☰ 逐行剖析 Vue.js 源码

1. 前言

在前几篇文章中，我们介绍了模板编译流程三大阶段中的第一阶段模板解析阶段，在这一阶段主要做的工作是用解析器将用户所写的模板字符串解析成 AST 抽象语法树，理论上来讲，有了 AST 就可直接进入第三阶段生成 render 函数了。其实不然， Vue 还是很看重性能的，只要有一点可以优化的地方就要将其进行优化。在之前介绍虚拟 DOM 的时候我们说过，有一种节点一旦首次渲染上了之后不管状态再怎么变化它都不会变了，这种节点叫做静态节点，如下：

```html
<ul>
    <li>我是文本信息</li>
    <li>我是文本信息</li>
    <li>我是文本信息</li>
    <li>我是文本信息</li>
    <li>我是文本信息</li>
</ul>
```

在上面代码中， ul 标签下面有5个 li 标签，每个 li 标签里的内容都是不含任何变量的纯文本，也就是说这种标签一旦第一次被渲染成 DOM 节点以后，之后不管状态再怎么变化它都不会变了，我们把像 li 的这种节点称之为静态节点。而这5个 li 节点的父节点是 ul 节点，也就是说 ul 节点的所有子节点都是静态节点，那么我们把像 ul 的这种节点称之为静态根节点。

OK，有了静态节点和静态根节点这两个概念之后，我们再仔细思考，模板编译的最终目的是用模板生成一个 render 函数，而用 render 函数就可以生成与模板对应的 VNode ，之后再进行 patch 算法，最后完成视图渲染。这中间的 patch 算法又是用来对比新旧 VNode 之间存在的差异。在上面我们还说了，静态节点不管状态怎么变化它是不会变的，基于此，那我们就可以在 patch 过程中不用去对比这些静态节点了，这样不就又可以提高一些性能了吗?

所以我们在模板编译的时候就先找出模板中所有的静态节点和静态根节点，然后给它们打上标记，用于告诉后面 patch 过程打了标记的这些节点是不需要对比的，你只要把它们克隆一份去用就好啦。这就是优化阶段存在的意义。

上面也说了，优化阶段其实就干了两件事：

1. 在 AST 中找出所有静态节点并打上标记；
2. 在 AST 中找出所有静态根节点并打上标记；

☰ 逐行剖析 Vue.js 源码

```js
export function optimize (root: ?ASTElement, options: CompilerOptions) {
  if (!root) return
  isStaticKey = genStaticKeysCached(options.staticKeys || '')
  isPlatformReservedTag = options.isReservedTag || no
  // 标记静态节点
  markStatic(root)
  // 标记静态根节点
  markStaticRoots(root, false)
}
```

接下来，我们就对所干的这两件事逐个分析。

2. 标记静态节点

从 AST 中找出所有静态节点并标记其实不难，我们只需从根节点开始，先标记根节点是否为静态节点，然后看根节点如果是元素节点，那么就去向下递归它的子节点，子节点如果还有子节点那就继续向下递归，直到标记完所有节点。代码如下：

```js
function markStatic (node: ASTNode) {
  node.static = isStatic(node)
  if (node.type === 1) {
    // do not make component slot content static. this avoids
    // 1. components not able to mutate slot nodes
    // 2. static slot content fails for hot-reloading
    if (
      !isPlatformReservedTag(node.tag) &&
      node.tag !== 'slot' &&
      node.attrsMap['inline-template'] == null
    ) {
      return
    }
    for (let i = 0, l = node.children.length; i < l; i++) {
      const child = node.children[i]
      markStatic(child)
      if (!child.static) {
        node.static = false
      }
    }
    if (node.ifConditions) {
      for (let i = 1, l = node.ifConditions.length; i < l; i++) {
```

逐行剖析 Vue.js 源码

```
          if (!node.static) {
            node.static = false
          }
        }
      }
    }
  }
```

