任务与返回堆栈

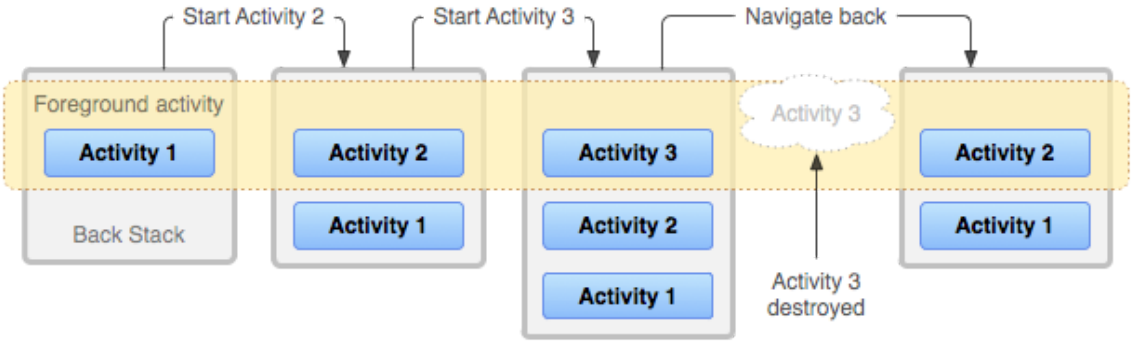
原创 xyzso1z 2020-07-12 14:18:40 30 收藏 编辑 版权
分类专栏： Android

任务

任务是用户在执行某项工作时与之互动的一系列 `Activity` 的集合。这些 `Activity` 按照每个 `Activity` 打开顺序排列在一个返回堆栈中。

大多数任务都从设备主屏幕上启动。当用户轻触应用启动器中的图标(或主屏幕上的快捷方式)时，该应用的任务就会转到前台运行，如果该应用没有任务存在(应该最近没有使用过)，则会创建一个新的任务，并且该应用的“主” `Activity` 将会作为堆栈的根 `Activity` 打开。

在当前 `Activity` 启动另一个 `Activity` 时，新的 `Activity` 将被推送到堆栈顶部并获得焦点。上一个 `Activity` 仍保留在堆栈中，但会停止。当 `Activity` 停止时，系统会保留其界面的当前状态。当用户按返回按钮时，当前 `Activity` 会从堆栈顶部退出（该 `Activity` 销毁），上一个 `Activity` 会恢复（界面会恢复到上一个状态）。堆栈中的 `Activity` 永远不会重新排列，只会被送入和退出，在当前 `Activity` 启动时被送入堆栈，在用户使用返回按钮离开时从堆栈中退出。因此，返回堆栈按照“后进先出”的对象结构运作。如下图 借助一个时间轴直观地显示了这种行为。该时间轴显示了 `Activity` 之间的进展以及每个时间点的当前返回堆栈。



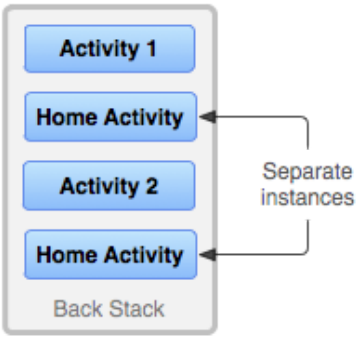
有关任务中的每个新 `Activity` 如何添加到返回堆栈的图示。当用户按返回按钮时，当前 `Activity` 会销毁，上一个 `Activity` 将恢复。

如果用户继续按**返回**，则堆栈中的 `Activity` 会逐个退出，以显示前一个`Activity` ，直到用户返回到主屏幕（或任务开始时运行的 `Activity` ）。移除堆栈中的所有 `Activity` 后，该任务将不复存在。

任务是一个整体单元，当用户开始一个新任务或通过Home按钮进入主屏幕时，任务可移至“后台”。**在后台时，任务重所有Activity 都会停止，但任务的返回堆栈会保持不变**，当其他任务启动时，当前任务只是失去了焦点。

