☰ 逐行剖析 Vue.js 源码

1. 前言

在上一篇文章介绍 VNode 的时候我们说了， VNode 最大的用途就是在数据变化前后生成真实 DOM 对应的虚拟 DOM 节点，然后就可以对比新旧两份 VNode ，找出差异所在，然后更新有差异的 DOM 节点，最终达到以最少操作真实 DOM 更新视图的目的。而对比新旧两份 VNode 并找出差异的过程就是所谓的 DOM-Diff 过程。 DOM-Diff 算法是整个虚拟 DOM 的核心所在，那么接下来，我们就以源码出发，深入研究一下 Vue 中的 DOM-Diff 过程是怎样的。

2. patch

在 Vue 中，把 DOM-Diff 过程叫做 patch 过程。patch,意为“补丁”，即指对旧的 VNode 修补，打补丁从而得到新的 VNode ，非常形象哈。那不管叫什么，其本质都是把对比新旧两份 VNode 的过程。我们在下面研究 patch 过程的时候，一定把握住这样一个思想：所谓旧的 VNode (即 oldVNode)就是数据变化之前视图所对应的虚拟 DOM 节点，而新的 VNode 是数据变化之后将要渲染的新的视图所对应的虚拟 DOM 节点，所以我们要以生成的新的 VNode 为基准，对比旧的 oldVNode ，如果新的 VNode 上有的节点而旧的 oldVNode 上没有，那么就在旧的 oldVNode 上加上去；如果新的 VNode 上没有的节点而旧的 oldVNode 上有，那么就在旧的 oldVNode 上去掉；如果某些节点在新的 VNode 和旧的 oldVNode 上都有，那么就以新的 VNode 为准，更新旧的 oldVNode ，从而让新旧 VNode 相同。

可能你感觉有点绕，没关系，我们在说的通俗一点，你可以这样理解：假设你电脑上现在有一份旧的电子版文档，此时老板又给了你一份新的纸质板文档，并告诉你这两份文档内容大部分都是一样的，让你以新的纸质版文档为准，把纸质版文档做一份新的电子版文档发给老板。对于这个任务此时，你应该有两种解决方案：一种方案是不管它旧的文档内容是什么样的，统统删掉，然后对着新的纸质版文档一个字一个字的敲进去，这种方案就是不用费脑，就是受点累也能解决问题。而另外一种方案是以新的纸质版文档为基准，对比看旧的电子版文档跟新的纸质版文档有什么差异，如果某些部分在新的文档里有而旧的文档里没有，那就在旧的文档里面把这些部分加上；如果某些部分在新的文档里没有而旧的文档里有，那就在旧的文档里把这些部分删掉；如果某些部分在新旧文档里都有，那就对比看有没有需要更新的，最后在旧的文档里更新一下，最终达到把旧的文档变成跟手里纸质版文档一样，完美解决。

对比以上两种方案，显然你和 Vue 一样聪明，肯定会选择第二种方案。第二种方案里的旧的电子版文档对应就是已经渲染在视图上的 oldVNode ，新的纸质版文档对应的是将要渲染在

☰ 逐行剖析 Vue.js 源码

说了这么多，听起来感觉好像很复杂的样子，其实不然，我们仔细想想，整个 `patch` 无非就是干三件事：

- 创建节点：新的 `VNode` 中有而旧的 `oldVNode` 中没有，就在旧的 `oldVNode` 中创建。
- 删除节点：新的 `VNode` 中没有而旧的 `oldVNode` 中有，就从旧的 `oldVNode` 中删除。
- 更新节点：新的 `VNode` 和旧的 `oldVNode` 中都有，就以新的 `VNode` 为准，更新旧的 `oldVNode` 。

OK，到这里，你就对 `Vue` 中的 `patch` 过程理解了一半了，接下来，我们就逐个分析，看 `Vue` 对于以上三件事都是怎么做的。

3. 创建节点

在上篇文章中我们分析了， `VNode` 类可以描述6种类型的节点，而实际上只有3种类型的节点能够被创建并插入到 `DOM` 中，它们分别是：元素节点、文本节点、注释节点。所以 `Vue` 在创建节点的时候会判断在新的 `VNode` 中有而旧的 `oldVNode` 中没有的这个节点是属于哪种类型的节点，从而调用不同的方法创建并插入到 `DOM` 中。

