ISSN 1009-3044
Computer Knowledge and Technology 电脑知识与技术
Vol.13, No18, June 2017

E-mail: xsjl@dnzs.net.cn
http://www.dnzs.net.cn
Tel:+86-551-65690963 65690964

基于Dom Diff 算法分析React 刷新机制

严新巧,白俊峰

(四川托普信息技术职业学院 计算机系,四川 成都 611743)

摘要: React 组件化思想为开发者前端开发提供了新的思路,由于 React的Visual Dom让开发者不用担心刷新页面带来渲染方面性能问题,而Visual Dom的核心算法就是React Diff算法,它确保只对界面上需要刷新的部分进行刷新,让开发者只需关注于业务本身。该文基于对React Diff算法的研究来分析React组件的生命周期,理解Virtual Dom 的核心算法,方便以后React程序优化。

关键词: React 组件;虚拟Dom 算法;渲染

中图分类号:TP312 文献标识码:A 文章编号:1009-3044(2017)18-0076-03

React Refresh Mechanism Analysis Based on Virtual Dom Diff Algorithm

YAN Xin-qiao, BAI Jun-feng

(Sichuan TOP IT Vocational Institute, Chengdu 611743, China)

Abstract: React component-based thinking allows developers to have a new ideal for front-end development. The developer do not have to care about the rendering performance problems of refresh the page due to Visual Dom. The Visual Dom core algorithm is React Diff algorithm, which ensures that only the portion of the interface need to refresh. That allows developers only need to focus on the business itself. This paper is based on an algorithm React Diff analysis lifecycle React components, understanding the core algorithm Virtual Dom, React on future programs can be optimized.

Key words: React component; Virtual Dom Algorithm; render

1 背景介绍

对于动态网站,需要经常对网页的元素进行操作,一般通过Dom树结点的操作来更新界面,对于一些比较复杂的界面,需频繁操作Dom结点会导致性能问题,并且对开发者而言,增加了开发的难度。基于此,React专门实现了一套Visual Dom机制。基于React开发程序的操作都是通过内置的Visual Dom来实现,当有数据状态发生改变时,会将前后两个Dom树进行对比,对比后的结果只对有区别的部分进行更新。基于一些策略,React能更好地处理Dom树发生改变的部分。

Visual Dom的核心算法就是React Diff,它通过算法的优化,让Dom树的更新不再是难题,这也让开发者不用担心由于Dom结点更新而导致的性能问题,通过计算出每次更新的Dom结点,只需要对局部进行操作就可以渲染出界面,这样保证了更新的效率。

2 传统的Diff算法与Virtual Dom 算法介绍

传统web应用,一般是直接更新DOM操作,但是DOM更新通常比较昂贵。React为尽可能减少对DOM的操作,提供一种不同且强大的方式来更新DOM,从而代替直接DOM操作。Virtual DOM,一个轻量级虚拟的DOM,是React抽象出来的一个对象,描述DOM的样子,以及如何呈现。通过Virtual DOM更新真实DOM,由Virtual DOM管理真实DOM的更新。

Virtual DOM操作更新更快,因为React的diff算法如图1,更新Virtual DOM并不一定立即影响真实DOM,React会等到事件循环结束,然后利用diff算法,通过当前新DOM表述与之前的作比较,计算出最少步骤更新真实DOM。

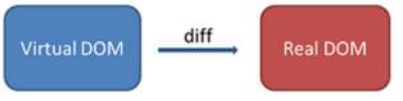

图1 Virtual DOM操作

对于一次刷新操作,传统的计算方法是通过循环递归进行一一对比,从而分析出需要改变的Dom树,其中的算法复杂度是$O(n^3)$,这就意味着如果对1000个节点进行比较的话,传统算法就需要进行十亿次操作。

传统 diff 算法的复杂度为 $O(n^3)$,显然这无法满足性能要求。React 通过制定策略,将 $O(n^3)$ 复杂度转换成 O(n) 复杂度。

由于传统算法的复杂度无法满足性能要求,而React通过算法的优化,将指数的复杂度变成线性的复杂度,因为Dom树在应用中具有以下特点,才让Visual Dom得以实现:

