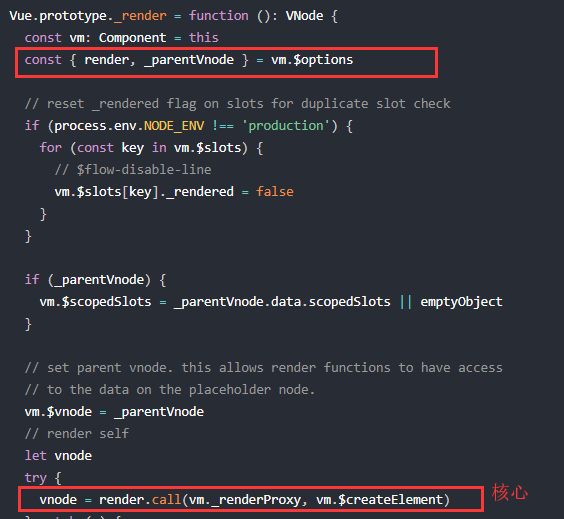
# Vue源码分析——render&createElement

### mountComponent函数内部主要设定了观察者watcher和updateComponent回调，在updateComponenta回调中又主要执行了vm.\_render方法。那vm.\_render主要做了那些事情?

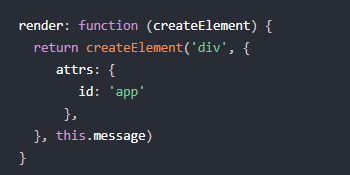
答：其实vm.\_render主要的职责就是执行vm.render，然后在vm.\_render中调用render函数。最终render函数返回的是this.$createElement方法创建VNode的结果。所以就是创建VNode的过程。

我们可以找到它最核心的那句话：



在前面我们可以发现render函数是从vm.$options中解构出来的。不要忘记在之前讲到的new Vue初始化的时候我们通过merge融合了我们手写的所有配置对象至vm.$options上。如果我们没手写render，但是写了template，那么template就会被compileToFunction方法转换为render函数在赋值给vm.$options.render。

无论是compileTofuncion生成的render函数，还是我们手写的render函数都是长这个样子：



结合vm.\_render中圈住的最后一行，我们就能发现render函数中的第一个参数createElement或者是简写的h事实上是vm.$createElement。



### vm.$createElement是如何创建VNode的？

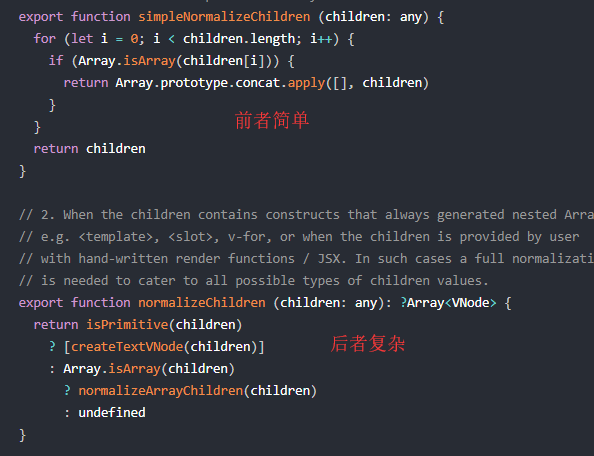
答：vm.$createElement事实上只是封装了\_createElement的，可以带入参数更佳的灵活。但是最核心的部分就是调用了\_creatElemenet方法去创建VNode节点。

而\_createElement做的事情比较多，但是最关键部分只做了两件事：

① 规范children数组

② 创建VNode

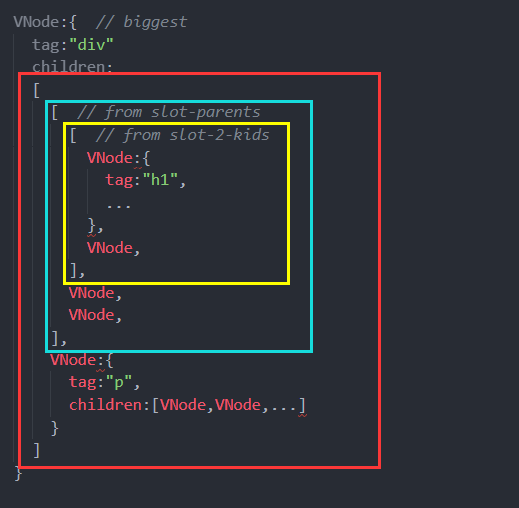
（1）规范children数组的时候，Vue提供了两种方法：simpleNormalizeChildren()和normalizeChildren()。他们做的事情都是一样的，只不过前者用于render函数由Vue编译而成的情景，而后者是render事用户手写的。编译而成的render函数的children大多数都是接近规范的，所以不需要太多判断和处理就可以对直接对children进行规范，后者则情况多变，所以判断的逻辑和处理也就比较复杂。



