
黑马程序员™
www.itheima.com
传智播客旗下
高端IT教育品牌
Vue全家桶-项目实战（下）

目录 Contents

1. 用户管理

1.1 用户管理概述

通过后台管理用户的账号信息，具体包括用户信息的展示、添加、修改、删除、角色分配、账号启用/注销等功能。

- 用户信息列表展示
- 添加用户
- 修改用户
- 删除用户
- 启用或禁用用户
- 用户角色分配

1. 用户管理

1.2 用户信息列表展示

1. 用户列表布局

- 面包屑导航 el-breadcrumb
- Element-UI 栅格系统基本使用 el-row
- 表格布局 el-table、el-pagination

1.2 用户信息列表展示

2. 用户状态列和操作列处理

作用域插槽

```
<template slot-scope="scope">
  <!-- 开关 -->
  <el-switch v-model="scope.row.mg_state"
     @change="stateChanged(scope.row.id, scope.row.mg_state)">
  </el-switch>
</template>
```

1. 用户管理

1.2 用户信息列表展示

3. 表格数据填充

- 调用后台接口
- 表格数据初填充

```
const { data: res } = await this.$http.get('users', { params: this.queryInfo })
if (res.meta.status !== 200) {
  return this.$message.error('查询用户列表失败！')
}
this.total = res.data.total
this.userlist = res.data.users
```

1.2 用户信息列表展示

4. 表格数据分页

分页组件用法：

① 当前页码：pagenum
② 每页条数：pagesize
③ 记录总数：total
④ 页码变化事件
⑤ 每页条数变化事件
⑥ 分页条菜单控制

```
<el-pagination
    @size-change="handleSizeChange"
    @current-change="handleCurrentChange"
    :current-page="queryInfo.pagenum"
    :page-sizes="[2, 3, 5, 10]"
    :page-size="queryInfo.pagesize"
    layout="total, sizes, prev, pager, next, jumper"
    :total="total">
  </el-pagination>
```

1.2 用户信息列表展示

5. 搜索功能

将搜索关键字，作为参数添加到列表查询的参数中。

```
<el-input
    placeholder="请输入搜索的内容"
    v-model="queryInfo.query"
    clearable
    @clear="getUserList">
    <el-button slot="append"
      icon="el-icon-search"
      @click="getUserList"></el-button>
  </el-input>
```

1.3 用户状态控制

1. 开关组件的用法
2. 接口调用更改用户的状态

```
<el-switch
  v-model="scope.row.mg_state"
  @change="stateChanged(scope.row.id, scope.row.mg_state)">
</el-switch>
```

```
async stateChanged(id, newState) {
  const { data: res } = await this.$http.put(`users/${id}/state/${newState}`)
  if (res.meta.status !== 200) {
    return this.$message.error('修改状态失败！')
  }
}
```

1. 用户管理

1.4 添加用户

1. 添加用户表单弹窗布局

- 弹窗组件用法
- 控制弹窗显示和隐藏

```
<el-dialog title="添加用户" :visible.sync="addDialogVisible" width="50%">
   <el-form :model="addForm" label-width="70px">
    <el-form-item label="用户名" prop="username">
      <el-input v-model="addForm.username"></el-input>
    </el-form-item>
    <!-- 更多表单项 -->
  </el-form>
  <span slot="footer" class="dialog-footer">
    <el-button @click="resetAddForm">取 消</el-button>
    <el-button type="primary" @click="addUser">确 定</el-button>
  </span>
</el-dialog>
```

1. 用户管理

1.4 添加用户

2. 表单验证

内置表单验证规则

```
<el-form :model="addForm" :rules="addFormRules" ref="addFormRef" >
  <!-- 表单 -->
</el-form>
```

```
addFormRules: {
  username: [{ required: true, message: '请输入用户名', trigger: 'blur' }],
  password: [{ required: true, message: '请输入密码', trigger: 'blur' }],
}
```

```
this.$refs.addFormRef.validate(async valid => {
  if (!valid) return
})
```

1.4 添加用户

2. 表单验证

自定义表单验证规则

```
const checkMobile = (rule, value, cb) => {
  let reg = /^(0|86|17951)?(13[0-9]|15[012356789]|17[678]|18[0-9]|14[57])[0-9]{8}$/
  if (reg.test(value)) {
    cb()
  } else {
    cb(new Error('手机号码格式不正确'))
  }
}
```

```
mobile: [
  { required: true, message: '请输入手机号', trigger: 'blur' },
  { validator: checkMobile, trigger: 'blur' }
]
```

1.4 添加用户

3. 表单提交

将用户信息作为参数，并调用后台接口添加用户。

```
this.$refs.addFormRef.validate(async valid => {
  if (!valid) return
  const { data: res } = await this.$http.post('users', this.addForm)
  if (res.meta.status !== 201) {
    return this.$message.error('添加用户失败！')
  }
  this.$message.success('添加用户成功！')
  this.addDialogVisible = false
  this.getUserList()
})
```

