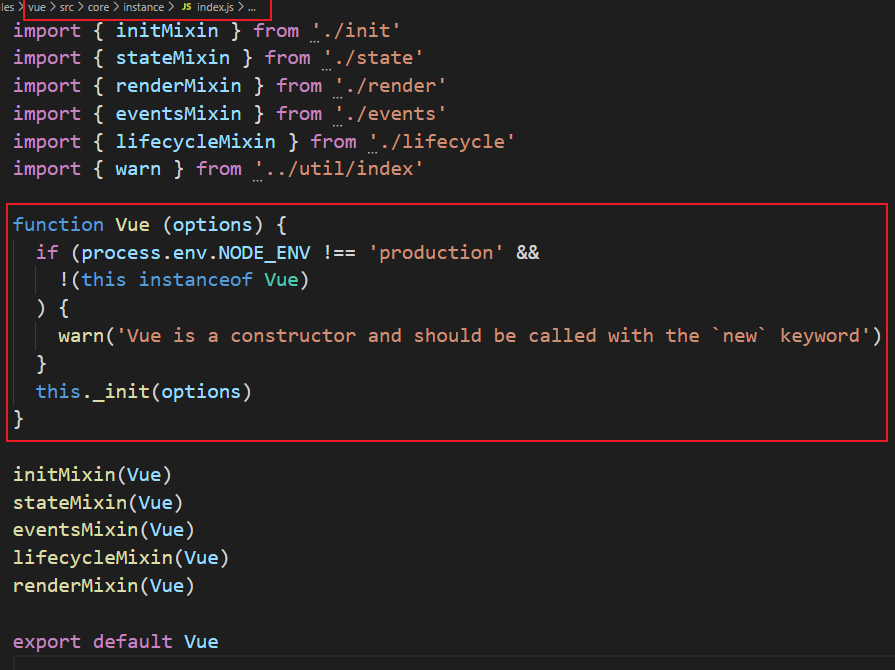
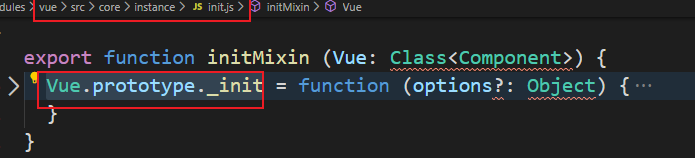
**new Vue过程**



**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。

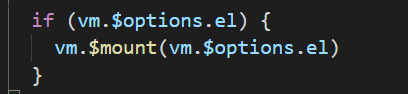
**this.\_init(options)调用原型方法（初始化）**



Vue 初始化主要就干了几件事情，合并配置，初始化生命周期，初始化事件中心，初始化渲染，初始化 data、props、computed、watcher 等等。

**总结：**Vue 初始化通过initMixin(Vue)，在Vue上挂载一个\_init方法，new Vue时调用这个方法，在\_init中，先合并options到$options，接下来就初始化生命周期，初始化事件中心，初始化渲染，初始化 data、props、computed、watcher 等等。在initState(vm)时，它初始化了props，methods，data。初始化data调用initData(),在initData中，首先检测了method和props与data命名是否冲突，然后通过proxy代理使其this.xxx能得到this.\_data.xxx。最后监听data。 \_init最后进行挂载。

**初始化的最后进行挂载**

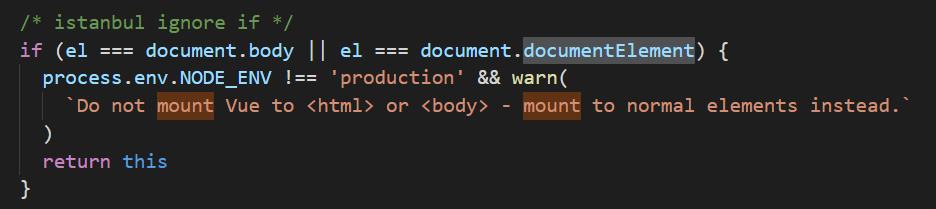


在初始化的最后，检测到如果有 el 属性，则调用 vm.$mount 方法挂载 vm，挂载的目标就是把模板渲染成最终的 DOM，那么接下来我们来分析 Vue 的挂载过程。

