link: null
title: 珠峰架构师成长计划
description: 数据分成两部分返回，文件夹列表以及文件列表,因为二者有包含关系，需要前端根据返回的数据渲染成树形结构数据。
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=55 sentences=102, words=782

实现树菜单接口

数据分成两部分返回，文件夹列表以及文件列表,因为二者有包含关系，需要前端根据返回的数据渲染成树形结构数据。

- 安装express

```
npm init -y
npm install express
```

- parent数据代表文件夹数据，child的数据代表文件

```
let express = require('express');
let app = express();
app.all('*', function (req, res, next) {
  res.header('Access-Control-Allow-Origin', '*');
  if(req.method === 'OPTIONS'){
      return res.send();
  }
  next();
});
app.get('/getTreeList',function(req,res){
  res.json({
      code:0,
      parent:[
              {name:'&#x6587;&#x4EF6;&#x5939;1',pid:0,id:1},
              {name:'&#x6587;&#x4EF6;&#x5939;2',pid:0,id:2},
              {name:'&#x6587;&#x4EF6;&#x5939;3',pid:0,id:3},
              {name:'&#x6587;&#x4EF6;&#x5939;1-1',pid:1,id:4},
              {name:'&#x6587;&#x4EF6;&#x5939;2-1',pid:2,id:5},
      ],
      child:[
              {name:'&#x6587;&#x4EF6;1',pid:1,id:10001},
              {name:'&#x6587;&#x4EF6;2',pid:1,id:10002},
              {name:'&#x6587;&#x4EF6;2-1',pid:2,id:10003},
              {name:'&#x6587;&#x4EF6;2-2',pid:2,id:10004},
              {name:'&#x6587;&#x4EF6;1-1-1',pid:4,id:10005},
              {name:'&#x6587;&#x4EF6;2-1-1',pid:5,id:10006}
      ]
  });
});
app.listen(3000);
```

引入element-ui

```
import Vue from 'vue';
// &#x5F15;&#x5165;&#x7EC4;&#x4EF6;&#x5E93;&#x884C;
import ElementUI from 'element-ui';
// &#x5F15;&#x5165;&#x7EC4;&#x4EF6;&#x5E93;&#x6837;&#x5F0F;
import 'element-ui/lib/theme-chalk/index.css';
// &#x5F15;&#x5165;App&#x6839;&#x7EC4;&#x4EF6;
import App from './App.vue';
// &#x4F7F;&#x7528;&#x63D2;&#x4EF6;
Vue.use(ElementUI);
export default new Vue({
    el:'#app',
    render:h=>h(App)
});
```

通过axios调取接口

创建api.js 到出获取列表方法

```
import axios from 'axios';
axios.defaults.baseURL = 'http://localhost:3000';
export const getTreeList = async () =>{
    return axios.get('/getTreeList');
}
```

在组件中获取数据

对elememnt-ui树组件进行二次封装，封装TreeComponent组件

```
<template>

</template>
<script>
import {getTreeList} from './api.js';
import TreeComponent from './TreeComponent.vue';
export default {
    data(){
        return {
            treeList:[]
        }
    },
    async mounted(){
        let {data} = await getTreeList();
        // 增加标示 如果是文件夹 增加type标示
        let parent = data.parent.map(item=>(item.type="child",item));
        this.treeList = [...parent,...data.child];
    },
    components:{
        TreeComponent
    }
}
</script>
```

格式化数据-转化树列表

处理数据时不能对父组件传递的数据直接更改，所以操作前需要进行数据的拷贝

- 安装lodash

```
npm install lodash
```

```
<template>
    <el-tree :data="treeList">
    </el-tree>
</template>
<script>
import _ from 'lodash'
export default {
    props:{
        // 树的信息列表
        data:{
            type:Array,
            default:()=>[]
        }
    },
    data(){
        return {
            treeList:[] // 格式化后的树的数据结构
        }
    },
    methods: {
        processData(){
            // 如果没有数据不进行任何处理
            if(this.data.length != 0){
                // 创造一个id的映射表，通过映射表创造关系,多数据操作时不要直接修改原数据
                let cloneList = _.cloneDeep(this.data);
                let mapList = cloneList.reduce((memo,current)=>{
                    memo[current['id']] = current;
                    return memo;
                },{});
                // 去列表中找 进行分类,最后返回数组结构
                this.treeList = cloneList.reduce((result,current)=>{
                    current.label = current.name;// 树的结构必须要有label属性
                    let parent = mapList[current.pid]; // 拿到当前项的父id去列表中查找，如果找到说明是儿子，就将它放到父亲的children属性中
                    if(parent){
                        parent.children? parent.children.push(current): parent.children = [current];
                    }else if(current.pid === 0){ // 说明这个是根把根放进到result中
                        result.push(current);
                    }
                    return result
                },[]);
            }
        }
    },
    watch: {
        data:{
            handler(){ // 监控data的变化，如果数据有更新重新处理数据
                this.processData();
            },
            immediate:true // 默认上来就调用一次
        }
    },
}
</script>
```