> 注意：**多个任务可以同时在后台进行。但是，如果用户同时运行很多后台任务，系统可能会为了恢复内存而开始销毁后台 `Activity` ,导致 `Activity` 状态丢失。**
> 由于返回堆栈中的 `Activity` 不会重新排序，如果应用允许用户从多个 `Activity` 启动特定的 `Activity` ，系

> 统便会创建该 `Activity` 的新实例并将其推送到堆栈中(而不是将该 `Activity` 的某个先前的实例移至顶部)。这样一来，应用中的一个 `Activity` 就可能被多次实例化,如下图：



因此，如果用户使用使用返回按钮向后导航，`Activity` 的每个实例将按照它们被打开的顺序显示出来。不过如果不希望某个 `Activity` 被实例化多次，可以修该此行为。

`Activity` 和任务的默认行为总结如下：

- 当 `Activity A` 启动 `Activity B` 时，`Activity A` 会停止，但系统会保留其状态(例如滚动位置和输入到表单中的文本)。如果用户在 `Activity B` 中按返回按钮，系统会恢复 `Activity A` 及其状态。
- 当用户通过按Home键离开任务时，当前 `Activity` 会停止，其任务会转到后台，系统会保留任务中每个 `Actvitity` 的状态。如果用户稍后通过点该任务的启动图标来回复该该任务，该任务会进入前台并恢复堆栈顶部的 `Activity` 。
- 如果用户按返回按钮，当前 `Activity` 将从堆栈中退出并销毁。堆栈中的上一个 `Activity` 将恢复。`Activity` 被销毁后，系统不会保留该 `Activity` 的状态。
- `Activity` 可以多次实例化，甚至是从其他任务对其进行实例化。

管理任务

如果我们希望应用中的某个 `Activity` 在启动是开启一个新的任务(而不是被放入当前的任务中)或者当启动某个 `Activity` 时，您希望调用它的一个现有实例(而不是在返回堆栈顶部创建一个新实例)，或者希望在用户离开任务时清除返回堆栈中除根 `Acivity` 以外的所有 `Activity`。
我们可以借助 `<activity>` 清单元素中的属性以及传递给 `startActivity()` 的 `intent` 中的标记来实现上述目的。
在这方面，可以使用的主要 `<activity>` 属性包括：

- `taskAffinity`
- `launchMode`
- `allTaskReparenting`
- `clearTaskOnLaunch`
- `alwaysRetainTaskState`
- `finishOnTaskLaunch`

可以使用的主要 `intent` 标记包括：

- `FLAG_ACTIVITY_NEW_TASK`
- `FLAG_ACTIVITY_CLEAR_TOP`
- `FLAG_ACTIVITY_SINGLE_TOP`

定义启动模式

可以通过启动模式定义 `Activity` 的新实例如何与当前任务关联。有两种方式定义不同的启动模式：

- 使用清单文件：在清单文件中声明 `Activity` 时，可以指定该 `Activity` 在启动时如何与任务关联。
- 使用 `Intent` 标记：当调用 `startActivity()` 时，可以在 `Intent` 中添加一个标记，用于声明新 `Activity` 如何与当前任务相关联。
  因此，如果 `Activity A` 启动 `Activity B` ，`Activity B` 可在清单中定义如何与当前任务相关，`Activity A` 也可以请求 `Activity B` 应该如何与当前任务关联。如果两个 `Activity` 都定义了 `Activity B` 应如何与任务关联，将优先遵循 `Activity A` 的请求（**在 `intent` 中定义**），而不是 `Activity B` 的请求（**在清单中定义**）

使用清单文件

在清单文件中声明 `Activity` 时，可以使用 `<activity>` 元素的 `launchMode` 属性指定 `Activity` 应该如何与任务关联。
`launchMode` 属性说明了 `Activity` 应如何启动到任务中。有4种不同的启动模式：

1. `standard` (默认模式)
   默认值。系统在启动该Activity的任务中创建Activity的新实例，并将intent 传送给该实例。Activity可以多次实例化，每个实例可以属于不同的任务，一个任务可以拥有多个实例。
2. `singleTop`
   如果当前任务的**顶部**已存在 `Activity` 的实例，则系统会通过调用其 `onNewIntent()` 方法来将 `intent` 转送给该实例，而不是创建 `Activity` 的新实例。