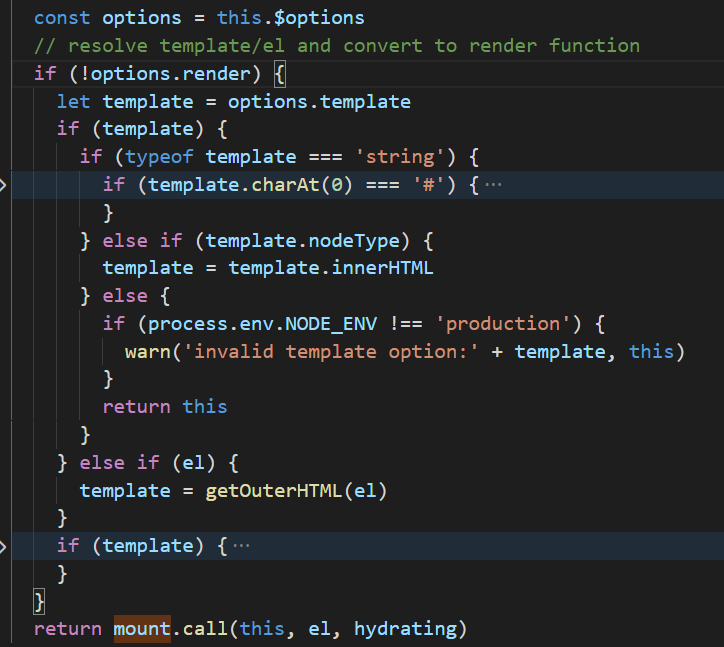
**挂载过程（compiler 版本的 $mount）**



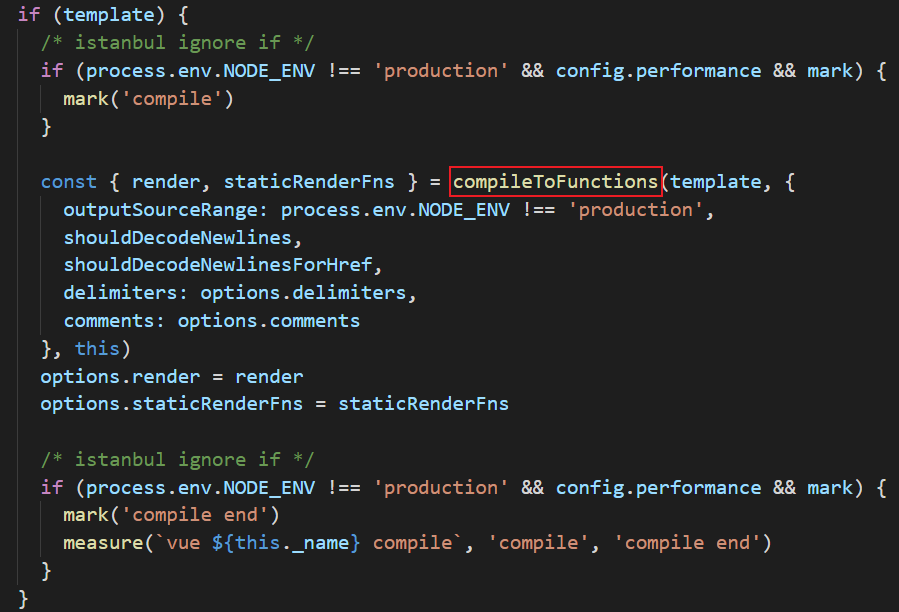
首先缓存了原型上的 $mount 方法,在重新定义该方法。



Vue 不能挂载在 body、html 这样的根节点上。



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。如果有template就通过template转换，否则通过el转换为template后再转换。

