Sunday • 更新于 2022-10-19

上一节 03：响应性数据... 05：响应性数据... 下一节

04：响应性数据的编辑器处理：JavaScript AST 转化逻辑

代码块

```
// 需要新增的 JavaScript AST 结构
{
  "type": 8,
  "loc": {},
  "children": [
    {
      "type": 2,
      "content": " hello ",
      "loc": {}
    },
    " + ",
    {
      "type": 5, // NodeTypes.INTERPOLATION
      "content": {
        "type": 4, // NodeTypes.SIMPLE_EXPRESSION
        "isStatic": false,
        "constType": 0,
        "content": "msg",
        "loc": {}
      },
      "loc": {}
    }
  ]
}
```

对于 `JavaScript AST 转化逻辑` 我们主要需要明确两个地方：

1. 加号的拼接
2. `NodeTypes.INTERPOLATION` 的处理

加号的拼接

加号的拼接，我们之前已经处理过了。

在 `packages/compiler-core/src/transforms/transformText.ts` 中，我们存在一个 `transformText` 方法，该方法就可以处理复合表达式，生成对应的加号拼接

NodeTypes.INTERPOLATION

在 `vue-next-mini` 中的 `packages/compiler-core/src/transform.ts` 模块下，有一个 `traverseNode` 方法，该方法可以帮助我们处理节点的转化逻辑。

1. 节点的处理需要额外增加 `toDisplayString` 方法，所以我们需要在 `packages/compiler-core/src/runtimeHelpers.ts` 中新增 `MAP`：

代码块

```
...
export const TO_DISPLAY_STRING = Symbol('toDisplayString')

/**
```

```
 */
export const helperNameMap = {
    // 在 renderer 中，通过 export { createVNode as createElementVNode }
    ...
    [TO_DISPLAY_STRING]: 'toDisplayString'
}
```

2. 在 packages/compiler-core/src/transform.ts 中的 traverseNode 下，新增 NodeTypes.INTERPOLATION 处理：

代码块

```
export function traverseNode(node, context: TransformContext) {
    ...

    // 继续转化子节点
    switch (node.type) {
        ...
        // 处理插值表达式 {{}}
        case NodeTypes.INTERPOLATION:
            context.helper(TO_DISPLAY_STRING)
            break
    }

    ...
}
```

打印此时生成的 JavaScript AST （不要忘记为 helpers 增加 [CREATE_ELEMENT_VNODE, TO_DISPLAY_STRING] ）：

代码块

```
{
  type: 0,
  children: [
    {
      type: 1,
      tag: 'div',
      tagType: 0,
      props: [],
      children: [
        {
          type: 8,
          children: [
            { type: 2, content: ' hello ' },
            ' + ',
            {
              type: 5,
              content: { type: 4, isStatic: false, content: 'msg' }
            },
            ' + ',
            { type: 2, content: ' ' }
          ]
        }
      ],
      codegenNode: {
        type: 13,
        tag: '"div"',
        props: [],
        children: [
          {
            type: 8,
            children: [
              { type: 2, content: ' hello ' },
              ' + ',
              {
                type: 5,
                content: { type: 4, isStatic: false, content: 'msg' }
              },
              ' + ',
```

```
            }
          ]
        }
      }
    ],
    loc: {},
    codegenNode: {
      type: 13,
      tag: '"div"',
      props: [],
      children: [
        {
          type: 8,
          children: [
            { type: 2, content: ' hello ' },
            ' + ',
            {
              type: 5,
              content: { type: 4, isStatic: false, content: 'msg' }
            },
            ' + ',
            { type: 2, content: ' ' }
          ]
        }
      ]
    },
    helpers: [CREATE_ELEMENT_VNODE, TO_DISPLAY_STRING],
    components: [],
    directives: [],
    imports: [],
    hoists: [],
    temps: [],
    cached: []
  }
```

我们可以尝试把该内容放入到 `vue 3` 源码的 `baseCompile` 方法中，可以正常渲染。证明 `JavaScript AST` 渲染完成。

03：响应性数据的编辑器处理：AST 解析逻辑 ◂ 上一节　　下一节 ▸ 05：响应性数据的编辑器处理：render 转化...