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 **最终的目的是吧** 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

```
<script>
  const { compile, h, render } = Vue
  // 创建 template
  const template = `<div>hello world</div>`

  // 生成 render 函数
  const renderFn = compile(template)

  // 创建组件
  const component = {
    render: renderFn
  }

  // 通过 h 函数，生成 vnode
  const vnode = h(component)

  // 通过 render 函数渲染组件
  render(vnode, document.querySelector('#app'))
</script>
```

以上代码将得到如下的 AST 和 JavaScript AST：

代码块

```
// AST
{
  "type": 0,
  "children": [
    {
      "type": 1,
      "ns": 0,
      "tag": "div",
      "tagType": 0,
      "props": [],
```

```
      "children": [
        {
          "type": 2,
          "content": "hello world",
          "loc": {
            "start": { "column": 6, "line": 1, "offset": 5 },
            "end": { "column": 17, "line": 1, "offset": 16 },
            "source": "hello world"
          }
        }
      ],
      "loc": {
        "start": { "column": 1, "line": 1, "offset": 0 },
        "end": { "column": 23, "line": 1, "offset": 22 },
        "source": "<div>hello world</div>"
      }
    }
  ],
  "helpers": [],
  "components": [],
  "directives": [],
  "hoists": [],
  "imports": [],
  "cached": 0,
  "temps": 0,
  "loc": {
    "start": { "column": 1, "line": 1, "offset": 0 },
    "end": { "column": 23, "line": 1, "offset": 22 },
    "source": "<div>hello world</div>"
  }
}
```

代码块

```
// JavaScript AST
{
  "type": 0,
  "children": [
    {
      "type": 1,
      "ns": 0,
      "tag": "div",
      "tagType": 0,
      "props": [],
      "isSelfClosing": false,
      "children": [
        {
          "type": 2,
          "content": "hello world",
          "loc": {
            "start": { "column": 6, "line": 1, "offset": 5 },
            "end": { "column": 17, "line": 1, "offset": 16 },
            "source": "hello world"
          }
        }
      ],
      "loc": {
        "start": { "column": 1, "line": 1, "offset": 0 },
        "end": { "column": 23, "line": 1, "offset": 22 },
        "source": "<div>hello world</div>"
      },
      "codegenNode": {
        "type": 13,
        "tag": "\"div\"",
        "children": {
          "type": 2,
          "content": "hello world",
          "loc": {
            "start": { "column": 6, "line": 1, "offset": 5 },
            "end": { "column": 17, "line": 1, "offset": 16 },
```

```
        },
        "isBlock": true,
        "disableTracking": false,
        "isComponent": false,
        "loc": {
          "start": { "column": 1, "line": 1, "offset": 0 },
          "end": { "column": 23, "line": 1, "offset": 22 },
          "source": "<div>hello world</div>"
        }
      }
    }
  ],
  "helpers": [xxx, xxx],
  "components": [],
  "directives": [],
  "hoists": [],
  "imports": [],
  "cached": 0,
  "temps": 0,
  "codegenNode": {
    "type": 13,
    "tag": "\"div\"",
    "children": {
      "type": 2,
      "content": "hello world",
      "loc": {
        "start": { "column": 6, "line": 1, "offset": 5 },
        "end": { "column": 17, "line": 1, "offset": 16 },
        "source": "hello world"
      }
    },
    "isBlock": true,
    "disableTracking": false,
    "isComponent": false,
    "loc": {
      "start": { "column": 1, "line": 1, "offset": 0 },
      "end": { "column": 23, "line": 1, "offset": 22 },
      "source": "<div>hello world</div>"
    }
  },
  "loc": {
    "start": { "column": 1, "line": 1, "offset": 0 },
    "end": { "column": 23, "line": 1, "offset": 22 },
    "source": "<div>hello world</div>"
  }
}
```

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

```
{
  ...
  "children": [
    {
      ...
      // 代码生成节点
      "codegenNode": {
        "type": 13, // NodeTypes.VNODE_CALL
        "tag": "\"div\"", // 标签
        "children": { // 子节点
          "type": 2, // NodeTypes.TEXT
          "content": "hello world", // 内容
          "loc": { // 位置
            "start": { "column": 6, "line": 1, "offset": 5 },
            "end": { "column": 17, "line": 1, "offset": 16 },
            "source": "hello world"
          }
        },
```

```
          "isComponent": false, // 不是组件
          "loc": {
            "start": { "column": 1, "line": 1, "offset": 0 },
            "end": { "column": 23, "line": 1, "offset": 22 },
            "source": "<div>hello world</div>"
          }
        }
      }
    ],
    ...
    // 与上面相同
    "codegenNode": {
      "type": 13,
      "tag": "\"div\"",
      "children": {
        "type": 2,
        "content": "hello world",
        "loc": {
          "start": { "column": 6, "line": 1, "offset": 5 },
          "end": { "column": 17, "line": 1, "offset": 16 },
          "source": "hello world"
        }
      },
      "isBlock": true,
      "disableTracking": false,
      "isComponent": false,
      "loc": {
        "start": { "column": 1, "line": 1, "offset": 0 },
        "end": { "column": 23, "line": 1, "offset": 22 },
        "source": "<div>hello world</div>"
      }
    },
    "loc": {...}
  }
```

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

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

意见反馈　收藏教程　标记书签