provide和inject的用法和原理简单分析

官网说明：

这对选项需要一起使用，以允许一个祖先组件向其所有子孙后代注入一个依赖，不论组件层次有多深，并在起上下游关系成立的时间里始终生效。

实例

html部分

<div id="app">

<component v-bind:is="myComponent"></component>

</div>

js部分

var Child1 = {

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

data(){

return{

message:'ssss1'

}

},

inject: ['foo'],

}

var Child = {

template: '<div>A custom child !{{ foo }}<child1></child1></div>',

data(){

return{

message:'ssss1'

}

},

inject: ['foo'],

components:{

child1:Child1

}

}

var vue = new Vue({

el:"#app",

name:'ss',

data(){

return {

message:'ss',

myComponent:"myComponent"

}

},

provide: {

foo: 'bar'

},

components: {

// <my-component> 将只在父组件模板中可用

'myComponent': {

data(){

return{

message:'123',

ssss:'sssss'

}

},

template: '<div>A custom component!<child></child></div>',

components:{

child:Child

},

}

}

})

页面显示

A custom component!

A custom child !bar

A custom child1 !bar

简单说明：1个Vue()实例，2个VueComponent实例，child1是child的子组件，child是vue的子组件。在vue里provide一个对象，里面有一个foo属性，在child和child1中inject: ['foo']即可调动。

源码简单分析：

在instance的inject.js中

provide：

export function initProvide (vm: Component) {

const provide = vm.$options.provide

if (provide) {

vm.\_provided = typeof provide === 'function'

? provide.call(vm)

: provide

}

}

把vm.$options.provide的东西，如果是函数，执行下，不是就直接绑在 vm.\_provided上。至于vm.$options.provide怎么来的，在instance/init.js中，可以自己去看。（vm.$options.inject也一样）主要在util/options.js里把写的options和vm的构造函数，合并一下。

Injections：

export function initInjections (vm: Component) {

const result = resolveInject(vm.$options.inject, vm) //找vm上的\_provided的值去匹配，返回一个对象

if (result) {

toggleObserving(false) ???

Object.keys(result).forEach(key => {

/\* istanbul ignore else \*/

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

defineReactive(vm, key, result[key], () => { //vm.key = value

warn(

`Avoid mutating an injected value directly since the changes will be ` +

`overwritten whenever the provided component re-renders. ` +

`injection being mutated: "${key}"`,

vm

)

})

} else {

defineReactive(vm, key, result[key])

}

})

toggleObserving(true) ？？？

}

}

每次初始化Vue对象的时候，把provide的对象绑定到vm.\_provided上，把inject的值，和vm.\_provided匹配定义带实例上。

问题 ，为什么每个vm对象都存在\_provided么，但是我只是在祖先上写了

resolveInject函数里

while (source) {

if (source.\_provided && hasOwn(source.\_provided, provideKey)) {

result[key] = source.\_provided[provideKey]

break

}

source = source.$parent

}

代码验证

在 myComponent中加入下面代码

provide: {

foo: 'bar1'

},

页面显示

A custom component!

A custom child !bar1

A custom child1 !bar1

实现A→B->C传值还可以用$attrs和$listener