

在上一小节中，我们完成了 `Element` 的更新操作，但是我们之前的更新操作是针对 **相同** 元素的，那么在 **不同** 元素下，`ELEMENT` 的更新操作会产生什么样的变化呢？

创建对应的测试实例 `packages/vue/examples/imooc/runtime/render-element-update-2.html`

<> 代码块

```
1 <script>
2   const { h, render } = Vue
3
4   const vnode = h('div', {
5     class: 'test'
6   }, 'hello render')
7   // 挂载
8   render(vnode, document.querySelector('#app'))
9
10  // 延迟两秒，生成新的 vnode，进行更新操作
11  setTimeout(() => {
12    const vnode2 = h('h1', {
13      class: 'active'
14    }, 'update')
15    render(vnode2, document.querySelector('#app'))
16  }, 2000);
17 </script>
```

然后我们跟踪代码的逻辑，**注意：**这次我们从 `render` 函数开始 `debugger`：

1. 等待第二次进入 `render`
2. `vnode` 存在, 执行 `patch` 方法
 1. 进入 `patch` 方法
 2. 此时所有参数分别为 (**重点关注** `n1` 、 `n2`) :

```
// style in order to prevent being inlined by minifiers.  
const patch: PatchFn = (  
  n1, n2 = { __v_isNode: true, __v_skip: true, type: 'div', props: {...}, key: null  
    n1, n2 = { __v_isNode: true, __v_skip: true, type: 'h1', props: {...}, key: null,  
    container, container = div$app {__vnode: {...}, align: '', title: '', lang: '', tr  
    anchor = null, anchor = null  
    parentComponent = null, parentComponent = null  
    parentSuspense = null, parentSuspense = null  
    isSVG = false, isSVG = false  
    slotScopeIds = null, slotScopeIds = null  
    optimized = __DEV__ && isHmrUpdating ? false : !!n2.dynamicChildren optimized =  
  ) => {
```

- ### 3. 代码执行 `if (n1 && !isSameVNodeType(n1, n2)) :`
1. 此时 `n1` 肯定存在
 2. 那么 `isSameVNodeType` 方法又是干什么的呢？我们来看一下
 1. 进入 `isSameVNodeType` 方法
 2. 可以发现该方法非常简单，只有一句话

<> 代码块

```
1 return n1.type === n2.type && n1.key === n2.key
```

3. 由以上代码可知，`isSameVNodeType` 的作用主要就是判断 `n1` 和 `n2` 是否为：

1. 相同类型的元素。比如都是 `div`
2. 同一个元素。`key` 相同 表示为同一个元素

4. 那么此时则执行 `unmount` 方法，该方法从方法名看是 卸载 的方法

1. 进入 `unmount` 方法
2. 在 `unmount` 方法中，虽然代码很多，但是大多数代码都没有执行
3. 执行到 `remove(vnode)`，表示删除 `vnode`

1. 进入 `remove` 方法
2. 同样大多数代码没有执行
3. 直接到 `performRemove()`

1. 进入 `performRemove`
2. 执行 `hostRemove(el!)`

1. 进入 `hostRemove`，触发的是 `nodeOps` 中的 `remove` 方法

1. 代码为 `parent.removeChild(child)`

5. 至此 `el` 被删除

6. 然后将 `n1 = null`

7. 此时，进入 `switch`，触发 `processElement`

8. 因为 `n1 === null`，所以会触发 `mountElement` 挂载新节点 操作

由以上代码可知：

1. 当节点元素不同时，更新操作执行的其实是：**先删除、后挂载**的逻辑
2. 删除元素的代码从 `unmount` 开始，虽然逻辑很多，但是最终其实是触发了 `nodeOps` 下的 `remove` 方法，通过 `parent.removeChild(child)` 完成的删除操作。

07: 框架实现：渲染更新，ELEMENT 节点... < 上一节 下一节 > 09: 框架实现：处理新旧节点不同元素时，

✎ 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