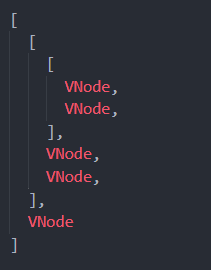
**normalizeChildren()中，他的目的就是将children参数规范成一个有层级关系的多维数组。因为存在带入的children数组中可能存在类似this.$slots.default的情况。而this.$slots.default就是一个VNode数组。如果不对children进行规范化就很难实现将children转换为dom然后渲染到页面上的需求。**

**回看补充：这里的规范化我想应该是为了解决对原始类型children值的规范，因为在最后通过patch的时候我们需要他是数组类型，而且数组的子元素必须是VNode类型。而children若是一个原始类型的值，一不符合VNode类型，二不符合children应该是数组，尽管他只有一个节点，也是需要转换成一个数组的。**

**渲染后的结果类似这样：**

（其中原始类型转换成了VNOde同时成为了数组）

缩减版类似这样：



**来自官网对this.$slots.default的说明：**



实现规范化的核心逻辑就是遍历children，先判断children[i]是不是原始类型，如果是就直接调用createTextVNode去直接创建一个文本VNode**（这个children是一个数组，即使children参数只带入一个原始类型的数据，normalizeChildren都会将转换好的TextVNode规范成为一个数组）**。如果children[i]是一个数组（可能来自slot或者v-for循环），那就需要递归调用normalizeChildren去继续规范化这个子VNode数组。如果children[i]已经是一个VNode类型（由createElement创建出来的），那就不需要管他。

值得注意的是，如果两个相邻的children[i]都是text类型，就会把他们合并在一块。

1. 在创建VNode的时候，其实逻辑比较简单，首先就是判断带入\_createElement的第一个参数tag是不是一个字符串类型的值，如果是字符串类型的值，就回去查看是不是html平台保留的标签，如果是标签就直接new VNode()即可，带入参数（tag, children....）。如果不是标签，就去判断是不是一个已经注册好的组件名，如果是组件名，就用createComponent方法以这个组件为基础去创建一个VNode。如果tag是一个组件类型，就直接调用createElement去创建VNode。至于createComponent方法我们以后再说，他也是返回一个VNode。



***TIPS：（1）VNode只能够返回一个根节点。至于原因，之后会说。***

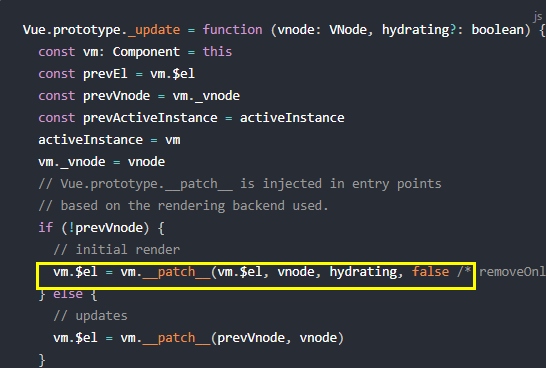
***（2）为什么slot会产生一个数组，这是因为vm.$slot.xxx本身就返回一个数组。详见官网：[https://cn.vuejs.org/v2/guide/render-function.html#%E6%8F%92%E6%A7%BD](https://cn.vuejs.org/v2/guide/render-function.html" \l "%E6%8F%92%E6%A7%BD)***

1. ***经过对children的规范化，children变成了一个类型为VNode的Array（可能多维）。***

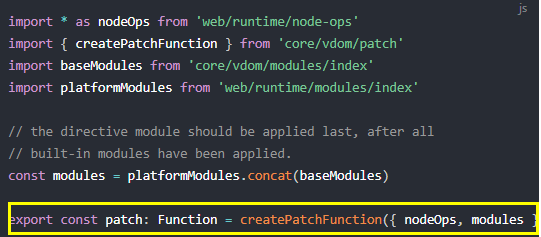
### 3、那得到VNode Tree之后如何渲染成为一个真正的DOM？

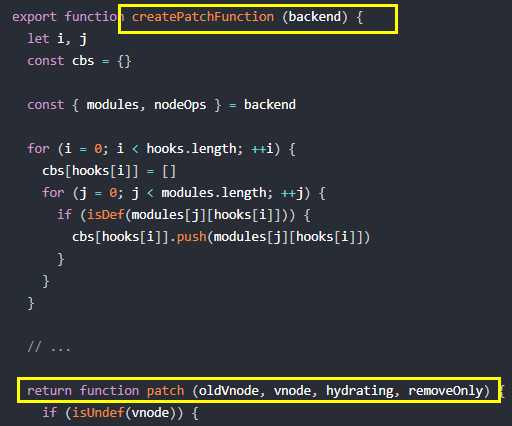
答：主要就是通过Vue实例的\_update方法来实现这个行为。这个\_update方法正是为上一步获得的VNode节点进行处理，然后作为真实的DOM渲染在vm.$el上用的。在初始化的代码中可以看到，\_update的调用有两个场景，一个场景是首次渲染的时候，一个是在数据更新的时候调用的。

