

全部开发者教程 三

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 01: 前言 03: 响应性数据... 下一节 ▶

02：响应性数据的编辑器处理：响应性数据的处理逻辑

那么首先我们先来看响应性数据的编辑器处理逻辑。他具体指的是什么呢？我们来看如下测试实例 `packages/vue/examples/compiler/compiler-rective.html`：

<> 代码块

```
1  <script>
2    const { compile, h, render } = Vue
3    // 创建 template
4    const template = `<div> hello {{ msg }} </div>`
5
6    // 生成 render 函数
7    const renderFn = compile(template)
8
9    // 创建组件
10   const component = {
11     data() {
12       return {
13         msg: 'world'
14       }
15     },
16     render: renderFn
17   }
18
19   // 通过 h 函数，生成 vnode
20   const vnode = h(component)
21
22   // 通过 render 函数渲染组件
23   render(vnode, document.querySelector('#app'))
24 </script>
```

在以上代码中，我们通过 `data` 声明了一个响应式数据，然后在 `tempalte` 中通过 `{{}}` 进行使用使用。从而得到了 `hello {{ msg }}` 这样一个表达式，这样的表达式我们把它叫做 **复合表达式**

我们可以在 `vue` 的源码的 `baseCompile` 方法中分别查看 `AST`、`JavaScript AST` 和 `render` 函数的值：

<> 代码块

```
1  // AST
2  {
3    "type": 0,
4    "children": [
5      {
6        "type": 1,
7        "tag": "div",
8        "tagType": 0,
9        "props": [],
10       "children": [
11         {
12           "type": 2,
13           "content": " hello ",
14           "loc": {}
15         },
16       ]
17     }
18   }
```

索引目录

02: 响应性数据的

📄

?

📱

💬

```
19         "type": 4, // NodeTypes.SIMPLE_EXPRESSION
20         "isStatic": false,
21         "constType": 0,
22         "content": "msg",
23         "loc": {}
24     },
25     "loc": {}
26 }
27 ],
28 "loc": {}
29 }
30 ],
31 ...
32 }
```

<> 代码块

```
1 // JavaScript AST
2 {
3   "type": 0,
4   "children": [
5     {
6       "type": 1,
7       "ns": 0,
8       "tag": "div",
9       "tagType": 0,
10      "props": [],
11      "isSelfClosing": false,
12      "children": [
13        {
14          "type": 8,
15          "loc": {},
16          "children": [
17            {
18              "type": 2,
19              "content": " hello ",
20              "loc": {}
21            },
22            " + ",
23            {
24              "type": 5, // NodeTypes.INTERPOLATION
25              "content": {
26                "type": 4, // NodeTypes.SIMPLE_EXPRESSION
27                "isStatic": false,
28                "constType": 0,
29                "content": "msg",
30                "loc": {}
31              },
32              "loc": {}
33            }
34          ]
35        }
36      ],
37      "loc": {},
38      "codegenNode": {...},
39      "loc": {}
40    }
41  ]
42 },
43 "helpers": [Symbol("openBlock"), Symbol("createElementBlock"), Symbol("toDisplayString")],
44 "codegenNode": {...},
45 "loc": {}
46 },
47 "loc": {}
48 }
```

<> 代码块

```
1 // context.code
2 ...
```

[意见反馈](#)

[收藏教程](#)

[标记书签](#)



```

4   return function render(_ctx, _cache) {
5     with (_ctx) {
6       const { toDisplayString: _toDisplayString, openBlock: _openBlock, createElementBlock
7
8       return (_openBlock(), _createElementBlock("div", null, " hello " + _toDisplayString(
9     }
10  }

```

由以上内容可以看出，当我们增加了复合表达式之后，AST、JavaScript AST 和 render 函数中多出了如下内容：

<> 代码块

```

1   // AST
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  }

```

<> 代码块

```

1   // JavaScript AST
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  }

```

<> 代码块

```

1   // render
2   const _Vue = Vue
3
4   return function render(_ctx, _cache) {
5     + with (_ctx) {
6     +   const { toDisplayString: toDisplayString, openBlock: _openBlock, createElementBloc
7
8     +   return (_openBlock(), createElementBlock("div", null, " hello " + toDisplayString
9     + }
10  }

```

那么当我们处理复合表达式的编译时，同样也是需要从差异入手，我们只需要填充对应的数据差异，就可以完成最终 render 的生成。



 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