全部开发者教程

则

06：为什么说框架的设计过程其实是一个不断取舍的过程?

07：.vue 中的 html 是真实的 html 吗?

08：什么是运行时?

09：什么是编译时?

10：运行时 + 编译时

11：什么是副作用

12：Vue 3 框架设计概述

13：扩展：所谓良好的 `TypeScript` 支持，是如何提供的?

14：总结

第三章：Vue 3源码结构 - 搭建框架雏形

01：前言

02：探索源码设计：Vue3 源

Sunday • 更新于 2022-10-19

上一节 09：什么是编译… 11：什么是副作用 下一节

10：运行时 + 编译时

前面两小节我们已经分别了解了 **运行时 和 编译时**，同时我们也知道了：**vue 是一个 运行时+编译时** 的框架！

> vue 通过 compiler 解析 html 模板，生成 render 函数，然后通过 runtime 解析 render ，从而挂载真实 dom 。

那么看到这里可能有些同学就会有疑惑了，既然 **compiler 可以直接解析 html 模板**，那么为什么还要生成 render 函数，然后再去进行渲染呢？为什么不直接利用 compiler 进行渲染呢？

即：**为什么 vue 要设计成一个 运行时+编译时的框架呢?**

那么想要理清楚这个问题，我们就需要知道 **dom 渲染是如何进行的。**

对于 dom 渲染而言，可以被分为两部分：

1. **初次渲染** ，我们可以把它叫做 **挂载**
2. **更新渲染** ，我们可以把它叫做 **打补丁**

初次渲染

那么什么是初次渲染呢?

当初始 div 的 innerHTML 为空时，

代码块

```
<div id="app"></div>
```

我们在该 div 中渲染如下节点：

代码块

```
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
</ul>
```

那么这样的一次渲染，就是 **初始渲染**。在这样的一次渲染中，我们会生成一个 ul 标签，同时生成三个 li 标签，并且把他们挂载到 div 中。

更新渲染

那么此时如果 ul 标签的内容发生了变化：

代码块

```
<ul>
    <li>3</li>
    <li>1</li>
    <li>2</li>
</ul>
```

索引目录

10：运行时 + 编译

初次渲染

更新渲染

`li - 3` 上升到了第一位，那么此时大家可以想一下：**我们期望浏览器如何来更新这次渲染呢?**

浏览器更新这次渲染无非有两种方式：

1. 删除原有的所有节点，重新渲染新的节点
2. 删除原位置的 `li - 3`，在新位置插入 `li - 3`

那么大家觉得这两种方式哪一种方式更好呢？那么我们来分析一下：

1. 首先对于第一种方式而言：它的好处在于不需要进行任何的比对，需要执行 6 次（删除 3 次，重新渲染 3 次） `dom` 处理即可。
2. 对于第二种方式而言：在逻辑上相对比较复杂。他需要分成两步来做：
    1. 对比 **旧节点** 和 **新节点** 之间的差异
    2. 根据差异，删除一个 **旧节点**，增加一个 **新节点**

那么根据以上分析，我们知道了：

1. 第一种方式：会涉及到更多的 `dom` 操作
2. 第二种方式：会涉及到 `js` 计算 + 少量的 `dom` 操作

那么这两种方式，哪一种更快呢？我们来实验一下：

代码块

```
const length = 10000
// 增加一万个dom节点，耗时 3.992919921875 ms
console.time('element')
for (let i = 0; i < length; i++) {
  const newEle = document.createElement('div')
  document.body.appendChild(newEle)
}
console.timeEnd('element')

// 增加一万个 js 对象，耗时 0.402099609375 ms
console.time('js')
const divList = []
for (let i = 0; i < length; i++) {
  const newEle = {
    type: 'div'
  }
  divList.push(newEle)
}
console.timeEnd('js')
```

从结果可以看出， `dom` 的操作要比 `js` 的操作耗时多得多，即： `dom` **操作比** `js` **更加耗费性能。**

那么根据这样的一个结论，回到我们刚才所说的场景中：

> 1. 首先对于第一种方式而言：它的好处在于不需要进行任何的比对，仅需要执行 6 次（删除 3 次，重新渲染 3 次） `dom` 处理即可。
> 2. 对于第二种方式而言：在逻辑上相对比较复杂。他需要分成两步来做：
>   1. 对比 **旧节点** 和 **新节点** 之间的差异
>   2. 根据差异，删除一个 **旧节点**，增加一个 **新节点**

根据结论可知：**方式一会比方式二更加消耗性能（即：性能更差）。**

那么得出这样的结论之后，我们回过头去再来看最初的问题：**为什么 vue 要设计成一个 运行时+编译时的框架呢?**

答：

1. 针对于 **纯运行时** 而言：因为不存在编译器，所以我们只能够提供一个复杂的 `JS` 对象。
2. 针对于 **纯编译时** 而言：因为缺少运行时，所以它只能把分析差异的操作，放到 **编译时** 进行，同样因为省略了运行时，所以速度可能会更快。但是这种方式这将损失灵活性（具体可查看第六章虚拟 `DOM` ，或可点击 这里 查看官方示例）。比如 svelte ，它就是一个纯编译时的框架，但是它的实际运行速度可能达不到理论上的速度。
3. **运行时 + 编译时**：比如 `vue` 或 `react` 都是通过这种方式来进行构建的，使其可以在保持灵活性的基础上，尽量的进行性能的优化，从而达到一种平衡。

09：什么是编译时？ 上一节 下一节 11：什么是副作用

我要提出意见反馈

意见反馈 收藏教程 标记书签