Sunday • 更新于 2022-10-19

上一节 12：基于编辑器... 14：总结 下一节

13：基于编辑器的指令(v-xx)处理：生成 render 函数

当 JavaScript AST 构建完成之后，最后我们只需要生成对应的 render 函数即可。

代码块

```
const _Vue = Vue

return function render(_ctx, _cache) {
  with (_ctx) {
    const { createElementVNode: _createElementVNode, createCommentVNode: _createCommentV


    return _createElementVNode("div", [], [
      " hello world ",
      isShow
        ? _createElementVNode("h1", null, ["你好，世界"])
        : _createCommentVNode("v-if", true),
      " "
    ])
  }
}
```

依据以上模板，可以看出， render 的核心处理在于 当前的 三元表达式（ children ）处理：

代码块

```
[
      " hello world ",
      isShow
        ? _createElementVNode("h1", null, ["你好，世界"])
        : _createCommentVNode("v-if", true),
      " "
    ]
```

而对于 codegen 模块而言，解析当前参数的函数为 genNode ，所以我们需要在 genNode 中增加对应的节点处理：

1. 在 packages/compiler-core/src/codegen.ts 中的 genNode 方法下，增加节点处理：

代码块

```
function genNode(node, context) {
    switch (node.type) {
        case NodeTypes.ELEMENT:
        case NodeTypes.IF:
            genNode(node.codegenNode!, context)
            break
        ...
        // JS调用表达式的处理
        case NodeTypes.JS_CALL_EXPRESSION:
            genCallExpression(node, context)
            break
        // JS条件表达式的处理
```

```
            break
        }
    }
```

2. 创建 `genCallExpression` 方法：

代码块

```
/**
 * JS调用表达式的处理
 */
function genCallExpression(node, context) {
    const { push, helper } = context
    const callee = isString(node.callee) ? node.callee : helper(node.callee)
    push(callee + `(`, node)
    genNodeList(node.arguments, context)
    push(`)`)
}
```

3. 创建 `genConditionalExpression` 方法：

代码块

```
/**
 * JS条件表达式的处理。
 * 例如：
 *  isShow
        ? _createElementVNode("h1", null, ["你好，世界"])
        : _createCommentVNode("v-if", true),
 */
function genConditionalExpression(node, context) {
    const { test, consequent, alternate, newline: needNewline } = node
    const { push, indent, deindent, newline } = context
    if (test.type === NodeTypes.SIMPLE_EXPRESSION) {
        // 写入变量
        genExpression(test, context)
    }
    // 换行
    needNewline && indent()
    // 缩进++
    context.indentLevel++
    // 写入空格
    needNewline || push(` `)
    // 写入 ?
    push(`? `)
    // 写入满足条件的处理逻辑
    genNode(consequent, context)
    // 缩进 --
    context.indentLevel--
    // 换行
    needNewline && newline()
    // 写入空格
    needNewline || push(` `)
    // 写入:
    push(`: `)
    // 判断 else 的类型是否也为 JS_CONDITIONAL_EXPRESSION
    const isNested = alternate.type === NodeTypes.JS_CONDITIONAL_EXPRESSION
    // 不是则缩进++
    if (!isNested) {
        context.indentLevel++
    }
    // 写入 else （不满足条件）的处理逻辑
    genNode(alternate, context)
    // 缩进--
    if (!isNested) {
        context.indentLevel--
    }
    // 控制缩进 + 换行
    needNewline && deindent()
}
```

此时生成的 `render` 函数为：

代码块

```
const _Vue = Vue

return function render(_ctx, _cache) {
  with (_ctx) {
    const { createElementVNode: _createElementVNode, createCommentVNode: _createCommentV


    return _createElementVNode("div", [], [" hello world ", isShow
      ? _createElementVNode("h1", null, ["你好，世界"])
      : _createCommentVNode("v-if", true)
    , " "])
  }
}
```

在上述 `render` 中，因为使用了 `createCommentVNode` ，所以我们需要创建并导出该函数。

1. 在 `packages/runtime-core/src/vnode.ts` 中，创建该函数：

代码块

```
/**
 * 创建注释节点
 */
export function createCommentVNode(text) {
    return createVNode(Comment, null, text)
}
```

2. 在 `packages/runtime-core/src/index.ts` 中导出：

代码块

```
export {
    ...
    createCommentVNode
} from './vnode'
```

3. 在 `packages/vue/src/index.ts` 中导出：

代码块

```
export {
    ...
    createCommentVNode
} from '@vue/runtime-core'
```

运行测试实例，效果可以正常展示。

同时，我们可以修改 `isShow` 的值，增加一个延迟的数据变化：

代码块

```
<script>
  const { compile, h, render } = Vue
  // 创建 template
  const template = `<div> hello world <h1 v-if="isShow">你好，世界</h1> </div>`

  // 生成 render 函数
  const renderFn = compile(template)
  console.log(renderFn.toString());
  // 创建组件
  const component = {
    data() {
      return {
        isShow: false
      }
    },
```

```
      created() {
        setTimeout(() => {
          this.isShow = true
        }, 2000);
      }
    }

    // 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>
```

响应式的数据渲染，依然可以正常展示。

12：基于编辑器的指令(v-xx)处理：JavaScri... 上一节 下一节 14：总结

我要提出意见反馈