# 依赖收集

### Vue是在什么时候进行依赖收集的？

答：

Vue将props、data等数据添加Oberser之后，就有了getter，getter中包含的逻辑就是依赖收集。所以只要当数据的getter得到触发就会触发依赖收集的过程，相反即使添加了响应是对象，如果他的getter没有得到触发也不会触发依赖收集。

Vue在初次渲染DOM的时候，首先需要将render函数转换为VNode，也就是说在执行vm.\_render函数的时候访问了模板上所‘提到’的变量，因此这些变量的getter也就得到了触发。而那些没有‘提到’的响应式对象并不会进行依赖收集。

Vue这样做也是十分合情合理的，比如props、data中提到的变量没有被当前模板用到，说明他们的改变并不会直接改变dom的更新，那就没有必要立刻进行依赖收集。

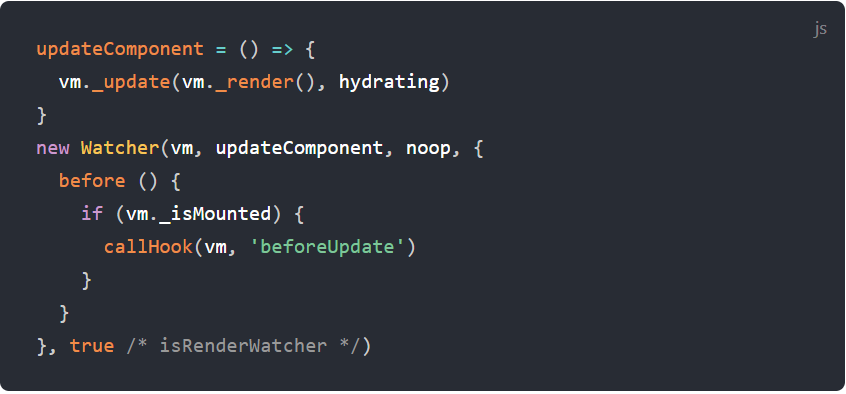
### Vue是如何进行依赖收集的？

答：

再依赖收集之前，Vue还需要作出一些提前准备，其中包括将：数据变成响应式，为Dep.target添加全局的渲染Watcher（render Watcher）。

数据便响应式我们之前已经提到过了。现在主要来看Vue是如何添加渲染Watcher的。

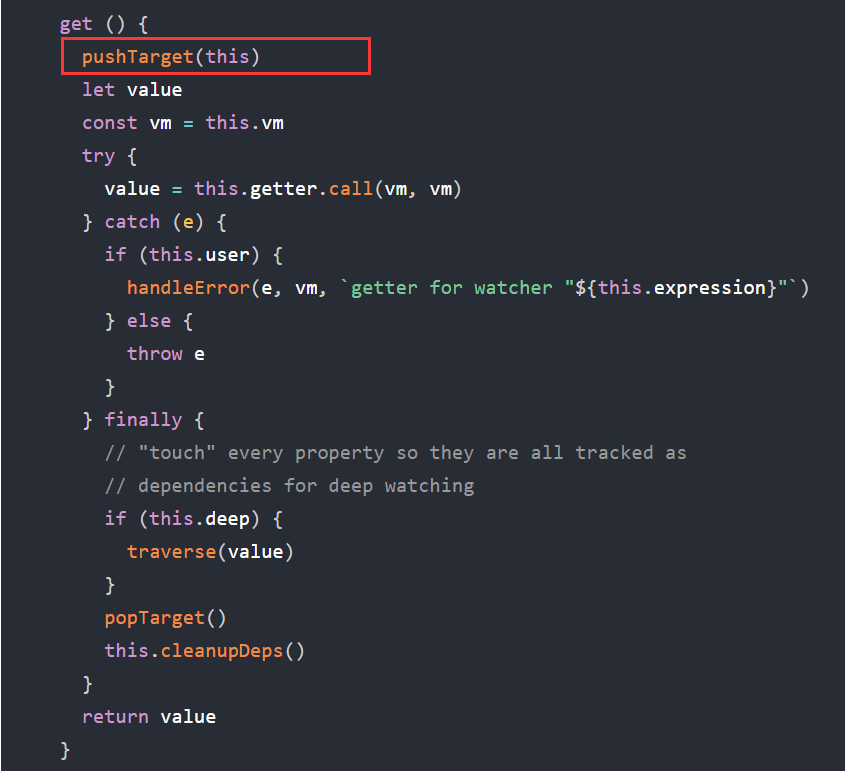
之前我们讨论过，Vue在执行mountComponent企图去渲染并挂载DOM的时候执行了这么一段核心逻辑：