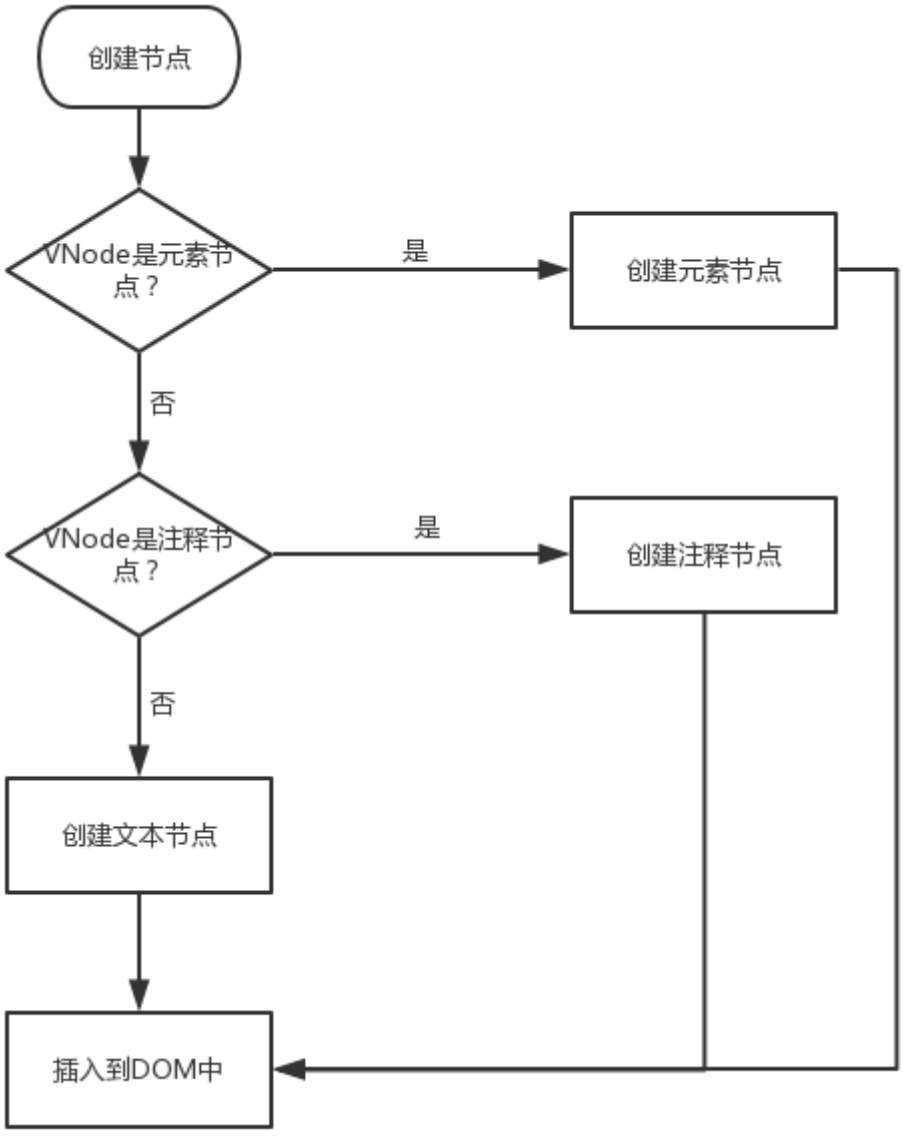
其实判断起来也不难，因为这三种类型的节点其特点非常明显，在源码中是怎么判断的：

```js
// 源码位置: /src/core/vdom/patch.js
function createElm (vnode, parentElm, refElm) {
    const data = vnode.data
    const children = vnode.children
    const tag = vnode.tag
    if (isDef(tag)) {
      vnode.elm = nodeOps.createElement(tag, vnode)   // 创建元素节点
      createChildren(vnode, children, insertedVnodeQueue) // 创建元素节点的
      insert(parentElm, vnode.elm, refElm)       // 插入到DOM中
    } else if (isTrue(vnode.isComment)) {
      vnode.elm = nodeOps.createComment(vnode.text)  // 创建注释节点
      insert(parentElm, vnode.elm, refElm)           // 插入到DOM中
    } else {
      vnode.elm = nodeOps.createTextNode(vnode.text)  // 创建文本节点
      insert(parentElm, vnode.elm, refElm)           // 插入到DOM中
    }
  }

