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

Q 🏋 💄 我的课程



Q

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 > 13:扩展知识: render 函数的生成方案

全部开发者教程 ः≡

08: 扩展知识: AST 到 JavaScript AST 的转化策略和 注意事项

09:源码阅读:编译器第二步:转化 AST,得到

JavaScript AST 对象

10: 框架实现: 转化 JavaScript AST,构建深度优 先的 AST 转化逻辑

11: 框架实现: 构建 transformXXX 方法, 转化对 ^{应节点}

12:框架实现:处理根节点的 转化,生成 JavaScript AST

13: 扩展知识: render 函数的 生成方案

14:源码阅读:编译器第三步:生成 render 函数

15: 框架实现:构建 CodegenContext 上下文对象

13:扩展知识:render 函数的生成方案

当我们得到了 JavaScript AST 之后, 下面我们就可以生成对应的 render 函数了。

那么我们如何根据 JavaScript AST 来生成对应的 render 函数呢?

我们先来看 vue 源码生成的 render:

1. 在 packages/compiler-core/src/compile.ts 中的 baseCompile 方法下,使用此代码(咱们自己生成的 JavaScript AST):

```
<>代码块
 1
      return generate(
 2
          type: 0,
          children: [
 5
              type: 1,
 6
              tag: 'div',
 8
              tagType: 0,
             props: [],
10
              children: [{ type: 2, content: ' hello world ' }],
11
              codegenNode: {
12
               type: 13,
               tag: '"div"',
13
14
               props: [],
               children: [{ type: 2, content: ' hello world ' }]
15
17
18
          ],
          loc: {},
19
20
          codegenNode: {
21
           type: 13,
            tag: '"div"',
23
            props: [],
            children: [{ type: 2, content: ' hello world ' }]
2.4
25
          },
          // 此处需要主动写入
2.6
27
          helpers: [CREATE ELEMENT VNODE],
28
          components: [],
29
          directives: [],
30
          imports: [],
31
          hoists: [],
          temps: [],
32
33
          cached: []
34
35
        extend({}, options, {
36
          prefixIdentifiers
37
        })
38
      )
```

代替原有的 generate 方法调用。

在 packages/compiler-core/src/codegen.ts 文件中的 generate 方法的最后位置,打印 context. code:

✓ 意见反馈





索引目录

13: 扩展知识: re 函数本质上就是字符拼接的方式 关于字符串的格

?

··

```
/>代码块

console.log(context.code)
```

运行测试实例,可以得到如下打印:

```
c>代码块

1   const _Vue = Vue
2
3   return function render(_ctx, _cache) {
4    with (_ctx) {
5    const { createElementVNode: _createElementVNode } = _Vue
6
7
8
9   return _createElementVNode("div", [], [" hello world "])
10   }
11 }
```

该函数就是通过 generate 方法转化得到的 render 函数, 在该 render 中存在一个 with (_ctx) 这个代码在我们最终期望得到的 render 函数中是不需要的。所以我们最终期望得到的 render 函数为:

```
const _Vue = Vue

return function render(_ctx, _cache) {
    const { createElementVNode: _createElementVNode } = _Vue

return _createElementVNode("div", [], [" hello world "])
}
```

那么下面我们来分析一下上面这个函数的生成,即:生成方案。

函数的生成方案,分为三部分:

- 1. 函数本质上就是一段字符
- 2. 字符串的拼接方式
- 3. 字符串拼接的格式处理

函数本质上就是一段字符

函数本质上就是一段字符,所以我们可以把以上函数比较一个大的 字符串。

那么想要生成这样的一个大字符串,本质上就是各个小的字符串的拼接。

例如,我们可以期望如下的拼接:

```
c>代码块
context.code = `
const_Vue = Vue \n\n return function render(_ctx, _cache) { \n\n const { createElem}
}
```

把以上字符串处理之后, 我们就可以得到一样函数格式的字符:

```
context.code = `
const _Vue = Vue

return function render(_ctx, _cache) {
    const { createElementVNode: _createElementVNode } = _Vue
    return _createElementVNode("div", [], [" hello world "])
}
```

♪ 意见反馈

♡ 收藏教程

口 标记书签

∷

② []

_ ⊙ 当我们明确好了函数本身就是字符,这样的概念之后,那么接下来就是如何拼接这样的字符。

我们把上面的函数分成 4 个部分:

```
1. 函数的前置代码: const _Vue = Vue
```

2. 函数名: function render

3. 函数的参数: _ctx, _cache

4. 函数体:

```
c>代码块

const { createElementVNode: _createElementVNode } = _Vue
 return _createElementVNode("div", [], [" hello world "])
```

我们只需要把以上的内容拼接到一起,那么就可以得到最终的目标结果。

那么为了完成对应的拼接,我们可以提供一个 push 函数:

```
/>代码块

function push (code) {

context.code += code
}

}
```

以此来完成对应的拼接

关于字符串的格式

在去处理这样的一个字符串的过程中,我们不光需要处理拼接,还需要处理对应的格式问题,比如:

```
context.code = `
const _Vue = Vue
(换行)
return function render(_ctx, _cache) {
(缩进) const { createElementVNode: _createElementVNode } = _Vue
return _createElementVNode("div", [], [" hello world "])
}
```

?

 \odot

对于字符串而言, 我们知道换行可以通过 \n 来进行表示, 缩进就是 空格的处理。

所以我们需要再提供对应的方法,来进行对应的处理,比如:

```
<> 代码块
1 context.indentLevel = 0 // 表示缩进
3 // 换行
4 function newline(n: number) {
         newline(context.indentLevel)
5
6 }
8
    // 缩进+换行
9
    function indent(n: number) {
10
          newline(++context.indentLevel)
11
12
13 // 取消缩进 + 换行
14 function deindent(n: number) {
15
           newline(--context.indentLevel)
16 }
17
18 function newline(n: number) {
          context.code += '\n' + ` `.repeat(n)
19
20
```

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 A

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

⊡

?

0