☰ 逐行剖析 Vue.js 源码

1. 前言

虚拟DOM，这个名词作为当下的前端开发人员你一定不会陌生，至少会略有耳闻，但不会闻所未闻吧。这也是现在求职面试考察中非常高频的一个考点了。因为在当下的前端三大框架中关于虚拟DOM或多或少都有所涉及，那么接下来，我们就从源码角度出发，看看 Vue 中的虚拟DOM时怎样的。

2. 虚拟DOM简介

由于本系列文章是针对 Vue 源码深入学习的，所以着重分析在 Vue 中对虚拟DOM是如何实现的，而对于虚拟DOM本身这个概念不做大篇幅的展开讨论，仅从以下几个问题简单介绍：

1. 什么是虚拟DOM?

所谓虚拟DOM，就是用一个 JS 对象来描述一个 DOM 节点，像如下示例：

```js
<div class="a" id="b">我是内容</div>

{
  tag:'div',         // 元素标签
  attrs:{            // 属性
    class:'a',
    id:'b'
  },
  text:'我是内容',  // 文本内容
  children:[]        // 子元素
}
```

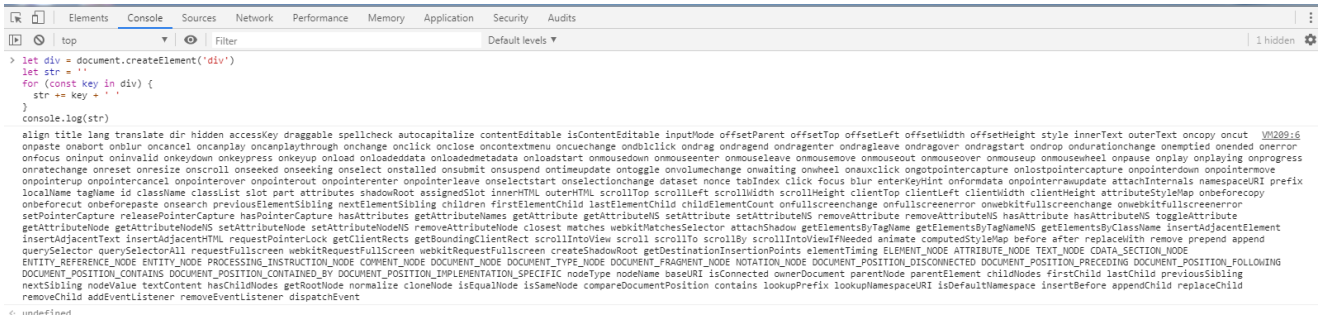
我们把组成一个 DOM 节点的必要东西通过一个 JS 对象表示出来，那么这个 JS 对象就可以用来描述这个 DOM 节点，我们把这个 JS 对象就称为是这个真实 DOM 节点的虚拟 DOM 节点。

2. 为什么要有虚拟DOM?

我们知道， Vue 是数据驱动视图的，数据发生变化视图就要随之更新，在更新视图的时候难免要操作 DOM ,而操作真实 DOM 又是非常耗费性能的，这是因为浏览器的标准就把 DOM 设计的非常复杂，所以一个真正的 DOM 元素是非常庞大的，如下所示：

☰ 逐行剖析 Vue.js 源码

```
3  for (const key in div) {
4    str += key + ''
5  }
6  console.log(str)
```



上图中我们打印一个简单的空 `div` 标签，就打印出这么多东西，更不用说复杂的、深嵌套的 `DOM` 节点了。由此可见，真实的 `DOM` 节点数据会占据更大的内存，当我们频繁的去做 `DOM` 更新，会产生一定的性能问题，因为 `DOM` 的更新有可能带来页面的重绘或重排。

那么有没有什么解决方案呢？当然是有的。我们可以用 `JS` 的计算性能来换取操作 `DOM` 所消耗的性能。

既然我们逃不掉操作 `DOM` 这道坎,但是我们可以尽可能少的操作 `DOM` 。那如何在更新视图的时候尽可能少的操作 `DOM` 呢？最直观的思路就是我们不要盲目的去更新视图，而是通过对比数据变化前后的状态，计算出视图中哪些地方需要更新，只更新需要更新的地方，而不需要更新的地方则不需关心，这样我们就可以尽可能少的操作 `DOM` 了。这也就是上面所说的用 `JS` 的计算性能来换取操作 `DOM` 的性能。