```

- 判断是否为元素节点只需判断该 VNode 节点是否有 tag 标签即可。如果有 tag 属性即认为是元素节点，则调用 createElement 方法创建元素节点，通常元素节点还会有子节点，那就递归遍历创建所有子节点，将所有子节点创建好之后 insert 插入到当前元素节点里面，最后把当前元素节点插入到 DOM 中。
- 判断是否为注释节点，只需判断 VNode 的 isComment 属性是否为 true 即可，若为 true 则为注释节点，则调用 createComment 方法创建注释节点，再插入到 DOM 中。
- 如果既不是元素节点，也不是注释节点，那就认为是文本节点，则调用 createTextNode 方法创建文本节点，再插入到 DOM 中。

> 代码中的 nodeOps 是 Vue 为了跨平台兼容性，对所有节点操作进行了封装，例如 nodeOps.createTextNode() 在浏览器端等同于 document.createTextNode()

以上就完成了创建节点的操作，其完整流程图如下：



4. 删除节点

的 `oldVNode` 中删除。删除节点非常简单，只需在要删除节点的父元素上调用 `removeChild` 方法即可。源码如下：

```js
function removeNode (el) {
  const parent = nodeOps.parentNode(el)  // 获取父节点
  if (isDef(parent)) {
    nodeOps.removeChild(parent, el)  // 调用父节点的removeChild方法
  }
}
```

5. 更新节点

创建节点和删除节点都比较简单，而更新节点就相对较为复杂一点了，其实也不算多复杂，只要理清逻辑就能理解了。

更新节点就是当某些节点在新的 `VNode` 和旧的 `oldVNode` 中都有时，我们就需要细致比较一下，找出不一样的地方进行更新。

介绍更新节点之前，我们先介绍一个小的概念，就是什么是静态节点？我们看个例子：

```html
<p>我是不会变化的文字</p>
```

上面这个节点里面只包含了纯文字，没有任何可变的变量，这也就是说，不管数据再怎么变化，只要这个节点第一次渲染了，那么它以后就永远不会发生变化，这是因为它不包含任何变量，所以数据发生任何变化都与它无关。我们把这种节点称之为静态节点。

OK，有了这个概念以后，我们开始更新节点。更新节点的时候我们需要对以下3种情况进行判断并分别处理：

1. 如果 `VNode` 和 `oldVNode` 均为静态节点

   我们说了，静态节点无论数据发生任何变化都与它无关，所以都为静态节点的话则直接跳过，无需处理。

2. 如果 `VNode` 是文本节点

   如果 `VNode` 是文本节点即表示这个节点内只包含纯文本，那么只需看 `oldVNode` 是否也是文本节点，如果是，那就比较两个文本是否不同，如果不同则把 `oldVNode` 里的文本改成

☰ 逐行剖析 Vue.js 源码

3. 如果 `VNode` 是元素节点

   如果 `VNode` 是元素节点，则又细分以下两种情况：

   - 该节点包含子节点

     如果新的节点内包含了子节点，那么此时要看旧的节点是否包含子节点，如果旧的节点里也包含了子节点，那就需要递归对比更新子节点；如果旧的节点里不包含子节点，那么这个旧节点有可能是空节点或者是文本节点，如果旧的节点是空节点就把新的节点里的子节点创建一份然后插入到旧的节点里面，如果旧的节点是文本节点，则把文本清空，然后把新的节点里的子节点创建一份然后插入到旧的节点里面。

