

# vue源码

## Vue3 MVVM - 2. 收集依赖

承接上一章我们讨论了 reactive 和 proxy 的原理以及使用方法，我们再最后有提出 watchEffect(()=>{}) 是如何在每次响应式对象的属性更新会自动调用内部的函数？我们在本章来讲实现原理以及源码实现。

首先，vue3 源码 reactive 这个 package 里面有2个东西，一个是 reactive 还有一个是 effect。是的 watchEffect 源码内部调用的就是 effect 来实现的。那么我们需要看懂 effect 的原理就可以知道 watchEffect 是如何运行的。

首先 effect 和 watchEffect 一样，接受一个函数为参数，这个函数内部所有使用到的 reactive 响应式对象的属性，都会在被更新的同时自动调用这个函数。这里的更新的意思就是当给这个对象赋新值的时候。

回想上一章 reactive，是不是 proxy 那里我们有 set 方法，此方法会被调用当我们给一个数据更新。那么 proxy 的 set 方法那里我们调用 watchEffect(()=>{}) 里面的函数不就可以实现这个功能了么？

```
1 let a = {
2   'abc': 123,
3   'abcd': 456
4 }
5 let b = reactive(a)
6
7 watchEffect(()=>{
8   return a.abc + a.abcd
9 })
```

上面的代码在 vue3 可以在每次 a.abc 或者 a.abcd 数据更新，会自动调用给 watchEffect 作参数的函数，也就是，

```
1 ()=>{ return a.abc + a.abcd }
```

重点来了，我们仔细看这个函数内部是不是有使用 a 对象的属性 abc 和 abcd，这里的使用换一个词不就是访问么？那么回想上一章 reactive 当我们访问对象的某个属性，proxy 里面有一个 get 方法会被调用，那我们在那里不就可以知道 watchEffect(()=>{...}) 内部使用了哪几个属性了么？

这也解释了 vue3 文档里面描述的。所有 watchEffect(()=>{...}) 都会自动被调用一次，其目的就是为了找出对象上面的哪几个属性在更新之后需要调用这个 watchEffect 函数。

那么换一句话说就是，我们需要组件初始化自动调用一次 watchEffect 函数，在 proxy 里面的 get 方法确认使用了哪几个对象属性，然后收集起来这些个属性。

当我们在 proxy set 更新对象属性值，查看属性值是否有收集过对应的 watchEffect 函数，如果有我们就一个一个运行，这样就可以解决我们的问题了。那么我们把我们的逻辑和上一节用的代码实现出来，

```
1 let activeCallback
2
```

```

1  function effect(callback) {
2      activeCallback = callback
3      callback()
4      activeCallback = null
5  }
6
7
8  function watchEffect(callback) {
9      effect(callback)
10 }
11
12 let maps = new Weakmap()
13
14 function collectDependencies(target, key) {
15     if (!activeCallback) return
16     let dependency
17     let dependencyCallbacks
18
19     let m = maps.get(target)
20     if (!m) {
21         maps.set(target, (m = new Map()))
22     }
23     let mProperty = m.get(key)
24     if (!mProperty) {
25         m.set(key, (mProperty = new Set()))
26     }
27     mProperty.add(activeCallback)
28 }
29

```

我们看一看上面的代码，数据结构是这样的，weakmap -> map -> set，

1. weakmap的key就是reactive对象（target），值就是一个map，
2. map里面的key 就是在watchCallback 里面使用过的对象属性，值是一个set为了防止重复添加
3. set里面放置的就是 watchCallback()=>{} 里面的 ()=>

这个就是经典的vue3 依赖收集，在proxy 里面的get 收集，在 set里面调用对应的函数。那么我们现在来完成上一节的reactive函数，即添加收集依赖和调用依赖的代码。

```

1  let activeCallback
2
3  function effect(callback) {
4      activeCallback = callback
5      callback()
6      activeCallback = null
7  }
8
9  function watchEffect(callback) {
10     effect(callback)
11 }
12
13

```

```

    let maps = new Weakmap()
14
15 function collectDependencies(target, key) {
16     if (!activeCallback) return
17     let dependency
18     let dependencyCallbacks
19
20     let m = maps.get(target)
21     if (!m) {
22         maps.set(target, (m = new Map()))
23     }
24     let mProperty = m.get(key)
25     if (!mProperty) {
26         m.set(key, (mProperty = new Set()))
27     }
28     mProperty.add(activeCallback)
29 }
30
31 function trigger(target, key) {
32     const depsMap = maps.get(target);
33     if (!depsMap) return;
34     const dep = depsMap.get(key);
35     if (dep) {
36         dep.forEach((effect) => effect());
37     }
38 }
39
40
41 function reactive(obj) {
42     // 如果不是对象，直接返回
43     if (!(obj !== null && typeof obj === 'object')) {
44         return obj
45     }
46
47
48
49     // 如果是对象，创建对象proxy的handler
50     const handler = {
51         get(target, key, receiver) {
52
53             collectDependencies(target, key)
54             result = Reflect.get(target, key, receiver)
55             console.log(target, key)
56             return clean(result)
57
58         },
59
60         set(target, key, value, receiver) {
61             console.log('set-----', target, key, value)
62             let result = false
63             result = Reflect.set(target, key, value, receiver)
64             trigger(target, key)
65             return result

```

```

66     },
67
68     deleteProperty(target, key) {
69         console.log('delete-----', target, key)
70
71         const result = Reflect.deleteProperty(target, key)
72         if (result && Object.prototype.hasOwnProperty.call(target, key)) {
73             trigger(target, key)
74         }
75         return result
76     }
77 }
78
79 return new Proxy(obj, handler)
80 }

```

实现的过程总结一下，首先运行一次watchEffect里面的函数(vue 里面的watchEffect函数都会自动先运行一次)，所有里面使用过的对象的属性会在proxy 里面的get里面收集依赖，这里的依赖就是watchEffect里面的函数。当为这个对象里面的属性赋值的则 proxy 里面的set会调用该属性对应的收集的方法。

这个是不是就是观察者/发布订阅模式？