双11返场进行中！优惠Go on，快来捡漏！ 02 天 14 时 38 分 28 秒 Go >>>

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：composition API，setup 函数挂载逻辑

那么到现在我们已经处理好了组件非常多的概念，但是我们还知道对于 vue 3 而言，提供了 composition API，即 setup 函数的概念。

那么如果我们想要通过 setup 函数来进行一个响应性数据的挂载，那么又应该怎么做呢?

我们来看下面的这个测试案例 packages/vue/examples/imooc/runtime/redner-component-setup.html：

代码块

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

  const component = {
    setup() {
      const obj = reactive({
        name: '张三'
      })

      return () => h('div', obj.name)
    }
  }

  const vnode = h(component)
  // 挂载
  render(vnode, document.querySelector('#app'))
</script>
```

在上面的代码中，我们构建了一个 setup 函数，并且在 setup 函数中 return 了一个函数，函数中返回了一个 vnode。

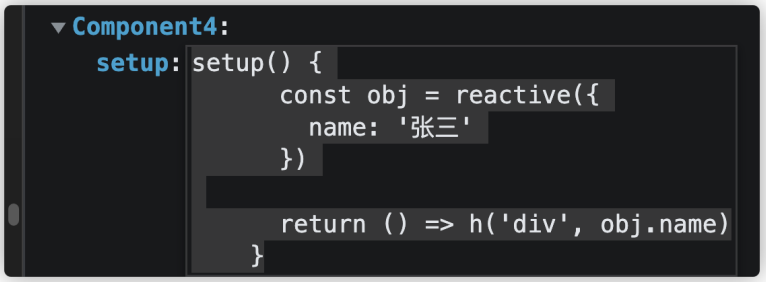
上面的代码运行之后，浏览器会在一个 div 中渲染 张三。

那么对于以上的代码， vue 内部又是如何进行处理的呢?

我们知道， vue 对于组件的挂载，本质上是触发 mountComponent，在 mountComponent 中调用了 setupComponent 函数，通过此函数来对组件的选项进行标准化。

那么 setup 函数本质上就是一个 vue 组件的选项，所以对于 setup 函数处理的核心逻辑，就在 setupComponent 中。我们在这个函数内部进行 debugger。

1. 进入 setupComponent
2. 关注 setupStatefulComponent 函数的触发
    1. 进入 setupStatefulComponent 函数
    2. 代码执行 const Component = instance.type as ComponentOptions，此时得到的 Component 的值为：



3. 代码执行 `const { setup } = Component`，由上面 `component` 的值可知 `setup` 是存在的，所以会进入 `if`

    1. 代码执行：

    ```
    代码块
    1     const setupResult = callWithErrorHandling(
    2           setup,
    3           instance,
    4           ErrorCodes.SETUP_FUNCTION,
    5           [__DEV__ ? shallowReadonly(instance.props) : instance.props, setupConte
    6         )
    ```

    2. 我们知道 `callWithErrorHandling` 本质上是一个 `try...catch`，所以以上代码“约等于” `setup()` 的执行。
    3. 由此得到 `setupResult` 的值为 `() => h('div', obj.name)`。即：**setup 函数的返回值**
    4. 代码继续执行，触发 `handleSetupResult(instance, setupResult, isSSR)`

        1. 进入 `handleSetupResult`，此时各参数为：

        ```
        export function handleSetupResult(
          instance: ComponentInternalInstance,  instance = {uid: 0, vnode: {…}, type:
          setupResult: unknown,  setupResult = () => h('div', obj.name)
          isSSR: boolean  isSSR = false
        ) {
        ```

        2. 执行 `if (isFunction(setupResult)) {...}`，由以上参数可知 `setupResult` 是一个函数，所以会进入 `if` 判断

            1. 执行 `instance.render = setupResult`。此时：`instance.render` **有值**

                1. 回顾一下我们之前所学习的 **有状态的响应性组件挂载逻辑**，当 `instance.render` 存在值时，我们后面的渲染就会变得非常简单了。

            3. 代码继续执行，触发 `finishComponentSetup`，这个方法我们之前是熟悉的

                1. 进入 `finishComponentSetup` 方法
                2. 执行 `if (!instance.render)`，因为当前 `instance` 已经存在 `render`，所以 **不会再次为** `render` 赋值

            4. 后面的逻辑就是 **有状态的响应性组件挂载逻辑** 的逻辑了。这里就不再详细说了。

由以上代码可知：

1. 对于 `setup` 函数的 `composition API` 语法的组件挂载，本质上只是多了一个 `setup` 函数的处理
2. 因为 `setup` 函数内部，可以完成对应的 **自洽**，所以我们 **无需** 通过 `call` 方法来改变 `this` 指向，即可得到真实的 `render`
3. 得到真实的 `render` 之后，后面就是正常的组件挂载了

我要提出意见反馈

Copyright © 2022 imooc.com All Rights Reserved | 京ICP备 12003892号-11 京公网安备11010802030151号