> 本文由 简悦 SimpRead 转码， 原文地址 kaiwu.lagou.com

上节课，我们已经知道了，Vue.js 提供了内置的 Transition 组件帮我们实现动画过渡效果。在之前的分析中我把 Transition 组件的实现分成了三个部分：组件的渲染、钩子函数的执行、模式的应用。这节课我们从钩子函数的执行继续探究 Transition 组件的实现原理。

钩子函数的执行

这个部分我们先来看 beforeEnter 钩子函数。

在 patch 阶段的 mountElement 函数中，在插入元素节点前且存在过渡的条件下会执行 vnode.transition 中的 beforeEnter 函数，我们来看它的定义：

```
beforeEnter(el) {

  let hook = onBeforeEnter

if (!state.isMounted) {

if (appear) {

      hook = onBeforeAppear || onBeforeEnter

    }

else {

return

    }

  }

if (el._leaveCb) {

    el._leaveCb(true )

  }

const leavingVNode = leavingVNodesCache[key]
```

```
if (leavingVNode &&

    isSameVNodeType(vnode, leavingVNode) &&

    leavingVNode.el._leaveCb) {

    leavingVNode.el._leaveCb()

  }

  callHook(hook, [el])

}
```

beforeEnter 钩子函数主要做的事情就是根据 appear 的值和 DOM 是否挂载，来执行 onBeforeEnter 函数或者是 onBeforeAppear 函数，其他的逻辑我们暂时先不看。

appear、onBeforeEnter、onBeforeAppear 这些变量都是从 props 中获取的，那么这些 props 是怎么初始化的呢？回到 Transition 组件的定义：

```
const Transition = (props, { slots }) => h(BaseTransition,
resolveTransitionProps(props), slots)
```

可以看到，传递的 props 经过了 resolveTransitionProps 函数的封装，我们来看它的定义：

```
function resolveTransitionProps(rawProps) {

  let { name = 'v', type, css = true, duration, enterFromClass = `${name}-enter-
from`, enterActiveClass = `${name}-enter-active`, enterToClass = `${name}-enter-
to`, appearFromClass = enterFromClass, appearActiveClass = enterActiveClass,
appearToClass = enterToClass, leaveFromClass = `${name}-leave-from`,
leaveActiveClass = `${name}-leave-active`, leaveToClass = `${name}-leave-to` } =
rawProps

const baseProps = {}

for (const key in rawProps) {

if (!(key in DOMTransitionPropsValidators)) {

      baseProps[key] = rawProps[key]
```

```
    }

  }

if (!css) {

return baseProps

  }

const durations = normalizeDuration(duration)

const enterDuration = durations && durations[0]

const leaveDuration = durations && durations[1]

const { onBeforeEnter, onEnter, onEnterCancelled, onLeave, onLeaveCancelled,
onBeforeAppear = onBeforeEnter, onAppear = onEnter, onAppearCancelled =
onEnterCancelled } = baseProps

const finishEnter = (el, isAppear, done) => {

    removeTransitionClass(el, isAppear ? appearToClass : enterToClass)

    removeTransitionClass(el, isAppear ? appearActiveClass : enterActiveClass)

    done && done()

  }

const finishLeave = (el, done) => {

    removeTransitionClass(el, leaveToClass)

    removeTransitionClass(el, leaveActiveClass)

    done && done()

  }
```

```
const makeEnterHook = (isAppear) => {

return (el, done) => {

const hook = isAppear ? onAppear : onEnter

const resolve = () => finishEnter(el, isAppear, done)

      hook && hook(el, resolve)

      nextFrame(() => {

        removeTransitionClass(el, isAppear ? appearFromClass : enterFromClass)

        addTransitionClass(el, isAppear ? appearToClass : enterToClass)

if (!(hook && hook.length > 1)) {

if (enterDuration) {

            setTimeout(resolve, enterDuration)

          }

else {

            whenTransitionEnds(el, type, resolve)

          }

        }

      })

    }

  }
```

```
return extend(baseProps, {

    onBeforeEnter(el) {

      onBeforeEnter && onBeforeEnter(el)

      addTransitionClass(el, enterActiveClass)

      addTransitionClass(el, enterFromClass)

    },

    onBeforeAppear(el) {

      onBeforeAppear && onBeforeAppear(el)

      addTransitionClass(el, appearActiveClass)

      addTransitionClass(el, appearFromClass)

    },

    onEnter: makeEnterHook(false),

    onAppear: makeEnterHook(true),

    onLeave(el, done) {

const resolve = () => finishLeave(el, done)

      addTransitionClass(el, leaveActiveClass)

      addTransitionClass(el, leaveFromClass)

      nextFrame(() => {

        removeTransitionClass(el, leaveFromClass)

        addTransitionClass(el, leaveToClass)
```

```
if (!(onLeave && onLeave.length > 1)) {

if (leaveDuration) {

            setTimeout(resolve, leaveDuration)

          }

else {

            whenTransitionEnds(el, type, resolve)

          }

        }

      })

      onLeave && onLeave(el, resolve)

    },

    onEnterCancelled(el) {

      finishEnter(el, false)

      onEnterCancelled && onEnterCancelled(el)

    },

    onAppearCancelled(el) {

      finishEnter(el, true)

      onAppearCancelled && onAppearCancelled(el)

    },
```

```
    onLeaveCancelled(el) {

      finishLeave(el)

      onLeaveCancelled && onLeaveCancelled(el)

    }

  })

}
```

