在global-api/index.js:

Vue.set = set

Vue.delete = del

Vue.nextTick = nextTick

Vue.options = Object.create(null)

ASSET\_TYPES.forEach(type => { //'component','directive','filter'

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

})

Vue.options.\_base = Vue

extend(Vue.options.components, builtInComponents)

initUse(Vue)

initMixin(Vue)

initExtend(Vue)

initAssetRegisters(Vue)

Set:target不能是 Vue 实例，或者 Vue 实例的根数据对象。 这个方法主要用于避开 Vue 不能检测属性被添加的限制。

Vue.set = set

Vue.prototype.$set = set 在state.js中

export function set (target: Array<any> | Object, key: any, val: any): any {

//判断target是不是null，undefined，根数据，实例

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

(isUndef(target) || isPrimitive(target))

) {

warn(`Cannot set reactive property on undefined, null, or primitive value: ${(target: any)}`)

}

//target是不是array

if (Array.isArray(target) && isValidArrayIndex(key)) {

target.length = Math.max(target.length, key)

target.splice(key, 1, val)

return val

}

//target中有没有key这个属性，且key不是Object的原型方法或属性

if (key in target && !(key in Object.prototype)) {

target[key] = val

return val

}

const ob = (target: any).\_\_ob\_\_ //observer对象

//target是不是this.$data

if (target.\_isVue || (ob && ob.vmCount)) {

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

'Avoid adding reactive properties to a Vue instance or its root $data ' +

'at runtime - declare it upfront in the data option.'

)

return val

}

if (!ob) {

target[key] = val

return val

}

defineReactive(ob.value, key, val)

ob.dep.notify()

return val

}

问题，为什么Vue 不能检测属性被添加，set能。

defineReactive方法。Object.defineProperty的set方法在属性添加的时候不触发。

set方法手动触发ob.dep.notify()。

实例：

1.data(){

message:['ss','asd'],

}

created:function(){

this.$set(this.message,2,'bar') //{{ message }}=>'ss','asd',’bar’

},

methods:{

ons(){ //绑定点击事件

this.message.push("bar")//{{ message }}=>'ss','asd',’bar’

this.$set(this.message,2,'bar')//{{ message }}=>'ss','asd',’bar’

}

},

2.Vue.set(Vue,"foo",'bar');

console.log(this.\_\_proto\_\_.constructor.foo) //bar

3.data:

message:{"constructor":'123'},

created:function(){

this.$set(this.message,"foo",'bar') //{{ message }}=> {"constructor":'123',"foo":'bar'}

this.$set(this.message,"constructor",'bar')//{{ message }}=>{"constructor":'123'},

},

methods:{

ons(){

this.message.aa = '123' //{{ message }}=>{"constructor":'123'},

this.$set(this.message,"constructor",'bar') //{{ message }}=>{"constructor":'123'},

this.$set(this.message,"constructor1",'bar') //{{ message }}=>{"constructor":'123',"constructor1":'bar'},

}

},

Del和set相反，可以自己看代码

NextTick:将回调延迟到下次 DOM 更新循环之后执行。在修改数据之后立即使用它，然后等待 DOM 更新

Vue.nextTick和vm.$nextTick不同的是回调的 this 自动绑定到调用它的实例上。

实例：

methods:{

ons(){

this.message = '1111'

console.log(this.$el.textContent+1)

this.$nextTick(function(){

console.log(this.$el.textContent+2)

})

let self = this;

Vue.nextTick(function(){

console.log(self.$el.textContent+6)

})

console.log(this.$el.textContent+3)

}

},

beforeUpdate:function(){

console.log(this.$el.textContent+4)

},

updated:function(){

console.log(this.$el.textContent+5)

}

页面输出:

ss  
 123 A custom component!A custom child !A custom child1 !1  
 ss  
 123 A custom component!A custom child !A custom child1 !3  
 ss  
 123 A custom component!A custom child !A custom child1 !4  
 1111  
 123 A custom component!A custom child !A custom child1 !5  
 1111  
 123 A custom component!A custom child !A custom child1 !2  
 1111  
 123 A custom component!A custom child !A custom child1 !6

Vue.nextTick()的this要指定。nextTick是在 updated钩子之后执行的

源码：

Vue.nextTick = nextTick

Vue.prototype.$nextTick = function (fn: Function) {

return nextTick(fn, this)

}

export function nextTick (cb?: Function, ctx?: Object) {

let \_resolve

callbacks.push(() => {

if (cb) {

try {

cb.call(ctx)

} catch (e) {

handleError(e, ctx, 'nextTick')

}

} else if (\_resolve) {

\_resolve(ctx)

}

})

if (!pending) {

pending = true

if (useMacroTask) {

macroTimerFunc()