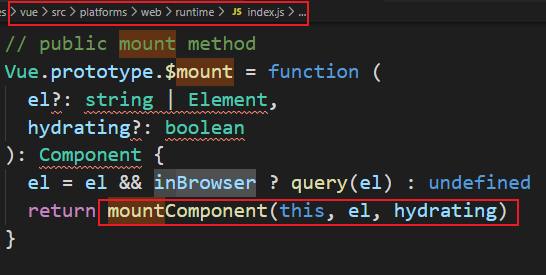


在 Vue 2.0 版本中，所有 Vue 的组件的渲染最终都需要 render 方法，无论我们是用单文件 .vue 方式开发组件，还是写了 el 或者 template 属性，最终都会转换成 render 方法，那么这个过程是 Vue 的一个“在线编译”的过程，它是调用 compileToFunctions 生成render函数。



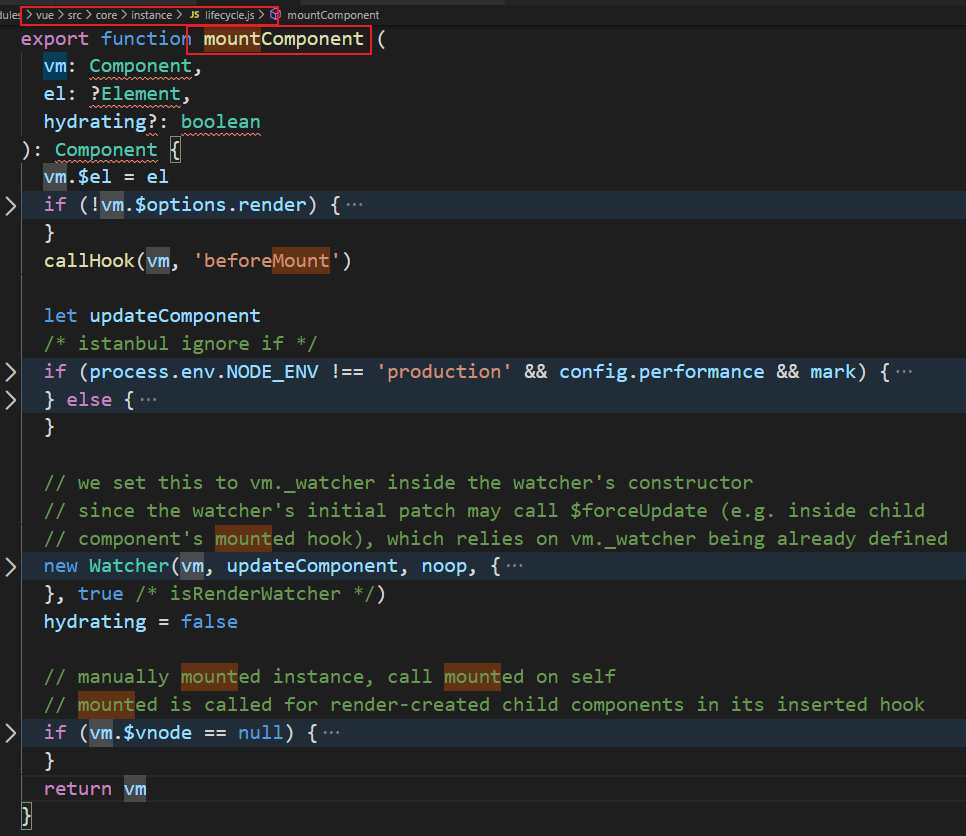
