后台管理项目

一、创建项目

```
vue init webpack mydemo
创建的2.x脚手架的项目
```

二、通过分析需求，优先下载依赖

```
npm i axios vuex element-ui stylus stylus-
loader

+ vuex@3.5.1
+ element-ui@2.13.2
+ axios@0.19.2
+ stylus-loader@3.0.2
+ stylus@0.54.8
```

三、把SQL导入到数据库（工作中不需要这个步骤）

安装失败
第一，系统设置高级=》环境变量=》path=》mysql全部删除
第二，点击安装文件=》remove 删除所有
第三，删除注册表
第四，重装，下一步下一步 ，写一下密码
第五，看看是否存在隐藏目录（c盘，mySql）

导入表结构
点击链接右键=>新建数据库=》添写数据库名，utf-8字符 =>
双击数据库名右键运行sql文件=》f5刷新就看见了空表

四、启动后端服务器（工作中不需要这个步骤）

一、修改配置文件

```
config =>global.js
exports.dbConfig = {
    host: 'localhost',  //主机
    user: 'root',       //数据库用户名
    password: '123',    //数据库密码
    port: 3306,         //默认的数据库端口
    database: 'uMallShop' // 你自己的数据库名字
}
```

二、修改app.js
37-47 注释。注释的是token内容

三、启动后端项目

```
npm start
```

四、如果想修改后端服务的端口
bin => www这个文件

```
大概15行的位置：var port = normalizePort(process.env.PORT || '3000');
默认是3000
```

五、前端开发之分析需求

分析需求得知，一二级路由

```
一级路由
login
index（承载二级）
二级路由
各个功能点
比如 菜单管理。。。


初始化项目创建路由
```

六、创建登录页

```
<template>
  <div class="login">
    <!--
          model 数据对象
          rules 验证规则
    -->
    <el-form
      :model="loginForm"
      :rules="rules"
      ref="ruleForm"
      label-width="100px"
      class="login_form"
    >
```

```
      <el-form-item label="用户名" prop="name">
        <el-input v-model="loginForm.name"
clearable></el-input>
      </el-form-item>
      <el-form-item label="密码" prop="pass">
        <el-input v-model="loginForm.pass"
clearable show-password></el-input>
      </el-form-item>
      <el-form-item>
        <el-button type="primary"
@click="login('ruleForm')">登录</el-button>
      </el-form-item>
    </el-form>
  </div>
</template>

<script>
export default {
  data() {
    return {
      loginForm: {
        name: "",
        pass: "",
      },
      rules: {
        name: [
          { required: true, message: "请输入用户
名", trigger: "blur" },
          {
            min: 2,
            max: 15,
            message: "长度在 2 到 15 个字符",
            trigger: "blur",
```

```
          },
        ],
        pass: [
          { required: true, message: "请输入密
码", trigger: "blur" },
          {
            min: 6,
            max: 18,
            message: "长度在 6 到 18 个字符",
            trigger: "blur",
          },
        ],
      },
    };
  },
  methods: {
    login(formName) {
      this.$refs[formName].validate((valid) =>
{
        if (valid) {
          //调取登录接口
          if(this.loginForm.name=='admin' &&
this.loginForm.pass=='123456'){
              //跳转首页
              this.$router.push('/index')
          }else{
              this.$message.error('用户名或者密码
错误');
          }

        } else {
          console.log("error submit!!");
          return false;
```

```
        }
      });
    },
  },
};
</script>

<style  lang="stylus" scoped>
@import '../stylus/index.styl';

.login {
  width: 100vw;
  height: 100vh;
  background: $oneBgColor;

  .login_form {
    position: absolute;
    background: $twoBgColor;
    left: 50%;
    top: 50%;
    margin: -150px 0 0 -150px;
    padding: 40px 10px;
    border-radius: 20px;
    width: 430px;
    height: 200px;

    .el-input {
      width: 80%;
    }
  }
}
</style>
```

