

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 02：响应性数据... 04：响应性数据... 下一节 ▶

### 03：响应性数据的编辑器处理：AST 解析逻辑

<> 代码块

```
1 // 需要新增的 AST 结构
2 {
3   "type": 5, // NodeTypes.INTERPOLATION
4   "content": {
5     "type": 4, // NodeTypes.SIMPLE_EXPRESSION
6     "isStatic": false,
7     "constType": 0,
8     "content": "msg",
9     "loc": {}
10  }
11 }
```

查看 `packages/compiler-core/src/parse.ts` 中的代码逻辑，找到 `parseChildren` 方法。

我们知道该方法的主要是用来解析子节点，内部存在如下的 `if` 逻辑：

<> 代码块

```
1 if (startsWith(s, '{{{')) {
2   ...
3 }
```

对于该逻辑而言，它就是用来处理复合表达式的对应逻辑，我们可以在该逻辑中，生成对应的 `node`：

<> 代码块

```
1 function parseChildren(context: ParserContext, ancestors) {
2   ...
3   while (!isEnd(context, ancestors)) {
4     ...
5     if (startsWith(s, '{{{')) {
6       + node = parseInterpolation(context)
7     }
8     // < 意味着一个标签的开始
9     else if (s[0] === '<') {
10      ...
11    }
12    ...
13  }
14  return nodes
15 }
```

然后增加 `parseInterpolation` 方法：

<> 代码块

```
1 /**
2  * 解析插值表达式 {{ xxx }}
3  */
4 function parseInterpolation(context: ParserContext) {
5   // open = {{
6   // close = }}
7   const [open, close] = ['{{{', '}}']
```

#### 索引目录

03：响应性数据的

📖

🔍

📱

💬

```

10
11 // 获取插值表达式中间的值
12 const closeIndex = context.source.indexOf(close, open.length)
13 const preTrimContent = parseTextData(context, closeIndex)
14 const content = preTrimContent.trim()
15
16 advanceBy(context, close.length)
17
18 return {
19   type: NodeTypes.INTERPOLATION,
20   content: {
21     type: NodeTypes.SIMPLE_EXPRESSION,
22     isStatic: false,
23     content
24   }
25 }
26 }

```

至此，我们成功解析了 AST。

打印解析之后的 AST 可得：

<> 代码块

```

1  const ast = {
2    type: 0,
3    children: [
4      {
5        type: 1,
6        tag: 'div',
7        tagType: 0,
8        props: [],
9        children: [
10         { type: 2, content: ' hello ' },
11         { type: 5, content: { type: 4, isStatic: false, content: 'msg' } },
12         { type: 2, content: ' ' }
13       ]
14     }
15   ],
16   loc: {}
17 }

```

我们可以把以上代码替换到源码的 `baseCompile` 中，发现可正常渲染。证明我们当前生成的 AST 没有问题。

02: 响应性数据的编辑器处理: 响应性数据... ‹ 上一节 下一节 › 04: 响应性数据的编辑器处理: JavaScript ...

✎ 我要提出意见反馈

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

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

✎ 意见反馈

♥ 收藏教程

🔖 标记书签