慕课网首页 免费课 实战课 体系课 **慕课教程** 专栏 手记 企业服务

Q 🏋 💄 我的课程



Q

07: 框架字现: 淮

?

··

索引目录

从所有教程的词条中查询…

首页 > 慕课教程 > Vue3源码分析与构建方案 > 07:框架实现: 渲染更新, ELEMENT 节点的更...

全部开发者教程 :Ξ

04:框架实现:基于 renderer 完成 ELEMENT 节点挂载

05: 框架实现: 合并渲染架构. 得到可用的 render 函数

06:源码阅读:渲染更新, ELEMENT 节点的更新操作

07:框架实现:渲染更新, ELEMENT 节点的更新实现

08: 源码阅读: 新旧节点不同 元素时, ELEMENT 节点的更 新操作

09: 框架实现: 处理新旧节点 不同元素时,

10:框架实现:删除元素, ELEMENT 节点的卸载操作

11: 源码阅读: class 属性和 其他属性的区分挂载

12:深入属性挂载: HTML Attributes 和 DOM

07: 框架实现: 渲染更新, ELEMENT 节点的更新实现

根据以上逻辑, 我们可以直接为 processElement 方法, 新增对应的 else 逻辑:

1. 在 packages/runtime-core/src/renderer.ts 中, 为 processElement 增加新的判断:

```
/**

/**

Element 的打补丁操作

if (oldVNode == null) {

// 挂载操作

mountElement(newVNode, container, anchor)

else {

// 更新操作

patchElement(oldVNode, newVNode)

}
```

2. 创建 patchElement 方法:

```
<> 代码块
 1
 2
      * element 的更新操作
 3
 4
     const patchElement = (oldVNode, newVNode) => {
      // 获取指定的 el
       const el = (newVNode.el = oldVNode.el!)
      // 新旧 props
 8
      const oldProps = oldVNode.props || EMPTY_OBJ
 9
10
      const newProps = newVNode.props || EMPTY OBJ
11
      // 更新子节点
13
      patchChildren(oldVNode, newVNode, el, null)
14
      // 更新 props
15
      patchProps(el, newVNode, oldProps, newProps)
16
17
```

3. 创建 patchChildren 方法:

```
/>代码块

/**
/**
/**
/*
/*
/*
/*
/*
/*
// BP节点打补丁
// 旧节点的 children = (oldVNode, newVNode, container, anchor) => {
// 旧节点的 children
// 旧节点的 children
// 旧节点的 prevShapeFlag
const prevShapeFlag = oldVNode ? oldVNode.shapeFlag : 0
// 新节点的 children
// 新节点的 children
// 新节点的 children
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
// **
```







```
12
      // 新节点的 shapeFlag
13
       const { shapeFlag } = newVNode
14
15
       // 新子节点为 TEXT_CHILDREN
16
      if (shapeFlag & ShapeFlags.TEXT_CHILDREN) {
        // 旧子节点为 ARRAY_CHILDREN
17
        if (prevShapeFlag & ShapeFlags.ARRAY_CHILDREN) {
18
          // TODO: 卸载旧子节点
19
20
21
        // 新旧子节点不同
22
       if (c2 !== c1) {
         // 挂载新子节点的文本
23
2.4
          hostSetElementText(container, c2 as string)
25
      } else {
27
        // 旧子节点为 ARRAY_CHILDREN
28
        if (prevShapeFlag & ShapeFlags.ARRAY_CHILDREN) {
29
          // 新子节点也为 ARRAY_CHILDREN
         if (shapeFlag & ShapeFlags.ARRAY_CHILDREN) {
30
31
           // TODO: 这里要进行 diff 运算
32
33
          // 新子节点不为 ARRAY_CHILDREN,则直接卸载旧子节点
34
35
           // TODO: 卸载
36
          }
37
        } else {
38
          // 旧子节点为 TEXT_CHILDREN
39
          if (prevShapeFlag & ShapeFlags.TEXT_CHILDREN) {
40
            // 删除旧的文本
41
            hostSetElementText(container, '')
42
          // 新子节点为 ARRAY_CHILDREN
43
          if (shapeFlag & ShapeFlags.ARRAY_CHILDREN) {
44
            // TODO: 单独挂载新子节点操作
45
46
47
        }
48
      }
49
     }
```

4. 创建 patchProps 方法:

```
<> 代码块
     * 为 props 打补丁
 4
    const patchProps = (el: Element, vnode, oldProps, newProps) => {
5
     // 新旧 props 不相同时才进行处理
     if (oldProps !== newProps) {
 6
        // 遍历新的 props,依次触发 hostPatchProp ,赋值新属性
       for (const key in newProps) {
8
          const next = newProps[key]
10
          const prev = oldProps[key]
11
         if (next !== prev) {
12
           hostPatchProp(el, key, prev, next)
13
          }
14
        // 存在旧的 props 时
15
       if (oldProps !== EMPTY_OBJ) {
17
         // 遍历旧的 props,依次触发 hostPatchProp ,删除不存在于新props 中的旧属性
18
         for (const key in oldProps) {
19
           if (!(key in newProps)) {
20
              hostPatchProp(el, key, oldProps[key], null)
21
22
23
24
25
```

至此, 更新操作完成。



?





```
<>代码块
    <script>
      const { h, render } = Vue
 3
     const vnode = h('div', {
       class: 'test'
     }, 'hello render')
  8
 9
     render(vnode, document.querySelector('#app'))
 10
     // 延迟两秒,生成新的 vnode,进行更新操作
 11
      setTimeout(() => {
 12
      const vnode2 = h('div', {
 13
         class: 'active'
 14
       }, 'update')
 15
       render(vnode2, document.querySelector('#app'))
 16
 17 }, 2000);
 18 </script>
测试更新成功。
   06: 源码阅读: 渲染更新, ELEMENT 节点... ◆ 上一节 下一节 ▶ 08: 源码阅读: 新旧节点不同元素时, ELEM...
                                ✔ 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

♦ 6 3

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