七、Index页面搭建基本布局并拆分侧边栏

```
<template>
  <div>
    <!-- el-menu的属性
                default-active 默认选中 取决于子标签的index
                background-color    背景色
                active-text-color 点击的文本颜色
                unique-opened    是否只保持一个子菜单的展开
                router  是否使用 vue-router 的模式，启用该模式会在激活导航时以 index 作为 path 进行路由跳转
    -->
    <el-row class="tac">
      <el-col :span="24">
        <el-menu
          :default-active="defaultActive"
          background-color="#545c64"
          text-color="#fff"
          active-text-color="#ffd04b"
          unique-opened
          router
        >
          <el-menu-item index="/home">
            <i class="el-icon-s-grid"></i>
            <span slot="title">首页</span>
          </el-menu-item>
          <el-submenu index="1">
            <template slot="title">
              <i class="el-icon-s-tools"></i>
              <span>系统管理</span>
```

```
            </template>
            <el-menu-item index="/menu">菜单管理
</el-menu-item>
            <el-menu-item index="/user">管理员管
理</el-menu-item>
          </el-submenu>
          <el-submenu index="2">
            <template slot="title">
              <i class="el-icon-s-goods"></i>
              <span>商城管理</span>
            </template>
            <el-menu-item index="/goods">商品管
理</el-menu-item>
          </el-submenu>
        </el-menu>
      </el-col>
    </el-row>
  </div>
</template>

<script>
export default {
  data() {
    return {
        defaultActive:'/home'
    };
  },
  mounted(){
      this.defaultActive = this.$route.path
  }
};
</script>
```

```
<style  lang="stylus" scoped>
@import '../stylus/index.styl';

.el-menu {
  min-height: 600px;
}
</style>
```

八、拆分面包屑

router=>index.js

```
{
          path:'/menu',
          component:()=>import('@/views/menu'),
          meta:{
            name:'菜单列表'
          }
        },

meta 自定义的对象配置
```

elBread.vue

```
<template>
  <div>
    <el-breadcrumb separator="/">
      <el-breadcrumb-item :to="{ path: '/home'
}">首页</el-breadcrumb-item>
      <el-breadcrumb-item>
        <a href="#">{{$route.meta.name}}</a>
```

```
      </el-breadcrumb-item>
    </el-breadcrumb>
  </div>
</template>

<script>
export default {
  data() {
    return {};
  },
};
</script>

<style  lang="stylus" scoped>
.el-breadcrumb {
  height: 40px;
}
</style>
```

九、实现菜单管理的基本静态页骨架

```
<template>
  <div>
    <!-- 面包屑 -->
    <el-bread></el-bread>
    <!-- 添加按钮 -->
    <div>
      <el-button @click="dialogShow = true"
type="primary" size="mini" plain>添加</el-
button>
    </div>
    <!-- table表格 -->
```

```
    <el-table :data="tableData" border
style="width: 90%">
      <el-table-column prop="id" label="菜单编
号" width="180"></el-table-column>
      <el-table-column prop="title" label="菜单
名称" width="180"></el-table-column>
      <el-table-column prop="pid" label="上级菜
单"></el-table-column>
      <el-table-column prop="icon" label="菜单图
标" width="180"></el-table-column>
      <el-table-column prop="url" label="菜单地
址" width="180"></el-table-column>
      <el-table-column prop="status" label="状
态">
        <template slot-scope="item">
          <el-tag v-if="item.row.status ==1"
type="success">正常</el-tag>
          <el-tag v-else type="danger">禁用</el-
tag>
        </template>
      </el-table-column>
      <el-table-column fixed="right" label="操
作">
        <template slot-scope="scope">
          <el-button type="primary" icon="el-
icon-edit" circle></el-button>
          <el-button type="danger" icon="el-
icon-delete" circle></el-button>
        </template>
      </el-table-column>
    </el-table>
    <!-- 弹出对话框 -->
```

```
    <el-dialog title="添加菜单"
:visible.sync="dialogShow" center>
      <el-form :model="menuForm">
        <el-form-item label="菜单名称：" :label-
width="formLabelWidth">
          <el-input v-model="menuForm.title">
</el-input>
        </el-form-item>
        <el-form-item label="上级菜单：" :label-
width="formLabelWidth">
          <el-select v-model="menuForm.pid"
placeholder="请选择">
            <el-option
              v-for="item in options"
              :key="item.value"
              :label="item.label"
              :value="item.value"
            ></el-option>
          </el-select>
        </el-form-item>
        <el-form-item label="菜单类型：" :label-
width="formLabelWidth">
          <el-radio v-model="menuForm.type"
label="1">目录</el-radio>
          <el-radio v-model="menuForm.type"
label="2">菜单</el-radio>
        </el-form-item>
        <el-form-item label="菜单地址：" :label-
width="formLabelWidth">
          <el-input v-model="menuForm.url">
</el-input>
        </el-form-item>
```

```
        <el-form-item label="状态：" :label-
width="formLabelWidth">
          <el-switch v-model="menuForm.status"
active-color="#13ce66" inactive-
color="#ff4949"></el-switch>
        </el-form-item>
      </el-form>
      <div slot="footer" class="dialog-footer">
        <el-button @click="dialogShow = false">
取 消</el-button>
        <el-button @click="add" type="primary">
确 定</el-button>
      </div>
    </el-dialog>
  </div>
</template>

<script>
import elBread from "../components/elBread";
export default {
  data() {
    return {
      options: [
        {
          value: "1",
          label: "系统管理",
        },
        {
          value: "2",
          label: "商城管理",
        },
      ],
      formLabelWidth: "120px", //lable宽度
```

```
      menuForm: {
        title: "",
        pid: 0,
        type: "1", //类型1目录2菜单
        url: "",
        status: "",
      },
      dialogShow: false, //控制对话框的显示隐藏
      tableData: [
        {
          id: 1, //菜单编号
          title: "管理", //菜单名称
          pid: 112, //上级菜单
          icon: "", //菜单图标
          url: "", //菜单地址
          status: 1, //状态1正常2禁用
        },
        {
          id: 2, //菜单编号
          title: "管理", //菜单名称
          pid: 112, //上级菜单
          icon: "", //菜单图标
          url: "", //菜单地址
          status: true, //状态1正常2禁用
        },
      ],
    };
  },
  methods: {
    //添加表单事件
    add() {
      //调取接口
```

```
      this.menuForm.status =
this.menuForm.status ? 1 : 2
            console.log(this.menuForm,'添加信
息')
    },
  },
  components: {
    elBread,
  },
};
</script>

<style  lang="stylus" scoped>
.el-button {
  margin-bottom: 15px;
}
</style>
```

