

会员管理系統实战-前后端分离

第一章 项目需求分析和技术架构

梦学谷会员管理系统:<http://vue.mengxuegu.com/>

项目相关源码位于：01-配套源码\02-VueProject-项目阶段源码.zip

1.1 项目介绍与效果图



序号	会员卡号	会员姓名	会员生日	手机号码	可用积分	开卡金额	支付类型	会员地址	操作
1	000001	小梦	1987-01-01	18888888888	98	300	支付宝	北京天安门广场1号正大门	<button>编辑</button> <button>删除</button>
2	000002	小梦2	1987-01-01	18888888888	98	300	微信	北京天安门广场1号正大门	<button>编辑</button> <button>删除</button>
3	000003	小梦3	1987-01-01	18888888888	98	300	现金	北京天安门广场1号正大门	<button>编辑</button> <button>删除</button>
4	000004	小梦4	1987-01-01	18888888888	98	300	支付宝	北京天安门广场1号正大门	<button>编辑</button> <button>删除</button>
5	000005	小梦2	1987-01-01	18888888888	98	300	微信	北京天安门广场1号正大门	<button>编辑</button> <button>删除</button>

1.2 项目需求

会员管理系统采用数据化管理会员、管理商品进销、供应商信息维护、员工管理等加快对店铺运营效率。

项目涉及功能模块：会员管理、供应商管理、商品管理、员工管理。

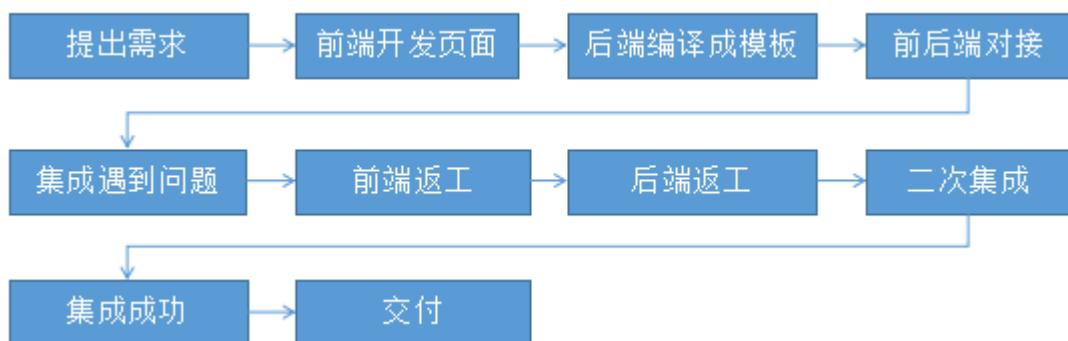
1.3 什么是前后端分离开发

传统系统架构：

1. 前端工程师负责编写HTML页面，完成前端页面设计。
2. 后端工程师使用模板技术将HTML页面代码转换为JSP页面，同时内嵌后端代码(如Java)；

前后端强依赖，后端必须要等前端的HTML开发好才能套转换成JSP。如果需求变更，前端HTML要改，后端JSP也要跟着变，这是件紧紧牵绊的事，使得开发效率降低。

3. 产品交付时，要将前后端代码全部进行打包，部署到同一服务器上，或者进行简单的动静态分离部署。



前后端分离架构：

1. 前后端约定好API接口&数据&参数

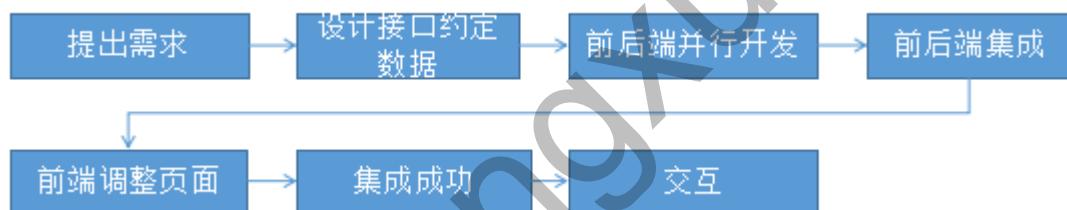
2. 前后端并行开发

前端工程师只需要编写HTML页面，通过HTTP请求调用后端提供的接口服务即可。

后端只要愉快的开发接口就行了。

无强依赖，如果需求变更，只要接口和参数不变，就不用两边都修改代码，开发效率高。

3. 除了开发阶段分离，在运行期前后端资源也会进行分离部署。



前后端分离已成为互联网项目开发的业界标准使用方式。传统的前后端混合开发模式，虽然久经考验，到现在依然还是能够支撑起应用的开发。但是放眼未来，社会分工更加精细化，前后端分离开发的精细化也是必然趋势。并且前后端分离会为以后的大型分布式架构、微服务架构、多端化服务（多种客户端，例如：浏览器，安卓，iOS，车载终端等等）打下坚实的基础。

1.4 项目前端技术架构

Vue.js 综合项目技术架构



第二章 分析 RESTful 架构风格

Restful 架构: 通过HTTP请求方式(GET/POST/PUT/DELETE...)来区分对资源CRUD操作 ,

请求资源 URI 是 /资源名称/资源标识

2.1 接口定义方式

对比传统方式与 Restful 方式定义接口 :

	传统	Restful
列表查询	getMemberList	member/list --- GET
查询	getMember?id=xxx	member/{id} --- GET
添加	addMember?xxx	member --- POST
修改	updateMember?id=xxx	member/{id} --- PUT
删除	deleteMember?id=xxx	member/{id} --- DELETE

2.2 项目使用Restful架构风格

项目功能	请求URI	请求方式
查询所有会员	member/list	GET
查询某位会员信息	member/{id}	GET
添加会员	member	POST
前往修改页面（查出会员信息进行信息回显）	member/{id}	GET
修改会员	member/{id}	PUT
删除会员	member/{id}	DELETE

第三章 Mock.js 数据生成器

3.1 解决什么问题

问题：

前后端分离项目，前端和后端人员都是根据 API 文档进行开发项目的，不应该直接相互依赖，前端人员不应该等待后端开发好接口后再进行测试，既然不依赖后端接口，那前端人员应该如何测试呢？

解决：

可以通过模拟数据生成器，通过一定规则（API 文档）生成模拟数据接口，提供给前端人员进行测试。

3.2 什么是 Mock.js

- 官网：<http://mockjs.com/>
- 文档：<https://github.com/nuysoft/Mock/wiki>
- Mock.js 是用于生成随机数据，拦截 Ajax 请求。

通过拦截 Ajax 请求，根据数据模板生成并返回模拟数据，让前端攻城师独立于后端进行开发，帮助编写单元测试。

- Mock.js 作用：

1. 前后端分离

让前端攻城师独立于后端进行开发。

2. 增加单元测试的真实性

通过随机数据，模拟各种场景。

3. 开发无侵入

不需要修改既有代码，就可以拦截 Ajax 请求，返回模拟的响应数据。

4. 用法简单

符合直觉的接口。

5. 数据类型丰富

支持生成随机的文本、数字、布尔值、日期、邮箱、链接、图片、颜色等。

6. 方便扩展

支持扩展更多数据类型，支持自定义函数和正则。

3.3 安装 Mock.js

在命令提示符窗口，用 npm 安装 mockjs

```
1 npm install mockjs
```

3.4 入门案例与语法规范

3.4.1 入门案例

需求：生成 4 条列表数据。

效果如下：

```
1 { memberList:  
2   [ { id: 1, name: '小梦' },  
3     { id: 1, name: '小梦' },  
4     { id: 1, name: '小梦' },  
5     { id: 1, name: '小梦' }]  
6 }
```

编码实现：

1. 新建 StudentProject\VueProject\mockjs-demo 目录，通过命令提示符窗口进入到该文件夹，执行下面命令进行初始化项目

```
1 npm init -y
```

2. Mock.js 安装

```
1 npm install mockjs
```

3. 新建 demo1.js 代码如下

```
1 const Mock = require('mockjs')  
2
```

```
3 const data = Mock.mock({
4   'memberList|4': [
5     {
6       'id': 1,
7       'name': '小梦'
8     }
9   ]
10 })
11 // stringify(数据, 数据转换函数, 缩进空格数)
12 console.log(JSON.stringify(data, null, 2))
13
14 //查看效果, 执行命令 node demo1.js
```

查看效果，执行命令

```
1 node demo1.js
```

```
1 **上面生成的4条数据是一样的, 如果希望它按一定规律生成随机数据, 就需要按 Mock.js 的语法规规范来定义。
**
```

3.4.2 语法规则

- Mock.js 的语法规规范包括两部分：
 1. 数据模板定义规范 (Data Template Definition , DTD)
 2. 数据占位符定义规范 (Data Placeholder Definition , DPD)

3.5. 数据模板定义规范 DTD

数据模板中的每个属性由 3 部分构成：属性名、生成规则、属性值：

```
1 // 属性名 | 生成规则 : 属性值
2 'name|rule': value
```

注意：

- 属性名和生成规则之间用竖线 | 分隔。
- 生成规则是可选的，生成规则有 7 种格式：
 1. 'name|min-max': value
 2. 'name|count': value
 3. 'name|min-max.dmin-dmax': value
 4. 'name|min-max.dcount': value
 5. 'name|count.dmin-dmax': value
 6. 'name|count.dcount': value
 7. 'name|+step': value

- 生成规则的含义需要依赖属性值的类型才能确定。



- 属性值中可以含有 @占位符。
- 属性值指定了最终值的初始值和类型。

3.5.1 属性值是字符串 String

1. 'name|count': string

通过重复 string 生成一个字符串，重复次数等于 count。

2. 'name|min-max': string

通过重复 string 生成一个字符串，重复次数大于等于 min，小于等于 max。

- 代码：

```
1 const data = Mock.mock({
2   'memberList|4': [
3     {
4       'id': 1,
5       'name|1-3': '小梦', // String, 随机生成 1到3个重复`小梦`
6       'phone|11': '8' // String, 生成 11 个 8
7     }
8   ]
9 })
```

- 效果：

```
1 {
2   "memberList": [
3     {
4       "id": 1,
5       "name": "小梦小梦",
6       "phone": "88888888888"
7     },
8     {
9       "id": 1,
10      "name": "小梦小梦",
11      "phone": "88888888888"
12    },
13    {
14      "id": 1,
15      "name": "小梦小梦小梦",
16      "phone": "88888888888"
17    },
18    {
19      "id": 1,
20      "name": "小梦小梦小梦",
21      "phone": "88888888888"
22    }
23  ]
24 }
```



3.5.2 属性值是数字 Number

1. 'name|+1': number

属性值自动加 1，初始值为 number。

2. 'name|min-max': number

生成一个大于等于 min、小于等于 max 的整数，属性值 number 只是用来确定类型。

3. 'name|min-max.dmin-dmax': number

生成一个浮点数，整数部分大于等于 min、小于等于 max，小数部分保留 dmin 到 dmax 位。

- 代码：

```
1 const data = Mock.mock({
2   'memberList|4': [
3     {
4       'id|+1': 1, // Number, 自增 1
5       'name|1-3': '小梦', // String, 随机生成 1到3个重复`小梦`
6       'phone|11': '8', // String, 生成 11 个 8
7       'age|1-120': 1, // Number, 随机生成 1到120
8       'salary|6000-8000.1-3': 0, // Number, 随机生成6000到8000, 有随机小数1到3位
9     }
10   ]
11 })
```

- 效果：

```
1 {
2   "memberList": [
3     {
4       "id": 1,
5       "name": "小梦",
6       "phone": "88888888888",
7       "age": 110,
8       "salary": 7617.958
9     },
10    {
11      "id": 2,
12      "name": "小梦小梦",
13      "phone": "88888888888",
14      "age": 53,
15      "salary": 6413.055
16    },
17    {
18      "id": 3,
19      "name": "小梦小梦",
20      "phone": "88888888888",
21      "age": 101,
22      "salary": 6733.44
23    },
24    {
25      "id": 4,
```

```
26     "name": "小梦小梦小梦",
27     "phone": "888888888888",
28     "age": 115,
29     "salary": 7462.9
30   }
31 ]
32 }
```

3.5.3 属性值是布尔型 Boolean

1. 'name|1': boolean

随机生成一个布尔值，值为 true 的概率是 1/2，值为 false 的概率同样是 1/2。

2. 'name|min-max': value

随机生成一个布尔值，值为 value 的概率是 min / (min + max)，值为 !value 的概率是 max / (min + max)。

- 代码：

```
1 const data = Mock.mock({
2   'memberList|4': [
3     {
4       'id|+1': 1, // Number, 自增 1
5       'name|1-3': '小梦', // String, 随机生成 1到3个重复`小梦`
6       'phone|11': '8', // String, 生成 11 个 8
7       'age|1-120': 1, // Number, 随机生成 1到120
8       'salary|6000-8000.1-3': 0, // Number, 随机生成6000到8000, 有随机小数1到3位
9       'status|1': true, // Boolean, 生成 true 或 false 概率都是 1/2
10      'open|2-4': true, // Boolean, 生成 true 概率 2/(2+4), false 概率 4/(2+4)
11    },
12  ],
13 })
```

- 效果：

```
1 {
2   "memberList": [
3     {
4       "id": 1,
5       "name": "小梦",
6       "phone": "888888888888",
7       "age": 64,
8       "salary": 6047.1,
9       "status": true,
10      "open": false
11    },
12    {
13      "id": 2,
14      "name": "小梦小梦",
15      "phone": "888888888888",
```

```
16     "age": 89,
17     "salary": 7191.87,
18     "status": true,
19     "open": false
20   },
21   ..
22   ..
23 ]
24 }
```

3.5.4 属性值是对象 Object

1. 'name|count': object

从属性值 `object` 中随机选取 `count` 个属性。

2. 'name|min-max': object

从属性值 `object` 中随机选取 `min` 到 `max` 个属性。

- 代码：

```
1 const data = Mock.mock({
2   'memberList|4': [
3     {
4       'id|+1': 1, // Number, 自增 1
5       'name|1-3': '小梦', // String, 随机生成 1到3个重复`小梦`
6       'phone|11': '8', // String, 生成 11 个 8
7       'age|1-120': 1, // Number, 随机生成 1到120
8       'salary|6000-8000.1-3': 0, // Number, 随机生成6000到8000, 有随机小数1到3位
9       'status|1': true, // Boolean, 生成 true 或 false 概率都是 1/2
10      'open|2-4': true, // Boolean, 生成 true 概率 2/(2+4), false 概率 4/(2+4)
11      'order|2': { id: 1, name: '订单1', price: 68.8 }, // 随机取对象中的2个属性
12      'order2|2-3': { id: 1, name: '洗发水', price: 68.8 }, // 对象中的2到3个属性
13    }
14  ]
15 })
```

- 效果：

```
1 {
2   "memberList": [
3     {
4       "id": 1,
5       "name": "小梦小梦小梦",
6       "phone": "88888888888",
7       "age": 91,
8       "salary": 7579.7,
9       "status": true,
10      "open": false,
11      "order": {
```



```
12      "id": 1,
13      "price": 68.8
14    },
15    "order2": {
16      "name": "洗发水",
17      "price": 68.8,
18      "id": 1
19    }
20  },
21  {
22    "id": 2,
23    "name": "小梦小梦",
24    "phone": "888888888888",
25    "age": 23,
26    "salary": 6377.72,
27    "status": true,
28    "open": false,
29    "order": {
30      "price": 68.8,
31      "id": 1
32    },
33    "order2": {
34      "id": 1,
35      "price": 68.8
36    }
37  },
38  ..
39  ..
40
41 ]
42 }
```

3.5.5 属性值是数组 Array

1. 'name|min-max': array

通过重复属性值 `array` 生成一个新数组，重复次数大于等于 `min`，小于等于 `max`。

2. 'name|count': array

通过重复属性值 `array` 生成一个新数组，重复次数为 `count`。

• 代码：

```
1 const data = Mock.mock({
2   'memberList|2-5': [ // Array, 随机生成数组中的元素 2到5 个
3     {
4       'id|+1': 1,
5       'name|1-3': '小梦',
6       'phone|11': '8',
7       'age|1-120': 1,
8       'salary|6000-8000.1-3': 0,
```



```
9      'status|1': true,
10     'open|2-4': true,
11     'order|2': { id: 1, name: '订单1', price: 68.8 },
12     'order2|2-3': { id: 1, name: '订单2', price: 68.8 },
13   }
14 ]
15 })
16
```

- 效果：随机生成 2 到 5 个元素

```
1  {
2    "memberList": [
3      {
4        "id": 1,
5        "name": "小梦小梦",
6        "phone": "88888888888",
7        "age": 78,
8        "salary": 7864.371,
9        "status": false,
10       "open": true,
11       "order": {
12         "price": 68.8,
13         "id": 1
14       },
15       "order2": {
16         "name": "订单2",
17         "price": 68.8
18       }
19     },
20     {
21       "id": 2,
22       "name": "小梦小梦",
23       "phone": "88888888888",
24       "age": 36,
25       "salary": 6113.168,
26       "status": false,
27       "open": true,
28       "order": {
29         "price": 68.8,
30         "name": "订单1"
31       },
32       "order2": {
33         "id": 1,
34         "price": 68.8,
35         "name": "订单2"
36       }
37     }
38   ]
39 }
```

3.5.6 值是正则表达式 RegExp

1. 'name': regexp

根据正则表达式 `regexp` 反向生成可以匹配它的字符串。用于生成自定义格式的字符串。

注意 `regexp` 是没有引号的

- 代码：

```
1 const data = Mock.mock({
2   'memberList|2-5': [ // Array, 随机生成数组中的元素 2到5 个
3     {
4       'id|+1': 1, // Number, 自增 1
5       'name|1-3': '小梦', // String, 随机生成 1到3个重复`小梦`
6       'phone|11': '8', // String, 生成 11 个 8
7       'age|1-120': 1, // Number, 随机生成 1到120
8       'salary|6000-8000.1-3': 0, // Number, 随机生成6000到8000, 有随机小数1到3位
9       'status|1': true, // Boolean, 生成 true 或 false 概率都是 1/2
10      'open|2-4': true, // Boolean, 生成 true 概率 2/(2+4), false 概率 4/(2+4)
11      'order|2': { id: 1, name: '订单1', price: 68.8 }, // 随机取对象中的2个属性
12      'order2|2-3': { id: 1, name: '订单2', price: 68.8 }, // 对象中的2到3个属性
13      'idCard': /\d{15}|\d{18}/ // 随机生成身份证号, 注意:正则表达式没有单引号 "
14    }
15  ]
16 })
```

- 效果：

```
1 {
2   "memberList": [
3     {
4       "id": 1,
5       "name": "小梦小梦",
6       "phone": "88888888888",
7       "age": 47,
8       "salary": 6202.85,
9       "status": false,
10      "open": false,
11      "order": {
12        "name": "订单1",
13        "id": 1
14      },
15      "order2": {
16        "id": 1,
17        "price": 68.8
18      },
19      "idCard": "583332714813595261"
20    },
21    {
22      "id": 2,
23      "name": "小梦",
24      "phone": "88888888888",
```

```
25     "age": 2,
26     "salary": 6790.81,
27     "status": true,
28     "open": false,
29     "order": {
30       "id": 1,
31       "price": 68.8
32     },
33     "order2": {
34       "name": "洗发水",
35       "id": 1,
36       "price": 68.8
37     },
38     "idCard": "353627221161244853"
39   }
40 ]
41 }
```

3.6. 数据占位符定义规范 DPD

Mock.Random 是一个工具类，用于生成各种随机数据。

Mock.Random 类中的方法在数据模板中称为『占位符』，书写格式为 @占位符(参数 [, 参数])。

占位符的格式为：

```
1 '@属性名': @占位符
```

Mock.Random 类中提供的完整方法（占位符）如下：

Type (类型)	Method (占位符)
Basic	boolean, natural (自然数, 大于等于 0 的整数), integer, float, character, string, range (整型数组),
Date	date (年月日), time (时分秒), datetime (年月日时分秒)
Image	image, dataimage
Color	color
Text	paragraph, sentence, word, title, cparagraph, csentence, cword, cttitle
Name	first, last, name, cfirst, clast, cname
Web	url, domain, email, ip, tld
Address	area, region
Helper	capitalize, upper, lower, pick, shuffle
Miscellaneous	guid, id

3.6.1 基本类型占位符

1. 随机生成基本数据类型的数据。

常用的占位符 : natural/integer/string/float/boolean

- 代码 :

```

1 const Mock = require('mockjs')
2
3 const data = Mock.mock({
4   'empList|3': [
5     'id|+1',
6     'name': '@string',
7     'price': '@float',
8     'status': '@boolean',
9   ],
10 }
11
12 console.log(JSON.stringify(data, null, 2))

```

- 效果 :

```

1 {
2   "empList": [
3     {
4       "id": 1,
5       "name": "oYys0",
6       "price": 2234370320974880.8,

```

```
7     "status": true
8   },
9   {
10    "id": 2,
11    "name": "bPWuAB",
12    "price": -7034682015170889,
13    "status": true
14  },
15  {
16    "id": 3,
17    "name": "jYpuON",
18    "price": 1074116758901916.8,
19    "status": false
20  }
21 ]
22 }
```

3.6.2 日期占位符

- 随机生成日期类型的数据 ,

占位符:

- date/date(format)
- time/time(format)
- datetime/datetime(format)

- 代码 :

```
1 const data = Mock.mock({
2   'empList|3': [
3     'id|+1': 1,
4     'name': '@string',
5     'price': '@float',
6     'status': '@boolean',
7     'birthday': '@date', // 默认 yyyy-MM-dd
8     'entryDate': '@date("yyyy/MM/dd")', // 指定日期格式 yyyy/MM/dd
9     'createDate': '@datetime', // 默认 yyyy-MM-dd HH:mm:ss
10    'updateDate': '@datetime("yyyy/MM/dd HH:mm:ss")' //默认yyyy/MM/dd HH:mm:ss
11  ],
12 })
```

- 效果 :

```
1 {
2   "empList": [
3     {
4       "id": 1,
5       "name": "HddBn",
6       "price": 2075727308720240.5,
7       "status": true,
```

```
8     "birthday": "1971-01-03",
9     "entryDate": "1992/03/29",
10    "createDate": "2007-05-20 01:34:12",
11    "updateDate": "2009/09/05 00:08:40"
12  },
13  {
14    "id": 2,
15    "name": "q)krlz",
16    "price": 3905546560859356,
17    "status": false,
18    "birthday": "1993-01-06",
19    "entryDate": "2008/12/18",
20    "createDate": "2009-03-29 08:05:22",
21    "updateDate": "1992/06/27 04:22:20"
22  },
23  {
24    "id": 3,
25    "name": "O0B5",
26    "price": -3833655335201049,
27    "status": true,
28    "birthday": "1976-04-01",
29    "entryDate": "1996/10/27",
30    "createDate": "2010-04-24 09:38:28",
31    "updateDate": "1993/12/29 08:56:26"
32  }
33 ]
34 }
```

3.6.3 图像占位符

- 随机生成图片地址，生成的浏览器可以打开

占位符：image

- 代码：

```
1 const data = Mock.mock({
2   'empList|3': [
3     'id|+1': 1,
4     'name': '@string',
5     'price': '@float',
6     'status': '@boolean',
7     'birthday': '@date', // 默认 yyyy-MM-dd
8     'entryDate': '@date("yyyy/MM/dd")', // 指定日期格式 yyyy/MM/dd
9     'createDate': '@datetime', // 默认 yyyy-MM-dd HH:mm:ss
10    'updateDate': '@datetime("yyyy/MM/dd HH:mm:ss")', // 默认yyyy/MM/dd HH:mm:ss
11    'pic': '@image',
12  ],
13 })
```

- 效果：

```
1  {
2    "empList": [
3      {
4        "id": 1,
5        "name": "($OU",
6        "price": 317108834629012.44,
7        "status": true,
8        "birthday": "2011-05-10",
9        "entryDate": "1983/12/04",
10       "createDate": "1976-02-20 07:10:18",
11       "updateDate": "1982/04/01 17:18:17",
12       "pic": "http://dummyimage.com/160x600"
13     },
14     {
15       "id": 2,
16       "name": "71Nmfv",
17       "price": -2832119950826008.5,
18       "status": false,
19       "birthday": "2000-08-29",
20       "entryDate": "2015/08/26",
21       "createDate": "1991-05-30 13:48:48",
22       "updateDate": "1970/06/29 17:05:55",
23       "pic": "http://dummyimage.com/250x250"
24     },
25     {
26       "id": 3,
27       "name": "!!!8v",
28       "price": -217074031302344.28,
29       "status": false,
30       "birthday": "1990-09-25",
31       "entryDate": "1997/09/14",
32       "createDate": "1996-09-03 19:20:55",
33       "updateDate": "1987/07/03 06:18:05",
34       "pic": "http://dummyimage.com/300x600"
35     }
36   ]
37 }
```

