26 | 虚拟DOM：虚拟DOM和实际的DOM有何不同?

2019-10-03 李兵

浏览器工作原理与实践

进入课程 >

讲述：李兵
时长 10:32　大小 12.06M

虚拟 DOM 是最近非常火的技术，两大著名前端框架 React 和 Vue 都使用了虚拟 DOM，所以我觉得非常有必要结合浏览器的工作机制对虚拟 DOM 进行一次分析。当然了，React 和 Vue 框架本身所蕴含的知识点非常多，而且也不是我们专栏的重点，所以在这里我们还是把重心聚焦在虚拟 DOM 上。

在本文我们会先聊聊 DOM 的一些缺陷，然后在此基础上介绍虚拟 DOM 是如何解决这些缺陷的，最后再站在双缓存和 MVC 的视角来聊聊虚拟 DOM。理解了这些会让你对目前的前端框架有一个更加底层的认识，这也有助于你更好地理解这些前端框架。

DOM 的缺陷

通过前面一系列文章的学习，你对 DOM 的生成过程应该已经有了比较深刻的理解，并且也知道了通过 JavaScript 操纵 DOM 是会影响到整个渲染流水线的。另外，DOM 还提供了一组 JavaScript 接口用来遍历或者修改节点，这套接口包含了 getElementById、removeChild、appendChild 等方法。

比如，我们可以调用`document.body.appendChild(node)`往 body 节点上添加一个元素，调用该 API 之后会引发一系列的连锁反应。首先渲染引擎会将 node 节点添加到 body 节点之上，然后触发样式计算、布局、绘制、栅格化、合成等任务，我们把这一过程称为**重排**。除了重排之外，还有可能引起**重绘**或者**合成**操作，形象地理解就是“**牵一发而动全身**”。另外，对于 DOM 的不当操作还有可能引发**强制同步布局**和**布局抖动**的问题，这些操作都会大大降低渲染效率。因此，对于 DOM 的操作我们时刻都需要非常小心谨慎。

当然，对于简单的页面来说，其 DOM 结构还是比较简单的，所以以上这些操作 DOM 的问题并不会对用户体验产生太多影响。但是对于一些复杂的页面或者目前使用非常多的单页应用来说，其 DOM 结构是非常复杂的，而且还需要不断地去修改 DOM 树，每次操作 DOM 渲染引擎都需要进行重排、重绘或者合成等操作，因为 DOM 结构复杂，所生成的页面结构也会很复杂，对于这些复杂的页面，执行一次重排或者重绘操作都是非常耗时的，这就给我们带来了真正的性能问题。

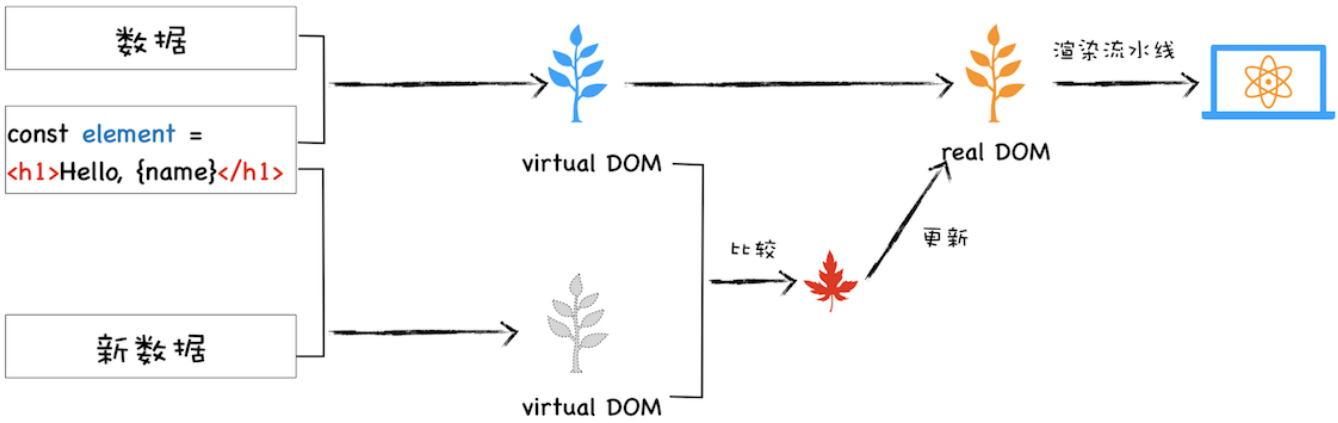
所以我们需要有一种方式来减少 JavaScript 对 DOM 的操作，这时候虚拟 DOM 就上场了。