十、封装接口并解决跨域问题

```
    proxyTable: {
      '/api':{
        target:'http://localhost:3000',
        changeOrigin:true,
        // pathRewrite:{
        //   '^/api':'http://localhost:3000'
        // }
        //如果你的请求地址的内容与你解决跨域的地址一致，不需要重写
        //比如你的地址是'/api/menuadd' 解决跨域也用的'/api'，不需要地址重写
      }
    },
```

十一、转化数据类型用于后端接口

```
一、下拉框 后端接收数值型
初始化 pid:0
视图： :value='0'
传参的时候： this.pid =0
二、单选框 后端接收数值型
初始化 type:1
视图： :label='1'
eg:          <el-radio v-model="menuForm.type" :label="1">目录</el-radio>
          <el-radio v-model="menuForm.type" :label="2">菜单</el-radio>
传参的时候： this.type =1
三、开关 el-switch 它的默认值 是true和false
方法①
初识值 status:true
视图
```

```
<el-switch v-model="menuForm.status" active-color="#13ce66" inactive-color="#ff4949"></el-switch>
```

由于MVVM模式，数据会影响视图（后端接收的时候是1和2），取值的时候，利用json序列化转化一下（深拷贝）

逻辑代码：

```
let data = JSON.parse(JSON.stringify(this.menuForm));
          //后端接收的时候要1,或者2 el-switch要是true或者false
          data.status = data.status ? 1 : 2;
```

方法②

active-value　switch 打开时的值　不管传入什么类型最终都是true

inactive-value　switch 关闭时的值　不管传入什么类型最终都是false

```
 <el-switch
            v-model="menuForm.status"
            active-color="#13ce66"
            inactive-color="#ff4949"
            :active-value="1"
            :inactive-value="2"
          ></el-switch>
```

初识值：status：1

逻辑代码：　不需要转化

十二、清除表单域以及关闭弹框

```
//关闭弹框事件
    //封装一个清空事件
    reset() {
      //方法一
      this.menuForm = {
```

```
        title: "",
        pid: 0,
        type: 1, //类型1目录2菜单
        url: "",
        //  status: true,
        status: 1,
        icon: "",
      };
      this.dialogShow = false;
      //方法二
      // 表单域 model 字段，在使用 validate、
resetFields 方法的情况下，prop属性是必填的
     // this.$refs['ruleForm'].resetFields();
    },
```

