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

我的课程



Q

05: JavaScript A

?

 $\odot$ 

索引目录

从所有教程的词条中查询…

#### 全部开发者教程 ≔

最长递增子序列

14: 源码阅读: 场景五: 乱序

下的 diff 比对

15: 框架实现: 场景五: 乱序 下的 diff 比对

16: 总结

#### 第十三章:compiler 编译 器 - 编译时核心设计原则

01: 前言

02: 模板编译的核心流程

03: 抽象语法树 - AST

04: AST 转化为 JavaScript AST,获取 codegenNode

05: JavaScript AST 牛成 render 函数代码

06: 总结

第十四章:compiler 编译 器 - 构建 compile 编译器

```
Sunday • 更新于 2022-10-19
```

↓ 上一节 04: AST 转化为...06: 总结 下一节 ▶

# 05: JavaScript AST 生成 render 函数代码

首页 > 慕课教程 > Vue3源码分析与构建方案 > 05: JavaScript AST 生成 render 函数代码

在上一小节我们已经成功了拿到了对应的 JavaScript AST, 那么接下来我们就根据它生成对应的 rende r 函数。

我们知道利用 render 函数可以完成对应的渲染,根据我们之前了解的规则, render 必须返回一个 vn

例如,我们想要渲染这样的一个结构: <div>hello world</div> , 那么可以构建这样的 render 函数:

```
<>代码块
    render() {
1
      return h('div', 'hello world')
```

### 那么上一小节同样的案例, 我们来看看:

```
<> 代码块
    const template = `<div>hello world</div>
```

这样的一个 template, 最终生成的 render 函数是什么?

#### 还是上一小节的例子:

```
<>代码块
 1
    <script>
      const { compile, h, render } = Vue
       // 创建 template
 3
       const template = `<div>hello world</div>`
       // 生成 render 函数
       const renderFn = compile(template)
 8
      // 打印 renderFn
10
     console.log(renderFn.toString());
11
12
     // 创建组件
13
     const component = {
14
        render: renderFn
1.5
       }
16
      // 通过 h 函数, 生成 vnode
       const vnode = h(component)
20
       // 通过 render 函数渲染组件
21
      render(vnode, document.querySelector('#app'))
     </script>
```

## 打印其中的 renderFn 函数:

```
<> 代码块
    function render(_ctx, _cache) {
   with (_ctx) {
                                           口 标记书签
             ▶ 意见反馈
                            ♡ 收藏教程
```

```
4
5     return (_openBlock(), _createElementBlock("div", null, "hello world"))
6    }
7  }
```

对于以上代码,存在一个 with 语法,这个语法是一个 **不被推荐** 的语法,我们无需太过于关注它,只需要知道它的作用即可:

摘自:《JavaScript 高级程序设计》 with 语句的作用是:**将代码的作用域设置到一个特定的对象中**... 由于大量使用 with 语句会导致性能下降,同时也会给调试代码造成困难,因此在开发大型应用程 序时,不建议使用 with 语句。

我们可以把该代码 (render) 略作改造,直接应用到 render 的渲染中:

```
<> 代码块
1 <script>
const { compile, h, render } = Vue
    // 创建组件
3
4
    const component = {
      render: function (_ctx, _cache) {
5
        with (_ctx) {
          const { openBlock: _openBlock, createElementBlock: _createElementBlock } = Vue /
          return (_openBlock(), _createElementBlock("div", null, "hello world"))
9
10
      }
11
12 }
13
14 // 通过 h 函数, 生成 vnode
15
    const vnode = h(component)
16
    // 通过 render 函数渲染组件
17
18
     render(vnode, document.querySelector('#app'))
    </script>
```

#### 发现可以得到与:

```
c>代码块

render() {
 return h('div', 'hello world')
}
```

同样的结果。

那么观察两个 render 可以发现:

- 1. compiler 最终生成的 render 函数,与我们自己的写的 render 会略有区别。
  - 1. 它会直接通过 createElementBlock 来渲染 块级元素 的方法,比 h 函数更加 "精确"
  - 2. 同时这也意味着, 生成的 render 函数会触发更精确的方法, 比如:
    - createTextVNode
    - createCommentVNode
    - createElementBlock
    - 4. ..
- 2. 虽然, 生成的 render 更加精确, 但是本质的逻辑并没有改变, 已然是一个: return vnode 进行 render 的过程。

▶ 意见反馈

♡ 收藏教程

口 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

k 🤏 😚 🖣

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

⊡

?

<u></u>