

全部开发者教程

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

16: 框架实现：解析 JavaScript AST，拼接 render 函数

17: 框架实现：新建 compat 模块，把 render 转化为 function

18: 总结

第十五章：compiler 编译器 - 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



Sunday • 更新于 2022-10-19

◀ 上一节 03：响应性数据... 05：响应性数据... 下一节 ▶

## 04：响应性数据的编辑器处理：JavaScript AST 转化逻辑

<> 代码块

```
1 // 需要新增的 JavaScript AST 结构
2 {
3   "type": 8,
4   "loc": {},
5   "children": [
6     {
7       "type": 2,
8       "content": " hello ",
9       "loc": {}
10    },
11    " + ",
12    {
13      "type": 5, // NodeTypes.INTERPOLATION
14      "content": {
15        "type": 4, // NodeTypes.SIMPLE_EXPRESSION
16        "isStatic": false,
17        "constType": 0,
18        "content": "msg",
19        "loc": {}
20      },
21      "loc": {}
22    }
23  ]
24 }
```

对于 JavaScript AST 转化逻辑 我们主要需要明确两个地方：

- 1. 加号的拼接
- 2. NodeTypes.INTERPOLATION 的处理

### 加号的拼接

加号的拼接，我们之前已经处理过了。

在 packages/compiler-core/src/transforms/transformText.ts 中，我们存在一个 transformText 方法，该方法就可以处理复合表达式，生成对应的加号拼接

### NodeTypes.INTERPOLATION

在 vue-next-mini 中的 packages/compiler-core/src/transform.ts 模块下，有一个 traverseNode 方法，该方法可以帮助我们处理节点的转化逻辑。

- 1. 节点的处理需要额外增加 toDisplayString 方法，所以我们需要在 packages/compiler-core/src/runtimeHelpers.ts 中新增 MAP：

<> 代码块

```
1 ...
2 export const TO_DISPLAY_STRING = Symbol('toDisplayString')
3
4 /**
```

#### 索引目录

- 04：响应性数据的加号的拼接
- NodeTypes.IN

?

?

?

?

```

7  */
8  export const helperNameMap = {
9    // 在 renderer 中, 通过 export { createElementVNode as createElementVNode }
10   ...
11   [TO_DISPLAY_STRING]: 'toDisplayString'
12 }

```

2. 在 `packages/compiler-core/src/transform.ts` 中的 `traverseNode` 下, 新增 `NodeTypes.INTERPOLATION` 处理:

<> 代码块

```

1  export function traverseNode(node, context: TransformContext) {
2    ...
3
4    // 继续转化子节点
5    switch (node.type) {
6      ...
7      // 处理插值表达式 {{}}
8      case NodeTypes.INTERPOLATION:
9        context.helper(TO_DISPLAY_STRING)
10       break
11    }
12
13    ...
14  }

```

打印此时生成的 JavaScript AST (不要忘记为 `helpers` 增加 `[CREATE_ELEMENT_VNODE, TO_DISPLAY_STRING]` ) :

<> 代码块

```

1  {
2    type: 0,
3    children: [
4      {
5        type: 1,
6        tag: 'div',
7        tagType: 0,
8        props: [],
9        children: [
10         {
11           type: 8,
12           children: [
13             { type: 2, content: ' hello ' },
14             ' + ',
15             {
16               type: 5,
17               content: { type: 4, isStatic: false, content: 'msg' }
18             },
19             ' + ',
20             { type: 2, content: ' ' }
21           ]
22         }
23       ],
24       codegenNode: {
25         type: 13,
26         tag: '"div"',
27         props: [],
28         children: [
29           {
30             type: 8,
31             children: [
32               { type: 2, content: ' hello ' },
33               ' + ',
34               {
35                 type: 5,
36                 content: { type: 4, isStatic: false, content: 'msg' }
37               },
38               ' + ',

```



```

41         }
42     ]
43 }
44 }
45 ],
46 loc: {},
47 codegenNode: {
48   type: 13,
49   tag: '"div"',
50   props: [],
51   children: [
52     {
53       type: 8,
54       children: [
55         { type: 2, content: ' hello ' },
56         ' + ',
57         {
58           type: 5,
59           content: { type: 4, isStatic: false, content: 'msg' }
60         },
61         ' + ',
62         { type: 2, content: ' ' }
63       ]
64     }
65   ]
66 },
67 helpers: [CREATE_ELEMENT_VNODE, TO_DISPLAY_STRING],
68 components: [],
69 directives: [],
70 imports: [],
71 hoists: [],
72 temps: [],
73 cached: []
74 }

```

我们可以尝试把该内容放入到 `vue 3` 源码的 `baseCompile` 方法中，可以正常渲染。证明 JavaScript AST 渲染完成。

03: 响应性数据的编辑器处理: AST 解析逻辑 ◀ 上一节      下一节 ▶ 05: 响应性数据的编辑器处理: render 转化...

 我要提出意见反馈

