

全部开发者教程

最长递增子序列

14: 源码阅读：场景五：乱序下的 diff 比对

15: 框架实现：场景五：乱序下的 diff 比对

16: 总结

第十三章：compiler 编译器 - 编译时核心设计原则

01: 前言

02: 模板编译的核心流程

03: 抽象语法树 - AST

04: AST 转化为 JavaScript AST，获取 codegenNode

05: JavaScript AST 生成 render 函数代码

06: 总结

第十四章：compiler 编译器 - 构建 compile 编译器



Sunday • 更新于 2022-10-19

← 上一节 03: 抽象语法树 ... 05: JavaScript ... 下一节 →

04: AST 转化为 JavaScript AST，获取 codegenNode

在上一小节中，我们大致了解了抽象语法树 AST 对应的概念。同时我们也知道，AST 最终会通过 transform 方法转化为 JavaScript AST。

那么 JavaScript AST 又是什么样子的呢？这一小节我们来看一下。

我们知道：compiler 最终的目的是把 template 转化为 render 函数。而整个过程分为三步：

1. 生成 AST
2. 将 AST 转化为 JavaScript AST
3. 根据 JavaScript AST 生成 render

所以，生成 JavaScript AST 的目的就是为了最终生成渲染函数最准备的。

上一小节的测试案例，过于复杂，得到的 JavaScript AST 会变得难以理解。所以我们将创建一个新的测试实例，来看一下 JavaScript AST

创建 packages/vue/examples/imooc/compiler/compiler-2.html：

<> 代码块

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

以上代码将得到如下的 AST 和 JavaScript AST：

<> 代码块

```
1 // AST
2 {
3   "type": 0,
4   "children": [
5     {
6       "type": 1,
7       "ns": 0,
8       "tag": "div",
9       "tagType": 0,
10      "props": [],
```

索引目录

04: AST 转化为 JavaScript AST



意见反馈

收藏教程

标记书签

```

12     "children": [
13       {
14         "type": 2,
15         "content": "hello world",
16         "loc": {
17           "start": { "column": 6, "line": 1, "offset": 5 },
18           "end": { "column": 17, "line": 1, "offset": 16 },
19           "source": "hello world"
20         }
21       }
22     ],
23     "loc": {
24       "start": { "column": 1, "line": 1, "offset": 0 },
25       "end": { "column": 23, "line": 1, "offset": 22 },
26       "source": "<div>hello world</div>"
27     }
28   }
29 ],
30 "helpers": [],
31 "components": [],
32 "directives": [],
33 "hoists": [],
34 "imports": [],
35 "cached": 0,
36 "temps": 0,
37 "loc": {
38   "start": { "column": 1, "line": 1, "offset": 0 },
39   "end": { "column": 23, "line": 1, "offset": 22 },
40   "source": "<div>hello world</div>"
41 }
42 }

```

<> 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": 2,
15           "content": "hello world",
16           "loc": {
17             "start": { "column": 6, "line": 1, "offset": 5 },
18             "end": { "column": 17, "line": 1, "offset": 16 },
19             "source": "hello world"
20           }
21         }
22       ],
23       "loc": {
24         "start": { "column": 1, "line": 1, "offset": 0 },
25         "end": { "column": 23, "line": 1, "offset": 22 },
26         "source": "<div>hello world</div>"
27       },
28       "codegenNode": {
29         "type": 13,
30         "tag": "\"div\"",
31         "children": {
32           "type": 2,
33           "content": "hello world",
34           "loc": {
35             "start": { "column": 6, "line": 1, "offset": 5 },
36             "end": { "column": 17, "line": 1, "offset": 16 },

```



[意见反馈](#)

[收藏教程](#)

[标记书签](#)

