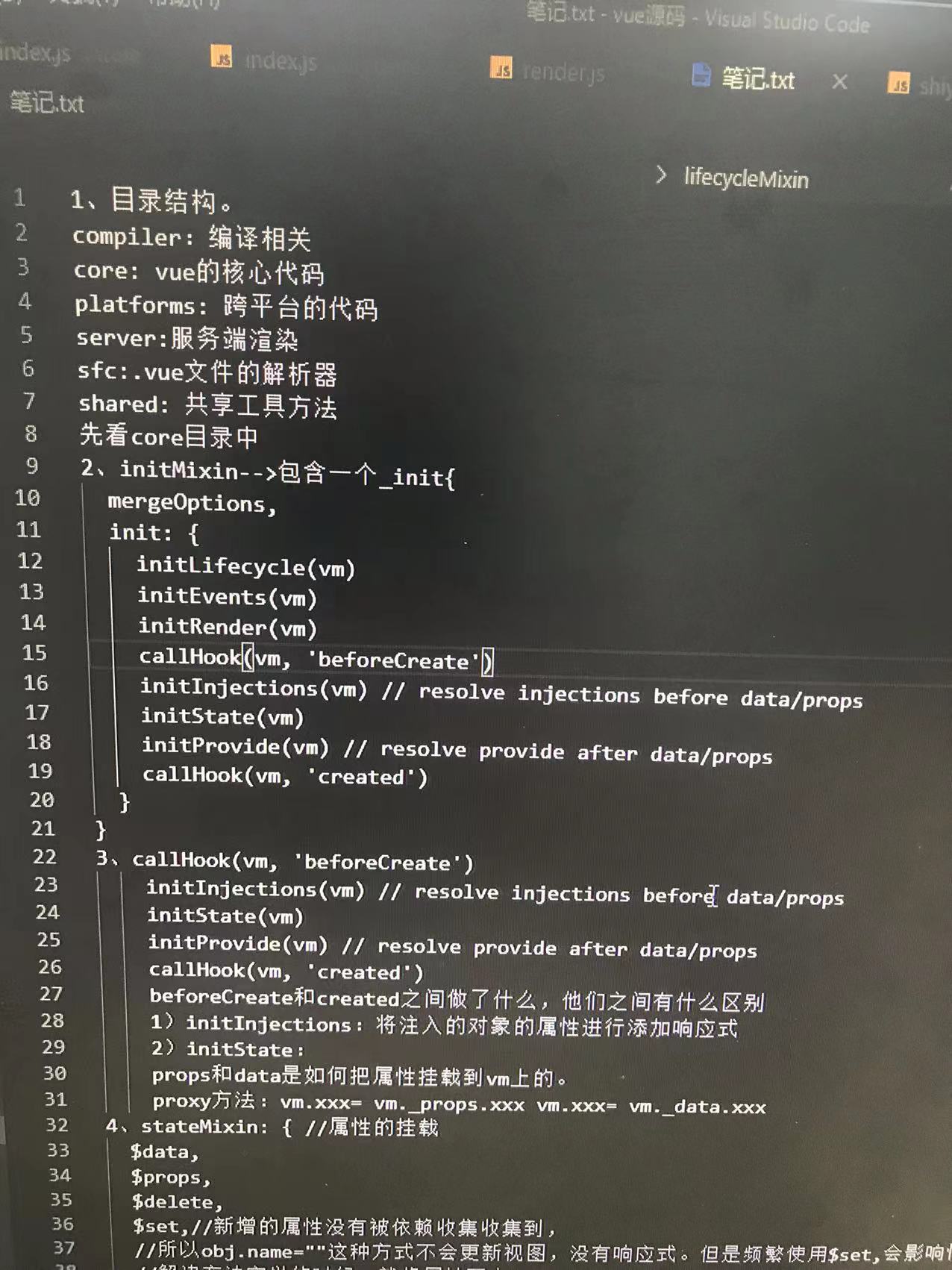
1. 整体目录结构



\_init函数中：

设置vm = this;

1）vm.uid----递增，每创建一个实例就增加1

2）标记程序执行的时间mark()函数

3）vm.\_isVue = true;

4）vm.$options

5）vm.renderProxy

6）initLifecycle(vm)

initEvents(vm)

initRender(vm)

callHook(vm, 'beforeCreate')

initInjections(vm) // resolve injections before data/props

initState(vm)

initProvide(vm) // resolve provide after data/props

callHook(vm, 'created')

7）$mount()其定义在entry-runtime-with-compiles.js中。

无论是采用el,render函数，还是template方式去写，最终都会被转换为render函数。挂载到options上面

编译过程：compileToFunctions=🡺

最终调用mountComponent方法，实现挂载：

{

beforeMount:,

new Watcher()🡪updateComponent {

vm.\_update()://更新，生成实际的DOM{

vm.\_\_patch\_\_: {

createPatchFunction{

patchVnode: //

createElm: {

native-createElement--生成实际的dom节点

createChildren:遍历子节点

}

}

}

}

vm.\_render(): //生成虚拟DOM,定义在instance的render.js中{

createElement{

notmalizeChildren: //children统一为vnode，

new Vnode: {

tag：为字符串 {

内置节点：直接创建普通的vnode,

已注册的组件名称：创建组件类型的方式-createComponents

未知节点：创建了一个未知标签类型的节点

直接创建组件

}

tag为component:

}

}

}

}

mounted: ,

}

