Sunday • 更新于 2022-10-19

上一节 13：框架实现：... 15：框架实现：... 下一节

14：源码阅读：ELEMENT 节点下， style 属性的挂载和更新

根据我们在 `packages/runtime-dom/src/patchProp.ts` 下 `patchProp` 中的方法可知，目前我们还缺少 `style` 和 `事件` 的挂载。

那么下面我们就来看下 `style` 属性的挂载的更新操作。

我们可以创建如下测试实例 `packages/vue/examples/imooc/runtime/render-element-style.html`：

代码块

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

  const vnode = h('div', {
    style: {
      color: 'red'
    },
  }, '你好，世界')
  // 挂载
  render(vnode, document.querySelector('#app'))

  setTimeout(() => {
    const vnode2 = h('div', {
      style: {
        fontSize: '32px'
      },
    }, '你好，世界')
    // 挂载
    render(vnode2, document.querySelector('#app'))
  }, 2000);
</script>
```

在 `patchProp` 方法中，跟踪源码实现：

1. 第一次进入，执行 **挂载** 操作

    1. 进入 `patchProp` 方法，此时各参数的值为：

    2. 代码进入 `patchStyle` 方法，开始进行打补丁操作
    3. 执行 `const style = (el as HTMLElement).style` ，此时 `style` 为当前 `element` 的 `style`
    4. 执行 `const isCssString = isString(next)` 方法：
    5. 此时 `next` 的值为：`{color: 'red'}`，不是一个 `string`
    6. 所以 `isCssString` 的值为 `false`
    7. 执行 `if (next && !isCssString)`，进入 `if`

1. 以 ``{color: 'red'} 为例，此时的 key 为 color`
2. 执行 `setStyle(style, key, next[key])` 方法：
3. 进入 `setStyle` 方法，此时各参数的值为：

```
function setStyle(
  style: CSSStyleDeclaration,  style = CSSStyleDeclaratior
  name: string,  name = "color"
  val: string | string[]  val = "red"
) {
```

4. `val` 不是 `Array` ，直接进入 `else`
5. 执行 `const prefixed = autoPrefix(style, name)`：

    1. 进入 `autoPrefix` 方法
    2. 执行 `let name = camelize(rawName)` 方法:

        1. `camelize` 方法的内容比较简单，就是把 `key` 变为 **驼峰格式** 字符串
        2. 因为 `rawName` 此时为 `color` ，所以 得到的 `name` 依然为 `color`

    3. 最后执行 `return (prefixCache[rawName] = name)` 把当前 `name` 进行 **缓存** ，并返回 `name` 的值

6. 得到 `prefixed = color`
7. 最后执行 `style[prefixed as any] = val` ，直接为 `style` 对象进行赋值操作

8. 至此 `style` 属性 **挂载完成**
9. 但是光指定完成还不够，我们还需要 **卸载旧的** `style` ，以完成 **更新**

2. **延迟两秒之后...**
3. 第二次进入，执行 **更新** 操作

    1. 忽略掉相同的挂载逻辑
    2. 代码执行到 `patchStyle` 方法下， `if (prev && !isString(prev)) {......}` 判断

        1. 此时存在两个变量：

            1. `prev`： 上一次的样式 `{color: 'red'}`
            2. `next`： 这一次的样式 `{fontSize: '32px'}`

        3. 满足 `if` 判断条件，进入 `if`

            1. 执行 `for (const key in prev)` ，**遍历旧样式**
            2. 执行 `if (next[key] == null)` ，**旧样式不存在于样式中**
            3. 则执行 `setStyle(style, key, '')` 方法

                1. 再次进入 `setStyle` 方法，此时的参数为：

```
function setStyle(
  style: CSSStyleDeclaration,  style = CSSStyleDe
  name: string,  name = "color"
  val: string | string[]  val = ""
) {
```

                2. **注意：** 此时 `val` 为 `''`
                3. 再次执行 `style[prefixed as any] = val` ，即： `style[color] = ''`
                4. 完成 **清理旧样式** 操作

        4. 至此 **更新** 操作完成

由以上代码可知：

2. 在 **不考虑边缘情况** 的前提下， `vue` 只是对 `style` 进行了 **缓存** 和 **赋值** 两个操作

3. 缓存是通过 `prefixCache = {}` 进行

4. 赋值则是直接通过 `style[xxx] = val` 进行

13：框架实现：区分处理 ELEMENT 节点的... 上一节 下一节 15：框架实现：ELEMENT 节点下，style 属...

我要提出意见反馈