```

39     },
40     "isBlock": true,
41     "disableTracking": false,
42     "isComponent": false,
43     "loc": {
44       "start": { "column": 1, "line": 1, "offset": 0 },
45       "end": { "column": 23, "line": 1, "offset": 22 },
46       "source": "<div>hello world</div>"
47     }
48   }
49 },
50 ],
51 "helpers": [xxx, xxx],
52 "components": [],
53 "directives": [],
54 "hoists": [],
55 "imports": [],
56 "cached": 0,
57 "temps": 0,
58 "codegenNode": {
59   "type": 13,
60   "tag": "\"div\"",
61   "children": {
62     "type": 2,
63     "content": "hello world",
64     "loc": {
65       "start": { "column": 6, "line": 1, "offset": 5 },
66       "end": { "column": 17, "line": 1, "offset": 16 },
67       "source": "hello world"
68     }
69   },
70   "isBlock": true,
71   "disableTracking": false,
72   "isComponent": false,
73   "loc": {
74     "start": { "column": 1, "line": 1, "offset": 0 },
75     "end": { "column": 23, "line": 1, "offset": 22 },
76     "source": "<div>hello world</div>"
77   }
78 },
79 "loc": {
80   "start": { "column": 1, "line": 1, "offset": 0 },
81   "end": { "column": 23, "line": 1, "offset": 22 },
82   "source": "<div>hello world</div>"
83 }
84 }

```

对于两段内容，其实我们可以发现，两段内容的大多数是完全相同的，唯一不同的地方就是 `codegenNode` 属性，我们把该属性拿出来单独看一下：

<> 代码块

```

1  {
2    ...
3    "children": [
4      {
5        ...
6        // 代码生成节点
7        "codegenNode": {
8          "type": 13, // NodeTypes.VNODE_CALL
9          "tag": "\"div\"", // 标签
10         "children": { // 子节点
11           "type": 2, // NodeTypes.TEXT
12           "content": "hello world", // 内容
13           "loc": { // 位置
14             "start": { "column": 6, "line": 1, "offset": 5 },
15             "end": { "column": 17, "line": 1, "offset": 16 },
16             "source": "hello world"
17           }
18         },
19         ...
20       }
21     ]
22   }
23 }

```

[意见反馈](#)

[收藏教程](#)

[标记书签](#)

```
21     "isComponent": false, // 不是组件
22     "loc": {
23       "start": { "column": 1, "line": 1, "offset": 0 },
24       "end": { "column": 23, "line": 1, "offset": 22 },
25       "source": "<div>hello world</div>"
26     }
27   }
28 }
29 ],
30 ...
31 // 与上面相同
32 "codegenNode": {
33   "type": 13,
34   "tag": "\"div\"",
35   "children": {
36     "type": 2,
37     "content": "hello world",
38     "loc": {
39       "start": { "column": 6, "line": 1, "offset": 5 },
40       "end": { "column": 17, "line": 1, "offset": 16 },
41       "source": "hello world"
42     }
43   },
44   "isBlock": true,
45   "disableTracking": false,
46   "isComponent": false,
47   "loc": {
48     "start": { "column": 1, "line": 1, "offset": 0 },
49     "end": { "column": 23, "line": 1, "offset": 22 },
50     "source": "<div>hello world</div>"
51   }
52 },
53 "loc": {...}
54 }
```

那么由以上对比可以发现，对于 **当前场景下** 的 AST 与 JavaScript AST，相差的就只有 `codegenNode` 这一个属性。

那么这个 `codegenNode` 是什么呢？

`codegenNode` 是 **代码生成节点**。根据我们之前所说的流程可知：JavaScript AST 的作用就是用来 **生成 render 函数**。

那么生成 `render` 函数的关键，就是这个 `codegenNode` 节点。

那么在这一小节我们知道了：

1. AST 转化为 JavaScript AST 的目的是为了最终生成 `render` 函数
2. 而生成 `render` 函数的核心，就是多出来的 `codegenNode` 节点
3. `codegenNode` 节点描述了如何生成 `render` 函数的详细内容

03: 抽象语法树 - AST ◀ 上一节 下一节 ▶ 05: JavaScript AST 生成 render 函数代码

✍ 我要提出意见反馈