<activity>

语法：

```
<activity android:allowEmbedded=["true" | "false"]
          android:allowTaskReparenting=["true" | "false"]
          android:alwaysRetainTaskState=["true" | "false"]
          android:autoRemoveFromRecents=["true" | "false"]
          android:banner="drawable resource"
          android:clearTaskOnLaunch=["true" | "false"]
          android:configChanges=["mcc", "mnc", "locale",
                                 "touchscreen", "keyboard", "keyboardHidden",
                                 "navigation", "screenLayout", "fontScale",
                                 "uiMode", "orientation", "screenSize",
                                 "smallestScreenSize"]
          android:documentLaunchMode=["intoExisting" | "always" |
                                  "none" | "never"]
          android:enabled=["true" | "false"]
          android:excludeFromRecents=["true" | "false"]
          android:exported=["true" | "false"]
          android:finishOnTaskLaunch=["true" | "false"]
          android:hardwareAccelerated=["true" | "false"]
          android:icon="drawable resource"
          android:label="string resource"
          android:launchMode=["standard" | "singleTop" |
                              "singleTask" | "singleInstance"]
          android:maxRecents="integer"
          android:multiprocess=["true" | "false"]
          android:name="string"
          android:noHistory=["true" | "false"]
          android:parentActivityName="string"
          android:permission="string"
          android:process="string"
          android:relinquishTaskIdentity=["true" | "false"]
          android:resizeableActivity=["true" | "false"]
          android:screenOrientation=["unspecified" | "behind" |
                                     "landscape" | "portrait" |
                                     "reverseLandscape" | "reversePortrait" |
                                     "sensorLandscape" | "sensorPortrait" |
                                     "userLandscape" | "userPortrait" |
                                     "sensor" | "fullSensor" | "nosensor" |
                                     "user" | "fullUser" | "locked"]
          android:stateNotNeeded=["true" | "false"]
          android:supportsPictureInPicture=["true" | "false"]
          android:taskAffinity="string"
          android:theme="resource or theme"
          android:uiOptions=["none" | "splitActionBarWhenNarrow"]
          android:windowSoftInputMode=["stateUnspecified",
                                       "stateUnchanged", "stateHidden",
                                       "stateAlwaysHidden", "stateVisible",
                                       "stateAlwaysVisible", "adjustUnspecified",
                                       "adjustResize", "adjustPan"] >
    . . .
</activity>
```

包含它的文件：

```
<application>
```

可包含：

`<intent-filter>`

`<meta-data>`

说明：

声明一个实现应用的部分可视化用户界面的 Activity（一个 `Activity`子类）。 所有 Activity 都必须由清单文件中的 `<activity>` 元素表示。 任何未在该处声明的 Activity 都将对系统不可见，并且也永远不会被运行。

属性：

`android:allowEmbedded`

表示该 Activity 可作为另一 Activity 的嵌入式子项启动。 它尤其适用于子项所在的容器（如 Display）为另一 Activity 所拥有的情况。 例如，用于 Wear 自定义通知的 Activity 必须声明此项，以便 Wear 在其上下文流中显示 Activity，后者位于另一进程中。

该属性的默认值为 `false`。

`android:allowTaskReparenting`

当启动 Activity 的任务接下来转至前台时，Activity 是否能从该任务转移至与其有亲和关系的任务 —“`true`”表示它可以转移，“`false`”表示它仍须留在启动它的任务处。

如果未设置该属性，则对 Activity 应用由 `<application>` 元素的相应 `allowTaskReparenting` 属性设置的值。 默认值为“`false`”。

正常情况下，当 Activity 启动时，会与启动它的任务关联，并在其整个生命周期中一直留在该任务处。您可以利用该属性强制 Activity 在其当前任务不再显示时将其父项更改为与其有亲和关系的任务。该属性通常用于使应用的 Activity 转移至与该应用关联的主任务。

例如，如果电子邮件包含网页链接，则点击链接会调出可显示网页的 Activity。 该 Activity 由浏览器应用定义，但作为电子邮件任务的一部分启动。 如果将其父项更改为浏览器任务，它会在浏览器下一次转至前台时显示，当电子邮件任务再次转至前台时则会消失。

Activity 的亲和关系由 `taskAffinity` 属性定义。 任务的亲和关系通过读取其根 Activity 的亲和关系来确定。因此，按照定义，根 Activity 始终位于具有相同亲和关系的任务之中。 由于具有“`singleTask`”或“`singleInstance`”启动模式的 Activity 只能位于任务的根，因此更改父项仅限于“`standard`”和“`singleTop`”模式。 （另请参阅 `launchMode` 属性。）

