

v-for比v-if优先级更高？面试官:回去等通知吧

前言

v-if和v-for哪个优先级更高呢？这是面试官常常问到的一个问题，如果是在三年前，我会毫不犹豫的回答当然是v-for哩，但在3202的今天，如果还这么答，显然是低估了前端技术的日新月异啰。下面我们就来结合编译结果，一探究竟吧。

注意了⚠，以上问题一般指的是v-for和v-if连用的情况，比如

```
1 js
2 复制代码
3 <div v-for='item in itemList' v-if='item.id === 1'{{item.name}}></div>
```

剖析

我们都知道，这个v-if是条件渲染，v-for是列表渲染，都是模版语法，叫这名字当然是因为它们只能在Vue的模版当中用啦。



这些模版语法不是Javascript原生的，因此需要经过一个编译的过程，将它们转为render函数。

经历render函数-->虚拟DOM-->真实DOM 这样的过程,呈现到页面当中。

因此，剖析这个问题的关键就是看编译成的 `render函数`。

在此有请Vue官方设计的工具，可以让我们实时看到编译成的render函数。

[.v2.template-explorer](#)

[.vue3-template-explorer](#)

首先介绍下我们的例子，无非就是渲染一个列表

```
1 js
2 复制代码
3 <template>    <ul      <li v-for="item in list" :key="item.id" v-if="item.id
=== 1"></li    </ul></template><script>    // 这里省略掉变成响应式的过程，因为vue2 3
写法不一      list:[        {name:"JetTsang",id:1},
{name:"juejin",id:2},        {name:"baidu",id:3},    ]    <script>
```

vue2中的编译结果

在V2当中，会看到如下的render函数

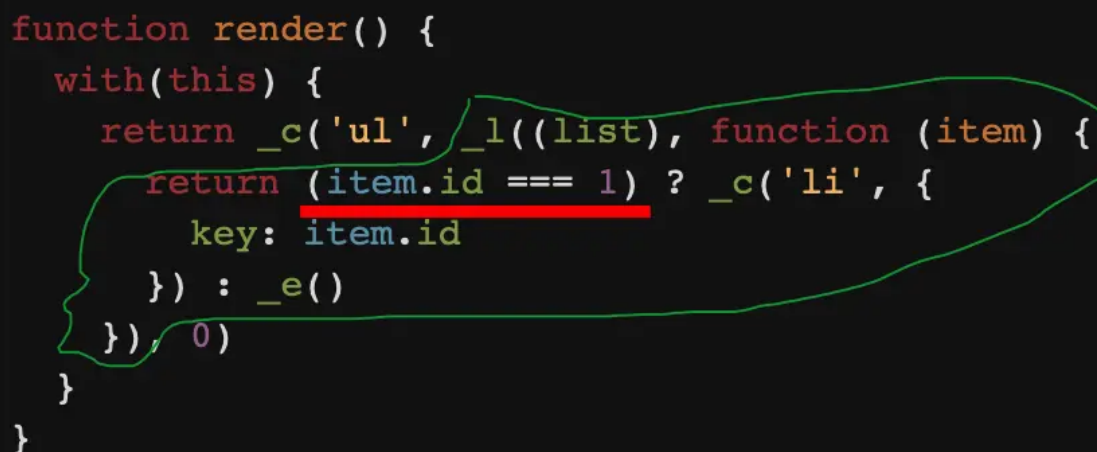


```
Vue Template Explorer (Vue version: 3.2.47) ☐ Serve  
  
1 <ul>  
2   <li v-for="item in list" :key="item.id" v-if="item.id === 1"></li>  
3 </ul>  
  
function render() {  
  with(this) {  
    return _c('ul', _l((list), function (item) {  
      return (item.id === 1) ? _c('li', {  
        key: item.id  
      }) : _e()  
    })), 0)  
  }  
}
```

简单解释一下，这里的_c()就是vm.\$createElement()，意思是创建一个虚拟的element，就是返回值是VNode。

_l就是renderlist函数，第2个参数是一个回调函数，里面会传入的item。

函数里面的item.id===1不就是v-if里的内容吗？



```
function render() {  
  with(this) {  
    return _c('ul', _l((list), function (item) {  
      return (item.id === 1) ? _c('li', {  
        key: item.id  
      }) : _e()  
    })), 0)  
  }  
}
```

The diagram highlights the conditional rendering logic: the function `function (item) { return (item.id === 1) ? _c('li', { key: item.id }) : _e() }` is circled in green. The condition `(item.id === 1)` is underlined in red, and the `_c('li', { key: item.id })` branch is also underlined in red.

