## **Vue 实例挂载的实现**

# **Vue 实例挂载的实现**

Vue 中我们是通过 $mount 实例方法去挂载 vm 的，$mount 方法在多个文件中都有定义，如 src/platform/web/entry-runtime-with-compiler.js、src/platform/web/runtime/index.js、src/platform/weex/runtime/index.js。因为 $mount 这个方法的实现是和平台、构建方式都相关的。接下来我们重点分析带 compiler 版本的 $monut 实现，因为抛开 webpack 的 vue-loader，我们在纯前端浏览器环境分析 Vue 的工作原理，有助于我们对原理理解的深入。

compiler 版本的 $monut 实现非常有意思，先来看一下 src/platform/web/entry-runtime-with-compiler.js 文件中定义：

const mount = Vue.prototype.$mount

Vue.prototype.$mount = function (

el?: string | Element,

hydrating?: boolean

): Component {

el = el && query(el)

/\* istanbul ignore if \*/

if (el === document.body || el === document.documentElement) {

process.env.NODE\_ENV !== 'production' && warn(

`Do not mount Vue to <html> or <body> - mount to normal elements instead.`

)

return this

}

const options = this.$options

// resolve template/el and convert to render function

if (!options.render) {

let template = options.template

if (template) {

if (typeof template === 'string') {

if (template.charAt(0) === '#') {

template = idToTemplate(template)

/\* istanbul ignore if \*/

if (process.env.NODE\_ENV !== 'production' && !template) {

warn(

`Template element not found or is empty: ${options.template}`,

this

)

}

}

} else if (template.nodeType) {

template = template.innerHTML

} else {

if (process.env.NODE\_ENV !== 'production') {

warn('invalid template option:' + template, this)

}

return this

}

} else if (el) {

template = getOuterHTML(el)

}

if (template) {

/\* istanbul ignore if \*/

if (process.env.NODE\_ENV !== 'production' && config.performance && mark) {

mark('compile')

}

const { render, staticRenderFns } = compileToFunctions(template, {

shouldDecodeNewlines,

shouldDecodeNewlinesForHref,

delimiters: options.delimiters,

comments: options.comments

}, this)

options.render = render

options.staticRenderFns = staticRenderFns

/\* istanbul ignore if \*/

if (process.env.NODE\_ENV !== 'production' && config.performance && mark) {

mark('compile end')

measure(`vue ${this.\_name} compile`, 'compile', 'compile end')

}

}

}

return mount.call(this, el, hydrating)

}

这段代码首先缓存了原型上的 $mount 方法，再重新定义该方法，我们先来分析这段代码。首先，它对 el 做了限制，Vue 不能挂载在 body、html 这样的根节点上。接下来的是很关键的逻辑 —— 如果没有定义 render 方法，则会把 el 或者 template 字符串转换成 render 方法。这里我们要牢记，在 Vue 2.0 版本中，所有 Vue 的组件的渲染最终都需要 render 方法，无论我们是用单文件 .vue 方式开发组件，还是写了 el 或者 template 属性，最终都会转换成 render 方法，那么这个过程是 Vue 的一个“在线编译”的过程，它是调用 compileToFunctions 方法实现的，编译过程我们之后会介绍。最后，调用原先原型上的 $mount 方法挂载。

原先原型上的 $mount 方法在 src/platform/web/runtime/index.js 中定义，之所以这么设计完全是为了复用，因为它是可以被 runtime only 版本的 Vue 直接使用的。

// public mount method

Vue.prototype.$mount = function (

el?: string | Element,

hydrating?: boolean

): Component {

el = el && inBrowser ? query(el) : undefined

return mountComponent(this, el, hydrating)

}

$mount 方法支持传入 2 个参数，第一个是 el，它表示挂载的元素，可以是字符串，也可以是 DOM 对象，如果是字符串在浏览器环境下会调用 query 方法转换成 DOM 对象的。第二个参数是和服务端渲染相关，在浏览器环境下我们不需要传第二个参数。

$mount 方法实际上会去调用 mountComponent 方法，这个方法定义在 src/core/instance/lifecycle.js 文件中：

export function mountComponent (

vm: Component,

el: ?Element,

hydrating?: boolean

): Component {

vm.$el = el

if (!vm.$options.render) {

vm.$options.render = createEmptyVNode

if (process.env.NODE\_ENV !== 'production') {

/\* istanbul ignore if \*/

if ((vm.$options.template && vm.$options.template.charAt(0) !== '#') ||

vm.$options.el || el) {

warn(

'You are using the runtime-only build of Vue where the template ' +

'compiler is not available. Either pre-compile the templates into ' +

'render functions, or use the compiler-included build.',

vm

)

} else {

warn(

'Failed to mount component: template or render function not defined.',

vm

)

}

}

}

callHook(vm, 'beforeMount')

let updateComponent

/\* istanbul ignore if \*/

if (process.env.NODE\_ENV !== 'production' && config.performance && mark) {

updateComponent = () => {

const name = vm.\_name

const id = vm.\_uid

const startTag = `vue-perf-start:${id}`

const endTag = `vue-perf-end:${id}`

mark(startTag)

const vnode = vm.\_render()

mark(endTag)

measure(`vue ${name} render`, startTag, endTag)

mark(startTag)

vm.\_update(vnode, hydrating)

mark(endTag)

measure(`vue ${name} patch`, startTag, endTag)

}

} else {

updateComponent = () => {

vm.\_update(vm.\_render(), hydrating)

}

}

// we set this to vm.\_watcher inside the watcher's constructor

// since the watcher's initial patch may call $forceUpdate (e.g. inside child

// component's mounted hook), which relies on vm.\_watcher being already defined

new Watcher(vm, updateComponent, noop, {

before () {

if (vm.\_isMounted) {

callHook(vm, 'beforeUpdate')

}

}

}, true /\* isRenderWatcher \*/)

hydrating = false

// manually mounted instance, call mounted on self

// mounted is called for render-created child components in its inserted hook

if (vm.$vnode == null) {

vm.\_isMounted = true

callHook(vm, 'mounted')

}

return vm

}

从上面的代码可以看到，mountComponent 核心就是先调用 vm.\_render 方法先生成虚拟 Node，再实例化一个渲染Watcher，在它的回调函数中会调用 updateComponent 方法，最终调用 vm.\_update 更新 DOM。

Watcher 在这里起到两个作用，一个是初始化的时候会执行回调函数，另一个是当 vm 实例中的监测的数据发生变化的时候执行回调函数，这块儿我们会在之后的章节中介绍。

函数最后判断为根节点的时候设置 vm.\_isMounted 为 true， 表示这个实例已经挂载了，同时执行 mounted 钩子函数。 这里注意 vm.$vnode 表示 Vue 实例的父虚拟 Node，所以它为 Null 则表示当前是根 Vue 的实例。

## **总结**

mountComponent 方法的逻辑也是非常清晰的，它会完成整个渲染工作，接下来我们要重点分析其中的细节，也就是最核心的 2 个方法：vm.\_render 和 vm.\_update。

****任务****

请仔细阅读本文档，为下面学习视频内容做准备。