`android:alwaysRetainTaskState`

系统是否始终保持 Activity 所在任务的状态 —“`true`”表示保持，“`false`”表示允许系统在特定情况下将任务重置到其初始状态。 默认值为“`false`”。该属性只对任务的根 Activity 有意义；对于所有其他 Activity，均忽略该属性。

正常情况下，当用户从主屏幕重新选择某个任务时，系统会在特定情况下清除该任务（从根 Activity 之上的堆栈中移除所有 Activity）。 系统通常会在用户一段时间（如 30 分钟）内未访问任务时执行此操作。

不过，如果该属性的值是“`true`”，则无论用户如何到达任务，将始终返回到最后状态的任务。 例如，在网络浏览器这类存在大量用户不愿失去的状态（如多个打开的标签）的应用中，该属性会很有用。

`android:autoRemoveFromRecents`

由具有该属性的 Activity 启动的任务是否一直保留在概览屏幕中，直至任务中的最后一个 Activity 完成为止。 若为 `true`，则自动从概览屏幕中移除任务。 它会替换调用方使用的 `FLAG_ACTIVITY_RETAIN_IN_RECENTS`。 它必须是布尔值“`true`”或“`false`”。

`android:banner`

一种为其关联项提供扩展图形化横幅的可绘制资源。 将其与 `<activity>` 标记联用可为特定 Activity 提供默认横幅，也可与 `<application>` 标记联用，为所有应用 Activity 提供横幅。

系统使用横幅在 Android TV 主屏幕中表示应用。 由于横幅只显示在主屏幕中，因此只应由包含的 Activity 能够处理 `CATEGORY_LEANBACK_LAUNCHER` Intent 的应用指定。

必须将该属性设置为对包含图像的可绘制资源的引用（例如 `"@drawable/banner"`）。 没有默认横幅。

如需了解详细信息，请参阅“面向电视的 UI 模式”设计指南中的横幅，以及“电视应用入门指南”中的提供主屏幕横幅。

`android:clearTaskOnLaunch`

是否每当从主屏幕重新启动任务时都从中移除根 Activity 之外的所有 Activity —“`true`”表示始终将任务清除到只剩其根 Activity；“`false`”表示不做清除。 默认值为“`false`”。该属性只对启动新任务的 Activity（根 Activity）有意义；对于任务中的所有其他 Activity，均忽略该属性。

当值为“`true`”时，每次用户再次启动任务时，无论用户最后在任务中正在执行哪个 Activity，也无论用户是使用*返回*还是*主屏幕*按钮离开，都会将用户转至任务的根 Activity。 当值为“`false`”时，可在某些情况下清除任务中的 Activity（请参阅 `alwaysRetainTaskState` 属性），但并非一律可以。

例如，假定有人从主屏幕启动了 Activity P，然后从那里转到 Activity Q。该用户接着按了*主屏幕*按钮，然后返回到 Activity P。正常情况下，用户将看到 Activity Q，因为那是其最后在 P 的任务中执行的 Activity。 不过，如果 P 将此标志设置为“`true`”，则当用户按下*主屏幕*将任务转入后台时，其上的所有 Activity（在本例中为 Q）都会被移除。 因此用户返回任务时只会看到 P。

如果该属性和 `allowTaskReparenting` 的值均为“`true`”，则如上所述，任何可以更改父项的 Activity 都将转移到与其有亲和关系的任务；其余 Activity 随即被移除。

`android:configChanges`

列出 Activity 将自行处理的配置更改。在运行时发生配置更改时，默认情况下会关闭 Activity 然后将其重新启动，但使用该属性声明配置将阻止 Activity 重新启动。 Activity 反而会保持运行状态，并且系统会调用其 `onConfigurationChanged()` 方法。

