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

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 > 14: 源码阅读: 编译器第三步: 生成 render 函...

## 全部开发者教程

Ξ

08: 扩展知识: AST 到 JavaScript AST 的转化策略和

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

12: 框架实现: 处理根节点的 转化、牛成 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

Sunday • 更新于 2022-10-19

◆ 上一节 13: 扩展知识: ... 15: 框架实现: ... 下一节 ▶

索引目录

14: 源码阅读: 编

?

··

## 14: 源码阅读: 编译器第三步: 生成 render 函数

我们知道生成 render 函数的代码, 主要是 packages/compiler-core/src/codegen.ts 中的 generate 方法,所以我们可以直接在该方法中打断点,进入 debugger (注意:此时我们使用的是 vuex-next-mi ni 生成 JavaScript AST):

- 1. 进入 generate 方法:
- 2. 执行 const context = createCodegenContext(ast, options) 得到 context 上下文:
  - 1. 进入 createCodegenContext 方法,对于 context 而言,我们现在是比较熟悉的了,知道它就 是一个全局变量
  - 2. 观察该方法,可以发现 context 内部存在很多属性和方法,这些属性和方法很多,但是我们不 需要全部关注,只需要关注如下内容即可:

```
<> 代码块
         const context = {
2
            // render 函数代码字符串
            code: ``,
 3
            // 运行时全局的变量名
 4
 5
            runtimeGlobalName: 'Vue'.
 6
            // 模板源
            source: ast.loc.source,
 8
            // 缩进级别
 9
            indentLevel: 0,
            // 需要触发的方法,关联 JavaScript AST 中的 helpers
10
11
            helper(key) {
                return `_${helperNameMap[key]}`
12
13
            },
14
             * 插入代码
15
16
17
            push(code) {
18
                context.code += code
19
            },
20
21
             * 新的一行
22
             */
23
            newline() {
24
                newline(context.indentLevel)
25
             },
26
27
             * 控制缩进 + 换行
28
29
             indent() {
3.0
                newline(++context.indentLevel)
31
            },
32
             * 控制缩进 + 换行
33
34
35
            deindent() {
36
                newline(--context.indentLevel)
37
             }
38
```

▶ 意见反馈

♡ 收藏教程

口 标记书签

```
3. 这些代码相对而言,比较简单,我们在上一小节也提到过对应的作用,这里就不在赘述了。
 3. 执行完成该方法之后,我们可以得到一个 context.code 目前值为 ""
 4. 接下来的代码执行,就是不断往 context.code 填充内容的过程
 5. 代码执行 genFunctionPreamble(ast, preambleContext):
    1. 进入 genFunctionPreamble 方法
    2. 执行 if (ast.helpers.length > 0) 满足条件
       1. 执行 push( const _Vue = ${VueBinding}\n )
       2. 当前的 VueBinding = Vue , 所以以上等同于 push(const _Vue = Vue\n)
       3. 此时, context.code = "const_Vue = Vue\n"
    3. 执行 newline(), 此时, context.code = "const_Vue = Vue\n\n"
    4. 执行 push( return ), 此时, context.code = "const_Vue = Vue\n\nreturn"
 6. genFunctionPreamble 执行完成,此时, context.code = "const_Vue = Vue\n\nreturn"
 7. 代码继续执行, 生成 functionName 和 args
 8. 执行 push( function functionName({signature}) { ) , 此时, context.code = ""const _Vue =
  Vue\n\nreturn function render(_ctx, _cache) {""
 9. 执行 indent() , 此时, context.code = "const _Vue = Vue\n\nreturn function render(_ctx,
   _cache) {\n "
10. 执行 push( with (_ctx) { ) 。
    1. 此时, context.code = "const_Vue = Vue\n\nreturn function render(_ctx, _cache) {\n with
11. 执行 indent() 。此时, context.code = "const _Vue = Vue\n\nreturn function render(_ctx,
  _cache) {\n with (_ctx) { \n"
12. 执行:
        push(`const { ${ast.helpers.map(aliasHelper).join(', ')} } = _Vue`)
        push(`\n`)
        newline()
13. 此时,
    <> 代码块
    context.code = "const _Vue = Vue\n\nreturn function render(_ctx, _cache) {\n with
14. 执行 push(return )
15. 此时:
    <> 代码块
         context.code = "const _Vue = Vue\n\nreturn function render(_ctx, _cache) {\n with
16. 那么到此为止,对于 code 而言,就只剩下最后一块内容,也就是:
    <>代码块
         _createElementVNode("div", [], [" hello world "])
17 而汶里 也是整个 generate 最复杂的—块逻辑
               ✔ 意见反馈
                              ♡ 收藏教程
                                              □ 标记书签
```

?

··

- 18. 这块逻辑由 genNode(ast.codegenNode, context) 开始,我们进入到 genNode 方法
  - 1. 进入 genNode 方法, 目前的参数 node 为:

```
vnode:
vchildren: Array(1)
b0: {type: 2, content: 'hello world '}
length: 1
b[[Prototype]]: Array(0)
props: []
tag: "\"div\""
type: 13
b[[Prototype]]: Object

Ci
```

## 2. 代码执行 switch:

- 1. 当前的 type 为 13 , 对应 case NodeTypes.VNODE\_CALL
- 2. 所以触发 genVNodeCall 方法
  - 1. 进入 genVNodeCall 方法
  - 2. 代码执行 const callHelper: symbol = xxx , 这里的 isBlock = undefined , 所以会 触发 getVNodeHelper(context.inSSR, isComponent) 方法:
    - 1. 进入 getVNodeHelper 方法:
    - 2. 该方法的内部执行非常简单:

```
(>)代码块
1 return ssr || isComponent ? CREATE_VNODE : CREATE_ELEMENT_VNODE
```

?

0

- 3. 返回了两个 Symbol ,分别对应 createVNode 和 createElementVNode 方法
- 4. 此处返回 CREATE\_ELEMENT\_VNODE
- 3. 执行 push(helper(callHelper) +(, node)
- 4. 此时:

```
<> 代码块
1      context.code = "const _Vue = Vue\n\nreturn function render(_ctx, _cache)
```

- 5. 接下来我们就需要为方法填充参数:
- 6. 执行 const args = genNullableArgs(...)
  - 1. 进入 genNullableArgs 方法, 此时的 arg 参数为:

```
▼args: Array(5)
    0: "\"div\""
    ▶1: []
    ▶2: [{...}]
    3: undefined
    4: undefined
    length: 5
    ▶ [[Prototype]]: Array(0)
    i: undefined
```

2. 执行 for 循环, 最终返回值为:

- 7. 代码执行 genNodeList(args, context) , **处理参数的** push
  - 1. 执行 for 循环, 循环会被触发 3 次:
    - 1. 第一次触发: node = 'div'
      - 1. 直接执行 push(node)
      - 2. 执行 push(', ')
    - 2. 第二次触发: node =[]
      - 1. 执行 genNodeListAsArray(node, context)
        - 1. 执行 context.push([)
        - 2. 执行 context.push(])
      - 2. 跳出方法, 执行 push(', ')
    - 3. 第三次触发:

- 1. 执行 genNodeListAsArray(node, context)
  - 1. 执行 context.push([)
  - 2. 执行 genNodeList(nodes, context, multilines)
    - 1. 通常触发 for 循环, 此时: node 的值为:

♪ 意见反馈

♡ 收藏教程

口 标记书签

⊡

② []

·

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 4

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

?

⊡

0

\_