resolveTransitionProps 函数主要作用是，在我们给 Transition 传递的 Props 基础上做一层封装，然后返回一个新的 Props 对象，由于它包含了所有的 Props 处理，你不需要一下子了解所有的实现，按需分析即可。

我们来看 onBeforeEnter 函数，它的内部执行了基础 props 传入的 onBeforeEnter 钩子函数，并且给 DOM 元素 el 添加了 enterActiveClass 和 enterFromClass 样式。

其中，props 传入的 onBeforeEnter 函数就是我们写 Transition 组件时添加的 beforeEnter 钩子函数。enterActiveClass 默认值是 v-enter-active，enterFromClass 默认值是 v-enter-from，如果给 Transition 组件传入了 name 的 prop，比如 fade，那么 enterActiveClass 的值就是 fade-enter-active，enterFromClass 的值就是 fade-enter-from。

原来这就是在 DOM 元素对象在创建后，插入到页面前做的事情：**执行 beforeEnter 钩子函数，以及给元素添加相应的 CSS 样式。**

onBeforeAppear 和 onBeforeEnter 的逻辑类似，就不赘述了，它是在我们给 Transition 组件传入 appear 的 Prop，且首次挂载的时候执行的。

执行完 beforeEnter 钩子函数，接着插入元素到页面，然后会执行 vnode.transition 中的 enter 钩子函数，我们来看它的定义：

```
enter(el) {

  let hook = onEnter

  let afterHook = onAfterEnter

  let cancelHook = onEnterCancelled

if (!state.isMounted) {

if (appear) {
```

```
      hook = onAppear || onEnter

      afterHook = onAfterAppear || onAfterEnter

      cancelHook = onAppearCancelled || onEnterCancelled

    }

else {

return

    }

  }

  let called = false

const done = (el._enterCb = (cancelled) => {

if (called)

return

    called = true

if (cancelled) {

      callHook(cancelHook, [el])

    }

else {

      callHook(afterHook, [el])

    }

if (hooks.delayedLeave) {
```

```
      hooks.delayedLeave()

    }

    el._enterCb = undefined

  })

if (hook) {

    hook(el, done)

if (hook.length <= 1) {

      done()

    }

  }

else {

    done()

  }

}
```

enter 钩子函数主要做的事情就是根据 appear 的值和 DOM 是否挂载，执行 onEnter 函数或者是 onAppear 函数，并且这个函数的第二个参数是一个 done 函数，表示过渡动画完成后执行的回调函数，它是异步执行的。

> 注意，当 onEnter 或者 onAppear 函数的参数长度小于等于 1 的时候，done 函数在执行完 hook 函数后同步执行。

在 done 函数的内部，我们会执行 onAfterEnter 函数或者是 onEnterCancelled 函数，其它的逻辑我们也暂时先不看。

同理，onEnter、onAppear、onAfterEnter 和 onEnterCancelled 函数也是从 Props 传入的，我们重点看 onEnter 的实现，它是 makeEnterHook(false) 函数执行后的返回值，如下：

```
const makeEnterHook = (isAppear) => {

return (el, done) => {

const hook = isAppear ? onAppear : onEnter

const resolve = () => finishEnter(el, isAppear, done)

    hook && hook(el, resolve)

    nextFrame(() => {

      removeTransitionClass(el, isAppear ? appearFromClass : enterFromClass)

      addTransitionClass(el, isAppear ? appearToClass : enterToClass)

if (!(hook && hook.length > 1)) {

if (enterDuration) {

          setTimeout(resolve, enterDuration)

        }

else {

          whenTransitionEnds(el, type, resolve)

        }

      }

    })

  }

}
```

在函数内部，首先执行基础 props 传入的 onEnter 钩子函数，然后在下一帧给 DOM 元素 el 移除了 enterFromClass，同时添加了 enterToClass 样式。