> **注**：应避免使用该属性，并且只应在万不得已的情况下使用。 如需了解有关如何正确处理配置更改所致重新启动的详细信息，请阅读处理运行时变更。

任何或所有下列字符串均是该属性的有效值。多个值使用“`|`”分隔 — 例如，“`locale|navigation|orientation`”。

值	说明
“`mcc`”	IMSI 移动国家/地区代码 (MCC) 发生了变化 - 检测到了 SIM 并更新了 MCC。
“`mnc`”	IMSI 移动网络代码 (MNC) 发生了变化 - 检测到了 SIM 并更新了 MNC。
“`locale`”	语言区域发生了变化 — 用户为文本选择了新的显示语言。
“`touchscreen`”	触摸屏发生了变化。（这种情况通常永远不会发生。）
“`keyboard`”	键盘类型发生了变化 — 例如，用户插入了一个外置键盘。
“`keyboardHidden`”	键盘无障碍功能发生了变化 — 例如，用户显示了硬件键盘。
“`navigation`”	导航类型（轨迹球/方向键）发生了变化。（这种情况通常永远不会发生。）
“`screenLayout`”	屏幕布局发生了变化 — 这可能是由激活了其他显示方式所致。
“`fontScale`”	字体缩放系数发生了变化 — 用户选择了新的全局字号。
“`uiMode`”	用户界面模式发生了变化 — 这可能是因用户将设备放入桌面/车载基座或夜间模式发生变化所致。 请参阅 `UiModeManager`。 *此项为 API 级别 8 中新增配置。*
“`orientation`”	屏幕方向发生了变化 — 用户旋转了设备。 **注**：如果您的应用面向 API 级别 13 或更高级别（按照 `minSdkVersion` 和 `targetSdkVersion` 属性所声明的级别），则还应声明 `"screenSize"` 配置，因为当设备在横向与纵向之间切换时，该配置也会发生变化。
“`screenSize`”	当前可用屏幕尺寸发生了变化。它表示当前可用尺寸相对于当前纵横比的变化，因此会在用户在横向与纵向之间切换时发生变化。 不过，如果您的应用面向 API 级别 12 或更低级别，则 Activity 始终会自行处理此配置变更（即便是在 Android 3.2 或更高版本的设备上运行，此配置变更也不会重新启动 Activity）。 *此项为 API 级别 13 中新增配置。*
“`smallestScreenSize`”	物理屏幕尺寸发生了变化。它表示与方向无关的尺寸变化，因此只有在实际物理屏幕尺寸发生变

	化（如切换到外部显示器）时才会变化。 对此配置的变更对应于smallestWidth 配置的变化。 不过，如果您的应用面向 API 级别 12 或更低级别，则 Activity 始终会自行处理此配置变更（即便是在 Android 3.2 或更高版本的设备上运行，此配置变更也不会重新启动 Activity）。 *此项为 API 级别 13 中新增配置。*
“layoutDirection”	布局方向发生了变化。例如，从从左至右 (LTR) 更改为从右至左 (RTL)。 *此项为 API 级别 17 中新增配置。*

所有这些配置变更都可能影响应用看到的资源值。 因此，调用 onConfigurationChanged() 时，通常有必要再次获取所有资源（包括视图布局、可绘制对象等），以正确处理变化。

android:documentLaunchMode

指定每次启动任务时应如何向其中添加新的 Activity 实例。 该属性允许用户让多个来自同一应用的文档出现在概览屏幕中。

该属性有四个值，会在用户使用该应用打开文档时产生以下效果：

值	说明
“intoExisting”	Activity 会为文档重复使用现有任务。使用该值与*不*设置 FLAG_ACTIVITY_MULTIPLE_TASK 标志、但设置 FLAG_ACTIVITY_NEW_DOCUMENT 标志所产生的效果相同，如使用 Intent 标志添加任务中所述。
“always”	Activity 为文档创建新任务，即便文档已打开也是如此。 这与同时设置 FLAG_ACTIVITY_NEW_DOCUMENT 和 FLAG_ACTIVITY_MULTIPLE_TASK 标志的效果相同。
“none”	该 Activity 不会为 Activity 创建新任务。这是默认值，它只会在设置了 FLAG_ACTIVITY_NEW_TASK 时创建新任务。 概览屏幕将按其默认方式对待此 Activity：为应用显示单个任务，该任务将从用户上次调用的任意 Activity 开始继续执行。
“never”	即使 Intent 包含 FLAG_ACTIVITY_NEW_DOCUMENT，该 Activity 也不会启动到新文档之中。 设置此值会替代 FLAG_ACTIVITY_NEW_DOCUMENT 和 FLAG_ACTIVITY_MULTIPLE_TASK 标志的行为（如果在 Activity 中设置了其中一个标志），并且概览屏幕将为应用显示单个任务，该任务将从用户上次调用的任意 Activity 开始继续执行。

> **注**：对于除“none”和“never”以外的值，必须使用 launchMode="standard" 定义 Activity。 如果未指定此属性，则使用 documentLaunchMode="none"。

android:enabled

系统是否可将 Activity 实例化 — "true" 表示可以，“false”表示不可以。 默认值为“true”。

<application> 元素具有自己的 enabled 属性，该属性适用于所有应用组件，包括 Activity。 <application> 和 <activity> 属性必须都是“true”（因为它们都默认使用该值），系统才能将 Activity 实例化。 如果任何一个属性是“false”，则无法进行实例化。

android:excludeFromRecents

是否应将该 Activity 启动的任务排除在最近使用的应用列表（即概览屏幕）之外。 也就是说，当该 Activity 是新任务的根 Activity 时，此属性确定任务是否应出现在最近使用的应用列表中。 如果应将任务*排除*在列表之外，请设置“true”；如果应将其*包括*在内，则设置“false”。 默认值为“false”。

android:exported

Activity 是否可由其他应用的组件启动 —“true”表示可以，“false”表示不可以。若为“false”，则 Activity 只能由同一应用的组件或使用同一用户 ID 的不同应用启动。