1. 用户管理

1.5 修改用户

1. 根据 ID 查询用户信息

```
<el-button type="primary" size="mini" icon="el-icon-edit"
            @click="showEditDialog(scope.row.id)"></el-button>
```

```
async showEditDialog(id) {
  const { data: res } = await this.$http.get('users/' + id)
  if (res.meta.status !== 200) {
    return this.$message.error('查询用户信息失败！')
  }
  // 把获取到的用户信息对象，保存到 编辑表单数据对象中
  this.editForm = res.data
  this.editDialogVisible = true
}
```

1. 用户管理

1.5 修改用户

2. 编辑提交表单

```
this.$refs.editFormRef.validate(async valid => {
  if (!valid) return
  // 发起修改的请求
  const { data: res } = await this.$http.put('users/' + this.editForm.id, {
    email: this.editForm.email,
    mobile: this.editForm.mobile
  })
  if (res.meta.status !== 200) {
    return this.$message.error('编辑用户信息失败！')
  }
  this.$message.success('编辑用户信息成功！')
  this.getUserList()
  this.editDialogVisible = false
})
```

1. 用户管理

1.6 删除用户

```
<el-button type="danger" size="mini" icon="el-icon-delete"
           @click="remove(scope.row.id)"></el-button>
```

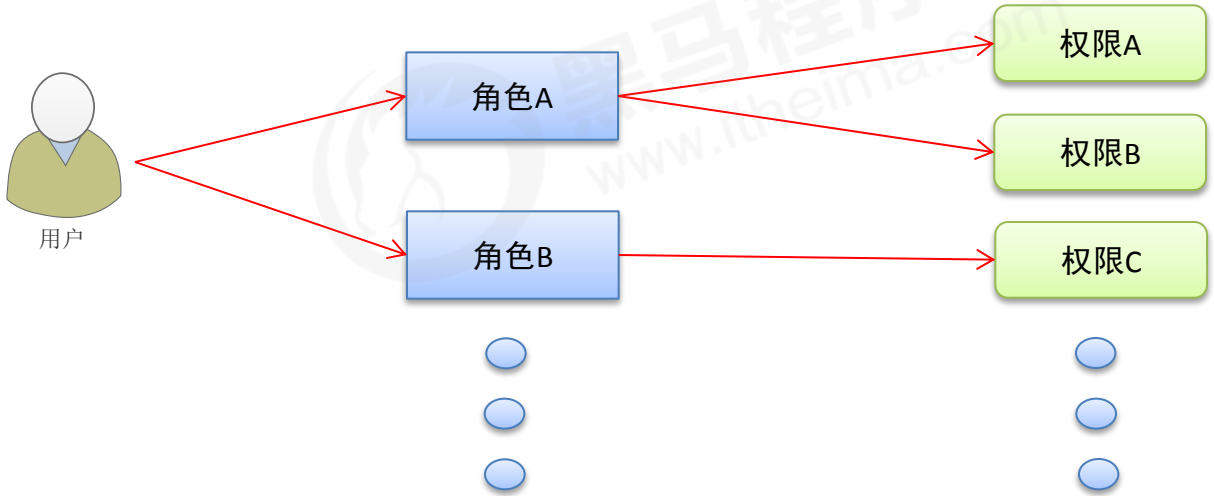
```
async remove(id) {
 // 询问是否要删除
 const confirmResult = await this.$confirm('此操作将永久删除该用户，是否继续?', '提示', {
   confirmButtonText: '确定',
   cancelButtonText: '取消',
   type: 'warning'
 }).catch(err => err)
 const { data: res } = await this.$http.delete('users/' + id)
 if (res.meta.status !== 200) return this.$message.error('删除用户失败！')
 this.$message.success('删除用户成功！')
 this.getUserList()
},
```

目录 Contents

2. 权限管理

2.1 权限管理业务分析

通过权限管理模块控制不同的用户可以进行哪些操作，具体可以通过角色的方式进行控制，即每个用户分配一个特定的角色，角色包括不同的功能权限。



2. 权限管理

2.2 权限列表展示

```
<template slot-scope="scope">
  <el-tag size="small" v-if="scope.row.level == 0">一级</el-tag>
  <el-tag type="success" size="small" v-else-if="scope.row.level == 1">二级</el-tag>
  <el-tag type="warning" size="small" v-else>三级</el-tag>
</template>
```

```
// 获取权限列表数据
async getRightsList() {
  const { data: res } = await this.$http.get('rights/list')
  if (res.meta.status !== 200) {
    return this.$message.error('获取权限列表失败！')
  }
  this.rightsList = res.data
}
```