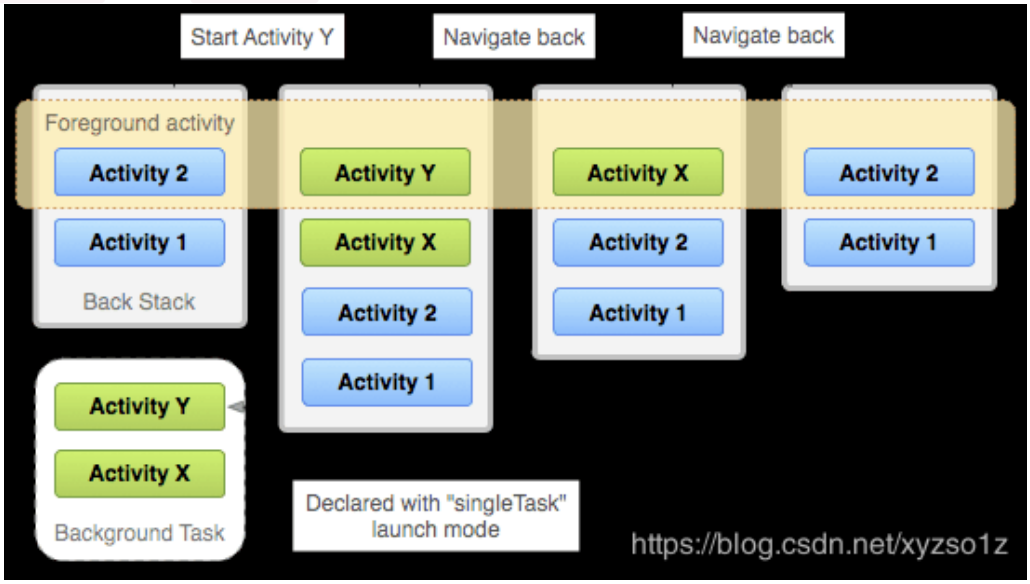
   > 创建 `Activity` 的实例后，用户可以按返回按钮返回到上一个 `Activity` 。但是，当由 `Activity` 的现有实例处理新 `intent` 时，用户将无法通过按返回按钮返回到 `onNewIntent()` 收到新 `intent` 之前的 `Activity` 状态。

3. `singleTask`
   系统会创建**新任务**，并实例化新任务的根 `Activity` 。但是，如果另外的任务中已存在该 `Activity` 的实例，则系统会通过调用其 `onNewIntent()` 方法将 `intent` 转送到该现有实例，而不是创建新实例。 `Activity` 一次只能有一个实例存在。

   > 虽然 `Activity` 在新任务中启动，但用户按返回按钮仍会返回到上一个 `Activity` 。

4. `singleInstance`
   与 `singleTask` 相似，唯一不同的是系统不会将任何其他 `Activity` 启动到包含该实例的任务中。该 `Activity` 始终是其任务唯一的成员；由该 `Activity` 启动的任何 `Activity` 都会在其他的任务中打开。

`singleTask` 使用举例：
无论 `Activity` 是新任务中启动的，还是在和启动它的 `Activity` 相同的任务中启动，用户按返回按钮都会回到上一个 `Activity` 。但是，如果启动了指定 `singleTas` 启动模式的 `Activity` ,而后台任务中已存在该 `Activity` 的实例，则**系统会将该后台任务整个转到前台运行。此时，返回堆栈包含了转到前台的任务中的所有 `Activity` ,这些 `Activity` 都位于堆栈的顶部。**如图：



使用 `Intent` 标记

启动 `Activity` 时，可以在传递给 `startActivity()` 的 `intent` 中添加相应的标记来修改 `Activity` 与其任务的默认关联。可以使用以下标记来修改默认行为：

