Android焦点分发过程解析

薛定谔的程序猫 LV.3
2020年09月12日 18:14 · 阅读 1884

关注

Android焦点分发逻辑

引言

今天，我们来简单分析一下Android系统焦点分发逻辑，那么焦点分发的起点在哪里呢?

分发起点：dispatchKeyEvent

首先，让我们来看看按下 **KEYCODE_DPAD_LEFT** 按键的时候发生了什么？我们知道整个 ViewTree 按键分发的起点是 *ViewRootImpl.processKeyEvent(...)*，那 *processKeyEvent(...)* 又是如何分发按键事件的呢?

ViewRootImpl.java

java 复制代码

```java
    private int processKeyEvent(QueuedInputEvent q) {
        ......
        // Deliver the key to the view hierarchy.
        if (mView.dispatchKeyEvent(event)) {
            return FINISH_HANDLED;
        }
        if (shouldDropInputEvent(q)) {
            return FINISH_NOT_HANDLED;
        }
        ......
        // Handle automatic focus changes.
        // 转换成焦点事件
        if (event.getAction() == KeyEvent.ACTION_DOWN) {
            int direction = 0;
            switch (event.getKeyCode()) {
                case KeyEvent.KEYCODE_DPAD_LEFT:
                    if (event.hasNoModifiers()) {
                        direction = View.FOCUS_LEFT;
                    }
                    break;
                .....
            }
            if (direction != 0) {
                // 查找当前获焦的View
                View focused = mView.findFocus();
                if (focused != null) {
                    // 查找下一个获焦的View
                    View v = focused.focusSearch(direction);
                    if (v != null && v != focused) {
                        // do the math the get the interesting rect
                        // of previous focused into the coord system of
                        // newly focused view
                        // 计算当前获焦的View的位置
                        focused.getFocusedRect(mTempRect);
                        if (mView instanceof ViewGroup) {
                            ((ViewGroup) mView).offsetDescendantRectToMyCoords(
                                    focused, mTempRect);
                            ((ViewGroup) mView).offsetRectIntoDescendantCoords(
                                    v, mTempRect);
                        }
                        // 尝试分发焦点给下一个获焦的View
                        if (v.requestFocus(direction, mTempRect)) {
                            playSoundEffect(SoundEffectConstants
                                    .getContantForFocusDirection(direction));
                            return FINISH_HANDLED;
                        }
                    }
                    // Give the focused view a last chance to handle the dpad key.
                    // 最后的善后机会
                    if (mView.dispatchUnhandledMove(focused, direction)) {
                        return FINISH_HANDLED;
                    }
                } else {
                    // find the best view to give focus to in this non-touch-mode with
                    // 当前无获焦的View，则默认查找原点为(0, 0)
                    View v = focusSearch(null, direction);
                    // 直接尝试将焦点分发给找到的View
                    if (v != null && v.requestFocus(direction)) {
                        return FINISH_HANDLED;
                    }
                }
            }
        }
        return FORWARD;
    }
```

为了简化代码，此处省略了部分逻辑。从源码我们可以看出，按键事件首先会尝试分发给 ViewTree 去处理(此处我们不深入讨论)，如果 ViewTree 不做处理，那么就会进入焦点分发逻

辑。就是在这里，**按键事件分发转变成了焦点事件分发**。

- 首先，根据不同的按键事件转变为不同焦点分发事件，例如 *KEYCODE_DPAD_LEFT* 转变为 *FOCUS_LEFT*。
- 接着，尝试查找当前已获焦的View，如果存在获焦的View，就调用这个View的 ***focusSearch(...)*** 方法查找下一个获焦的View：
    - 如果找到下一个获焦的View，且该View不是当前已获焦的View，那么就计算当前已获焦View的获焦区域(并通过坐标变换计算出这个区域相对于下一个获焦View的位置)，然后调用 ***requestFocus(...)*** 移动焦点。
    - 如果没有找到下一个获焦View，或者找到的View就是当前已获焦的View，或者找到下一个获焦的View但requestFocus 失败了，那么就调用 ***dispatchUnhandledMove(...)*** 做最后的善后处理。因此，**可以在这个方法里面处理边界 View 的回弹效果。**
- 如果当前不存在已获焦的View，那么就直接调用 *ViewRootImpl* 的 *focusSearch(...)* 方法。当然，在这种场景下，查找的原点默认是屏幕左上角或者右下角。

