Sunday • 更新于 2022-10-19

上一节 10：框架实现：... 12：深入属性挂... 下一节

11：源码阅读：class 属性和其他属性的区分挂载

查看我们的源代码 `packages/runtime-dom/src/patchProp.ts`，可以发现，目前针对我们的代码而言，只能挂载 **class** 属性，而不能挂载其他属性。

那么针对于 **其他属性** 是如何进行挂载的呢？我们通过一个测试实例来看一下，创建 `packages/vue/examples/imooc/runtime/render-element-props.html` 测试实例：

代码块

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

  const vnode = h('textarea', {
    class: 'test-class',
    value: 'textarea value',
    type: 'text'
  })
  // 挂载
  render(vnode, document.querySelector('#app'))
</script>
```

在这个测试实例中，我们为 `textarea` 挂载了三个属性 `class`、`value` 和 `type`，根据之前的源码阅读我们知道，属性的挂载是在 `packages/runtime-dom/src/patchProp.ts` 中的 `patchProp` 方法处进行的。

所以我们可以直接在这里进行 `debugger`，因为我们设置了三个属性，所以会 **三次**，我们一个一个来看：

1. 第一次进入，此时的 `key = class`：
    1. 执行：`if (key === 'class')`：
        1. 触发 `patchClass` 方法
        2. 通过 `el.className = value` 设置 `class`
    2. 这一块我们是比较熟悉的
2. 至此 `class` 设置完成，通过 `el.className`
3. 第二次进入，此时 `key = type`：
    1. 执行 `else if`，将触发 `shouldSetAsProp(el, key, nextValue, isSVG)` 方法
        1. 进入 `shouldSetAsProp` 方法
        2. 执行 `if (key === 'type' && el.tagName === 'TEXTAREA')`，当前 **满足条件**，则直接 **跳出**
    2. 执行 `else`：
        1. 触发 `patchAttr(el, key, nextValue, isSVG, parentComponent)` 方法
            1. 进入 `patchAttr` 方法
            2. 最终执行 `el.setAttribute(key, isBoolean ? '' : value)` 设置 `type`

4. 至此 `type` 设置完成，通过 `el.setAttribute`
5. 第三次进入，此时 `key = value`

    1. 执行 `else if`，将触发 `shouldSetAsProp(el, key, nextValue, isSVG)` 方法

        1. 进入 `shouldSetAsProp` 方法
        2. 执行 `return key in el` 表达式，因为 `el = textarea`， `key = value`
        3. 所以 `value in textarea DOM` ，返回为 `true`

    2. 执行 `patchDOMProp` 方法：

        1. 进入 `patchDOMProp` 方法
        2. 执行 `el[key] = value` 设置 `value`

6. 至此 `value` 设置完成，通过 `el[key] = value`

至此三个属性全部设置完成。

由以上代码可知：

1. 针对于三个属性， `vue` 通过了 **三种不同的方式** 来进行了设置：

    1. `class` 属性：通过 `el.className` 设定
    2. `textarea 的 type` 属性：通过 `el.setAttribute` 设定
    3. `textarea 的 value` 属性：通过 `el[key] = value` 设定

那么很多同学看到这里，就会非常疑惑了，为什么 **要通过三种不同的形式挂载属性呢?**

此时： 我们的测试案例已经成功运行到浏览器中了，让我们打开浏览器的 **控制台**，来测试如下代码：

代码块

```
// 初始状态：<textarea class="test-class" type="text"></textarea>

// 获取 dom 实例
const el = document.querySelector('textarea')

// 1：修改 class
el.setAttribute('class', 'm-class') // 成功
el['class'] = 'm-class' // 失败
el.className = 'm-class' // 成功

// 2：修改 type
el.setAttribute('type', 'input') // 成功
el['type'] = 'input' // 失败

// 3：修改 value
el.setAttribute('value', '你好 世界') // 失败
el['value'] = '你好 世界' // 成功

```

由以上代码可知，我们在 **针对不同属性，使用不同的 `API` 时**，得到的结果是 **不同** 的。

那么为什么会这样呢?

想要知道这个原因，那么我们就需要来看下一小节： `HTML Attributes 和 DOM Properties` 。

10：框架实现：删除元素，ELEMENT 节点... 上一节 下一节 12：深入属性挂载：HTML Attributes 和 D...

我要提出意见反馈