最后，调用原先原型上的 $mount 方法挂载。

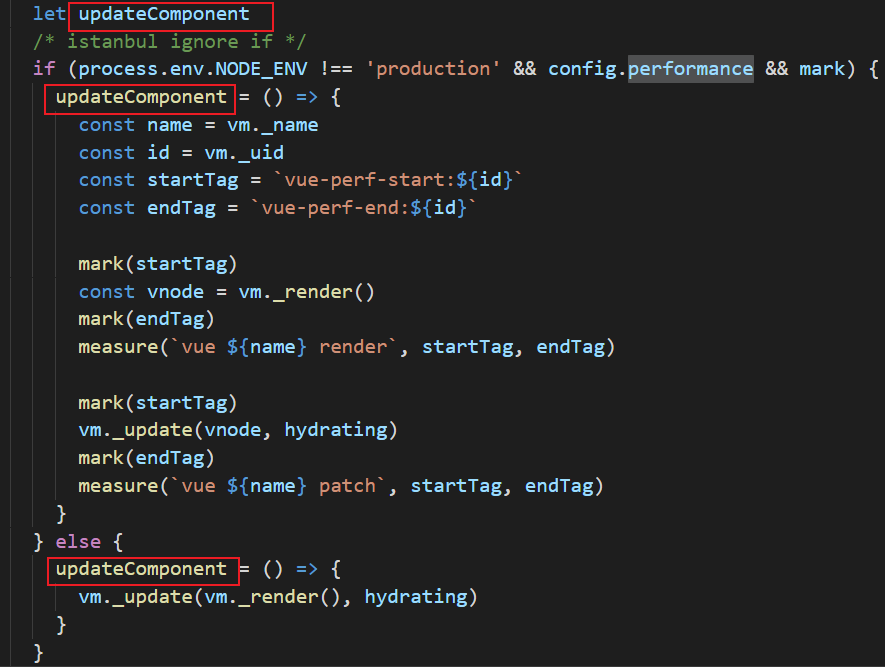
**原先的$ mount**



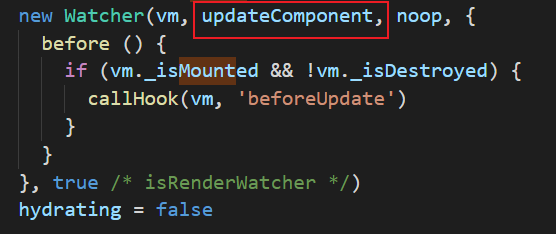
第一个是 el，它表示挂载的元素，可以是字符串，也可以是 DOM 对象，如果是字符串在浏览器环境下会调用 query 方法转换成 DOM 对象的。第二个参数是和服务端渲染相关，在浏览器环境下我们不需要传第二个参数。

