

You

```
export const sideupData: IMenu[] = [
  {
    header: "Getting Started",
  },
  {
    title: "定位",
    icon: "iconfont icon-xin",
    expanded: true,
    children: [
      { title: "定位 1", href: "/test/test-position1" },
      { title: "定位 2", href: "/test/test-position2" },
      {
        href: "/test/test-position3",
        title: "定位 3",
        icon: "iconfont icon-xin",
      },
      {
        href: "/test/test-position4",
        title: "定位 4",
        icon: "iconfont icon-xin",
      },
      {
        title: "有 3 级菜单",
        children: [
          { href: "/test/test-vue-dialog", title: "全向可移动缩放对话框", icon: "iconfont icon-xin" },
          {
            href: "/test/test-xy-movable",
            title: "Basic Usage",
            icon: "iconfont icon-xin",
          },
        ],
      },
    ],
  },
  {
    href: "/test/page",
```

```
title: "Dropdown Page",
icon: "iconfont icon-xin",
expanded: true,
children: [
{
href: "/test/page/sub-page-1",
title: "Sub Page 01",
icon: "iconfont icon-xin",
},
{
href: "/test/page/sub-page-2",
title: "Sub Page 02",
icon: "iconfont icon-xin",
},
],
],
export const sideupData: IMenu[] = [
{
header: "Getting Started",
},
{
title: "定位",
icon: "iconfont icon-xin",
expanded: true,
children: [
{ title: "定位 1", href: "/test/test-position1" },
{ title: "定位 2", href: "/test/test-position2" },
{
href: "/test/test-position3",
title: "定位 3",
icon: "iconfont icon-xin",
},
{
href: "/test/test-position4",
title: "定位 4",
icon: "iconfont icon-xin",
},
{
title: "有 3 级菜单",
children: [
```

```
{ href: "/test/test-vue-dialog", title: "全向可移动缩放对话框", icon: "iconfont icon-xin" },
{
  href: "/test/test-xy-movable",
  title: "Basic Usage",
  icon: "iconfont icon-xin",
},
],
},
],
},
{
  href: "/test/page",
  title: "Dropdown Page",
  icon: "iconfont icon-xin",
  expanded: true,
  children: [
    {
      href: "/test/page/sub-page-1",
      title: "Sub Page 01",
      icon: "iconfont icon-xin",
    },
    {
      href: "/test/page/sub-page-2",
      title: "Sub Page 02",
      icon: "iconfont icon-xin",
    },
  ],
},
]
```

前序遍历上面的数组, 如何在 vue 中用非递归的方式展示数据, 并能进行鼠标点击折叠和展开显示

Genie

你可以使用组件和 `computed` 属性来实现在 Vue 中用非递归的方式展示数据, 并能进行鼠标点击折叠和展开显示。

首先, 你可以创建一个 `<Menu>` 组件来渲染菜单。在这个组件中, 你可以定义一个 `renderNode` 方法来将 `sideupData` 数组转换为一个可嵌套的树形结构。在 `renderNode` 方法中, 可以使用 `stack` 数组来模拟遍历的过程, 从根节点开始, 循环遍历每个节点直到所有子节点都被处理。对于每个节点, 你可以使用 `isopen` 状

态来表示其是否处于展开状态。

然后，在 `<Menu>` 组件中，你可以使用计算属性 `menuList`，通过调用 `renderNode` 方法并传入 `sideupData` 数组，得到整个菜单树的节点列表。你还可以提供一个 `toggle` 方法，用于切换节点的展开和折叠状态。

最后，你可以在 `<Menu>` 组件的模板中使用 `v-for` 指令来渲染 `menuList`，并根据节点的 `level` 层级和 `isExpanded` 状态来设置样式和绑定事件。当用户点击节点时，你可以调用 `toggle` 方法来改变节点的 `isExpanded` 状态。

下面是一个基本的示例代码：

```
<template>
  <div class="menu">
    <ul>
      <li
        v-for="item in menuList"
        :key="item.id"
        :class="{ 'has-children': item.children &&
item.children.length, active: item.isActive }"
        :style="{ paddingLeft: item.level * 16 + 'px', display:
item.isvisible ? 'block' : 'none' }"
      >
        <a href="#" @click.prevent="toggle(item)">{{ item.title
}}</a>
        <ul v-if="item.children && item.isExpanded">
          <li
            v-for="child in item.children"
            :key="child.id"
            :class="{ 'has-children': child.children &&
child.children.length, active: child.isActive }"
            :style="{ paddingLeft: child.level * 16 + 'px',
display: child.isvisible ? 'block' : 'none' }"
          >
            <a href="#" @click.prevent="toggle(child)">{{
child.title }}</a>
            <ul v-if="child.children && child.isExpanded">
              <li
                v-for="grandchild in child.children"
                :key="grandchild.id"
                :class="{ active: grandchild.isActive }"
                :style="{ paddingLeft: grandchild.level * 16 +
'px', display: grandchild.isvisible ? 'block' : 'none' }"
              >
```

```

        <a href="#"
@click.prevent="grandchild.clickHandler">{{ grandchild.title }}
</a>

    </li>
</ul>
</li>
</ul>
</li>
</ul>
</div>
</template>

<script>
import { sideupData } from "../data"

export default {
  computed: {
    menuList() {
      return this.renderNode(sideupData)
    },
  },
  methods: {
    renderNode(data) {
      const stack = []
      let index = 0
      let level = 1
      let parentId = null

      return data.reduce((acc, item) => {
        const id = ++index
        const node = { ...item, id, level, parentId, isVisible:
true }

        if (stack.length > 0) {
          const parent = stack[stack.length - 1]
          if (!parent.children) {
            parent.children = []
          }
          node.parentId = parent.id
          parent.children.push(node)
        }
        acc.push(node)

        if (item.children && item.children.length) {
          stack.push(node)
        }
      }, [])
    }
  }
}

