的 `Bundle` 对象，然后使用 `putExtras()` 将`Bundle` 插入 `Intent` 中。

例如，使用 `ACTION_SEND` 创建用于发送电子邮件的 Intent 时，可以使用 `EXTRA_EMAIL` 键指定“目标”收件人，并使用 `EXTRA_SUBJECT` 键指定“主题”。

`Intent` 类将为标准化的数据类型指定多个 `EXTRA_*` 常量。如需声明自己的 extra 键（对于应用接收的 Intent），请确保将应用的软件包名称作为前缀。 例如：

```
static final String EXTRA_GIGAWATTS = "com.example.EXTRA_GIGAWATTS";
```

**标志**

在 `Intent` 类中定义的、充当 Intent 元数据的标志。 标志可以指示 Android 系统如何启动 Activity（例如，Activity 应属于哪个任务），以及启动之后如何处理（例如，它是否属于最近的 Activity 列表）。

如需了解详细信息，请参阅 `setFlags()` 方法。

# 显式 Intent 示例

显式 Intent 是指用于启动某个特定应用组件（例如，应用中的某个特定 Activity 或服务）的 Intent。 要创建显式 Intent，请为 `Intent` 对象定义组件名称 — Intent 的所有其他属性均为可选属性。

例如，如果在应用中构建了一个名为 `DownloadService`、旨在从网页下载文件的服务，则可使用以下代码启动该服务：

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

`Intent(Context, Class)` 构造函数分别为应用和组件提供 `Context` 和 `Class` 对象。因此，此 Intent 将显式启动该应用中的 `DownloadService` 类。

如需了解有关构建和启动服务的详细信息，请参阅服务指南。

# 隐式 Intent 示例

隐式 Intent 指定能够在可以执行相应操作的设备上调用任何应用的操作。 如果您的应用无法执行该操作而其他应用可以，且您希望用户选取要使用的应用，则使用隐式 Intent 非常有用。

例如，如果您希望用户与他人共享您的内容，请使用 `ACTION_SEND` 操作创建 Intent，并添加指定共享内容的 extra。 使用该 Intent 调用 `startActivity()` 时，用户可以选取共享内容所使用的应用。

> **注意：**用户可能没有*任何*应用处理您发送到 `startActivity()` 的隐式 Intent。如果出现这种情况，则调用将会失败，且应用会崩溃。要验证 Activity 是否会接收 Intent，请对 `Intent` 对象调用 `resolveActivity()`。如果结果为非空，则至少有一个应用能够处理该 Intent，且可以安全调用 `startActivity()`。 如果结果为空，则不应使用该 Intent。如有可能，您应停用发出该 Intent 的功能。

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

> **注：**在这种情况下，系统并没有使用 URI，但已声明 Intent 的数据类型，用于指定 extra 携带的内容。

调用 `startActivity()` 时，系统将检查已安装的所有应用，确定哪些应用能够处理这种 Intent（即：含 `ACTION_SEND` 操作并携带“text/plain”数据的 Intent ）。 如果只有一个应用能够处理，则该应用将立即打开并为其提供 Intent。 如果多个 Activity 接受 Intent，则系统将显示一个对话框，使用户能够选取要使用的应用。

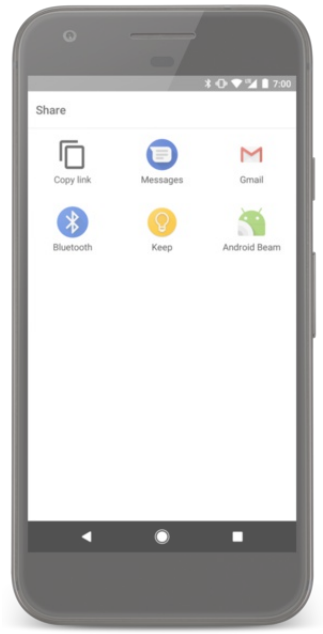


**图 2.** 选择器对话框。

## 强制使用应用选择器

如果有多个应用响应隐式 Intent，则用户可以选择要使用的应用，并将其设置为该操作的默认选项。 如果用户可能希望今后一直使用相同的应用执行某项操作（例如，打开网页时，用户往往倾向于仅使用一种网络浏览器），则这一点十分有用。

但是，如果多个应用可以响应 Intent，且用户可能希望每次使用不同的应用，则应采用显式方式显示选择器对话框。 选择器对话框每次都会要求用户选择用于操作的应用（用户无法为该操作选择默认应用）。 例如，当应用使用 `ACTION_SEND` 操作执行“共享”时，用户根据目前的状况可能需要使用另一不同的应用，因此应当始终使用选择器对话框，如图 2 中所示。

