33 真正理解虚拟DOM（一）——虚拟DOM思想

更新时间：2020-05-29 14:23:04

> 读一本好书，就是和许多高尚的人谈话。——歌德

学习虚拟 DOM，首先要知其所以然——要清楚它的出现是为了解决什么问题。

为了弄明白这一点，我们不得不从 DOM 操作的历史说起。

为什么我们需要虚拟 DOM——DOM 操作演化史

首先，我们一起来回顾一下那段没有虚拟 DOM 的日子。

远古前端的无奈——“人肉操作”时期

在前端这个工种的萌芽阶段，前端页面“展示”的属性远远强于其“交互”的属性。因此 JavaScript 在很长一段时间里都不是前端世界的主角，人们只用 JS 来做一些类似于拖拽、隐藏这样简单的动效。

这个时期里，前端工程师需要关心的 DOM 操作是有限的。这样看来，使用 JS、jQuery 来定点对 DOM 进行修改好像也不是什么特别让人头大的事情。于是，任劳任怨的程序员们就这样过了很多年人肉修改 DOM 的日子。

数据驱动的先驱——“模板助攻”时期

随着前端业务复杂度不断提升，前端页面对交互体验的要求越来越高，骤增的动态内容带来了大量的 DOM 修改需求。此时若再要求工程师们去逐一修改 DOM 节点，其工作量将大到令人绝望。

聪明的前端er们可不会这么容易被难倒，很快，我们的前辈创造出了“模板”这一解决方案。比如说我有一个学生信息表格需要展示，那么我可以给它一组初始化数据 students：

```
[
  {
    name: 'Bob',
    age: 21
  },
  {
    name: 'Maria',
    age: 22
  },
  {
    name: 'Lynn',
    age: 22
  }
]
```

然后把这组数据塞进 template 去：

```
<table>
  {% students.forEach(function(student){ %}
  <tr>
    <td>{% student.name %}</td>
    <td>{% student.age %}</td>
    </tr>
  {% }); %}
</table>
```

模板会帮我们做什么呢？它会把你的 students 这个数据源读进去，塞到上面这段 template 代码里，把它们融合在一起，吐出一段目标 HTML 给你。然后这段 HTML 代码就可以直接被拿去渲染到页面上，成为 DOM。

这个过程差不多是这样：

```
// 数据和模板融合出 HTML 代码
var targetDOM = template({data: students})
// 添加到页面中去
document.body.appendChild(targetDOM)
```

这样的操作，可以帮助我们程序员做到只关心数据及数据变化，而不必操心具体的 DOM 细节，大大解放了生产力。

模板带来的问题

模板这种形式的 DOM 方案，其实是非常粗糙的，蕴含了不小的隐患。

大家现在考虑一个常见的场景：如果我发现上述表格中某个同学的名字写错了——Maria 其实叫 Mariana。现在我要把这个名字改掉，于是我改了 students 里对应的姓名信息，模板会做什么呢？

