



下载APP



## 24 | 树：如何设计一个树形组件？

2021-12-15 大圣

《玩转Vue 3全家桶》

课程介绍 &gt;

**讲述：大圣**

时长 07:39 大小 7.02M



你好，我是大圣。

上一讲，我们一起学习了弹窗组件的设计与实现，这类组件的主要特点是需要渲染在最外层 body 标签之内，并且还需要支持 JavaScript 动态创建和调用组件。相信学完上一讲，你不但会对弹窗类组件的实现加深理解，也会对 TDD 模式更有心得。

除了弹窗组件，树形组件我们在前端开发中经常用到，所以今天我就跟你聊一下树形组件的设计思路跟实现细节。



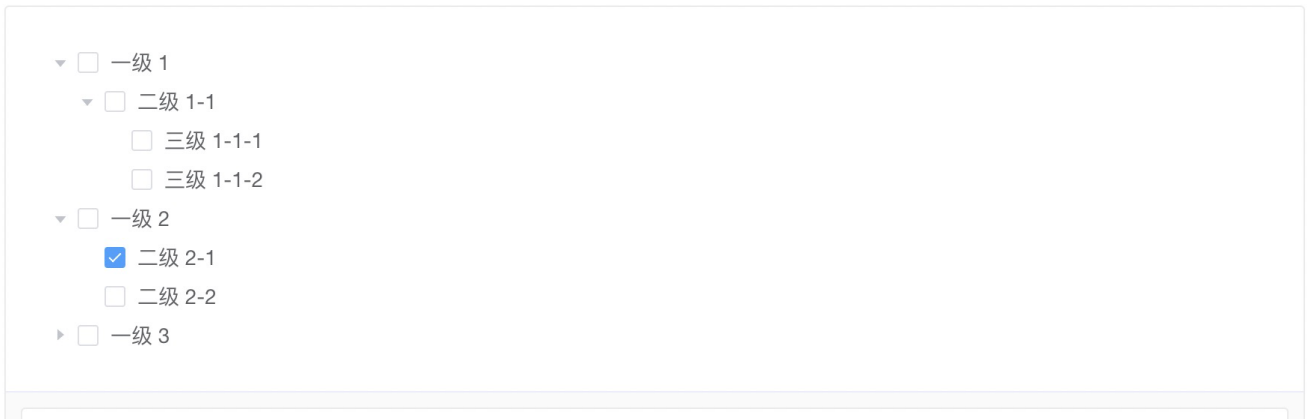
### 组件功能分析

我们进入 [Element3 的 Tree 组件文档页面](#)，现在我们对 Vue 的组件如何设计和实现已经很熟悉了，我重点挑跟之前组件设计不同的地方为你讲解。

在设计新组件的时候，我们需要重点考虑的就是树形组件和之前我们之前的 Container、Button、Notification 有什么区别。树形组件的主要特点是可以无限层级、这种需求在日常工作和生活中其实很常见，比如后台管理系统的菜单管理、文件夹管理、生物分类、思维导图等等。

## 默认展开和默认选中

可将 Tree 的某些节点设置为默认展开或默认选中



根据上图所示，我们可以先拆解出树形组件的功能需求。

首先，树形组件的节点可以无限展开，父节点可以展开和收起节点，并且每一个节点有一个复选框，可以切换当前节点和所有子节点的选择状态。另外，同一级所有节点选中的时候，父节点也能自动选中。

下面的代码是 Element3 的 Tree 组件使用方式，所有的节点配置都是一个 data 对象实现的。每个节点里的 label 用来显示文本；expanded 显示是否展开；checked 用来决定复选框选中列表，data 数据内部的 children 属性用来配置子节点数组，子节点的数据结构和父节点相同，可以递归实现。