默认值取决于 Activity 是否包含 Intent 过滤器。没有任何过滤器意味着 Activity 只能通过指定其确切的类名称进行调用。 这意味着 Activity 专供应用内部使用（因为其他应用不知晓其类名称）。 因此，在这种情况下，默认值为“false”。另一方面，至少存在一个过滤器意味着 Activity 专供外部使用，因此默认值为“true”。

该属性并非限制 Activity 对其他应用开放度的唯一手段。 您还可以利用权限来限制哪些外部实体可以调用 Activity（请参阅 permission 属性）。

`android:finishOnTaskLaunch`

每当用户再次启动其任务（在主屏幕上选择任务）时，是否应关闭（完成）现有 Activity 实例 —“`true`”表示应关闭，“`false`”表示不应关闭。 默认值为“`false`”。

如果该属性和 `allowTaskReparenting` 均为“`true`”，则优先使用该属性。 Activity 的亲和关系会被忽略。 系统不是更改 Activity 的父项，而是将其销毁。

`android:hardwareAccelerated`

是否应为此 Activity 启用硬件加速渲染 —“`true`”表示应启用，“`false`”表示不应启用。 默认值为“`false`”。

从 Android 3.0 开始，为应用提供了硬件加速 OpenGL 渲染器，以改善许多常见 2D 图形运算的性能。 启用硬件加速渲染器时，Canvas、Paint、Xfermode、ColorFilter、Shader 和 Camera 中的大多数运算都会得到加速。这可以提高动画、滚动的流畅度和总体响应速度，即便是并不明确使用框架 OpenGL 库的应用也会受益。 由于启用硬件加速会增加资源消耗，因此您的应用将占用更多内存。

请注意，并非所有 OpenGL 2D 运算都会得到加速。如果您启用硬件加速渲染器，请对应用进行测试，以确保其在利用渲染器时不会出错。

`android:icon`

一个表示 Activity 的图标。该图标会在需要在屏幕上表示 Activity 时显示给用户。 例如，代表启动任务的 Activity 的图标显示在启动器窗口中。该图标通常附带标签（请参阅 `android:label` 属性）。

必须将该属性设置为对包含图像定义的可绘制资源的引用。 如果未设置该属性，则改为使用为应用整体指定的图标（请参阅 `<application>` 元素的 `icon` 属性）。

这个 Activity 的图标 — 无论设置于此处还是由 `<application>` 元素设置 — 同时也是 Activity 所有 Intent 过滤器的默认图标（请参阅 `<intent-filter>` 元素的 `icon` 属性）。

`android:label`

一种可由用户读取的 Activity 标签。该标签会在必须将 Activity 呈现给用户时显示在屏幕上。 它通常与 Activity 图标一并显示。

如果未设置该属性，则改为使用为应用整体设置的标签（请参阅 `<application>` 元素的 `label` 属性）。

这个 Activity 的标签 — 无论设置于此处还是由 `<application>` 元素设置 — 同时也是 Activity 所有 Intent 过滤器的默认标签（请参阅 `<intent-filter>` 元素的 `label` 属性）。

应将该标签设置为对字符串资源的引用，以便可以像用户界面中的其他字符串那样进行本地化。不过，为便于您开发应用，也可将其设置为原始字符串。

`android:launchMode`

有关应如何启动 Activity 的指令。共有四种模式与 `Intent` 对象中的 Activity 标志（`FLAG_ACTIVITY_*` 常量）协同工作，以确定在调用 Activity 处理 Intent 时应执行的操作。 这些模式是：

“`standard`”
“`singleTop`”
“`singleTask`”
“`singleInstance`”

默认模式是“`standard`”。

如下表所示，这些模式分为两大类，“`standard`”和“`singleTop`”Activity 为一类，“`singleTask`”和“`singleInstance`”为另一类。使用“`standard`”或“`singleTop`”启动模式的 Activity 可多次实例化。 实例可归属任何任务，并且可以位于 Activity 堆栈中的任何位置。 它们通常启动到名为 `startActivity()` 的任务之中（除非 Intent 对象包含 `FLAG_ACTIVITY_NEW_TASK` 指令，在此情况下会选择其他任务 — 请参阅 taskAffinity 属性）。