我们可以用 `JS` 模拟出一个 `DOM` 节点，称之为虚拟 `DOM` 节点。当数据发生变化时，我们对比变化前后的虚拟 `DOM` 节点，通过 `DOM-Diff` 算法计算出需要更新的地方，然后去更新需要更新的视图。

这就是虚拟 `DOM` 产生的原因以及最大的用途。

另外，使用虚拟 `DOM` 也能使得 `Vue` 不再依赖于浏览器环境。我们可以很容易的在 `Broswer` 端或者服务器端操作虚拟 `DOM`，需要 `render` 时再将虚拟 `DOM` 转换为真实 `DOM` 即可。这也使得 `Vue` 有了实现服务器端渲染的能力。

3. Vue中的虚拟DOM

☰ 逐行剖析 Vue.js 源码

3.1 VNode类

我们说了，虚拟 DOM 就是用 JS 来描述一个真实的 DOM 节点。而在 Vue 中就存在了一个 VNode 类，通过这个类，我们就可以实例化出不同类型的虚拟 DOM 节点，源码如下：

```js
// 源码位置：src/core/vdom/vnode.js

export default class VNode {
  constructor (
    tag?: string,
    data?: VNodeData,
    children?: ?Array<VNode>,
    text?: string,
    elm?: Node,
    context?: Component,
    componentOptions?: VNodeComponentOptions,
    asyncFactory?: Function
  ) {
    this.tag = tag                                    /*当前节点的标签名*/
    this.data = data          /*当前节点对应的对象，包含了具体的一些数据信息，是一
    this.children = children  /*当前节点的子节点，是一个数组*/
    this.text = text      /*当前节点的文本*/
    this.elm = elm        /*当前虚拟节点对应的真实dom节点*/
    this.ns = undefined             /*当前节点的名字空间*/
    this.context = context           /*当前组件节点对应的Vue实例*/
    this.fnContext = undefined        /*函数式组件对应的Vue实例*/
    this.fnOptions = undefined
    this.fnScopeId = undefined
    this.key = data && data.key            /*节点的key属性，被当作节点的标志，
    this.componentOptions = componentOptions    /*组件的option选项*/
    this.componentInstance = undefined        /*当前节点对应的组件的实例*/
    this.parent = undefined            /*当前节点的父节点*/
    this.raw = false          /*简而言之就是是否为原生HTML或只是普通文本，inner
    this.isStatic = false          /*静态节点标志*/
    this.isRootInsert = true       /*是否作为跟节点插入*/
    this.isComment = false              /*是否为注释节点*/
    this.isCloned = false            /*是否为克隆节点*/
    this.isOnce = false                 /*是否有v-once指令*/
    this.asyncFactory = asyncFactory
    this.asyncMeta = undefined
    this.isAsyncPlaceholder = false
  }
```

```
    return this.componentInstance
  }
}
```

从上面的代码中可以看出： `VNode` 类中包含了描述一个真实 `DOM` 节点所需要的一系列属性，如 `tag` 表示节点的标签名， `text` 表示节点中包含的文本， `children` 表示该节点包含的子节点等。通过属性之间不同的搭配，就可以描述出各种类型的真实 `DOM` 节点。

3.2 VNode的类型

上一小节最后我们说了，通过属性之间不同的搭配， `VNode` 类可以描述出各种类型的真实 `DOM` 节点。那么它都可以描述出哪些类型的节点呢？通过阅读源码，可以发现通过不同属性的搭配，可以描述出以下几种类型的节点。

- 注释节点
- 文本节点
- 元素节点
- 组件节点
- 函数式组件节点
- 克隆节点

接下来，我们就把这几种类型的节点描述方式从源码中一一对应起来。

3.2.1 注释节点

注释节点描述起来相对就非常简单了，它只需两个属性就够了，源码如下：

```js
// 创建注释节点
export const createEmptyVNode = (text: string = '') => {
  const node = new VNode()
  node.text = text
  node.isComment = true
  return node
}
```

从上面代码中可以看到，描述一个注释节点只需两个属性，分别是： `text` 和 `isComment` 。其中 `text` 属性表示具体的注释信息， `isComment` 是一个标志，用来标识一个节点是否是注释节点。

