慕课网首页 免费课 体系课 慕课教程 专栏 手记 企业服务 实战课

Ē 我的课程



Q

08: 多层级模板的

从所有教程的词条中查询…

首页 > 慕课教程 > Vue3源码分析与构建方案 > 08: 多层级模板的编辑器处理: 多层级的处理...

## 全部开发者教程 Ξ

ポーユ車:compiler 編年器 - 深入編辑器处理逻辑

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

Sunday • 更新于 2022-10-19

↓ 上一节 07: 响应性数据...09: 基于编辑器... 下一节 →

索引目录

## 08: 多层级模板的编辑器处理: 多层级的处理逻辑

在我们处理好响应式的数据处理之后,接下来我们来看一下多层级的视图渲染。

什么叫做多层级的视图渲染呢?我们来看下面的测试实例:

```
<>代码块
     <script>
 1
       const { compile, h, render } = Vue
 3
       // 创建 template
       const template = `<div> <h1>hello world</h1> </div>`
      // 生成 render 函数
      const renderFn = compile(template)
 8
     // 创建组件
    const component = {
10
11
        render: renderFn
12
1.3
      // 通过 h 函数, 生成 vnode
14
      const vnode = h(component)
15
17
       // 通过 render 函数渲染组件
18
      render(vnode, document.querySelector('#app'))
     </script>
```

在该测试实例中, 我们的 template 包含了一个子节点 h1 元素。从现在的 vue-next-mini 中运行该测 试实例,大家可以发现是无法运行的。

那么如果想解析当前的子节点我们应该怎么做呢?

我们知道 compile 的作用就是把模板解析成 render 函数, 我们现在看一下, 现在所解析出的 render:

```
<> 代码块
     function render(_ctx, _cache) {
       with (_ctx) {
         const { createElementVNode: _createElementVNode } = _Vue
5
6
         return _createElementVNode("div", [], [" ", , " "])
7
       }
8
```

在以上代码中, 我们可以发现, 没有渲染出 h1 的原因, 其实就非常简单了, 就是因为第三个参数 ["

如果想要渲染出 h1 , 那么就需要提供出如下的 render:

```
<> 代码块
    function render(_ctx, _cache) {
     with ( ctx) {
       const { createElementVNode: _createElementVNode } = _Vue
              ▶ 意见反馈
                               ♡ 收藏教程
                                               □ 标记书签
```

?

··

```
return _createElementVNode("div", [], [" ", _createElementVNode("h1", [], ["hello wo
}
}
```

那么这样的 render 应该如何实现呢?

对于我们现在的代码而言,解析 render 的代码位于 packages/compiler-core/src/codegen.ts 中,该模块中包含一个 genNode 方法。

该方法是递归解析 codegenNode 的方法逻辑。那么我们可以打印一下当前的 codegenNode 来看一下:

```
<> 代码块
    // console.log(JSON.stringify(ast.codegenNode))
1
2 {
3
       "type": 13,
     "tag": "\"div\"",
 4
      "props": [],
     "children": [
       { "type": 2, "content": " " },
       {
 8
          "type": 1,
9
        "tag": "h1",
10
         "tagType": 0,
11
         "props": [],
12
13
         "children": [{ "type": 2, "content": "hello world" }],
14
         "codegenNode": {
           "type": 13,
15
         "type . ___
"tag": "\"h1\"",
16
         "props": [],
"children": [{ "type": 2, "content": "hello world" }]
18
        }
19
      },
{ "type": 2, "content": " " }
20
21
22 ]
23 }
```

从当前的 codegenNode 中可以看出, children 下,存在一个 type = 1 的节点,这个 **节点就是子节点** h1

而我们想要处理子节点渲染,就需要处理当前的 type = 1 的节点才可以。

我们知道 type = 1 对应的是 NodeTypes.ELEMENT 节点。

所以我们可以在 genNode 方法中增加如下节点处理:

```
<> 代码块
   * 区分节点进行处理
4 function genNode(node, context) {
     switch (node.type) {
6
         case NodeTypes.ELEMENT:
7
           genNode(node.codegenNode!, context)
8
              break
9
           . . .
      }
10
   }
11
```

经过此代码之后, 我们发现 render 函数中的 h1 被成功解析, 模板被成功渲染。

```
07: 响应性数据的编辑器处理: render 函数... ← 上一节 下一节 ▶ 09: 基于编辑器的指令(v-xx)处理: 指令解析...
```

▶ 我要提出意见反馈

✓ 意见反馈

♡ 收藏教程

□ 标记书签

∷

?

.

0

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

**6** 6 4

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

⊡

?

0