   - 该节点不包含子节点

     如果该节点不包含子节点，同时它又不是文本节点，那就说明该节点是个空节点，那就好办了，不管旧节点之前里面都有啥，直接清空即可。

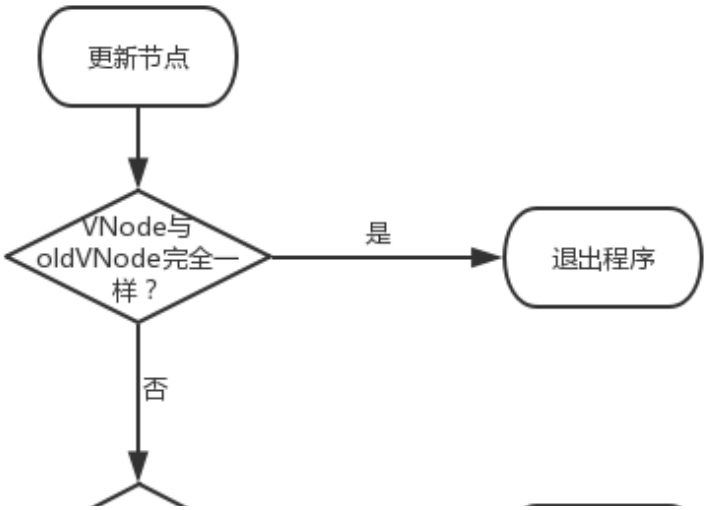
OK，处理完以上3种情况，更新节点就算基本完成了，接下来我们看下源码中具体是怎么实现的，源码如下：

```js
// 更新节点
function patchVnode (oldVnode, vnode, insertedVnodeQueue, removeOnly) {
  // vnode与oldVnode是否完全一样？若是，退出程序
  if (oldVnode === vnode) {
    return
  }
  const elm = vnode.elm = oldVnode.elm

  // vnode与oldVnode是否都是静态节点？若是，退出程序
  if (isTrue(vnode.isStatic) &&
    isTrue(oldVnode.isStatic) &&
    vnode.key === oldVnode.key &&
    (isTrue(vnode.isCloned) || isTrue(vnode.isOnce))
  ) {
    return
  }

  const oldCh = oldVnode.children
  const ch = vnode.children
  // vnode有text属性？若没有：
  if (isUndef(vnode.text)) {
    // vnode的子节点与oldVnode的子节点是否都存在？
```