2.3 角色列表展示

- 调用后台接口获取角色列表数据
- 角色列表展示

```
// 获取所有角色列表
async getRolesList() {
  const { data: res } = await this.$http.get('roles')
  if (res.meta.status !== 200) {
    return this.$message.error('获取角色列表失败！')
  }
  this.rolesList = res.data
},
```

2.4 用户角色分配

1. 展示角色对话框

① 实现用户角色对话框布局
② 控制角色对话框显示和隐藏
③ 角色对话框显示时，加载角色列表数据

```
async showSetRoleDialog(userInfo) {
  this.userInfo = userInfo
  // 发起请求，获取所有角色的列表
  const { data: res } = await this.$http.get('roles')
  if (res.meta.status !== 200) {
    return this.$message.error('获取角色列表失败！')
  }
  this.rolesList = res.data
  this.setRoleDialogVidible = true
}
```

2.4 用户角色分配

2. 完成角色分配功能

```
async saveNewRole() {
  if (this.selectedRoleId === '') {
    return this.$message.error('请选择新角色后再保存！')
  }
  const { data: res } = await this.$http.put(`users/${this.userInfo.id}/role`, {
    rid: this.selectedRoleId
  })
  if (res.meta.status !== 200) {
    return this.$message.error('分配角色失败！')
  }
  this.$message.success('分配角色成功！')
  this.getUserList()
  this.setRoleDialogVidible = false
}
```

2.5 角色权限分配

1. 表格行展开效果

通过 el-table-column 组件的 type = “expand” 方式实现表格行展开效果

```
<el-table :data="rolesList" border stripe>
  <!-- 展开行的列 -->
  <el-table-column type="expand">
    <template slot-scope="scope">
      <!-- 展开行内容填充 -->
    </template>
  </el-table-column>
</el-table>
```

2.5 角色权限分配

2. 渲染一级权限菜单

在表格展开行中渲染一级菜单

```
<el-row v-for="(item1, i1) in scope.row.children" :key="item1.id" class="centerRow">
  <!-- 这一列，专门渲染 一级权限 -->
  <el-col :span="5">
    <el-tag closable>{{item1.authName}}</el-tag>
    <i class="el-icon-caret-right"></i>
  </el-col>
  <!-- 还剩余 19 列，分配给二三级权限 -->
  <el-col :span="19">
    <!-- 这里显示二三级权限 -->
  </el-col>
</el-row>
```

2.5 角色权限分配

3. 渲染二、三级权限菜单

在表格展开行中渲染二、三级菜单

```
<el-row v-for="(item2, i2) in item1.children" :key="item2.id" class="centerRow">
  <!-- 放二级权限 -->
  <el-col :span="6">
    <el-tag closable type="success">{{item2.authName}}</el-tag>
    <i class="el-icon-caret-right"></i>
  </el-col>
  <!-- 放三级权限 -->
  <el-col :span="18">
    <el-tag closable type="warning" v-for="item3 in item2.children" :key="item3.id">
      {{item3.authName}}</el-tag>
  </el-col>
</el-row>
```

2.5 角色权限分配

4. 删除角色下的权限

点击权限菜单的删除按钮后，调用后台接口删除对应权限和其下的子权限。

```
<el-col :span="5">
  <el-tag closable @close="removeRight(scope.row, item1.id)">
    {{item1.authName}}
  </el-tag>
  <i class="el-icon-caret-right"></i>
</el-col>
```

```
const { data: res } = await this.$http.delete(`roles/${role.id}/rights/${rightId}`)
if (res.meta.status !== 200) {
  return this.$message.error('删除权限失败！')
}
this.$message.success('删除权限成功！')
```

2.5 角色权限分配

5. 给角色分配权限流程

① 实现角色分配权限对话框布局
② 控制对话框的显示和隐藏
③ 对话框显示时调用后台接口加载权限列表数据
④ 完成树形权限菜单的展示
⑤ 选中默认的权限
⑥ 保存选中的权限，调用后台接口完成角色权限的分配

2.5 角色权限分配

6. 实现权限分配对话框布局

- 实现对话框布局效果
- 控制对话框显示和隐藏

```
  <!-- 分配权限的对话框 -->
  <el-dialog title="分配权限" :visible.sync="setRightDialogVisible" width="50%"
@close="resetSetRightDialog">
    <!-- 权限菜单 -->
    <span slot="footer" class="dialog-footer">
      <el-button @click="setRightDialogVisible = false">取 消</el-button>
      <el-button type="primary" @click="saveRight">确 定</el-button>
    </span>
  </el-dialog>
```

