

全部开发者教程

第八章: runtime 运行时 - 运行时核心设计原则

01: 前言

02: HTML DOM 节点树与虚拟 DOM 树

03: 挂载与更新

04: h 函数 与 render 函数

05: 运行时核心设计原则

06: 总结

第九章: runtime 运行时 - 构建 h 函数, 生成 Vnode

01: 前言

02: 阅读源码: 初见 h 函数, 跟踪 Vue 3 源码实现基础逻辑

03: 框架实现: 构建 h 函数, 处理 ELEMENT + TEXT_CHILDREN 场景

04: 源码阅读: h 函数, 跟踪 ELEMENT +



Sunday • 更新于 2022-10-19

上一节 02: HTML DO... 04: h 函数 与 r... 下一节

03: 挂载与更新

这一小节，我们将通过一个极简的案例，来了解两个比较重要的概念：

- 1. 挂载：mount
- 2. 更新：patch

挂载：mount

首先我们先来构建这个案例（该案例在 第二章第七小节 《运行时》进行过大致讲解）：

```
<> 代码块
1  <script>
2    const VNode = {
3      type: 'div',
4      children: 'hello render'
5    }
6
7    // 创建 render 渲染函数
8    function render(oldVNode, newVNode, container) {
9      if (!oldVNode) {
10        mount(newVNode, container)
11      }
12    }
13
14    // 挂载函数
15    function mount(vnode, container) {
16      // 根据 type 生成 element
17      const ele = document.createElement(vnode.type)
18      // 把 children 赋值给 ele 的 innerText
19      ele.innerText = vnode.children
20      // 把 ele 作为子节点插入 body 中
21      container.appendChild(ele)
22    }
23
24    render(null, VNode, document.querySelector('#app'))
25  </script>
```

在当前案例中，我们首先创建了一个 render 渲染函数，该函数接收三个参数：

- 1. oldVNode：旧的 VNode
- 2. newVNode：新的 VNode
- 3. container：容器

当 oldVNode 不存在时，那么我们就认为这是一个全新的渲染，也就是 挂载。

所以以上的 mount 方法，我们就可以把它称为是一个 挂载方法。

更新：patch

旧的视图不可能被一直展示，它会在未来某一个时刻被更新为全新的视图。

比如：

索引目录

- 03: 挂载与更新
 - 挂载：mount
 - 更新：patch
 - 总结



<> 代码块

```
1  <script>
2    const VNode = {
3      type: 'div',
4      children: 'hello render'
5    }
6
7    const VNode2 = {
8      type: 'div',
9      children: 'patch render'
10   }
11
12   // 创建 render 渲染函数
13   function render(oldVNode, newVNode, container) {
14     if (!oldVNode) {
15       mount(newVNode, container)
16     } else {
17       patch(oldVNode, newVNode, container)
18     }
19   }
20
21   // 挂载函数
22   function mount(vnode, container) {
23     // 根据 type 生成 element
24     const ele = document.createElement(vnode.type)
25     // 把 children 赋值给 ele 的 innerText
26     ele.innerText = vnode.children
27     // 把 ele 作为子节点插入 body 中
28     container.appendChild(ele)
29   }
30
31   // 取消挂载
32   function unmount(container) {
33     container.innerHTML = ''
34   }
35
36   // 更新函数
37   function patch(oldVNode, newVNode, container) {
38     unmount(container)
39
40     // 根据 type 生成 element
41     const ele = document.createElement(newVNode.type)
42     // 把 children 赋值给 ele 的 innerText
43     ele.innerText = newVNode.children
44     // 把 ele 作为子节点插入 body 中
45     container.appendChild(ele)
46   }
47
48   render(null, VNode, document.querySelector('#app'))
49
50   setTimeout(() => {
51     render(VNode, VNode2, document.querySelector('#app'))
52   }, 2000);
53 </script>
```

我们在原有的代码中去新增了一部分逻辑，新增了 `patch` 函数。

在 `patch` 函数中，我们先 **删除了旧的 `VNode`**，然后**创建了一个新的 `VNode`**。这样的流程，我们就把它叫做 **挂载** `patch`

总结

本小节我们通过一个简单的例子讲解了 **挂载** `mount` 和 **更新** `patch` 的概念，这两个概念 [Vue 3 官方文档](#) 也对此进行了详细的介绍：

1. **挂载**：运行时渲染器调用渲染函数，遍历返回的虚拟 DOM 树，并基于它创建实际的 DOM 节点。
2. **更新**：当一个依赖发生变化后，副作用会重新运行，这时候会创建一个更新后的虚拟 DOM 树。运行时渲染器遍历这棵新树，将它与旧树进行比较，然后将必要的更新应用到真实 DOM 上去。

[意见反馈](#)

[收藏教程](#)

[标记书签](#)



这两个概念在我们后面去实现 `renderer` 渲染器的时候还会经常的使用到。

02: HTML DOM 节点树与虚拟 DOM 树 ◀ 上一节

下一节 ▶ 04: h 函数 与 render 函数

 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