Sunday • 更新于 2022-10-19

上一节 01：前言 03：框架实现：... 下一节

02：源码阅读：reactive 的响应性，跟踪 Vue 3 源码实现逻辑

我们知道在 vue 中想要实现响应式数据，拥有两种方式：

1. reactive
2. ref

在第三章中，我们在 vue 的源码中，创建了 packages/vue/examples/imooc/reactive.html 测试实例，在该实例中，我们通过 reactive 方法声明了一个响应式数据，通过 effect 注册了一个函数。

那么下面，我们就 **跟踪** 当前的代码，来详细看一下 vue 内容到底做了什么？

看的过程中我们需要时刻记住两点主线：

1. reactive 做了什么？
2. effect 是什么？

明确好了之后，那么下面我们来去看：

reactive 方法

1. 触发 reactive 方法
2. 创建 reactive 对象： return createReactiveObject
3. 进入 new Proxy
    1. 第一个参数 target：为传入的对象
    2. 第二个参数 handler： TargetType.COLLECTION = 2， targetType = 1，所以 handler 为 baseHandlers
    3. 那这个 baseHandlers 是什么呢？
4. 在 reactive 方法中可知， baseHandlers 是触发 createReactiveObject 传递的第三个参数： mutableHandlers
5. 而 mutableHandlers 则是 packages/reactivity/src/baseHandlers.ts 中导出的对象
6. 所以我们到 packages/reactivity/src/baseHandlers.ts 中，为它的 get（createGetter） 和 set（createSetter） 分别打入一个断点
7. 我们知道 get 和 set 会在 **取值** 和 **赋值** 时触发，所以此时这两个断点 **不会执行**
8. 最后 reactive 方法内执行了 proxyMap.set(target, proxy) 方法
9. 最后返回了代理对象。
10. 那么至此 reactive 方法执行完成。

由以上执行逻辑可知，对于 reactive 方法而言，其实做的事情非常简单：

1. 创建了 proxy
2. 把 proxy 加到了 proxyMap 里面
3. 最后返回了 proxy

effect

1. 在 `packages/reactivity/src/effect.ts` 第 `170` 行可以找到 `effect` 方法，在这里给一个断点
2. 执行 `new ReactiveEffect(fn)`，其中的 `fn` 就是我们传入的匿名函数：
    1. 这里涉及到了一个类 `ReactiveEffect`
    2. 查看该类可知，内部实现了两个方法：
        1. `run`
        2. `stop`
    3. 我们分别为这两个方法 **增加断点**
3. 代码继续进行
4. 可以发现执行了 `run` 方法，进入方法内部：
    1. 执行 `activeEffect = this`，赋值完成之后， `activeEffect` 为 **传入的匿名函数** `fn`
    2. 然后执行 `return this.fn()` 触发 `fn` 函数
    3. 我们知道 `fn` 函数其实就是 **传入的匿名函数**，所以 `document.querySelector('#app').innerText = obj.name`
5. 但是大家不要忘记， `obj` 是一个 `proxy`， **obj.name** 会 **触发** `getter`，所以接下来我们就会进入到 `mutableHandlers` 的 `createGetter` 中
    1. 在该代码中，触发了该方法 `const res = Reflect.get(target, key, receiver)`
    2. 此时的 `res` 即为 `张三`
    3. **注意**：接下来触发了 `track` 函数，该函数是一个重点函数， `track` 在此为 **跟踪** 的意思，我们来看它内部都做了什么：
        1. 在 `4-1` 步中，为 `activeEffect` 进行了赋值，我们知道 `activeEffect` 代表的就是 `fn` 函数
        2. 执行代码可知， `track` 内部主要做了两件事情：
            1. 为 `targetMap` 进行赋值， `targetMap` 的组成比较复杂：
                1. `key`： `target`
                2. `value`： `Map`
                    1. `key`： `key`
                    2. `value`： `Set`
            2. 最后执行了 `trackEffects(dep, eventInfo)`
                1. 其中 `eventInfo` 是一个对象，内部包含四个属性：**其中** `effect` **即为** `activeEffect` **即** `fn` **函数**
            3. 在 `trackEffects` 函数内部，核心也是做了两件事情：
                1. 为 `dep（targetMap[target][key] 得到的 Set 实例）` 添加了 `activeEffect` 函数
                2. 为 `activeEffect` 函数的 **静态属性** `deps`，增加了一个值 `dep`
                3. 即：**建立起了** `dep` **和** `activeEffect` **的联系**
        4. 那么至此，整个 `track` 的核心逻辑执行完成
        5. 我们可以把整个 `track` 的核心逻辑说成：**收集了** `activeEffect（即：fn）`
6. 最后在 `createGetter` 函数中返回了 `res（即：张三）`
7. 至此，整个 `effect` 执行完成

