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？我们在本章来讲述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。那么我们需要弄懂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其更新，自动调用这个函数。那么说到这里是否想到 更新 这个词好像就是当我们给这个对象赋值的时候，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响应式数据更新。那么proxy 的set 方法那里我们调用 watchEffect(()=>{}) 里面的函数不就好了么？

说到这里，是不是有个问题，就是我们则么知道哪几个对象的属性在 watchEffect 有被使用？我们用代码来演示一下我们现在问题，

```
let a = {
    'abc': 123,
    'abcd': 456
}
let b = reactive(a)

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

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watchEffect 内部的函数 也就是 ()=>{ 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(()=>{...}) 内部使用了哪几个函数了么。

那么换一句话说就是，我们需要调用一次watchEffect里面的函数，在proxy 里面的get 方法确认使用了哪几个，然后收集起来这个属性，然后让这个属性和这个watchEffect 里面的函

数一一对应。然后当我们在proxy set 更新对象属性值，查看属性值是否有对应的函数，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，

```
let activeCallback
function effect(callback) {
    activeCallback = callback
    callback()
    activeCallback = null
}

function watchEffect(callback) {
    effect(callback)
}

let maps = new Weakmap()

function collectDependencies(target, key) {
    if (!activeCallback) return
    let dependency
    let dependencyCallbacks

    let m = maps.get(target)
    if (!m) {
        maps.set(target, (m = new Map()))
    }
    let mProperty = m.get(key)
    if (!mProperty) {
        m.set(key, (mProperty = new Set()))
    }
    mProperty.add(activeCallback)
}

```

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 里面使用的reactive对象的属性，值是一个set
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。

```
let activeCallback

function effect(callback) {
    activeCallback = callback
    callback()
    activeCallback = null
}

function watchEffect(callback) {
    effect(callback)
}

let maps = new Weakmap()

function collectDependencies(target, key) {
    if (!activeCallback) return
    let dependency
    let dependencyCallbacks

    let m = maps.get(target)
    if (!m) {
        maps.set(target, (m = new Map()))
    }
    let mProperty = m.get(key)
    if (!mProperty) {
        m.set(key, (mProperty = new Set()))
    }
    mProperty.add(activeCallback)
}

function trigger(target, key) {
  const depsMap = maps.get(target);
  if (!depsMap) return;
  const dep = depsMap.get(key);
  if (dep) {
    dep.forEach((effect) => effect());
  }
}


function reactive(obj) {
     // 如果不是对象，直接返回
    if (!(obj!==null && typeof obj =='object')) {
        return obj
    }



    // 如果是对象，
```

```
    const handler = {
        get(target, key,receiver) {

            collectDependencies(target, key)
            result = Reflect.get(target,key,receiver)
            console.log(target, key)
            return clean(result)

        },

        set(target,key,value,receiver) {
            console.log('set--------', target, key, value)
            let result = false
            result = Reflect.set(target,key,value,receiver)
            trigger(target, key)
            return result
        },

        deleteProperty(target, key) {
            console.log('delete--------', target, key)

            const result = Reflect.deleteProperty(target, key)
            if (result && Object.prototype.hasOwnProperty.call(target, key
                trigger(target, key)
            }
            return result
        }
    }

    return new Proxy(obj, handler)
}
```

实现的过程总结一下，首先运行一次watchEffect里面的函数，所有里面使用过的reactive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reactive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方法。