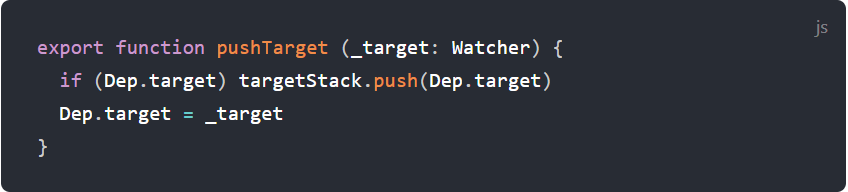
在这里Vue new了一个Watcher实例，这个Watcer就是render watcher。

Watcher的逻辑判断比较多，方法使用比较多，但是在new Watcher的时候执行了

This.get()方法，进入了get函数。

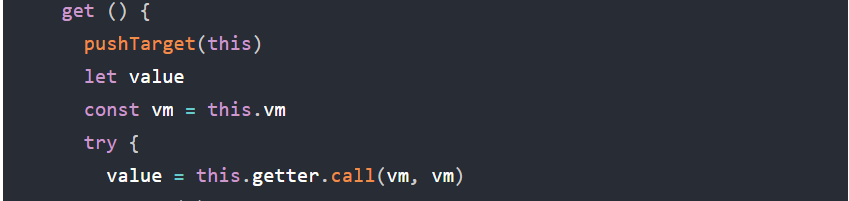


进入get中，首先执行了pushTarget方法：

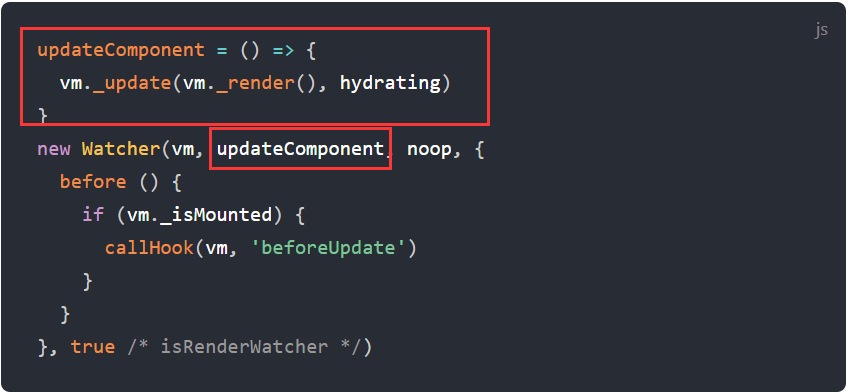


pushTarget的方法非常简单，他就是为Dep类的静态属性target赋值为当前带入得到参数watcher，这里带入的是全局的render Watcher，所以Dep.target现在就是这个render Watcher。之所以这样做，是因为Vue希望在同一时间内只有一个全局的唯一Watcher被计算。

