第三部分、模板编译技术

7、patch

- 创建节点：三种节点会被创建并插入到DOM中：元素节点、文本节点、注释节点
- 删除节点：对节点操作的封装，跨平台
- 7.4 更新节点
  - 静态节点：不会因为状态的变化而发生变化
  - 新虚拟节点：有文本属性：判断为文本节点，并且仅当文本属性不一样的时候直接把视图中的真实DOM节点的内容改成新虚拟节点的文本
  - 无文本属性
    - 新节点有children的情况：旧节点也有children的情况（7.5节介绍）
    - 新节点无children的情况：旧节点无children属性，空标签或者有文本的文本节点；旧节点的children中有什么删什么；清空节点内容，将新虚拟节点中的children挨个创建成真实的DOM元素节点插入到视图的DOM节点下面
- 7.5 更新子节点 CUMD
  - 创建子节点：插入到未处理节点的前面
  - 更新子节点：7.4
  - 移动子节点：将节点移动到所有未处理节点的最前面
  - 删除子节点：在oldChildren中没有被处理的节点
  - 优化策略
    - 快捷查找：将newChildren和oldChildren都划分为两边
    - 新前==旧前；新后==旧后：都不需要移动，只需要进行更新即可（==表示为同一节点，内容不确定）
    - 新后==旧前：移动到oldChildren中所有未处理的节点最后面
    - 新前==旧后：移动到oldChildren所有未处理的节点最前面
    - 总结：更新过的节点都不需要管，只需要在所有未更新的区间内进行移动和更新操作
    - 以上四种方法对比之后都没有找到相同的节点，再通过循环的方式去oldChildren中寻找
  - 双指针对比决定未更新的节点区域

8、模板编译

- 模板语法：声明式地描述状态和DOM之间地绑定关系
- 目标：生成渲染函数
- 作用：每次执行，根据状态生成一份新的vnode，使用这个vnode进行渲染
- 流程
  - 将模板解析为AST：解析器
  - 遍历AST标记静态节点：优化器
  - 使用AST生成渲染函数：代码生成器

9、解析器

- 解析器的作用：将模板解析成AST
- AST：用js对象描述一个节点，对象中的属性用来保存节点所需的各种数据
- 内部子解析器
  - 1、HTML解析器（最重要）
  - 2、文本解析器
  - 3、过滤解析器
- HTML解析器（词法分析）
  - 通过循环截取字符串，直到空字符串完成解析
  - 开始标签 可以读取tagName、attrs、unary信息
    - const ncname = `[a-zA-Z_][\\w\\-\\.]*` 变量名
    - const qnameCapture = `((?:${ncname}\\:)?${ncname})` 用于匹配svg:path这种情况的标签
    - const startTagOpen = new RegExp(`^<${qnameCapture}`)
    - const attribute = /^\s*([^\s"'<>\/=]+)(?:\s*(=)\s*(?:"([^"]*)"+|'([^']*)'+|([^\s"'=<>`]+)))?/
      - 起始开始匹配空白字符
      - 反向字符集
      - 非捕获 ?用于某些无值的属性
      - 分别匹配双引号、单引号、无引号的情况
    - const startTagClose = /^\s*(\/?)>/ 开始标签结尾：通过捕获的内容判断是否为自闭合标识
  - 结束标签：const endTag = new RegExp(`^<\\/${qnameCapture}[^>]*>`)
  - 注释：const comment = /^<!--/ 再使用indexOf('-->')寻找结束标签，使用substring(4, index)截取
  - 条件注释：const conditionalComment = /^<!\[/ 在使用indexOf(']>')寻找结束标签，使用substring(index+2)截取
  - DOCTYPE：const doctype = /^<!DOCTYPE [^>]+>/i 截取match的长度即可
  - 文本：while循环html 先寻找第一个<，之后开始截取是否符合endTag, startTagOpen, comment, conditionalComment 如果都不符合在剩下的字符串中继续找<,如果找到了截取前一段，继续对后面的进行.test()分析
  - 纯文本处理 script、style、textarea元素的内容
    - const reStackedTag = reCache[stackedTag] || (reCache[stackedTag] = new RegExp('([\\s\\S]*?)(</' + stackedTag + '[^>]*>)', 'i'))
    - reCache是一个缓存正则的对象
    - [\s\S]等于匹配所有字符，*?表示0或者多个，懒惰模式，匹配尽量少的字符
- 文本解析器（parseText）
  - 纯文本
  - 带变量的文本
    - const tagRe = /\{\{((?:.|\n)+?)\}\}/ (?:.|\n) 中 ?:是非捕获模式，.是非换行符以外的单字符 排除\r\n 所以就是非捕获\r以外的单字符
    - ()+? 匹配至少一个，尽可能少的字符，懒惰模式
    - 把parseText('你好{{text}}') 变成 "你好"+_s(text)

10、优化器

- 作用：在AST中找出静态子树并打上标记
- 标记静态子树的好处
  - 每次重新渲染时，不需要为静态子树创建新节点
  - 在虚拟DOM中打补丁patching的过程可以跳过
- 实现步骤
  - 在AST中找出所有静态节点并打上标记 DFS，回溯的时候检查子节点是否为静态节点，不是则改变父节点标记
    - 1、没有使用v-pre标签
    - 2、没有动态绑定
    - 3、没有v-if v-for等条件、循环
    - 4、不是内置标签，如slot或者component
    - 5、是标签名保留字，不是自己取名的组件：!(isHTMLTag(tag) || isSVG(tag))
    - 6、父节点不是带v-for指令的template
    - 7、不存在动态节点才会有的属性
  - 在AST中找出所有静态根节点并打上标记 同样使用DFS，检测到静态节点且有子节点，标记并剪枝：防止过优化，不给只有一个文本节点的元素节点打标记

@ hemisu