PS: 对 *focusSearch(...)* 方法感兴趣的，可以移步Android焦点搜索逻辑，此处我们先略过。

移动焦点：

接下来，我们来看一下 *requestFocus(...)* 方法是如何处理焦点移动的：

requestFocus

View.java

java 复制代码

```java
public boolean requestFocus(int direction, Rect previouslyFocusedRect) {
    return requestFocusNoSearch(direction, previouslyFocusedRect);
}

private boolean requestFocusNoSearch(int direction, Rect previouslyFocusedRect) {
    // need to be focusable
    if ((mViewFlags & FOCUSABLE_MASK) != FOCUSABLE ||
            (mViewFlags & VISIBILITY_MASK) != VISIBLE) {
        return false;
    }
    // need to be focusable in touch mode if in touch mode
    if (isInTouchMode() &&
        (FOCUSABLE_IN_TOUCH_MODE != (mViewFlags & FOCUSABLE_IN_TOUCH_MODE))) {
           return false;
    }
    // need to not have any parents blocking us
    if (hasAncestorThatBlocksDescendantFocus()) {
        return false;
    }
    handleFocusGainInternal(direction, previouslyFocusedRect);
    return true;
}
```

View 的 *requestFocus(...)* 方法直接调用 *requestFocusNoSearch(...)* 方法，而 *requestFocusNoSearch(...)* 方法的逻辑是：

- 如果该 View 可获焦且没有被上级 ViewGroup 拦截，则调用 *handleFocusGainInternal(...)* 方法将焦点分发给该View。

ViewGroup.java

java 复制代码

```java
@Override
public boolean requestFocus(int direction, Rect previouslyFocusedRect) {
    if (DBG) {
        System.out.println(this + " ViewGroup.requestFocus direction="
                + direction);
    }
    int descendantFocusability = getDescendantFocusability();
    switch (descendantFocusability) {
        case FOCUS_BLOCK_DESCENDANTS:
            return super.requestFocus(direction, previouslyFocusedRect);
        case FOCUS_BEFORE_DESCENDANTS: {
            final boolean took = super.requestFocus(direction, previouslyFocusedRect);
            return took ? took : onRequestFocusInDescendants(direction, previouslyFocuse
        }
        case FOCUS_AFTER_DESCENDANTS: {
            final boolean took = onRequestFocusInDescendants(direction, previouslyFocuse
            return took ? took : super.requestFocus(direction, previouslyFocusedRect);
        }
        default:
            throw new IllegalStateException("descendant focusability must be "
                    + "one of FOCUS_BEFORE_DESCENDANTS, FOCUS_AFTER_DESCENDANTS, FOCUS_
                    + "but is " + descendantFocusability);
    }
}
```

正如 *ViewGroup* 的 *addFocusables(...)* 方法一样，*ViewGroup* 的 *requestFocus(...)* 方法也与 **descendantFocusability** 有关：

- __FOCUS_BLOCK_DESCENDANTS __： 仅尝试将焦点分发给当前 *ViewGroup*
- **FOCUS_BEFORE_DESCENDANTS**： 先尝试将焦点分发给当前 *ViewGroup*，然后才尝试将焦点分发给ChildView。
- **FOCUS_AFTER_DESCENDANTS**： 先尝试将焦点分发给ChildView，然后才尝试将焦点分发给当前 *ViewGroup*。

java 复制代码

```java
protected boolean onRequestFocusInDescendants(int direction,
        Rect previouslyFocusedRect) {
    int index;
    int increment;
    int end;
    int count = mChildrenCount;
    if ((direction & FOCUS_FORWARD) != 0) {
        index = 0;
        increment = 1;
        end = count;
    } else {
        index = count - 1;
        increment = -1;
        end = -1;
    }
    final View[] children = mChildren;
    for (int i = index; i != end; i += increment) {
        View child = children[i];
        if ((child.mViewFlags & VISIBILITY_MASK) == VISIBLE) {
            if (child.requestFocus(direction, previouslyFocusedRect)) {
                return true;
            }
        }
    }
    return false;
}
```

onRequestFocusInDescendants(...) 尝试按顺序将焦点分发给 ChildView。因此，**可以通过覆写这两个方法来实现自定义焦点分发逻辑。**

