



Q

从所有教程的词条中查询…

从所有教柱的问余中亘词"

◆ 上一节 06: 框架实现: ... 下一节 →

首页 > 慕课教程 > Vue3源码分析与构建方案 > 07:源码阅读:场景三:新节点多余旧节点时...

## 

16: 总结

## 第十二章:runtime 运行时 - diff 算法核心实现

01: 前言

02: 前置知识: VNode 虚拟 节点 key 属性的作用

03:源码阅读:场景一:自前向后的 diff 对比

04: 框架实现: 场景一: 自前 向后的 diff 对比

05:源码阅读:场景二:自后 向前的 diff 对比

06: 框架实现: 场景二: 自后 向前的 diff 对比

## 07:源码阅读:场景三:新节 点多余旧节点时的 diff 比对

08: 框架实现: 场景三: 新节点多余旧节点时的 diff 比对

09:源码阅读:场景四:旧节 点多于新节点时的 diff 比对

## 07: 源码阅读: 场景三: 新节点多余旧节点时的 diff 比对

以上两种场景,新节点数量和旧节点数量都是完全一致的。

但是我们也知道一旦产生更新,那么新旧节点的数量是可能会存在不一致的情况,具体的不一致情况会分为两种:

1. 新节点的数量多于旧节点的数量

Sunday • 更新于 2022-10-19

2. 旧节点的数量多于新节点的数量

那么针对于这两种情况怎么进行处理呢?这就是我们接下来需要说的。

本小节,我们先来看一下 **新节点的数量多于旧节点的数量** 的场景,一旦出现这种场景,那么我们就需要 **挂载多余的新节点。** 

但是新节点的数量多于旧节点的数量的场景下,依然可以被细分为两种具体的场景:

- 1. 多出的新节点位于 尾部
- 2. 多出的新节点位于 头部

▶ 意见反馈

明确好了以上内容之后,我们来看如下测试实例 packages/vue/examples/imooc/runtime/render-elemen

```
<>代码块
 1
     <script>
 2
       const { h, render } = Vue
 3
       const vnode = h('ul', [
 4
 5
         h('li', {
           key: 1
         }, 'a'),
         h('li', {
 8
 9
           key: 2
         }, 'b'),
10
11
     ])
       // 挂载
12
13
       render(vnode, document.querySelector('#app'))
14
15
      // 延迟两秒,生成新的 vnode,进行更新操作
16
      setTimeout(() => {
        const vnode2 = h('ul', [
17
18
           h('li', {
19
            key: 1
20
           }, 'a'),
21
           h('li', {
22
            key: 2
           }, 'b'),
23
           h('li', {
24
25
            key: 3
26
           }, 'c')
27
28
         render(vnode2, document.querySelector('#app'))
       }, 2000);
29
    </script>
30
```

♡ 收藏教程

□ 标记书签

索引目录

07: 源码阅读: 均

?

··

根据以上代码进入 debugger , 忽略掉前两种场景 , 直接从第三种场景开始:

1. 代码进入场景三 3. common sequence + mount , 此时各参数的值为:

```
>c1: (2) [{...}, {...}]
>c2: (3) [{...}, {...}, {...}]
>container: ul
e1: 1
e2: 2
i: 2
isSVG: false
l2: 3
```

- 2. 满足 if (i > e1) 和 if (i <= e2) 的场景, 进入 if
  - 1. 执行:

```
const nextPos = e2 + 1
const anchor = nextPos < 12 ? (c2[nextPos] as VNode).el : parentAnchor</pre>
```

?

 $\odot$ 

- 1. 这两行代码的目的是为了计算出 anchor ,也就是**新增节点插入的 锚点(位置)。**
- 2. 当前场景下 anchor = parentAnchor , 即: 使用父级的 anchor
- 2. 执行 while (i <= e2) , 进入循环
- 3. 每次循环时, 执行 patch 方法, 新增节点即可。
- 4. 最后执行 i++

以上逻辑为:多出的新节点位于尾部的场景。

那么接下来我们来看:多出的新节点位于头部的场景:

▶ 意见反馈

```
<> 代码块
1 <script>
const { h, render } = Vue
 4
   const vnode = h('ul', [
      h('li', {
 5
        key: 1
 6
       }, 'a'),
       h('li', {
 8
 9
         key: 2
      }, 'b'),
10
    ])
11
   // 挂载
12
    render(vnode, document.querySelector('#app'))
13
14
15 // 延迟两秒, 生成新的 vnode, 进行更新操作
setTimeout(() => {
17
     const vnode2 = h('ul', [
18
       h('li', {
         key: 3
19
       }, 'c'),
20
21
        h('li', {
22
         key: 1
       }, 'a'),
23
       h('li', {
24
         key: 2
25
       }, 'b')
26
```

♡ 收藏教程

口 标记书签

```
29
    }, 2000);
30 </script>
根据以上代码,再次进入情景三:
1. 代码进入场景三 3. common sequence + mount , 此时各参数的值为:
       ▶c1: (2) [{...}, {...}]
▶c2: (3) [{...}, {...}, {...}]
       ▶ container: ul
         e1: -1
         e2: 0
         i: 0
         isSVG: false
         12: 3
2. 代码执行:
   <>代码块
      const nextPos = e2 + 1
      const anchor = nextPos < 12 ? (c2[nextPos] as VNode).el : parentAnchor</pre>
   1. 此时 nextPos > 12 , 所以 anchor 的值为 c2[1] , 即: 新增的节点位于 h('li', { key: 1 },
    'a') 之前
3. 后续的代码执行略过
由以上代码可知:
1. 对于 新节点多余旧节点 的场景具体可以在细分为两种情况:
   1. 多出的新节点位于 尾部
   2. 多出的新节点位于 头部
2. 这两种情况下的区别在于: 插入的位置不同
3. 明确好插入的位置之后,直接通过 patch 进行打补丁即可。

✓ 我要提出意见反馈
```

企业服务 网站地图 网站首页 关于我们 联系我们 讲师招募 帮助中心 意见反馈 代码托管

Copyright © 2022 imooc.com All Rights Reserved | 京ICP备 12003892号-11 京公网安备11010802030151号

**★ %** ♂ **4** 

?

0

✓ 意见反馈

✓ 收藏教程

□ 标记书签