十三、拆分组件

以后菜单管理为案例，拆分成三个组件，add.vue，list.vue，index.vue

修改路由component的地址 eg：`component: ()=>import('@/views/menu/index.vue'),`

index.vue

```
<template>
  <div>
    <!-- 面包屑 -->
    <el-bread></el-bread>
    <!-- 添加按钮 -->
    <div>
      <el-button @click="openDialog"
type="primary" size="mini" plain>添加</el-
button>
```

```
    </div>
    <!-- 表格渲染 -->
    <v-list @edit="edit"></v-list>
    <!-- 弹出对话框 -->
    <v-dialog ref="vDialog" :isShow="sonStatus"
@closeDialog="closeDialog"></v-dialog>
  </div>
</template>

<script>
import elBread from "../../components/elBread";
import vList from "./list";
import vDialog from "./add";

export default {
  data() {
    return {
      sonStatus: {
        isAdd: true, //是否是添加
        dialogShow: false, //控制对话框的显示隐藏
      },
    };
  },

  methods: {
    //封装打开弹框事件(新增按钮)
    openDialog() {
      this.sonStatus.isAdd = true;
      this.sonStatus.dialogShow = true;
    },
    //关闭弹框事件
    closeDialog(e) {
      this.sonStatus.dialogShow = e;
```

```
    },
    //编辑事件
    edit(e) {
      this.sonStatus.isAdd = e.isAdd;
      this.sonStatus.dialogShow = true;
      this.$refs.vDialog.update(e.id);
    },
  },
  components: {
    elBread,
    vList,
    vDialog,
  },
};
</script>

<style  lang="stylus" scoped>
.el-button {
  margin-bottom: 15px;
}
</style>
```

list.vue

```
<template>
  <div>
    <el-table
      :data="get_MenuList"
      border
      row-key="id"
      default-expand-all
      :tree-props="{children: 'children'}"
```

```
    >
      <el-table-column prop="id" label="菜单编
号" width="180"></el-table-column>
      <el-table-column prop="title" label="菜单
名称" width="180"></el-table-column>
      <el-table-column prop="pid" label="上级菜
单"></el-table-column>
      <el-table-column prop="icon" label="菜单图
标" width="180"></el-table-column>
      <el-table-column prop="url" label="菜单地
址" width="180"></el-table-column>
      <el-table-column prop="status" label="状
态">
        <template slot-scope="item">
          <el-tag v-if="item.row.status ==1"
type="success">正常</el-tag>
          <el-tag v-else type="danger">禁用</el-
tag>
        </template>
      </el-table-column>
      <el-table-column fixed="right" label="操
作">
        <template slot-scope="item">
          <el-button type="primary"
@click="edit(item.row.id)" icon="el-icon-edit"
circle></el-button>
          <el-button type="danger"
@click="del(item.row.id)" icon="el-icon-delete"
circle></el-button>
        </template>
      </el-table-column>
    </el-table>
  </div>
```

```
</template>

<script>
import { mapGetters, mapActions } from "vuex";
import { getMenuList, getMenuDel } from
"../../util/axios";
export default {
  data() {
    return {};
  },
  mounted() {
    //页面一加载就获取菜单列表
    this.getMenuListAction();
  },
  computed: {
    ...mapGetters(["get_MenuList"]),
  },
  methods: {
    //封装获取菜单列表
    ...mapActions(["getMenuListAction"]),
    //获取当前数据的id
    edit(id) {
      this.$emit("edit", {
        isAdd: false,
        id,
      });
    },
    //删除事件
    del(id) {
      this.$confirm("确定要删除该数据吗", "提示", {
        confirmButtonText: "确定",
        cancelButtonText: "取消",
        type: "warning",
```

```
      })
        .then(() => {
          //调取删除接口
          getMenuDel({ id }).then((res) => {
            if (res.code === 200) {
              this.$message({
                type: "success",
                message: res.msg,
              });
              //重新调取列表接口
              this.getMenuListAction();
            } else {
              this.$message.error(res.msg);
            }
          });
        })
        .catch(() => {
          this.$message({
            type: "info",
            message: "已取消删除",
          });
        });
    },
  },
};
</script>

<style  lang="" scoped>
</style>
```