相比之下，“`singleTask`”和“`singleInstance`”Activity 只能启动任务。 它们始终位于 Activity 堆栈的根位置。此外，设备一次只能保留一个 Activity 实例 — 只允许一个此类任务。

“standard”和“singleTop”模式只在一个方面有差异： 每次“standard”Activity 有新的 Intent 时，系统都会创建新的类实例来响应该 Intent。每个实例处理单个 Intent。同理，也可创建新的“singleTop”Activity 实例来处理新的 Intent。 不过，如果目标任务在其堆栈顶部已有一个 Activity 实例，那么该实例将接收新 Intent（通过调用 onNewIntent()）；此时不会创建新实例。在其他情况下 — 例如，如果“singleTop”的一个现有实例虽在目标任务内，但未处于堆栈顶部，或者虽然位于堆栈顶部，但不在目标任务中 — 则系统会创建一个新实例并将其推送到堆栈上。

同理，如果您向上导航到当前堆栈上的某个 Activity，该行为由父 Activity 的启动模式决定。 如果父 Activity 有启动模式 singleTop（或 up Intent 包含 FLAG_ACTIVITY_CLEAR_TOP），则系统会将该父项置于堆栈顶部，并保留其状态。 导航 Intent 由父 Activity 的 onNewIntent() 方法接收。 如果父 Activity 有启动模式 standard（并且 up Intent 不包含 FLAG_ACTIVITY_CLEAR_TOP），则系统会将当前 Activity 及其父项同时弹出堆栈，并创建一个新的父 Activity 实例来接收导航 Intent。

“singleTask”和“singleInstance”模式同样只在一个方面有差异： “singleTask”Activity 允许其他 Activity 成为其任务的组成部分。 它始终位于其任务的根位置，但其他 Activity（必然是“standard”和“singleTop”Activity）可以启动到该任务中。 相反，“singleInstance”Activity 则不允许其他 Activity 成为其任务的组成部分。它是任务中唯一的 Activity。 如果它启动另一个 Activity，系统会将该 Activity 分配给其他任务 — 就好像 Intent 中包含 FLAG_ACTIVITY_NEW_TASK 一样。

用例	启动模式	多个实例？	注释
大多数 Activity 的正常启动	“standard”	是	默认值。系统始终会在目标任务中创建新的 Activity 实例并向其传送 Intent。
	“singleTop”	有条件	如果目标任务的顶部已存在一个 Activity 实例，则系统会通过调用该实例的 onNewIntent() 方法向其传送 Intent，而不是创建新的 Activity 实例。
专用启动 *（不建议用作常规用途）*	“singleTask”	否	系统在新任务的根位置创建 Activity 并向其传送 Intent。 不过，如果已存在一个 Activity 实例，则系统会通过调用该实例的 onNewIntent() 方法向其传送 Intent，而不是创建新的 Activity 实例。
	“singleInstance”	否	与“singleTask""相同，只是系统不会将任何其他 Activity 启动到包含实例的任务中。 该 Activity 始终是其任务唯一仅有的成员。

如上表所示，standard 是默认模式，并且适用于大多数的 Activity 类型。对许多类型的 Activity 而言，SingleTop 也是一个常见并且有用的启动模式。 其他模式 — singleTask 和 singleInstance - 不适合 大多数应用因为它们所形成的交互模式可能让用户感到陌生，并且与大多数其他应用迥异。

无论您选择哪一种启动模式，请务必在启动期间以及使用*返回*按钮从其他 Activity 和任务返回该 Activity 时对其进行易用性测试。

如需了解有关启动模式及其与 Intent 标志交互的详细信息，请参阅任务和返回栈文档。

android:maxRecents

概览屏幕中位于此 Activity 根位置的任务数上限。 达到该条目数时，系统会从概览屏幕中移除最近最少使用的实例。 有效值为 1-50（低内存设备使用 25）；0 为无效值。 该值必须是整数，例如 50。默认值为 16。

android:multiprocess

是否可以将 Activity 实例启动到启动该实例的组件进程内 —“true”表示可以，“false”表示不可以。默认值为“false”。

正常情况下，新的 Activity 实例会启动到定义它的应用进程内，因此所有 Activity 实例都在同一进程内运行。 不过，如果该标志设置为“true”，Activity 实例便可在多个进程内运行，这样系统就能在任何使用实例的地方创建实例（前提是权限允许这样做），但这几乎毫无必要性或可取之处。

android:name

实现 Activity 的类的名称，是 Activity 的子类。 该属性值应为完全限定类名称（例如，“com.example.project.ExtracurricularActivity”）。不过，为了简便起见，如果名称的第一个字符是句点（例

