输入事件

本文内容

在 Android 系统中，从用户与应用的交互中截获事件的方法不止一种。如考虑截获用户界面内的事件，则可从用户与之交互的特定视图对象中捕获事件。 为此，View 类提供了多种方法。

在您将用于构建布局的各种 View 类中，您可能会注意到几种看起来适用于 UI 事件的公共回调方法。 当该对象上发生相应的操作时，Android 框架会调用这些方法。 例如，在触摸一个视图对象（例如“按钮”）时，对该对象调用 `onTouchEvent()` 方法。不过，为了截获此事件，您必须扩展 View 类并重写该方法。 然而，为了处理此类事件而扩展每个视图对象并不现实。 正因如此，View 类还包含一系列嵌套接口以及您可以更加轻松定义的回调。 这些接口称为事件侦听器，是您捕获用户与 UI 之间交互的票证。

尽管您通常会使用事件侦听器来侦听用户交互，但有时您确实需要扩展 View 类以构建自定义组件。也许，您想扩展 `Button` 类来丰富某些内容的样式。 在这种情况下，您将能够使用该类的事件处理程序为类定义默认事件行为。

事件侦听器

事件侦听器是 `View` 类中包含一个回调方法的接口。 当用户与 UI 项目之间的交互触发已注册此视图的侦听器时，Android 框架将调用这些方法。

各事件侦听器接口包含的回调方法如下：

`onClick()`

> 在 `View.OnClickListener` 中。 当用户触摸项目（处于触摸模式下）时，或者使用导航键或轨迹球聚焦于项目，然后按适用的“Enter”键或按下轨迹球时，将调用此方法。

`onLongClick()`

> 在 `View.OnLongClickListener` 中。 当用户触摸并按住项目（处于触摸模式下）时，或者使用导航键或轨迹球聚焦于项目，然后按住适用的“Enter”键或按住轨迹球（持续一秒钟）时，将调用此方法。

`onFocusChange()`

> 在 `View.OnFocusChangeListener` 中。 当用户使用导航键或轨迹球导航到或远离项目时，将调用此方法。

`onKey()`

> 在 `View.OnKeyListener` 中。 当用户聚焦于项目并按下或释放设备上的硬按键时，将调用此方法。

`onTouch()`

> 在 `View.OnTouchListener` 中。 当用户执行可视为触摸事件的操作时，其中包括按下、释放或屏幕上的任何移动手势（在项目边界内），将调用此方法。

`onCreateContextMenu()`

在 `View.OnCreateContextMenuListener` 中。 当（因持续“长按”而）生成上下文菜单时，将调用此方法。请参见菜单开发者指南中有关上下文菜单的阐述。

这些方法是其相应接口的唯一成员。要定义其中一个方法并处理事件，请在 Activity 中实现嵌套接口或将其定义为匿名类。然后，将实现的实例传递给相应的 `View.set...Listener()` 方法。 （例如，调用 `setOnClickListener()` 并向其传递 `OnClickListener` 实现。）

以下示例显示了如何为按钮注册点击侦听器。

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
      // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedValues) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
    ...
}
```

您可能还会发现，将 OnClickListener 作为 Activity 的一部分来实现更为方便。这样可以避免加载额外的类和分配对象。例如：

