

全部开发者教程

06: 响应性数据的编辑器处理: generate 生成 render 函数

07: 响应性数据的编辑器处理: render 函数的执行处理

08: 多层级模板的编辑器处理: 多层级的处理逻辑

09: 基于编辑器的指令(v-xx)处理: 指令解析的整体逻辑

10: 基于编辑器的指令(v-xx)处理: AST 解析逻辑 (困难)

11: 基于编辑器的指令(v-xx)处理: JavaScript AST , 构建 vif 转化模块 (困难)

12: 基于编辑器的指令(v-xx)处理: JavaScript AST , transform 的转化逻辑

13: 基于编辑器的指令(v-xx)处理: 生成 render 函数

14: 总结

索引目录

12: 基于编辑器的



Sunday • 更新于 2022-10-19

◀ 上一节 11: 基于编辑器... 13: 基于编辑器... 下一节 ▶

12: 基于编辑器的指令(v-xx)处理: JavaScript AST , transform 的转化逻辑

当 vif 模块构建完成之后, 接下来我们就只需要在 transform 中针对 If 使用 vif 模块进行转化即可

我们知道转化的主要方法为 traverseNode 函数, 所以我们需要在该函数内增加如下代码:

<> 代码块

```
1 export function traverseNode(node, context: TransformContext) {
2   ...
3   // 循环获取节点的 transform 方法, 缓存到 exitFns 中
4   for (let i = 0; i < nodeTransforms.length; i++) {
5     const onExit = nodeTransforms[i](node, context)
6     if (onExit) {
7       + // 指令的 transforms 返回为 数组, 所以需要解构
8       + if (isArray(onExit)) {
9       +   exitFns.push(...onExit)
10      + } else {
11      +   exitFns.push(onExit)
12      + }
13    }
14    + // 因为触发了 replaceNode, 可能会导致 context.currentNode 发生变化, 所以需要在这里校
15    + if (!context.currentNode) {
16    +   // 节点已删除
17    +   return
18    + } else {
19    +   // 节点更换
20    +   node = context.currentNode
21    + }
22  }
23
24  // 继续转化子节点
25  switch (node.type) {
26    + case NodeTypes.IF_BRANCH:
27    case NodeTypes.ELEMENT:
28    case NodeTypes.ROOT:
29      traverseChildren(node, context)
30      break
31      // 处理插值表达式 {}{}
32    case NodeTypes.INTERPOLATION:
33      context.helper(TO_DISPLAY_STRING)
34      break
35    + // v-if 指令处理
36    + case NodeTypes.IF:
37    +   for (let i = 0; i < node.branches.length; i++) {
38    +     traverseNode(node.branches[i], context)
39    +   }
40    +   break
41  }
42  ...
43  ...
44 }
```

至此, 我们在 transform 中就拥有了处理 if 的能力。

📝 意见反馈

📖 收藏教程

🔖 标记书签



<> 代码块

```
1   export function baseCompile(template: string, options = {}) {
2     const ast = baseParse(template)
3
4     transform(
5       ast,
6       extend(options, {
7         + nodeTransforms: [transformElement, transformText, transformIf]
8       })
9     )
10    console.log(JSON.stringify(ast))
11    return generate(ast)
12  }
```

运行测试实例 `packages/vue/examples/compiler/compiler-directive.html`，打印出 JavaScript AST  
(注意：因为 Symbol 不会在 json 字符串下打印，所以我们需要手动加上)：

<> 代码块

```
1   {"type":0,"children":[{"type":1,"tag":"div","tagType":0,"props":[],"children":[{"type":2
```

直接把以上内容复制到 vue3 源码的 `generate` 方法调用处(替换 `ast`)，页面可正常渲染。证明当前的 JavaScript AST 处理完成。

11: 基于编辑器的指令(v-xx)处理: JavaScri... < 上一节 下一节 > 13: 基于编辑器的指令(v-xx)处理: 生成 ren...

✎ 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

Copyright © 2022 imooc.com All Rights Reserved | 京ICP备 12003892号-11 京公网安备11010802030151号



✎ 意见反馈

♥ 收藏教程

🔖 标记书签