3.6.4 文本占位符

1. 随机生成一段文本

占位符：

- `ctitle` 随机生成一句中文标题。
- `csentence(mix?, max?)` 随机生成一段中文文本。

• 代码：

```
1  const data = Mock.mock({
2    'empList|3': [{
```

```
3     'id|+1': 1,
4     'name': '@string',
5     'price': '@float',
6     'status': '@boolean',
7     'birthday': '@date', // 默认 yyyy-MM-dd
8     'entryDate': '@date("yyyy/MM/dd")', // 指定日期格式 yyyy/MM/dd
9     'createDate': '@datetime', // 默认 yyyy-MM-dd HH:mm:ss
10    'updateDate': '@datetime("yyyy/MM/dd HH:mm:ss")', // 默认 yyyy/MM/dd HH:mm:ss
11    'pic': '@image', // 图片地址
12    'title': '@ctitle(3, 6)', // 中文标题(3到6个字)
13    'content': '@csentence(8, 12)', // 一段中文文本(8到12个字)
14  }]
15 })
```

- 效果：

```
1  {
2   "empList": [
3   {
4     "id": 1,
5     "name": "XAPUq",
6     "price": 719596174326132.4,
7     "status": false,
8     "birthday": "1979-07-10",
9     "entryDate": "1995/07/28",
10    "createDate": "2001-04-03 08:06:28",
11    "updateDate": "2005/10/25 14:00:33",
12    "pic": "http://dummyimage.com/120x240",
13    "title": "术题种已",
14    "content": "市手据来而正收党取科流。"
15  },
16  ..
17  ..
18 ]
19 }
```

3.6.5 名称占位符

- 随机生成名称。

占位符：

- first 英文名。
- last 英文姓。
- name 英文姓名。
- cfirst 中文名。
- clast 中文姓。
- cname 中文姓名。

- 代码：

```
1 const data = Mock.mock({
2   'empList|3': [
3     'id|+1': 1,
4     'name': '@cname', //中文姓名
5     'price': '@float',
6     'status': '@boolean',
7     'birthday': '@date', // 默认 yyyy-MM-dd
8     'entryDate': '@date("yyyy/MM/dd")', // 指定日期格式 yyyy/MM/dd
9     'createDate': '@datetime', // 默认 yyyy-MM-dd HH:mm:ss
10    'updateDate': '@datetime("yyyy/MM/dd HH:mm:ss")', //默认yyyy/MM/dd HH:mm:ss
11    'pic': '@image', // 图片地址
12    'title': '@ctitle(3, 6)', // 中文标题(3到6个字)
13    'content': '@csentence(8, 12)', // 一段中文文本(8到12个字)
14    'first': '@cfirst', // 中文名
15    'last': '@last', // 英文姓
16  ],
17 },
18 })
```

- 效果：

```
1 {
2   "empList": [
3     {
4       "id": 1,
5       "name": "杨娜",
6       "price": 1104108454288808.2,
7       "status": true,
8       "birthday": "2006-02-14",
9       "entryDate": "2000/07/14",
10      "createDate": "1974-08-19 01:43:30",
11      "updateDate": "1974/01/25 17:41:55",
12      "pic": "http://dummyimage.com/120x60",
13      "title": "容京给习反市",
14      "content": "则物例边技我细利质保拉。",
15      "first": "方",
16      "last": "Jones"
17    },
18    ...
19    ...
20  ]
21 }
```

3.6.6 网络占位符

1. 可随机生成 URL、域名、IP 地址、邮件地址

占位符:

- url(protocol?, host?) 生成 URL。



- protocol : 协议 , 如 http
- host : 域名和端口号 , 如 mengxuegu.com
- domain 生成域名。
- ip 生成 IP 地址。
- email 生成邮件地址 。

- 代码 :

```
1 const data = Mock.mock({
2   'empList|3': [
3     'id|+1': 1,
4     'name': '@cname', //中文姓名
5     'price': '@float',
6     'status': '@boolean',
7     'birthday': '@date', // 默认 yyyy-MM-dd
8     'entryDate': '@date("yyyy/MM/dd")', // 指定日期格式 yyyy/MM/dd
9     'createDate': '@datetime', // 默认 yyyy-MM-dd HH:mm:ss
10    'updateDate': '@datetime("yyyy/MM/dd HH:mm:ss")', //默认yyyy/MM/dd HH:mm:ss
11    'pic': '@image', // 图片地址
12    'title': '@ctitle(3, 6)', // 中文标题(3到6个字)
13    'content': '@csentence(8, 12)', // 一段中文文本(8到12个字)
14    'first': '@cfirst', // 中文名
15    'last': '@last', // 英文姓
16    'url': '@url("http", "mengxuegu.com")', // URL
17    'domain': '@domain', // 域名
18    'ip': '@ip', // IP
19    'email': '@email' // 邮箱地址
20  ],
21 })
```

- 效果 :

```
1 {
2   "empList": [
3     {
4       "id": 1,
5       "name": "康娟",
6       "price": -7720628302340068,
7       "status": true,
8       "birthday": "1977-05-03",
9       "entryDate": "2017/06/19",
10      "createDate": "1988-10-31 12:25:19",
11      "updateDate": "1972/01/19 02:20:00",
12      "pic": "http://dummyimage.com/160x600",
13      "title": "究着导据",
14      "content": "产厂术象至公花规口题律到。",
15      "first": "郭",
16      "last": "Martinez",
17      "url": "http://mengxuegu.com/hvvprrd",
18      "domain": "bslpkaozi.na",
19      "ip": "41.167.74.180",
```

```
20     "email": "v.gmoczmc@pawsqbcvvn.hr"
21   },
22   ..
23   ..
24 ]
25 }
```

3.6.7 地址占位符

1. 随机生成区域、省市县、邮政编码

占位符：

- region 区域。如：华南
- county(true) 省市县。
- zip 邮政编码。

• 代码：

```
1 const data = Mock.mock({
2   'empList|3': [
3     {
4       'id|+1': 1,
5       'name': '@cname', // 中文姓名
6       'price': '@float',
7       'status': '@boolean',
8       'birthday': '@date', // 默认 yyyy-MM-dd
9       'entryDate': '@date("yyyy/MM/dd")', // 指定日期格式 yyyy/MM/dd
10      'createDate': '@datetime', // 默认 yyyy-MM-dd HH:mm:ss
11      'updateDate': '@datetime("yyyy/MM/dd HH:mm:ss")', // 默认 yyyy/MM/dd HH:mm:ss
12      'pic': '@image', // 图片地址
13      'title': '@ctitle(3, 6)', // 中文标题(3到6个字)
14      'content': '@csentence(8, 12)', // 一段中文文本(8到12个字)
15      'first': '@cfirst', // 中文名
16      'last': '@last', // 英文姓
17      'url': '@url("http", "mengxuegu.com")', // URL
18      'domain': '@domain', // 域名
19      'ip': '@ip', // IP
20      'email': '@email', // 邮箱地址
21      'area': '@region', // 区域
22      'address': '@county(true)', // 省市县
23      'zipCode': '@zip' // 邮政编码
24    }
25 })
```

• 效果：

```
1 {
2   "empList": [
3     {
4       "id": 1,
5       "name": "何静",
```

```
6   "price": 922632759748656.4,
7   "status": true,
8   "birthday": "2005-08-08",
9   "entryDate": "1990/10/03",
10  "createDate": "1983-08-22 15:59:26",
11  "updateDate": "1975/02/22 08:32:13",
12  "pic": "http://dummyimage.com/240x400",
13  "title": "难县日地队",
14  "content": "方大两金金压气可收以。",
15  "first": "梁",
16  "last": "Garcia",
17  "url": "http://mengxuegu.com/kwyxot",
18  "domain": "cuf.tv",
19  "ip": "167.120.135.116",
20  "email": "b.sljtq@nptuzefkd.ar",
21  "area": "华东",
22  "address": "吉林省 吉林市 永吉县",
23  "zipCode": "416278"
24 }
25 .
26 .
27 ]
28 }
```

第四章 EasyMock 数据接口

4.1 什么是EasyMock

Easy Mock 是一个可视化，并且能快速生成模拟数据的服务。是杭州大搜车无线团队出品的一个极其简单、高效、可视化、并且能快速生成模拟数据的在线 Mock 服务。

现在 Easy Mock 内置了 Mock.js，我们可以更愉快的伪造数据了。

官网：<https://www.easy-mock.com/>

文档：<https://www.easy-mock.com/docs>

4.2 EasyMock 基本使用

4.2.1 登录或注册

访问 <https://www.easy-mock.com> 后，点击 **开始** 后输入用户名和密码。**如果不存在会自动注册。**

注意：没有找回密码功能，请牢记密码！



登录成功



The dashboard of the Easy Mock platform. At the top, there is a dark navigation bar with the title "Easy Mock", a search bar, and several menu items: "# 我的项目", "工作台", "数据", "文档", and "彩蛋". On the left, there is a sidebar with a user icon and the heading "个人项目" (Personal Projects), which includes a sub-section "演示项目" (Demo Project) showing a list of mock types. The main area features a banner for "CODING CLOUD DEVELOPMENT" with the text "一键开启企业 DevOps 全流程" (One-click to start enterprise DevOps full process) and a "免费体验" (Free Experience) button. There is also a watermark "mengxuegu.com" diagonally across the dashboard.

4.2.2 创建项目

右下角  图标进行创建项目

Easy Mock Search Easy Mock # 我的项目 ▾ ⌂ 工作台 ⌂ 数据 ⌂ 文档 ⌂ 彩蛋 ▾

个人项目 这里将展示你的个人项目，当然也包括协同项目。

全部 我创建的 我加入的

CODING | 一键开启 企业 DevOps 全流程 免费体验

演示项目 已创建多种 Mock 类型，只需点... /example

1562575230069 / mxg-cms

项目基础 URL ? / example

项目描述 梦学谷后台管理系统

Swagger Docs API (可选) URL http://example.com/swagger.json

如果后台有提供 Swagger 文档（并且没有验证授权的问题），于是我们可以在此处填写 Swagger 的接口地址，Easy Mock 会自动基于此接口创建 Mock 接口。?

邀请成员 协同编辑 (可选) 用户昵称、用户名，支持模糊匹配

梦学谷

创建成功效果图：

Easy Mock

Search Easy Mock

我的项目 ▾

↔ 工作台

数据



个人项目

这里将展示你的个人项目，当然也包括协同项目。

CODING
CLOUD DEVELOPMENT

一键开启 企业 DevOps 全

mxg-cms

梦学谷后台管理系统

/



演示项目

已创建多种 Mock 类型，只需点...

/example



☆

4.2.3 接口配置

1. 创建接口，点击左下角项目。

Easy Mock Search Easy Mock # 我的项目 ↴ 工作台

个人项目 这里将展示你的个人项目，当然也包括协同项目。

CODING CLOUD DEVELOPMENT 一键开启企业 DevOps



mxg-cms ★
梦学谷后台管理系统
/ /example
+ - 🎁

演示项目 ★
已创建多种 Mock 类型，只需点...
/example 🎁

2. 进入项目工作台页面，点击 创建接口

Easy Mock Search Easy Mock # 我的项目 ↴ 工作台 ⚡ 数据 📄 文档 🎀 彩蛋

mxg-cms 个人项目

梦学谷后台管理系统

Base URL: <https://www.easy-mock.com/mock/5d3d62c0250e9354d3640960>
Project ID: 5d3d62c0250e9354d3640960

+ 创建接口 ★ 工作台 ⚡ 同步 Swagger ↴ 打包下载

Method	URL	描述	操作

3. 左侧编辑窗口输入 mock.js 代码，右侧定义 Method 、 Url 、描述等信息。

```
1+ {  
2   "data": {}  
3 }
```

mock.js 代码

4. 将 `mockjs-demo` 工程目录下的 `demo2.js` 中的对象放入左侧编辑窗口

```
1  {  
2   'empList|3': [  
3     'id|+1': 1,  
4     'name': '@cname', //中文姓名  
5     'price': '@float',  
6     'status': '@boolean',  
7     'birthday': '@date', // 默认 yyyy-MM-dd  
8     'entryDate': '@date("yyyy/MM/dd")', // 指定日期格式 yyyy/MM/dd  
9     'createDate': '@datetime', // 默认 yyyy-MM-dd HH:mm:ss  
10    'updateDate': '@datetime("yyyy/MM/dd HH:mm:ss")', //默认yyyy/MM/dd HH:mm:ss  
11    'pic': '@image', // 图片地址  
12    'title': '@ctitle(3, 6)', // 中文标题(3到6个字)  
13    'content': '@csentence(8, 12)', // 一段中文文本(8到12个字)  
14    'first': '@cfirst', // 中文名  
15    'last': '@last', // 英文姓  
16    'url': '@url("http", "mengxuegu.com")', // URL  
17    'domain': '@domain', // 域名  
18    'ip': '@ip', // IP  
19    'email': '@email', // 邮箱地址  
20    'area': '@region', // 区域  
21    'address': '@county(true)', // 省市县  
22    'zipCode': '@zip' // 邮政编码  
23  ]  
24 }
```

5. 右侧窗口选择请求方式，URL 和描述，然后点击 `创建`

```

1~ {
2~   'empList|3': [
3~     {
4~       'id|+1': 1,
5~       'name': '@cname', //中文姓名
6~       'price': '@float',
7~       'status': '@boolean',
8~       'birthday': '@date', // 默认 yyyy-MM-dd
9~       'entryDate': '@date("yyyy/MM/dd")', // 指定日期格式 yyyy/MM/dd
10~      'createDate': '@datetime', // 默认 yyyy-MM-dd HH:mm:ss
11~      'updateDate': '@datetime("yyyy/MM/dd HH:mm:ss")', //默认yyyy-MM-dd HH:mm:ss
12~      'pic': '@image', // 图片地址
13~      'title': '@ctitle(3, 6)', // 中文标题(3到6个字)
14~      'content': '@csentence(8, 12)', // 一段中文文本(8到12个字)
15~      'first': '@cfirst', // 中文名
16~      'last': '@last', // 英文姓
17~      'url': '@url("http", "mengxuegu.com")', // URL
18~      'domain': '@domain', // 域名
19~      'ip': '@ip', // IP
20~      'email': '@email', // 邮箱地址
21~      'area': '@region', // 区域
22~      'address': '@county(true)', // 省市县
23~      'zipCode': '@zip' // 邮政编码
24~   }
}

```



4.2.4 接口测试

1. 接口右侧预览接口和复制接口地址

Method	URL	描述	操作
GET	/test	测试	复制接口地址 预览接口 修改

2. 修改接口和克隆接口和删除接口

Method	URL	描述	操作
GET	/test	测试	修改 克隆 下载 删除

4.3 阿里云部署 EasyMock 应用



部署方式参考[梦学谷]文章：<https://mp.weixin.qq.com/s/hVHDIMZUerTiXTYgX0ylwQ>

顺便可关注公众号：梦学谷

后台 进程进行 easymock

```
1 nohup npm run start &
2
3 exit
```

查看与结束项目

```
1 ps -ef|grep node
2
3 kill -9 进程ID
```

第五章 项目环境搭建

5.1 基于 Vue-CLI 3.x 创建项目

Vue CLI 环境要求：需要安装 Node.js 8.9+ (推荐 8.11.0+)

5.1.1 Vue CLI 安装

1. 全局安装 **Vue-CLI**

```
1 npm install -g @vue/cli@3.10.0
```

2. 安装成功后，在命令行可以使用 **vue** 命令，比如 查看版本

```
1 # 大写V
2 vue -V
```

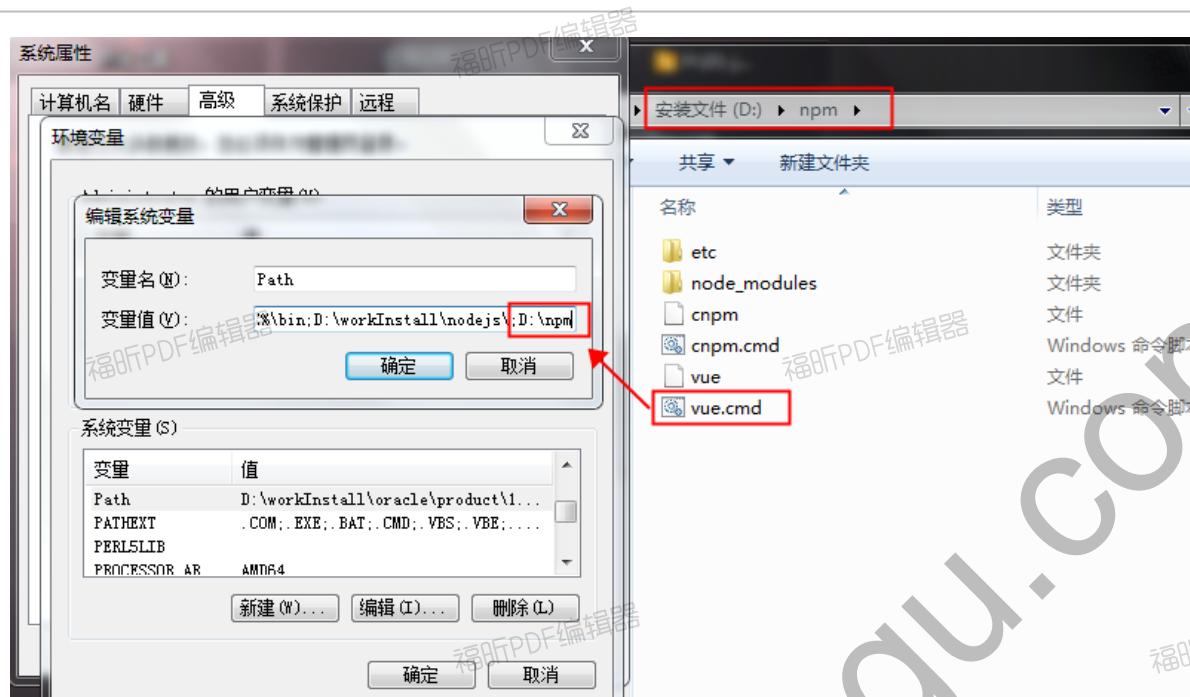
- 如果执行上面后，命令行提示 '**vue**' 不是内部或外部命令
- 解决方法：配置环境变量

1. 查看全局安装目录 **npm root -g**

```
C:\Users\Administrator>npm root -g
d:\npm\node_modules
```

2. 在我的电脑 进入全局安装目录下，找到 **vue.cmd**

3. 右键计算机，属性—>高级系统设置—>环境变量，将 **vue.cmd** 的路径加入环境变量，点击“确定”



4. 重启命令行窗口 , vue -V 执行正常。

5.1.2 Vue CLI 创建项目

- 创建项目命令 :

会员管理系统(Member Management System, 简称 MMS)

```
1 vue create mxg-mms

? Check the features needed for your project: <Press <space> to select, <a> to t
? Check the features needed for your project:
  (*) Babel
  ( ) TypeScript
  ( ) Progressive Web App (PWA) Support
  (*) Router
  ( ) Vuex
  (*) CSS Pre-processors
  (*) Linter / Formatter
  ( ) Unit Testing
  ( ) E2E Testing
```

资料中提供了创建好的项目，位于：Vue-CLI 综合实战项目资源\vue-cli-defined.zip

直接解压，将目录重命名为 mxg-mms，然后将 package.json 中 name 的值改为 mxg-mms

5.1.3 启动项目测试

启动项目：

```
1 npm run serve
```

退出，命令行按 **Ctrl + c**，输入 **y**

[Home](#) | [About](#)

Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).

Installed CLI Plugins

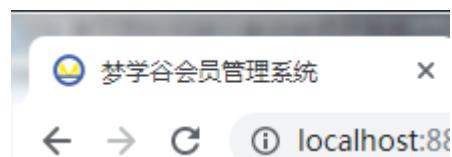
[babel](#) [eslint](#)

5.2 初始化项目

5.2.1 更改标题

找到 public\index.html 页面，修改 title 内容：`<title>梦学谷会员管理系统</title>`

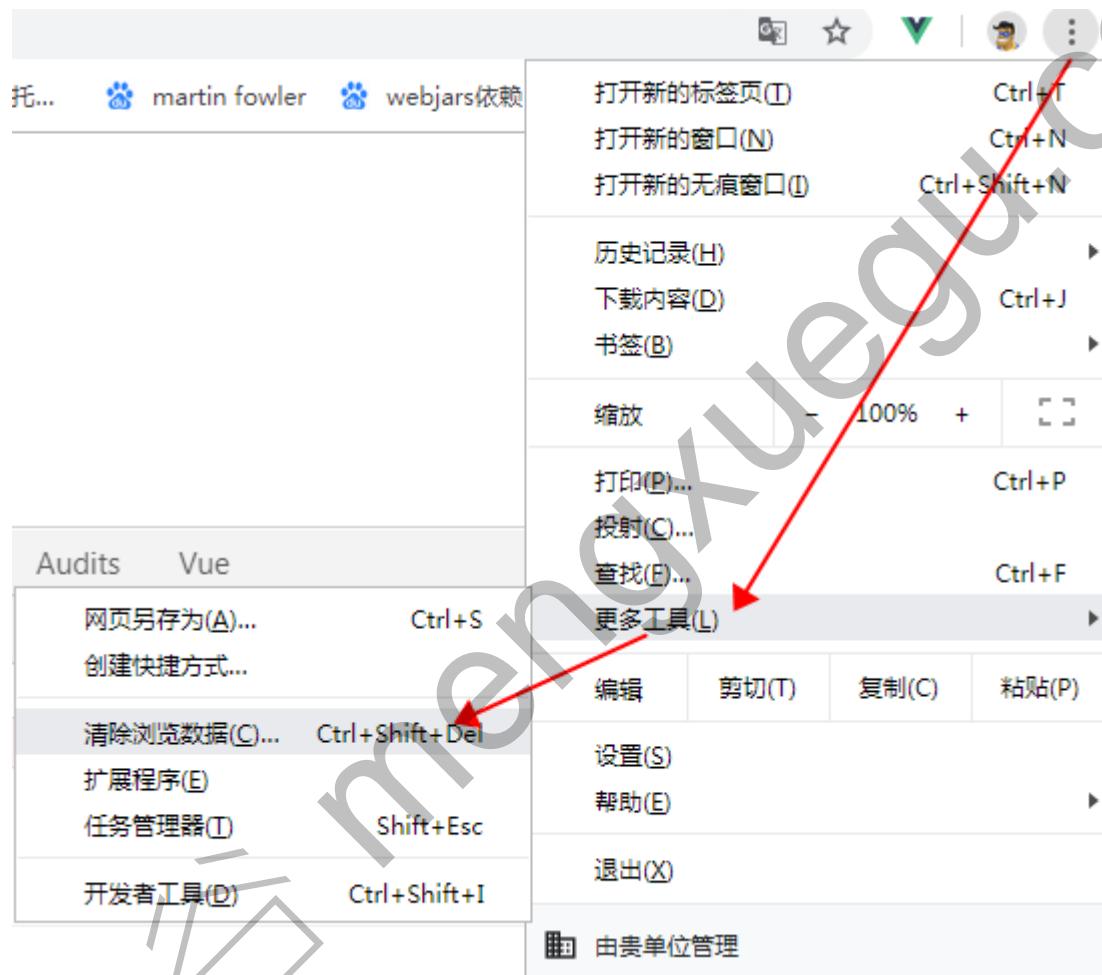
```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width,initial-scale=1.0">
7     <link rel="icon" href="<%= BASE_URL %>favicon.ico">
8     <title>梦学谷会员管理系统</title>
9   </head>
10  <body>
11    <div id="app"></div>
12    <!-- built files will be auto injected -->
13  </body>
14 </html>
```



5.2.2 更改ICO图标

把目录下的 favicon.ico 粘贴到 mxg-mms\public 目录下，替换原有的图标。然后清除浏览器的缓存。

注意是将**时间不限**的缓存清除





5.2.3 配置 vue.config.js

在 mxg-mms 根目录下创建 `vue.config.js`，添加如下配置：

注意，不要直接复制pdf中的，复制 `Vue-CLI 综合实战项目资源\vue.config.js`

```
1 module.exports = {
2   devServer: {
3     port: 8888, // 端口号，如果端口号被占用，会自动提升1
4     host: "localhost", // 主机名，127.0.0.1，真机0.0.0.0
5     https: false, // 协议
6     open: true, // 启动服务时自动打开浏览器访问
7   },
8
9   lintOnSave: false, // 关闭格式检查
10  productionSourceMap: false, // 打包时不会生成.map文件，加快打包速度
11 }
```

视频中这里打包了，只是示范下不生成map的效果

5.2.4 整合第三方库

1. 安装 `axios`，处理异步请求

```
1 npm i -S axios
```

2. 安装 `pubsub-js`，实现非父子组件间通信

在 Vue.js 教程中的 6.7 非父子组件间通信 PubSubJS 讲解

视频：[第8章第14节_非父子\(兄弟\)组件间通信分析](#) [第8章第15节_BootStrap案例-非父子\(兄弟\)组件间通信](#)

```
1 npm i -S pubsub-js
```

3. 查看 `package.json` 中是否有对应依赖

5.3 整合 ElementUI

5.3.1 ElementUI 简介

Element 是饿了么平台推出的一套基于 Vue.js 开发的后台页面组件库。

官网：<https://element.eleme.cn/>



5.3.2 ElementUI 安装

将 `element-ui` 模块通过本地安装为生产依赖。在 `mxg-ssm` 目录下的命令行窗口，输入以下命令：

```
1 npm i -S element-ui
```