如，“.ExtracurricularActivity”），则名称将追加到 <manifest> 元素中指定的软件包名称。

应用一旦发布，即不应更改该名称（除非您设置了 android:exported="false"）。

没有默认值。必须指定该名称。

android:noHistory

当用户离开 Activity 并且其在屏幕上不再可见时，是否应从 Activity 堆栈中将其移除并完成（调用其 finish() 方法）—“true”表示应将其完成，“false”表示不应将其完成。 默认值为“false”。

“true”一值表示 Activity 不会留下历史轨迹。 它不会留在任务的 Activity 堆栈内，因此用户将无法返回 Activity。 在此情况下，如果您启动另一个 Activity 来获取该 Activity 的结果，系统永远不会调用 onActivityResult()。

该属性是在 API 级别 3 引入的。

android:parentActivityName

Activity 逻辑父项的类名称。此处的名称必须与为相应 <activity> 元素的 android:name 属性指定的类名称一致。

系统会读取该属性，以确定当用户按下操作栏中的“向上”按钮时应该启动哪一个 Activity。 系统还可以利用这些信息通过 TaskStackBuilder 合成 Activity 的返回栈。

要支持 API 级别 4 - 16，您还可以使用为 "android.support.PARENT_ACTIVITY" 指定值的 <meta-data> 元素来声明父 Activity。例如：

```
<activity
    android:name="com.example.app.ChildActivity"
    android:label="@string/title_child_activity"
    android:parentActivityName="com.example.app.MainActivity" >
    <!-- Parent activity meta-data to support API level 4+ -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.example.app.MainActivity" />
</activity>
```

如需了解有关声明父 Activity 以支持向上导航的详细信息，请阅读提供向上导航。

该属性是在 API 级别 16 引入的。

android:permission

客户端启动 Activity 或以其他方式令其响应 Intent 而必须具备的权限的名称。 如果系统尚未向 startActivity() 或 startActivityForResult() 的调用方授予指定权限，其 Intent 将不会传递给 Activity。

如果未设置该属性，则对 Activity 应用 <application> 元素的 permission 属性设置的权限。 如果这两个属性均未设置，则 Activity 不受权限保护。

如需了解有关权限的详细信息，请参阅简介的权限一节和另一份文档安全与权限。

android:process

应在其中运行 Activity 的进程的名称。正常情况下，应用的所有组件都在为应用创建的默认进程名称内运行，您无需使用该属性。但在必要时，您可以使用该属性替换默认进程名称，以便让应用组件散布到多个进程中。

如果为该属性分配的名称以冒号（“:”）开头，则会在需要时创建应用专用的新进程，并且 Activity 会在该进程中运行。如果进程名称以小写字符开头，Activity 将在该名称的全局进程中运行，前提是它拥有相应的权限。这可以让不同应用中的组件共享一个进程，从而减少资源占用。

<application> 元素的 process 属性可为所有组件设置一个不同的默认进程名称。

android:relinquishTaskIdentity

Activity 是否将其任务标识符交给任务栈中在其之上的 Activity。 如果任务根 Activity 的该属性设置为“true”，则任务会用其内的下一个 Activity 的 Intent 替换基本 Intent。 如果下一个 Activity 的该属性也设置为“true”，则该 Activity 会将基本 Intent 给予其在同

一任务中启动的任何 Activity。 系统继续为每个 Activity 执行此过程，直至遇到的某个 Activity 将该属性设置为“false”为止。 默认值为“false”。

如果该属性设置为“true”，则 Activity 还可利用 ActivityManager.TaskDescription 来更改概览屏幕中的标签、颜色和图标。

`resizeableActivity`

指定应用是否支持多窗口显示。您可以在 <activity> 或 <application> 元素中设置该属性。

如果您将该属性设置为 true，则用户可以分屏和自由形状模式启动 Activity。 如果您将该属性设置为 false，Activity 将不支持多窗口模式。 如果该值为 false，且用户尝试在多窗口模式下启动 Activity，该 Activity 将全屏显示。