pushTarget函数执行完成，回到get()，在get()中执行了this.getter方法，也就是相当于执行了渲染Watcher的回调函数。



执行完pushTarget，Vue在依赖收集前进行的依赖收集准备已经就绪，可以进行依赖手机了，现在执行了回调updateComponet：

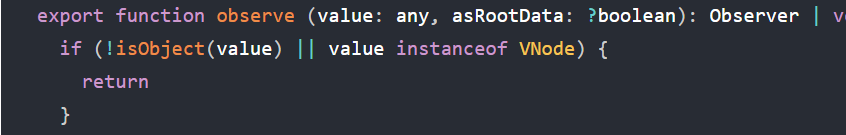


刚才提到过\_render函数的执行会触发模板上‘提到’的变量的getter。

也就执行了变量的getter逻辑：



在确定Dep.target存在后，首先执行了dep.depend()方法。然后执行了childOb.depand方法，这个地方突然看可能会有点懵，其实只要知道childOb就是当前要defineReactive的数据对象即可，它相当于这个数据对象所持有的dep。如果这个数据不是对象或者数组，那么childOb就是undefined。Vue为什么要这样做？不要忘记，在为这个数据对象oberser(value)的时候，首先判断了他是不是一个非VNode的对象，如果不是的话直接返回undefined。

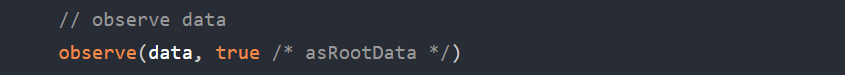


所以只有非Vnode的对象才有权执行observe去添加Observer，从而为这个非VNode对象添加了一个\_\_ob\_\_属性，然后为这个对象执行defineReactive去添加getter和setter。这个\_\_ob\_\_属性和这个非VNode对象所持有的dep实例虽不是一个实例，但是却‘长’的一样。我想，这也许就是Vue为了将数据对象所持有的dep实例的状态映射到的\_\_ob\_\_.dep上，从而可以在外界查看到这个对象所持有的dep实例的状态。

这事你可能存在疑惑，那非对象和非数组对象该怎么办？难道就不为他们也变成响应式了么？

至于这一点，我们需要回到defineReactive和initState方法中，以data中的数据为例，

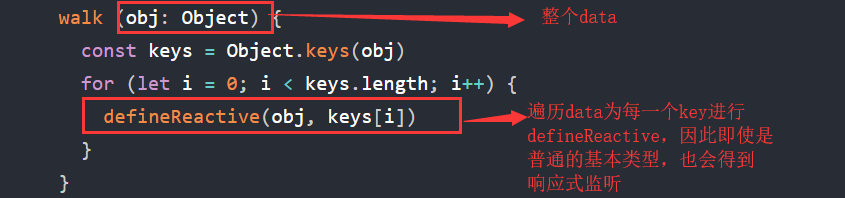
initData的最后有一个细节：



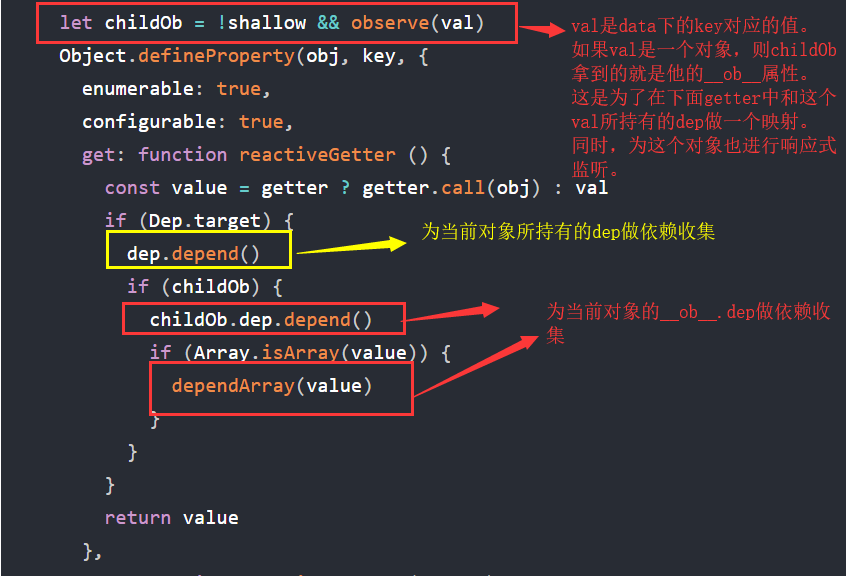
可以看到，observe在这里直接将整个data都带入了observe方法中去添加Observer，