1)对于一些节点操作来说,Dom结点之间的跨层操作是非常少的

2)对于同一个类生成相同Dom的树形结构,不同的类生成不同的Dom树结构。

收稿日期:2017-05-09

3)对于同一层次的子结点,他们的ID是不一样的

3 Diff 策略分析

基于以上三个Dom的特点,React分别用三种不同的算法tree diff、component diff以及element diff进行Visual Dom操作。

3.1 tree diff 策略

对于特点1,React采用了tree diff算法,因为更新都是基于同一层次的结点进行比较,即每一次只对同一层次的节点进行比较。这样的操作对于Dom而言操作较少,React通过updateDepth算法来对Virtual Dom进行比较,即它会对同一个父结点进行子结点比较,当发现节点不存在了,则把该节点以及子结点都删除,通过这个算法,只需要一次遍历操作就可以完成对整个Dom树的比较。

下面通过一个例子来说明这种情况,如图2,需要将A结点以及A下面的结点移到到D结点下,由于React只对同一层次的结点考虑位置变换,所以对于这种情况,就会存在着创建与删除的操作。当根结点发现A结点消失后,就会直接删除A,而当发现D多了一个结点后,就会在D下面创建结点与子结点,所以它们的执行顺序是:create A → create B → create C → delete A。

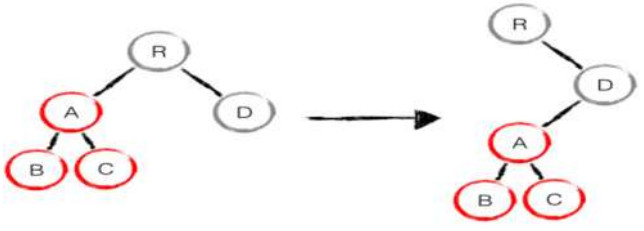

图2　DOM节点跨层级的移动操作

由上面例子说明,当节点出现跨层次的移动时,是通过创建与删除操作来执行的,这样比较影响性能,所以在使用React开发的时候尽量不要使用这个跨层次操作,如果在实际应用中必须有此需求,建议采用CSS隐藏或者显示节点操作。

3.2 component diff算法

对同一个组件更新时,如果内部没有发生变化,则继续后面的更新,同时React也会提供特定的算法来进行组件更新。若非同一个类型的组件,则会将该组件判断为dirty组件,会直接替换组件下的所有节点如图3。

当结点D需要替换成结点G时,即使他们的结构很类似,并且子结点都相同,但是React还是会直接删除结点D,并且重新创建结点G,而不是直接进行简单的替换。虽然这种算法会影响性能,但是对于这种情况相对较少,React没有进行优化,因为在实际应用中很难因为此操作而影响到性能问题。

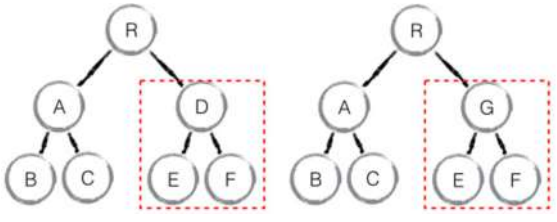

图3　不同类型但结构相似的组件

3.3 element diff

React对同一层次结点进行大量的操作,因为基于此操作的情况最多,React会提供三种不同节点的操作:INSERT_MARKUP(插入)、MOVE_EXISTING(移动)和REMOVE_NODE(删除)。

如果是全新的节点,则直接进行插入操作,而如果是可以移动的操作,则复用以前的操作,如果不能直接进行复用与更新操作,则进行删除操作。

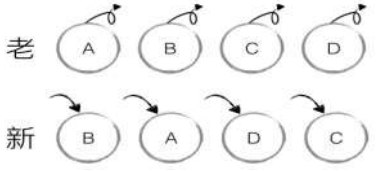

图4　相同节点的删除和创建

如图4,如果之前的Dom树上包含结点ABCD,更新后的顺序为BADC,这个时候就会进行diff差异化比较,发现新的结点是B,这时会先删除A,再插入B,然后删除BCD,插入ADC,由些可见,这个操作也是比较繁琐,并且没有效率,执行移动操作会比执行插入与删除有效率得多,React针对这种情况提出了优化操作,允许对同一层次的同一个位置的结点,通过不同的Key来进行区分。