如果您的应用面向 API 级别 24 或更高级别，但未对该属性指定值，则该属性的值默认设为 true。

该属性是在 API 级别 24 添加的。

`android:screenOrientation`

Activity 在设备上的显示方向。如果 Activity 是在多窗口模式下运行，系统会忽略该属性。

其值可以是下列任一字符串：

“unspecified”	默认值。由系统选择方向。在不同设备上，系统使用的政策以及基于政策在特定上下文所做的选择可能有所差异。
“behind”	与 Activity 栈中紧接着它的 Activity 的方向相同。
“landscape”	横向方向（显示的宽度大于高度）。
“portrait”	纵向方向（显示的高度大于宽度）。
“reverseLandscape”	与正常横向方向相反的横向方向。*API 级别 9 中的新增配置。*
“reversePortrait”	与正常纵向方向相反的纵向方向。*API 级别 9 中的新增配置。*
“sensorLandscape”	横向方向，但根据设备传感器，可以是正常或反向的横向方向。*API 级别 9 中的新增配置。*
“sensorPortrait”	纵向方向，但根据设备传感器，可以是正常或反向的纵向方向。*API 级别 9 中的新增配置。*
“userLandscape”	横向方向，但根据设备传感器和用户的传感器首选项，可以是正常或反向的横向方向。 如果用户锁定了基于传感器的旋转，其行为与 landscape 相同，否则，其行为与 sensorLandscape 相同。*API 级别 18 中的新增配置。*
“userPortrait”	纵向方向，但根据设备传感器和用户的传感器首选项，可以是正常或反向的纵向方向。 如果用户锁定了基于传感器的旋转，其行为与 portrait 相同，否则，其行为与 sensorPortrait 相同。*API 级别 18 中的新增配置。*
“sensor”	方向由设备方向传感器决定。显示方向取决于用户如何手持设备，它会在用户旋转设备时发生变化。 但一些设备默认情况下不会旋转到所有四种可能的方向。要允许全部四种方向，请使用 "fullSensor"。
“fullSensor”	方向由 4 种方向中任一方向的设备方向传感器决定。这与 "sensor" 类似，不同的是它允许所有 4 种可能的屏幕方向，无论设备正常情况下采用什么方向（例如，一些设备正常情况下不使用反向纵向或反向横向，但它支持这些方向）。 *API 级别 9 中的新增配置。*
“nosensor”	决定方向时不考虑物理方向传感器。传感器会被忽略，因此显示不会随用户对设备的移动而旋转。除了这个区别，系统在选择方向时使用的政策与“unspecified”设置相同。
“user”	用户当前的首选方向。
"fullUser"	如果用户锁定了基于传感器的旋转，其行为与 user 相同，否则，其行为与 fullSensor 相同，允许所有 4 种可能的屏幕方向。 *API 级别 18 中的新增配置。*
“locked”	将方向锁定在其当前的任意旋转方向。*API 级别 18 中的新增配置。*

> **注**：如果您声明其中一个横向或纵向值，系统将其视为对 Activity 运行方向的硬性要求。 因此，您声明的值支持通过 Google Play 之类的服务进行过滤，这样就能将您的应用只提供给支持 Activity 所要求方向的设备。 例如，如果您声明了 "landscape"、"reverseLandscape" 或 "sensorLandscape"，则您的应用将只提供给支持横向方向的设备。 不过，您还

> 应通过 <uses-feature> 元素明确声明，您的应用要求采用纵向或横向方向。 例如，<uses-feature android:name="android.hardware.screen.portrait"/>。这纯粹是 Google Play（以及其他支持它的服务）提供的一种过滤行为，平台本身并不能控制当设备仅支持特定方向时您的应用能否安装。

`android:stateNotNeeded`

能否在不保存 Activity 状态的情况下将其终止并成功重新启动 —“true”表示可在不考虑其之前状态的情况下重新启动，“false”表示需要之前状态。 默认值为“false”。

正常情况下，为保存资源而暂时关闭 Activity 前，系统会调用其 onSaveInstanceState() 方法。 该方法将 Activity 的当前状态存储在一个 Bundle 对象中，然后在 Activity 重新启动时将其传递给 onCreate()。 如果该属性设置为“true”，系统可能不会调用 onSaveInstanceState()，并且会向 onCreate() 传递 null 而不是 Bundle - 这与它在 Activity 首次启动时完全一样。

“true”设置可确保 Activity 能够在未保留状态时重新启动。 例如，显示主屏幕的 Activity 可以使用该设置来确保其由于某种原因崩溃时不会被移除。

`supportsPictureInPicture`

指定 Activity 是否支持画中画 显示。如果 android:resizeableActivity 是 false，系统会忽略该属性。

该属性是在 API 级别 24 添加的。

`android:taskAffinity`

与 Activity 有着亲和关系的任务。从概念上讲，具有相同亲和关系的 Activity 归属同一任务（从用户的角度来看，则是归属同一“应用”）。 任务的亲和关系由其根 Activity 的亲和关系确定。

亲和关系确定两件事 - Activity 更改到的父项任务（请参阅 allowTaskReparenting 属性）和通过 FLAG_ACTIVITY_NEW_TASK 标志启动 Activity 时将用来容纳它的任务。

默认情况下，应用中的所有 Activity 都具有相同的亲和关系。您可以设置该属性来以不同方式组合它们，甚至可以将在不同应用中定义的 Activity 置于同一任务内。 要指定 Activity 与任何任务均无亲和关系，请将其设置为空字符串。