5.3.3 完整引入 ElementUI

在 `mxg-ssm\src\main.js` 中导入 `element-ui` 和 `element-ui/lib/theme-chalk/index.css` ,

使用 `Vue.use(ElementUI)`

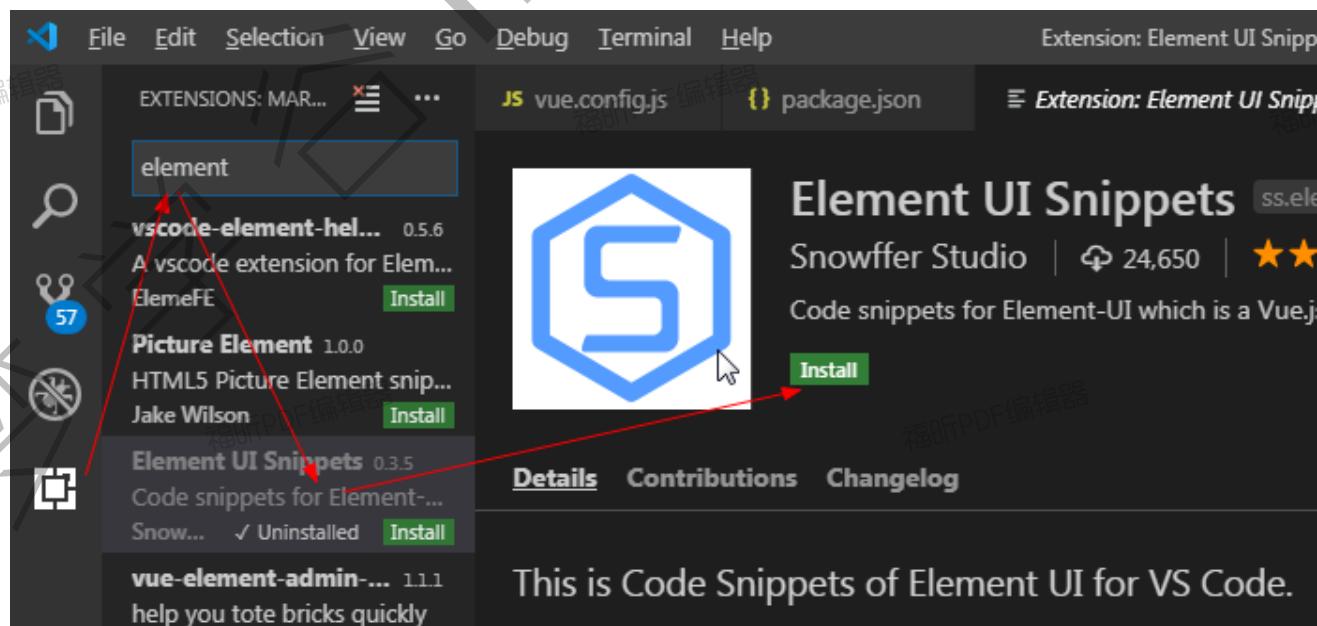
```
1 import Vue from "vue";
2 import ElementUI from 'element-ui'; // 组件库
3 import 'element-ui/lib/theme-chalk/index.css'; // 样式
4 import App from "./App.vue";
5 import router from "./router";
6
7 // 使用 ElementUI
8 Vue.use(ElementUI);
9
10 // 消息提示的环境配置，是否为生产环境：
11 // false 开发环境：Vue会提供很多警告来方便调试代码。
12 // true 生产环境：警告却没用，反而会增加应用的体积
13 Vue.config.productionTip = process.env.NODE_ENV === 'production';
14
15 new Vue({
16   router,
17   render: h => h(App)
18 }).$mount("#app");
```

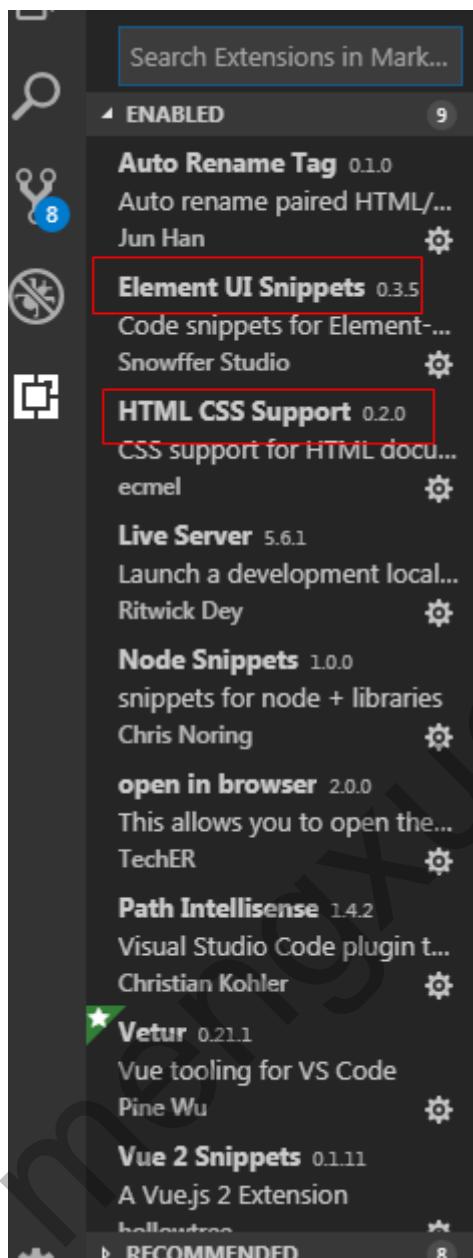
process.env是一个常量，用于获取环境配置

不重要

5.3.4 ElementUI 插件安装

在 VS Code 中安装 Element UI Snippets 插件，有 element 语法提示



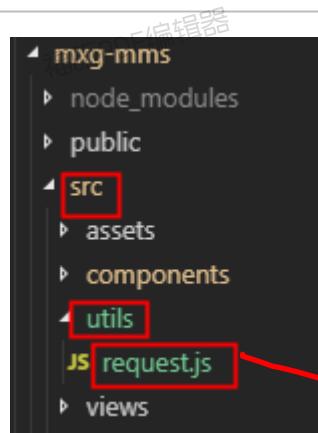


第六章 Axios 封装和跨域问题

6.1 封装 Axios 对象

因为项目中很多组件中要通过 Axios 发送异步请求，所以封装一个 Axios 对象。自己封装的 Axios 在后续可以使用 axios 中提供的拦截器。

1. 在 `src` 目录下创建 `utils` 目录及 `utils` 下面创建 `request.js` 文件



2. 内容如下：

参考：

<https://github.com/axios/axios#creating-an-instance>

<https://github.com/axios/axios#request-config>

```
1 import axios from 'axios'  
2  
3 // 手动创建一个 axios 对象, 参考: https://github.com/axios/axios#creating-an-instance  
4 const request = axios.create({  
5   // 请求配置参考: https://github.com/axios/axios#request-config  
6   // 根据不同环境设置 baseURL, 最终发送请求时的URL为: baseURL + 发送请求时写URL,  
7   // 比如 get('/test'), 最终发送请求是: /dev-api/test  
8   // baseURL: '/dev-api',  
9   // baseURL: '/',  
10  // timeout: 5000 // 请求超时  
11 })  
12  
13 export default request // 导出 axios 对象
```

6.2 测试封装的 Axios

1. 在 `public` 下创建 `db.json` 数据文件, 内容如下：

```
1 [  
2   {"name": "小梦", "age": 18},  
3   {"name": "小岳岳", "age": 28},  
4   {"name": "林志玲", "age": 38}  
5 ]
```

由于过程繁杂, 只用看开始这段最后代码, 但暂未涉及跨域



2. 在 `src` 目录下创建 `api` 目录及 `api` 下面创建 `test.js` 文件



解耦前

```
bvideo > src > utils > JS request.js > ...
1 import axios from 'axios'
2
3 // axios.get('').then(response=>{
4 //   const data = response.data
5 // })
6
7 // axios.get('/db.json').then(response=>{
8 //   console.log(response.data)
9 // })
10
11 const request = axios.create({
12   // baseURL: '/dev-api',
13   // 响应 http://localhost:3567/dev-api/db.json 404 (Not Found) ,因此知道baseURL的作用
14   baseURL: '/',
15   timeout: 5000
16 })
17
18 // request.get('/db.json').then(response=>{
19 //   console.log(response.data)
20 // })
21
22 request.interceptors.request.use(config=>{
23   return config
24 },error=>{
25   return Promise.reject(error);
26 })
27 request.interceptors.response.use(response=>{
28   return response
29 },error=>{
30   return Promise.reject(error);
31 })
32
33 export default request |
```

3. 测试请求 db.json

```
1 // @代表的
2 // import a
3 import rec
4
5 // 测试1 调用
6 request.ge
7 console.lo
8 }).catch(er
9 console.lo
10 });
31
32
33 export default request |
```

4. 在 HelloWorld.vue

```
1 <script>
2 // 直接导入 测试1 就...
3 import testApi from ...
4
5 </script>
```

```
bvideo > src > components > HelloWorld.vue > {} "HelloWorld.vue" > script
```

5. 访问 <http://localhost:8080>HelloWorld>

解耦后, 新增 test.js

[HMR] Waiting for update signal from browser

```
get1 ▶ (3) [{} , {} , {}]
  ▶ 0: {name: "小梦", age: 18}
  ▶ 1: {name: "小岳岳", age: 18}
  ▶ 2: {name: "林志玲", age: 18}
  length: 3
  __proto__: Array(0)
```

```
bvideo > src > api > JS test.js > ...
1 import request from '@/utils/request'
2
3 // request.get('/db.json').then(response => {
4 //   console.log(response.data)
5 // })
6
7 request({
8   method:'get',
9   url:'/db.json'
10 }).then(response=>{
11   console.log('get2', response.data)
12 })
13
14 export default {
15   getList(){
16     const req = request({
17       method: 'get',
18       url:'/db.json'
19     })
20     console.log(req) //Promise.then
21     return req
22   }
23 }
```

6. 测试2：采用 config 配置参数发送请求, 将URL前缀抽成一个常量方

```
1 import request from "@/utils/request";
```



```

2
3 const BASE_URL = 'http://localhost:8888'
4 // 测试1
5 request.get(BASE_URL + "/db.json").then(response => {
6   console.log("get1", response.data);
7 }).catch(error => {
8   console.log(error);
9 });
10
11 // 测试2, 使用对象形式传入请求配置, 如 请求url, method , params 等
12 request({
13   url: BASE_URL + "/db.json",
14   method: "get"
15 }).then(response => {
16   console.log("get2", response.data);
17 }).catch(error => {
18   console.log(error);
19 });

```

```

89 <script>
90 // import request from '../utils/request.js'
91 import testApi from '@/api/test'
92
93 export default {
94   data(){
95     return {
96       list:[]
97     }
98   },
99
100 created(){
101   this.fetchData()
102 },
103 methods: {
104   fetchData(){
105     testApi.getList().then(response=>{
106       console.log('get3',response.data)
107       this.list = response.data
108     })
109   }
110 },
111

```

hello.vue中也有变化

```

1 <template>
2   <div class="hello">
3     <h1>{{ list }}</h1>
4     <p>
5       For a guide and recipi...

```

msg改为list

重新刷新页面，发现控制台输入 get2 数据

7. 测试3: 导出对象方式， api\test.js 新增如下代码：

导出的默认对象里面是方法，方法返回值为 Promise 对象，通过它调用 then 获取响应数据

```

1 // 测试3 导出默认对象
2 export default {
3   getList() {
4     const req = request({
5       url: BASE_URL + "/db.json",
6       method: "get"
7     });
8     // console.log(req) // Promise
9     return req;
10   }
11 }

```

8. 在 HelloWorld.vue 中导入 test.js

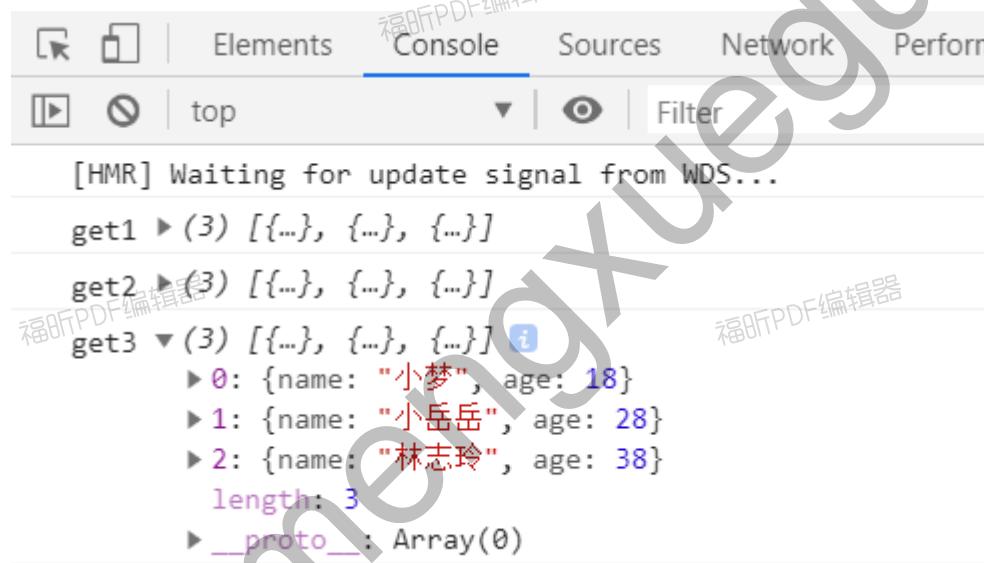
```

1 <script>
2 // 直接导入 测试1和测试2 就会执行
3 import testApi from '@/api/test'
4
5 //验证 测试3
6 export default {
7   created() {
8     // 钩子中初始化数据
9     this.fetchData()
10   },
11   methods: {
12     fetchData() {

```

```
13     testApi.getList().then(response => {
14         console.log('get3', response.data)
15     }).catch(error => {
16         console.log(error)
17     })
18 }
19 },
20
21 name: "HelloWorld",
22 props: {
23     msg: String
24 },
25 };
26 </script>
```

重新刷新页面，发现控制台输入 get3 数据。后面数据访问都采用这种方式



本章有些错误, 请直接到
6.5

6.3 演示跨域现象

1. 将 `mxg-mms\db.json` 拷贝到 `axios-demo\public\` 目录下

axios-demo 工程就是复制了 `mxg-mms` 重命名了

2. `axios-demo\vue-config.js` 更改端口号, 将端口号更改为 8001

3. 启动 `axios-demo`,

```
1 npm run serve
```

启动后, 查看控制台显示的访问地址, 我的是 <http://localhost:8001/>

```
DONE Compiled successfully in 252ms

App running at:
- Local:  http://localhost:8001/
- Network: http://localhost:8001/
```

4. 查询是否可以直接访问到数据, 访问 <http://localhost:8001/db.json>



localhost:8001/db.json

[{"name": "小梦", "age": 18}, {"name": "小岳岳", "age": 28}, {"name": "林志玲", "age": 38}]

5. 修改 mxg-mms\src\api\test.js , 发送请求给 8001 接口 , 修改下 BASE_URL 值

```
1 const BASE_URL = 'http://localhost:8001'
```

6. 重新刷新 页面, 查看控制台报错, 出现跨域问题.

```
Access to XMLHttpRequest at 'http://localhost:8001/db.json' from origin 'http://localhost:8888' has been blocked by
Control-Allow-Origin' header is present on the requested resource.

Error: Network Error
    at createError (webpack-internal:///e/createError.js:16)
    at XMLHttpRequest.handleError (webpack-internal:///adapters/xhr.js:81)

Access to XMLHttpRequest at 'http://localhost:8001/db.json' from origin 'http://localhost:8888' has been blocked by
Control-Allow-Origin' header is present on the requested resource.

Error: Network Error
    at createError (webpack-internal:///e/createError.js:16)
    at XMLHttpRequest.handleError (webpack-internal:///adapters/xhr.js:81)

Access to XMLHttpRequest at 'http://localhost:8001/db.json' from origin 'http://localhost:8888' has been blocked by
Control-Allow-Origin' header is present on the requested resource.

Error: Network Error
    at createError (webpack-internal:///e/createError.js:16)
    at XMLHttpRequest.handleError (webpack-internal:///adapters/xhr.js:81)
```

如何解决？先了解什么跨域。

6.4 什么是跨域

前后端分离时，前端和后端API服务器可能不在同一台主机上，就存在跨域问题，浏览器限制了跨域访问。

同源策略：是指协议，域名，端口都要相同，其中有一个不同都会产生跨域。

如下面示例存在跨域：

- 前端页面部署在：<http://localhost:8001> 即本地ip端口8001上；
- 后端API部署在：<http://localhost:8002> 即本地ip端口8002上；

它们IP一样，但是端口不一样，所以存在跨域问题。

跨域解决:

- 通过代理请求的方式解决，将 API 请求通过代理服务器请求到 API 服务器。
 - 开发环境中，在 `vue.config.js` 文件中使用 `devServer.proxy` 选项进行代理配置。

参考：<https://cli.vuejs.org/zh/config/#devserver>

本章过程繁杂，且有错误，所以请忽略教程，直接看粘贴进来的代码

6.5 解决开发环境跨域问题

1. 在 `vue.config.js` 文件中使用 `devServer.proxy` 选项进行代理配置

```
1 module.exports = {
2   devServer: {
3     port: 8888, // 端口号，如果端口号被占用，会自动提升
4     host: "localhost", // 主机名，127.0.0.1，真机 0.0.0.1
5     https: false, // 协议
6     open: true, // 启动服务时自动打开浏览器访问
7
8     proxy: {
9       // 匹配 /dev-api 开头的请求，
10      '/dev-api': {
11        // 目标服务器，代理访问到 https://localhost:8001
12        target: 'http://localhost:8001',
13        // 开启代理：在本地会创建一个虚拟服务端，然后代理到目标服务器
14        // 并同时接收请求的数据，这样服务端和服务端进行交互时
15        changOrigin: true, // 开启代理
16        pathRewrite: {
17          // 会将 /dev-api 替换为 "，也就是 /dev-api 会移除
18          // 如 /dev-api/db.json 代理到 https://localhost:8001/db.json
19          '^/dev-api': '',
20        }
21      }
22    },
23  },
24
25  lintOnSave: false, // 关闭格式检查
26  productionSourceMap: false // 打包时不会生成 .map 文件
27 }
```

2. 将 test.js 中的 BASE URL 修改如下：

```
1 const BASE_URI = '/api/v1';
```

```
bvideo > JS vue.config.js > [e] <unknown>
  1 module.exports = [
  2   ...
  3   devServer: {
  4     port: 3567,
  5     host: "localhost",
  6     https: false,
  7     open: true,
  8     proxy: {
  9       "/dev-api": {
 10         target: "http://localhost",
 11         changeOrigin: true,
 12         pathRewrite: {
 13           '^/dev-api': ''
 14         }
 15       }
 16     }
 17   },
 18   lintOnSave: false,
 19   productionSourceMap: false,
 20 ]
```

▼ HelloWorld.vue JS request.js JS test.js

bvideo > src > api > JS test.js > getList > [?] req

```
1 import request from '@/utils/request'
2
3 // request.get('/db.json').then(response => {
4 //   console.log(response.data)
5 // })
6
7 request({
8   method:'get',
9   url: '/dev-api/db.json'
10 }).then(response=>{
11   console.log('get2', response.data)
12 })
13
14 export default {
15   getList(){
16     const req = request({
17       method: 'get',
18       url: '/dev-api/db.json'
19     })
20     console.log(req) //Promise.then
21     return req
22   }
23 }
```

- 3 测试 ·

- #### ○ 重启项目，防止不生效

按 Ctrl+c 停止项目时，可能浏览器控制台会报错，忽略它，那个没有关系的

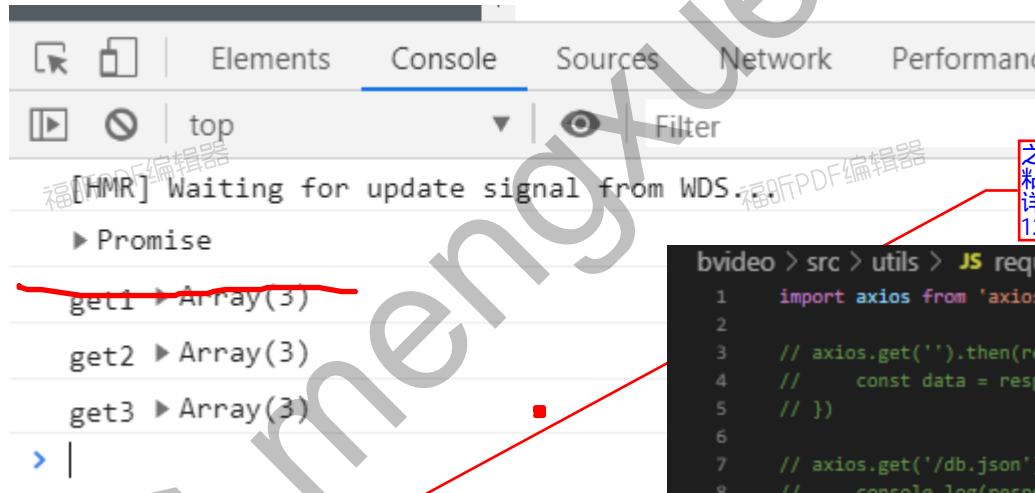
```
✖ ▶ GET http://localhost:8888/sockjs-node/info?t=1564409479548 net::ERR_CONNECTION_REFUSED
✖ ▶ GET http://localhost:8888/sockjs-node/info?t=1564409480598 net::ERR_CONNECTION_REFUSED
```

1. 找到mxg-mms/node_modules/sockjs-client/dist/sockjs.js
2. 找到代码的 1605行 注释掉 (找不到就ctrl+f搜索: self.xhr.send)

但是注释后，热加载有时间就不灵了。

```
1 try {
2     // self.xhr.send(payload);
3 } catch (e) {
4     self.emit('finish', 0, '');
5     self._cleanup(false);
6 }
```

- 访问 <http://localhost:8888/#/>，查看控制台响应了数据，说明代理成功。



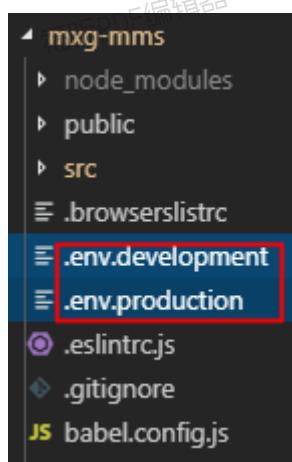
```
bvideo > src > utils > JS request.js > ...
1 import axios from 'axios'
2
3 // axios.get('').then(response=>{
4 //   const data = response.data
5 // })
6
7 // axios.get('/db.json').then(response=>{
8 //   console.log(response.data)
9 // })
10
11 const request = axios.create({
12   baseURL: '/dev-api',
13   // 响应 http://localhost:3567/dev-api/db.json
14   // baseURL: '/',
15   // baseURL: '/dev-api',
16   // baseURL: process.env.VUE_APP_BASE_API,
17   // 注意，此处设置有错 125 17:09，后更改过来了
18   timeout: 5000
19 })
20
21 // request.get('/db.json').then(response=>{
22 //   console.log(response.data)
23 // })
24
25 request.interceptors.request.use(config=>{
26   return config
27 }, error=>{
28   return Promise.reject(error);
29 })
30 request.interceptors.response.use(response=>{
31   return response
32 }, error=>{
33   return Promise.reject(error);
34 })
35
36 export default request
```

6.7 配置不同环境常量值

参考：<https://cli.vuejs.org/zh/guide/mode-and-env.html>

开发环境请求 Mock.js 获取数据，开发环境请求后台接口获取数据，不同环境匹配不同请求接口URL，通过路径前缀进行匹配。当前只针对开发环境

1. 在 mxg-mms 根目录下创建 .env.development 和 .env.production 文件



```
test.js > .env.development
1 import request from '@/utils/request'
2 // request.get('/db.json').then(response => {
3 //   console.log(response.data)
4 // })
5 const BASE_URI = process.env.VUE_APP_BASE_API
6
7 request({
8   method: 'get',
9   // url: BASE_URI + '/db.json'
10  url: '/db.json'
11 }).then(response => {
12   console.log('get2', response.data)
13 })
14
15 export default [
16   getList() {
17     const req = request({
18       method: 'get',
19       // url: BASE_URI + '/db.json'
20       url: '/db.json'
21     })
22     console.log(req) //Promise.then
23     return req
24   }
25 ]
```

2. .env.development 文件配置 (注意开发和生产环境配置不要搞反了)

VUE_APP_SERVICE_URL 值可更改为你自己配置的 Easy-Mock 地址

```
1 # 使用 VUE_APP_ 开头的变量会被 webpack 自动加载
2 # 定义请求的基础URL, 方便跨域请求时使用
3 VUE_APP_BASE_API = '/dev-api'
4
5 # 接口服务地址, 以你自己的为主
6 VUE_APP_SERVICE_URL = 'http://localhost:8001'
```