add.vue

```
<template>
  <div>
    <el-dialog
      :before-close="reset"
      :title="isShow.isAdd?'添加菜单':'编辑菜单'"
      :visible.sync="isShow.dialogShow"
      center
    >
      <el-form :model="menuForm" :rules="rules"
ref="ruleForm">
        <el-form-item prop="title" label="菜单名
称：" :label-width="formLabelWidth">
          <el-input v-model="menuForm.title">
</el-input>
        </el-form-item>
        <el-form-item prop="pid" label="上级菜
单：" :label-width="formLabelWidth">
          <el-select v-model="menuForm.pid"
placeholder="请选择">
            <el-option value disabled>--请选择--
</el-option>
            <el-option label="顶级菜单"
:value="0">顶级菜单</el-option>
            <el-option
              v-for="item in get_MenuList"
              :key="item.id"
              :label="item.title"
              :value="item.id"
            ></el-option>
          </el-select>
        </el-form-item>
        <el-form-item label="菜单类型：" :label-
width="formLabelWidth">
```

```
          <el-radio v-model="menuForm.type"
:label="1">目录</el-radio>
          <el-radio v-model="menuForm.type"
:label="2">菜单</el-radio>
        </el-form-item>
        <el-form-item
          v-if="menuForm.type==1"
          prop="icon"
          label="菜单图标："
          :label-width="formLabelWidth"
        >
          <el-input v-model="menuForm.icon">
</el-input>
        </el-form-item>
        <el-form-item v-if="menuForm.type==2"
label="菜单地址：" :label-width="formLabelWidth">
          <el-input v-model="menuForm.url">
</el-input>
        </el-form-item>
        <el-form-item label="状态：" :label-
width="formLabelWidth">
          <el-switch
            v-model="menuForm.status"
            active-color="#13ce66"
            inactive-color="#ff4949"
            :active-value="1"
            :inactive-value="2"
          ></el-switch>
        </el-form-item>
      </el-form>
      <div slot="footer" class="dialog-footer">
        <el-button @click="reset">取 消</el-
button>
```

```
        <el-button @click="add('ruleForm')"
type="primary">确 定</el-button>
      </div>
    </el-dialog>
  </div>
</template>

<script>
import { mapActions,mapGetters } from "vuex";
import { getMenuAdd, getMenuInfo, getMenuEdit }
from "../../util/axios";
export default {
  props: ["isShow"],
  computed: {
      ...mapGetters(['get_MenuList'])
  },
  data() {
    return {
      editId: 0,
      formLabelWidth: "120px", //lable宽度
      menuForm: {
        title: "",
        pid: 0,
        type: 1, //类型1目录2菜单
        url: "",
        //  status: true,
        status: 1,
        icon: "",
      },
      //表单验证（根据产品需求制定）
      rules: {
        title: [
          //代表加红色星标
```

```
          { required: true, message: "请输入菜单
名称", trigger: "blur" },
          //验证字符数
          { min: 2, max: 9, message: "长度在 3
到 9 个字符", trigger: "blur" },
        ],
        pid: [
          //代表加红色星标
          { required: true, message: "请选择上级
菜单", trigger: "change" },
        ],
      },
    };
  },
  methods: {
    //编辑事件
    update(id) {
      //赋值 给调取编辑接口用
      this.editId = id;
      //更改isAdd状态
      this.isAdd = false;
      getMenuInfo({ id }).then((res) => {
        if (res.code == 200) {
          this.menuForm = res.list;
        }
      });
    },
    //调取行动
    ...mapActions(["getMenuListAction"]),
    //关闭弹框事件
    //封装一个清空事件
    reset() {
      //方法一
```

```
        this.menuForm = {
          title: "",
          pid: 0,
          type: 1, //类型1目录2菜单
          url: "",
          //  status: true,
          status: 1,
          icon: "",
        };
        //子组件关闭弹框要去修改父组件dialogShow这个数
据
        this.$emit("closeDialog", false);
      },
      //添加表单事件
      add(formName) {
        this.$refs[formName].validate((valid) =>
{
          if (valid) {
            //调取接口
            //是否调取新增事件还是编辑事件
            if (this.isShow.isAdd) {
              //调取添加接口

 getMenuAdd(this.menuForm).then((res) => {
                if (res.code === 200) {
                  this.$message.success(res.msg);
                  //关闭弹框并清空
                  this.reset();
                  //重新调取列表
                  this.getMenuListAction();
                }
              });
            } else {
```

```
            //id编号，必填项  对数据进行编辑
            this.menuForm.id = this.editId;
            //调取添加接口

 getMenuEdit(this.menuForm).then((res) => {
              if (res.code === 200) {
                this.$message.success(res.msg);
                //关闭弹框并清空
                this.reset();
                //重新调取列表
                this.getMenuListAction();
              }
            });
          }
        } else {
          console.log("error submit!!");
          return false;
        }
      });
    },
  },
};
</script>

<style  lang="" scoped>
</style>
```