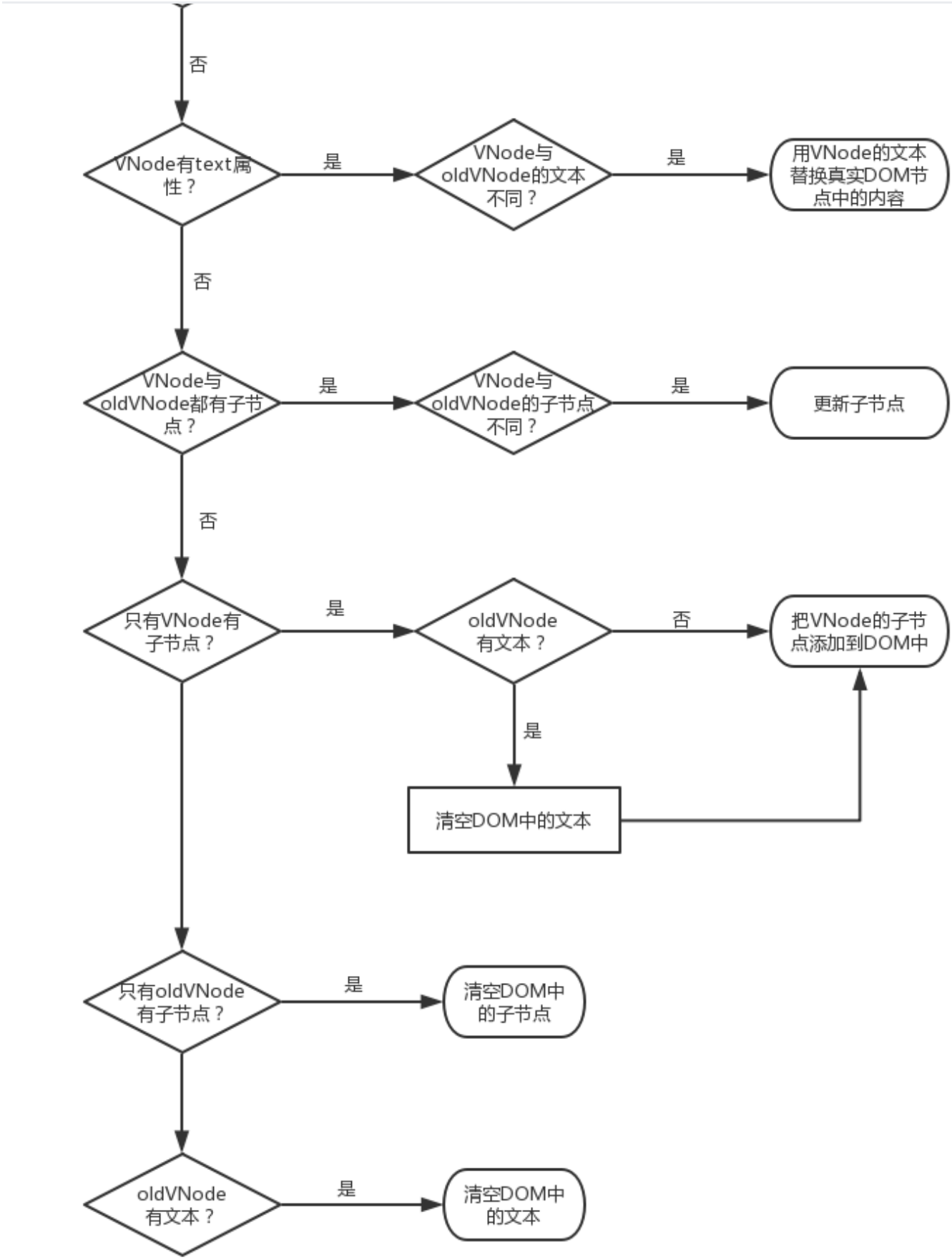
☰ 逐行剖析 Vue.js 源码

```
      if (oldCh !== ch) updateChildren(elm, oldCh, ch, insertedVnodeQue
    }
    // 若只有vnode的子节点存在
    else if (isDef(ch)) {
      /**
       * 判断oldVnode是否有文本?
       * 若没有，则把vnode的子节点添加到真实DOM中
       * 若有，则清空Dom中的文本，再把vnode的子节点添加到真实DOM中
       */
      if (isDef(oldVnode.text)) nodeOps.setTextContent(elm, '')
      addVnodes(elm, null, ch, 0, ch.length - 1, insertedVnodeQueue)
    }
    // 若只有oldnode的子节点存在
    else if (isDef(oldCh)) {
      // 清空DOM中的子节点
      removeVnodes(elm, oldCh, 0, oldCh.length - 1)
    }
    // 若vnode和oldnode都没有子节点，但是oldnode中有文本
    else if (isDef(oldVnode.text)) {
      // 清空oldnode文本
      nodeOps.setTextContent(elm, '')
    }
    // 上面两个判断一句话概括就是，如果vnode中既没有text，也没有子节点，那么对应的c
  }
  // 若有，vnode的text属性与oldVnode的text属性是否相同?
  else if (oldVnode.text !== vnode.text) {
    // 若不相同：则用vnode的text替换真实DOM的文本
    nodeOps.setTextContent(elm, vnode.text)
  }
}
```

上面代码里注释已经写得很清晰了，接下来我们画流程图来梳理一下整个过程，流程图如下：



☰ 逐行剖析 Vue.js 源码



通过对照着流程图以及代码，相信更新节点这部分逻辑你很容易就能理解了。

开学习。

6. 总结

在本篇文章中我们介绍了 `Vue` 中的 `DOM-Diff` 算法：patch过程。我们先介绍了算法的整个思想流程，然后通过梳理算法思想，了解了整个 `patch` 过程干了三件事，分别是：创建节点，删除节点，更新节点。并且对每件事情都对照源码展开了细致的学习，画出了其逻辑流程图。另外对于更新节点中，如果新旧 `VNode` 里都包含了子节点，我们就需要细致的去更新子节点，关于更新子节点的过程我们在下一篇文章中展开学习。

在 GitHub 上编辑此页

上次更新: 3/24/2020, 5:37:47 AM

← Vue中的虚拟DOM 更新子节点 →