3. .env.production 文件配置 (注意开发和生产环境配置不要搞反了)

```
1 # 使用 VUE_APP_ 开头的变量会被 webpack 自动加载
2 # 定义请求的基础URL, 方便跨域请求时使用
3 VUE_APP_BASE_API = '/pro-api'
```

4. 测试：在 main.js 中添加以下代码，看下浏览器控制台是否会输出

在项目任意模块文件中，都可以使用 process.env.VUE_APP_BASE_API 获取值

```
1 console.log(process.env.VUE_APP_BASE_API)
```

6.8 重构代理配置

1. 在重构 vue.config.js 中的 devServer.proxy

```
1 module.exports = {
2   devServer: {
3     port: 8888, // 端口号, 如果端口号被占用, 会自动加 1
4     host: "localhost", // 主机名, 127.0.0.1
5     https: false, // 协议
6     open: true, // 启动服务时自动打开浏览器
7
8     proxy: {
```

```
JS main.js > JS vue.config.js
```

```
bvideo > src > JS main.js > ...
1 import Vue from "vue";
2 import ElementUI from 'element-ui';
3 import 'element-ui/lib/theme-chalk/index.css';
4 import App from "./App.vue";
5 import router from "./router";
6
7 Vue.use(ElementUI);
8
9 Vue.config.productionTip = process.env.NODE_ENV === 'production';
10 console.log(process.env.VUE_APP_SERVICE_URL)
11
12 new Vue({
13   router,
14   render: h => h(App)
15 }).$mount("#app");
```

```
9      // 匹配 /dev-api 开头的请求 ,  
10     // '/dev-api': {  
11       [process.env.VUE_APP_BASE_API]: {  
12         // 目标服务器, 代理访问到 https://local  
13         // target: 'http://localhost:8001',  
14         target: process.env.VUE_APP_SERVICE  
15         // 开启代理 : 在本地会创建一个虚拟服务  
16         // 并同时接收请求的数据 , 这样服务端和  
17         changeOrigin: true, //开启代理  
18         pathRewrite: {  
19           // 会将 /dev-api 替换为 "", 也就是 /dev-  
20           // 如 /dev-api/db.json 代理到 https://  
21           // '^/dev-api': '',  
22           ['^' + process.env.VUE_APP_BASE_AP  
23         }  
24     }  
25   },  
26 },  
27  
28   lintOnSave: false, // 关闭格式检查  
29   productionSourceMap: false // 打包时不会生成 .map 文件 , 加快打包速度  
30 }
```

```
1 module.exports = {  
2   devServer: {  
3     port: 3567,  
4     host: "localhost",  
5     https: false,  
6     open: true,  
7     proxy: [  
8       // '/dev-api': {  
9         [process.env.VUE_APP_BASE_API]: {  
10           target: process.env.VUE_APP_SERVICE_URL,  
11           changeOrigin: true,  
12           pathRewrite: {  
13             // '^/dev-api': '',  
14             ['^' + process.env.VUE_APP_BASE_API]: ''  
15           }  
16         }  
17       }  
18     },  
19   },  
20   lintOnSave: false,  
21   productionSourceMap: false,  
22 }
```

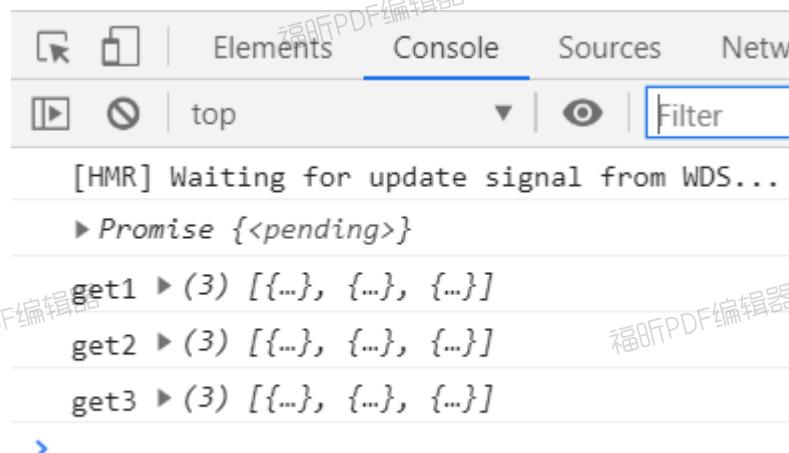
2. 修改 `utils/request.js` 文件配置 : `baseURL: process.env.VUE_APP_BASE_API,`

```
1 import axios from "axios"  
2 // 手动创建一个 axios 对象, 参考: https://github.com/axios/axios#creating-an-instance  
3 const request = axios.create({  
4   // 请求配置参考: https://github.com/axios/axios#request-config  
5   // 根据不同环境设置 baseURL, 最终发送请求时的URL为: baseURL + 发送请求时写URL ,  
6   // 比如 get('/test'), 最终发送请求是: /dev-api/test  
7   // baseURL: '/dev-api',  
8   // baseURL: '/',  
9   // 根目录下的 .env.development 与 .env.production 中配置 VUE_APP_BASE_API  
10  baseURL: process.env.VUE_APP_BASE_API, // "/dev-api"  
11  timeout: 5000 // 请求超时  
12 })  
13  
14 export default request // 导出 axios 对象
```

3. 将 `test.js` 中的 `BASE_URL` 变成空字符

```
1 const BASE_URL = ""
```

4. 重启项目 , 再次访问<http://localhost:8888/#/> 一样效果

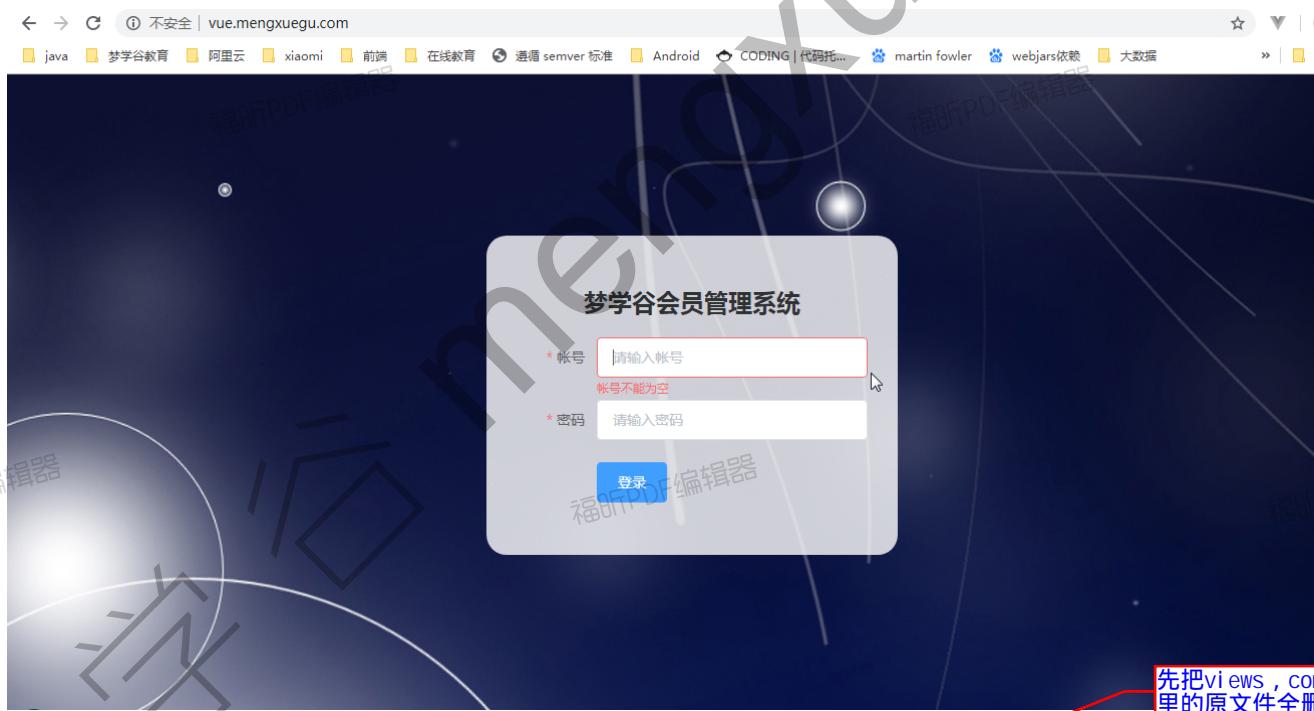


```
[HMR] Waiting for update signal from WDS...
▶ Promise {<pending>}
get1 ▶ (3) [...], [...], [...]
get2 ▶ (3) [...], [...], [...]
get3 ▶ (3) [...], [...], [...]
```

第七章 系统登录管理

7.1 需求分析

开发登录页面，当输入帐号和密码验证通过后，才允许进行到首页。效果图如下：



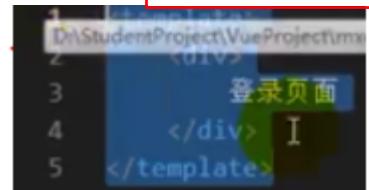
7.2 路由配置

1. 在 `src\views` 目录下新建 `login` 目录及此目录下新建文件 `index.vue`

说明：通过 `import Login from './views/login'` 导入组件，当前只指定了组件路径，默认导入的就是指定路径下的 `index.vue` 组件

2. 在 `src\router.js` 中配置路由（把原有的路由配置删除），如下：

文件如图所示





```
1 import Vue from "vue";
2 import Router from "vue-router";
3 // 默认导入目录下的 index.vue 文件，等价于 ./views/login/index.vue
4 import Login from './views/login'
5
6 Vue.use(Router);
7
8 export default new Router({
9   routes: [
10     {
11       // 登录页
12       path: '/login',
13       name: 'login', // 路由名称
14       component: Login
15     }
16   ],
17 });
18
```

3. 路由渲染出口，在 App.vue 中，注意样式保持一致

```
1 <template>
2   <div>
3     <!-- 路由组件渲染出口 -->
4     <router-view></router-view>
5   </div>
6 </template>
7
8 <style >
9 /* 上面不要加 scoped */
10 body{
11   font-family: "微软雅黑";
12   /* 满屏显示，不显示默认边距 margin:8px, */
13   margin: 0px auto;
14 }
15 </style>
```

7.3 登录页面

参考：Form 表单 » 典型表单

文档：<https://element.eleme.cn/#/zh-CN/component/form#dian-xing-biao-dan>

1. 在 `src\views` 目录下的 `index.vue` 文件中添加登录表单模板

表单输入框，如果没有使用 `v-model` 绑定值，是不允许输入值的。

```
1 <template>
2   <div class="login-container">
3     <!-- ref 相当于 id，:model 表单数据对象, label-width 表单域标签的宽度 -->
```



```
4 <el-form ref="form" :model="form" label-width="80px" class="login-form">
5   <h2 class="login-title">梦学谷会员管理系统</h2>
6   <el-form-item label="帐号" >
7     <el-input v-model="form.username" placeholder="请输入帐号"></el-input>
8   </el-form-item>
9   <el-form-item label="密码" >
10    <el-input type="password" v-model="form.password" placeholder="请输入密码">
11    </el-input>
12  </el-form-item>
13  <el-form-item>
14    <el-button type="primary" @click="onSubmit">登录</el-button>
15  </el-form-item>
16 </el-form>
17 </div>
18 </template>
```

先访问查看效果：<http://localhost:8888/#/login>

2. 添加样式

- 注意：

- 将 `Vue-CLI 综合实战项目资源\login.jpg` 图片拷贝到 `src\assets` 目录下
- 模板中添加类名 `login-container`、`login-form`、`login-title`

```
1 <style scoped>
2 .login-form {
3   width: 350px;
4   /* 上下间隙150px, 左右自动 */
5   margin: 160px auto;
6   /*透明背景色*/
7   background-color: rgb(255,255,255,0.8);
8   padding: 30px;
9   /*圆角*/
10  border-radius: 20px;
11 }
12 .login-container {
13   position: absolute;
14   width: 100%;
15   height: 100%;
16   background: url("../assets/login.jpg");
17   overflow: hidden;
18 }
19 /*标题*/
20 .login-title{
21   text-align: center;
22   color: #303133;
23 }
24 </style>
```

7.4 表单校验

校验输入的帐号和密码都不允许为空

通过 `rules` 属性传入约定的验证规则

<https://hub.com/yiminghe/async-validation>

1. 在 el-form 上添加绑定属性

```

1 <template>
2   <div class="login-container">
3     <!-- :rules="rules" -->
4     <el-form :rules="rules" ref="form" model="form" label-width="80px" class="login-form">
5       <h2 class="login-title">Peilin会员管理系统</h2>
6       <el-form-item label="账号" prop="username">
7         <el-input v-model="form.username" placeholder="请输入账号"></el-input>
8       </el-form-item>
9       <el-form-item label="密码" prop="password">
10        <el-input type="password" v-model="form.password" placeholder="请输入密码"></el-input>
11      </el-form-item>
12      <el-form-item>
13        <el-button type="primary" @click="submitForm('form')">登录</el-button>
14      </el-form-item>
15    </el-form>
16  </div>
17 </template>

```

2. 在组件实例的 data 选项中

```

1 data() {
2   return {
3     form: {},
4     // 表单校验
5     rules: {
6       username: [ // -->
7         { required: true, message: "请输入有效账号", trigger: "blur" }
8       ],
9       password: [
10         { required: true, message: "请输入有效密码", trigger: "blur" }
11       ]
12     }
13   };
14 }

```

3. 添加提交表单处理函数，注意换了函数名是 submitForm

form, fromName 相当于 id, 这里就是定位到相对应的地方



```
1 methods: {  
2   // 注意：按钮上调用的函数名要一致 submitForm  
3   submitForm(formName) {  
4     this.$refs[formName].validate((valid) => {  
5       if (valid) {  
6         alert('submit!');  
7       } else {  
8         console.log('error submit!!');  
9       }  
10      return false;  
11    }  
12  },  
13}
```

7.4 EasyMock 添加登陆认证模拟接口

7.4.1 修改服务接口地址

1. 因为要访问EasyMock 模拟接口, 所以要把 接口地址改一下 , 在 `.env.development` 文件中修改如下 :

注意 : 改成你自己的EasyMock上的接口服务地址

```
1 # 使用 VUE_APP_ 开头的变量会被 webpack 自动加载  
2 # 定义请求的基础URL, 方便跨域请求时使用  
3 VUE_APP_BASE_API = '/dev-api'  
4  
5 # 接口服务地址  
6 # VUE_APP_SERVICE_URL = 'http://localhost:8001'  
7 # Easy-Mock 接口服务地址  
8 VUE_APP_SERVICE_URL = 'https://www.easy-mock.com/mock/5d3d62c0250e9354d3640960'
```

2. 然后把 `mxg-mmss\src\api\test.js` 的 `BASE_URL` 改成回最初值 , 防止出错:

8888 端口就是当前项目的端口号 , 这样访问的就是8888下的 `public\db.json`

```
1 const BASE_URL = 'http://localhost:8888'
```

做完后 , 重启下 `npm run serve` , 看下是否正常

7.4.2 配置 Mock.js

1. 当登录成功后 , 响应状态码 `2000` 和 `token` 值。 `token`用来认证

后面请求只要是成功的 , 状态码均为 `2000` , 这个是与后台接口的约定

- 请求URL : `/user/login`
- 请求方式 : `post`



- 描述：登陆认证
- mock.js 配置：

```
1  {
2    "code": 2000, // 状态码
3    "flag": true,
4    "message": '验证成功',
5    "data": {
6      "token": "admin"
7    }
8 }
```

注意：直接复制文档有问题，要手动改为英文空格

2. 添加响应应用用户信息模拟接口

- 请求URL：`/user/info/{token}`
- 请求方式：`get`
- 描述：响应应用用户信息
- mock.js 配置：

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": '成功获取用户信息',
5    "data": {
6      "id | 1-10000": 1,
7      "name": "@cname",
8      "roles": ["manager"]
9    }
10 }
```

未检测出问题

7.5 登录逻辑实现

- 在 `src/api` 下创建 `login.js`，添加内容如下：

```
1 import request from '@/utils/request'
2
3 // 注意：导出的是普通成员函数
4 // 登录
5 export function login(username, password) {
6   return request({
7     url: '/user/login',
8     method: 'post',
9     data: {
10       username,
11       password
12     }
13   })
}
```

```
14  }
15 // 获取用户信息
16 export function getUserInfo(token) {
17   return request({
18     url: `/user/info/${token}` // 注意是反单引号 ``
19     method: 'get'
20   })
21 }
```

- 在 src\views\login\index.vue 的 submitForm 中进行逻辑处理.

1. 导入 import {login, getUserInfo} from '@/api/login'
2. 在 submitForm 函数进行登录判断和获取用户信息

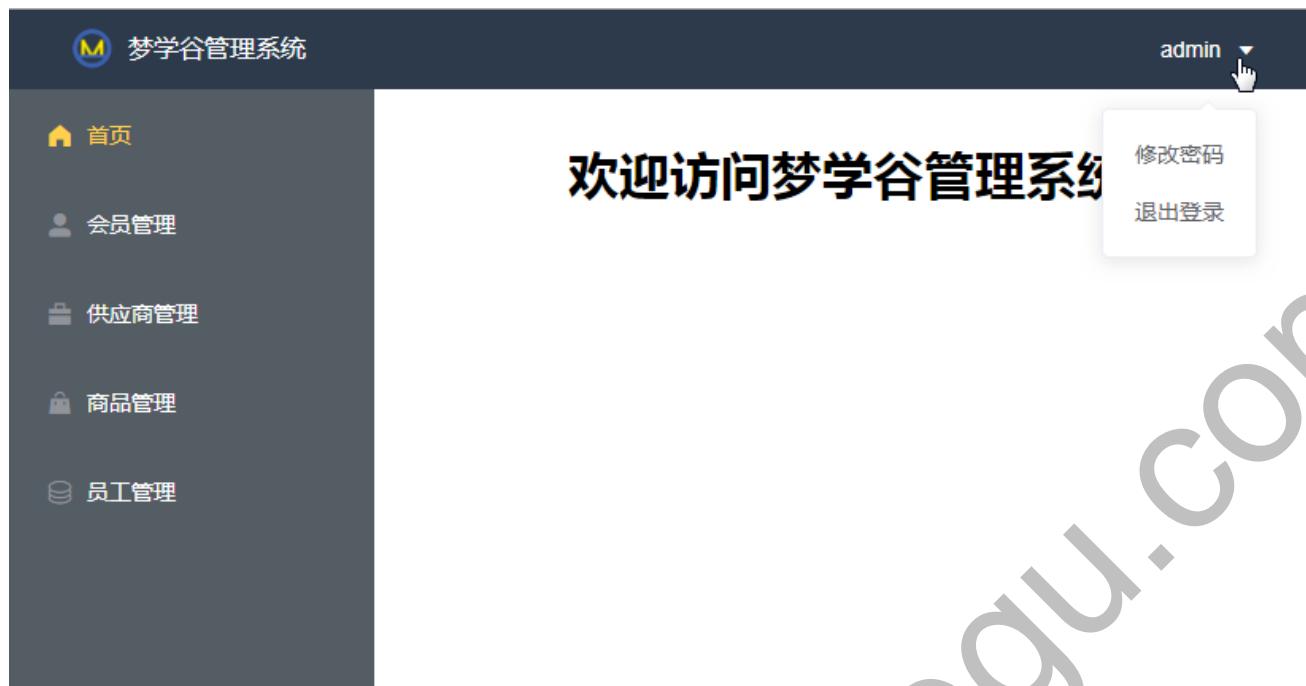
```
1 <script>
2 import {login, getUserInfo} from '@/api/login'
3
4 export default {
5   methods: {
6     // 注意：按钮上调用的函数名要一致 submitForm
7     submitForm(formName) {
8       this.$refs[formName].validate((valid) => {
9         if (valid) {
10           // 验证帐号和密码是否通过 ,
11           login(this.form.username, this.form.password).then(response => {
12             const resp = response.data
13             console.log(resp.code, resp.message, resp.data.token, resp.code === 2000)
14             if (resp.flag) {
15               // 通过，获取用户信息 异步请求
16               getUserInfo(resp.data.token).then(response => {
17                 // 存入session中
18                 const respUser = response.data
19                 if (respUser.flag) {
20                   // 将信息保存到浏览器的 localStorage 中
21                   localStorage.setItem('mgx-msm-user', JSON.stringify(respUser.data))
22                   // 方便后面重新验证
23                   localStorage.setItem('mgx-msm-token', resp.data.token)
24                   // 前往首页
25                   this.$router.push("/")
26                 }else {
27                   // 获取信息失败, 弹出警告
28                   this.$message({
29                     message: respUser.message,
30                     type: "warning"
31                   })
32                 }
33               })
34             }else {
35               // 未通过,弹出警告
36               this.$message({
37                 message: resp.message,
38                 type: "warning"
39               })
40             }
41           })
42         })
43       }
44     }
45   }
46 }
```

```
40      }
41    })
42
43  } else {
44    console.log('error submit!!');
45    return false;
46  }
47 });
48 },
49 },
50 data() {
51   return {
52     form: {},
53     // 表单校验
54     rules: {
55       username: [ // 对应el-form-item 的 prop 值
56         { required: true, message: "帐号不能为空", trigger: "blur" }
57       ],
58       password: [
59         { required: true, message: "密码不能为空", trigger: "blur" }
60       ]
61     }
62   };
63 },
64 };
65 </script>
```

第八章 项目布局和权限&退出

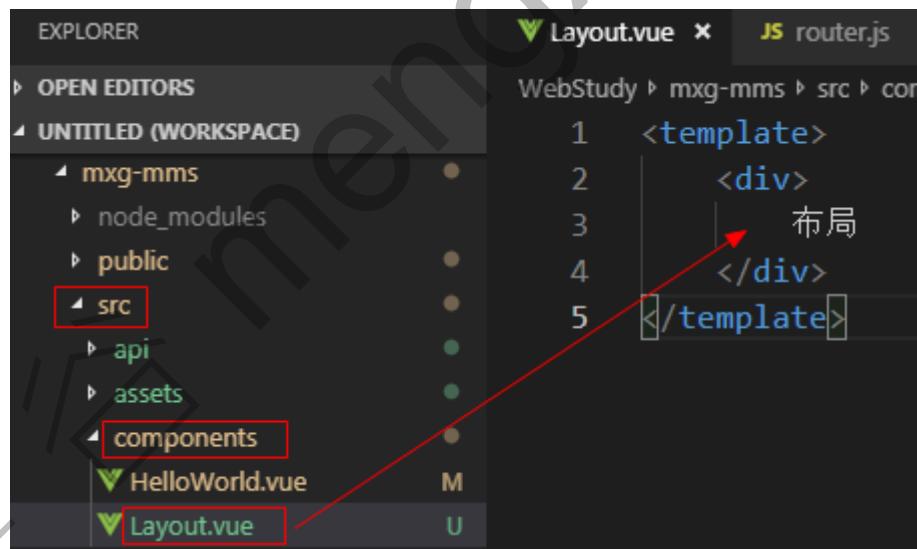
8.1 需求分析

采用后台管理系统常用的上左右布局，分别为：头部，左侧导航、右侧主区域



8.2 路由配置

1. 在 `src\components` 目录下新建布局组件文件 `Layout.vue`，基础布局我们都放到 `components` 下面

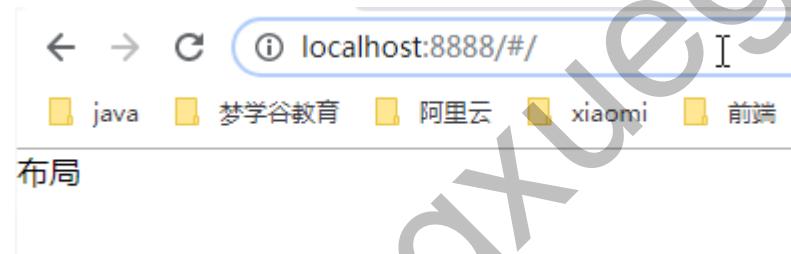


2. 在 `src\router.js` 中增加路由配置，如下：

```
1 import Vue from "vue";
2 import Router from "vue-router";
3 // 默认导入目录下的 index.vue 文件，等价于 ./views/login/index.vue
4 import Login from './views/login'
5 // 布局组件
6 import Layout from './components/Layout.vue'
7
8 Vue.use(Router);
9
10 export default new Router({
```