要显示选择器，请使用 `createChooser()` 创建 `Intent`，并将其传递给 `startActivity()`。例如：

```
Intent sendIntent = new Intent(Intent.ACTION_SEND);
...

// Always use string resources for UI text.
// This says something like "Share this photo with"
String title = getResources().getString(R.string.chooser_title);
// Create intent to show the chooser dialog
Intent chooser = Intent.createChooser(sendIntent, title);

// Verify the original intent will resolve to at least one activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```

这将显示一个对话框，其中有响应传递给 `createChooser()` 方法的 Intent 的应用列表，并且将提供的文本用作对话框标题。

# 接收隐式 Intent

要公布应用可以接收哪些隐式 Intent，请在清单文件中使用 `<intent-filter>` 元素为每个应用组件声明一个或多个 Intent 过滤器。每个 Intent 过滤器均根据 Intent 的操作、数据和类别指定自身接受的 Intent 类型。 仅当隐式 Intent 可以通过 Intent 过滤器之一传递时，系统才会将该 Intent 传递给应用组件。

> **注：**显式 Intent 始终会传递给其目标，无论组件声明的 Intent 过滤器如何均是如此。

应用组件应当为自身可执行的每个独特作业声明单独的过滤器。例如，图像库应用中的一个 Activity 可能会有两个过滤器，分别用于查看图像和编辑图像。 当 Activity 启动时，它将检查 `Intent` 并根据 `Intent` 中的信息决定具体的行为（例如，是否显示编辑器控件）。

每个 Intent 过滤器均由应用清单文件中的 `<intent-filter>` 元素定义，并嵌套在相应的应用组件（例如，`<activity>` 元素）中。 在 `<intent-filter>` 内部，您可以使用以下三个元素中的一个或多个指定要接受的 Intent 类型：

`<action>`

在 `name` 属性中，声明接受的 Intent 操作。该值必须是操作的文本字符串值，而不是类常量。

`<data>`

使用一个或多个指定数据 URI 各个方面（`scheme`、`host`、`port`、`path` 等）和 MIME 类型的属性，声明接受的数据类型。

`<category>`

在 `name` 属性中，声明接受的 Intent 类别。该值必须是操作的文本字符串值，而不是类常量。

> **注：**为了接收隐式 Intent，**必须**将 `CATEGORY_DEFAULT` 类别包括在 Intent 过滤器中。 方法 `startActivity()` 和 `startActivityForResult()` 将按照已申明 `CATEGORY_DEFAULT` 类别的方式处理所有 Intent。 如果未在 Intent 过滤器中声明此类别，则隐式 Intent 不会解析为您的 Activity。

例如，以下是一个使用包含 Intent 过滤器的 Activity 声明，当数据类型为文本时，系统将接收 `ACTION_SEND` Intent ：

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

您可以创建一个包括多个 `<action>`、`<data>` 或 `<category>` 实例的过滤器。创建时，仅需确定组件能够处理这些过滤器元素的任何及所有组合即可。

如需仅以操作、数据和类别类型的特定组合来处理多种 Intent，则需创建多个 Intent 过滤器。

系统通过将 Intent 与所有这三个元素进行比较，根据过滤器测试隐式 Intent。 隐式 Intent 若要传递给组件，必须通过所有这三项测试。如果 Intent 甚至无法匹配其中任何一项测试，则 Android 系统不会将其传递给组件。 但是，由于一个组件可能有多个 Intent 过滤器，因此未能通过某一组件过滤器的 Intent 可能会通过另一过滤器。如需了解有关系统如何解析 Intent 的详细信息，请参阅下文的 Intent 解析部分。

> **注意：**为了避免无意中运行不同应用的 `Service`，请始终使用显式 Intent 启动您自己的服务，且不必为该服务声明 Intent 过滤器。

> **限制对组件的访问**
>
> 使用 Intent 过滤器时，无法安全地防止其他应用启动组件。 尽管 Intent 过滤器将组件限制为仅响应特定类型的隐式 Intent，但如果开发者确定您的组件名称，则其他应用有可能通过使用显式 Intent 启动您的应用组件。如果必须确保*只有您自己的应用才*能启动您的某一组件，请针对该组件将 `exported` 属性设置为 `"false"`。

> **注：**对于所有 Activity，您必须在清单文件中声明 Intent 过滤器。但是，广播接收器的过滤器可以通过调用 `registerReceiver()` 动态注册。 稍后，您可以使用 `unregisterReceiver()` 注销该接收器。这样一来，应用便可仅在应用运行时的某一指定时间段内侦听特定的广播。

## 过滤器示例

为了更好地了解一些 Intent 过滤器的行为，我们一起来看看从社交共享应用的清单文件中截取的以下片段。