其中，props 传入的 onEnter 函数就是我们写 Transition 组件时添加的 enter 钩子函数，enterFromClass 是我们在 beforeEnter 阶段添加的，会在当前阶段移除，新增的 enterToClass 值默认是 v-enter-to，如果给 Transition 组件传入了 name 的 prop，比如 fade，那么 enterToClass 的值就是 fade-enter-to。

注意，当我们添加了 enterToClass 后，这个时候浏览器就开始根据我们编写的 CSS 进入过渡动画了，那么动画何时结束呢？

Transition 组件允许我们传入 enterDuration 这个 prop，它会指定进入过渡的动画时长，当然如果你不指定，Vue.js 内部会监听动画结束事件，然后在动画结束后，执行 finishEnter 函数，来看它的实现：

```
const finishEnter = (el, isAppear, done) => {

  removeTransitionClass(el, isAppear ? appearToClass : enterToClass)

  removeTransitionClass(el, isAppear ? appearActiveClass : enterActiveClass)

  done && done()

}
```

其实就是给 DOM 元素移除 enterToClass 以及 enterActiveClass，同时执行 done 函数，进而执行 onAfterEnter 钩子函数。

至此，元素进入的过渡动画逻辑就分析完了，接下来我们来分析元素离开的过渡动画逻辑。

当元素被删除的时候，会执行 remove 方法，在真正从 DOM 移除元素前且存在过渡的情况下，会执行 vnode.transition 中的 leave 钩子函数，并且把移动 DOM 的方法作为第二个参数传入，我们来看它的定义：

```
leave(el, remove) {

const key = String(vnode.key)

if (el._enterCb) {

    el._enterCb(true )

  }

if (state.isUnmounting) {
```

```
return remove()

  }

  callHook(onBeforeLeave, [el])

  let called = false

const done = (el._leaveCb = (cancelled) => {

if (called)

return

    called = true

remove()

if (cancelled) {

      callHook(onLeaveCancelled, [el])

    }

else {

      callHook(onAfterLeave, [el])

    }

    el._leaveCb = undefined

if (leavingVNodesCache[key] === vnode) {

      delete leavingVNodesCache[key]

    }

  })
```

```
  leavingVNodesCache[key] = vnode

if (onLeave) {

    onLeave(el, done)

if (onLeave.length <= 1) {

      done()

    }

  }

else {

    done()

  }

}
```

leave 钩子函数主要做的事情就是执行 props 传入的 onBeforeLeave 钩子函数和 onLeave 函数，onLeave 函数的第二个参数是一个 done 函数，它表示离开过渡动画结束后执行的回调函数。

done 函数内部主要做的事情就是执行 remove 方法移除 DOM，然后执行 onAfterLeave 钩子函数或者是 onLeaveCancelled 函数，其它的逻辑我们也先不看。

接下来，我们重点看一下 onLeave 函数的实现，看看离开过渡动画是如何执行的。

```
onLeave(el, done) {

const resolve = () => finishLeave(el, done)

  addTransitionClass(el, leaveActiveClass)

  addTransitionClass(el, leaveFromClass)

  nextFrame(() => {
```

```
    removeTransitionClass(el, leaveFromClass)

    addTransitionClass(el, leaveToClass)

if (!(onLeave && onLeave.length > 1)) {

if (leaveDuration) {

        setTimeout(resolve, leaveDuration)

      }

else {

        whenTransitionEnds(el, type, resolve)

      }

    }

  })

  onLeave && onLeave(el, resolve)

}
```

onLeave 函数首先给 DOM 元素添加 leaveActiveClass 和 leaveFromClass，并执行基础 props 传入的 onLeave 钩子函数，然后在下一帧移除 leaveFromClass，并添加 leaveToClass。

其中，leaveActiveClass 的默认值是 v-leave-active，leaveFromClass 的默认值是 v-leave-from，leaveToClass 的默认值是 v-leave-to。如果给 Transition 组件传入了 name 的 prop，比如 fade，那么 leaveActiveClass 的值就是 fade-leave-active，leaveFromClass 的值就是 fade-leave-from，leaveToClass 的值就是 fade-leave-to。

注意，当我们添加 leaveToClass 时，浏览器就开始根据我们编写的 CSS 执行离开过渡动画了，那么动画何时结束呢?