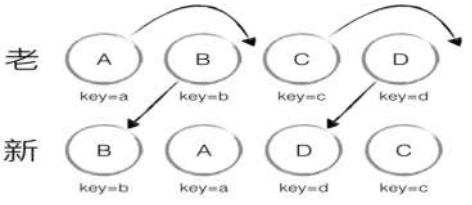

图5　唯一Key区分同组子节点

如图5,与上例相同,对新老集合的节点进行diff差异化对比,通过key值来发现新老Dom组的节点都是相同结点,所以不需要进行删除与创建操作,只需进行简单的移动操作就可以达到目标。下面通过源码进行详细分析利用diff达到上述操作。

首先对新老Dom树的节点进行一次循环遍历,通过唯一的key值来判断新老Dom树的节点是否相同。如果结点相同,则进行移动操作,但在移动操作之前会将当前节点顺序与最后一个结点顺序进行比较,若非最后一个节点,才会进行移动操作。

以上图5为例,更为清晰直观描述diff差异对比过程:

从新集合中取得B,判断老集合中存在相同节点B,通过对比节点位置判断是否进行移动操作,B在老集合中的位置B._mountIndex = 1,此时lastIndex = 0,不满足child._mountIndex < lastIndex的条件,因此不对B进行移动操作;更新lastIndex = Math.max(prevChild._mountIndex, lastIndex),其中prevChild._mountIndex表示B在老集合中的位置,则lastIndex = 1,并将B的位置更新为新集合中的位置prevChild._mountIndex = nextIndex,此时新集合中B._mountIndex = 0,nextIndex++进入下一个节点的判断。

从新集合中取得A,判断老集合中存在相同节点A,通过对比节点位置判断是否进行移动操作,A在老集合中的位置A._mountIndex = 0,此时lastIndex = 1,满足child._mountIndex < lastIndex的条件,因此对A进行移动操作enqueueMove(this, child._mountIndex, toIndex),其中toIndex其实就是nextIndex,表示A需要移动到的位置;更新lastIndex = Math.max(prevChild._mountIndex, lastIndex),则lastIndex = 1,并将A的位置更新为新集合中的位置prevChild._mountIndex = nextIndex,此时新集合中A._mountIndex = 1,nextIndex++进入下一个节点的判断。

从新集合中取得D，判断老集合中存在相同节点D，通过对比节点位置判断是否进行移动操作，D在老集合中的位置D._mountIndex = 3，此时lastIndex = 1，不满足child._mountIndex < lastIndex的条件，因此不对D进行移动操作；更新lastIndex = Math.max(prevChild._mountIndex, lastIndex)，则lastIndex = 3，并将D的位置更新为新集合中的位置prevChild._mountIndex = nextIndex，此时新集合中D._mountIndex = 2，nextIndex++进入下一个节点的判断。

从新集合中取得C，判断老集合中存在相同节点C，通过对比节点位置判断是否进行移动操作，C在老集合中的位置C._mountIndex = 2，此时lastIndex = 3，满足child._mountIndex < lastIndex的条件，因此对C进行移动操作enqueueMove(this, child._mountIndex, toIndex)；更新lastIndex = Math.max(prevChild._mountIndex, lastIndex)，则lastIndex = 3，并将C的位置更新为新集合中的位置prevChild._mountIndex = nextIndex，此时新集合中A._mountIndex = 3，nextIndex++进入下一个节点的判断，由于C已经是最后一个节点，因此diff到此完成。