```
<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>
```

第一个 Activity `MainActivity` 是应用的主要入口点。当用户最初使用启动器图标启动应用时，该 Activity 将打开：

- `ACTION_MAIN` 操作指示这是主要入口点，且不要求输入任何 Intent 数据。
- `CATEGORY_LAUNCHER` 类别指示此 Activity 的图标应放入系统的应用启动器。 如果 `<activity>` 元素未使用 `icon` 指定图标，则系统将使用 `<application>` 元素中的图标。

这两个元素必须配对使用，Activity 才会显示在应用启动器中。

第二个 Activity `ShareActivity` 旨在便于共享文本和媒体内容。 尽管用户可以通过从 `MainActivity` 导航进入此 Activity，但也可以从发出隐式 Intent（与两个 Intent 过滤器之一匹配）的另一应用中直接进入 `ShareActivity`。

> **注：**MIME 类型 `application/vnd.google.panorama360+jpg` 是一个指定全景照片的特殊数据类型，您可以使用 Google panorama API 对其进行处理。

# 使用待定 Intent

`PendingIntent` 对象是 `Intent` 对象的包装器。`PendingIntent` 的主要目的是授权外部应用使用包含的 `Intent`，就像是它从您应用本身的进程中执行的一样。

待定 Intent 的主要用例包括：

- 声明用户使用您的通知执行操作时所要执行的 Intent（Android 系统的 `NotificationManager` 执行 `Intent`）。
- 声明用户使用您的 应用小部件执行操作时要执行的 Intent（主屏幕应用执行 `Intent`）。
- 声明未来某一特定时间要执行的 Intent（Android 系统的 `AlarmManager` 执行 `Intent`）。

由于每个 `Intent` 对象均设计为由特定类型的应用组件（`Activity`、`Service` 或 `BroadcastReceiver`）进行处理，因此还必须基于相同的考虑因素创建 `PendingIntent`。使用待定 Intent 时，应用不会使用调用（如 `startActivity()`）执行该 Intent。相反，通过调用相应的创建器方法创建 `PendingIntent` 时，您必须声明所需的组件类型：

- `PendingIntent.getActivity()`，适用于启动 `Activity` 的 `Intent`。
- `PendingIntent.getService()`，适用于启动 `Service` 的 `Intent`。
- `PendingIntent.getBroadcast()`，适用于启动 `BroadcastReceiver` 的 `Intent`。

除非您的应用正在从其他应用中接收待定 Intent，否则上述用于创建 `PendingIntent` 的方法可能是您所需的唯一 `PendingIntent` 方法。

每种方法均会提取当前的应用 `Context`、您要包装的 `Intent` 以及一个或多个指定应如何使用该 Intent 的标志（例如，是否可以多次使用该 Intent）。

如需了解有关使用待定 Intent 的详细信息，请参阅通知和应用小部件 API 指南等手册中每个相应用例的相关文档。

# Intent 解析

当系统收到隐式 Intent 以启动 Activity 时，它根据以下三个方面将该 Intent 与 Intent 过滤器进行比较，搜索该 Intent 的最佳 Activity：

- Intent 操作
- Intent 数据（URI 和数据类型）
- Intent 类别

下文根据如何在应用的清单文件中声明 Intent 过滤器，描述 Intent 如何与相应的组件匹配。

## 操作测试

要指定接受的 Intent 操作，Intent 过滤器既可以不声明任何 `<action>` 元素，也可以声明多个此类元素。例如：

```
<intent-filter>
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.VIEW" />
    ...
</intent-filter>
```

要通过此过滤器，您在 `Intent` 中指定的操作必须与过滤器中列出的某一操作匹配。

如果该过滤器未列出任何操作，则 Intent 没有任何匹配项，因此所有 Intent 均无法通过测试。 但是，如果 `Intent` 未指定操作，则会通过测试（只要过滤器至少包含一个操作）。

## 类别测试

要指定接受的 Intent 类别， Intent 过滤器既可以不声明任何 `<category>` 元素，也可以声明多个此类元素。 例如：

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

若要使 Intent 通过类别测试，则 `Intent` 中的每个类别均必须与过滤器中的类别匹配。反之则未必然，Intent 过滤器声明的类别可以超出 `Intent` 中指定的数量，且 `Intent` 仍会通过测试。 因此，不含类别的 Intent 应当始终会通过此测试，无论过滤器中声明何种类别均是如此。

> **注**：Android 会自动将 `CATEGORY_DEFAULT` 类别应用于传递给 `startActivity()` 和 `startActivityForResult()` 的所有隐式 Intent。因此，如需 Activity 接收隐式 Intent，则必须将 `"android.intent.category.DEFAULT"` 的类别包括在其 Intent 过滤器中（如上文的 `<intent-filter>` 示例所示）。

## 数据测试

要指定接受的 Intent 数据， Intent 过滤器既可以不声明任何 `<data>` 元素，也可以声明多个此类元素。 例如：