复制代码

```
1 <el-tree
2   :data="data"
3   show-checkbox
4   v-model:expanded="expandedList"
5   v-model:checked="checkedList"
6   :defaultNodeKey="defaultNodeKey"
```

```
7 >
8 </el-tree>
9 <script>
10   export default {
11     data() {
12       return {
13         expandedList: [4, 5],
14         checkedList: [5],
15         data: [
16           {
17             id: 1,
18             label: '一级 1',
19             children: [
20               {
21                 id: 4,
22                 label: '二级 1-1',
23                 children: [
24                   {
25                     id: 9,
26                     label: '三级 1-1-1'
27                   },
28                   {
29                     id: 10,
30                     label: '三级 1-1-2'
31                   }
32                 ]
33               }
34             ]
35           },
36           {
37             id: 2,
38             label: '一级 2',
39             children: [
40               {
41                 id: 5,
42                 label: '二级 2-1'
43               },
44               {
45                 id: 6,
46                 label: '二级 2-2'
47               }
48             ]
49           }
50         ],
51         defaultNodeKey: {
52           childNodes: 'children',
53           label: 'label'
54         }
55       }
56     }
57   }
58 }
```

```
59 </script>
60
```

## 递归组件

这里父节点和子节点的样式操作完全一致，并且可以无限嵌套，这种需求需要组件递归来实现，也就是组件内部渲染自己渲染自己。

想要搞定递归组件，我们需要先明确什么是递归，递归的概念也是我们前端进阶过程中必须要掌握的知识点。

前端的场景中，树这个数据结构出现的频率非常高，浏览器渲染的页面是 Dom 树，我们内部管理的是虚拟 Dom 树，**树形结构是一种天然适合递归的数据结构。**

我们先来做一个算法题感受一下，我们来到 [leetcode 第 226 题反转二叉树](#)，题目的描述很简单，就是把属性结构反转，下面是题目的描述：

每一个节点的 val 属性代表显示的数字，left 指向左节点，right 指向右节点，如何实现 invertTree 去反转这一个二叉树，也就是所有节点的 left 和 right 互换位置呢？

[复制代码](#)

```
1
2 输入
3      4
4    /  \
5   2    7
6  / \  / \
7 1  3 6  9
8 输出
9      4
10   /  \
11  7    2
12 / \  / \
13 9  6 3  1
14 节点的构造函数
15 /**
16  * Definition for a binary tree node.
17  * function TreeNode(val, left, right) {
18  *     this.val = (val===undefined ? 0 : val)
19  *     this.left = (left===undefined ? null : left)
20  *     this.right = (right===undefined ? null : right)
21  * }
22  */
```

输入的左右位置正好相反，而且每个节点的结构都相同，这就是非常适合递归的场景。递归的时候，我们首先需要思考递归的核心逻辑如何实现，这里就是两个节点如何交换，然后就是递归的终止条件，否则递归函数就会进入死循环。

下面的代码中，设置 `invertTree` 函数的终止条件是 `root` 是 `null` 的时候，也就是如果节点不存在的时候不需要反转。这里我们只用了一行解构赋值的代码就实现了，值得注意的是右边的代码中我们递归调用了 `invertTree` 去递归执行，最终实现了整棵树的反转。

[复制代码](#)

```
1 var invertTree = function(root) {  
2   // 递归 终止条件  
3   if(root==null) {  
4     return root  
5   }  
6   // 递归的逻辑  
7   [root.left, root.right] = [invertTree(root.right), invertTree(root.left)]  
8   return root  
9 }
```

树形组件的数据结构内部的 `children` 可以无限嵌套，处理这种数据结构，就需要使用递归的算法思想。有了上面这个算法题的基础后，我们后面再学习树形组件如何实现就能更加顺畅了。

## 组件实现

首先我们进入到 `Element3` 的 `tree` 文件夹内部，然后找到 `tree.vue` 文件。`tree.vue` 是组件的入口容器，用于接收和处理数据，并将数据传递给 `TreeNode.vue`；`TreeNode.vue` 负责渲染树形组件的选择框、标题和递归渲染子元素。

在下面的代码中，我们提供了 `el-tree` 的容器，还导入了 `el-tree-node` 进行渲染。`tree.vue` 通过 `provide` 向所有子元素提供 `tree` 的数据，通过 `useExpand` 判断树形结构的展开状态，并且用到了 `watchEffect` 去向组件外部通知 `update:expanded` 事件。


