

### 最长递增子序列

### 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



◀ 上一节 02: 模板编译的... 04: AST 转化为... 下一节 ▶

通过上一小节的内容，我们可以知道，利用 `parse` 方法可以得到一个 `AST`，那么这个 `AST` 是什么呢？这一小节我们就来说一下。

抽象语法树 (AST) 是一个用来描述模板的 JS 对象，我们以下面的模板为例：

### <> 代码块

```
1 <div v-if="isShow">
2   <p class="m-title">hello world</p>
3 </div>
```

生成的 AST 为:

### <> 代码块

```

1 {
2   "type": 0, // NodeTypes.ROOT
3   "children": [
4     {
5       "type": 1, // NodeTypes.ELEMENT
6       "ns": 0,
7       "tag": "div",
8       "tagType": 0,
9       "props": [
10        {
11          "type": 7, // NodeTypes.DIRECTIVE
12          "name": "if",
13          "exp": {
14            "type": 4, // NodeTypes.SIMPLE_EXPRESSION
15            "content": "isShow",
16            "isStatic": false,
17            "constType": 0,
18            "loc": {
19              "start": { "column": 12, "line": 2, "offset": 12 },
20              "end": { "column": 18, "line": 2, "offset": 18 },
21              "source": "isShow"
22            }
23          },
24          "modifiers": [],
25          "loc": {
26            "start": { "column": 6, "line": 2, "offset": 6 },
27            "end": { "column": 19, "line": 2, "offset": 19 },
28            "source": "v-if=\"isShow\""
29          }
30        }
31      ],
32      "isSelfClosing": false,
33      "children": [
34        {
35          "type": 1, // NodeTypes.ELEMENT
36          "ns": 0,
37          "tag": "p",
38          "tagType": 0,
39          "props": [
40            {

```



②

•

©

 意见反馈

♥ 收藏教程

🔖 标记书签

索引目录

### 03: 抽象语法树 -

```

42         "name": "class",
43         "value": {
44             "type": 2, // NodeTypes.TEXT
45             "content": "title",
46             "loc": {
47                 "start": { "column": 12, "line": 3, "offset": 32 },
48                 "end": { "column": 19, "line": 3, "offset": 39 },
49                 "source": "\"title\""
50             }
51         },
52         "loc": {
53             "start": { "column": 6, "line": 3, "offset": 26 },
54             "end": { "column": 19, "line": 3, "offset": 39 },
55             "source": "class=\"title\""
56         }
57     },
58 ],
59 "isSelfClosing": false,
60 "children": [
61     {
62         "type": 2, // NodeTypes.ELEMENT
63         "content": "hello world",
64         "loc": {
65             "start": { "column": 20, "line": 3, "offset": 40 },
66             "end": { "column": 31, "line": 3, "offset": 51 },
67             "source": "hello world"
68         }
69     }
70 ],
71 "loc": {
72     "start": { "column": 3, "line": 3, "offset": 23 },
73     "end": { "column": 35, "line": 3, "offset": 55 },
74     "source": "<p class=\"title\">hello world</p>"
75 }
76 },
77 ],
78 "loc": {
79     "start": { "column": 1, "line": 2, "offset": 1 },
80     "end": { "column": 7, "line": 4, "offset": 64 },
81     "source": "<div v-if=\"isShow\">\n  <p class=\"title\">hello world</p> \n</div>"
82 }
83 },
84 ],
85 "helpers": [],
86 "components": [],
87 "directives": [],
88 "hoists": [],
89 "imports": [],
90 "cached": 0,
91 "temps": 0,
92 "loc": {
93     "start": { "column": 1, "line": 1, "offset": 0 },
94     "end": { "column": 5, "line": 5, "offset": 69 },
95     "source": "\n<div v-if=\"isShow\">\n  <p class=\"title\">hello world</p> \n</div>\n"
96 }
97 }

```

对于以上这段 AST 而言，内部包含了一些关键属性，需要我们了解：

1. `type`：这里的 `type` 对应一个 `enum` 类型的数据 `NodeTypes`，表示 **当前节点类型**。比如是一个 `ELEMENT` 还是一个 `指令`
  1. `NodeTypes` 可在 `packages/compiler-core/src/ast.ts` 中进行查看 25 行
2. `children`：表示子节点
3. `loc`：location 内容的位置

3. `source` : 原值

4. 注意: 不同的 `type` 类型具有不同的属性值:

1. `NodeTypes.ROOT` -- 0 : 根节点

1. 必然包含一个 `children` 属性, 表示对应的子节点

2. `NodeTypes.ELEMENT` -- 1 : DOM 节点

1. `tag` : 标签名称

2. `tagType` : 标签类型, 对应 `ElementTypes`

3. `props` : 标签属性, 是一个数组

3. `NodeTypes.DIRECTIVE` -- 7 : 指令节点 节点

1. `name` : 指令名

2. `modifiers` : 修饰符

3. `exp` : 表达式

1. `type` : 表达式的类型, 对应 `NodeTypes.SIMPLE_EXPRESSION`, 共有如下类型:

1. `SIMPLE_EXPRESSION` : 简单的表达式

2. `COMPOUND_EXPRESSION` : 复合表达式

3. `JS_CALL_EXPRESSION` : JS 调用表达式

4. `JS_OBJECT_EXPRESSION` : JS 对象表达式

5. `JS_ARRAY_EXPRESSION` : JS 数组表达式

6. `JS_FUNCTION_EXPRESSION` : JS 函数表达式

7. `JS_CONDITIONAL_EXPRESSION` : JS 条件表达式

8. `JS_CACHE_EXPRESSION` : JS 缓存表达式

9. `JS_ASSIGNMENT_EXPRESSION` : JS 赋值表达式

10. `JS_SEQUENCE_EXPRESSION` : JS 序列表达式

2. `content` : 表达式的内容

4. `NodeTypes.ATTRIBUTE` -- 6 : 属性节点

1. `name` : 属性名

2. `value` : 属性值

5. `NodeTypes.TEXT` -- 2 : 文本节点

1. `content` : 文本内容

由以上的 `AST` 解析可知:

1. 所谓的 `AST` 抽象语法树本质上只是一个对象

2. 不同的属性下, 有对应不同的选项, 分别代表了不同的内容。

3. 每一个属性都详细描述了该属性的内容以及存在的位置

4. 指令的解析也包含在 `AST` 中

所以我们可以说: `AST` 描述了一段 `template` 模板的所有内容。

02: 模板编译的核心流程 ◀ 上一节      下一节 ▶ 04: AST 转化为 JavaScript AST, 获取 co...

✍ 我要提出意见反馈

✍ 意见反馈

♥ 收藏教程

🔖 标记书签