在上面代码中，首先调用 `isStatic` 函数标记节点是否为静态节点，该函数若返回 `true` 表示该节点是静态节点，若返回 `false` 表示该节点不是静态节点，函数实现如下：

```js
function isStatic (node: ASTNode): boolean {
  if (node.type === 2) { // 包含变量的动态文本节点
    return false
  }
  if (node.type === 3) { // 不包含变量的纯文本节点
    return true
  }
  return !!(node.pre || (
    !node.hasBindings && // no dynamic bindings
    !node.if && !node.for && // not v-if or v-for or v-else
    !isBuiltInTag(node.tag) && // not a built-in
    isPlatformReservedTag(node.tag) && // not a component
    !isDirectChildOfTemplateFor(node) &&
    Object.keys(node).every(isStaticKey)
  ))
}
```

该函数的实现过程其实也说明了如何判断一个节点是否为静态节点。还记得在 `HTML` 解析器在调用钩子函数创建 `AST` 节点时会根据节点类型的不同为节点加上不同的 `type` 属性，用 `type` 属性来标记 `AST` 节点的节点类型，其对应关系如下：

type取值	对应的AST节点类型
1	元素节点
2	包含变量的动态文本节点
3	不包含变量的纯文本节点

所以在判断一个节点是否为静态节点时首先会根据 `type` 值判断节点类型，如果 `type` 值为 2，那么该节点是包含变量的动态文本节点，它就肯定不是静态节点，返回 `false` ；

☰ 逐行剖析 Vue.js 源码

```
    }
```

如果 `type` 值为2，那么该节点是不包含变量的纯文本节点，它就肯定是静态节点，返回 `true` ；

```js
if (node.type === 3) { // 不包含变量的纯文本节点
    return true
}
```

如果 `type` 值为1,说明该节点是元素节点，那就需要进一步判断。

```js
node.pre ||
(
    !node.hasBindings && // no dynamic bindings
    !node.if && !node.for && // not v-if or v-for or v-else
    !isBuiltInTag(node.tag) && // not a built-in
    isPlatformReservedTag(node.tag) && // not a component
    !isDirectChildOfTemplateFor(node) &&
    Object.keys(node).every(isStaticKey)
)
```

如果元素节点是静态节点，那就必须满足以下几点要求：

- 如果节点使用了 `v-pre` 指令，那就断定它是静态节点；
- 如果节点没有使用 `v-pre` 指令，那它要成为静态节点必须满足：
  - 不能使用动态绑定语法，即标签上不能有 `v-` 、 `@` 、 `:` 开头的属性；
  - 不能使用 `v-if` 、 `v-else` 、 `v-for` 指令；
  - 不能是内置组件，即标签名不能是 `slot` 和 `component` ；
  - 标签名必须是平台保留标签，即不能是组件；
  - 当前节点的父节点不能是带有 `v-for` 的 `template` 标签；
  - 节点的所有属性的 `key` 都必须是静态节点才有的 `key` ，注：静态节点的 `key` 是有限的，它只能是 `type` , `tag` , `attrsList` , `attrsMap` , `plain` , `parent` , `children` , `attrs` 之一；

标记完当前节点是否为静态节点之后，如果该节点是元素节点，那么还要继续去递归判断它的子节点，如下：

☰ 逐行剖析 Vue.js 源码

```
        markStatic(child)
        if (!child.static) {
            node.static = false
        }
    }
```

注意，在上面代码中，新增了一个判断：

```js
if (!child.static) {
    node.static = false
}
```

这个判断的意思是如果当前节点的子节点有一个不是静态节点，那就把当前节点也标记为非静态节点。为什么要这么做呢？这是因为我们在判断的时候是从上往下判断的，也就是说先判断当前节点，再判断当前节点的子节点，如果当前节点在一开始被标记为了静态节点，但是通过判断子节点的时候发现有一个子节点却不是静态节点，这就有问题了，我们之前说过一旦标记为静态节点，就说明这个节点首次渲染之后不会再发生任何变化，但是它的一个子节点却又是可以变化的，就出现了自相矛盾，所以我们需要当发现它的子节点中有一个不是静态节点的时候，就得把当前节点重新设置为非静态节点。