[复制代码](#)

```
1 <template>  
2   <div class="el-tree">
```

```
3     <el-tree-node v-for="child in tree.root.childNodes" :node="child" :key="ch
4   </div>
5 </template>
6
7 <script>
8 import ElTreeNode from './TreeNode.vue'
9 const instance = getCurrentInstance()
10 const tree = new Tree(props.data, props.defaultNodeKey, {
11   asyncLoadFn: props.asyncLoadFn,
12   isAsync: props.async
13 })
14 const state = reactive({
15   tree
16 })
17 provide('elTree', instance)
18 useTab()
19 useExpand(props, state)
20
21 function useExpand(props, state) {
22   const instance = getCurrentInstance()
23   const { emit } = instance
24
25   if (props.defaultExpandAll) {
26     state.tree.expandAll()
27   }
28
29   watchEffect(() => {
30     emit('update:expanded', state.tree.expanded)
31   })
32
33   watchEffect(() => {
34     state.tree.setExpandedByIdList(props.expanded, true)
35   })
36
37   onMounted(() => {
38     state.tree.root.expand(true)
39   })
40 }
41
42
43 </script>
```

然后我们进入到 `Tree.Node.vue` 文件中，`tree-node` 组件是树组件的核心，一个 `TreeNode` 组件包含四个部分：展开按钮、文本的多选框、每个节点的标题和递归的 `children` 子节点。

我们先来看 `TreeNode.vue` 的模板基本结构，可以把下面的 `div` 标签分成四个部分：`el-tree-node__content` 负责每个树节点的渲染，第一个 `span` 就是渲染展开符；`el-checkbox` 组件负责显示复选框，并且绑定了 `node.isChecked` 属性；`el-tree-node__contentn` 负责渲染树节点的标题；`el-tree__children` 负责递归渲染 `el-tree-node` 节点，组件内部渲染自己，这就是组件递归的写法。

 复制代码

```
1 <div
2   v-show="node.isVisable"
3   class="el-tree-node"
4   :class="{
5     'is-expanded': node.isExpanded,
6     'is-current': elTree.proxy.dragState.current === node,
7     'is-checked': node.isChecked,
8   }"
9   role="TreeNode"
10  ref="TreeNode"
11  :id="'TreeNode' + node.id"
12  @click.stop="onClickNode"
13 >
14  <div class="el-tree-node__content">
15    <span
16      :class="[
17        { expanded: node.isExpanded, 'is-leaf': node.isLeaf },
18        'el-tree-node__expand-icon',
19        elTree.props.iconClass
20      ]"
21      @click.stop="
22        node.isLeaf ||
23        (elTree.props.accordion ? node.collapse() : node.expand())
24      ">
25    </span>
26    <el-checkbox
27      v-if="elTree.props.showCheckbox"
28      :modelValue="node.isChecked"
29      @update:modelValue="onChangeCheckbox"
30      @click="elTree.emit('check', node, node.isChecked, $event)"
31    >
32    </el-checkbox>
33    <el-node-content
34      class="el-tree-node__label"
35      :node="node"
36    ></el-node-content>
37  </div>
38  <div
39    class="el-tree-node__children"
40    v-show="node.isExpanded"
41    v-if="!elTree.props.renderAfterExpand || node.isRendered"
```

```
42     role="group"
43     :aria-expanded="node.isExpanded"
44   >
45     <el-tree-node
46       v-for="child in node.childNodes"
47       :key="child.id"
48       :node="child"
49     >
50     </el-tree-node>
51   </div>
52 </div>
```

然后我们看下 tree-node 中我们需要处理的数据有哪些。下面的代码中，我们先通过 inject 注入 tree 组件最完成的配置。然后在点击节点的时候，通过判断 elTree 的全局配置，去决定点击之后的切换功能，并且在展开和 checkbox 切换的同时，通过 emit 对父组件触发事件。

[复制代码](#)

```
1  const elTree = inject('elTree')
2  const onClickNode = (e) => {
3    !elTree.props.expandOnClickNode ||
4    props.node.isLeaf ||
5    (elTree.props.accordion ? props.node.collapse() : props.node.expand())
6
7    !elTree.props.checkOnClickNode ||
8    props.node.setChecked(undefined, elTree.props.checkStrictly)
9
10   elTree.emit('node-click', props.node, e)
11   elTree.emit('current-change', props.node, e)
12   props.node.isExpanded
13     ? elTree.emit('node-expand', props.node, e)
14     : elTree.emit('node-collapse', props.node, e)
15 }
16
17 const onChangeCheckbox = (e) => {
18   props.node.setChecked(undefined, elTree.props.checkStrictly)
19   elTree.emit('check-change', props.node, e)
20 }
```