```
public class ExampleActivity extends Activity implements OnClickListener {
    protected void onCreate(Bundle savedValues) {
        ...
        Button button = (Button)findViewById(R.id.corky);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
      // do something when the button is clicked
    }
    ...
}
```

请注意，上述示例中的 `onClick()` 回调没有返回值，但是其他某些事件侦听器方法必须返回布尔值。具体原因取决于事件。 对于这几个事件侦听器，必须返回布尔值的原因如下：

- `onLongClick()`：此方法返回一个布尔值，表示您是否已处理完事件，以及是否应该将它继续传下去。 也就是说，返回 *true* 表示您已经处理事件且事件应就此停止；如果您尚未处理事件和/或事件应该继续传递给其他任何点击侦听器，则返回 *false*。
- `onKey()`：此方法返回一个布尔值，表示您是否已处理完事件，以及是否应该将它继续传下去。 也就是说，返回 *true* 表示您已经处理事件且事件应就此停止；如果您尚未处理事件和/或事件应该继续传递给其他任何按键侦听器，则返回 *false*。
- `onTouch()`： 此方法返回一个布尔值，表示侦听器是否处理完此事件。重要的是，此事件可以拥有多个分先后顺序的操作。 因此，如果在收到关闭操作事件时返回 *false*，则表示您并未处理完此事件，而且对其后续操作也不感兴趣。 因此，您无需执行事件内的任何其他操作，如手势或最终操作事件。

请记住，硬按键事件总是传递给目前处于焦点的视图对象。它们从视图 层次结构的顶部开始分派，然后向下，直至到达合适的目的地。如果您的视图对象（或视图对象的子项）目前具有焦点，那么您可以看到事件经由 `dispatchKeyEvent()` 方法的分派过程。除了通过视图捕获按键事件，您还可以使用 `onKeyDown()` 和 `onKeyUp()` 接收 Activity 内部的所有事件。

此外，考虑应用的文本输入时，请记住：许多设备只有软件输入法。 此类方法无需基于按键；某些可能使用语音输入、手写等。尽管输入法提供了类似键盘的界面，但它通常**不会**触发 `onKeyDown()` 系列的事件。除非您要将应用限制为带有硬键盘的设备，否则，在设计 UI 时切勿要求必须通过特定按键进行控制。 特别是，当用户按下返回键时，不要依赖这些方法验证输入；请改用 `IME_ACTION_DONE` 等操作让输入法知晓您的应用预计会作何反应，这样，可以通过一种有意义的方式更改其 UI。不要推断软件输入法应如何工作，只要相信它能够为应用提供已设置格式的文本即可。

> **注：**Android 会先调用事件处理程序，然后从类定义调用合适的默认处理程序。 因此，从这些事件侦听器返回 *true* 会停止将事件传播到其

> 他事件侦听器，还会阻止回调视图对象中的默认事件处理程序。 因此，在返回 *true* 时请确保您要终止事件。

事件处理程序

如果您从视图构建自定义组件，则将能够定义几种用作默认事件处理程序的回调方法。在有关自定义组件的文档中，您将了解某些用于事件处理的常见回调，其中包括：

- `onKeyDown(int, KeyEvent)`：在发生新的按键事件时调用
- `onKeyUp(int, KeyEvent)`：在发生按键弹起事件时调用
- `onTrackballEvent(MotionEvent)`：在发生轨迹球运动事件时调用
- `onTouchEvent(MotionEvent)`：在发生触摸屏运动事件时调用
- `onFocusChanged(boolean, int, Rect)`：在视图获得或失去焦点时调用

还有一些其他方法值得您注意，尽管它们并非 View 类的一部分，但可能会直接影响所能采取的事件处理方式。 因此，在管理布局内更复杂的事件时，请考虑使用以下其他方法：

- `Activity.dispatchTouchEvent(MotionEvent)`：此方法允许 `Activity` 在分派给窗口之前截获所有触摸事件。
- `ViewGroup.onInterceptTouchEvent(MotionEvent)`：此方法允许 `ViewGroup` 监视分派给子视图的事件。
- `ViewParent.requestDisallowInterceptTouchEvent(boolean)`： 对父视图调用此方法表明不应使用 `onInterceptTouchEvent(MotionEvent)` 截获触摸事件。

触摸模式

当用户使用方向键或轨迹球导航用户界面时，必须聚焦到可操作项目上（如按钮），以便用户看到将接受输入的对象。 但是，如果设备具有触摸功能且用户开始通过触摸界面与之交互，则不再需要突出显示项目或聚焦到特定视图对象上。 因此，有一种交互模式称为“触摸模式”。

对于支持触摸功能的设备，当用户触摸屏幕时，设备会立即进入触摸模式。 自此以后，只有 `isFocusableInTouchMode()` 为 true 的视图才可聚焦，如文本编辑小部件。其他可触摸的视图（如按钮）在用户触摸时不会获得焦点；按下时它们只是触发点击侦听器。

无论何时，只要用户点击方向键或滚动轨迹球，设备就会退出触摸模式并找到一个视图使其获得焦点。 现在，用户可在不触摸屏幕的情况下继续与用户界面交互。

整个系统（所有窗口和 Activity）都将保持触摸模式状态。要查询当前状态，您可以调用`isInTouchMode()` 来检查设备目前是否处于触摸模式。

处理焦点

该框架将处理例行焦点移动来响应用户输入。其中包括在视图被移除或隐藏时或在新视图变得可用时更改焦点。 视图对象表示愿意通过 `isFocusable()` 方法获得焦点。要设置视图能否获得焦点，请调用 `setFocusable()`。在触摸模式中，您可以使用 `isFocusableInTouchMode()` 查询视图是否允许聚焦，而且可以使用 `setFocusableInTouchMode()` 对此进行更改。

焦点移动所使用的算法会查找指定方向上距离最近的元素。 在极少数情况下，默认算法可能与开发者的期望行为不一致。 在这些情况下，您可以在布局文件中显式重写以下 XML 属性：***nextFocusDown***、***nextFocusLeft***、***nextFocusRight*** 和 ***nextFocusUp***。 将其中一个属性添加到失去焦点的视图。 将属性的值定义为应该聚焦的视图的 ID。例如：

```
<LinearLayout
    android:orientation="vertical"
    ... >
  <Button android:id="@+id/top"
          android:nextFocusUp="@+id/bottom"
          ... />
  <Button android:id="@+id/bottom"
          android:nextFocusDown="@+id/top"
          ... />
</LinearLayout>
```

一般来说，在此垂直布局中，从第一个按钮向上导航或从第二个按钮向下导航，焦点都不会移到任何其他位置。 现在，顶部按钮已将底部按钮定义为 ***nextFocusUp***（反之亦然），因而导航焦点将自上而下和自下而上循环往复。

若要将一个视图声明为在 UI 中可聚焦（传统上并非如此），请在布局声明中将 `android:focusable` XML 属性添加到该视图。将值设置为 ***true***。 此外，您还可以使用 `android:focusableInTouchMode` 将视图声明为在触摸模式下可聚焦。

要请求要获得焦点的特定视图，请调用 `requestFocus()`。

要侦听焦点事件（视图获得或失去焦点时会收到通知），请使用 `onFocusChange()`， 如上文的事件侦听器部分中所述。