循环 `node.children` 后还不算把所有子节点都遍历完，因为如果当前节点的子节点中有标签带有 `v-if` 、 `v-else-if` 、 `v-else` 等指令时，这些子节点在每次渲染时都只渲染一个，所以其余没有被渲染的肯定不在 `node.children` 中，而是存在于 `node.ifConditions` ，所以我们还要把 `node.ifConditions` 循环一遍，如下：

```js
if (node.ifConditions) {
    for (let i = 1, l = node.ifConditions.length; i < l; i++) {
        const block = node.ifConditions[i].block
        markStatic(block)
        if (!block.static) {
            node.static = false
        }
    }
}
```

同理，如果当前节点的 `node.ifConditions` 中有一个子节点不是静态节点也要将当前节点设置为非静态节点。

3. 标记静态根节点

寻找静态根节点根寻找静态节点的逻辑类似，都是从 AST 根节点递归向下遍历寻找，其代码如下：

```js
function markStaticRoots (node: ASTNode, isInFor: boolean) {
  if (node.type === 1) {
    if (node.static || node.once) {
      node.staticInFor = isInFor
    }
    // For a node to qualify as a static root, it should have children t
    // are not just static text. Otherwise the cost of hoisting out wil
    // outweigh the benefits and it's better off to just always render i
    if (node.static && node.children.length && !(
      node.children.length === 1 &&
      node.children[0].type === 3
    )) {
      node.staticRoot = true
      return
    } else {
      node.staticRoot = false
    }
    if (node.children) {
      for (let i = 0, l = node.children.length; i < l; i++) {
        markStaticRoots(node.children[i], isInFor || !!node.for)
      }
    }
    if (node.ifConditions) {
      for (let i = 1, l = node.ifConditions.length; i < l; i++) {
        markStaticRoots(node.ifConditions[i].block, isInFor)
      }
    }
  }
}
```

上面代码中，首先 markStaticRoots 第二个参数是 isInFor ，对于已经是 static 的节点或者是 v-once 指令的节点， node.staticInFor = isInFor ，如下：

```js
if (node.static || node.once) {
    node.staticInFor = isInFor
```

☰ 逐行剖析 Vue.js 源码

接着判断该节点是否为静态根节点，如下：

```js
// For a node to qualify as a static root, it should have children that
// are not just static text. Otherwise the cost of hoisting out will
// outweigh the benefits and it's better off to just always render it f
// 为了使节点有资格作为静态根节点，它应具有不只是静态文本的子节点。 否则，优化的成本将
if (node.static && node.children.length && !(
    node.children.length === 1 &&
    node.children[0].type === 3
)) {
    node.staticRoot = true
    return
} else {
    node.staticRoot = false
}
```

从代码和注释中我们可以看到，一个节点要想成为静态根节点，它必须满足以下要求：

- 节点本身必须是静态节点；
- 必须拥有子节点 `children` ；
- 子节点不能只是只有一个文本节点；

否则的话，对它的优化成本将大于优化后带来的收益。

如果当前节点不是静态根节点，那就继续递归遍历它的子节点 `node.children` 和 `node.ifConditions` ，如下：

```js
if (node.children) {
    for (let i = 0, l = node.children.length; i < l; i++) {
        markStaticRoots(node.children[i], isInFor || !!node.for)
    }
}
if (node.ifConditions) {
    for (let i = 1, l = node.ifConditions.length; i < l; i++) {
        markStaticRoots(node.ifConditions[i].block, isInFor)
    }
}
```

这里的原理跟寻找静态节点相同，此处就不再重复。

4. 总结

本篇文章介绍了模板编译过程三大阶段的第二阶段——优化阶段。

首先，介绍了为什么要有优化阶段，是为了提高虚拟 `DOM` 中 `patch` 过程的性能。在优化阶段将所有静态节点都打上标记，这样在 `patch` 过程中就可以跳过对比这些节点。

接着，介绍了优化阶段主要干了两件事情，分别是从构建出的 `AST` 中找出并标记所有静态节点和所有静态根节点。

最后，分别通过逐行分析源码的方式分析了这两件事具体的内部工作原理。

在 GitHub 上编辑此页

上次更新: 3/24/2020, 5:37:47 AM

← 模板解析阶段(文本解析器) 代码生成阶段 →