□

到这里，树结构的渲染其实就结束了。



但是有些场景我们需要对树节点的渲染内容进行自定。比如后面这段代码，我们在节点的右侧加上 append 和 delete 操作按钮，这种需求在菜单树的管理中很常见。

这个时候我们节点需要支持内容的自定义，然后我们注册了 el-node-content 组件。这个组件使用起来非常简单，由于我们还需要支持节点的自定义渲染，所以要把这部分抽离成组件。当 slots.default 为函数的时候，返回函数的执行内容；或者传递的 renderContent 是函数的话，也要返回函数执行的结果。

[复制代码](#)

```
1 import { TreeNode } from './entity/TreeNode'
2 import { inject, h } from 'vue'
3
4 render(ctx) {
5   const elTree = inject('elTree')
6   if (typeof elTree.slots.default === 'function') {
7     return elTree.slots.default({ node: ctx.node, data: ctx.node.data.raw })
8   } else if (typeof elTree.props.renderContent === 'function') {
9     return elTree.props.renderContent({
10       node: ctx.node,
11       data: ctx.node.data.raw
12     })
13   }
14
15   return h('span', ctx.node.label)
16 }
```

这样，用户就可以利用 render-content 属性传递一个函数的方式，去实现内容的自定义渲染。

我们还是结合代码例子做理解，下面的代码中用了 render-content 的方式返回树形结构的渲染结果，render-content 传递的函数内部会根据 node 和 data 数据，返回对应的标题，并且新增了两个 el-button 组件。

[复制代码](#)

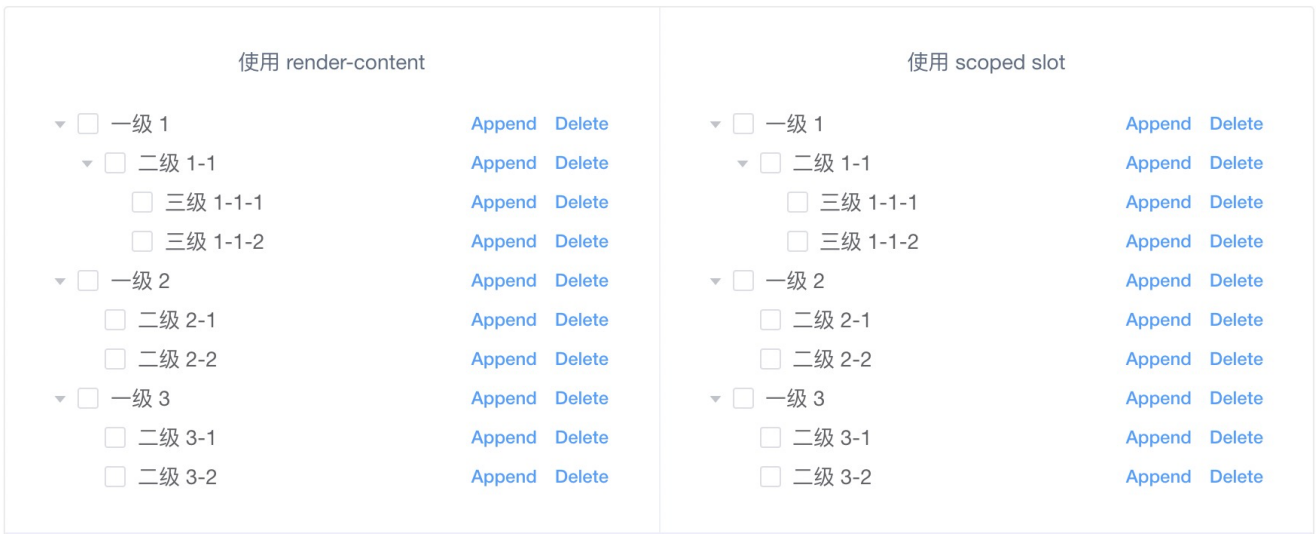
```
1 <div class="custom-tree-container">
2   <div class="block">
3     <p>使用 render-content</p>
4     <el-tree
5       :data="data1"
6       show-checkbox
7       default-expand-all
```

```
8       :expand-on-click-node="false"
9       :render-content="renderContent"
10    >
11    </el-tree>
12  </div>
13 </div>
14 <script>
15 function renderContent({ node, data }) {
16   return (
17     <span class="custom-tree-node">
18       <span>{data.label}</span>
19       <span>
20         <el-button
21           size="mini"
22           type="text"
23           onClick={() => this.append(node, data)}
24         >
25           Append
26         </el-button>
27         <el-button
28           size="mini"
29           type="text"
30           onClick={() => this.remove(node, data)}
31         >
32           Delete
33         </el-button>
34       </span>
35     </span>
36   )
37 }
38 </script>
```