2.5 角色权限分配

7. 渲染权限的树形结构

- 树形组件 el-tree 的基本使用
- 权限数据的加载与填充

```
<el-tree ref="tree" :data="rightTree" :props="treeConfig" show-checkbox node-key="id" default-expand-all :default-checked-keys="defaultCheckedKeys"></el-tree>
```

```
// 在展示对话框之前，先获取到权限的树形结构数据
const { data: res } = await this.$http.get('rights/tree')
if (res.meta.status !== 200) return this.$message.error('初始化权限失败！')
// 把权限的树形结构数据，保存到data中，供页面渲染使用
this.rightTree = res.data
```

2.5 角色权限分配

8. 设置默认权限菜单选中

- 获取所有叶子节点的 id
- 设置权限节点选中

```
// 根据指定的节点和keys数组，递归获取所有三级节点的Id
getLeafIds(node, keys) {
  if (!node.children) {
    keys.push(node.id)
  } else {
    node.children.forEach(item => this.getLeafIds(item, keys))
  }
}
```

```
const keys = [] // 专门存放所有三级节点的Id
this.getLeafIds(role, keys)
this.defaultCheckedKeys = keys
```

2.5 角色权限分配

9. 完成角色授权

- 获取所有选中的权限节点 id
- 调用接口完成角色权限的分配

```
// 获取树形控件中，所有半选和全选节点的Id数组
const arr1 = this.$refs.tree.getCheckedKeys()
const arr2 = this.$refs.tree.getHalfCheckedKeys()
const rids = [...arr1, ...arr2].join(',')
```

```
const { data: res } = await this.$http.post(`roles/${this.selectedRoleId}/rights`, { rids })
if (res.meta.status !== 200) {
  return this.$message.error('分配权限失败！')
}
this.$message.success('分配权限成功！')
```

目录 Contents

3. 分类管理

3.1 商品分类概述

商品分类用于在购物时，快速找到所要购买的商品，可以通过电商平台主页直观的看到。

3.2 商品分类列表

基本布局与数据获取

- 实现基本布局
- 实现分类列表数据加载

```
const { data: res } = await this.$http.get('categories', { params: this.queryInfo })
if (res.meta.status !== 200) {
  return this.$message.error('获取商品分类失败！')
}
this.cateList = res.data.result
this.total = res.data.total
```

3. 分类管理

3.3 树形表格

1. 第三方树形表格的基本使用

- 安装依赖包（地址：https://github.com/MisterTaki/vue-table-with-tree-grid）

```
npm i vue-table-with-tree-grid -S
```

- 基本使用

```
import Vue from 'vue'
import ZkTable from 'vue-table-with-tree-grid'
Vue.use(ZkTable)
```

3. 分类管理

3.3 树形表格

2. 实现分类树形列表

- 实现树形列表布局并进行数据填充
- 自定义表格列

```
  <tree-table :data="cateList" :columns="columns" border :selection-type="false"
:expand-type="false" show-index index-text="#" class="tree-table">
    <!-- 操作的模板列 -->
    <!-- 排序的模板列 -->
    <!-- 是否有效的模板列 -->
    <template slot="isok" slot-scope="scope">
      <i class="el-icon-success" style="color:#20B2AA;" v-if="scope.row.cat_deleted
=== false"></i>
      <i class="el-icon-error" style="color:#F92672;" v-else></i>
    </template>
  </tree-table>
```

3. 分类管理

3.4 分页功能

- 实现分页组件效果
- 分页组件数据处理

```
<!-- 分页区域 -->
<el-pagination
  @current-change="handleCurrentChange"
  :current-page="queryInfo.pagenum"
  :page-size="queryInfo.pagesize"
  layout="total, prev, pager, next, jumper"
  :total="total">
</el-pagination>
```

3. 分类管理

3.5 添加分类

1. 实现分类树形列表

- 实现添加分类对话框布局
- 控制对话框显示和隐藏

```
<el-dialog title="添加分类" :visible.sync="addDialogVisible" width="50%" @close="resetForm">
  <el-form :model="addForm" :rules="addFormRules" ref="addFormRef" label-width="100px">
    <el-form-item label="分类名称：" prop="cat_name">
      <el-input v-model="addForm.cat_name"></el-input>
    </el-form-item>
    <el-form-item label="父级分类：">
      <!-- 分类菜单 -->
    </el-form-item>
  </el-form>
</el-dialog>
```

3. 分类管理

3.5 添加分类

2. 实现分类级联菜单效果

- 实现级联菜单效果
- 级联菜单数据加载与填充

```
<el-cascader
    expand-trigger="hover"
    :options="parentCateList"
    :props="cascaderConfig"
    v-model="selectedCateList"
    @change="handleChange"
    change-on-select
    clearable>
 </el-cascader>
```

```
// 先获取所有父级分类的数据列表
const { data: res } = await this.$http.get('categories', { params: { type: 2 } })
if (res.meta.status !== 200) return this.$message.error('获取父级分类失败！')
// 把父级分类数据，挂载到data中
this.parentCateList = res.data
```

