

全部开发者教程

08: 扩展知识: AST 到 JavaScript AST 的转化策略和注意事项

09: 源码阅读: 编译器第二步: 转化 AST, 得到 JavaScript AST 对象

10: 框架实现: 转化 JavaScript AST, 构建深度优先的 AST 转化逻辑

11: 框架实现: 构建 transformXXX 方法, 转化对应节点

12: 框架实现: 处理根节点的转化, 生成 JavaScript AST

13: 扩展知识: render 函数的生成方案

14: 源码阅读: 编译器第三步: 生成 render 函数

15: 框架实现: 构建 CodegenContext 上下文对象



Sunday • 更新于 2022-10-19

上一节 10: 框架实现: ... 12: 框架实现: ... 下一节

11: 框架实现: 构建 transformXXX 方法, 转化对应节点

在上一小节, 我们会依次触发 `exitFns[i]()` 方法, 我们知道这些方法其实是 `transformXXX` 方法, 那么我们依次进行实现:

首先是 `transformElement` 方法:

1. 在 `packages/compiler-core/src/transforms/transformElement.ts` 模块中实现 `transformElement` 方法:

<> 代码块

```
1  /**
2   * 对 element 节点的转化方法
3   */
4  export const transformElement = (node, context) => {
5    return function postTransformElement() {
6      node = context.currentNode!
7
8      // 仅处理 ELEMENT 类型
9      if (node.type !== NodeTypes.ELEMENT) {
10        return
11      }
12
13      const { tag } = node
14
15      let vnodeTag = `${tag}`
16      let vnodeProps = []
17      let vnodeChildren = node.children
18
19      node.codegenNode = createVNodeCall(
20        context,
21        vnodeTag,
22        vnodeProps,
23        vnodeChildren
24      )
25    }
26  }
```

2. 在 `packages/compiler-core/src/ast.ts` 中, 创建 `createVNodeCall` 方法:

<> 代码块

```
1  export function createVNodeCall(context, tag, props?, children?) {
2    if (context) {
3      context.helper(CREATE_ELEMENT_VNODE)
4    }
5
6    return {
7      type: NodeTypes.VNODE_CALL,
8      tag,
9      props,
10     children
11   }
12 }
```

索引目录

11: 框架实现: 转...



<> 代码块

```
1 export const CREATE_VNODE = Symbol('createVNode')
2
3 /**
4  * const {xxx} = Vue
5  * 即: 从 Vue 中可以被导出的方法, 我们这里统一使用 createVNode
6  */
7 export const helperNameMap = {
8   // 在 renderer 中, 通过 export { createVNode as createElementVNode }
9   [CREATE_ELEMENT_VNODE]: 'createElementVNode',
10  [CREATE_VNODE]: 'createVNode'
11 }
```

其次是 transformText 方法:

1. 在 packages/compiler-core/src/transforms/transformText.ts 中, 完成 transformText 方法:

<> 代码块

```
1 /**
2  * 将相邻的文本节点和表达式合并为一个表达式。
3  *
4  * 例如:
5  * <div>hello {{ msg }}</div>
6  * 上述模板包含两个节点:
7  * 1. hello: TEXT 文本节点
8  * 2. {{ msg }}: INTERPOLATION 表达式节点
9  * 这两个节点在生成 render 函数时, 需要被合并: 'hello' + _toDisplayString(_ctx.msg)
10 * 那么在合并时就要多出来这个 + 加号。
11 * 例如:
12 * children:[
13 *   { TEXT 文本节点 },
14 *   " + ",
15 *   { INTERPOLATION 表达式节点 }
16 * ]
17 */
18 export const transformText = (node, context) => {
19   if (
20     node.type === NodeTypes.ROOT ||
21     node.type === NodeTypes.ELEMENT ||
22     node.type === NodeTypes.FOR ||
23     node.type === NodeTypes.IF_BRANCH
24   ) {
25     return () => {
26       // 获取所有的子节点
27       const children = node.children
28       // 当前容器
29       let currentContainer
30       // 循环处理所有的子节点
31       for (let i = 0; i < children.length; i++) {
32         const child = children[i]
33         if (isText(child)) {
34           // j = i + 1 表示下一个节点
35           for (let j = i + 1; j < children.length; j++) {
36             const next = children[j]
37             // 当前节点 child 和 下一个节点 next 都是 Text 节点
38             if (isText(next)) {
39               if (!currentContainer) {
40                 // 生成一个复合表达式节点
41                 currentContainer = children[i] = createCompoundExpression(
42                   [child],
43                   child.loc
44                 )
45               }
46               // 在 当前节点 child 和 下一个节点 next 中间, 插入 "+" 号
47               currentContainer.children.push(` + `, next)
48               // 把下一个删除
49               children.splice(j, 1)
50               j--
51             }

```



[意见反馈](#)

[收藏教程](#)

[标记书签](#)

点,

```

53         else {
54             currentContainer = undefined
55             break
56         }
57     }
58 }
59 }
60 }
61 }
62 }
63

```

2. 在 `packages/compiler-core/src/ast.ts` 中, 创建 `createCompoundExpression` 方法:

```

<> 代码块
1  /**
2   * return hello {{ msg }} 复合表达式
3   */
4   export function createCompoundExpression(children, loc) {
5       return {
6           type: NodeTypes.COMPOUND_EXPRESSION,
7           loc,
8           children
9       }
10  }

```

3. 创建 `packages/compiler-core/src/utlis.ts` 模块, 创建 `isText` 方法:

```

<> 代码块
1   export function isText(node) {
2       return node.type === NodeTypes.INTERPOLATION || node.type === NodeTypes.TEXT
3   }

```

至此, 两个 `transformXXX` 方法, 都已经创建完成。

此时创建测试实例:

```

<> 代码块
1   <script>
2       const { compile } = 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   </script>

```

应该可以打印出 `root` 之外的 `children` 的 `codegen`

10: 框架实现: 转化 JavaScript AST, 构建... < 上一节 下一节 > 12: 框架实现: 处理根节点的转化, 生成 Ja...

 我要提出意见反馈