作业：

一、整理笔记

二、案例（一定先综合写再拆分）

三、周末小U商城项目移动端（所有页面包括登录注册静态页）

四、有兴趣的同学康康角色列表

回顾上周

路由的基本概念 路由导航守卫

6个， 全局（前置导航钩子 beforeEach((to,from,next)=>{}) 后置afterEach(()=>{to,from}))

路由独享 beforeEnter(to,from,next)=>{}

组件内部 beforeRouteEnter(to,from,next)=>{} beforeRouteUpdate(to,from,next)=>{}

beforeRouteLeave(to,from,next)=>{}

路由懒加载

component:()=>import(组件地址')

Vuex

state

getters

mutations 同步 是修改state的唯一方式

actions 异步 不能直接修改state 而是commit mutation

moudles 模块 防止state过于臃肿

RBAC（权限）Role-Based-Access-Control

侧边栏是动态控制的

```
eg: 后台管理（公司内部使用）
who ： 超管（超集管理员） 它拥有一切权限
      普通管理员（一些权限，必须查看统计数量，商品管理，会员一些管理，它没有分配权限的权利）
      普通用户（商品管理，对商品的上架下架，维护商品）

超管 可以看到所有功能点
管理用 可以看除了 菜单，角色之外的功能点
普通用户 只能看到商城管理的所有功能点
```

侧边栏是写死

（一共有6个模块）

没有菜单的设置，没有角色的控制，所有的账号，是Java提供。当你登录的时候，返回来的内容，本身就存在一个菜单数组，包含了有哪些功能点。可以通过v-if去显示隐藏菜单，还有组件或者独享守卫

十四、创建模块的步骤

- 创建组件
- 设置路由
- 搭建静态骨架
- 调取接口渲染动态数据
- 提取出共通的接口封装到vuex中

十五、实现角色列表的增删改查

搭建静态骨架

```
<!--                树形结构
            data      展示数据      array
            node-key      每个树节点用来作为唯一标识的属性，整棵树应该是唯一的   String
            props     配置选项，具体看下表   object
            default-expand-all   是否默认展开所有节点   boolean
          show-checkbox 节点是否可被选择
boolean

          getCheckedKeys      若节点可被选择（即 show-checkbox 为 true），则返回目前被选中的节点的 key 所组成的数组
setCheckedKeys   通过 keys 设置目前勾选的节点，使用此方法必须设置 node-key 属性
default-checked-keys      默认勾选的节点的 key 的数组
          -->
          <el-tree
            :data="get_MenuList"
            show-checkbox
            default-expand-all
            node-key="id"
            ref="tree"
            :props="defaultProps"
            :default-checked-keys="defaultChecked"
          ></el-tree>
```

动态渲染

- 角色列表封装到vuex
- 根据后端对于参数格式的需求，做了二次封装

```
传值
//菜单权限 存放的是菜单编号，用逗号隔开
'1,2,3,4,27'
this.roleForm.menus =
this.$refs.tree.getCheckedKeys().join(",");

//取值
//得到的数组要做二次转换
this.defaultChecked = this.roleForm.menus
            ? this.roleForm.menus.split(",")
            : [];
```

十六、分页器

视图=>list.vue

```
      <!--
        分页视图
        background  是否为分页按钮添加背景色
        total    总条目数
        page-size    每页显示条目个数，支持 .sync 修饰符
        current-change  currentPage 改变时会触发 可以获取点击页码
      -->
  <el-pagination
        background
        layout="prev, pager, next"
        :total="total"
        :page-size="pageInfo.size"
```

```
      @current-change="getChange"
    ></el-pagination>
```