handleFocusGainInternal

java 复制代码

```java
/**
 * Give this view focus. This will cause
 * {@link #onFocusChanged(boolean, int, android.graphics.Rect)} to be called.
 */
void handleFocusGainInternal(@FocusRealDirection int direction, Rect previouslyFocusedRe
    if (DBG) {
        System.out.println(this + " requestFocus()");
    }
    if ((mPrivateFlags & PFLAG_FOCUSED) == 0) {
        mPrivateFlags |= PFLAG_FOCUSED;
        View oldFocus = (mAttachInfo != null) ? getRootView().findFocus() : null;
        if (mParent != null) {
            mParent.requestChildFocus(this, this);
        }
        if (mAttachInfo != null) {
            mAttachInfo.mTreeObserver.dispatchOnGlobalFocusChange(oldFocus, this);
        }
        onFocusChanged(true, direction, previouslyFocusedRect);
        refreshDrawableState();
    }
}
```

handleFocusGainInternal(...) 方法先检查当前 View 是否已获焦，如果已获焦则不做处理；如果未获焦，则：

- 设置获焦状态 PFLAG_FOCUSED
- 层层往上调用 *requestChildFocus(...)* 方法，通知 *mParent* 焦点变化事件
- 调用 *dispatchOnGlobalFocusChange(...)* 方法，通知 ViewTreeObserver 焦点变化事件
- 调用 *onFocusChanged(...)* 方法，通知当前View焦点变化事件
- 调用 *refreshDrawableState(...)* 刷新当前View的状态

java 复制代码

```java
protected void onFocusChanged(boolean gainFocus, @FocusDirection int direction,
        @Nullable Rect previouslyFocusedRect) {
    if (gainFocus) {
        sendAccessibilityEvent(AccessibilityEvent.TYPE_VIEW_FOCUSED);
    } else {
        notifyViewAccessibilityStateChangedIfNeeded(
                AccessibilityEvent.CONTENT_CHANGE_TYPE_UNDEFINED);
    }
    InputMethodManager imm = InputMethodManager.peekInstance();
    if (!gainFocus) {
        if (isPressed()) {
            setPressed(false);
        }
        if (imm != null && mAttachInfo != null
                && mAttachInfo.mHasWindowFocus) {
            imm.focusOut(this);
        }
        onFocusLost();
    } else if (imm != null && mAttachInfo != null
            && mAttachInfo.mHasWindowFocus) {
        imm.focusIn(this);
    }
    invalidate(true);
    ListenerInfo li = mListenerInfo;
    if (li != null && li.mOnFocusChangeListener != null) {
        li.mOnFocusChangeListener.onFocusChange(this, gainFocus);
    }
    if (mAttachInfo != null) {
        mAttachInfo.mKeyDispatchState.reset(this);
    }
}
```

但是我们看到 *onFocusChanged(...)* 方法并没有做什么特别处理，那原来获焦的那个 View 怎么办？它又是如何知道自己失去焦点了呢?

既然 *onFocusChanged(...)* 方法没有做处理，那么我们不妨来看看是不是 *mParent.requestChildFocus(...)* 这个方法做处理了:

requestChildFocus

[ViewGroup.java]

java 复制代码

```java
public void requestChildFocus(View child, View focused) {
    if (DBG) {
        System.out.println(this + " requestChildFocus()");
    }
    if (getDescendantFocusability() == FOCUS_BLOCK_DESCENDANTS) {
        return;
    }
    // Unfocus us, if necessary
    super.unFocus(focused);
    // We had a previous notion of who had focus. Clear it.
    if (mFocused != child) {
        if (mFocused != null) {
            mFocused.unFocus(focused);
        }
        mFocused = child;
    }
    if (mParent != null) {
        mParent.requestChildFocus(this, focused);
    }
}

void unFocus(View focused) {
    if (DBG) {
        System.out.println(this + " unFocus()");
    }
    if (mFocused == null) {
        super.unFocus(focused);
    } else {
        mFocused.unFocus(focused);
        mFocused = null;
    }
}
```

requestChildFocus(...) 方法的处理逻辑：

- 如果 descendantFocusability 的值等于FOCUS_BLOCK_DESCENDANTS，则说明拦截了ChildView的获焦事件，此时我们不需要继续向上一层级透传。

- 调用 *super.unFocus(...)* 方法清除当前 ViewGroup 的焦点(如果当前 ViewGroup 是原来获焦的View)