3.5 添加分类

3. 控制父级分类的选择

父级分类选择时，获取对应的分类 id。

```
handleChange() {
  if (this.selectedCateList.length === 0) {
    // 证明没有选中任何父级分类
    this.addForm.cat_pid = 0
    this.addForm.cat_level = 0
  } else {
    // 选中父级分类
    this.addForm.cat_pid = this.selectedCateList[this.selectedCateList.length - 1]
    // 设置分类等级
    this.addForm.cat_level = this.selectedCateList.length
  }
}
```

3.5 添加分类

4. 完成分类添加

将分类名称、分类等级和父分类 id 提交到后台，完成分类添加。

```
const { data: res } = await this.$http.post('categories', this.addForm)
if (res.meta.status !== 201) {
  return this.$message.error('添加分类失败！')
}
this.$message.success('添加分类成功！')
```

目录 Contents

4. 参数管理

4.1 参数管理概述

商品参数用于显示商品的固定的特征信息，可以通过电商平台商品详情页面直观的看到。

动态参数

静态属性

4.2 商品分类选择

1. 选择商品分类

- 页面基本布局
- 加载商品分类数据
- 实现商品分类的级联选择效果

```
// 获取所有商品的分类列表
async getAllCateList() {
  const { data: res } = await this.$http.get('categories')
  if (res.meta.status !== 200) {
    return this.$message.error('获取商品分类列表失败！')
  }
  this.cateList = res.data
}
```

4. 参数管理

4.2 商品分类选择

2. 控制级联菜单分类选择

- 只允许选择三级分类
- 通过计算属性的方式获取分类 ID

```
cascaderChanged() {
  if (this.selectedCateList.length !== 3) {
    // 没有选中三级分类，把分类重置为空
    this.selectedCateList = []
    this.manyTableData = []
    this.onlyTableData = []
  } else {
    // 选中了三级分类后，获取该分类对应的参数列表数据
    this.getParamsList()
  }
}
```

```
cateId() {
  if (this.selectedCateList.length === 3) {
    return this.selectedCateList[this.selectedCateList.length - 1]
  } else {
    return null
  }
}
```

4.3 实现参数列表

1. 根据选择的商品分类加载对应的参数数据

- 参数列表布局
- 根据分类 id 加载参数列表数据

```
// 获取所有商品的分类列表
const { data: res } = await this.$http.get(`categories/${this.cateId}/attributes`, {
  params: { sel: this.activeName }
})
```

4.3 实现参数列表

2. 处理标签数据格式

将字符串形式的数据分隔为数组。

```
res.data.forEach(item => {
  // 把字符串的可选项，分割为数组，重新赋值给 attr_vals
  item.attr_vals = item.attr_vals.length > 0 ? item.attr_vals.split(',') : []
})
```

4.3 实现参数列表

3. 控制添加标签文本框的显示

$nextTick 的执行时机： **DOM 更新完毕之后**

```
<el-button size="small" v-else @click="showTagInput(scope.row)">+ New Tag</el-button>
```

```
showTagInput(row) {
  row.tagInputVisible = true
  // 当我们修改了 data 中 tagInputVisible 的值以后，如果要操作文本框，必须等页面重新渲染完毕之后才可以，所以，必须把操作文本框的代码放到 $nextTick 中，当作回调去执行（$nextTick 的执行时机，是在 DOM 更新完毕之后）
  this.$nextTick(() => {
    this.$refs.saveTagInput.$refs.input.focus()
  })
}
```

4.3 实现参数列表

4. 实现标签动态添加的文本框控制逻辑

- 控制标签输入域的显示和隐藏
- 对输入的内容进行数据绑定

```
res.data.forEach(item => {
  // 把字符串的可选项，分割为数组，重新赋值给 attr_vals
  item.attr_vals = item.attr_vals.length > 0 ? item.attr_vals.split(',') : []
  // 为每个数据行，添加自己的 tagInputVisible ，从而控制自己展开行中的输入框的显示与隐藏
  item.tagInputVisible = false
  // 把文本框中输入的值，双向绑定到 item.tagInputValue 上
  item.tagInputValue = ''
})
```

4.3 实现参数列表

5. 实现标签的添加和删除操作

添加标签和删除标签使用的是同一个接口，参数是一样的。

```
const { data: res } = await this.$http.put(
  `categories/${this.cateId}/attributes/${row.attr_id}`,
  {
    attr_name: row.attr_name,
    attr_sel: row.attr_sel,
    attr_vals: row.attr_vals.join(' ')
  }
)
if (res.meta.status !== 200) {
  return this.$message.error('更新参数项失败！')
}
this.$message.success('更新参数项成功！')
```