```

```

        } else {
            while (stack.length > 0 && !stack[stack.length -
1].children) {
                stack.pop()
            }
        }

        return acc
    }, [])
},
toggle(node) {
    node.isExpanded = !node.isExpanded
    this.updateTree(node)
},
updateTree(node) {
    this.menuList.forEach((item) => {
        if (item.parentId === node.id) {
            item.isVisible = node.isExpanded && item.parentId ===
node.id
            this.updateTree(item)
        }
    })
},
},
}
}
</script>

<style scoped>
/* 根据需要设置样式 */
</style>

```

You

上面代码可以用 vue3.0 setup 语法糖和 ts 重写下吗?

Genie

当然可以，下面是一个用 Vue 3.0 setup 语法糖和 TypeScript 重写的示例代码：

```

<template>
  <div class="menu">
    <ul>
      <li
        v-for="item in menuList"
        :key="item.id"

```

```

        :class="{ 'has-children': item.children &&
item.children.length, active: item.isActive }"
        :style="{ paddingLeft: item.level * 16 + 'px', display:
item.isVisible ? 'block' : 'none' }"
    >
    <a href="#" @click.prevent="toggle(item)">{{ item.title
}}</a>
    <ul v-if="item.children && item.isExpanded">
        <li
            v-for="child in item.children"
            :key="child.id"
            :class="{ 'has-children': child.children &&
child.children.length, active: child.isActive }"
            :style="{ paddingLeft: child.level * 16 + 'px',
display: child.isVisible ? 'block' : 'none' }"
        >
            <a href="#" @click.prevent="toggle(child)">{{
child.title }}</a>
            <ul v-if="child.children && child.isExpanded">
                <li
                    v-for="grandchild in child.children"
                    :key="grandchild.id"
                    :class="{ active: grandchild.isActive }"
                    :style="{ paddingLeft: grandchild.level * 16 +
'px', display: grandchild.isVisible ? 'block' : 'none' }"
                >
                    <a href="#"
@click.prevent="grandchild.clickHandler">{{ grandchild.title }}
</a>
                </li>
            </ul>
        </li>
    </ul>
</div>
</template>

<script lang="ts">
import { defineComponent, reactive } from "vue"
import { sideupData, IMenu } from "../data"

export default defineComponent({
  setup() {
    const state = reactive({

```

```

    menuList: [] as IMenu[],
  })

const renderNode = (data: IMenu[]) => {
  const stack = []
  let index = 0
  let level = 1
  let parentId = null

  return data.reduce((acc, item) => {
    const id = ++index
    const node = { ...item, id, level, parentId, isVisible:
true }

    if (stack.length > 0) {
      const parent = stack[stack.length - 1]
      if (!parent.children) {
        parent.children = []
      }
      node.parentId = parent.id
      parent.children.push(node)
    }
    acc.push(node)

    if (item.children && item.children.length) {
      stack.push(node)
    } else {
      while (stack.length > 0 && !stack[stack.length -
1].children) {
        stack.pop()
      }
    }

    return acc
  }, [])
}

const toggle = (node: IMenu) => {
  node.isExpanded = !node.isExpanded
  updateTree(node)
}

const updateTree = (node: IMenu) => {
  state.menuList.forEach((item) => {
    if (item.parentId === node.id) {

```



```

        item.isVisible = node.isExpanded && item.parentId ===
node.id
        updateTree(item)
      }
    })
  }

  // 初始化
  state.menuList = renderNode(sideupData)

  return { ...state, toggle }
},
})
</script>

<style scoped>
/* 根据需要设置样式 */
</style>

```

在上面的代码中使用了 `reactive` 函数来创建响应式对象，使用 `defineComponent` 和 `setup` 语法糖来创建组件。同时，将 `renderNode`、`toggle` 和 `updateTree` 方法以及 `menuList` 状态都定义在了 `setup` 函数中，并通过 `return { ...state, toggle }` 将其暴露给模板使用。

这个示例中还使用了 TypeScript 来指定类型，需要注意的是，在 Vue 3.0 中可以直接在 `setup` 函数中使用 TypeScript，而不必再用单独的 `.ts` 文件。