} else {

microTimerFunc()

}

}

// $flow-disable-line

if (!cb && typeof Promise !== 'undefined') {

return new Promise(resolve => {

\_resolve = resolve

})

}

}

macroTimerFunc:宏观 先setImmediate（IE10，node 可以），后setTimeout

microTimerFunc:微观 Promise.resolve()

js的事件循环机制 <https://jakearchibald.com/2015/tasks-microtasks-queues-and-schedules/>

Use：安装 Vue.js 插件。如果插件是一个对象，必须提供 install 方法。如果插件是一个函数，它会被作为 install 方法。install 方法调用时，会将 Vue 作为参数传入。 当 install 方法被同一个插件多次调用，插件将只会被安装一次。

Vue.use(myPlugins)

源码：

Vue.use = function (plugin: Function | Object) {

//判断是否是同名插件

const installedPlugins = (this.\_installedPlugins || (this.\_installedPlugins = []))

if (installedPlugins.indexOf(plugin) > -1) {

return this

}

// additional parameters

const args = toArray(arguments, 1)

args.unshift(this)

//执行你的插件

if (typeof plugin.install === 'function') {

plugin.install.apply(plugin, args)

} else if (typeof plugin === 'function') {

plugin.apply(null, args)

}

//加入数组，方便判断

installedPlugins.push(plugin)

return this

}

Mixin:

使用

全局注册

Vue.mixin({ //全局mixin，之后所以的Vue实例都有

created:function(){

console.log(1)

}

}

局部注册

let myMinix = {

created:function(){

console.log(1)

}

}

mixins:[myMinix]

源码：

在global-api/mixin.js

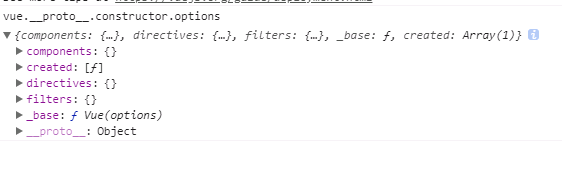
Vue.mixin = function (mixin: Object) {

this.options = mergeOptions(this.options, mixin)

return this

}

把mixin的东西合并到Vue.options对象上



这个对象是每次初始化vue实例的 parent，也是下面的parent，所以是对所以的Vue实例都有影响。

在util/options.js

if (child.mixins) {

for (let i = 0, l = child.mixins.length; i < l; i++) {

parent = mergeOptions(parent, child.mixins[i], vm)

}

}

把mixin的东西合并到$options对象上

Extend:

使用：

var Child1 = Vue.extend({

template: '<div>A custom child1 !</div>',

data(){

return{

message:'ssss1'

}

},

})

Child1// ƒ VueComponent(options) {  
 this.\_init(options);  
 }

new Child1() //vue实例

源码：golbal-api/extend.js

初始化Sub，最后返回Sub（一个跟Vue差不多，合并了自己option的函数）

const Sub = function VueComponent (options) {

this.\_init(options)

}

Sub.prototype = Object.create(Super.prototype)

Sub.prototype.constructor = Sub

Sub.cid = cid++

//合并options

Sub.options = mergeOptions(

Super.options,

extendOptions

)

Sub['super'] = Super

//往Sub的原型上添加????

if (Sub.options.props) {

initProps(Sub)

}

if (Sub.options.computed) {

initComputed(Sub)

}

Sub.extend = Super.extend

Sub.mixin = Super.mixin

Sub.use = Super.use

ASSET\_TYPES.forEach(function (type) {

Sub[type] = Super[type]

})

// enable recursive self-lookup

if (name) {

Sub.options.components[name] = Sub

}

return Sub

Filter,componment,directive:

使用:

Vue,Filter(‘my-filter’,function(){})

Vue,directive(‘my-directive’,function(){})

Vue,componment(‘my-componment’,{name:’ds’,...})

源码：

ASSET\_TYPES.forEach(type => { //ASSET\_TYPES= [Filter,componment,directive]

Vue[type] = function (

id: string,

definition: Function | Object

): Function | Object | void {

//如果没有定义，就是第二个参数没有传函数或对象，返回对应属性的值

if (!definition) {

return this.options[type + 's'][id]

} else {

//组件名字的检查

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

validateComponentName(id)

}

// this.options.\_base是基本的Vue函数，把自己写的东西合并上去。

if (type === 'component' && isPlainObject(definition)) {

definition.name = definition.name || id

definition = this.options.\_base.extend(definition)

}

//如果是指令，且参数是一个函数，则默认绑定在bind和update上

if (type === 'directive' && typeof definition === 'function') {

definition = { bind: definition, update: definition }

}

//最后都绑定在options上

this.options[type + 's'][id] = definition

return definition

}

}

})