而在Observer中Vue为整个data对象添加了\_\_ob\_\_，然后执行了defineReactive方法去为整个data对象进行响应式添加。



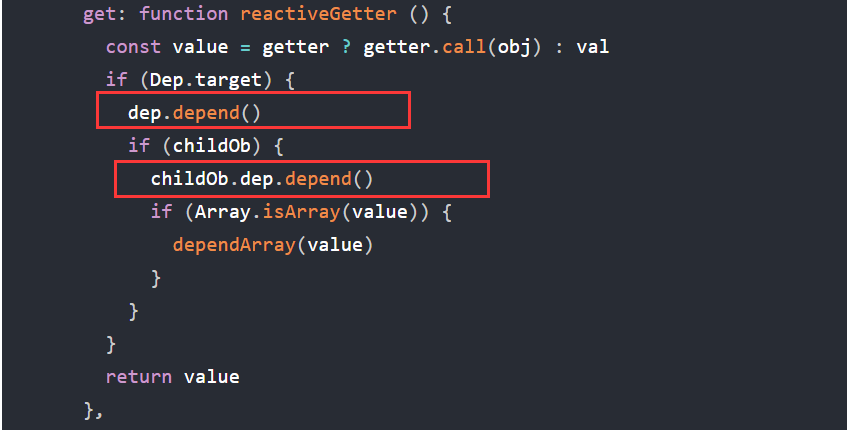


然后在definedReactive中：



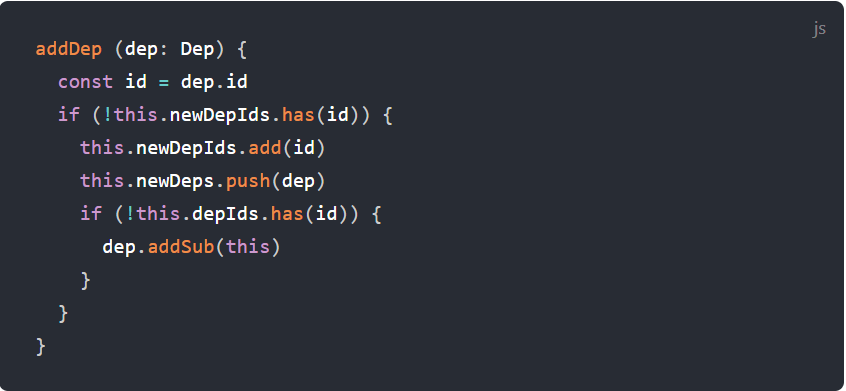
因为data必定是一个对象，所以最初的observer不会直接return，他的每一个子对象和属性都必然会经过一个defineReactive的过程。这样经过一层层递归，即使是普通的基本类型也会有机会设置getter和setter。并且子对象也会被递归的变化成响应式的。

在得知data中的每个值都会被设置getter和setter后，一旦触发getter就会进行依赖收集的过程。依赖收集就是先执行这个数据对象所持有的dep实例的depend方法：



执行depend方法，就会执行Dep.target.addDep(this)。

Dep.target是当前正在计算的render watcher，因此addDep是Watcher实例的方法：



这个watcher实例，此时也就是render watcher就会把当前的这个数据对象所持有的dep通过塞进自己的newDeps数组中，然后通过调用这个dep的addSubs方法把自己添加到这个dep的subs数组中。这个过程可以视为是当前正在计算的全局唯一watcher和来自各个部门的dep进行名片交换的过程。watcher不仅仅要记录下自己订阅的dep都有谁，也得让dep认识自己。**所以在这可以清晰的看到，Vue的订阅者（watcher）和发布者（dep）是多对多的关系。**

截至至此，Vue进行依赖收集的过程就算是结束了。