You

对 `renderNode` 方法看不懂，我原来的数据是 树形结构，用 `preOrder` 进行了遍历，现在用 `renderNode` 进行树形结构，不是往复了 2 次吗？

Genie

`renderNode` 方法是将原本的树形结构转化为扁平的节点数组，可以方便地进行遍历和处理。在这个方法中使用了栈来辅助实现将树形结构转化为扁平结构。

具体实现过程如下：

1. 首先定义一个空栈 `stack` 和三个计数器 `index`、`level`、`parentId`，分别表示节点序号、层级、父节点 ID。
2. 使用 `Array.reduce()` 对原始数据进行遍历，对于每一个节点，都创建一个新对象，以及对应的节点 ID、层级和父节点 ID。
3. 如果当前节点有父节点，将其加入到父节点的 `children` 属性数组中，并更新当前节点的父节点 ID。

4. 将当前节点添加至最终的结果数组 `acc` 中。
5. 如果当前节点有子节点，将其放入栈中等待后续处理。
6. 如果当前节点没有子节点，就将栈中的元素出栈，直到找到有子节点的节点为止。
7. 返回结果数组 `acc`。

由此可见，`renderNode` 方法只进行了一次遍历，并且时间复杂度为 $O(n)$ 。不会多次进行递归和遍历，所以不会影响性能。如果你的原始数据已经是扁平的节点数组，也可以不用此方法，直接对节点数组进行操作。

You

原来 `preOrder` 是用来扁平化的，只是不知道如何在 `vue` 中进行折叠展开操作，我再试试你这个方法，
我想知道 返回的 `acc` 是什么样子的

Genie

好的，你可以尝试使用这个方法转化原始数据为扁平的节点数组。

返回的 `acc` 是一个由每个节点对象组成的数组，每个节点对象包含以下属性：

- `id`：节点 ID。
- `name`：节点名称。
- `parentId`：父节点 ID。如果该节点是根节点，则该属性值为 `null`。
- `level`：节点层级。
- `index`：节点在数组中的序号。
- `children`：子节点数组，如果该节点没有子节点，则该属性值为 `undefined`。

下面是一个示例 `acc` 数组：

```
;[
  {
    id: 1,
    name: "Node 1",
    parentId: null,
    level: 0,
    index: 0,
    children: [
      {
        id: 2,
        name: "Node 1.1",
        parentId: 1,
        level: 1,
        index: 1,
```

```

        children: [
            { id: 3, name: "Node 1.1.1", parentId: 2, level: 2,
index: 2 },
            { id: 4, name: "Node 1.1.2", parentId: 2, level: 2,
index: 3 },
            ],
        },
        { id: 5, name: "Node 1.2", parentId: 1, level: 1, index: 4
},
    ],
},
{
    id: 6,
    name: "Node 2",
    parentId: null,
    level: 0,
    index: 5,
    children: [{ id: 7, name: "Node 2.1", parentId: 6, level: 1,
index: 6 }],
},
// ...
]

```

在这个示例中，有两个根节点，第一个根节点下共有两层子节点，第二个根节点下只有一层子节点。每个节点对象中包含了节点的所有信息以及父子节点之间的关系。

你可以根据需要对这个数组进行遍历或其他操作。

You

请问这个数据结构是无限级的吗？

Genie

是的，这个数据结构可以表示无限级别的树形结构。每个节点对象中都包含了它自己的层级信息（`level` 属性）、父节点的 ID（`parentId` 属性），以及可能存在的子节点数组（`children` 属性），所以可以表示任意级数的树形结构。

```

import { nanoid } from "nanoid"
import { IMenu, data } from "../data"
import util from "util"
const renderNode = (data: IMenu[]) => {
    // 建立一个空栈，用于存放 IMenu 节点

```

```

//          id      level  parentId isVisible
const stack: IMenu[] = [] // 空栈
// 节点的层级
const level = 1
// 父节点
const parentId = null
const isVisible = true

return data.reduce((acc: IMenu[], item: IMenu) => {
  const id = nanoid(4)
  // item 是节点里本来已有的属性, id, level, parentId, isVisible 是
  // 在这里加进去的属性
  // const node: IMenu = [item, id, level, parentId, isVisible
]
  // 第1个根节点入栈
  // stack.push([item, id, level, parentId, isVisible])
  stack.push(Object.assign(item, { level, id, parentId,
isVisible }))
  if (stack.length > 0) {
    // 根节点出栈
    const node = stack.pop()

    if (node) {
      // 保存到 acc
      acc.push(node)
      // 获取该节点的子节点
      if (node.children) {
        const child = node.children
        // node.level = level + 1
        if (child) {
          for (let i = child.length - 1; i >= 0; i--) {
            const childId = nanoid(4)
            // 子节点从右到左入栈
            stack.push(Object.assign(child, { id: childId,
level: node.level! + 1, parentId: node.id, isVisible }))
          }
        }
      }
    }
  }

  return acc
}, <IMenu[]>[])
}
// renderNode(data)

```

```
console.log(util.inspect(renderNode(data)))  
// console.log(JSON.stringify(renderNode(data)))
```