4.4 实现动态参数与静态属性添加

- **动态参数与静态属性表单重用**
- 添加动态参数与静态属性使用的是同一个接口，参数是一样的

```
const { data: res } = await this.$http.post(`categories/${this.cateId}/attributes`, {
  // 参数的名称
  attr_name: this.addForm.attr_name,
  // 参数类型    many  only
  attr_sel: this.activeName
})
if (res.meta.status !== 201) return this.$message.error('添加参数失败！')
this.$message.success('添加参数成功！')
```

目录 Contents

5. 商品管理

5.1 商品管理概述

商品管理模块用于维护电商平台的商品信息，包括商品的类型、参数、图片、详情等信息。通过商品管理模块可以实现商品的添加、修改、展示和删除等功能。

5. 商品管理

5.1 商品列表

- 实现商品列表布局效果
- 调用后台接口获取商品列表数据

```
const { data: res } = await this.$http.get('goods', { params: this.queryInfo })
if (res.meta.status !== 200) {
  return this.$message.error('初始化商品列表失败！')
}
// 为商品列表赋值
this.goodsList = res.data.goods
// 为总数量赋值
this.total = res.data.total
```

5.2 添加商品

1. 基本布局与分布条效果

- 添加商品基本布局
- 分布条组件用法

```
<el-steps :active="activeName-0" finish-status="success" align-center>
  <el-step title="基本信息"></el-step>
  <el-step title="商品参数"></el-step>
  <el-step title="商品属性"></el-step>
  <el-step title="商品图片"></el-step>
  <el-step title="商品内容"></el-step>
  <el-step title="完成"></el-step>
</el-steps>
```

5.2 添加商品

2. 商品信息选项卡Tab布局效果

Tab 组件的基本使用

```
<el-tabs tab-position="left" v-model="activeName" :before-leave="beforeTabLeave">
   <el-tab-pane label="基本信息" name="0"><!-- 基本信息面板 --></el-tab-pane>
   <el-tab-pane label="商品参数" name="1"><!-- 商品参数面板 --></el-tab-pane>
   <el-tab-pane label="商品属性" name="2"><!-- 商品静态属性面板 --></el-tab-pane>
   <el-tab-pane label="商品图片" name="3"><!-- 图片上传面板 --></el-tab-pane>
   <el-tab-pane label="商品内容" name="4"><!-- 商品描述面板 --></el-tab-pane>
 </el-tabs>
```

5.2 添加商品

3. 商品基本信息

- 商品基本信息表单布局
- 表单数据绑定
- 表单验证

```
addFormRules: {
  goods_name: [{ required: true, message: '请填写商品名称', trigger: 'blur' }],
  goods_price: [{ required: true, message: '请填写商品价格', trigger: 'blur' }],
  goods_weight: [{ required: true, message: '请填写商品重量', trigger: 'blur' }],
  goods_number: [{ required: true, message: '请填写商品数量', trigger: 'blur' }],
  goods_cat: [{ required: true, message: '请选择商品分类', trigger: 'blur' }]
}
```

5.2 添加商品

4. 商品分类信息

- 商品分类布局
- 商品分类数据加载

```
<el-cascader expand-trigger="hover" :options="cateList" :props="cascaderConfig"
v-model="addForm.goods_cat" @change="handleCascaderChange"></el-cascader>
```

```
const { data: res } = await this.$http.get('categories')
if (res.meta.status !== 200) {
  return this.$message.error('初始化商品分类失败！')
}
this.cateList = res.data
```

5.2 添加商品

5. 商品动态参数

- 获取商品动态参数数据
- 商品动态参数布局

```
const { data: res } = await this.$http.get(`categories/${this.cateId}/attributes`, {
  params: { sel: 'many' }
})
if (res.meta.status !== 200) return this.$message.error('获取动态参数列表失败！')
// 把动态参数中的每一项数据中的 attr_vals，都从字符串分割为数组
res.data.forEach(item => {
  item.attr_vals = item.attr_vals.length === 0 ? [] : item.attr_vals.split(' ')
})
this.manyData = res.data
```

5.2 添加商品

6. 商品静态属性

- 获取商品静态属性数据
- 商品静态属性布局

```
const { data: res } = await this.$http.get(`categories/${this.cateId}/attributes`, {
  params: { sel: 'only' }
})
if (res.meta.status !== 200) {
  return this.$message.error('获取动态参数列表失败！')
}
this.onlyData = res.data
```

5.2 添加商品

7. 商品图片上传

图片上传组件基本使用

```
<el-upload
  action="http://47.96.21.88:8888/api/private/v1/upload"
  :headers="uploadHeaders"
  :on-preview="handlePreview"
  :on-remove="handleRemove"
  :on-success="handleSuccess"
  list-type="picture">
    <el-button size="small" type="primary">点击上传</el-button>
</el-upload>
```

5.2 添加商品

7. 商品图片上传

图片预览

```
// 预览图片时候，触发的方法
handlePreview(result) {
  this.previewImgSrc = result.response.data.url
  this.previewVisible = true
}
```

