Sunday • 更新于 2022-10-19

上一节 06：源码阅读：… 08：源码阅读：… 下一节

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：

代码块

```
/**
 * Element 的打补丁操作
 */
const processElement = (oldVNode, newVNode, container, anchor) => {
  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

```
/**
 * element 的更新操作
 */
const patchElement = (oldVNode, newVNode) => {
  // 获取指定的 el
  const el = (newVNode.el = oldVNode.el!)

  // 新旧 props
  const oldProps = oldVNode.props || EMPTY_OBJ
  const newProps = newVNode.props || EMPTY_OBJ

  // 更新子节点
  patchChildren(oldVNode, newVNode, el, null)

  // 更新 props
  patchProps(el, newVNode, oldProps, newProps)
}
```

3. 创建 `patchChildren` 方法：

代码块

```
/**
 * 为子节点打补丁
 */
const patchChildren = (oldVNode, newVNode, container, anchor) => {
  // 旧节点的 children
  const c1 = oldVNode && oldVNode.children
  // 旧节点的 prevShapeFlag
  const prevShapeFlag = oldVNode ? oldVNode.shapeFlag : 0
  // 新节点的 children
```

```
    // 新节点的 shapeFlag
    const { shapeFlag } = newVNode

    // 新子节点为 TEXT_CHILDREN
    if (shapeFlag & ShapeFlags.TEXT_CHILDREN) {
      // 旧子节点为 ARRAY_CHILDREN
      if (prevShapeFlag & ShapeFlags.ARRAY_CHILDREN) {
        // TODO: 卸载旧子节点
      }
      // 新旧子节点不同
      if (c2 !== c1) {
        // 挂载新子节点的文本
        hostSetElementText(container, c2 as string)
      }
    } else {
      // 旧子节点为 ARRAY_CHILDREN
      if (prevShapeFlag & ShapeFlags.ARRAY_CHILDREN) {
        // 新子节点也为 ARRAY_CHILDREN
        if (shapeFlag & ShapeFlags.ARRAY_CHILDREN) {
          // TODO: 这里要进行 diff 运算
        }
        // 新子节点不为 ARRAY_CHILDREN，则直接卸载旧子节点
        else {
          // TODO: 卸载
        }
      } else {
        // 旧子节点为 TEXT_CHILDREN
        if (prevShapeFlag & ShapeFlags.TEXT_CHILDREN) {
          // 删除旧的文本
          hostSetElementText(container, '')
        }
        // 新子节点为 ARRAY_CHILDREN
        if (shapeFlag & ShapeFlags.ARRAY_CHILDREN) {
          // TODO: 单独挂载新子节点操作
        }
      }
    }
  }
```

4. 创建 `patchProps` 方法：

代码块

```
/**
 * 为 props 打补丁
 */
const patchProps = (el: Element, vnode, oldProps, newProps) => {
  // 新旧 props 不相同时才进行处理
  if (oldProps !== newProps) {
    // 遍历新的 props，依次触发 hostPatchProp ，赋值新属性
    for (const key in newProps) {
      const next = newProps[key]
      const prev = oldProps[key]
      if (next !== prev) {
        hostPatchProp(el, key, prev, next)
      }
    }
    // 存在旧的 props 时
    if (oldProps !== EMPTY_OBJ) {
      // 遍历旧的 props，依次触发 hostPatchProp ，删除不存在于新props 中的旧属性
      for (const key in oldProps) {
        if (!(key in newProps)) {
          hostPatchProp(el, key, oldProps[key], null)
        }
      }
    }
  }
}
```

至此，更新操作完成。

代码块

```
<script>
  const { h, render } = Vue

  const vnode = h('div', {
    class: 'test'
  }, 'hello render')


  render(vnode, document.querySelector('#app'))

  // 延迟两秒，生成新的 vnode，进行更新操作
  setTimeout(() => {
    const vnode2 = h('div', {
      class: 'active'
    }, 'update')
    render(vnode2, document.querySelector('#app'))
  }, 2000);
</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

意见反馈 收藏教程 标记书签