☰ 逐行剖析 Vue.js 源码

3.2.2 文本节点

文本节点描述起来比注释节点更简单，因为它只需要一个属性，那就是 `text` 属性，用来表示具体的文本信息。源码如下：

```js
// 创建文本节点
export function createTextVNode (val: string | number) {
  return new VNode(undefined, undefined, undefined, String(val))
}
```

3.2.3 克隆节点

克隆节点就是把一个已经存在的节点复制一份出来，它主要是为了做模板编译优化时使用，这个后面我们会说到。关于克隆节点的描述，源码如下：

```js
// 创建克隆节点
export function cloneVNode (vnode: VNode): VNode {
  const cloned = new VNode(
    vnode.tag,
    vnode.data,
    vnode.children,
    vnode.text,
    vnode.elm,
    vnode.context,
    vnode.componentOptions,
    vnode.asyncFactory
  )
  cloned.ns = vnode.ns
  cloned.isStatic = vnode.isStatic
  cloned.key = vnode.key
  cloned.isComment = vnode.isComment
  cloned.fnContext = vnode.fnContext
  cloned.fnOptions = vnode.fnOptions
  cloned.fnScopeId = vnode.fnScopeId
  cloned.asyncMeta = vnode.asyncMeta
  cloned.isCloned = true
  return cloned
}
```

☰ 逐行剖析 Vue.js 源码

3.2.4 元素节点

相比之下，元素节点更贴近于我们通常看到的真实 `DOM` 节点，它有描述节点标签名词的 `tag` 属性，描述节点属性如 `class` 、 `attributes` 等的 `data` 属性，有描述包含的子节点信息的 `children` 属性等。由于元素节点所包含的情况相比而言比较复杂，源码中没有像前三种节点一样直接写死（当然也不可能写死），那就举个简单例子说明一下：

```js
// 真实DOM节点
<div id='a'><span>难凉热血</span></div>

// VNode节点
{
  tag:'div',
  data:{},
  children:[
    {
      tag:'span',
      text:'难凉热血'
    }
  ]
}
```

我们可以看到，真实 `DOM` 节点中: `div` 标签里面包含了一个 `span` 标签，而 `span` 标签里面有一段文本。反应到 `VNode` 节点上就如上所示: `tag` 表示标签名， `data` 表示标签的属性 `id` 等， `children` 表示子节点数组。

3.2.5 组件节点

组件节点除了有元素节点具有的属性之外，它还有两个特有的属性：

- componentOptions :组件的option选项，如组件的 `props` 等
- componentInstance :当前组件节点对应的 `Vue` 实例

3.2.6 函数式组件节点

函数式组件节点相较于组件节点，它又有两个特有的属性：

- fnContext:函数式组件对应的Vue实例
- fnOptions: 组件的option选项

3.2.7 小结

以上就是 VNode 可以描述的多种节点类型，它们本质上都是 VNode 类的实例，只是在实例化的时候传入的属性参数不同而已。

3.3 VNode的作用

说了这么多，那么 VNode 在 Vue 的整个虚拟 DOM 过程起了什么作用呢?

其实 VNode 的作用是相当大的。我们在视图渲染之前，把写好的 template 模板先编译成 VNode 并缓存下来，等到数据发生变化页面需要重新渲染的时候，我们把数据发生变化后生成的 VNode 与前一次缓存下来的 VNode 进行对比，找出差异，然后有差异的 VNode 对应的真实 DOM 节点就是需要重新渲染的节点，最后根据有差异的 VNode 创建出真实的 DOM 节点再插入到视图中，最终完成一次视图更新。

4. 总结

本章首先介绍了虚拟 DOM 的一些基本概念和为什么要有虚拟 DOM ，其实说白了就是以 JS 的计算性能来换取操作真实 DOM 所消耗的性能。接着从源码角度我们知道了在 Vue 中是通过 VNode 类来实例化出不同类型的虚拟 DOM 节点，并且学习了不同类型节点生成的属性的不同，所谓不同类型的节点其本质还是一样的，都是 VNode 类的实例，只是在实例化时传入的属性参数不同罢了。最后探究了 VNode 的作用，有了数据变化前后的 VNode ，我们才能进行后续的 DOM-Diff 找出差异，最终做到只更新有差异的视图，从而达到尽可能少的操作真实 DOM 的目的，以节省性能。

在 GitHub 上编辑此页

上次更新: 3/24/2020, 5:37:47 AM

← Array的变化侦测 Vue中的DOM-Diff →