```
11 routes: [
12 {
13   // 登录页
14   path: '/login',
15   name: 'login', //路由名称
16   component: Login
17 },
18 {
19   // 基础布局
20   path: '/',
21   name: 'layout',
22   component: Layout
23 },
24 ],
25 );
26
```

3. 测试访问 <http://localhost:8888/#/>, 页面显示 布局 两个字



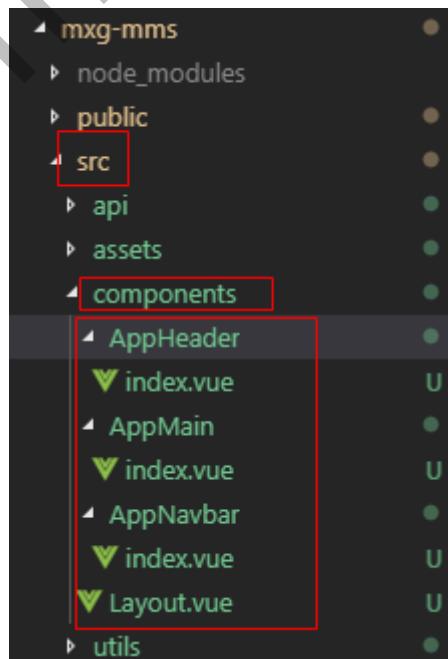
8.3 布局实现

1. 在 `src\components\Layout.vue` 文件中定义布局样式

```
1 <template>
2   <div>
3     <!-- 头部区域 -->
4     <div class="header"> header </div>
5     <!-- 左侧导航区域 -->
6     <div class="navbar"> 左侧导航区域 </div>
7     <!-- 右侧主区域 -->
8     <div class="main"> 右侧主区域 </div>
9   </div>
10  </template>
11
12 <script>
13 export default {
14   components: {},
15 };
16 </script>
17
18 <style scoped>
19 /* 头部区域 */
20 .header {
21   position: absolute;
22   line-height: 50px;
```

```
23 padding: 0px;
24 top: 0px;
25 left: 0px;
26 right: 0px;
27 background-color: #2d3a4b;
28 }
29 /* 左侧导航栏 */
30 .navbar {
31 position: absolute;
32 width: 230px;
33 top: 50px;
34 left: 0px;
35 bottom: 0px;
36 overflow-y: auto;
37 background-color: #545c64;
38 }
39 /* 右侧主区域 */
40 .main {
41 position: absolute;
42 top: 50px;
43 left: 230px;
44 right: 0px;
45 bottom: 0px;
46 padding: 10px;
47 overflow-y: auto;
48 background-color: red;
49 }
50 </style>
```

2. 在 `src\components` 目录下新建3个子目录存放子组件：`AppHeader`、`AppNavbar`、`AppMain`
3. 在新建的3个子目录下分别新建 `index.vue` 文件



4. 将 `Layout.vue` 文件中三大组成部分，分别把对应模板剪切到对应 `index.vue` 中。

注意：不要样式，样式就保存在 Layout.vue 父组件中



- o src\components\AppHeader\index.vue

```
1 <template>
2   <div class="header"> header</div>
3 </template>
4 <script>
5 export default {
6 }
7 </script>
8 <style scoped>
9 </style>
```

- o src\components\AppNavbar\index.vue

```
1 <template>
2   <div class="main"> 左侧导航区域 </div>
3 </template>
4 <script>
5 export default {
6 }
7 </script>
8 <style scoped>
9 </style>
```

- o src\components\AppMain\index.vue

```
1 <template>
2   <div class="main"> 右侧主区域 </div>
3 </template>
4 <script>
5 export default {
6 }
7 </script>
8 <style scoped>
9 </style>
```

5. 在 Layout.vue 中使用 components 选项将3个组件作为子组件引入。

```
1 <template>
2   <div>
3     <!-- 头部区域 -->
4     <app-header/>
5     <!-- 左侧导航区域 -->
6     <app-navbar/>
7     <!-- 右侧主区域 -->
8     <app-main/>
9   </div>
10 </template>
11
12 <script scoped>
```



```
13 // 1. 导入子组件
14 import AppHeader from './AppHeader'
15 import AppNavbar from './AppNavbar'
16 import AppMain from './AppMain'
17
18 export default {
19   // 2. 注册为子组件
20   components: {
21     AppHeader, AppNavbar, AppMain
22   }
23 }
24 </script>
25
26 <style scoped>
27 /* 头部区域 */
28 .header {
29   position: absolute;
30   line-height: 50px;
31   padding: 0px;
32   top: 0px;
33   left: 0px;
34   right: 0px;
35   background-color: #2d3a4b;
36 }
37 /* 左侧导航栏 */
38 .navbar {
39   position: absolute;
40   width: 230px;
41   top: 50px;
42   left: 0px;
43   bottom: 0px;
44   overflow-y: auto;
45   background-color: #545c64;
46 }
47 /* 右侧主区域 */
48 .main {
49   position: absolute;
50   top: 50px;
51   left: 230px;
52   right: 0px;
53   bottom: 0px;
54   padding: 10px;
55   overflow-y: auto;
56   background-color: red;
57 }
58 </style>
```



8.4 头部区域实现



1. 头部左侧有显示 Logo 与 标题 , 实现如下

```
1 <template>
2   <div class="header">
3     <!-- 头部左侧Logo和标题 -->
4     <a href="#">
5       
6       <span class="company">梦学谷管理系统</span>
7     </a>
8   </div>
9 </template>
10 <style scoped>
11 .logo {
12   vertical-align: middle;
13   /* 上右下左 */
14   padding: 0 10px 0 40px;
15 }
16 .company {
17   position: absolute;
18   color: white;
19 }
20 </style>
```

2. 头部右侧下拉菜单, 实现如下 :

下拉菜单参考 : <https://element.eleme.cn/#/zh-CN/component/dropdown#zhi-ling-shi-jian>

```
1 <template>
2   <div class="header">
3     <!-- 头部左侧Logo和标题 -->
4     <a href="#">
5       
6       <span class="company">梦学谷管理系统</span>
7     </a>
8
9     <!-- 头部右侧下拉菜单 -->
10    <el-dropdown @command="handleCommand">
11      <span class="el-dropdown-link">
12        admin<i class="el-icon-arrow-down el-icon--right"></i>
13      </span>
14      <el-dropdown-menu slot="dropdown">
15        <el-dropdown-item command="a">修改密码</el-dropdown-item>
16        <el-dropdown-item command="b">退出系统</el-dropdown-item>
17      </el-dropdown-menu>
18    </el-dropdown>
19  </div>
20 </template>
21 <script>
22   export default {
23     data() {
24       return {
25         value: 'admin'
26       }
27     },
28     methods: {
29       handleCommand(command) {
30         if (command === 'a') {
31           alert('修改密码')
32         } else if (command === 'b') {
33           alert('退出系统')
34         }
35       }
36     }
37   }
38 </script>
```

```
18    </el-dropdown>
19    </div>
20  </template>
21
22  <script>
23  export default {
24    methods: {
25      handleCommand(command) {
26        this.$message('click on item ' + command);
27      }
28    }
29  }
30 </script>
31
32 <style scoped>
33 .logo {
34   vertical-align: middle;
35   /* 上右下左 */
36   padding: 0 10px 0 40px;
37 }
38 .company {
39   position: absolute;
40   color: white;
41 }
42 /* 下拉菜单 */
43 .el-dropdown {
44   float: right;
45   margin-right: 40px;
46 }
47
48 .el-dropdown-link {
49   cursor: pointer;
50   color: #fff;
51 }
52 </style>
```

8.5 左侧导航实现

1. 效果图，有下图5个导航菜单

左侧导航参考：<https://element.eleme.cn/#/zh-CN/component/menu#ce-lan>



2. 导航组件实现

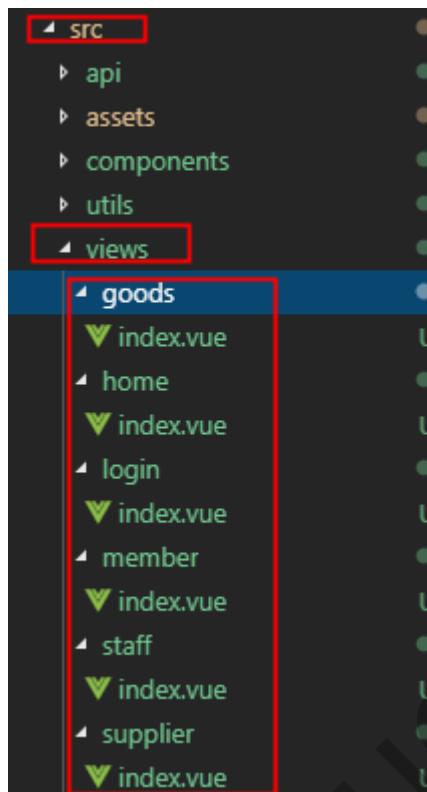
- el-menu 标签上有个 `:router="true"` 属性开启导航路由功能，开启后 el-menu-item 的 `index` 属性指定路由地址。
- `default-active` 属性默认选中哪个菜单，注意 `v-bind:default-active` 才可以指定表达式

```
1 <template>
2   <div class="navbar">
3     <!-- :router 开启路由功能,则 index指定路由地址
4       default-active 默认选中哪个菜单,选中为黄色,
5         注意 v-bind:default-active 才可以指定表达式 -->
6     <el-menu
7       :router="true"
8       :default-active="$route.path"
9       class="el-menu-vertical-demo"
10      background-color="#545c64"
11      text-color="#fff"
12      active-text-color="#ffd04b">
13       <el-menu-item index="/home">
14         <i class="el-icon-s-home"></i>
15         <span slot="title">首页</span>
16       </el-menu-item>
17       <el-menu-item index="/member/">
```

```
18      <i class="el-icon-user-solid"></i>
19      <span slot="title">会员管理</span>
20  </el-menu-item>
21  <el-menu-item index="/supplier/">
22      <i class="el-icon-s-cooperation"></i>
23      <span slot="title">供应商管理</span>
24  </el-menu-item>
25  <el-menu-item index="/goods/">
26      <i class="el-icon-s-goods"></i>
27      <span slot="title">商品管理</span>
28  </el-menu-item>
29  <el-menu-item index="/staff/">
30      <i class="el-icon-user"></i>
31      <span slot="title">员工管理</span>
32  </el-menu-item>
33  </el-menu>
34 </div>
35 </template>
36
37 <style scoped>
38 .el-menu {
39   border-right: none;
40 }
41 </style>
```

8.6 所有菜单路由配置

- 在 `src\views` 目录下创建 `home`、`member`、`supplier`、`goods`、`staff` 目录，并且每个目录下创建一个 `index.vue`



2. 配置所有菜单路由信息，在 router.js 中配置如下：

- 先采用 import 导入组件。再路由引用组件。

```
1 import Vue from "vue";
2 import Router from "vue-router";
3 // 默认导入目录下的 index.vue 文件，等价于 ./views/login/index.vue
4 import Login from './views/login'
5 // 布局组件
6 import Layout from './components/Layout.vue'
7 import Home from './views/home'
8 import Member from './views/member'
9 import Supplier from './views/supplier'
10 import Goods from './views/goods'
11 import Staff from './views/staff'
12
13 Vue.use(Router);
14
15 export default new Router({
16   routes: [
17     {
18       // 登录页
19       path: '/login',
20       name: 'login', // 路由名称
21       component: Login
22     },
23     {
24       // 基础布局
25       path: '/',
26       name: 'layout',
27       component: Layout,
28     }
29   ]
30 })
```



```
28      redirect: '/home',// 重定向到子路由
29      children: [
30        {
31          path: '/home',
32          component: Home,
33          meta: {title: '首页'}
34        }
35      ]
36    },
37    {
38      // 会员管理
39      path: '/member',
40      component: Layout,
41      children: [
42        {
43          path: '/',
44          component: Member,
45          meta: { title: '会员管理' }
46        }
47      ]
48    },
49    {
50      // 供应商管理
51      path: '/supplier',
52      component: Layout,
53      children: [
54        {
55          path: '/',
56          component: Supplier,
57          meta: { title: '供应商管理' }
58        }
59      ]
60    },
61    {
62      // 商品管理
63      path: '/goods',
64      component: Layout,
65      children: [
66        {
67          path: '/',
68          component: Goods,
69          meta: { title: '商品管理' }
70        }
71      ]
72    },
73    {
74      // 员工管理
75      path: '/staff',
76      component: Layout,
77      children: [
78        {
79          path: '/',
80          component: Staff,
```

禁
止
复
制

```

81     meta: { title: '员工管理' },
82   }
83 ]
84 },
85 ],
86 });
87

```

8.7 右侧主区域实现

功能：当点击左侧菜单项后，对应路由组件渲染在右侧主区域中。所以要在右侧指定组件渲染出口。



序号	会员卡号	会员姓名	会员生日	手机号码	可用积分	开卡金额	支付类型	会员地址
1	000001	小梦	1987-01-01	18888888888	98	300	支付宝	北京天安门广场1号门
2	000002	小梦2	1987-01-01	18888888888	98	300	微信	北京天安门广场1号门
3	000003	小梦3	1987-01-01	18888888888	98	300	现金	北京天安门广场1号门
4	000004	小梦4	1987-01-01	18888888888	98	300	支付宝	北京天安门广场1号门

1. 主区域中除了首页没有横向指示导航，其他模块中都有，所以先渲染出面包屑。

参考 Breadcrumb 面包屑：

<https://element.eleme.cn/#/zh-CN/component/breadcrumb#tu-biao-fen-ge-fu>

- 在 Layout.vue 中的移除右侧主区域背景色，.main 中指定的。

```

1 <template>
2 <div class="main">
3   <el-breadcrumb separator-class="el-icon-arrow-right">
4     <el-breadcrumb-item class="line" :to="{ path: '/' }">首页</el-breadcrumb-item>
5   </el-breadcrumb>
6   <!-- 子路由渲染出口 -->
7   <router-view></router-view>
8 </div>
9 </template>
10
11 <script>
12 </script>
13
14 <style scoped>
15 .el-breadcrumb {

```



```
16 height: 10px;
17 padding: 20px;
18 border-radius: 4px;
19 /* 投影 */
20 box-shadow: 0 2px 12px 0 rgba(0, 0, 0, 0.1);
21 }
22 .line {
23 border-left: 3px solid #31c17b;
24 padding-left: 10px;
25 }
26 </style>
```

2. 将 横向指示导航 抽取为子组件 `AppMain\Link.vue` ,

其中获取导航名称使用 `$route.meta.title` , 路由地址使用 `$route.path`

```
1 <template>
2   <el-breadcrumb separator-class="el-icon-arrow-right">
3     <el-breadcrumb-item class="line" :to="{path: $route.path}">{{ $route.meta.title }}</el-breadcrumb-
item>
4   </el-breadcrumb>
5 </template>
6 <style scoped>
7 .el-breadcrumb {
8   height: 10px;
9   padding: 20px;
10  border-radius: 4px;
11  /* 投影 */
12  box-shadow: 0 2px 12px 0 rgba(0, 0, 0, 0.1);
13 }
14 .line {
15   border-left: 3px solid #31c17b;
16   padding-left: 10px;
17 }
18 </style>
```

3. 在 `AppMain\index.vue` 中导入 `Link.vue`

- 使用 `v-show` 判断路由为 `/home` 时不显示横向指示导航。

```
1 <template>
2   <div class="main">
3     <!-- v-show 当路由为 /home 不显示 -->
4     <app-link v-show="$route.path != '/home'" />
5     <!-- 子路由渲染出口 -->
6     <router-view></router-view>
7   </div>
8 </template>
9 <script>
10 import AppLink from './Link.vue'
11 export default {
12   components: { AppLink }
13 }
```

```
14  </script>
15  <style scoped>
16  </style>
```

8.8 首页 Home 组件

在 `src\views\login\index.vue` 中简单添加标题测试，如需其他功能自行扩展。

```
1 <template>
2   <div>
3     <h1>欢迎访问梦学谷管理系统</h1>
4   </div>
5 </template>
6 <script>
7 </script>
8 <style scoped>
9   div {
10     text-align: center;
11   }
12 </style>
```

8.9 退出系统

8.9.1 EasyMock 添加退出模拟接口

- 请求URL : `/user/logout`
- 请求方式 : `post`
- 提交参数: `data: {token}`
- 描述 : 退出系统
- mock.js 配置 :

```
1 {
2   "code": 2000, // 状态码
3   "flag": true,
4   "message": '退出成功'
5 }
```

8.9.2 定义退出方法 logout

在 `src\api\login.js` 添加退出方法 `logout`

```
1 export function logout(token) {  
2     return request({  
3         url: '/user/logout',  
4         method: 'post',  
5         data: {  
6             token  
7         }  
8     })  
9 }
```

8.9.3 组件中实现退出功能

在 src\components\AppHeader\index.vue 实现退出功能

```
1 <script>  
2 import {logout} from '@/api/login'  
3  
4 export default {  
5     methods: {  
6         handleCommand(command) {  
7             switch (command) {  
8                 case 'a':  
9                     // 修改密码  
10                    break;  
11                 case 'b':  
12                     // 退出系统  
13                     logout(localStorage.getItem(localStorage.getItem('mgx-msm-token'))).then(response => {  
14                         // 清除本地数据  
15                         localStorage.removeItem('mgx-msm-token')  
16                         localStorage.removeItem('mgx-msm-user')  
17                         this.$router.push('/login')  
18                     })  
19                     break;  
20                 default:  
21                     break;  
22             }  
23         }  
24     }  
25 }  
26 </script>
```

8.10 权限校验

需求：当用户未登录时，不让访问非登录页面，应该回到登录页面进行登录。

实现：采用 Vue Router 中的路由前置钩子函数(`beforeEach`)，在前置中根据路由地址校验是否此路由是否允许访问。

路由钩子函数参考：<https://router.vuejs.org/zh/guide/advanced/navigation-guards.html>



1. 新建 `src\permission.js` 权限校验文件，代码如下：

```
1  /*
2   * 权限校验：
3   * 通过router路由前置钩子函数 beforeEach() ,
4   * 在跳转路由前进行拦截判断是否已经登录 ,
5   * 如果已登录，则进行路由跳转，如果没有则回到登录页
6   */
7  import router from './router'
8  import {getUserInfo} from './api/login'
9  /*
10   * 前置路由钩子函数
11   * to : 即将要进入的目标路由对象
12   * from : 当前导航正要离开的路由对象
13   * next : 调用该方法，进入目标路由
14  */
15 router.beforeEach((to, from, next) => {
16   // 获取 token
17   const token = localStorage.getItem('mgx-msm-token')
18
19   // 没有 token
20   if (!token) {
21     // 不允许访问其他路由，回到登录页
22     if (to.path !== '/login') {
23       next({path: '/login'})
24     } else {
25       next()
26     }
27   } else {
28     // 有 token
29     if (to.path === '/login') {
30       // 进入目标路由
31       next()
32     } else {
33       // 有token, 看下是否有用户信息
34       const userInfo = localStorage.getItem('mgx-msm-user')
35       if (userInfo) {
36         // 有信息, 进入目标路由
37         next()
38       } else {
39         // 通过token获取用户信息
40         getUserInfo(token).then((response) => {
41           const resp = response.data
42           console.log('prer', resp)
43           if (resp.flag) {
44             // 获取到, 保存数据
45             localStorage.setItem('mgx-msm-user', JSON.stringify(resp.data))
46             next()
47           } else {
48             // 未获取到,重新登录
49             next({path: '/login'})
50           }
51         })
52       }
53     }
54   }
55 })
```

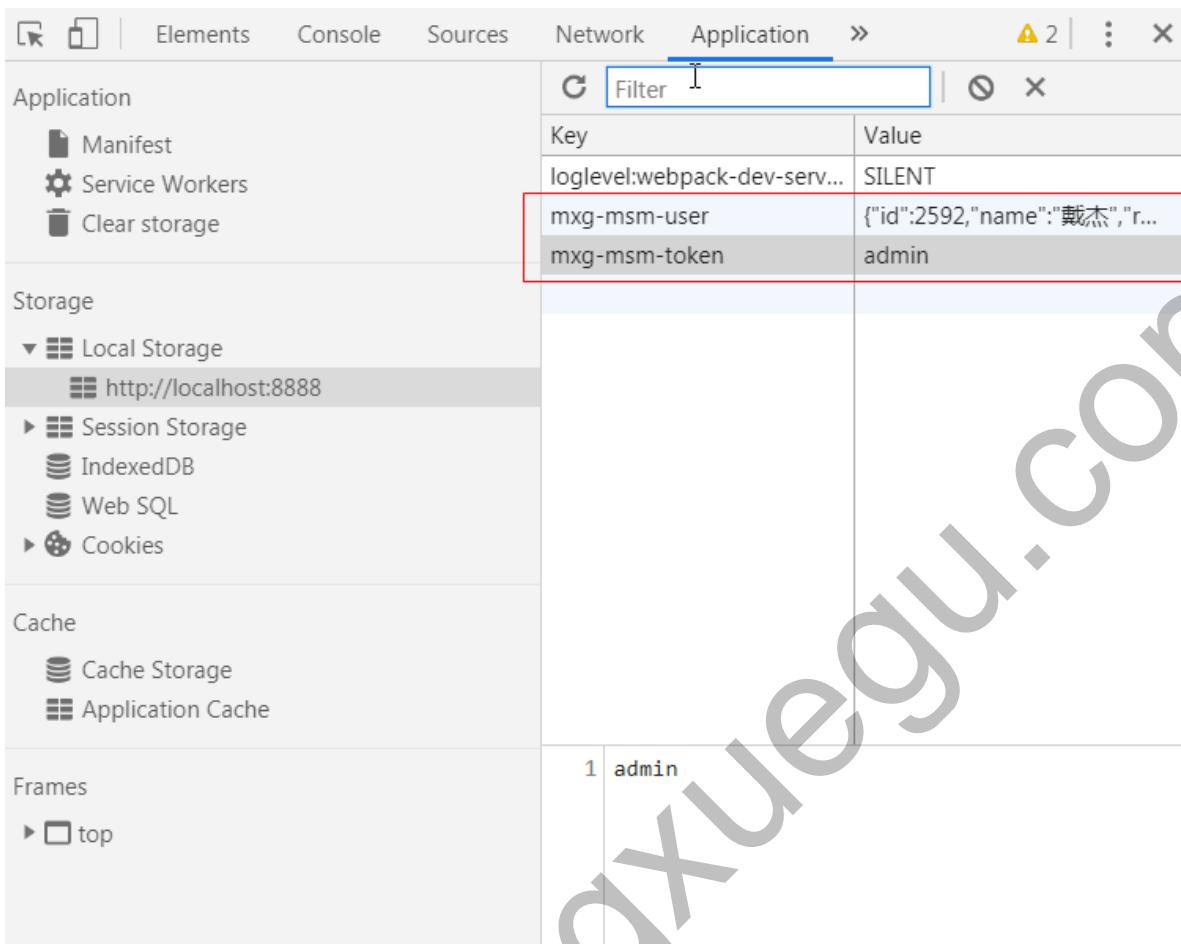
```
52      }
53    }
54  }
55 })
```

2. 将权限添加到全局民中