什么是虚拟 DOM

在谈论什么是虚拟 DOM 之前，我们先来看看虚拟 DOM 到底要解决哪些事情。

- 将页面改变的内容应用到虚拟 DOM 上，而不是直接应用到 DOM 上。
- 变化被应用到虚拟 DOM 上时，虚拟 DOM 并不急着去渲染页面，而仅仅是调整虚拟 DOM 的内部状态，这样操作虚拟 DOM 的代价就变得非常轻了。
- 在虚拟 DOM 收集到足够的改变时，再把这些变化一次性应用到真实的 DOM 上。

基于以上三点，我们再来看看什么是虚拟 DOM。为了直观理解，你可以参考下图：



虚拟 DOM 执行流程

该图是我结合 React 流程画的一张虚拟 DOM 执行流程图，下面我们就结合这张图来分析下虚拟 DOM 到底怎么运行的。

> **创建阶段**。首先依据 JSX 和基础数据创建出来虚拟 DOM，它反映了真实的 DOM 树的结构。然后由虚拟 DOM 树创建出真实 DOM 树，真实的 DOM 树生成完后，再触发渲染流水线往屏幕输出页面。
>
> **更新阶段**。如果数据发生了改变，那么就需要根据新的数据创建一个新的虚拟 DOM 树；然后 React 比较两个树，找出变化的地方，并把变化的地方一次性更新到真实的 DOM 树上；最后渲染引擎更新渲染流水线，并生成新的页面。

既然聊到虚拟 DOM 的更新，那我们就不得不聊聊最新的**React Fiber 更新机制**。通过上图我们知道，当有数据更新时，React 会生成一个新的虚拟 DOM，然后拿新的虚拟 DOM 和之前的虚拟 DOM 进行比较，这个过程会找出变化的节点，然后再将变化的节点应用到 DOM 上。

这里我们重点关注下比较过程，最开始的时候，比较两个虚拟 DOM 的过程是在一个递归函数里执行的，其**核心算法是 reconciliation**。通常情况下，这个比较过程执行得很快，不过当虚拟 DOM 比较复杂的时候，执行比较函数就有可能占据主线程比较久的时间，这样就会导致其他任务的等待，造成页面卡顿。为了解决这个问题，React 团队重写了 reconciliation 算法，新的算法称为 Fiber reconciler，之前老的算法称为 Stack reconciler。

在前面《20 | async/await：使用同步的方式去写异步代码》那篇文章中我们介绍了协程，其实协程的另外一个称呼就是 Fiber，所以在这里我们可以把 Fiber 和协程关联起来，那么

所谓的 Fiber reconciler 相信你也很清楚了，就是在执行算法的过程中出让主线程，这样就解决了 Stack reconciler 函数占用时间过久的问题。至于具体的实现过程在这里我就不详细分析了，如果感兴趣的话，你可以自行查阅相关资料进行学习。

了解完虚拟 DOM 的大致执行流程，你应该也就知道为何需要虚拟 DOM 了。不过以上都从单纯的技术视角来分析虚拟 DOM 的，那接下来我们再从双缓存和 MVC 模型这两个视角来聊聊虚拟 DOM。

1. 双缓存

在开发游戏或者处理其他图像的过程中，屏幕从前缓冲区读取数据然后显示。但是很多图形操作都很复杂且需要大量的运算，比如一幅完整的画面，可能需要计算多次才能完成，如果每次计算完一部分图像，就将其写入缓冲区，那么就会造成一个后果，那就是在显示一个稍微复杂点的图像的过程中，你看到的页面效果可能是一部分一部分地显示出来，因此在刷新页面的过程中，会让用户感受到界面的闪烁。

而使用双缓存，可以让你先将计算的中间结果存放在另一个缓冲区中，等全部的计算结束，该缓冲区已经存储了完整的图形之后，再将该缓冲区的图形数据一次性复制到显示缓冲区，这样就使得整个图像的输出非常稳定。