由以上逻辑可知，整个 `effect` 主要做了3 件事情：

2. 触发 `fn` 方法，从而激活 `getter`
3. 建立了 `targetMap` 和 `activeEffect` 之间的联系

    1. `dep.add(activeEffect)`
    2. `activeEffect.deps.push(dep)`

那么至此：**页面中即可展示 `obj.name` **，但是不要忘记，等待两秒之后，我们会修改 `obj.name` 的值，我们知道，这样会触发 `setter`，那么我们接下来来看 `setter` 中又做了什么呢？

1. 两秒之后触发 `setter`，会进入到 `packages/reactivity/src/baseHandlers.ts` 中的的 `createSetter` 方法中
2. 创建变量： `oldValue = 张三`
3. 创建变量： `value = 李四`、
4. 执行 `const result = Reflect.set(target, key, value, receiver)`，即：修改了 `obj` 的值为“李四”
5. 触发： `trigger(target, TriggerOpTypes.SET, key, value, oldValue)`，此时各参数的值为：

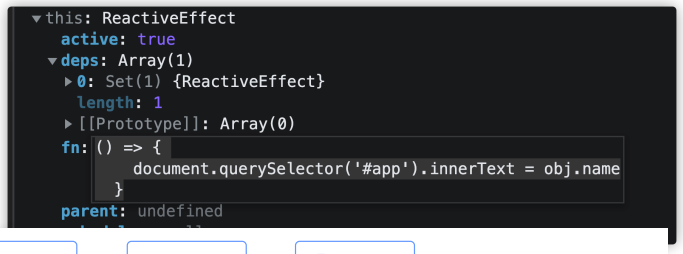
6. `trigger` 在这里为 **触发** 的意思，那么我们来看 `trigger` 内部做了什么？

    1. 首先执行： `const depsMap = targetMap.get(target)`，其中 `targetMap` 即我们在 `track` 函数中，保存 `activeEffect` 的 `targetMap`
    2. 然后代码执行到： `deps.push(depsMap.get(key))`。 `depsMap.get(key)` 获取到的即为之前保存的 `activeEffect`，即 `fn` 函数
    3. 然后触发 `triggerEffects(deps[0], eventInfo)`，我们来看 `triggerEffects` 中做了什么：

        1. 声明常量： `const effects = isArray(dep) ? dep : [...dep]`，此时的 `effects` 保存的为 `fn` 的集合
        2. 遍历 `effects`，执行： `triggerEffect(effect, debuggerEventExtraInfo)` 方法，那么我们来看 `triggerEffect` 做了什么

            1. 执行 `effect.run()` 方法，已知： `effect` 是一个 `ReactiveEffect` 类型的对象，则 `run` 方法会触发 `ReactiveEffect` 的 `run`，那么我们接下来来看 **这一次** 进入 `run` 方法时，内部做了什么？

                1. 首先还是为 `activeEffect = this` 赋值，但是要 **注意：** 此时的 `this` 不再是一个 `fn`，而是一个复杂对象：



2. 最后执行 `this.fn()` 即： `effect` 时传入的匿名函数
3. 至此， `fn` 执行，意味着： `document.querySelector('#app').innerText = 李四` ，页面将发生变化

3. `triggerEffect` 完成

4. `triggerEffects` 完成

7. `trigger` 完成
8. `setter` 回调完成

由以上逻辑可知，整个 `setter` 主要做了 2 件事情：

1. 修改 `obj` 的值
2. 触发 `targetMap` 下保存的 `fn` 函数

总结

那么到这里，我们就整个的跟踪了 `packages/vue/examples/imooc/reactive.html` 实例中：

1. `reactive` 函数
2. `effect` 函数
3. `obj.name = xx` 表达式

这三块代码背后， `vue` 究竟都做了什么。虽然整个的过程比较复杂，但是如果我们简单来去看，其实内部的完成还是比较简单的：

1. 创建 `proxy`
2. 收集 `effect` 的依赖
3. 触发收集的依赖

那么接下来我们自己的实现，将会围绕着这三个核心的理念进行。

01：前言 ◂ 上一节　　下一节 ▸ 03：框架实现：构建 reactive 函数，获取 pr...

我要提出意见反馈