慕课网首页 免费课 实战课 体系课 慕课教程 专栏 手记 企业服务

Sunday • 更新于 2022-10-19

2 📜 💄 我的课程



Q

07: 响应性数据的

?

··

索引目录

从所有教程的词条中查询…

↓ 上一节 06: 响应性数据...08: 多层级模板... 下一节 ▶

首页 > 慕课教程 > Vue3源码分析与构建方案 > 07: 响应性数据的编辑器处理: render 函数的...

## 

弗丁ユ卓:complier 編庠 器 - 深入编辑器处理逻辑

- 01: 前言
- 02: 响应性数据的编辑器处
- 理:响应性数据的处理逻辑
- 03:响应性数据的编辑器处
- 理: AST 解析逻辑
- 04: 响应性数据的编辑器处
- 理: JavaScript AST 转化逻辑
- 05:响应性数据的编辑器处
- 理: render 转化逻辑分析
- 06: 响应性数据的编辑器处
- 理: generate 生成 render 函
- 数
- 07: 响应性数据的编辑器外
- 理: render 函数的执行处理
- 08: 多层级模板的编辑器处
- 理: 多层级的处理逻辑
- 09: 基于编辑器的指令(v-xx) 处理: 指令解析的整体逻辑

```
07: 响应性数据的编辑器处理: render 函数的执行处理
```

现在我们已经成功得到了 render 函数, 但是如果我们此时运行测试实例, 将得到对应的错误:

```
* TypeError: Cannot convert undefined or null to object
    at Proxy.render (eval at compileToFunction (index.ts:6:17), <anonymous>:6:3)
    at renderComponentRoot (componentRenderUtils.ts:15:36)
    at ReactiveEffect.componentUpdateFn [as fn] (renderer.ts:177:41)
    at ReactiveEffect.run (effect.ts:130:15)
    at instance.update (renderer.ts:220:50)
    at setupRenderEffect (renderer.ts:223:3)
    at mountComponent (renderer.ts:157:3)
    at processComponent (renderer.ts:143:4)
    at patch (renderer.ts:571:6)
    at Object.render (renderer.ts:591:4)
```

该错误指向 componentRenderUtils.ts 模块中的 renderComponentRoot :

那么出现该错误的原因是什么呢?

我们知道, 当前的 render 函数代码为;

```
const _Vue = Vue

return function render(_ctx, _cache) {
 with (_ctx) {
 const { toDisplayString: _toDisplayString, createElementVNode: _createElementVNode }

return _createElementVNode("div", [], [" hello " + _toDisplayString(msg) + " "])

return _createElementVNode("div", [], [" hello " + _toDisplayString(msg) + " "])
}
```

因为我们使用了 with 所以改变了 作用域 指向,即: msg 等同于 \_ctx.msg 。如果此时 \_ctx 为 null ,则会抛出对应的错误。

通过下面的实例,我们可以看的更加清楚 packages/vue/examples/compiler/with.html:

```
// 代码块

// script>
function render(_ctx) {
    with (_ctx) {
        console.log(msg);
    }
}

// const data = {
    msg: 'world'

// script>
// minus for a console.log(msg);
// minus for a
```

✔ 意见反馈

♡ 收藏教程

□ 标记书签

```
11 render.call(data) // 报错
12 </script>
```

那么由此我们就明白了,当触发 call 方法时,我们需要传递第二个参数为 data,以此作为  $_{
m ctx}$  的 fi

所有,我们需要修改如下代码:

1. 在 packages/runtime-core/src/componentRenderUtils.ts 中:

```
<> 代码块
1 /**
2 *解析 render 函数的返回值
3
4
    export function renderComponentRoot(instance) {
    + // 因为存在 with,所以我们必须保证 data 不能为 undefined
    + const { vnode, render, data = {} } = instance
      let result
8
      try {
9
          // 解析到状态组件
10
          if (vnode.shapeFlag & ShapeFlags.STATEFUL_COMPONENT) {
11
              // 获取到 result 返回值,如果 render 中使用了 this,则需要修改 this 指向
13 + result = normalizeVNode(render!.call(data, data))
14
      } catch (err) {
15
         console.error(err)
16
17
18
19
      return result
20
```

2. 创建 packages/shared/src/toDisplayString.ts, 增加 toDisplayString 方法:

?

 $\odot$ 

3. 在 packages/shared/src/index.ts 中导入 toDisplayString.ts 模块中的所有方法:

4. 在 packages/vue/src/index.ts 中, 导入 toDisplayString 方法:

```
(*)代码块
1 export { toDisplayString } from '@vue/shared'
```

此时运行测试实例, <div> hello world </div> 被正常渲染。

此时我们也可以增加对应的生命周期钩子,修改数据:

```
/> 代码块

// 《script》
const { compile, h, render } = Vue
// 创建 template
const template = `<div> hello {{ msg }} </div>`

// 生成 render 函数
const renderFn = compile(template)

// 意见反馈
// 收藏教程
// 标记书签
```

```
10
    const component = {
 11
       data() {
 12
        return {
 13
         msg: 'world'
       }
 14
      },
 15
 16
    render: renderFn,
 17
      created() {
 18
       setTimeout(() => {
         this.msg = '世界'
 19
 20
        }, 2000);
     }
21
 22
    }
 23
     // 通过 h 函数,生成 vnode
 24
    const vnode = h(component)
 25
 26
    // 通过 render 函数渲染组件
27
render(vnode, document.querySelector('#app'))
29 </script>
响应式数据渲染完成。
   06:响应性数据的编辑器处理:generate 生... ← 上一节 下一节 → 08:多层级模板的编辑器处理:多层级的处...
                              ✔ 我要提出意见反馈
```

企业服务 网站地图 网站首页 关于我们 联系我们 讲师招募 帮助中心 意见反馈 代码托管

**★ %** ♂ **♣** 

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