1. `FLAG_ACTIVITY_NEW_TASK`
   在新任务中启动 `Activity` 。如果现在启动的 `Activity` 已经有任务在运行，则系统会将该任务转到前台并恢复期最后的状态，而 `Activity` 将在 `onNewIntent()` 中收到新的 `intent` 。与上面讲解的 `singleTask` 行为相同。
2. `FLAG_ACTIVITY_SINGLE_TOP`
   如果要启动的 `Activity` 是当前 `Activity` (即位于返回堆栈顶部的 `Activity`)，则现有实例会收到对 `onNewIntent()` 的调用，而不会创建 `Activity` 的新实例。与上面讲解的 `singleTop` **行为相同。**
3. `FLAG_ACTIVITY_CLEAR_TOP`
   如果要启动的 `Activity` 已经在当前任务中运行，则不会启动该 `Activity` 的新实例，而是会销毁位于它之上的所有其他 `Activity` ，并通过 `onNewIntent()` 将此 `intent` 传送给它的已恢复实例。 `launchMode` **属性没有可产生此行为的值。**

`FLAG_ACTIVITY_CLEAR_TOP` 最常与 `FLAG_ACTIVITY_NEW_TASK` 结合使用。将这两个标记结合使用，可以查找其他任务中的现有 `Activity` ，并将其置于能够响应 `intent` 的位置。

处理亲和性(`Handling affinities`)

“亲和性” 表示 `Activity` 倾向于属于哪个任务。默认情况下，同一应用中的所有Activity彼此具有亲和性。因此，在默认情况下，同一应用中的所有 `Activity` 都倾向于位于同一任务。不过，也可以修改 `Activity` 的默认亲和性。在不同应用中定义的 `Activity` 可以具有相同的亲和性，或者在同一应用中定义的 `Activity` 也可以被指定不同的任务亲和性。

可以使用 `<activity>` 元素的 `taskAffinity` 属性修改任何给定 `Activity` 的亲和性。
`taskAffinity` 属性采用字符串值，该值必须不同于 `<manifest>` 元素中声明的默认软件包名称，因为系统使用该名称来标识应用的默认任务亲和性。
亲和性可在两种情况下发挥作用：

1. 启动 `Activity` 的 `intent` 包含 `FLAG_ACTIVITY_NEW_TASK` 标记时。
   默认情况下，新 `Activity` 会启动到调用 `startActivity()` 的 `Activity` 的任务中。它会被推送到调用方 `Activity` 所在的返回堆栈中。但是，如果传递给 `startActivity()` 的 `intent` 包含 `FLAG_ACTIVITY_NEW_TASK` 标记，则系统会寻找其他任务来容纳新 `Activity`。通常会是一个新任务，但也可能不是。如果已存在与新 `Activity` 具有相同亲和性的现有任务，则会将 `Activity` 启动到该任务中。如果不存在，则会启动一个新任务。
2. 当 `Activity` 的 `allowTaskReparenting` 属性设为 `true` 时。
   在这种情况下，一旦和 `Activity` 有亲和性的任务进入前台运行，`Activity` 就可以从其启动的任务转移到该任务。

清除返回堆栈

如果用户离开任务较长时间，系统会清除任务中除根 `Activity` 以外的所有 `Activity`。当用户再次返回到该任务时，只有根 `Activity` 会恢复。系统之所以采取这种行为是因为，经过一段时间后，用户可能已经放弃了之前的操作，现在返回任务是为了开始某项新的操作。
我们可以使用一些 `Activity` 属性来修改此行为：

- `alwaysRetainTaskState`
  如果在任务的根 `Activity` 中将该属性设为 `true`，则不会发生上述默认行为。即使经过很长一段时间后，任务仍会在其堆栈中保留所有 `Activity`。
- `clearTaskOnLaunch`
  如果在任务的根 `Activity` 中将该属性设为 `true`，那么只要用户离开任务再返回堆栈就会被清除到只剩根 `Activity`。也就是说，它与 `alwaysRetainTaskState` 正好相反。用户始终会返回到任务的初始状态，即便只是短暂离开任务也是如此。
- `finishOnTaskLaunch`
  该属性与 `clearTaskOnLaunch` 类似，但它**只会作用于单个 `Activity` 而非整个任务**。它还可导致任何 `Activity` 消失，包括根 `Activity`。如果将该属性设为 `true`，则 `Activity` 仅在当前会话中归属于任务。如果用户离开任务再返回，则该任务将不再存在。