5.2 添加商品

7. 商品图片上传

图片删除

```
  // 当移除图片，会触发这个方法
  handleRemove(result) {
    // 根据 result.response.data.temp_path 从 addForm.pics 数组中，找到要删除那个对象的索引值
    const index = this.addForm.pics.findIndex(item => item.pic === result.response.data.tmp_path)
    // 根据索引删除对应的图片信息对象
    this.addForm.pics.splice(index, 1)
  }
```

5.2 添加商品

7. 商品图片上传

完成图片上传

```
// 图片上传成功
handleSuccess(result) {
  if (result.meta.status === 200) {
    // 把上传成功后，图片的临时路径，保存到 addForm.pics 数组中，作为对象来保存
    this.addForm.pics.push({
      pic: result.data.tmp_path
    })
  }
}
```

5. 商品管理

5.2 添加商品

8. 商品详情

富文本编辑器基本使用

```
// 安装vue-quill-editor
npm install vue-quill-editor -S
```

```
import VueQuillEditor from 'vue-quill-editor'
Vue.use(VueQuillEditor)
```

```
<quill-editor v-model="addForm.goods_introduce"></quill-editor>
```

5.2 添加商品

9. 完成商品添加

- 处理商品相关数据格式
- 调用接口完成商品添加

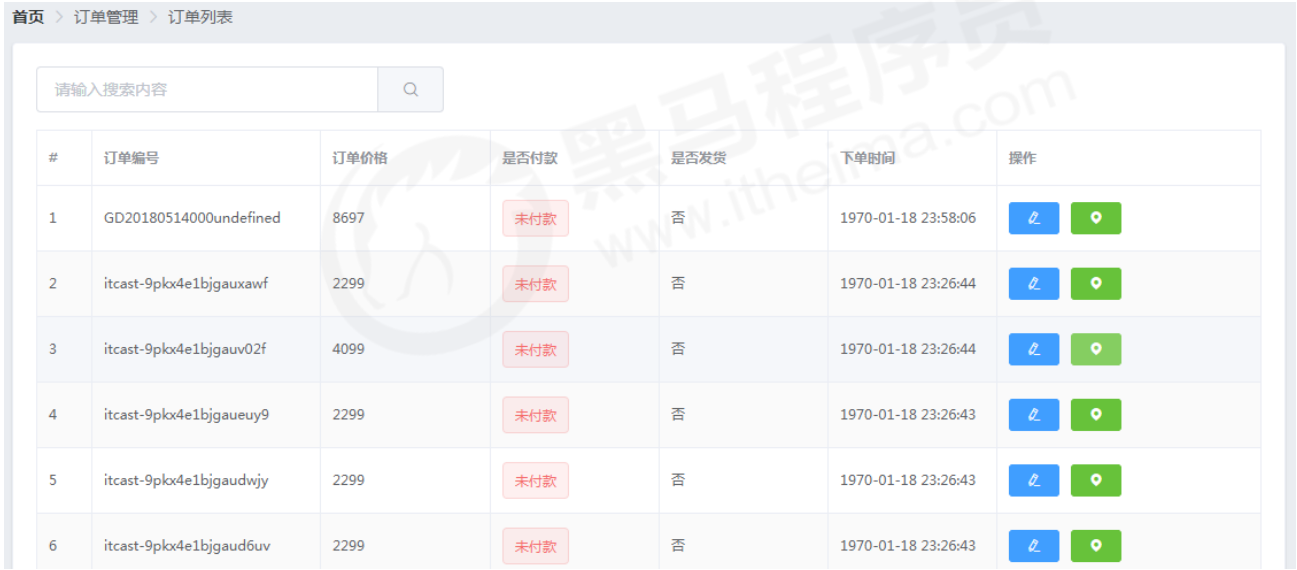
```
// 先处理好商品相关的数据格式，然后再提交
const newForm = _.cloneDeep(this.addForm)
newForm.goods_cat = newForm.goods_cat.join(',')
// 到此位置，商品相关数据已经准备好，可以提交了
const { data: res } = await this.$http.post('goods', newForm)
if (res.meta.status !== 201) return this.$message.error(res.meta.msg)
this.$message.success('添加商品成功！')
// 跳转到商品列表页
this.$router.push('/goods/list')
```

目录 Contents

6. 订单管理

6.1 订单管理概述

订单管理模块用于维护商品的订单信息，可以查看订单的商品信息、物流信息，并且可以根据实际的运营情况对订单做适当的调整。



6. 订单管理

6.2 订单列表

1. 订单列表展示

- 订单数据加载
- 订单列表布局

```
const { data: res } = await this.$http.get('orders', { params: this.queryInfo })
if (res.meta.status !== 200) {
  return this.$message.error('获取订单列表失败！')
}
this.orderList = res.data.goods
this.total = res.data.total
```