分页逻辑=>list.vue

```
  data() {
    return {
      total: 0, //总条数
      pageInfo: {
        //分页的信息 你自己定义
        size: 2, //查询条数
        page: 1, //页码数
      },
    };
  },

  methods: {
      //当页面发生变化的时候会调用当前方法
    getChange(n){
        //获取页码并复制给后端要求的参数
      this.pageInfo.page = n

      //重新调取列表
      this.getUserListAction(this.pageInfo)
    },
     //获取总数
      getCount() {
      getUserCount().then((res) => {
        if (res.code === 200) {
          console.log(res, "总条数");
          this.total = res.list[0].total;
            //重新调取列表
```

```
this.getUserListAction(this.pageInfo)
          }
        });
      },
    }


  /* size 当天显示条数    page 页码

  总数 18条数据   每页显示10
  页码2

  第一次调取接口   size：10
                  page:1
  后端返回前10条

  点击第二页
    size：10
    page：2
  后端再返回十条，如果不够 都返回了 返回8条 */
```

管理员添加

逻辑=》add.vue

//调取添加接口
//刷新列表并刷新总条数
实现方式两种：
第一种 直接调取总条数接口
第二种 封装到vuex（不建议）
第三种 传值
list =>index.vue(在index.vue利用ref获取getCount)
index.vue => add.vue(父传子 赋值给sonStatus)
在add.vue就能获取 this.isShow,它是一个对象
添加成功之后就调用

十七、文件上传格式

post提交方式如果是普通(非上传)的表单提交，我们发送给服务器的格式
表单数据会编码为 "application/x-www-form-urlencoded"
但凡你有上传文件，包括文件，图片。。。这个时候application/x-www-form-urlencoded，服务器无法解析，必须把enctype 转化成multipart/form-data

application/x-www-form-urlencoded 在发送前编码所有字符（默认）
multipart/form-data
不对字符编码。
在使用包含文件上传控件的表单时，必须使用该值。
text/plain 空格转换为 "+" 加号，但不对特殊字符编码。

原生js提交文件上传的时候利用发方式,它可以new 实例，去改变我们上传格式编码，改成multipart/form-data

```
new FormData()

首先有一个表单对象{key1:value1,key2:value2}，如果包含上传文件不能这么提交

let obj = {key1:value1,key2:value2}
let file = new FormData()
for(let i  in obj){
    file.append(i,obj[i])
}

直接提交file
```

调用el-upload

```
  <!-- 图片上传
          action 如有直接的上传服务器，把服务器地址写这，如果走接口，直接写#
          on-preview on-preview 点击文件列表中已上传的文件时的钩子
          on-remove 文件列表移除文件时的钩子
          auto-upload    是否在选取文件后立即进行上传（默认上传element官网）
          limit 最大允许上传个数
          on-exceed文件超出个数限制时的钩子
function(files, fileList)
          file-list 上传的文件列表, 例如: [{name: 'food.jpg', url: 'https://xxx.cdn.com/xxx.jpg'}]
          on-change 文件状态改变时的钩子，添加文件、上传成功和上传失败时都会被调用
          -->
```

```
          <el-upload
            action="#"
            list-type="picture-card"
            :on-preview="handlePreview"
            :on-remove="handleRemove"
            :auto-upload="false"
            :limit="1"
            :on-exceed="handleExceed"
            :file-list='fileList'
            :on-change = 'changeInfo'
          >
            <i class="el-icon-plus"></i>
          </el-upload>
          <el-dialog
:visible.sync="dialogVisible">
            <img width="100%"
:src="dialogImageUrl" alt />
          </el-dialog>
```

十八、动态添减input

```
视图
<el-form-item
          :label-width="formLabelWidth"
          v-for="(item, index) in specsArr"
          label="规格属性:"
          :key="index"
        >
          <el-input style="width:70%" v-
model="item.value"></el-input>
          <el-button v-if="index==0"
@click="addSpecs(item)" type="primary">新增规格属
性</el-button>
```

```
          <el-button v-else
@click="delSpecs(item)">删除</el-button>
        </el-form-item>

data(){
    return{
     specsArr: [
        {
          value: "",
        },
      ], //规格属性
    }
}
methods：{
    //添加规格属性
    addSpecs(item) {
      //判断
      if (this.specsArr.length <= 6) {
        this.specsArr.push({
          value: "",
        });
      }else{
        this.$message.warning('最多只能添加6个')
      }
    },
    //删除规格属性
    delSpecs(item) {
      let index = this.specsArr.indexOf(item);
      if (index !== -1) {
        this.specsArr.splice(index, 1);
      }
    },
}
```