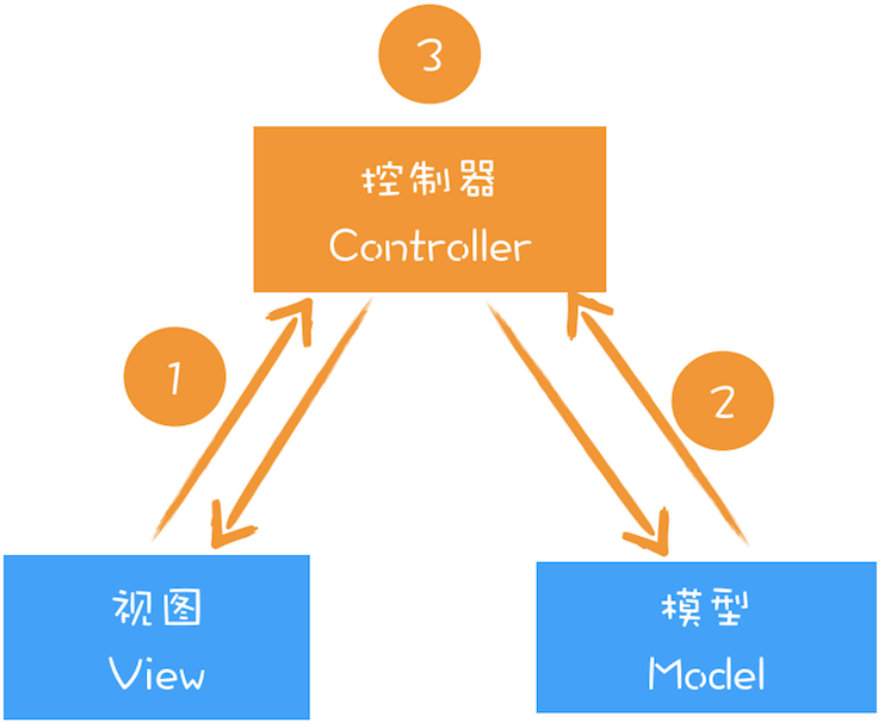
在这里，你可以把虚拟 DOM 看成是 DOM 的一个 buffer，和图形显示一样，它会在完成一次完整的操作之后，再把结果应用到 DOM 上，这样就能减少一些不必要的更新，同时还能保证 DOM 的稳定输出。

2. MVC 模式

到这里我们了解了虚拟 DOM 是一种类似双缓存的实现。不过如果站在技术角度来理解虚拟缓存，依然不能全面理解其含义。那么接下来我们再来看看虚拟 DOM 在 MVC 模式中所扮演的角色。

在各大设计模式当中，MVC 是一个非常重要且应用广泛的模式，因为它能将数据和视图进行分离，在涉及到一些复杂的项目时，能够大大减轻项目的耦合度，使得程序易于维护。

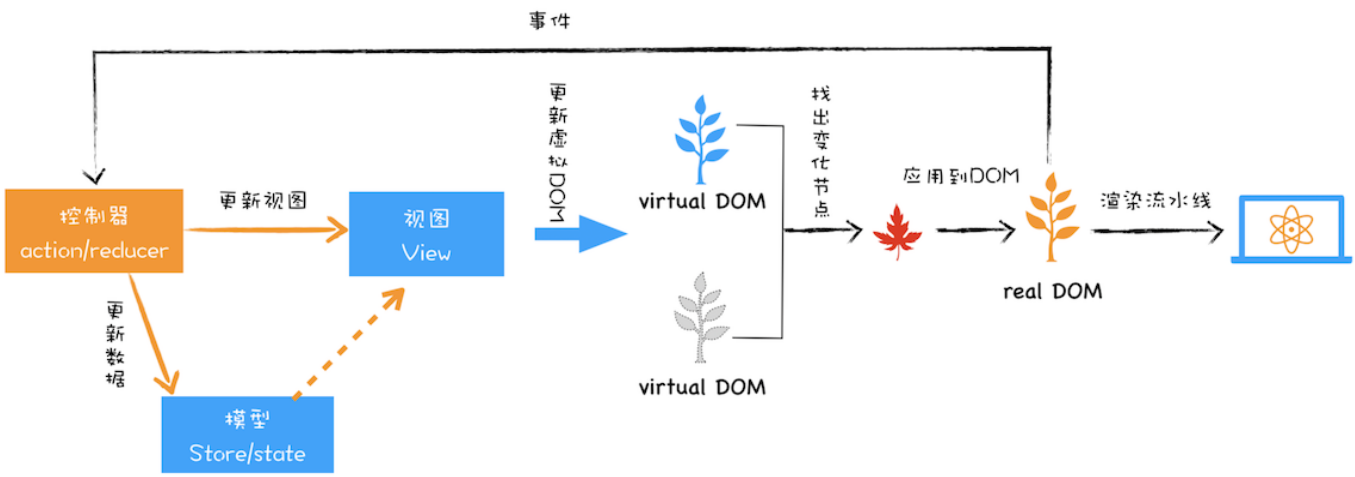
关于 MVC 的基础结构，你可以先参考下图：



MVC 基础结构

通过上图你可以发现，MVC 的整体结构比较简单，由模型、视图和控制器组成，其**核心思想就是将数据和视图分离**，也就是说视图和模型之间是不允许直接通信的，它们之间的通信都是通过控制器来完成的。通常情况下的通信路径是视图发生了改变，然后通知控制器，控制器再根据情况判断是否需要更新模型数据。当然还可以根据不同的通信路径和控制器不同的实现方式，基于 MVC 又能衍生出很多其他的模式，如 MVP、MVVM 等，不过万变不离其宗，它们的基础骨架都是基于 MVC 而来。