这里总结一下就是:先走v-for的逻辑，再根据v-if的条件去判断是否渲染li这个元素，如果没命中v-if的条件，则渲染一个注释节点。

ps:注释节点长这样

不难发现，这里多少有点浪费性能了，如果我list里面有好多个，但符合v-if条件的却比较少，这样先循环，在判断的逻辑，编译出来的render函数效率就比较低下。

其实，如果想要在vue2里的实现这样的逻辑，写代码的时候，就会有提示，让我们先过滤掉list里面的数组，再在模版里面使用。

```
[vue/no-use-v-if-with-v-for]
```

```
The 'arr' variable inside 'v-for' directive should be replaced with a computed property that returns filtered array instead. You should not mix 'v-for' with 'v-if'. eslint-plugin-vue
```

vue3中的编译结果

将上面的template模版粘贴到上面的vue3编译工具网页当中，得到下面的结果

分析一下，这里是先走v-if，再去走v-for，这样在循环当中，就无需屡次判断，并且能在最大程度上，节约性能。

```
export function render(_ctx, _cache, $props, $setup, $data, $options) {
  return (_openBlock(), _createElementBlock("ul", null, [
    (_ctx.item.id === 1)
    ? (_openBlock(true), _createElementBlock(_Fragment, { key: 0 }, _renderList(_ctx.list, (item) => {
      return (_openBlock(), _createElementBlock("li", {
        key: item.id
      })))
    })), 128 /* KEYED_FRAGMENT */)
    : _createCommentVNode("v-if", true)
  ]))
}
```

先判断

再循环

它这里将v-if的条件判断提升到前方了，可以这样去理解

```
<ul>
  <template v-if="item.id === 1">
    <li v-for="item in list" :key="item.id" ></li>
  </template>
</ul>
```

将v-if放在template上，并且将v-for里的逻辑包起来，不妨一下生成的render函数

```
export function render(_ctx, _cache, $props, $setup, $data, $options) {
  return (_openBlock(), _createElementBlock("ul", null, [
    (_ctx.item.id === 1)
    ? (_openBlock(true), _createElementBlock(_Fragment, { key: 0 }, _renderList(_ctx.list, (item) => {
      return (_openBlock(), _createElementBlock("li", {
        key: item.id
      })))
    })), 128 /* KEYED_FRAGMENT */)
    : _createCommentVNode("v-if", true)
  ]))
}
```

完全一模一样有木有？明显vue3当中就是这么去优化的吧？

因此这里就带来一个问题，在里面根本用不了item呢，这样编译必然报错的

这当然就需要看官们的奇思妙想去优化这逻辑啰，比如用计算属性啰,返回id是1的item不就行啰

ps: (当然有人会说，你都渲染id为1的了，干嘛用for啊，这里只是这么举例，实际开发当中肯定有用得上的类似需求)

```
1 js
2 复制代码
3 const newList = computed((()=>list.filter(i=>i.id===1)))// 后续再v-for去渲染即可
```

总结

显然，在V2当中，v-for的优先级更高，而在V3当中，则是v-if的优先级更高。在V3当中，做了v-if的提升优化，去除了没有必要的计算，但同时也会带来一个无法取到v-for当中遍历的item问题，这就需要开发者们采取其他灵活的方式去解决这种问题。

看到这里是不是对vue的编译有了更深刻的体会，原来vue在编译过程当中做了这么多细节的优化，是啊，一个好的产品就是需要不断的打磨呢。

加餐

或许你还会对vue的编译过程感兴趣？ 那你需要去了解什么是[抽象语法树AST](#)。

vue的编译过程是：

- 1.先从字符串生成AST --- parse
- 2.对AST进行优化处理（标记静态节点等） --- optimize
- 3.将AST对象转为字符串形式的JS代码 --- generate