首先，模板引擎会把 targetDOM 这个节点整个给注销掉；

然后，再重新走一遍刚刚走过的渲染流程：

1. 数据+模板=HTML代码
2. 把 HTML 代码渲染到页面上，形成真实的 DOM

注意到没有？这一通操作波及了太多无辜群众啊！

本来我只是想改 Maria 的名字，现在整个表格都需要被重新渲染。DOM 操作的范围，从小小的一个表格字段位，扩大到了整个表格。这不合理。

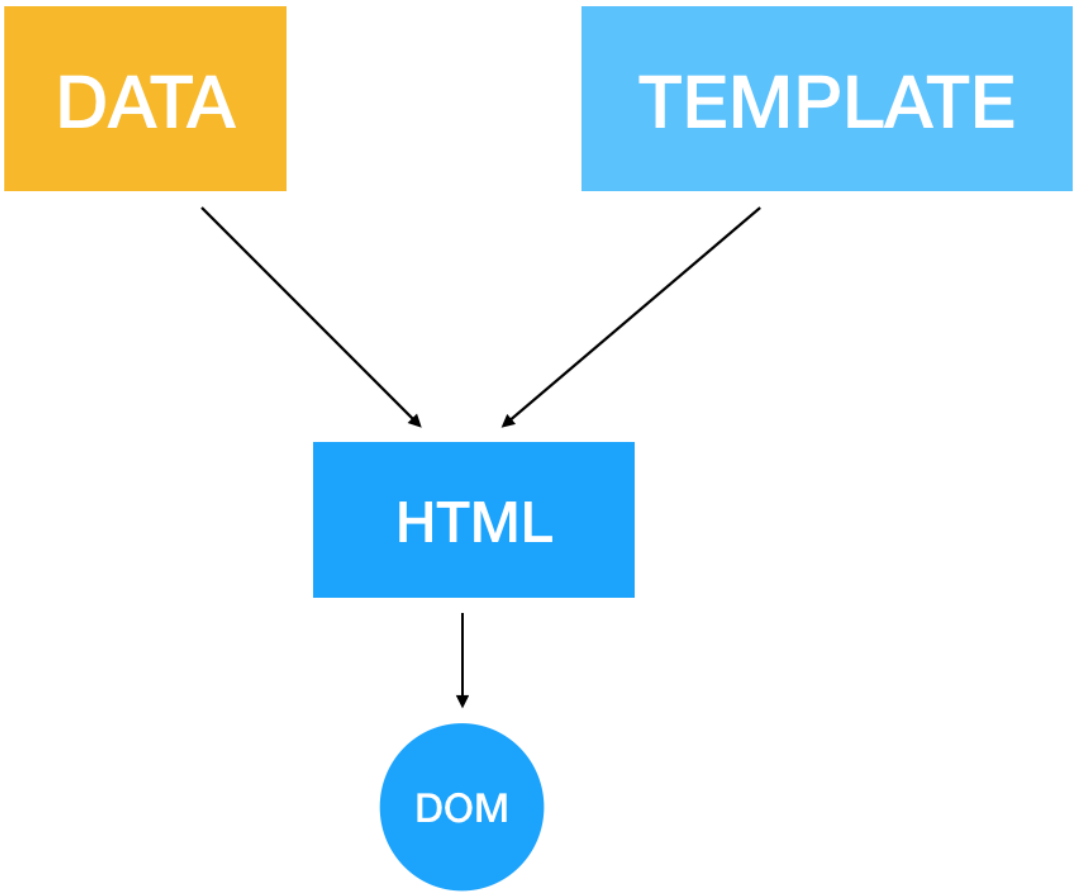
同时，由于上述更新过程中涉及的 DOM 节点注销和 DOM 节点添加，都是真刀真枪、真耗性能的 DOM 操作。当我们更新频率稍微高一点时，页面就会吃不消了。因此，模板渲染方案并不能很好地解决更新的问题。

