

全部开发者教程

17: 深入事件更新: vue event invokers

18: 框架实现: ELEMENT 节点下, 事件的挂载和更新

19: 局部总结: ELEMENT 节点的挂载、更新、props 打补丁等行为总结

20: 源码阅读: renderer 渲染器下, Text 节点的挂载、更新行为

21: 框架实现: renderer 渲染器下, Text 节点的挂载、更新行为

22: 源码阅读: renderer 渲染器下, Comment 节点的挂载行为

23: 框架实现: renderer 渲染器下, Comment 节点的挂载行为

24: 源码阅读: renderer渲染器下, Fragment 节点的挂



Sunday • 更新于 2022-10-19

← 上一节 19: 局部总结: ... 21: 框架实现: ... 下一节 →

## 20: 源码阅读: renderer 渲染器下, Text 节点的挂载、更新行为

这一小节, 我们来看 Text 文本 节点的挂载、更新、卸载 行为。

首先创建测试实例 packages/vue/examples/imooc/runtime/render-text.html :

<> 代码块

```
1 <script>
2   const { h, render, Text } = Vue
3
4   const vnode = h(Text, 'hello world')
5   // 挂载
6   render(vnode, document.querySelector('#app'))
7
8   // 延迟两秒, 生成新的 vnode, 进行更新操作
9   setTimeout(() => {
10     const vnode2 = h(Text, '你好, 世界')
11     render(vnode2, document.querySelector('#app'))
12   }, 2000);
13 </script>
```

我们知道, 对于节点的打补丁操作是从 packages/runtime-core/src/renderer.ts 中的 render 函数开始的, 所以我们可以直接在这里进行 debugger :

1. 第一次进入 render , 执行挂载操作:

1. 进入 patch 方法, 此时的参数为:

```
const patch: PatchFn = (
  n1, n1 = null
  n2, n2 = { __v_isVNode: true, __v_skip: true, type: Symbol(),
  container, container = div#app {align: '', title: '',
  anchor = null, anchor = null
  parentComponent = null, parentComponent = null
  parentSuspense = null, parentSuspense = null
  isSVG = false, isSVG = false
  slotScopeIds = null, slotScopeIds = null
```

2. 执行 switch 判定, 进入 processText 方法:

1. 进入 processText , 此时的各参数为:

```
anchor: null
▶ container: div#app
n1: null
▶ n2: { __v_isVNode: true, __v_skip: true, type: Symbol(T
```

2. 因为 n1 === null , 所以执行 hostInsert 和 hostCreateText 方法, 即: 挂载 操作

1. 首先进入 hostCreateText 方法:

1. 实际触发的是 packages/runtime-dom/src/nodeOps.ts 下的 createText 方法:

索引目录

20: 源码阅读: re



意见反馈

收藏教程

标记书签

<> 代码块

```
1 createText: text => doc.createTextNode(text),
```

该方法比较简单，直接通过 `doc.createTextNode(text)` 生成 `Text` 节点

2. 其次进入 `hostInsert` 方法：

1. 该方法实际触发的是 `packages/runtime-dom/src/nodeOps.ts` 下的 `insert` 方法：

<> 代码块

```
1 parent.insertBefore(child, anchor || null)
```

该方法我们之前实现过，就不在多说了。

2. 至此 **挂载** 操作完成

3. 延迟两秒，第二次进入 `render` 方法，执行 **更新操作**

1. 进入 `patch` 方法，此时的参数为：

```
const patch: PatchFn = (
  n1, n1 = {__v_isVNode: true, __v_skip: true, type: Symbol(),
  n2, n2 = {__v_isVNode: true, __v_skip: true, type: Symbol(),
  container, container = div#app {__vnode: {...}, align: '',
  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.dynamic
```

2. 执行 `switch`，触发 `processText` 方法

1. 进入 `processText` 方法，此时参数为：

```
this: undefined
anchor: null
▶ container: div#app
▶ n1: {__v_isVNode: true, __v_skip: true, type: Symbol(T
▶ n2: {__v_isVNode: true, __v_skip: true, type: Symbol(T
```

2. 此时 `n1 !== null`，所以进入 `else` 逻辑

1. 执行 `const e1 = (n2.e1 = n1.e1!)` 获取同样的 `e1`

2. 执行 `hostSetText(e1, n2.children as string)`

1. 进入 `hostSetText` 方法，其实触发的是 `packages/runtime-dom/src/nodeOps.ts` 下的 `setText` 方法：

<> 代码块

```
1 node.nodeValue = text
```

2. 该方法非常简单，只是修改 `value` 而已

4. 至此，更新操作完成。

由以上代码可知：

1. 对于 `Text` 节点的 **挂载**和**更新**，整体是非常简单的：

1. 挂载：通过 `doc.createTextNode(text)` 生成节点，在通过 `insertBefore` 插入

2. 更新：通过 `node.nodeValue` 直接指定即可。

[意见反馈](#)

[收藏教程](#)

[标记书签](#)

 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