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

```
/**
 * 对 element 节点的转化方法
 */
export const transformElement = (node, context) => {
    return function postTransformElement() {
        node = context.currentNode!

        // 仅处理 ELEMENT 类型
        if (node.type !== NodeTypes.ELEMENT) {
            return
        }

        const { tag } = node

        let vnodeTag = `"${tag}"`
        let vnodeProps = []
        let vnodeChildren = node.children

        node.codegenNode = createVNodeCall(
            context,
            vnodeTag,
            vnodeProps,
            vnodeChildren
        )
    }
}
```

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

代码块

```
export function createVNodeCall(context, tag, props?, children?) {
    if (context) {
        context.helper(CREATE_ELEMENT_VNODE)
    }

    return {
        type: NodeTypes.VNODE_CALL,
        tag,
        props,
        children
    }
}
```

代码块

```
export const CREATE_VNODE = Symbol('createVNode')

/**
 * const {xxx} = Vue
 * 即：从 Vue 中可以被导出的方法，我们这里统一使用  createVNode
 */
export const helperNameMap = {
    // 在 renderer 中，通过 export { createVNode as createElementVNode }
    [CREATE_ELEMENT_VNODE]: 'createElementVNode',
    [CREATE_VNODE]: 'createVNode'
}
```

其次是 `transformText` 方法：

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

代码块

```
/**
 * 将相邻的文本节点和表达式合并为一个表达式。
 *
 * 例如：
 * <div>hello {{ msg }}</div>
 * 上述模板包含两个节点：
 * 1. hello：TEXT 文本节点
 * 2. {{ msg }}：INTERPOLATION 表达式节点
 * 这两个节点在生成 render 函数时，需要被合并： 'hello' + _toDisplayString(_ctx.msg)
 * 那么在合并时就要多出来这个 + 加号。
 * 例如：
 * children:[
 *  { TEXT 文本节点 },
 *  " + ",
 *  { INTERPOLATION 表达式节点 }
 * ]
 */
export const transformText = (node, context) => {
    if (
        node.type === NodeTypes.ROOT ||
        node.type === NodeTypes.ELEMENT ||
        node.type === NodeTypes.FOR ||
        node.type === NodeTypes.IF_BRANCH
    ) {
        return () => {
            // 获取所有的子节点
            const children = node.children
            // 当前容器
            let currentContainer
            // 循环处理所有的子节点
            for (let i = 0; i < children.length; i++) {
                const child = children[i]
                if (isText(child)) {
                    // j = i + 1 表示下一个节点
                    for (let j = i + 1; j < children.length; j++) {
                        const next = children[j]
                        // 当前节点 child 和 下一个节点 next 都是 Text 节点
                        if (isText(next)) {
                            if (!currentContainer) {
                                // 生成一个复合表达式节点
                                currentContainer = children[i] = createCompoundExpre
                                    [child],
                                    child.loc
                                )
                            }
                            // 在 当前节点 child 和 下一个节点 next 中间，插入 "+" 号
                            currentContainer.children.push(` + `, next)
                            // 把下一个删除
                            children.splice(j, 1)
                            j--
                        }
```

```
                        else {
                            currentContainer = undefined
                            break
                        }
                    }
                }
            }
        }
    }

```

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

代码块

```
/**
 * return hello {{ msg }} 复合表达式
 */
export function createCompoundExpression(children, loc) {
    return {
        type: NodeTypes.COMPOUND_EXPRESSION,
        loc,
        children
    }
}
```

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

代码块

```
export function isText(node) {
    return node.type === NodeTypes.INTERPOLATION || node.type === NodeTypes.TEXT
}
```

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

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

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