- 如果原来获焦的是当前 ViewGroup 的 ChildView，则调用 *mFocused.unFocus(...)* 方法清除其焦点

- 调用 *mParent.requestChildFocus(...)* 方法透传通知上一层级焦点变化事件

因此，当ChildView获得焦点的时候，ParentView都可以通过 *requestChildFocus(...)* 方法接收到焦点变化事件，如图所示：

java 复制代码

```
                           *  ViewRootImpl
                         /
      requestChildFocus *
                      /
   requestChildFocus * unFocus
                   /   \
requestChildFocus *     * unFocus
```

```
                      /         \
   requestChildFocus *           * unFocus
                   /
     requestFocus *
```

我们接着往下看 *unFocus(...)* 方法是如何清除焦点的：

unFocus

View.java

```java
void unFocus(View focused) {
    if (DBG) {
        System.out.println(this + " unFocus()");
    }
    clearFocusInternal(focused, false, false);
}

void clearFocusInternal(View focused, boolean propagate, boolean refocus) {
    if ((mPrivateFlags & PFLAG_FOCUSED) != 0) {
        mPrivateFlags &= ~PFLAG_FOCUSED;
        if (propagate && mParent != null) {
            mParent.clearChildFocus(this);
        }
        onFocusChanged(false, 0, null);
        refreshDrawableState();
        if (propagate && (!refocus || !rootViewRequestFocus())) {
            notifyGlobalFocusCleared(this);
        }
    }
}
```

可以看到 *unFocus(...)* 方法是直接调用 *clearFocusInternal(...)* 方法尝试清除当前View的获焦状态。 *clearFocusInternal(...)* 方法先检查当前View是否已获焦，如果未获焦则无需处理，如果当前View已获焦，则：

- 清除焦点状态PFLAG_FOCUSED
- 调用 *mParent.clearChildFocus(...)* 方法通知上一层级焦点清除事件
- 调用 *onFocusChanged(...)* 通知当前View焦点变化事件
- 调用 *refreshDrawableState(...)* 刷新当前View的显示状态

- 如果refocus为true，则调用 *rootViewRequestFocus(...)* 方法重新分发焦点。

清除焦点

那什么时候需要重新分发焦点呢？**当我们调用手动 clearFocus() 清除焦点 或者 获焦的 View 被移除(隐藏不可见)的时候，就需要重新分发焦点**：

clearFocus

java 复制代码

```java
protected void removeDetachedView(View child, boolean animate) {
    ......
    if (child == mFocused) {
        child.clearFocus();
    }
    ......
}

public void clearFocus() {
    if (DBG) {
        System.out.println(this + " clearFocus()");
    }
    clearFocusInternal(null, true, true);
}

boolean rootViewRequestFocus() {
    final View root = getRootView();
    return root != null && root.requestFocus();
}
```

因此，当 ChildView 失去焦点的时候，ParentView 都可以通过 *clearChildFocus(...)* 方法接收到焦点清除事件，如图所示：

java 复制代码

```java
                           *  ViewRootImpl
                         /
                        * clearChildFocus
                      /
                     * clearChildFocus
                   /   \
                  *     * clearChildFocus
                /         \
               *           * clearFocus
```

```
 /
*
```

校正焦点

focusableViewAvailable

focusableViewAvailable(...) 是官方提供的实时初始化焦点或者校正焦点的机制：简单的说，当一个View 变为可获焦的状态之后，就会通过 *focusableViewAvailable(...)* 层层透传至 *ViewRootImpl*，由 *ViewRootImpl* 来初始化焦点或者校正焦点。

View.java

java 复制代码

```java
void setFlags(int flags, int mask) {
    final boolean accessibilityEnabled =
            AccessibilityManager.getInstance(mContext).isEnabled();
    final boolean oldIncludeForAccessibility = accessibilityEnabled && includeForAccess
    int old = mViewFlags;
    mViewFlags = (mViewFlags & ~mask) | (flags & mask);
    int changed = mViewFlags ^ old;
    if (changed == 0) {
        return;
    }
    int privateFlags = mPrivateFlags;
    // 检查可获焦状态是否改变
    /* Check if the FOCUSABLE bit has changed */
    if (((changed & FOCUSABLE_MASK) != 0) &&
            ((privateFlags & PFLAG_HAS_BOUNDS) !=0)) {
        if (((old & FOCUSABLE_MASK) == FOCUSABLE)
                && ((privateFlags & PFLAG_FOCUSED) != 0)) {
            /* Give up focus if we are no longer focusable */
            clearFocus();
        } else if (((old & FOCUSABLE_MASK) == NOT_FOCUSABLE)
                && ((privateFlags & PFLAG_FOCUSED) == 0)) {
            /*
             * Tell the view system that we are now available to take focus
             * if no one else already has it.
             */
            if (mParent != null) mParent.focusableViewAvailable(this);
        }
    }
    // 检查可见状态是否改变
    final int newVisibility = flags & VISIBILITY_MASK;
    if (newVisibility == VISIBLE) {
        if ((changed & VISIBILITY_MASK) != 0) {
            /*
             * If this view is becoming visible, invalidate it in case it changed while
             * it was not visible. Marking it drawn ensures that the invalidation will
             * go through.
             */
            mPrivateFlags |= PFLAG_DRAWN;
            invalidate(true);
            needGlobalAttributesUpdate(true);
            // a view becoming visible is worth notifying the parent
            // about in case nothing has focus.  even if this specific view
            // isn't focusable, it may contain something that is, so let
            // the root view try to give this focus if nothing else does.
            if ((mParent != null) && (mBottom > mTop) && (mRight > mLeft)) {
                mParent.focusableViewAvailable(this);
            }
        }
    }
    .....
}
```