现代前端框架的基石——虚拟 DOM

发现问题并不是一件坏事——如果我们能够恰如其分地解决问题，那么问题就变成了机会。善于解决问题的工程师，才是真正的创造者。

当这帮创造者发现模板方案纵有千好万好、无奈性能不好时，他们并没有气馁。反而是转念一想：操作真实 DOM 太费力，那我操作假的不就行了？

更进一步想：过去，我们利用模板来实现 DOM 操作，初始化过程是这样的：



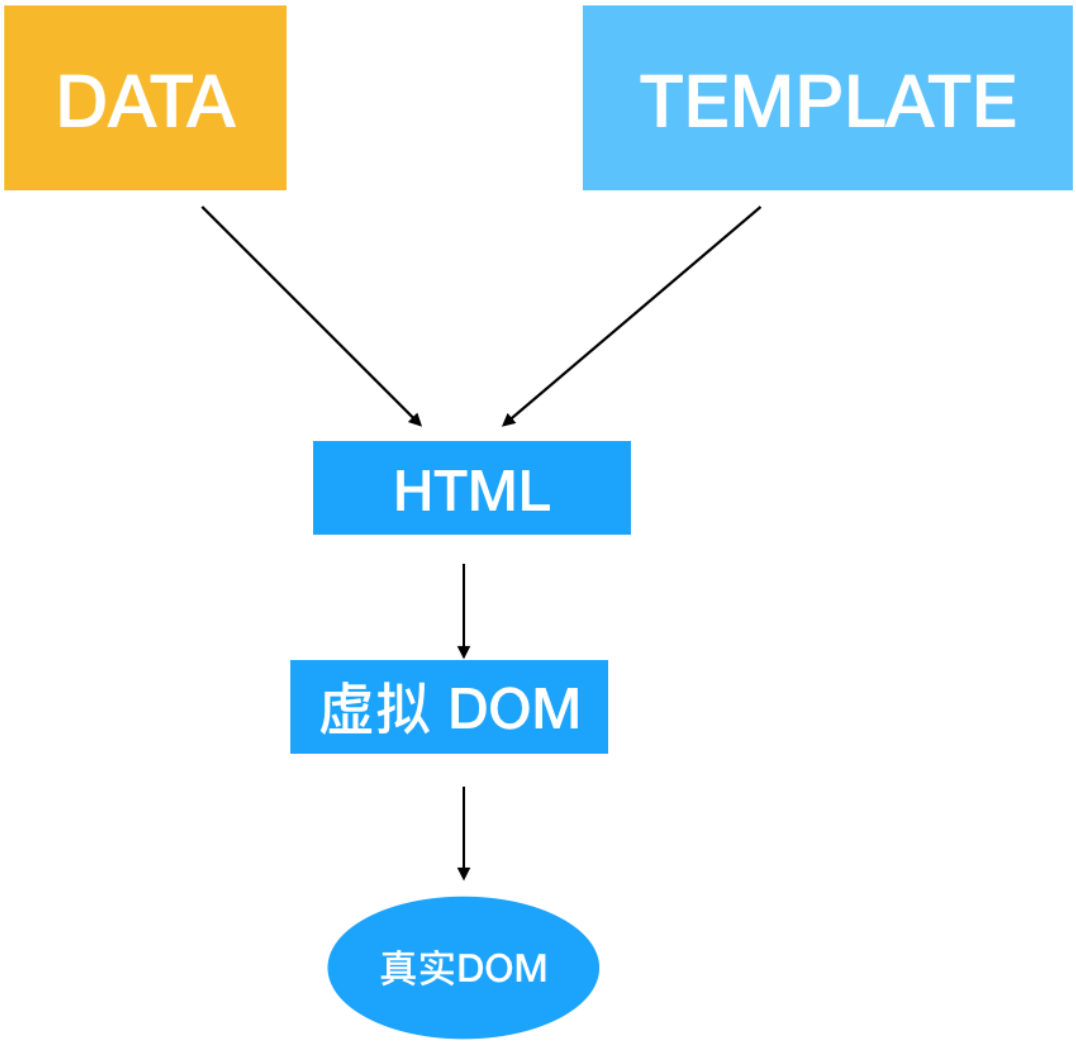
当更新发生时：

```
注销旧 DOM -> 数据 + 模板 => 新的一套HTML 代码 -> 挂载新 DOM
```

这里的“旧 DOM”、“新 DOM”指的都是模板对应的整块 DOM 的整体更新。我们错就错在每次都整体更新——如果有一种方法，可以既帮我们保持住模板方案的数据驱动思想，又做到像人肉 JS、jQuery 一样能够定点只对需要修改的 DOM 做小范围操作，那该多好！

你品，你细品！DOM 操作从“一刀切”到“精细化”，中间需要的是啥？需要的是 diff ！

虚拟 DOM + diff，新的 DOM 操作解决方案应运而生！初始化过程是这样的：



当更新发生时：

```
数据 + 模板 = 虚拟 DOM -> diff 新旧两套虚拟 DOM 的差异，得到补丁集 -> 把“补丁”打到真实 DOM 上
```

其中，虚拟 DOM 这一层是用 JS 实现的。也就是说在这个阶段所有的更改、对比操作都是纯 JS 层面的计算。JS vs DOM操作，其性能消耗完全不在一个量级上。

如此一来，简单粗暴的“删了重写”，变成了灵活精确的“定点打击”！

模板渲染带来的性能问题，就这样被 Virtual DOM 完美地解决了。

}

32 React基本功（三）——命题思路点拨
34 真正理解虚拟DOM（二）——Diff算法 & 面试考点解析