```
1 import Vue from "vue";
2 // 注意引入在 Vue的下面
3 // ElementUI 组件库
4 import ElementUI from 'element-ui';
5 import 'element-ui/lib/theme-chalk/index.css';
6 import App from "./App.vue";
7 import router from "./router";
8
9 // 使用 ElementUI
10 Vue.use(ElementUI);
11
12 // 权限拦截
13 import './permission'
14
15 Vue.config.productionTip = process.env.NODE_ENV === 'production';
16 console.log(process.env.VUE_APP_SERVICE_URL) // 开发环境 development, 生产环境 production
17
18 new Vue({
19   router,
20   render: h => h(App)
21 }).$mount("#app");
22
```

3. 测试,

1. 清空浏览器保存的用户数据和token



The screenshot shows the Chrome DevTools Application tab. On the left sidebar, under 'Storage', 'Local Storage' is expanded, showing an item for 'http://localhost:8888'. The main area displays a table of stored data:

Key	Value
loglevel:webpack-dev-serv...	SILENT
mxg-msm-user	{"id":2592,"name":"戴杰","r...
mxg-msm-token	admin

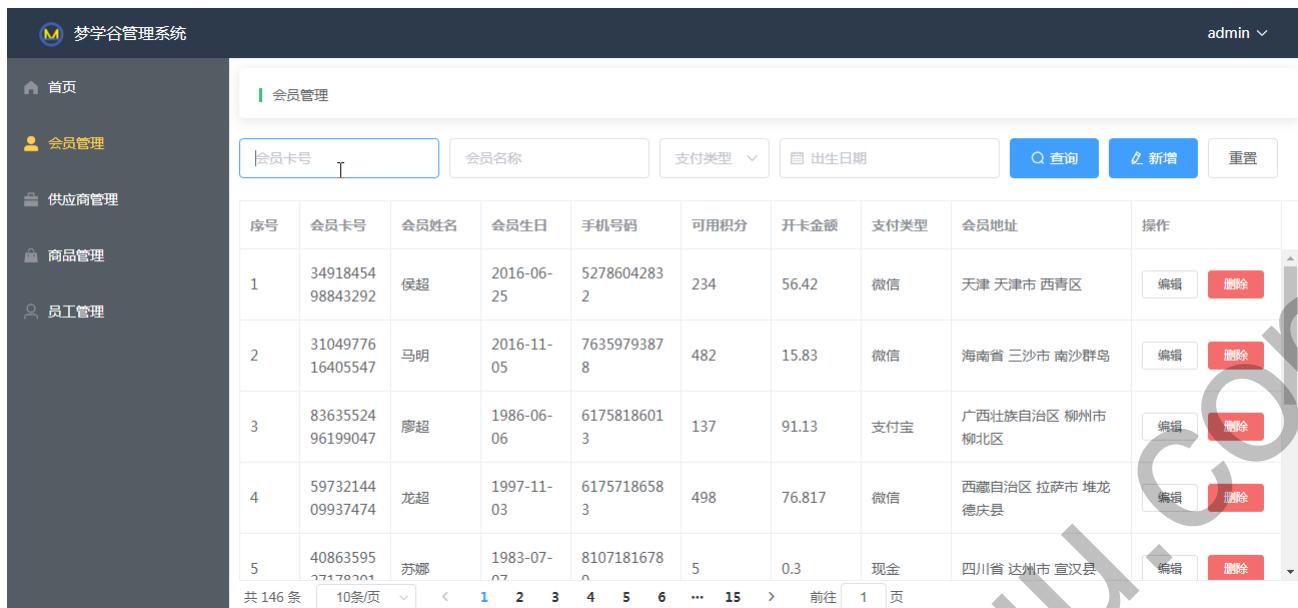
A red box highlights the last two rows of the table.

2. 在未登录情况下访问 <http://localhost:8888/#/home>, 会回到 登录
3. 只有登录后, 才可以访问首页

第九章 会员管理-列表查询

9.1 需求分析

会员管理主要针对充值会员进行管理，首先开发会员管理模块中的列表功能，包含条件查询、下拉框、日期功能、数据列表、分页。



序号	会员卡号	会员姓名	会员生日	手机号码	可用积分	开卡金额	支付类型	会员地址	操作
1	34918454 98843292	侯超	2016-06- 25	5278604283 2	234	56.42	微信	天津 天津市 西青区	<button>编辑</button> <button>删除</button>
2	31049776 16405547	马明	2016-11- 05	7635979387 8	482	15.83	微信	海南省 三沙市 南沙群岛	<button>编辑</button> <button>删除</button>
3	83635524 96199047	廖超	1986-06- 06	6175818601 3	137	91.13	支付宝	广西壮族自治区 柳州市 柳北区	<button>编辑</button> <button>删除</button>
4	59732144 09937474	龙超	1997-11- 03	6175718658 3	498	76.817	微信	西藏自治区 拉萨市 堆龙 德庆县	<button>编辑</button> <button>删除</button>
5	40863595 77170201	苏娜	1983-07- 07	8107181678 0	5	0.3	现金	四川省 达州市 宣汉县	<button>编辑</button> <button>删除</button>

9.2 会员数据列表

9.2.1 EasyMock 添加数据列表模拟接口

- 请求URL : /member/list
- 请求方式 : get
- 描述 : 会员数据列表
- mock.js 配置 :

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": "查询成功",
5    "data|20": [
6      "id|+1": 10,
7      "cardNum": "@integer(10000)", //大于1000的正整数
8      "name": "@cname",
9      "birthday": "@date",
10     "phone|11": "@integer(0,9)", // 11个数字0-9间的数字
11     "integral": "@integer(0, 500)",
12     "money": "@float(0, 1000, 1, 3)", // 0-1000小数,1-3位小数位
13     "payType|1": ['1','2','3','4'], // 4选 其1
14     "address": "@county(true)"
15   ]
16 }
```

9.2.2 Api 调用接口

1. 在 src/api 下创建 member.js , 调用接口代码如下:

```
1 import request from '@/utils/request"
2
3 export default {
4   // 获取会员列表
5   getList(){
6     return request({
7       url:'/member/list',
8       method:'get'
9     })
10  },
11 }
```

2. 在 `src\views\member\index.vue` 中, 添加 JS 代码如下 :

```
1 <template>
2 </template>
3 <script>
4 import memberApi from '@/api/member'
5
6 export default {
7   data(){
8     return {
9       list: []
10    }
11  },
12  // 钩子函数获取数据
13  created (){
14    this.fetchData()
15  },
16  methods: {
17    fetchData () {
18      memberApi.getList().then(response => {
19        this.list = response.data.data
20      })
21    }
22  },
23 },
24 </script>
```

9.2.3 列表模板

1. 修改 `src\views\member\index.vue` , 编写模板代码 :

表格采用的是固定表头 , 参考 :

<https://element.eleme.cn/#/zh-CN/component/table#gu-ding-biao-tou>

```
1 <template>
2 <div>
3   <!-- 列表-->
4   <!-- :data 绑定渲染的数据 ; border: 边框; -->
```



```
5   <el-table :data="list" height="380" border style="width: 100%">
6     <!--type="index" 获取索引值，从1开始； label 显示的标题; prop 数据字段名； width 列的宽度 -->
7     <el-table-column type="index" label="序号" width="60"></el-table-column>
8     <el-table-column prop="cardNum" label="会员卡号"></el-table-column>
9     <el-table-column prop="name" label="会员姓名" ></el-table-column>
10    <el-table-column prop="birthday" label="会员生日"></el-table-column>
11    <el-table-column prop="phone" label="手机号码" width="110"></el-table-column>
12    <el-table-column prop="integral" label="可用积分"></el-table-column>
13    <el-table-column prop="money" label="开卡金额"></el-table-column>
14    <el-table-column prop="payType" label="支付类型" ></el-table-column>
15    <el-table-column prop="address" label="会员地址" width="180"></el-table-column>
16    <el-table-column label="操作" width="150">
17      <template slot-scope="scope">
18        <el-button size="mini" @click="handleEdit(scope.row.id)">编辑</el-button>
19        <el-button size="mini" type="danger" @click="handleDelete(scope.row.id)">删除</el-button>
20      </template>
21    </el-table-column>
22  </el-table>
23 </div>
24 </template>
```

2. 在 methods 中添加 handleEdit 编辑和 handleDelete 删除方法，后面需要使用。

```
1 methods: {
2   handleEdit(id) {
3     console.log('编辑')
4   },
5   handleDelete(id) {
6     console.log('删除')
7   },
8   ...
9   ...
10  ...
11 }
```

9.2.4 过滤器实现数据转换

1. 渲染后发现支付类型是编号，应该将类型编号 转为名称。通过 filters 选项来定义过滤器来实现转换。
- JS 脚本处声明一个全局支付类型数组，组件对象的 filters 选项指定转换规则

```
1 <script>
2 import memberApi from '@/api/member'
3
4 // 支付类型
5 const payTypeOptions = [
6   { type: '1', name: '现金' },
7   { type: '2', name: '微信' },
8   { type: '3', name: '支付宝' },
9   { type: '4', name: '银行卡' }
```

```
10 ]
11
12 export default {
13   ..
14   ..
15
16   filters: {
17     // filters 中 this 指向的不是vue实例, 所有无法直接获取 data 中的数据
18     payTypeFilter (type) {
19       // 全局的 payTypeOptions , 返回一个满足条件的对象
20       const obj = payTypeOptions.find(obj => obj.type === type)
21       // 非空返回类型名称
22       return obj ? obj.name: null
23     }
24   },
25 }
```

- 修改 模板代码

```
1 <el-table-column prop="payType" label="支付类型" >
2   <template slot-scope="scope">
3     <span>{{ scope.row.payType | payTypeFilter }}</span>
4   </template>
5 </el-table-column>
```

9.3 分页功能实现

为列表数据添加分页功能，使用分页组件完成分页功能。

参考：<https://element.eleme.cn/#/zh-CN/component/pagination>

序号	会员卡号	会员姓名	会员生日	手机号码	可用积分	开卡金额	支付类型	会员地址
2	000002	小梦2	1987-01-01	18888888888	98	300	微信	北京天安门广场1号正大门
3	000003	小梦3	1987-01-01	18888888888	98	300	现金	北京天安门广场1号正大门
4	000004	小梦4	1987-01-01	18888888888	98	300	支付宝	北京天安门广场1号正大门
5	000005	小梦2	1987-01-01	18888888888	98	300	微信	北京天安门广场1号正大门
6	000006	小梦2	1987-01-01	18888888888	98	300	现金	北京天安门广场1号正大门

共 400 条 100条/页 < 1 2 3 4 > 前往 页

9.3.1 EasyMock 添加会员分页模拟接口

- 请求URL : /member/list/search/{page}/{size}

page 当前要查询的页码 , size 每页显示条数 , 通过这两个值 , 可查询出当前请求要响应的数据。

- 请求方式 : post

- 描述 : 会员列表数据分页

- mock.js 配置 :

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": "查询成功",
5    "data": {
6      "total": "@integer(100,200)", // 查询出来的总记录数
7      "rows|10": [{} // 返回当前页的记录数 10 条, 即每页显示 10 条记录
8        "id|+1": 10,
9        "cardNum": "@integer(10000)", // 大于1000的正整数
10       "name": "@cname",
11       "birthday": "@date",
12       "phone|11": "@integer(0,9)", // 11个位数字
13       "integral": "@integer(0, 500)",
14       "money": "@float(0, 100, 1, 3)", // 0-1000小数,1-3位小数位
15       "payType|1": ["1", "2", "3", "4"],
16       "address": "@county(true)"
17     ]}
18   }
19 }
```

9.3.2 Api 调用接口

- 修改 src\api 下的 member.js , 在导出的默认对象中 , 增加分页查询方法

```
1  ,
2  // 分页查询,
3  // page当前页码, size 每页显示条数, searchMap 条件
4  // 后台要通过 page 和 size 统计本次请求响应的数据
5  search(page, size, searchMap) {
6    return request({
7      url: `/member/list/search/${page}/${size}`,
8      method: 'post',
9      data: searchMap
10     })
11   }
```

- 在 src\views\member\index.vue 中, 添加 JS 代码如下 :

声明分页属性 , 调用 getList 改为调用 search 方法

```
1
```

```
2 <script>
3 import memberApi from '@/api/member'
4 ...
5 ...
6
7 export default {
8   data() {
9     return {
10       list: [],
11       total: 0, // 总记录数
12       currentPage: 1, // 当前页, 默认第1页
13       pageSize: 10, // 每页显示条数 , 10条
14       searchMap: {}, // 条件查询绑定条件值
15     }
16   },
17
18   created () {
19     this.fetchData()
20   },
21
22   methods: {
23     fetchData () {
24       // 调用分页方法查询, 不要少了 this
25       memberApi.search(this.currentPage, this.pageSize, this.searchMap).then(response => {
26         const reqs = response.data
27         this.total = reqs.data.total
28         this.list = reqs.data.rows
29         console.log(this.total, this.list)
30       })
31       /* memberApi.getList().then(response => {
32         this.list = response.data.data
33       }) */
34     }
35   },
36
37 ...
38 ...
39 }
40 </script>
```

○ 测试查看控制台否打印

9.3.3 分页模板

1. 修改 `src\views\member\index.vue` , `template` 标签中添加页面模板代码 :

注意 : 添加在 `div` 里面。因为 `template` 里面只能有唯一根节点

`current-change` 和 `size-change` 事件都是调用 `fetchData`

```

1 <!-- 分页，添加在div里面 -->
2 <el-pagination
3   @size-change="fetchData"
4   @current-change="fetchData"
5   :current-page="currentPage"
6   :page-sizes="[10, 20, 50]"
7   :page-size="pageSize"
8   layout="total, sizes, prev, pager, next, jumper"
9   :total="total">
10 </el-pagination>

```

Attributes 属性说明：

参数	说明	类型	可选值	默认值
page-sizes	每页显示条数下拉框的选项设置	number[]	—	[10, 20, 30, 40, 50, 100]
page-size	每页显示条目个数	number	—	10
total	总条目数	number	—	—
current-page	当前页数，支持.sync 修饰符	number	—	1
layout	当前分页组件布局，子组件名用逗号分隔	String	sizes, prev, pager, next, jumper (跳页输入框), >, total, slot	'prev, pager, next, jumper, -, >, total'

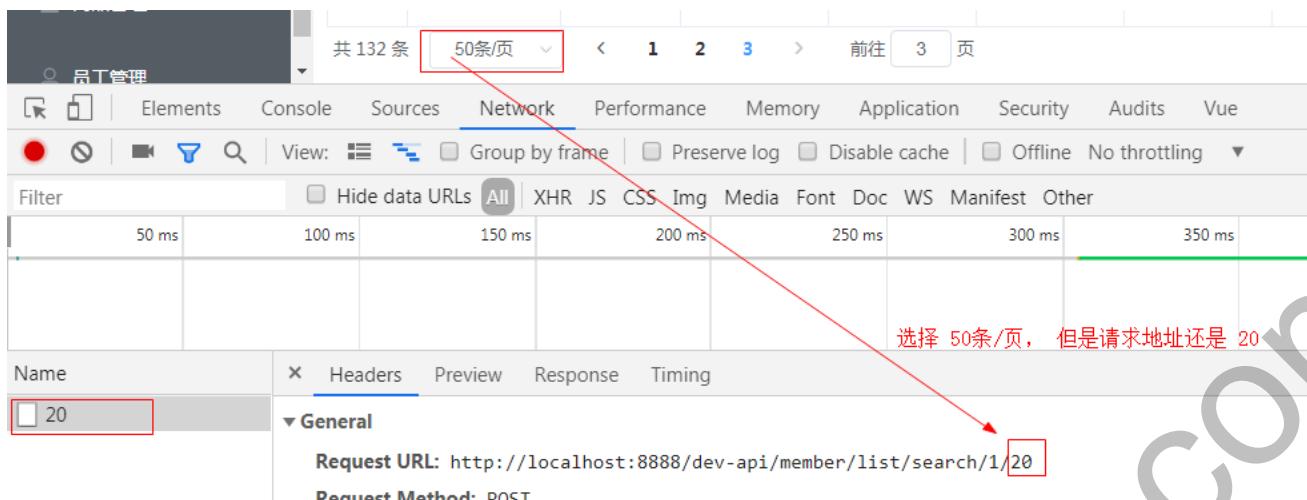
Events 事件说明：

事件名称	说明	回调参数
size-change	pageSize 改变时会触发	每页条数
current-change	currentPage 改变时会触发	当前页

此章节做完能实现功能，但是会有报错。video代码中未修复报错。另教程中先有分页，后有条件查询。但视频中顺序相反。

9.3.4 解决动态页码和当前页

问题：上面当你改变页码后，请求URL的 {page}/{size} 不会变，一直是初始值，那是因为没有监听到改变的值。



选择 50条/页，但是请求地址还是 20

解决：

1. 将 @size-change 和 @current-change 事件处理的函数名改变为 handleSizeChange 和 handleCurrentChange

```
1 <el-pagination
2   @size-change="handleSizeChange"
3   @current-change="handleCurrentChange"
```

2. 在 methods 中添加函数，

```
1 methods: {
2   handleSizeChange(val) {
3     this.pageSize = val
4     this.fetchData()
5   },
6   handleCurrentChange(val) {
7     this.currentPage = val
8     this.fetchData()
9   },

```

3. 测试可通过

9.4 条件查询实现

在列表上方添加查询功能。

Form 表单参考：<https://element.eleme.cn/#/zh-CN/component/form#xing-nei-biao-dan>

日期参考：<https://element.eleme.cn/#/zh-CN/component/date-picker>



会员管理

会员卡号	I	会员名称	支付类型	出生日期	查询	重置
------	---	------	------	------	----	----

序号	会员卡号	会员姓名	会员生日	手机号码	可用积分	开卡金额	支付类型	会员地址	操作
5	715591609 3057145	贺洋	2015-03-07	5842270139 8	150	6.533	支付宝	重庆 重庆市 长寿区	
6	184367604 2493665	金敏	1972-09-15	1285515844 2	425	62.246	现金	贵州省 铜仁市 江口县	

9.4.1 条件查询模板

1. 修改 `src\views\member\index.vue` , 增加条件查询模板代码 :

```
1 <!-- 条件查询。 inline 属性设置表单变为行内的表单域 -->
2 <el-form ref="searchForm" :inline="true" :model="searchMap" style="margin-top: 20px;" >
3 <!-- 有prop才可以重置 -->
4 <el-form-item prop="cardNum">
5   <el-input v-model="searchMap.cardNum" placeholder="会员卡号" style="width: 200px;" ></el-input>
6 </el-form-item>
7 <el-form-item prop="name">
8   <el-input v-model="searchMap.name" placeholder="会员名称" style="width: 200px;" ></el-input>
9 </el-form-item>
10 <el-form-item prop="payType">
11   <el-select v-model="searchMap.payType" placeholder="支付类型" style="width: 110px;" >
12     <!--key 标识 , label 下拉显示的文本 , value 表单值 -->
13     <el-option
14       v-for="option in payTypeOptions"
15       :key="option.type"
16       :label="option.name"
17       :value="option.type"
18     />
19   </el-select>
20 </el-form-item>
21 <el-form-item prop="birthday">
22   <!-- value-format 是指定绑定值的格式 -->
23   <el-date-picker value-format="yyyy-MM-dd" v-model="searchMap.birthday" type="date" placeholder="出生日期"></el-date-picker>
24 </el-form-item>
25 <el-form-item>
26   <el-button type="primary" icon="el-icon-search" @click="fetchData">查询</el-button>
27 </el-form-item>
28 </el-form>
```

2. 在 data 选项声明 `payTypeOptions` 属性用于下拉框 , 并将全局的赋值给它

```
1 data() {
2   return {
```

```
3   list: [],
4   total: 0, // 总记录数
5   currentPage: 1, // 当前页, 默认第1页
6   pageSize: 10, // 每页显示条数 , 10条
7   searchMap: { // 条件查询绑定的条件值
8     cardNum: '',
9     name: '',
10    payType: '',
11    birthday: ''
12  },
13  payTypeOptions, // 支付类型 , ES6语法
14 }
15 },
```

- 测试

9.4.2 重置功能

- 在 条件查询区域添加一个 重置 按钮

```
1 <el-form-item>
2   <el-button type="primary" icon="el-icon-search" @click="fetchData">查询</el-button>
3   <el-button @click="resetForm('searchForm')">重置</el-button>
4 </el-form-item>
```

- methods 选项中添加重置处理方法 resetForm

注意: 在 el-form-item 标签属性 prop 上, 指定了字段名, 重置才会生效

```
1 // 表单重置 ,
2 // 在 el-form-item 标签属性 prop 上, 指定了字段名, 重置才会生效
3 resetForm(formName) {
4   this.$refs[formName].resetFields();
5 },
```

- 测试

- 问题 :

1. 如果条件输入框输入不了内容 , searchMap指定搜索字段 , 然后多刷新几页面 , 如果还不行 , 检查下 prop 有没有指定字段名 , 并且它是指定在 el-form-item 标签上
2. 以下报错 , 是因为请求太过于频繁 , 接口服务器有限制 , 放慢手步即可~~

```
* ▶ POST http://localhost:8888/dev-api/member/list/search/1/10 429 (Too Many Requests)
* ▶ Uncaught (in promise) Error: Request failed with status code 429
  at createError (createError.js?2d83:16)                                     点太快, 请求太多后台会提示你
  at settle (settle.js?467f:17)
  at XMLHttpRequest.handleLoad (xhr.js?b50d:59)
```

第十章 会员管理-新增

10.1 需求分析

1. 点击 新增 按钮后，对话框形式弹出新增窗口
2. 输入会员信息后，点击 确定 提交表单数据；



10.2 新增窗口实现

1. 在 `src\views\member\index.vue` 中分页区域的下方，新增对话框形式表单数据模板

弹出功能参考：<https://element.eleme.cn/#/zh-CN/component/dialog#zi-ding-yi-nei-rong>

- o `el-dialog` 标签属性 `title` 窗口标题，`visible.sync` 是否弹出窗口



- 注意其中包含多行文本编辑框。

```
1 <el-dialog title="会员编辑" :visible.sync="dialogFormVisible" width="500px">
2   <!-- status-icon 当表单校验不通过时, 输入框右侧有个 x 小图标 -->
3   <el-form
4     status-icon
5     ref="pojoForm"
6     :model="pojo"
7     label-width="100px"
8     label-position="right"
9     style="width: 400px;">
10  >
11    <el-form-item label="会员卡号" prop="cardNum">
12      <el-input v-model="pojo.cardNum" />
13    </el-form-item>
14    <el-form-item label="会员姓名" prop="name">
15      <el-input v-model="pojo.name" />
16    </el-form-item>
17    <el-form-item label="会员生日" prop="birthday">
18      <el-date-picker
19        v-model="pojo.birthday"
20        type="date"
21        placeholder="请点击选择"
22      />
23    </el-form-item>
24    <el-form-item label="手机号码" prop="phone">
25      <el-input v-model="pojo.phone" />
26    </el-form-item>
27    <el-form-item label="开卡金额" prop="money">
28      <el-input v-model="pojo.money" />
29    </el-form-item>
30    <el-form-item label="可用积分" >
31      <el-input v-model="pojo.integral" />
32    </el-form-item>
33    <el-form-item label="支付类型" prop="payType">
34      <el-select v-model="pojo.payType" class="filter-item" placeholder="请点击选择">
35        <el-option
36          v-for="option in payTypeOptions"
37          :key="option.type"
38          :label="option.name"
39          :value="option.type"
40        />
41      </el-select>
42    </el-form-item>
43    <el-form-item label="会员地址">
44      <el-input v-model="pojo.address" type="textarea"
45        :autosize="{ minRows: 2, maxRows: 4}" placeholder="请输入地址"
46      />
47    </el-form-item>
48  </el-form>
49  <div slot="footer" class="dialog-footer">
50    <el-button @click="dialogFormVisible = false">取消</el-button>
51    <el-button type="primary" @click="addData('pojoForm')">确定</el-button>
```



```
52  </div>
53  </el-dialog>
```

2. data 选项中声明变量数据对象 `pojo` 和 `dialogFormVisible` 当它为 `true` 弹出 , `false` 不弹。

```
1  export default {
2    data() {
3      return {
4        list: [],
5        total: 0, // 总记录数
6        currentPage: 1, // 当前页, 默认第1页
7        pageSize: 10, // 每页显示条数 , 10条
8        searchMap: { // 条件查询绑定的条件值
9          cardNum: '',
10         name: '',
11         payType: '',
12         birthday: ''
13       },
14       payTypeOptions, // 支付类型 , ES6语法
15     }
16     pojo: {
17       cardNum: '',
18       name: '',
19       birthday: '',
20       phone: '',
21       money: 0,
22       integral: 0,
23       payType: '',
24       address: ''
25     }, // 提交的数据
26     dialogFormVisible: false, // 控制新增对话框
27   }
28 },
29 ...
30 }
```

3. 在 `methods` 中添加 `addData()` 函数,提交表单数据用的

```
1  // 提交新增数据
2  addData(formName) {
3    this.$refs[formName].validate(valid => {
4      if (valid) {
5        // 验证通过 , 提交添加
6        alert('Add submit!');
7      } else {
8        // 验证不通过
9        return false;
10      }
11    })
12  },
```



4. 在 template 中的查询按钮旁边添加一个 新增 按钮，用于打开新增会员对话框

```
1 <el-form-item>
2   <el-button type="primary" icon="el-icon-search" @click="fetchData">查询</el-button>
3   <el-button type="primary" icon="el-icon-edit"
4     @click="handleAdd">新增</el-button>
5   <el-button @click="resetForm('searchForm')">重置</el-button>
6 </el-form-item>
```

5. 在 methods 中添加 handleAdd() 函数, 打开新增对话框

关闭窗口后，再次打开窗口，会发现表单里依然有数据，应当清除数据。

```
1 // 打开新增窗口
2 handleAdd() {
3   this.dialogFormVisible = true
4   this.$nextTick(() => {
5     // this.$nextTick()它是一个异步事件，当渲染结束之后，它的回调函数才会被执行
6     // 弹出窗口打开之后，需要加载Dom, 就需要花费一点时间，我们就应该等待它加载完dom之后，再进行调
7     // 用resetFields方法，重置表单和清除样式
8     this.$refs['pojoForm'].resetFields()
9   })
10 },
```

6. 测试是否正常打开新增对话框

10.3 表单数据校验

1. 在新增窗口的 el-form 上绑定属性 :rules="rules"

```
1 <el-dialog title="会员编辑" :visible.sync="dialogFormVisible" width="500px">
2   <el-form
3     :rules="rules"
```

2. 在 data 选项中添加 rules 属性进行校验，对卡号，姓名、支付类型 校验

```
1 data() {
2   return {
3     ...
4     // 校验规则
5     rules: {
6       cardNum: [{ required: true, message: '卡号不能为空', trigger: 'blur' }],
7       name: [{ required: true, message: '姓名不能为空', trigger: 'blur' }],
8       payType: [{ required: true, message: '请选择支付类型', trigger: 'change' }],
9     },
10   }
11 },
```

10.4 表单数据提交

当点击新增窗口中的确认按钮时，提交表单数据，后台API服务接口响应新增成功或失败。

10.4.1 EasyMock 添加新增会员模拟接口

- 请求URL : /member
- 请求方式 : post
- 描述 : 新增会员
- mock.js 配置 :

```
1  {
2      "code": 2000,
3      "flag": true,
4      "message": "新增成功"
5 }
```

10.4.2 Api 调用接口

1. src\api\member.js 导出的默认对象中，添加调用新增接口的方法