在\_update函数中主要调用了vm.\_\_patch\_\_方法，这个\_\_patch\_\_方法又调用了Patch方法有两种情况，一种是指向patch方法，一种是noop也就是空函数。因为\_\_patch\_\_方法的作用是将VNode渲染成为真正的DOM然后渲染到父节点容器上去，而非浏览器环境下并不需要渲染，所以为noop。而在浏览器环境下就是patch方法，patch方法又是由createPatchFunction方法返回的一个闭包，是的，createPatchFunction返回的闭包patch才是patch方法的真身，这也才是真正处理VNode的第一个环节。









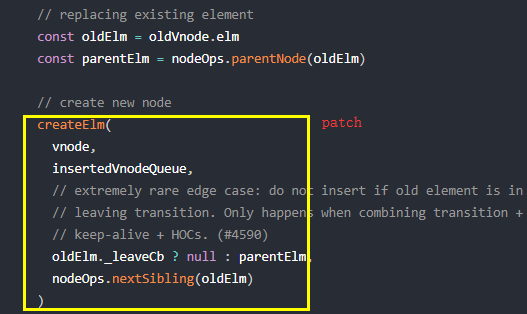
***TIP：之所以这么绕弯子，不在调用\_\_patch\_\_方法的时候直接调用patch方法的逻辑，正是因为采用了函数式变成的思想——函数柯里化。将判断和逻辑分离。***

***可以看张鑫的博客了解函数柯里化：***

***<https://www.zhangxinxu.com/wordpress/2013/02/js-currying/>***

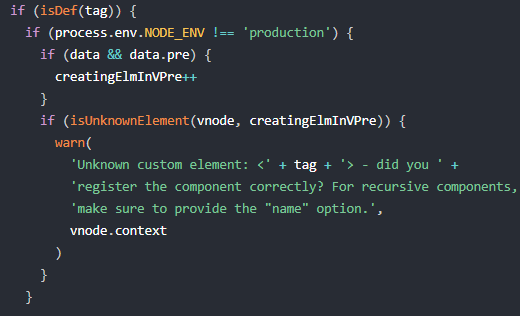
再来谈谈闭包返回的patch方法：

在这个方法中做了很多分支的功能，关键所在在于createElem方法，这个方法是patch方法的核心所在。

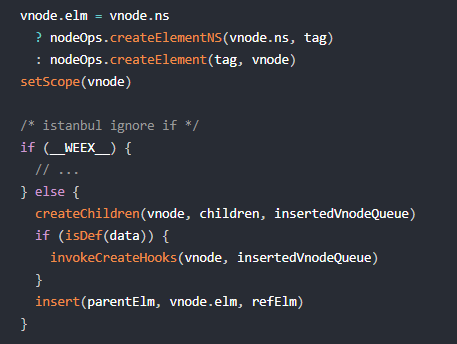
 在这个方法之中，首先先接收到了vnode，这个vnode就是之前render函数所返回的vnode。首先需要强调的是vnode一共有三个vnode.tag、vnode.elm、vnode.children。

分别对应着当前vnode的dom标签，对应的真实dom节点，还有这个vnode的子节点，子节点的类型还是vnode。

在createElm的方法中首先判断了vnode.tag。判断这个tag是否是一个平台中注册了的tag（组件或者是一个原生的标签）。如果这个标签根本不知道是什么，说明，当前这个vnode节点中肯定存在还没定义的东西，通常就是那些没有注册的组件。这是一个非常常见的错误：

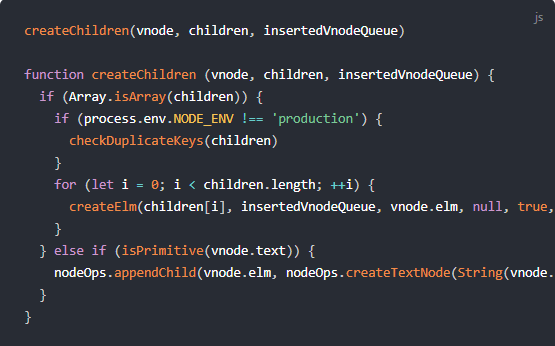


再判断完tag的合法性之后，调用了核心方法createChildren，这个方法是真正渲染DOM的核心！



可以看到，在createChildren前，先对vnode.elm做了一个createElement(),将当前vnode的tag转换为了真实的dom。值得注意的是这里有一个细节，nodeOps.createElement()的第二项是vnode，也就是说这个封装的createElemenet的时候利用了vnode的其他属性去丰富这个dom，为创建的这个dom设置了相应的属性。

然后在createChildren中遍历vnode.children。



可以看到，在createElement的时候通过深度优先遍历在children。如果children是一个原始类型，那就直接将他转换为原生TextNode，然后append到vnode.elm上，如果是一个VNode的数组，那就遍历这个数组，然后对其中的每一项递归调用createElm的方法。因此，我们可以判断出，createElm是深度优先的去将子节点渲染成DOM的。

在通过createElement将子元素都转换成为dom然后append到由最大节点的vnode（也就是之前render函数返回的vnode）转换成的dom后，将调用invpkeCreateHooks方法执行所有的create钩子，并把vnode给push到insertedVnodeQueue中，最终createElm将调用insert方法将其append到vm.$el上。这样就完整的实现了VNode -> dom的转换。