

全部开发者教程

10：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 对比

08：框架实现：场景三：新节点多余旧节点时的 diff 对比

09：源码阅读：场景四：旧节点多于新节点时的 diff 对比



Sunday • 更新于 2022-10-19

上一节 04：框架实现：... 06：框架实现：... 下一节

05：源码阅读：场景二：自后向前的 diff 对比

上一小节的代码，只可能处理 **自前向后完全相同的 vnode**。如果 vnode 不是自前向后完全相同的则无法进行处理，比如我们看下面的例子 `packages/vue/examples/imooc/runtime/render-element-diff-2.html`：

<> 代码块

```
1  <script>
2    const { h, render } = Vue
3
4    const vnode = h('ul', [
5      h('li', {
6        key: 1
7      }, 'a'),
8      h('li', {
9        key: 2
10     }, 'b'),
11     h('li', {
12       key: 3
13     }, 'c'),
14   ])
15   // 挂载
16   render(vnode, document.querySelector('#app'))
17
18   // 延迟两秒，生成新的 vnode，进行更新操作
19   setTimeout(() => {
20     const vnode2 = h('ul', [
21       h('li', {
22         key: 4
23       }, 'a'),
24       h('li', {
25         key: 2
26       }, 'b'),
27       h('li', {
28         key: 3
29       }, 'd')
30     ])
31     render(vnode2, document.querySelector('#app'))
32   }, 2000);
33 </script>
```

在上面的例子中，vnode 2 的第一个子节点的 key = 4，这就会导致一个情况：****如果我们从前往后进行 diff 对比，那么第一个 child 无法满足 isSameVNodeType，就会直接跳出****

所以以上案例，在 **我们现在的 vue-next-mini 中**，是无法进行正确更新的。

那么想要以上场景可以正确更新，就需要继续来查看 **vue** 中对于 **diff** 的第二个场景处理 **自后向前的 diff 对比**：

1. 进入 `patchKeyedChildren` 方法，此时各参数的值为：

索引目录

05：源码阅读：场景二：自后向前的 diff 对比



```
const patchKeyedChildren = (
  c1: VNode[], c1 = (3) [{...}, {...}, {...}]
  c2: VNodeArrayChildren, c2 = (3) [{...}, {...}, {...}]
  container: RendererElement, container = ul {__vnode: {...}, __vuePa
  parentAnchor: RendererNode | null, parentAnchor = null
  parentComponent: ComponentInternalInstance | null, parentComponen
  parentSuspend: SuspenseBoundary | null, parentSuspend = null
  isSVG: boolean, isSVG = false
  slotScopeIds: string[] | null, slotScopeIds = null
  optimized: boolean optimized = false
) => {
```

1. 其中 `c1` 表示为：旧的子节点，即： `oldChildren`
2. `c2` 表示为：新的子节点，即： `newChildren`
2. 执行 `let i = 0`，声明了一个 **计数变量 `i`**，初始为 `0`
3. 执行 `const l2 = c2.length`。此时的 `l2` 表示为 **新的子节点的长度**，即： `newChildrenLength`
4. 执行 `let e1 = c1.length - 1`。此时的 `e1` 表示为 **旧的子节点最大（最后一个）下标**，即： `oldChildrenEnd`
5. 执行 `let e2 = l2 - 1`。此时的 `e2` 表示为 **新的子节点最大（最后一个）下标**，**即： `newChildrenEnd`
6. 进入第一个 `while`，此时因为不满足 `isSameVNodeType` 的场景，所以会直接 **跳出 `while`**
7. 进入 **第二个 `while`**，此时各变量的值为：

```
e1: 2
e2: 2
i: 0
isSVG: false
l2: 3
```

1. 执行 `while` 循环： `while (i <= e1 && i <= e2)`

1. **第一次** 进入 `while` 循环：

1. 此时 `n1` 的值为：

<> 代码块

```
1 h('li', {
2   key: 3
3 }, 'c')
```

2. 此时 `n2` 的值为：

<> 代码块

```
1 h('li', {
2   key: 3
3 }, 'd')
```

3. 那么根据上一小节所说，我们知道，此时 `isSameVNodeType(n1, n2)` 会被判定为 `true`
4. 所以此时执行 `patch` 方法，进行打补丁即可。
5. 最后执行：`e1--` 和 `e2--`
6. 至此，第一次循环完成（此时浏览器视图已经更新）

1. 根据刚才所知，此时的 `n1` 和 `n2` 依然符合 `isSameVNodeType(n1, n2)` 的判定
2. 所以，依然会执行 `patch` 方法，进行打补丁。
3. 最后执行：`e1--` 和 `e2--`
4. 至此，第二次循环完成

3. 第三次进入 `while` 循环：

1. 根据刚才所知，此时的 `n1` 和 `n2` **不再符合** `isSameVNodeType(n1, n2)` 的判定
2. 所以，**会直接跳出循环**
3. 此时，各变量的值为：

```
e1: 0
e2: 0
i: 0
isSVG: false
l2: 3
```

2. 三次循环全部完成，此时，我们查看浏览器，可以发现 `children` 的 **更新操作已经完成**。
3. 后续的代码无需关心。

那么由以上可知：

1. `vue` 的 `diff` 首先会 **自前向后** 和 **自后向前**，处理所有的 **相同的 `VNode` 节点**
2. 每次处理成功之后，会自减 `e1` 和 `e2`，表示：**新、旧节点中已经处理完成节点（自后向前）**

04：框架实现：场景一：自前向后的 `diff` 对比 ◀ 上一节 下一节 ▶ 06：框架实现：场景二：自后向前的 `diff` 对比

✎ 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

✎ 意见反馈

♥ 收藏教程

🔖 标记书签