6.2 订单列表

2. 查看订单地址信息

- 省市区三级联动效果
- 省市区数据格式分析

```
<el-cascader
  :options="cityOptions"
  v-model="selectedArea"
  @change="changeProvince"
  change-on-select
  style="width: 100%;">
</el-cascader>
```

6. 订单管理

6.1 订单列表

3. 查看订单物流信息

- 调用接口获取物流数据
- 实现物流信息列表效果

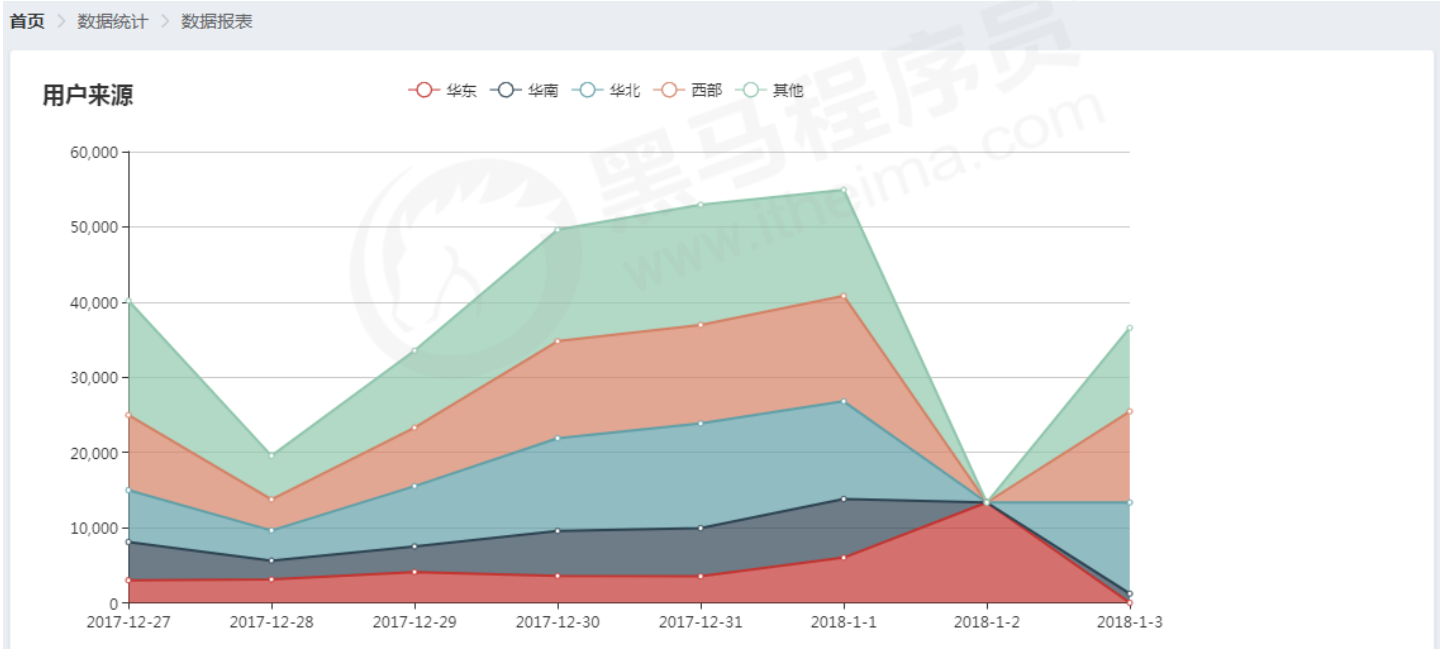
```
const { data: res } = await this.$http.get('/kuaidi/110121212622')
if (res.meta.status !== 200) {
  return this.$message.error('获取物流进度失败！')
}
this.wlList = res.data
```

目录 Contents

7. 数据统计

7.1 数据统计概述

数据统计模块主要用于统计电商平台运营过程的中的各种统计数据，并通过直观的可视化方式展示出来，方便相关运营和管理人员查看。



7.2 用户来源数据统计报表

1. Echarts 第三方可视化库的基本使用

```
// 安装echarts库
npm install echarts -S
```

```
// 导入echarts接口
import echarts from 'echarts'
```

7.2 用户来源数据统计报表

2. 实现用户来源数据统计报表

① 调用接口获取后台接口数据

② 通过echarts的api实现报表效果

```
// 基于准备好的dom，初始化echarts实例
var myChart = echarts.init(this.$refs.main)
const { data: res } = await this.$http.get('reports/type/1')
if (res.meta.status !== 200) return this.$message.error('初始化折线图失败！')
const data = _.merge(res.data, this.options)
// 绘制图表
myChart.setOption(data)
```

传智播客旗下高端IT教育品牌