以上情况主要针对新老Dom存在节点相同，但位置不同时进行的移动操作，而当新集合中存在新的节点插入并且需要执行删除操作时，需要进行diff操作。

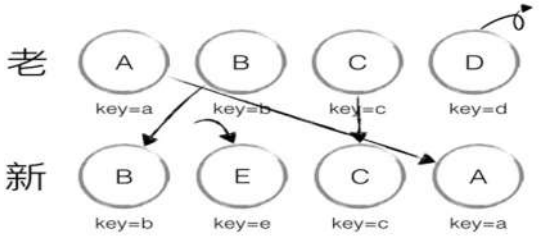

图6 加入新节点和删除老节点

如图6，新的Dom在判断第一个节点是B时，会在老Dom里去查找是否有B，而B在老集合的位置序号是1，因为当前判断节点是0，所以不需要进行移动操作，更新当前节点位置为1，并将B的位置序号改为0，这样依次循环。

从新集合中取得E，判断老集合中不存在相同节点E，则创建新节点E；更新lastIndex = 1，并将E的位置更新为新集合中的位置，nextIndex++进入下一个节点的判断。

依次进行上述操作，对所有的结点进行Diff操作后，最后需要对老Dom树进行遍历操作，判断是否有删除操作，在上述例子中，需要删除D，至此所有操作完成。

4 总结

React通过模拟一个新的Visual Dom来解决更新问题，从而将复杂度为$O(n^3)$转换成线性复杂度问题，这样非常适合处理一些复杂的应用场景。但通过算法的具体分析发现，在写React时应该尽量保持Dom树的结构，并尽量减少大范围的移动结点操作，从而使得React的渲染能力达到最优。

5 结束语

本文通过对React的Diff算法分析，深入剖析了Visual dom的更新原则，这让开发者在写代码时无需关心这部分内容，而只需注重业务逻辑问题，从而简化了开发过程。正是由于第2节中提到的Dom的三个基本特性，使得算法得以实现。通过对算法的深入研究，对今后React开发优化有了深入的理论基础。

参考文献：

[1] 武佳佳, 王建忠. 基于HTML5实现智能手机跨平台应用开发[J]. 软件导刊, 2013(2): 66-67.
[2] 唐灿. 下一代Web界面前端技术综述[J]. 重庆工商大学学报:自然科学版, 2009(4): 351-355.
[3] 张景安, 张杰, 卫泽丹. Android移动端浏览器的设计与开发[J]. 软件, 2014, 35(8): 7-10.
[4] Facebook.React Native[EB/OL]. (2015-12).http://facebook.github.io/react-native.
[5] Facebook.React JS[EB/OL]. (2014-07).http://facebook.github.io/react/.

(上接第87页)

3 系统的实现

系统基于Visual studio2010+sql server2008平台进行开发，为便于后期维护和功能扩充，系统的实现过程完全采用三层架构、面向对象，对于数据库数据的操作尽量用sql server2008自带的存储过程功能来实现，需要的数据库操作结果通过存储过程返回。

为便于找到报道入口，在学校的门户网站、招生服务平台、微信公众平台、微博平台等处设置报道服务平台的链接。

5 结束语

高职院校学生注册(报到)服务平台的使用可极大地提高报到工作效率，减少人力、物力和财力成本，可极大地避免因报到人员、家长及服务人员相对集中而带来的安全隐患，为学生和学校提供便利。

参考文献：

[1] 钱春花, 刘海明, 强鹤群,等. 基于Web的高职院校新生报到管理平台的设计与实现[J]. 电脑迷, 2016(11): 127.
[2] 刘恒祥. 高职院校新生录取系统的设计与实现[J]. 电子设计工程, 2016(8): 109-112.
[3] 朱伟华, 刘志宝. 基于C#的高职新生报到信息管理系统设计与实现[J]. 软件工程师, 2015(7): 30-32.
[4] 杨振宇. 高校招生数据采集及分析系统设计与实现[J]. 软件, 2015(5): 61-66.
[5] 赵均水. 微信在高职院校新生报到管理中的应用[J]. 中国管理信息化, 2017(8): 210-211.