```
<intent-filter>
    <data android:mimeType="video/mpeg" android:scheme="http" ... />
    <data android:mimeType="audio/mpeg" android:scheme="http" ... />
    ...
</intent-filter>
```

每个 `<data>` 元素均可指定 URI 结构和数据类型（MIME 媒体类型）。 URI 的每个部分均包含单独的 `scheme`、`host`、`port` 和 `path` 属性：

```
<scheme>://<host>:<port>/<path>
```

例如：

```
content://com.example.project:200/folder/subfolder/etc
```

在此 URI 中，架构是 `content`，主机是 `com.example.project`，端口是 `200`，路径是 `folder/subfolder/etc`。

在 `<data>` 元素中，上述每个属性均为可选，但存在线性依赖关系：

- 如果未指定架构，则会忽略主机。
- 如果未指定主机，则会忽略端口。
- 如果未指定架构和主机，则会忽略路径。

将 Intent 中的 URI 与过滤器中的 URI 规范进行比较时，它仅与过滤器中包含的部分 URI 进行比较。 例如：

- 如果过滤器仅指定架构，则具有该架构的所有 URI 均与该过滤器匹配。
- 如果过滤器指定架构和权限，但未指定路径，则具有相同架构和权限的所有 URI 都会通过过滤器，无论其路径如何均是如此。
- 如果过滤器指定架构、权限和路径，则仅具有相同架构、权限和路径的 URI 才会通过过滤器。

> **注：**路径规范可以包含星号通配符 (*)，因此仅需部分匹配路径名即可。

数据测试会将 Intent 中的 URI 和 MIME 类型与过滤器中指定的 URI 和 MIME 类型进行比较。 规则如下：

a. 仅当过滤器未指定任何 URI 或 MIME 类型时，不含 URI 和 MIME 类型的 Intent 才会通过测试。

b. 对于包含 URI 但不含 MIME 类型（既未显式声明，也无法通过 URI 推断得出）的 Intent，仅当其 URI 与过滤器的 URI 格式匹配、且过滤器同样未指定 MIME 类型时，才会通过测试。

c. 仅当过滤器列出相同的 MIME 类型且未指定 URI 格式时，包含 MIME 类型、但不含 URI 的 Intent 才会通过测试。

d. 仅当 MIME 类型与过滤器中列出的类型匹配时，同时包含 URI 类型和 MIME 类型（通过显式声明，或可以通过 URI 推断得出）的 Intent 才会通过测试的 MIME 类型部分。 如果 Intent 的 URI 与过滤器中的 URI 匹配，或者如果 Intent 具有 `content:` 或 `file:` URI 且过滤器未指定 URI，则 Intent 会通过测试的 URI 部分。 换言之，如果过滤器只是列出 MIME 类型，则假定组件支持 `content:` 和 `file:` 数据。

最后一条规则，即规则 (d)，反映了期望组件能够从文件中或内容提供程序获得本地数据。因此，其过滤器可以仅列出数据类型，而不必显式命名 `content:` 和 `file:` 架构。这是一个典型的案例。 例如，下文中的 `<data>` 元素向 Android 指出，组件可从内容提供商处获得并显示图像数据：

```
<intent-filter>
    <data android:mimeType="image/*" />
    ...
</intent-filter>
```

由于大部分可用数据均由内容提供商分发，因此指定数据类型（而非 URI）的过滤器也许最为常见。

另一常见的配置是具有架构和数据类型的过滤器。例如，下文中的 `<data>` 元素向 Android 指出，组件可从网络中检索视频数据以执行操作：

```
<intent-filter>
    <data android:scheme="http" android:type="video/*" />
    ...
</intent-filter>
```

## Intent 匹配

通过 Intent 过滤器匹配 Intent，这不仅有助于发现要激活的目标组件，还有助于发现设备上组件集的相关信息。 例如，主页应用通过使用指定 `ACTION_MAIN` 操作和 `CATEGORY_LAUNCHER` 类别的 Intent 过滤器查找所有 Activity，以此填充应用启动器。

您的应用可以采用类似的方式使用 Intent 匹配。`PackageManager` 提供了一整套 `query...()` 方法来返回所有能够接受特定 Intent 的组件。此外，它还提供了一系列类似的 `resolve...()` 方法来确定响应 Intent 的最佳组件。 例如，`queryIntentActivities()` 将返回能够执行那些作为参数传递的 Intent 的所有 Activity 列表，而 `queryIntentServices()` 则可返回类似的服务列表。这两种方法均不会激活组件，而只是列出能够响应的组件。 对于广播接收器，有一种类似的方法： `queryBroadcastReceivers()`。