Sunday • 更新于 2022-10-19

上一节 08：多层级模板... 10：基于编辑器... 下一节

09：基于编辑器的指令(v-xx)处理：指令解析的整体逻辑

在 vue 中，指令是一个非常重要的环节。 vue 的指令处理主要集中在 compiler 编辑器中。那么接下来我们就来看一下 vue 中的指令处理逻辑。

vue 中提供的指令非常多，大家可以点击 这里来查看所有的内置指令 ，针对于那么多的指令，我们不可能全部进行讲解实现逻辑，所以我们在这里就以 v-if 为例，来为大家讲解指令的解析与处理方案。

我们创建如下测试实例 packages/vue/examples/imooc/compiler/compiler-directive.html：

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
      }
    },
    render: renderFn
  }

  // 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>
```

查看生成的 render 函数：

代码块

```
function render(_ctx, _cache) {
  with (_ctx) {
    const { openBlock: _openBlock, createElementBlock: _createElementBlock, createCommen

    return (_openBlock(), _createElementBlock("div", null, [
      _hoisted_1,
      isShow
        ? (_openBlock(), _createElementBlock("h1", _hoisted_2, "你好，世界"))
        : _createCommentVNode("v-if", true)
    ]))
  }
}
```

根据之前的经验和上面的代码可知：

2. `isShow ? xx : xx`。这个三元表达式是渲染的关键。我们 `v-if` 本质上就是一个 `if` 判断，满足条件则渲染，不满足则不渲染。

那么明确好了对应的 `render` 逻辑之后，接下来我们就来看对应的 `ast` 和 `JavaScript AST`：

代码块

```
{
  "type": 0,
  "children": [
    {
      "type": 1,
      ...
      "children": [
        {
          "type": 2,
          ...
        },
        {
          "type": 1, // NodeTypes.ELEMENT
          "ns": 0,
          "tag": "h1",
          "tagType": 0,
          "props": [
            {
              "type": 7, // NodeTypes.DIRECTIVE
              "name": "if", // 指令名
              // express: 表达式
                "exp": {
                "type": 4, // NodeTypes.SIMPLE_EXPRESSION
                "content": "isShow", // 值
                "isStatic": false,
                "constType": 0,
                "loc": {...}
              },
              "modifiers": [],
              "loc": {...}
            }
          ],
          "isSelfClosing": false,
          "children": [
            {
              "type": 2,
              "content": "你好，世界",
              "loc": {...}
            }
          ],
          "loc": {... }
        }
      ],
      ...
    }
  ],
  ...
}

```

以上的 `AST`，我们进行了对应的简化，主要看备注部分。

由以上 `AST` 可知，针对于指令的处理，主要集中在 `props` 选项中，所以针对于 `AST` 而言，我们 **只需要额外增加 属性（`props`） 的处理即可。**

接下来我们来看 `JavaScript AST` 。

`JavaScript AST` 决定了最终的 `render` 渲染，它的处理更加复杂。我们之前创建过 `transformElement` 与 `transformText` 用来处理 `element` 和 `text` 的渲染，那么同样的道理，针对于指令的处理，我们也需要创建对应的 `transformXXX` 才可以进行处理。

如果以 `v-if` 为例，那么我们需要增加对应的 `vif.ts` 模块。

代码块

```
{
  "type": 9,
  // 分支处理
  "branches": [
    {
      "type": 10, // NodeTypes.IF_BRANCH
      "condition": {
        "type": 4, // NodeTypes.SIMPLE_EXPRESSION
        "content": "isShow",
        "isStatic": false,
        "loc": {}
      },
      "children": [
        {
          "type": 1,
          "tag": "h1",
          "tagType": 0,
          "props": [],
          "children": [{ "type": 2, "content": "你好，世界" }],
          "codegenNode": {
            "type": 13, // NodeTypes.VNODE_CALL
            "tag": "\"h1\"",
            "children": [{ "type": 2, "content": "你好，世界" }]
          }
        }
      ]
    }
  ],
  "codegenNode": {
    "type": 19, // NodeTypes.JS_CONDITIONAL_EXPRESSION
    "test": {
      "type": 4,
      "content": "isShow",
      "isStatic": false,
      "loc": {}
    },
    "consequent": {
      "type": 13, // NodeTypes.VNODE_CALL
      "tag": "\"h1\"",
      "children": [{ "type": 2, "content": "你好，世界" }]
    },
    "alternate": {
      "type": 14, // NodeTypes.JS_CALL_EXPRESSION
      "callee": CREATE_COMMENT,
      "loc": {},
      "arguments": ["\"v-if\"", "true"]
    },
    "newline": true,
    "loc": {}
  }
}
...
```

总结

到这里我们知道了，想要处理指令的编辑逻辑，那么 `AST` 和 `JavaScript AST`，我们都需要进行额外处理：

1. `AST`：额外增加 `props` 属性
2. `JavaScript AST`：额外增加 `branches` 属性

08：多层级模板的编辑器处理：多层级的处... 上一节 下一节 10：基于编辑器的指令(v-xx)处理：AST 解...