如果未设置该属性，则 Activity 继承为应用设置的亲和关系（请参阅 <application> 元素的 taskAffinity 属性）。 应用默认亲和关系的名称是 <manifest> 元素设置的软件包名称。

`android:theme`

对定义 Activity 总体主题的样式资源的引用。它会自动将 Activity 的上下文设置为使用该主题（请参阅 setTheme()），它还可以引发 Activity 启动前的“启动”动画（以更加符合 Activity 的实际外观）。

如果未设置该属性，则 Activity 继承通过 <application> 元素的 theme 属性为应用整体设置的主题。 如果该属性也未设置，则使用默认系统主题。如需了解详细信息，请参阅样式和主题开发者指南。

`android:uiOptions`

针对 Activity UI 的附加选项。

必须是下列值之一。

值	说明
"none"	无附加 UI 选项。这是默认值。
"splitActionBarWhenNarrow"	当水平空间受限时（例如在手持设备上的纵向模式下时）在屏幕底部添加一个栏以显示*应用栏*（也称为*操作栏*）中的操作项）。 应用栏不是以少量操作项形式出现在屏幕顶部的应用栏中，而是分成了顶部导航区和底部操作项栏。 这可以确保操作项以及顶部的导航和标题元素都能获得合理的空间。 菜单项不会拆分到两个栏中，它们始终一起出现。

如需了解有关应用栏的详细信息，请参阅添加应用栏培训课。

该属性是在 API 级别 14 添加的。

`android:windowSoftInputMode`

Activity 的主窗口与包含屏幕软键盘的窗口的交互方式。 该属性的设置影响两个方面：

- 当 Activity 成为用户注意的焦点时软键盘的状态 — 隐藏还是可见。
- 对 Activity 主窗口所做的调整 — 是否将其尺寸调小以为软键盘腾出空间，或者当窗口部分被软键盘遮挡时是否平移其内容以使当前焦点可见。

该设置必须是下表所列的值之一，或者是一个“`state...`”值加上一个“`adjust...`”值的组合。 在任一组中设置多个值（例如，多个“`state...`”值）都会产生未定义结果。各值之间使用垂直条 (`|`) 分隔。 例如：

```
<activity android:windowSoftInputMode="stateVisible|adjustResize" . . . >
```

此处设置的值（“`stateUnspecified`”和“`adjustUnspecified`”除外）替换主题中设置的值。

值	说明
"`stateUnspecified`"	不指定软键盘的状态（隐藏还是可见）。 将由系统选择合适的状态，或依赖主题中的设置。 这是对软键盘行为的默认设置。
“`stateUnchanged`”	当 Activity 转至前台时保留软键盘最后所处的任何状态，无论是可见还是隐藏。
“`stateHidden`”	当用户选择 Activity 时 — 也就是说，当用户确实是向前导航到 Activity，而不是因离开另一 Activity 而返回时 — 隐藏软键盘。
“`stateAlwaysHidden`”	当 Activity 的主窗口有输入焦点时始终隐藏软键盘。
“`stateVisible`”	在正常的适宜情况下（当用户向前导航到 Activity 的主窗口时）显示软键盘。
“`stateAlwaysVisible`”	当用户选择 Activity 时 — 也就是说，当用户确实是向前导航到 Activity，而不是因离开另一 Activity 而返回时 — 显示软键盘。
“`adjustUnspecified`”	不指定 Activity 的主窗口是否调整尺寸以为软键盘腾出空间，或者窗口内容是否进行平移以在屏幕上显露当前焦点。 系统会根据窗口的内容是否存在任何可滚动其内容的布局视图来自动选择其中一种模式。 如果存在这样的视图，窗口将进行尺寸调整，前提是可通过滚动在较小区域内看到窗口的所有内容。 这是对主窗口行为的默认设置。
“`adjustResize`”	始终调整 Activity 主窗口的尺寸来为屏幕上的软键盘腾出空间。
“`adjustPan`”	不调整 Activity 主窗口的尺寸来为软键盘腾出空间， 而是自动平移窗口的内容，使当前焦点永远不被键盘遮盖，让用户始终都能看到其输入的内容。 这通常不如尺寸调正可取，因为用户可能需要关闭软键盘以到达被遮盖的窗口部分或与这些部分进行交互。

该属性是在 API 级别 3 引入的。

引入的版本：

API 级别 1，为 `noHistory` 和 `windowSoftInputMode` 之外的所有属性引入，这两个属性则是在 API 级别 3 中增加。

另请参阅：

`<application>`
`<activity-alias>`