所以在分析基于 React 或者 Vue 这些前端框架时，我们需要先重点把握大的 MVC 骨架结构，然后再重点查看通信方式和控制器的具体实现方式，这样我们就能从架构的视角来理解这些前端框架了。比如在分析 React 项目时，我们可以把 React 的部分看成是一个 MVC 中的视图，在项目中结合 Redux 就可以构建一个 MVC 的模型结构，如下图所示：



在该图中，我们可以把虚拟 DOM 看成是 MVC 的视图部分，其控制器和模型都是由 Redux 提供的。其具体实现过程如下：

图中的控制器是用来监控 DOM 的变化，一旦 DOM 发生变化，控制器便会通知模型，让其更新数据；

模型数据更新好之后，控制器会通知视图，告诉它模型的数据发生了变化；

视图接收到更新消息之后，会根据模型所提供的数据来生成新的虚拟 DOM；

新的虚拟 DOM 生成好之后，就需要与之前的虚拟 DOM 进行比较，找出变化的节点；

比较出变化的节点之后，React 将变化的虚拟节点应用到 DOM 上，这样就会触发 DOM 节点的更新；

DOM 节点的变化又会触发后续一系列渲染流水线的变化，从而实现页面的更新。

在实际工程项目中，你需要学会分析出这各个模块，并梳理出它们之间的通信关系，这样对于任何框架你都能轻松上手了。

总结

好了，今天就介绍到这里，下面我来总结下本文的主要内容。

首先我们分析了直接操作 DOM 会触发渲染流水线的一系列反应，如果对 DOM 操作不当的话甚至还会触发强制同步布局和布局抖动的问题，这也是我们在操作 DOM 时需要非常小心谨慎的原因。

在此分析的基础上，我们介绍了虚拟 DOM 是怎么解决直接操作 DOM 所带来的问题以及 React Fiber 更新机制。

要聊前端框架，就绕不开设计模式，所以接下来我们又从双缓存和 MVC 角度分析了虚拟 DOM。双缓存是一种经典的思路，应用在很多场合，能解决页面无效刷新和闪屏的问题，虚拟 DOM 就是双缓存思想的一种体现。而基于 MVC 的设计思想也广泛地渗透到各种场合，并且基于 MVC 又衍生出了很多其他模式（如 MVP、MVVM 等），不过万变不离其宗，它们的基础骨架都是基于 MVC 而来。站在 MVC 视角来理解虚拟 DOM 能让你看到更为“广阔的世界”。

思考时间

今天留给你的思考题是：虚拟 DOM 都解决了哪些问题？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

上一篇　25 | 页面性能：如何系统地优化页面？

下一篇　27 | 渐进式网页应用（PWA）：它究竟解决了Web应用的哪些问题？

精选留言 (6)

写留言

mfist
2019-10-03

1. 频繁更新dom引起的性能问题
2. 将真实DOM和js操作解耦，减少js操作dom复杂性。
老师答疑的时候可以介绍下react的fiber吗？感觉李老师的文章通俗易懂，很受益（网上文章分析参差不齐的）

今日总结：...

展开

3

小兵

2019-10-03

老师，文中的虚拟Dom收集到足够的变化是什么意思？会不会导致页面的响应变慢？

2

宇宙全栈

2019-10-04

基于 React 和 Redux 构建 MVC 模型的配图中，控制器是不是不能直接改变视图？因为 redux 模型是单向数据流吧

展开

隔夜果酱

2019-10-03

李老师后面会考虑React源码类的专栏么?
网上的视频和文章很多都流于表面,或者生搬硬套的进行解释,看的让人头大.
如果李老师有精力和兴趣的话,希望可以开专栏为我们指点迷津呀.

展开

柒月

2019-10-03

主要还是解决频繁操作DOM引起页面响应慢的问题。
虚拟DOM就是一个JS对象，通过diff算法比较新老DOM树的差别，来达到最小化局部更新的目的。
其本质不过是用JS的运算性能的消耗来换取操作DOM的性能消耗。

展开

Hurry

2019-10-03

频繁DOM操作是非常消耗浏览器性能的，虚拟DOM核心还是将批量DOM操作后的变化一次性更新到浏览器。

展开