Sunday • 更新于 2022-10-19

上一节 02：模板编译的… 04：AST 转化为… 下一节

03：抽象语法树 - AST

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` 是什么东西呢？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

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

生成的 `AST` 为：

代码块

```
{
  "type": 0, // NodeTypes.ROOT
  "children": [
    {
      "type": 1, // NodeTypes.ELEMENT
      "ns": 0,
      "tag": "div",
      "tagType": 0,
      "props": [
        {
          "type": 7, // NodeTypes.DIRECTIVE
          "name": "if",
          "exp": {
            "type": 4, // NodeTypes.SIMPLE_EXPRESSION
            "content": "isShow",
            "isStatic": false,
            "constType": 0,
            "loc": {
              "start": { "column": 12, "line": 2, "offset": 12 },
              "end": { "column": 18, "line": 2, "offset": 18 },
              "source": "isShow"
            }
          },
          "modifiers": [],
          "loc": {
            "start": { "column": 6, "line": 2, "offset": 6 },
            "end": { "column": 19, "line": 2, "offset": 19 },
            "source": "v-if=\"isShow\""
          }
        }
      ],
      "isSelfClosing": false,
      "children": [
        {
          "type": 1, // NodeTypes.ELEMENT
          "ns": 0,
          "tag": "p",
          "tagType": 0,
          "props": [
```

```
                "name": "class",
                "value": {
                  "type": 2, // NodeTypes.TEXT
                  "content": "title",
                  "loc": {
                    "start": { "column": 12, "line": 3, "offset": 32 },
                    "end": { "column": 19, "line": 3, "offset": 39 },
                    "source": "\"title\""
                  }
                },
                "loc": {
                  "start": { "column": 6, "line": 3, "offset": 26 },
                  "end": { "column": 19, "line": 3, "offset": 39 },
                  "source": "class=\"title\""
                }
              }
            ],
            "isSelfClosing": false,
            "children": [
              {
                "type": 2, // NodeTypes.ELEMENT
                "content": "hello world",
                "loc": {
                  "start": { "column": 20, "line": 3, "offset": 40 },
                  "end": { "column": 31, "line": 3, "offset": 51 },
                  "source": "hello world"
                }
              }
            ],
            "loc": {
              "start": { "column": 3, "line": 3, "offset": 23 },
              "end": { "column": 35, "line": 3, "offset": 55 },
              "source": "<p class=\"title\">hello world</p>"
            }
          }
        ],
        "loc": {
          "start": { "column": 1, "line": 2, "offset": 1 },
          "end": { "column": 7, "line": 4, "offset": 64 },
          "source": "<div v-if=\"isShow\">\n  <p class=\"title\">hello world</p>  \n</div>
        }
      }
    ],
    "helpers": [],
    "components": [],
    "directives": [],
    "hoists": [],
    "imports": [],
    "cached": 0,
    "temps": 0,
    "loc": {
      "start": { "column": 1, "line": 1, "offset": 0 },
      "end": { "column": 5, "line": 5, "offset": 69 },
      "source": "\n<div v-if=\"isShow\">\n  <p class=\"title\">hello world</p>  \n</div>\n
    }
  }
```


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： loction 内容的位置
    1. start：开始位置

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** 。