自定义tree组件

文件夹添加文件夹标识，文件添加文件标识

通过render函数扩展树组件

```
<el-tree :data="treeList" :render-content="renderFn" :expand-on-click-node="false" default-expand-all>
</el-tree>
isParent(type){
    return type === 'parent';
},
renderFn(h, { node, data, store }){
    let parent = this.isParent(data.type);
    return (<div>
        {parent?

            node.expanded?

                :
                :
        }
        {data.label}
    </div>)
},
```

扩展操作列表

列表数据应该从外部传入，对文件夹和文件实现不同的操作

```
<treecomponent :data="treeList" :rootlist="rootList" :childlist="childList"></treecomponent>
rootList:[
    {name:'rename',text:'&#x4FEE;&#x6539;&#x6587;&#x4EF6;&#x5939;&#x540D;&#x5B57;'},
    {name:'delete',text:'&#x5220;&#x9664;&#x6587;&#x4EF6;&#x5939;'}
],
childList:[
    {name:'rename',text:'&#x4FEE;&#x6539;&#x6587;&#x4EF6;&#x540D;&#x5B57;'},
    {name:'delete',text:'&#x5220;&#x9664;&#x6587;&#x4EF6;'}
]
```

实现下拉菜单

```
let list = parent? this.rootList:this.childList;
<el-dropdown placement="bottom-start" trigger="click" on-command="{this.handleCommand}">

    <el-dropdown-menu slot="dropdown">
        {list.map(item=>(
            <el-dropdown-item command="{item.name}">{item.text}</el-dropdown-item>
        ))}
    </el-dropdown-menu>
</el-dropdown>
handleCommand(name){
    if(name === 'rename'){
        // &#x4FEE;&#x6539;&#x6587;&#x4EF6;&#x540D;
    }else{
        // &#x5220;&#x9664;&#x6587;&#x4EF6;
    }
}
```

点击修改切换输入框

```
if(name === 'rename'){
    this.currentId = data.id;
    this.currentContent = data.label;
}
{this.currentId === data.id?

    :data.label
}
```

确认修改

使用.sync 同步修改后的数据

```
<el-button type="text" on-click="{this.check.bind(this,data)}">

</el-button>

check(data){
    let list = _.cloneDeep(this.data);
    list = list.map(item=>{
        if(item.id === data.id){
            item.name = this.currentContent;
        }
        return item;
    });
    this.$emit('update:data',list);
    this.currentId = '';
}
```

取消修改

```
<el-button type="text" on-click="{this.close}">

</el-button>
close(){
    this.currentId = '';
}
```

删除文件或文件夹

```
this.$confirm(`确定删除 ${data.type==='parent'?'文件夹':'文件'}吗?`,"警告",{
    confirmButtonText: '确定',
    cancelButtonText: '取消',
    type: 'warning'
}).then(()=>{
    this.handleDelete(data); // 删除文件
    this.$message({
        type:'success',
        message:'删除成功'
    })
}).catch(err=>{
    this.$message({
        type:'info',
        message:'已取消删除'
    })
});

handleDelete(data){
    let list = _.cloneDeep(this.data);
    list = list.filter(item=> item.id !== data.id);
    this.$emit('update:data',list);
    this.currentId = '';
}
```

调用接口删除文件

如果用户传递了delete方法，调用成功后在更新页面数据

```
<treecomponent :data.sync="treeList" :rootlist="rootList" :childlist="childList" :delete="deleteFn"></treecomponent>

this.delete && this.delete(data.id).then(()=>{
    this.handleDelete(data);
    this.$message({
        type:'success',
        message:'删除成功'
    })
})
```