View 类中的 *setFlags(...)* 方法中检查可获焦状态或者可见状态是否改变，如变为可获焦状态，则调用 *mParent.focusableViewAvailable(...)* 方法通知上级节点。

ViewGroup.java

java 复制代码

```java
public void focusableViewAvailable(View v) {
    if (mParent != null
            // shortcut: don't report a new focusable view if we block our descendants
            // getting focus
            && (getDescendantFocusability() != FOCUS_BLOCK_DESCENDANTS)
            && (isFocusableInTouchMode() || !shouldBlockFocusForTouchscreen())
            // shortcut: don't report a new focusable view if we already are focused
            // (and we don't prefer our descendants)
            //
            // note: knowing that mFocused is non-null is not a good enough reason
            // to break the traversal since in that case we'd actually have to find
            // the focused view and make sure it wasn't FOCUS_AFTER_DESCENDANTS and
            // an ancestor of v; this will get checked for at ViewAncestor
            && !(isFocused() && getDescendantFocusability() != FOCUS_AFTER_DESCENDANTS)
        mParent.focusableViewAvailable(v);
    }
}
```

ViewGroup 类中的 *focusableViewAvailable(...)* 负责检查并向上一层级透传，直至 *ViewRootImpl*。这里我们看到有几种条件下是不往上透传的：

- ParentView 设置 *descendantFocusability* 值为 *FOCUS_BLOCK_DESCENDANTS*，即拦截 ChildView 获焦。

- ParentView 设置 *descendantFocusability* 值不为 *FOCUS_AFTER_DESCENDANTS* 且 ParentView 处于获焦状态，因为这个状态下无需校正焦点。

ViewRootImpl.java

java 复制代码

```java
@Override
public void focusableViewAvailable(View v) {
    checkThread();
    if (mView != null) {
        if (!mView.hasFocus()) {
            v.requestFocus();
        } else {
            // the one case where will transfer focus away from the current one
            // is if the current view is a view group that prefers to give focus
            // to its children first AND the view is a descendant of it.
            View focused = mView.findFocus();
            if (focused instanceof ViewGroup) {
                ViewGroup group = (ViewGroup) focused;
                if (group.getDescendantFocusability() == ViewGroup.FOCUS_AFTER_DESCENDAN
                        && isViewDescendantOf(v, focused)) {
                    v.requestFocus();
                }
            }
        }
    }
}
```

ViewRootImpl 类中的 *focusableViewAvailable(...)* 方法：

- 如果当前不存在焦点，则直接尝试将焦点分发给这个可获焦的 View

- 如果存在焦点，则检查是否需要将焦点转移到这个可获焦的 View

那么什么情况下需要将焦点转移给这个可获焦的 View 呢？如果当前获焦的 ViewGroup 是这个可获焦的 View 的上级节点，且其 descendantFocusability 值为 *FOCUS_AFTER_DESCENDANTS*，则会尝试将焦点分发给这个可获焦的View。

也就是说 *focusableViewAvailable(...)* 这个方法一方面负责处理焦点初始化的逻辑，另一方面也会实时校正 *FOCUS_AFTER_DESCENDANTS* 的 ViewGroup 的焦点分发。

这是因为 *FOCUS_AFTER_DESCENDANTS* 表示的是 ChildView 优先获焦，如果因为 ChildView 不可获焦而让 ParentView 先获焦了，当 ChildView 变为可获焦了，则 ParentView 应当及时将焦点转移给 ChildView。

分类： Android 标签： Android