调用mountComponent方法

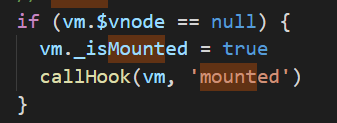




updateComponent方法中调用 vm.\_render 方法先生成虚拟 Node，最终调用 vm.\_update 更新 DOM。

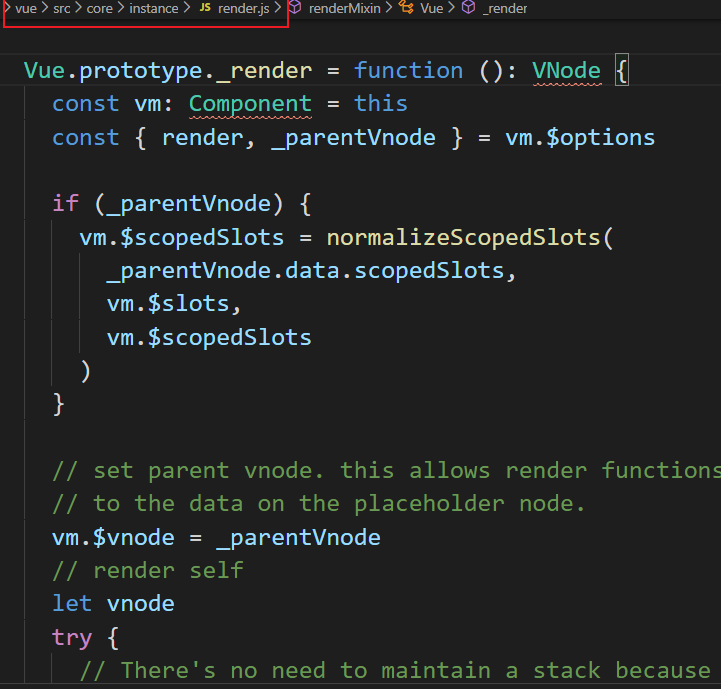


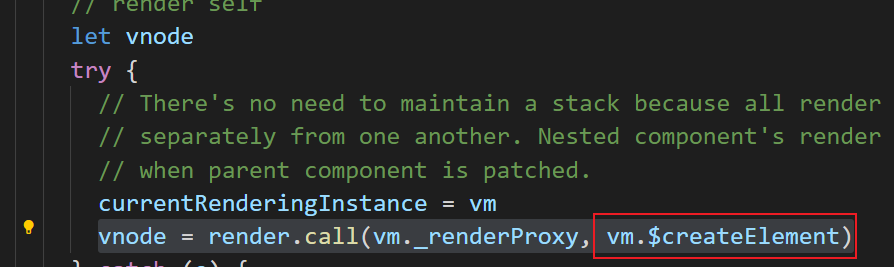
Watcher 在这里起到两个作用，一个是初始化的时候会执行回调函数，另一个是当 vm 实例中的监测的数据发生变化的时候执行回调函数。



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。

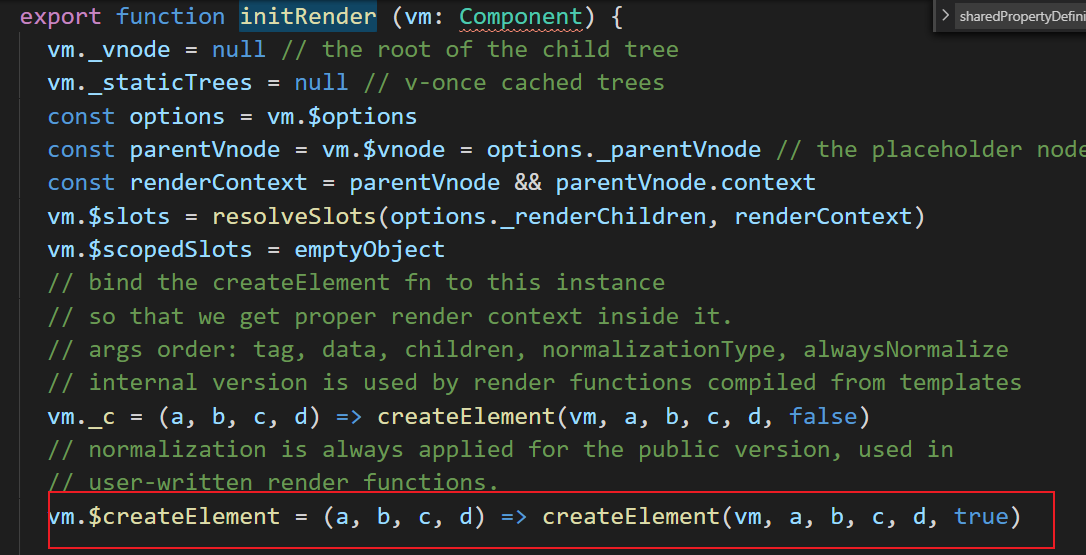
**vm.\_render（用户来生成虚拟dom）**





**通过用户写的render函数或者编译生成的render函数来生成vnode，**

**$createElement是在\_init中initRender生成的**



vm.\_c 方法，它是被模板编译成的 render 函数使用，而 vm.$createElement 是用户手写 render 方法使用的，