```
1 // 新增会员
2 add(pojo) {
3     return request({
4         url: '/member',
5         method: 'post',
6         data: pojo
7     })
8 }
```

2. 在 src\views\member\index.vue 中的 addData 方法中提交数据，代码如下：

```
1 // 提交新增数据
2 addData(formName) {
3     this.$refs[formName].validate(valid => {
4         if (valid) {
5             // 验证通过，提交添加
6             memberApi.add(this.pojo).then(response => {
7                 const resp = response.data
8                 console.log(resp)
9                 if (resp.flag) {
10                     // 新增成功，刷新列表数据
11                     this.fetchData()
12                     this.dialogFormVisible=false //关闭窗口
13                 }else {
14                     // 失败，弹出提示
15                     this.$message({
```

```
16         message: resp.message,
17         type: 'warning'
18     });
19 }
20 } else {
21     // 验证不通过
22     return false;
23 }
24 }
25 })
26 },
```

3. 测试新增功能。

第十一章 会员管理-修改

11.1 需求分析

当点击 **编辑** 按钮后，弹出编辑会员窗口，并查询出会员信息渲染。修改后点击确定 提交修改数据。



11.2 EasyMock 添加模拟接口

11.2.1 ID 查询数据接口

- 请求URL : /member/{id}

- 请求方式 : `get`
- 描述 : 会员 ID 查询数据接口
- mock.js 配置 :

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": "查询成功",
5    "data": {
6      "id": 10,
7      "cardNum": "test01", //大于1000的正整数
8      "name": "测试会员",
9      "birthday": "@date",
10     "phone|11": "@integer(0,9)", // 11个位数字
11     "integral": "@integer(0, 500)",
12     "money": "@float(0, 100, 1, 3)", // 0-1000小数,1-3位小数位
13     "payType|1": [1, 2, 3, 4],
14     "address": "@county(true)"
15   }
16 }
```

11.2.2 提交修改数据接口

- 请求URL : `/member/{id}`
- 说明 : id 是修改的那条数据
- 请求方式 : `put`
- 描述 : 会员数据更新
- mock.js 配置 :

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": "修改成功"
5 }
```

11.3 Api 调用接口

1. `src\api\member.js` 导出的默认对象中 , 添加ID查询方法 `getById` 和 更新方法 `update`

```
1  // 查询
2  getById(id) {
3    return request({
4      url: `/member/${id}` // 反单引号 ``
```

```
5     method: 'get'
6   })
7 },
8
9 // 更新
10 update(pojo) {
11   return request({
12     url: `/member/${pojo.id}` // 反单引号 ``
13     method: 'put', // put 方式提交
14     data: pojo
15   })
16 }
```

2. 在 `src\views\member\index.vue` 中的 `handleEdit` 方法做如下修改：

```
1 // 打开编辑窗口
2 handleEdit(id) {
3   // 清空原数据
4   this.handleAdd()
5   // 通过Id查询数据
6   memberApi.getById(id).then(response => {
7     const resp = response.data
8     if(resp.flag) {
9       this.pojo = resp.data
10    }
11  })
12},
```

3. 测试，打开修改页面是否有数据

4. 在 `data` 选项的 `pojo` 对象中添加 `id` 属性

```
1 data() {
2   return {
3     ...,
4     pojo: {
5       id: null, // 增加 id
6       cardNum: '',
7       name: '',
8       birthday: '',
9       phone: '',
10      money: 0,
11      integral: 0,
12      payType: '',
13      address: ''
14    },
15  }
```

5. 修改对话框中的确认按钮点击事件，`pojo.id === null` 值为 `true` 是新增，有 `id` 说明是更新。

```
1 <el-button type="primary" @click="pojo.id === null ? addData('pojoForm'): updateData('pojoForm')">确定</el-button>
```

6. 提交更新数据

```
1 // 更新数据
2 updateData(formName) {
3   this.$refs[formName].validate(valid => {
4     if (valid) {
5       // 验证通过，提交
6       memberApi.update(this.pojo).then(response => {
7         const resp = response.data
8         if(resp.flag) {
9           this.fetchData() // 刷新列表数据
10          this.dialogFormVisible = false // 关闭窗口
11        }else {
12          // 失败，弹出提示
13          this.$message({
14            message: resp.message,
15            type: 'warning'
16          });
17        }
18      })
19    } else {
20      // 验证不通过
21      return false;
22    }
23  })
24},
```

第十二章 会员管理-删除

12.1 需求分析

当点击删除按钮后，弹出提示框，点击确定后，执行删除并刷新列表数据

确认消息弹框参考：<https://element.eleme.cn/#/zh-CN/component/message-box#que-ren-xiao-xi>



会员管理

会员卡号	会员姓名	会员生日	手机号码	可用积分	开卡金额	支付类型	会员地址	操作
349184549 8843292	侯超					微信	天津 天津市 西青区	<button>编辑</button> <button>删除</button>
310497761 6405547	马明					微信	海南省 三沙市 南沙群岛	<button>编辑</button> <button>删除</button>
836355249 6199047	廖超	1986-06-06	6175818601 3	137	91.13	支付宝	广西壮族自治区 柳州市 柳北区	<button>编辑</button> <button>删除</button>
597321440	龙超	1997-11-03	6175718658	498	76.817	微信	西藏自治区 拉萨市 堆龙	<button>编辑</button> <button>删除</button>

提示

! 确认删除这条记录吗？

取消确定

12.2 EasyMock 添加模拟接口

- 请求URL : `/member/{id}`
- 请求方式 : `delete`
- 描述 : 根据会员ID删除
- mock.js 配置 :

```
1 {
2   "code": 2000,
3   "flag": true,
4   "message": "删除成功"
5 }
```

12.3 Api 调用接口

1. `src\api\member.js` 导出的默认对象中，添加 `deleteById` 方法

```
1 deleteById(id) {
2   return request({
3     url: `/member/${id}`, // 反单引号 ``
4     method: 'delete', // delete 方式提交
5   })
6 }
```

2. 在 `src\views\member\index.vue` 中的 `handleDelete` 方法做如下修改：

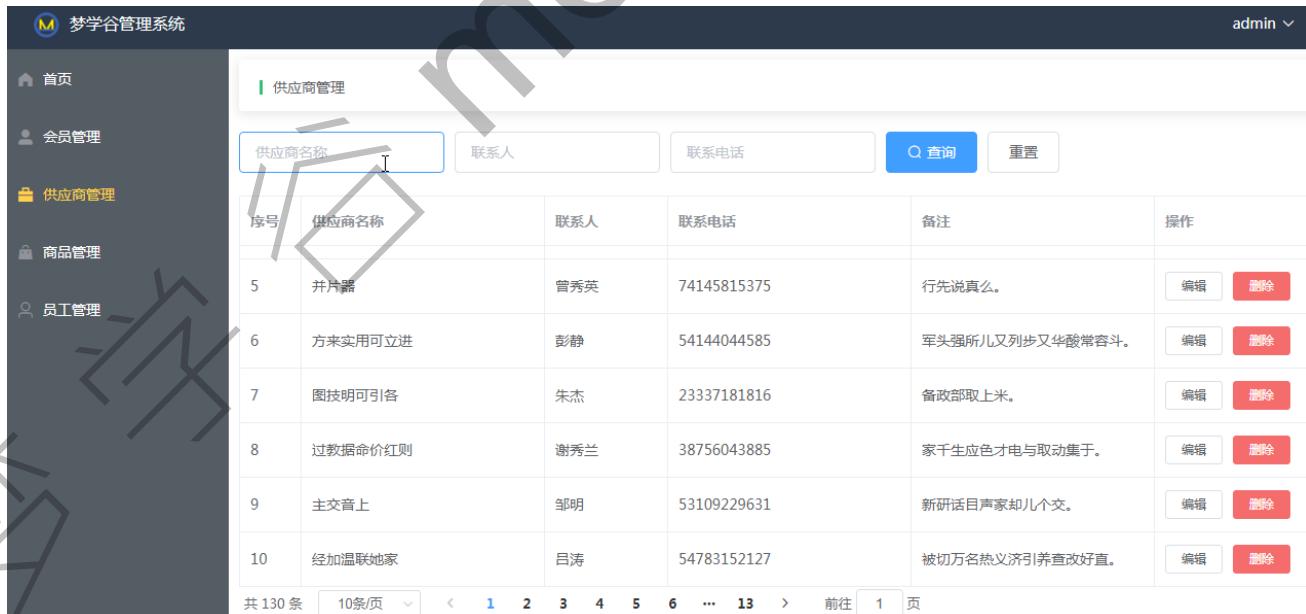
```
1 handleDelete(id) {
2   console.log('删除', id)
3   this.$confirm('确认删除这条记录吗？', '提示', {
4     confirmButtonText: '确定',
5   })
6 }
```

```
5     cancelButtonText: '取消',
6     type: 'warning'
7 }).then(() => {
8     // 确认
9     memberApi.deleteById(id).then(response => {
10        const resp = response.data
11        // 提示信息
12        this.$message({
13            type: resp.flag ? 'success': 'error',
14            message: resp.message
15        });
16        if(resp.flag) {
17            // 删除成功，刷新列表
18            this.fetchData()
19        }
20    })
21 }).catch(() => {
22     // 取消删除，不理会
23 });
24 },
```

第十三章 供应商管理-列表查询

13.1 需求分析

用于管理商品的供应商信息，首页开发供应商管理模块中的列表功能，包含条件查询、数据列表、分页。



序号	供应商名称	联系人	联系电话	备注	操作
5	并片器	曾秀英	74145815375	行先说真么。	<button>编辑</button> <button>删除</button>
6	方来实用可立进	彭静	54144044585	军头强所儿又列步又华酸常容斗。	<button>编辑</button> <button>删除</button>
7	图技明可引各	朱杰	23337181816	备政部取上米。	<button>编辑</button> <button>删除</button>
8	过数据命价红则	谢秀兰	38756043885	家千生应色才电与取动集于。	<button>编辑</button> <button>删除</button>
9	主交音上	邹明	53109229631	新研话目声家却儿个文。	<button>编辑</button> <button>删除</button>
10	经加温联她家	吕涛	54783152127	被切万名热义济引养查改好直。	<button>编辑</button> <button>删除</button>

13.2 供应商数据列表

13.2.1 EasyMock 添加数据列表模拟接口

- 请求URL : /supplier/list
- 请求方式 : get
- 描述 : 供应商数据列表
- mock.js 配置 :

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": "查询成功",
5    "data|20": [
6      "id|+1": 10,
7      "name": "@ctitle",
8      "linkman": "@cname",
9      "mobile|11": "@integer(0, 9)", // 11个数字0-9间的数字
10     "remark": "@csentence(5, 15)",
11   ]
12 }
```

13.2.2 Api 调用接口

1. 在 `src/api` 下创建 `supplier.js` , 调用接口代码如下:

```
1 import request from "@/utils/request"
2
3 export default {
4   // 获取列表
5   getList(){
6     return request({
7       url:'/supplier/list',
8       method:'get'
9     })
10   },
11 }
```

2. 在 `src\views\supplier\index.vue` 中, 添加 JS 代码如下 :

```
1 <template>
2
3 </template>
4
5 <script>
6 import supplierApi from '@/api/supplier'
7
8 export default {
9   data() {
10     return {
```

```
11     list: []
12   }
13 },
14 // 钩子函数获取数据
15 created () {
16   this.fetchData()
17 },
18
19 methods: {
20   fetchData () {
21     supplierApi.getList().then(response => {
22       // console.log(response.data.data)
23       this.list = response.data.data
24     })
25   }
26 },
27 }
28 </script>
```

13.2.3 列表模板

1. 修改 `src\views\supplier\index.vue`，编写模板代码：

```
1 <template>
2 <div>
3   <!-- 列表 -->
4   <el-table :data="list" height="380" border style="width: 100%" >
5     <el-table-column type="index" label="序号" width="60"></el-table-column>
6     <el-table-column prop="name" label="供应商名称"></el-table-column>
7     <el-table-column prop="linkman" label="联系人" width="120"></el-table-column>
8     <el-table-column prop="mobile" label="联系电话"></el-table-column>
9     <el-table-column prop="remark" label="备注"></el-table-column>
10    <el-table-column label="操作" width="150">
11      <template slot-scope="scope">
12        <el-button size="mini" @click="handleEdit(scope.row.id)">编辑</el-button>
13        <el-button size="mini" type="danger" @click="handleDelete(scope.row.id)">删除</el-button>
14      </template>
15    </el-table-column>
16  </el-table>
17 </div>
18 </template>
```

2. 在 `methods` 中添加 `handleEdit` 编辑和 `handleDelete` 删除方法，后面需要使用。

```
1 methods: {
2   ...
3   ...
4
5   handleEdit(id) {
6     console.log('编辑')
7   },
8   handleDelete(id) {
9     console.log('删除')
10  },
11 }
```

3. 测试查看效果

13.3 分页功能实现

为列表数据添加分页功能，使用分页组件完成分页功能。

参考：<https://element.eleme.cn/#/zh-CN/component/pagination>

13.3.1 EasyMock 添加商品分页模拟接口

- 请求URL： /supplier/list/search/{page}/{size}
- 请求方式： post
- 描述：供应商列表数据分页
- mock.js 配置：

```
1 {
2   "code": 2000,
3   "flag": true,
4   "message": "查询成功",
5   "data": {
6     "total": "@integer(100,200)", // 查询出来的总记录数
7     "rows|10": [{} // 返回当前页的记录数 10 条, 即每页显示 10 条记录
8       "id|+1": 10,
9       "name": "@ctitle",
10      "linkman": "@cname",
11      "mobile|11": "@integer(0, 9)", // 11个数字0-9间的数字
12      "remark": "@csentence(5, 15)"
13    ]
14  }
15 }
```

13.3.2 Api 调用接口



1. 修改 `src\api` 下的 `supplier.js` , 在导出的默认对象中 , 增加分页查询方法

```
1 // 分页查询
2 search(page, size, searchMap) {
3     return request({
4         url: `/supplier/list/search/${page}/${size}`,
5         method: 'post',
6         data: searchMap
7     })
8 }
```

2. 在 `src\views\supplier\index.vue` 中, 添加 JS 代码如下 :

声明分页属性 , 调用 `getList` 改为调用 `search` 方法

```
1 <script>
2 import supplierApi from '@/api/supplier'
3
4 export default {
5     data() {
6         return {
7             list: [],
8             total: 0, // 总记录数
9             currentPage: 1, // 当前页, 默认第1页
10            pageSize: 10, // 每页显示条数 , 10条
11            searchMap: {
12                name: '',
13                linkman: '',
14                mobile: ''
15            }, // 条件查询的绑定字段值
16        }
17    },
18
19    created () {
20        this.fetchData()
21    },
22
23    methods: {
24        fetchData () {
25            supplierApi.search(this.currentPage, this.pageSize, this.searchMap)
26            .then(response => {
27                const reqs = response.data
28                this.total = reqs.data.total
29                this.list = reqs.data.rows
30                console.log(this.list)
31            })
32        },
33        ...
34        ...
35    },
36 }
```

```
37 </script>
```

- 测试查看控制台否打印

13.3.3 分页模板

1. 修改 `src\views\supplier\index.vue` , `template` 标签中添加页面模板代码 :

注意 : 添加在 `div` 里面。因为 `template` 里面只能有唯一根节点

```
1 <!-- 分页，添加在div里面 -->
2 <el-pagination
3   @size-change="handleSizeChange"
4   @current-change="handleCurrentChange"
5   :current-page="currentPage"
6   :page-sizes="[10, 20, 50]"
7   :page-size="pageSize"
8   layout="total, sizes, prev, pager, next, jumper"
9   :total="total">
10 </el-pagination>
```

2. 在 `methods` 中添加方法 `handleSizeChange` 和 `handleCurrentChange` 动态赋值与刷新列表

```
1 methods: {
2   handleSizeChange(val) {
3     this.pageSize = val
4     this.fetchData()
5   },
6   handleCurrentChange(val) {
7     this.currentPage = val
8     this.fetchData()
9   },
}
```

3. 测试可通过

13.4 条件查询和重置实现

在列表上方添加查询功能，通过 供应商名称、联系人、联系电话 等条件查询

13.4.1 条件查询模板

1. 修改 `src\views\supplier\index.vue` , 增加条件查询模板代码 :

```
1 <el-form ref="searchForm" :inline="true" :model="searchMap" style="margin-top: 20px;" >
2   <el-form-item prop="name">
3     <el-input v-model="searchMap.name" placeholder="商品名称" style="width: 200px;" ></el-input>
4   </el-form-item>
```

```
5   <el-form-item prop="code">
6     <el-input v-model="searchMap.code" placeholder="商品编号" style="width: 200px;"></el-input>
7   </el-form-item>
8   <el-form-item prop="supplierId">
9     <el-input v-model="searchMap.supplierName" placeholder="供应商" style="width: 200px;"></el-input>
10  </el-form-item>
11  <el-form-item>
12    <el-button type="primary" icon="el-icon-search" >查询</el-button>
13  </el-form-item>
14 </el-form>
```

- 测试查看页面效果

商品管理



商品管理查询界面展示了商品列表。上方有三个输入框：商品名称、商品编号、供应商，以及一个带搜索图标和“查询”按钮的按钮。下方是一个表格，列出了商品的序号、商品名称、商品编码、商品规格、零售价、进货价、库存数量、供货商等信息。表格中有一条记录：

序号	商品名称	商品编码	商品规格	零售价	进货价	库存数量	供
1	今低马反求	sxdobj	3L	70.366	24.9	35	据

13.4.2 重置功能

- 在 条件查询区域添加一个 重置 按钮

```
1 <el-button @click="resetForm('searchForm')">重置</el-button>
```

- methods 选项中添加重置处理方法 resetForm

注意: 在 el-form-item 标签属性 prop 上, 指定了字段名, 重置才会生效

```
1 // 表单重置
2 // 在 el-form-item 标签属性 prop 上, 指定了字段名, 重置才会生效
3 resetForm(formName) {
4   this.$refs[formName].resetFields();
5 },
```

- 测试,

如果条件输入框输入不了内容, 多刷新几页面, 如果还不行, 检查下 prop 有没有指定字段名

第十四章 供应商管理-新增

14.1 需求分析

1. 点击 新增 按钮后，对话框形式弹出新增窗口
2. 输入供应商信息后，点击 确定 提交表单数据；



14.2 新增窗口实现

1. 在 `src\views\supplier\index.vue` 中分页区域的下方，新增对话框形式表单数据模板

```
1 <el-dialog title="供应商编辑" :visible.sync="dialogFormVisible" width="500px">
2   <el-form
3     :rules="rules"
4     status-icon
5     ref="pojoForm"
6     :model="pojo"
7     label-width="100px"
8     label-position="right"
9     style="width: 400px;">
10    >
11    <el-form-item label="供应商名称" prop="name">
12      <el-input v-model="pojo.name" />
13    </el-form-item>
14    <el-form-item label="联系人" prop="linkman">
15      <el-input v-model="pojo.linkman" />
```



```
16      </el-form-item>
17      <el-form-item label="联系电话" prop="mobile">
18          <el-input v-model="pojo.mobile" />
19      </el-form-item>
20      <el-form-item label="备注" prop="remark">
21          <el-input v-model="pojo.remark" type="textarea"
22              :autosize="{ minRows: 2, maxRows: 4}" placeholder="请输入地址"
23          />
24      </el-form-item>
25  </el-form>
26  <div slot="footer" class="dialog-footer">
27      <el-button @click="dialogFormVisible = false">取消</el-button>
28      <el-button type="primary" @click="pojo.id === undefined ? addData('pojoForm'):
29          updateData('pojoForm')">确定</el-button>
30  </div>
31 </el-dialog>
```

2. data 选项中声明变量属性:

- `pojo` 提交表单对象
- `dialogFormVisible` 当它为 `true` 弹出 , `false` 不弹。
- `rules` : 定义校验规则

```
1  data() {
2      return {
3          list: [],
4          total: 0, // 总记录数
5          currentPage: 1, // 当前页, 默认第1页
6          pageSize: 10, // 每页显示条数 , 10条
7          searchMap: {
8              name: '',
9              linkman: '',
10             mobile: ''
11         }, // 条件查询的绑定字段值
12
13         pojo: {
14             id: null,
15             name: '',
16             linkman: '',
17             mobile: '',
18             remark: ''
19         },
20         dialogFormVisible: false,
21         rules: {
22             name: [{ required:true,message:'供应商名称不能为空',trigger: 'blur' }],
23             linkman: [{ required:true,message:'联系人不能为空', trigger: 'blur' }],
24         },
25     }
26 },
27 ...
28 ...
```



3. 在 `methods` 中添加 `addData()` 函数, 提交表单数据用的

```
1 // 提交新增数据
2 addData(formName) {
3     this.$refs[formName].validate(valid => {
4         if (valid) {
5             // 验证通过，提交添加
6             alert('Add submit!');
7         } else {
8             // 验证不通过
9             return false;
10        }
11    })
12},
```

4. 在 `template` 中的查询按钮旁边添加一个 `新增` 按钮，用于打开新增会员对话框

```
1 <el-button type="primary" icon="el-icon-edit" @click="handleAdd">新增</el-button>
```

5. 在 `methods` 中添加 `handleAdd()` 函数, 打开新增对话框

关闭窗口后，再次打开窗口，会发现表单里依然有数据，应当清除数据。

```
1 // 打开新增窗口
2 handleAdd() {
3     this.dialogFormVisible = true
4     this.$nextTick(() => {
5         // this.$nextTick()它是一个异步事件，当渲染结束之后，它的回调函数才会被执行
6         // 弹出窗口打开之后，需要加载Dom, 就需要花费一点时间，我们就应该等待它加载完dom之后，再进行调
7         // 用resetFields方法，重置表单和清除样式
8         this.$refs['pojoForm'].resetFields()
9     })
10},
```

6. 测试是否正常打开新增对话框

14.3 表单数据提交

当点击新增窗口中的确认按钮时，提交表单数据，后台API服务接口响应新增成功或失败。

14.3.1 EasyMock 添加新增供应商模拟接口

- 请求URL：`/supplier`
- 请求方式：`post`
- 描述：新增供应商
- mock.js 配置：

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": "新增成功"
5 }
```

14.4.2 Api 调用接口

1. src\api\supplier.js 导出的默认对象中，添加调用新增接口的方法

```
1 // 新增
2 add(pojo) {
3   return request({
4     url: '/supplier',
5     method: 'post',
6     data: pojo
7   })
8 }
```

2. 在 src\views\supplier\index.vue 中的 addData 方法中提交数据，代码如下：

```
1 // 提交新增数据
2 addData(formName) {
3   this.$refs[formName].validate(valid => {
4     if (valid) {
5       // 验证通过，提交添加
6       supplierApi.add(this.pojo).then(response => {
7         const resp = response.data
8         if(resp) {
9           this.fetchData()
10          this.dialogFormVisible = false
11        }else{
12          // 验证不通过
13          this.$message({
14            message: resp.message,
15            type: 'warning'
16          })
17        }
18      })
19    } else {
20      return false;
21    }
22  })
23 },
```

3. 测试新增功能。

第十五章 供应商管理-修改

15.1 需求分析

当点击 **编辑** 按钮后，弹出编辑供应商窗口，并查询出信息渲染。修改后点击确定 提交修改数据。



15.2 EasyMock 添加模拟接口

15.2.1 ID 查询数据接口

- 请求URL : `/supplier/{id}`
- 请求方式 : `get`
- 描述 : 供应商 ID 查询数据接口
- mock.js 配置 :

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": "查询成功",
5    "data": {
6      "id|+1": 10,
7      "name": "@ctitle",
8      "linkman": "@cname",
9      "mobile|11": "@integer(0, 9)", // 11个数字0-9间的数字
10     "remark": "@csentence(5, 15)",
11   }
12 }
```

15.2.2 提交修改数据接口

- 请求URL : `/supplier/{id}`
说明 : id 是修改的那条数据
- 请求方式 : `put`
- 描述 : 供应商数据更新
- mock.js 配置 :

```
1  {
2      "code": 2000,
3      "flag": true,
4      'message': "修改成功"
5 }
```

15.3 Api 调用接口

1. src\api\supplier.js 导出的默认对象中，添加ID查询方法 `getById` 和 更新方法 `update`

```
1 // 查询
2 getById(id) {
3     return request({
4         url: `/supplier/${id}` , // 反单引号 ``
5         method: 'get'
6     })
7 },
8
9 // 更新
10 update(pojo) {
11     return request({
12         url: `/supplier/${pojo.id}` , // 反单引号 ``
13         method: 'put' , // put 方式提交
14         data: pojo
15     })
16 }
```

2. 在 `src\views\supplier\index.vue` 中的 `handleEdit` 方法做如下修改：

```
1 // 打开编辑窗口
2 handleEdit(id) {
3     // 重用打开新增窗口方法, 不要少了 this
4     this.handleAdd()
5     // 查询数据
6     supplierApi.getById(id).then(response => {
7         const resp = response.data
8         if(resp.flag) {
9             this.pojo = resp.data
10        }
11    })
12},
```