上面的代码会渲染出下面的示意图的效果。

## 自定义节点内容

节点的内容支持自定义，可以在节点区添加按钮或图标等内容



最后，我们还可以对树实现更多操作方式的支持。

比如我们可以支持树形结构的拖拽修改、可以把任何任意节点拖拽到其他树形内部、修改整个树形结构的内容。想要实现这些功能，我们就需要监听节点的 `drag-over`、`drag-leave` 等拖拽事件，在 `drop` 事件执行的时候，把拖拽的节点数据，复制给拖拽的节点中完成修改即可。这部分代码，同学们可以自行去 Element3 拓展学习。

## 总结

今天的主要内容就讲完啦，我们来总结一下今天学到的内容吧。

首先我们分析了树形组件的设计需求、我们需要递归组件的形式去实现树形节点的无限嵌套，然后通过算法题的形式掌握了递归的概念，这个概念在 Vue 组件中也是一样的，每个组件返回 `name` 后，可以通过这个 `name` 在组件内部来调用自己，这样就可以很轻松地实现 Tree 组件。

tree 组件具体要分成三个组件进行实现。最外层的 tree 组件负责整个树组件的容器，内部会通过 `provide` 方法为子元素提供全局的配置和操作方法。每个 tree 的配置中的 `title`、`expanded`、`checked` 树形作为树组件显示的主体内容。`children` 是一个深层嵌套的数组，我们需要用递归组件的方式渲染出完成的树，tree 内部的 `tree-node` 组件就负责递归渲染出完成的树形结构。

最后，我们想支持树节点的自定义渲染，这就需要在 `teree-node` 内部定制 `tree-node-content` 组件，用来渲染用户传递的 `render-content` 或者默认的插槽函数。

树形数据在我们日常开发项目中也很常见，菜单、城市选择、权限等数据都很适合树形结构，学会树形结构的处理，能很好地帮助我们在日常开发中应对更复杂的需求。

## 思考题

最后留一个思考题吧。我们的树形组件现在是全部节点的渲染，如果我们有 1000 个节点要渲染，如何对这个树形节点做性能优化呢？

欢迎你在评论区分享你的答案，也欢迎你把这一讲的内容分享给你的同事和朋友们，我们下一讲再见。

分享给需要的人，Ta订阅后你可得 **20 元现金奖励**

 生成海报并分享

 赞 2     提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇   23 | 弹窗：如何设计一个弹窗组件？

下一篇   25 | 表格：如何设计一个表格组件

## 更多课程推荐

# 跟月影学可视化

## 系统掌握图形学与可视化核心原理

月影

奇虎 360 奇舞团团长

可视化 UI 框架 SpriteJS 核心开发者



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

### 精选留言 (5)

写留言



一个自闭的人

2021-12-15

element3源码阅读和分析 -.-|||

展开 ∨

作者回复：你好，这一讲主要是演示和剖析element3源码，后面会有ts组件库的加餐的



👍 2



亮

2021-12-17

在引入一个组件后，node\_modules 报错，比如vue3.0引入element-plus后会报一些类似这样的错误：Uncaught Error: Cannot find module 'vue' at webpackMissingModule (app.js:formatted:2896) at Module../src/main.js；网上找了一遍也没找到能解决方法，遇到这种情况，应该从哪些方面入手呢，直接调试node\_modules吗？

展开 ∨





**费城的二鹏**

2021-12-15

如果渲染节点比较多，可以考虑重用节点，用屏幕区域的少量节点复用的方式，随着滚动动态展示对应节点。

展开 ∨



**海阔天空**

2021-12-15

树形组件在实际项目中懒加载的模式用得比较多。teree-node 内部定制 tree-node-content 组件基本上可以满足我们对树的操作了。

展开 ∨



**joker**

2021-12-15

能不能加个餐～讲讲简历

展开 ∨