和进入动画类似，Transition 组件允许我们传入 leaveDuration 这个 prop，指定过渡的动画时长，当然如果你不指定，Vue.js 内部会监听动画结束事件，然后在动画结束后，执行 resolve 函数，它是执行 finishLeave 函数的返回值，来看它的实现：

```
const finishLeave = (el, done) => {
```

```
  removeTransitionClass(el, leaveToClass)

  removeTransitionClass(el, leaveActiveClass)

  done && done()

}
```

其实就是给 DOM 元素移除 leaveToClass 以及 leaveActiveClass，同时执行 done 函数，进而执行 onAfterLeave 钩子函数。

至此，元素离开的过渡动画逻辑就分析完了，可以看出离开过渡动画和进入过渡动画是的思路差不多，本质上都是在添加和移除一些 CSS 去执行动画，并且在过程中执行用户传入的钩子函数。

模式的应用

前面我们在介绍 Transition 的渲染过程中提到过模式的应用，模式有什么用呢，我们还是通过示例说明，把前面的例子稍加修改：

```
<template>

<div>

<button @click="show = !show">

      Toggle render

</button>

<transition >

<p v-if="show">hello</p>

<p v-else>hi</p>

</transition>

</div>

</template>
```

```
<script>

export default {

    data() {

return {

show: true

      }

    }

  }

</script>

<style>

.fade-enter-active,

.fade-leave-active {

transition: opacity 0.5s ease;

  }

.fade-enter-from,

.fade-leave-to {

opacity: 0;

  }

</style>
```

我们在 show 条件为 false 的情况下，显示字符串 hi，你可以运行这个示例，然后会发现这个过渡效果有点生硬，并不理想。

然后，我们给这个 Transition 组件加一个 out-in 的 mode：

```
<transition mode="out-in" >

<p v-if="show">hello</p>

<p v-else>hi</p>

</transition>
```

我们会发现这个过渡效果好多了，hello 文本先完成离开的过渡后，hi 文本开始进入过渡动画。

模式非常适合这种两个元素切换的场景，Vue.js 给 Transition 组件提供了两种模式， in-out 和 out-in ，它们有什么区别呢？

- 在 in-out 模式下，新元素先进行过渡，完成之后当前元素过渡离开。
- 在 out-in 模式下，当前元素先进行过渡，完成之后新元素过渡进入。

在实际工作中，你大部分情况都是在使用 out-in 模式，而 in-out 模式很少用到，所以接下来我们就来分析 out-in 模式的实现原理。

我们先不妨思考一下，为什么在不加模式的情况下，会出现示例那样的过渡效果。

当我们点击按钮，show 变量由 true 变成 false，会触发当前元素 hello 文本的离开动画，也会同时触发新元素 hi 文本的进入动画。由于动画是同时进行的，而且在离开动画结束之前，当前元素 hello 是没有被移除 DOM 的，所以它还会占位，就把新元素 hi 文本挤到下面去了。当 hello 文本的离开动画执行完毕从 DOM 中删除后，hi 文本才能回到之前的位置。

那么，我们怎么做才能做到当前元素过渡动画执行完毕后，再执行新元素的过渡呢？

我们来看一下 out-in 模式的实现：

```
const leavingHooks = resolveTransitionHooks(oldInnerChild, rawProps, state,
instance)


setTransitionHooks(oldInnerChild, leavingHooks)


if (mode === 'out-in') {


  state.isLeaving = true


  leavingHooks.afterLeave = () => {


    state.isLeaving = false
```

```
    instance.update()

  }

return emptyPlaceholder(child)

}
```

当模式为 out-in 的时候，会标记 state.isLeaving 为 true，然后返回一个空的注释节点，同时更新当前元素的钩子函数中的 afterLeave 函数，内部执行 instance.update 重新渲染组件。

这样做就保证了在当前元素执行离开过渡的时候，新元素只渲染成一个注释节点，这样页面上看上去还是只执行当前元素的离开过渡动画。

然后当离开动画执行完毕后，触发了 Transition 组件的重新渲染，这个时候就可以如期渲染新元素并执行进入过渡动画了，是不是很巧妙呢？

总结

好的，到这里我们这一节的学习就结束啦，通过这节课的学习，你应该了解了 Transition 组件是如何渲染的，如何执行过渡动画和相应的钩子函数的，以及当两个视图切换时，模式的工作原理是怎样的。

最后，给你留一道思考题，Transition 组件在 beforeEnter 钩子函数里会判断 el.*leaveCb 是否存在，存在则执行，在 leave 钩子函数里会判断 el.*enterCb 是否存在，存在则执行，这么做的原因是什么？欢迎你在留言区与我分享。

> 本节课的相关代码在源代码中的位置如下：
> packages/runtime-core/src/components/BasetTransition.ts
> packages/runtime-core/src/renderer.ts
> packages/runtime-dom/src/components/Transition.ts