Sunday • 更新于 2022-10-19

上一节 04：框架实现：... 06：源码阅读：... 下一节

05：框架实现：合并渲染架构，得到可用的 render 函数

现在三大模块中，该完成的业务代码我们都已经完成了。但是还存在一个问题，那就是我们应该 **如何使用 `render` 函数呢?**

我们知道，在源码中，我们可以直接：

代码块

```
const { render } = Vue

render(vnode, document.querySelector('#app'))
```

但是我们来看咱们现在的代码，发现是 **不可以** 直接这样导出并使用的。

那么 `vue` 是怎么做的呢?

源代码

查看源代码，我们可以发现在 `packages/runtime-dom/src/index.ts` 中，存在这样的一段代码：

代码块

```
export const render = ((...args) => {
  ensureRenderer().render(...args)
}) as RootRenderFunction<Element | ShadowRoot>
```

而这里导出的 `render` 就是我们 **解构出来的** `render` 。

在这段代码中涉及到一个方法 `ensureRenderer` ，我们查看它的实现，可以发现：

代码块

```
function ensureRenderer() {
  return (
    renderer ||
    (renderer = createRenderer<Node, Element | ShadowRoot>(rendererOptions))
  )
}
```

它触发了我们导出的 `createRenderer` ，并且传递了一个 `rendererOptions` 参数，查看 `rendererOptions` ：

代码块

```
const rendererOptions = /*#__PURE__*/ extend({ patchProp }, nodeOps)
```

可以发现 `rendererOptions` 就是一个合并了 `prop` 和 `nodeOps` 的对象。

这样：我们就成功的把 `options` 参数传递到了 `createRenderer` 函数中，得到了 `renderer` 渲染器，从而最终拿到 `render` 函数了。

那么明确好了源代码的实现之后，下面我们就尝试自己实现一下。

实现

代码块

```
import { createRenderer } from '@vue/runtime-core'
import { extend } from '@vue/shared'
import { nodeOps } from './nodeOps'
import { patchProp } from './patchProp'

const rendererOptions = extend({ patchProp }, nodeOps)

let renderer

function ensureRenderer() {
    return renderer || (renderer = createRenderer(rendererOptions))
}

export const render = (...args) => {
    ensureRenderer().render(...args)
}
```

2. 在 `packages/runtime-core/src/index.ts` 中导出 `createRenderer`

3. 在 `packages/vue/src/index.ts` 中导出 `render`

4. 在测试实例 `packages/vue/examples/runtime/render-element.html` ：

代码块

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

  const vnode = h('div', {
    class: 'test'
  }, 'hello render')

  console.log(vnode);

  render(vnode, document.querySelector('#app'))
</script>
```

`vnode` 可以正常渲染。

04：框架实现：基于 renderer 完成 ELEME... 上一节 下一节 06：源码阅读：渲染更新，ELEMENT 节点...

我要提出意见反馈