## **从入口开始**

# **从入口开始**

我们之前提到过 Vue.js 构建过程，在 web 应用下，我们来分析 Runtime + Compiler 构建出来的 Vue.js，它的入口是 src/platforms/web/entry-runtime-with-compiler.js：

/\* @flow \*/

import config from 'core/config'

import { warn, cached } from 'core/util/index'

import { mark, measure } from 'core/util/perf'

import Vue from './runtime/index'

import { query } from './util/index'

import { compileToFunctions } from './compiler/index'

import { shouldDecodeNewlines, shouldDecodeNewlinesForHref } from './util/compat'

const idToTemplate = cached(id => {

const el = query(id)

return el && el.innerHTML

})

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)

}

/\*\*

\* Get outerHTML of elements, taking care

\* of SVG elements in IE as well.

\*/

function getOuterHTML (el: Element): string {

if (el.outerHTML) {

return el.outerHTML

} else {

const container = document.createElement('div')

container.appendChild(el.cloneNode(true))

return container.innerHTML

}

}

Vue.compile = compileToFunctions

export default Vue

那么，当我们的代码执行 import Vue from 'vue' 的时候，就是从这个入口执行代码来初始化 Vue，  
那么 Vue 到底是什么，它是怎么初始化的，我们来一探究竟。

## **Vue 的入口**

在这个入口 JS 的上方我们可以找到 Vue 的来源：import Vue from './runtime/index'，我们先来看一下这块儿的实现，它定义在 src/platforms/web/runtime/index.js 中：

import Vue from 'core/index'

import config from 'core/config'

import { extend, noop } from 'shared/util'

import { mountComponent } from 'core/instance/lifecycle'

import { devtools, inBrowser, isChrome } from 'core/util/index'

import {

query,

mustUseProp,

isReservedTag,

isReservedAttr,

getTagNamespace,

isUnknownElement

} from 'web/util/index'

import { patch } from './patch'

import platformDirectives from './directives/index'

import platformComponents from './components/index'

// install platform specific utils

Vue.config.mustUseProp = mustUseProp

Vue.config.isReservedTag = isReservedTag

Vue.config.isReservedAttr = isReservedAttr

Vue.config.getTagNamespace = getTagNamespace

Vue.config.isUnknownElement = isUnknownElement

// install platform runtime directives & components

extend(Vue.options.directives, platformDirectives)

extend(Vue.options.components, platformComponents)

// install platform patch function

Vue.prototype.\_\_patch\_\_ = inBrowser ? patch : noop

// public mount method

Vue.prototype.$mount = function (

el?: string | Element,

hydrating?: boolean

): Component {

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

return mountComponent(this, el, hydrating)

}

// ...

export default Vue

这里关键的代码是 import Vue from 'core/index'，之后的逻辑都是对 Vue 这个对象做一些扩展，可以先不用看，我们来看一下真正初始化 Vue 的地方，在 src/core/index.js 中：

import Vue from './instance/index'

import { initGlobalAPI } from './global-api/index'

import { isServerRendering } from 'core/util/env'

import { FunctionalRenderContext } from 'core/vdom/create-functional-component'

initGlobalAPI(Vue)

Object.defineProperty(Vue.prototype, '$isServer', {

get: isServerRendering

})

Object.defineProperty(Vue.prototype, '$ssrContext', {

get () {

/\* istanbul ignore next \*/

return this.$vnode && this.$vnode.ssrContext

}

})

// expose FunctionalRenderContext for ssr runtime helper installation

Object.defineProperty(Vue, 'FunctionalRenderContext', {

value: FunctionalRenderContext

})

Vue.version = '\_\_VERSION\_\_'

export default Vue

这里有 2 处关键的代码，import Vue from './instance/index' 和 initGlobalAPI(Vue)，初始化全局 Vue API（我们稍后介绍），我们先来看第一部分，在 src/core/instance/index.js 中：

### **Vue 的定义**

import { initMixin } from './init'

import { stateMixin } from './state'

import { renderMixin } from './render'

import { eventsMixin } from './events'

import { lifecycleMixin } from './lifecycle'

import { warn } from '../util/index'

function Vue (options) {

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

!(this instanceof Vue)

) {

warn('Vue is a constructor and should be called with the `new` keyword')

}

this.\_init(options)

}

initMixin(Vue)

stateMixin(Vue)

eventsMixin(Vue)

lifecycleMixin(Vue)

renderMixin(Vue)

export default Vue

在这里，我们终于看到了 Vue 的庐山真面目，它实际上就是一个用 Function 实现的类，我们只能通过 new Vue 去实例化它。

有些同学看到这不禁想问，为何 Vue 不用 ES6 的 Class 去实现呢？我们往后看这里有很多 xxxMixin 的函数调用，并把 Vue 当参数传入，它们的功能都是给 Vue 的 prototype 上扩展一些方法（这里具体的细节会在之后的文章介绍，这里不展开），Vue 按功能把这些扩展分散到多个模块中去实现，而不是在一个模块里实现所有，这种方式是用 Class 难以实现的。这么做的好处是非常方便代码的维护和管理，这种编程技巧也非常值得我们去学习。

### initGlobalAPI

Vue.js 在整个初始化过程中，除了给它的原型 prototype 上扩展方法，还会给 Vue 这个对象本身扩展全局的静态方法，它的定义在 src/core/global-api/index.js 中：

export function initGlobalAPI (Vue: GlobalAPI) {

// config

const configDef = {}

configDef.get = () => config

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

configDef.set = () => {

warn(

'Do not replace the Vue.config object, set individual fields instead.'

)

}

}

Object.defineProperty(Vue, 'config', configDef)

// exposed util methods.

// NOTE: these are not considered part of the public API - avoid relying on

// them unless you are aware of the risk.

Vue.util = {

warn,

extend,

mergeOptions,

defineReactive

}

Vue.set = set

Vue.delete = del

Vue.nextTick = nextTick

Vue.options = Object.create(null)

ASSET\_TYPES.forEach(type => {

Vue.options[type + 's'] = Object.create(null)

})

// this is used to identify the "base" constructor to extend all plain-object

// components with in Weex's multi-instance scenarios.

Vue.options.\_base = Vue

extend(Vue.options.components, builtInComponents)

initUse(Vue)

initMixin(Vue)

initExtend(Vue)

initAssetRegisters(Vue)

}

这里就是在 Vue 上扩展的一些全局方法的定义，Vue 官网中关于全局 API 都可以在这里找到，这里不会介绍细节，会在之后的章节我们具体介绍到某个 API 的时候会详细介绍。有一点要注意的是，Vue.util 暴露的方法最好不要依赖，因为它可能经常会发生变化，是不稳定的。

## **总结**

那么至此，Vue 的初始化过程基本介绍完毕。这一节的目的是让同学们对 Vue 是什么有一个直观的认识，它本质上就是一个用 Function 实现的 Class，然后它的原型 prototype 以及它本身都扩展了一系列的方法和属性，那么 Vue 能做什么，它是怎么做的，我们会在后面的章节一层层帮大家揭开 Vue 的神秘面纱。

****任务****

请仔细阅读本文档，为下面学习视频内容做准备