

全部开发者教程 :三

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

01: 前言

02: 扩展知识: JavaScript与有限自动状态机

03: 扩展知识: 扫描 tokens 构建 AST 结构的方案

04: 源码阅读: 编译器第一步: 依据模板, 生成 AST 抽象语法树

05: 框架实现: 构建 parse 方法, 生成 context 实例

06: 框架实现：构建有限自动状态机解析模板，扫描 token 生成 AST 结构

07: 框架实现: 生成 `AST`, 构建测试

NO. 425400. ACT 211



Sunday • 更新于 2022-10-19

◀ 上一节 06: 框架实现: ... 08: 扩展知识: ... 下一节 ▶

07: 框架实现: 生成 AST, 构建测试

当 `parseChildren` 处理完成之后，我们可以到 `children`，那么最后我们就只需要利用 `createRoot` 方法，把 `children` 放到 `ROOT` 节点之下即可。

1. 创建 createRoot 方法:

<> 代码块

```

1  /**
2   * 生成 root 节点
3   */
4  export function createRoot(children) {
5      return {
6          type: NodeTypes.ROOT,
7          children,
8          // loc: 位置，这个属性并不影响渲染，但是它必须存在，否则会报错。所以我们给了他一个
9          loc: {}
10     }
11 }

```

2. 在 `baseParse` 中使用该方法:

<> 代码块

```

1  /**
2   * 基础的 parse 方法，生成 AST
3   * @param content tempalte 模板
4   * @returns
5   */
6  export function baseParse(content: string) {
7      // 创建 parser 对象，未解析器的上下文对象
8      const context = createParserContext(content)
9      const children = parseChildren(context, [])
10     return createRoot(children)
11 }

```

至此整个 `parse` 解析流程完成。我们可以在 `packages/compiler-core/src/compile.ts` 中打印得到的 AST：

<> 代码块

```
1 export function baseCompile(template: string, options) {
2   const ast = baseParse(template)
3   console.log(JSON.stringify(ast))
4
5   return {}
6 }
```

得到的内容为：

代码块

```
1 {
2   "type": 0,
```

 意见反馈

♥ 收藏教程

🔖 标记书签

```

5      "type": 1,
6      "tag": "div",
7      "tagType": 0,
8      "props": [],
9      "children": [{ "type": 2, "content": " hello world " }]
10    }
11  ],
12  "loc": {}
13 }

```

我们可以把得到的该 AST 放入到 vue 的源码中进行解析，以此来验证是否正确。

在 vue 源码的 packages/compiler-core/src/compile.ts 模块下 baseCompile 方法中：

<> 代码块

```

1  export function baseCompile(
2    template: string | RootNode,
3    options: CompilerOptions = {}
4  ): CodegenResult {
5    ...
6
7    const ast = isString(template) ? baseParse(template, options) : template
8    + const ast = {
9    +   type: 0,
10   +   children: [
11   +     {
12   +       type: 1,
13   +       tag: 'div',
14   +       tagType: 0,
15   +       props: [],
16   +       children: [{ type: 2, content: ' hello world ' }]
17   +     }
18   +   ],
19   +   loc: {}
20   + }
21
22   ...
23 }

```

运行源码的 compile 方法，浏览器中应该可以渲染 hello world：

<> 代码块

```

1  <script>
2    const { compile, h, render } = Vue
3    // 创建 template
4    const template = ``
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>

```

成功运行，标记着我们的 AST 处理完成。