3. 测试，打开修改页面是否有数据, 如果报404, 看下是easyMock上URL错了或者supplier.js 路径错了
4. 检查 data 选项中的 pojo 对象中有没有 id: null 属性，如果没有则加上

```
1 data() {
2     return {
3         ...
4         pojo: {
5             id: null,
6             name: '',
7             linkman: '',
8             mobile: '',
9             remark: ''
10        },
11    }
12}
```

5. 修改对话框中的确认按钮点击事件，pojo.id === null 值为true是新增，有id说明是更新。

```
1 <el-button type="primary" @click="pojo.id === null ? addData('pojoForm'): updateData('pojoForm')">确定
</el-button>
```

6. 提交更新数据

```
1 updateData(formName) {
2     this.$refs[formName].validate(valid => {
3         if (valid) {
4             // 验证通过，提交添加
5             supplierApi.update(this.pojo).then(response => {
6                 const resp = response.data
7                 if(resp.flag) {
8                     this.fetchData()
9                     this.dialogFormVisible = false
10                }else{
11                    // 验证不通过
12                    this.$message({
13                        message: resp.message,
```

```
14      type: 'warning'
15    })
16  }
17})
18} else {
19  return false;
20}
21})
22},
```

7. 测试效果

第十六章 供应商管理-删除

16.1 需求分析

当点击删除按钮后，弹出提示框，点击确定后，执行删除并刷新列表数据

确认消息弹框参考：<https://element.eleme.cn/#/zh-CN/component/message-box#que-ren-xiao-xi>



16.2 EasyMock 添加模拟接口

- 请求URL：/supplier/{id}
- 请求方式：`delete`
- 描述：根据供应商 ID 删除
- mock.js 配置：

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": "删除成功"
5 }
```

16.3 Api 调用接口

1. 在 `src\api\supplier.js` 导出的默认对象中，添加 `deleteById` 方法

```
1 deleteById(id) {
2     return request({
3         url: `/member/${id}`, // 反单引号 ``
4         method: 'delete', // delete 方式提交
5     })
6 }
```

2. 在 `src\views\supplier\index.vue` 中的 `handleDelete` 方法做如下修改：

```
1 handleDelete(id) {
2     console.log('删除', id)
3     this.$confirm('确认删除这条记录吗？', '提示', {
4         confirmButtonText: '确定',
5         cancelButtonText: '取消',
6         type: 'warning'
7     }).then(() => {
8         // 确认
9         memberApi.deleteById(id).then(response => {
10             const resp = response.data
11             // 提示信息
12             this.$message({
13                 type: resp.flag ? 'success': 'error',
14                 message: resp.message
15             });
16             if(resp.flag) {
17                 // 删除成功，刷新列表
18                 this.fetchData()
19             }
20         })
21     }).catch(() => {
22         // 取消删除，不理会
23     });
24 },
```

第十七章 商品管理-列表查询

17.1 需求分析

有大改，备份一下上面工程。

用于商品进货管理，首页开发商品管理模块中的列表功能，包含查询供应商、条件查询、数据列表、分页。

重点掌握：弹出窗口选择供应商，并回显供应商信息。通过父子组件间通信。

17.2 分页查询商品数据列表

17.2.1 EasyMock 添加分页数据列表模拟接口

- 请求URL：`/goods/list/search/{page}/{size}`
- 请求方式：`post`
- 描述：商品分页数据列表
- mock.js 配置：

```
1  {
2   "code": 2000,
3   "flag": true,
4   "message": "查询成功",
5   "data": {
6     "total": "@integer(100,200)", // 查询出来的总记录数
7     "rows|10": [{ // 返回当前页的记录数 10 条, 即每页显示 10 条记录
8       "id|+1": 10,
9       "name|5": "@cword",
10      "code": "@word",
11      "spec": "@integer(0,9)L", // 11个数字0-9间的数字
12      "retailPrice": "@float(0, 100, 1, 3)", // 0-1000小数,1-3位小数位
13      "purchasePrice": "@float(0, 100, 1, 3)", // 0-1000小数,1-3位小数位
14      "storageNum": "@integer(0,500)", // 4选其1
15      "supplierName": "@ctitle",
16      "supplierID": "@integer(1,500)"
17    }]
18  }
19 }
```

17.2.2 Api 调用接口

1. 在 `src/api` 下创建 `goods.js`，在导出的默认对象中，增加带分页查询方法调用接口代码如下：



```
1 import request from '@/utils/request"
2
3 export default {
4     // 分页获取列表
5     search(page, size, searchMap) {
6         return request({
7             url: `/goods/list/search/${page}/${size}`,
8             method: 'post',
9             data: searchMap
10        })
11    },
12 }
```

2. 在 `src\views\goods\index.vue` 中, 添加 JS 代码如下 :

```
1 <template>
2
3 </template>
4 <script>
5 import goodsApi from '@/api/goods'
6
7 export default {
8     data() {
9         return {
10            list: [],
11            total: 0, // 总记录数
12            currentPage: 1, // 当前页, 默认第1页
13            pageSize: 10, // 每页显示条数 , 10条
14            searchMap: {}, // 条件查询绑定条件值
15        }
16    },
17
18    created () {
19        this.fetchData()
20    },
21
22    methods: {
23        fetchData () {
24            goodsApi.search(this.currentPage, this.pageSize, this.searchMap)
25                .then(response => {
26                    const reqs = response.data
27                    this.total = reqs.data.total
28                    this.list = reqs.data.rows
29                    //console.log(this.list)
30                })
31        },
32    },
33 }
34 </script>
```

17.2.3 列表与分页模板

1. 修改 `src\views\goods\index.vue` , 编写模板代码 :

```
1 <template>
2   <div>
3     <!-- 列表 -->
4     <el-table :data="list" height="380" border style="width: 100%" highlight-current-row>
5       <el-table-column type="index" label="序号" width="60"></el-table-column>
6       <el-table-column prop="name" label="商品名称"></el-table-column>
7       <el-table-column prop="code" label="商品编码"></el-table-column>
8       <el-table-column prop="spec" label="商品规格"></el-table-column>
9       <el-table-column prop="retailPrice" label="零售价"></el-table-column>
10      <el-table-column prop="purchasePrice" label="进货价"></el-table-column>
11      <el-table-column prop="storageNum" label="库存数量"></el-table-column>
12      <el-table-column prop="supplierName" label="供应商"></el-table-column>
13      <el-table-column label="操作" width="150">
14        <template slot-scope="scope">
15          <el-button size="mini" @click="handleEdit(scope.row.id)">编辑</el-button>
16          <el-button size="mini" type="danger" @click="handleDelete(scope.row.id)">删除</el-button>
17        </template>
18      </el-table-column>
19    </el-table>
20
21    <!-- 分页，添加在div里面 -->
22    <el-pagination
23      @size-change="handleSizeChange"
24      @current-change="handleCurrentChange"
25      :current-page="currentPage"
26      :page-sizes="[10, 20, 50]"
27      :page-size="pageSize"
28      layout="total, sizes, prev, pager, next, jumper"
29      :total="total">
30    </el-pagination>
31  </div>
32 </template>
```

2. 在 `methods` 中

- 添加 `handleEdit` 编辑和 `handleDelete` 删除方法 , 后面需要使用。
- 添加 `handleSizeChange` 和 `handleCurrentChange` 方法 , 实现动态赋值与刷新列表。

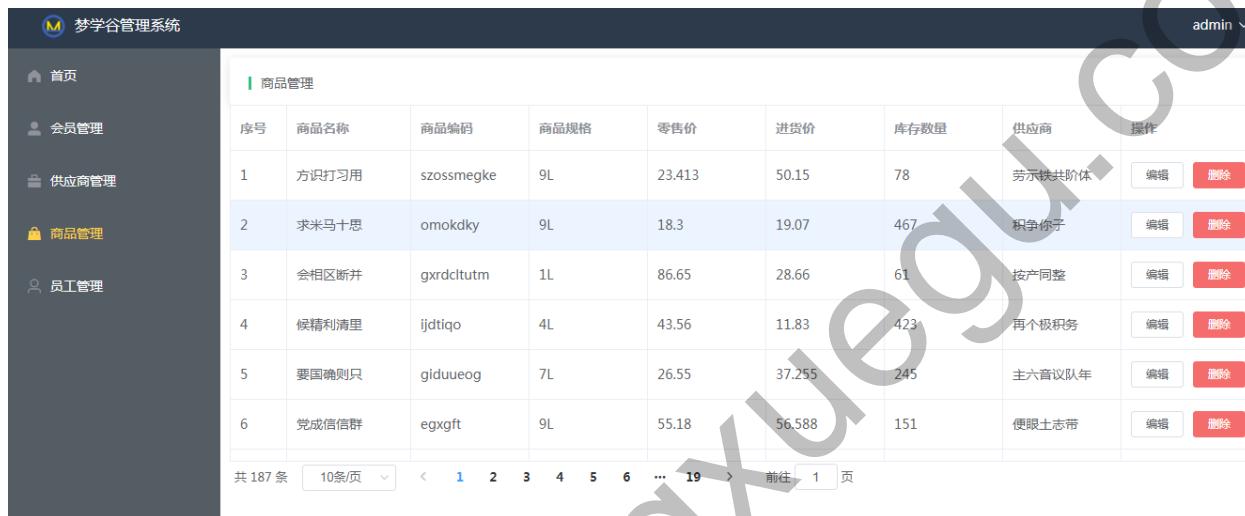
```
1 methods: {
2   ...
3
4   handleSizeChange(val) {
5     this.pageSize = val
6     this.fetchData()
7   },
8   handleCurrentChange(val) {
9     this.currentPage = val
10    this.fetchData()
```

```

11 },
12 handleEdit(id) {
13   console.log('编辑')
14 },
15 handleDelete(id) {
16   console.log('删除')
17 },
18 }

```

3. 测试查看效果



序号	商品名称	商品编码	商品规格	零售价	进货价	库存数量	供应商	操作
1	方识打习用	szossmegke	9L	23.413	50.15	78	劳示扶共阶体	<button>编辑</button> <button>删除</button>
2	求米马十思	omokdky	9L	18.3	19.07	467	积争你子	<button>编辑</button> <button>删除</button>
3	会相区斯并	gxrddltutm	1L	86.65	28.66	61	按产同整	<button>编辑</button> <button>删除</button>
4	候精利清里	ijdtiqo	4L	43.56	11.83	423	再个极积劣	<button>编辑</button> <button>删除</button>
5	要国确则只	gididueog	7L	26.55	37.255	245	主六音议队年	<button>编辑</button> <button>删除</button>
6	党成信信群	egxgft	9L	55.18	56.588	151	便眼土志带	<button>编辑</button> <button>删除</button>

共 187 条 10条/页 < 1 2 3 4 5 6 ... 19 > 前往 页

17.3 条件查询实现

在列表上方添加查询功能，通过 供应商名称、联系人、联系电话 等条件查询

重点掌握：弹出窗口选择供应商，并回显供应商信息。通过父子组件通信。

17.3.1 条件查询模板

1. 修改 `src\views\goods\index.vue`，增加条件查询模板代码：

```

1 <el-form ref="searchForm" :inline="true" :model="search" style="margin-top: 20px;" >
2   <el-form-item prop="name">
3     <el-input v-model="search.name" placeholder="商品名称" style="width: 200px;" ></el-input>
4   </el-form-item>
5   <el-form-item prop="code">
6     <el-input v-model="search.supplierName" placeholder="供应商" style="width: 200px;" ></el-input>
7   </el-form-item>
8   <el-form-item prop="supplierName">
9     <el-input readonly
10       @click.native="dialogSupplierVisible = true"
11       v-model="search.supplierName"
12       placeholder="供应商" style="width: 200px;" ></el-input>
13   </el-form-item>
14   <el-form-item>
15     <el-button type="primary" icon="el-icon-search" >查询</el-button>

```

```
16      <el-button type="primary" icon="el-icon-edit" @click="handleAdd">新增</el-button>
17      <el-button @click="resetForm('searchForm')">重置</el-button>
18      </el-form-item>
19  </el-form>
20
```

- data 选项中的 searchMap 对象中指定条件字段

```
1  searchMap: { // 条件查询绑定条件值
2    name: '',
3    code: '',
4    supplierName: ''
5  },
```

- 测试查询效果

17.3.2 供应商选择

上面供应商条件通过输入了名称查询，这种方式不太好，要模糊查询。

更好的效果是：点击输入框，弹出一个对话框，对话框里直接复用供应商管理组件，



1. 将 supplier\index.vue 导入 goods\index.vue 作为子组件使用。

注意： components 记得注册 Supplier 作为子组件

```
1 <script>
2 import goodsApi from '@/api/goods'
3 // Supplier 作为子组件
4 import Supplier from '@/views/supplier'
5
6 export default {
7   // 不要忘记注册
8   components: { Supplier },
9   ...
10  ...
11 }
```

2. 使用 dialog 对话框组件，包裹 Supplier 组件

```
1 <el-dialog title="选择供应商" :visible.sync="dialogSupplierVisible" width="500px">
2   <supplier></supplier>
3 </el-dialog>
```

3. 在 goods\index.vue 组件的 data 选项中定义 dialogSupplierVisible 控制对话框

```
1 data() {
2   return {
3     list: [],
4     total: 0, // 总记录数
5     currentPage: 1, // 当前页, 默认第1页
6     pageSize: 10, // 每页显示条数 , 10条
7     searchMap: { // 条件查询绑定条件值
8       name: '',
9       code: '',
10      supplierName: ''
11    },
12    dialogSupplierVisible: false // 控制供应商对话框
13  }
14}
```

4. 修改 供应商 输入框

- `readonly` 只读框
- `@click.native` 点击事件。

注意：`el-input` 是组件，要在组件元素监听原生事件，需要使用 `v-on:原生事件名.native="处理函数"`

```

1 <el-form-item prop="supplierId">
2   <!-- <el-input v-model="searchMap.supplierName" placeholder="供应商" style="width: 200px;" ></el-
input> -->
3   <el-input readonly
4     @click.native="dialogSupplierVisible = true"
5     v-model="searchMap.supplierName"
6     placeholder="选择供应商" style="width: 200px;" ></el-input>
7 </el-form-item>

```

5. 浏览弹出效果,发现太丑,



6. 供应商页面在弹出时候很多是我们不需要的, 进行优化,

- 父组件向子组件传递数据 : 通过 props 声明接收
向供应商 `Supplier` 子组件传入一个参数 `isDialog`, 如果是弹出窗口, 则弹出页面隐藏一些功能.
- 在 `goods\index.vue` 上的 `<supplier>` 组件标签上绑定属性 `:isDialog="true"`

```

1 <supplier :isDialog="true"></supplier>

```

- 在 `supplier\index.vue` 中通过 props 选项声明接收 `isDialog`

```
1 export default {  
2   props: {  
3     // 接收父组件传递过来 的数据,通过isDialog来判断 是否为弹框  
4     // 如果为 true, 则是弹框, false 就是列表  
5     isDialog: Boolean  
6   },  
7  
8   . . .
```

- 搜索只要 供应商名称 ; 列表只要 序号、供应商名称、联系人 ; 分页只要 向前、向后和页码
采用 v-if="!isDialog" 来隐藏

搜索隐藏：

```
1 <el-input v-if="!isDialog"  
2   v-model="searchMap.linkman" placeholder="联系人" style="width: 200px;">  
3 </el-input>  
4 <el-input v-if="!isDialog"  
5   v-model="searchMap.mobile" placeholder="联系电话" style="width: 200px;"></el-input>  
6  
7 <el-button v-if="!isDialog"  
8   type="primary" icon="el-icon-edit" @click="handleAdd">新增</el-button>  
9  
10 <el-button v-if="!isDialog" @click="resetForm('searchForm')">重置</el-button>
```

列表隐藏：

```
1 <el-table-column v-if="!isDialog" prop="mobile" label="联系电话"></el-table-column>  
2 <el-table-column v-if="!isDialog" prop="remark" label="备注"></el-table-column>  
3 <el-table-column v-if="!isDialog" label="操作" width="150">
```

分页修改：

```
:layout="!isDialog ? 'total, sizes, prev, pager, next, jumper': 'prev, pager, next'"
```

```
1 <!-- 分页组件 -->  
2 <el-pagination  
3   :layout="!isDialog ? 'total, sizes, prev, pager, next, jumper': 'prev, pager, next'"  
4  
5   @size-change="handleSizeChange"  
6   @current-change="handleCurrentChange"  
7   :current-page="currentPage"  
8   :page-sizes="[10, 20, 50]"  
9   :page-size="pageSize"  
10  :total="total">  
11 </el-pagination>
```

```
1 supplier/index.vue 中的弹出新增页面：
2
3
4     ```html
5     <el-dialog v-if="!isDialog"
6             title="供应商编辑" :visible.sync="dialogFormVisible" width="500px">
```

7. 当点击列表中某一行供应商，即选中；选中的那个供应商数据传回到父组件 商品管理 中。

参考表格单选：

<https://element.eleme.cn/#/zh-CN/component/table#dan-xuan>

在 `supplier\index.vue` 组件的 `el-table` 表格组件上配置 `highlight-current-row` 属性可实现单选；

通过 `current-change` 事件管理选中时触发的事件函数 `clickCurrentChange`，并传入 `currentRow`，`oldCurrentRow` 参数

注意：是 `clickCurrentChange` 函数，不能取 `handleCurrentChange` 函数名，因为在分页查询有定义这个函数名

```
1     <!-- highlight-current-row 激活单选行, isDialog为true时激活
2     @current-change 当点击某一行后,会触发这个事件,从而调用对应的函数clickCurrentChange
3     clickCurrentChange函数会接收两个参数:currentRow , oldCurrentRow
4         -->
5     <el-table
6         :highlight-current-row = "isDialog"
7         @current-change="clickCurrentChange"
8         :data="list"
9         height="380"
10        border
11        style="width: 100%">
```

8. 在 `supplier\index.vue` 组件定义 `clickCurrentChange` 函数，向父组件传递点击数据。

子组件向父组件传递数据：通过 `$emit` 触发事件的方式

```
1     methods: {
2         . . .
3
4         // 当点击某一行时,会调用这个函数进行处理
5         clickCurrentChange(currentRow) {
6             console.log(currentRow)
7             // 点击后,要将点击的数据传递到父组件(商品管理中),
8             // 则可以通过触发父组件中的option-supplier, 触发之后,
9             // 父组件可以在 option-supplier 这个事件对应的处理函数中进行接收数据currentRow
10            this.$emit('option-supplier', currentRow)
11        }
12    },
13 },
```

9. 在父组件 `goods\index.vue` 的组件元素 `supplier` 上绑定自定义事件 `option-supplier`



```
1 <el-dialog title="选择供应商" :visible.sync="dialogSupplierVisible" width="500px">
2   <supplier @option-supplier="optionSupplier" :isDialog="true"></supplier>
3 </el-dialog>
```

10. 在父组件 goods\index.vue 中添加 option-supplier 事件触发的函数 optionSupplier 进行回显数据

```
1 methods: {
2   . . .
3
4   optionSupplier(currentRow) { // currentRow 子组件传递的数据
5     console.log('parent', currentRow)
6     this.searchMap.supplierName = currentRow.name // 供应商名称
7     this.searchMap.supplierId = currentRow.id // 供应商ID
8     this.dialogSupplierVisible = false // 关闭窗口
9   }
10 }
```

17.3.3 重置功能

- 在 条件查询区域添加一个 重置 按钮, 直接在上面调用重置方法, 注意 'searchForm'

```
1 <el-button @click="$refs['searchForm'].resetFields()">重置</el-button>
```

- 如果条件输入框有时输入不了内容, 和供应商无法重置, 在 searchMap 对象中明确属性名

```
1 data() {
2   return {
3     list: [],
4     total: 0, // 总记录数
5     currentPage: 1, // 当前页, 默认第1页
6     pageSize: 10, // 每页显示条数, 10条
7     searchMap: { // 条件查询绑定条件值
8       supplierName: '',
9       code: '',
10      name: ''
11    },
12    dialogSupplierVisible: false // 控制供应商对话框
13  }
14},
```

如果还不行, 检查下在 el-form-item 标签属性 prop 上, 有没指定指定了字段名, 重置才会生效

第十八章 商品管理-新增

18.1 需求分析

1. 点击 新增 按钮后，对话框形式弹出新增窗口
2. 输入商品信息后，点击 确定 提交表单数据

18.2 新增窗口实现

1. 在 `src\views\supplier\index.vue` 中分页区域的下方，新增对话框形式表单数据模板

```
1  <!-- 新增 or 编辑页面 -->
2  <el-dialog title="商品编辑" :visible.sync="dialogFormVisible" width="500px">
3  <el-form
4    :rules="rules"
5    status-icon
6    ref="pojoForm"
7    :model="pojo"
8    label-width="100px"
9    label-position="right"
10   style="width: 400px;">
11  </el-form>
12  <el-form-item label="商品名称" prop="name">
13    <el-input v-model="pojo.name" />
14  </el-form-item>
15  <el-form-item label="商品编码" prop="code">
16    <el-input v-model="pojo.code" />
17  </el-form-item>
18  <el-form-item label="商品规格" prop="spec">
19    <el-input v-model="pojo.spec" />
20  </el-form-item>
21  <el-form-item label="零售价" prop="retailPrice">
22    <el-input v-model="pojo.retailPrice" />
23  </el-form-item>
24  <el-form-item label="进货价" prop="purchasePrice">
25    <el-input v-model="pojo.purchasePrice" />
26  </el-form-item>
27  <el-form-item label="库存数量" prop="storageNum">
28    <el-input v-model="pojo.storageNum" />
29  </el-form-item>
30  <el-form-item label="供应商" prop="supplierId">
31    <el-input readonly
32      @click.native="dialogSupplierVisible = true"
33      v-model="pojo.supplierName"
34      placeholder="选择供应商" style="width: 200px;" ></el-input>
35  </el-form-item>
36 </el-form>
37 <div slot="footer" class="dialog-footer">
38   <el-button @click="dialogFormVisible = false">取消</el-button>
39   <el-button
40     type="primary"
41     @click="pojo.id === null ? addData(pojoForm): updateData(pojoForm)">
```



```
42      >确定</el-button>
43    </div>
44  </el-dialog>
```

2. data 选项中声明变量属性:

- o `pojo` 提交表单对象
- o `dialogFormVisible` 当它为 true 弹出 , false 不弹。
- o `rules` : 定义校验规则

```
1  data() {
2    return {
3      . . . ,
4
5      pojo: {
6        id: null,
7        name: '',
8        code: '',
9        spec: '',
10       retailPrice: 0.0,
11       purchasePrice: 0.0,
12       storageNum: 0,
13       supplierName: '',
14       supplierId: null
15     },
16     dialogFormVisible: false,
17     rules: {
18       name: [{ required: true, message: '商品名称不能为空', trigger: 'blur' }],
19       code: [{ required: true, message: '商品编码不能为空', trigger: 'blur' }],
20       retailPrice: [{ required: true, message: '零售价不能为空', trigger: 'blur' }],
21     },
22   }
23 },
24 . . . ,
```

3. 在 `methods` 中添加 `addData()` 函数,提交表单数据用的

```
1 // 提交新增数据
2 addData(formName) {
3   this.$refs[formName].validate(valid => {
4     if (valid) {
5       // 验证通过 , 提交添加
6       alert('Add submit!');
7     } else {
8       // 验证不通过
9       return false;
10    }
11  })
12},
```

4. 在 template 中的查询按钮旁边添加一个 `新增` 按钮 , 用于打开新增会员对话框



```
1 <el-button type="primary" icon="el-icon-edit" @click="handleAdd">新增</el-button>
```

5. 在 `methods` 中添加 `handleAdd()` 函数, 打开新增对话框

关闭窗口后，再次打开窗口，会发现表单里依然有数据，应当清除数据。

```
1 // 打开新增窗口
2 handleAdd() {
3     this.dialogFormVisible = true
4     this.$nextTick(() => {
5         // this.$nextTick()它是一个异步事件，当渲染结束之后，它的回调函数才会被执行
6         // 弹出窗口打开之后，需要加载Dom, 就需要花费一点时间，我们就应该等待它加载完dom之后，再进行调
7         // 用resetFields方法，重置表单和清除样式
8         this.$refs['pojoForm'].resetFields()
9     })
}
```

6. 测试发现，点击选择供应商，数据没有回显，实际上是回显到搜索框(searchMap)了，

解决：定义一个中间变量 `isEdit: false`, 打开时赋值 `true`, 然后在回显处，判断回显 `searchMap`还是 `pojo`

- 修改编辑窗口的供应商名称输入框绑定一个处理函数 `editOptionSupplier`

```
1 <el-form-item label="供应商名称" prop="supplierName">
2     <el-input readonly
3         @click.native="editOptionSupplier"
4         v-model="pojo.supplierName"
5         placeholder="选择供应商" style="width: 200px;" ></el-input>
6 </el-form-item>
```

- data 中声明一个 `isEdit: false`，在 `methods` 中定义函数 `editOptionSupplier`，
修改 `optionSupplier` 函数作判断赋值。

```
1 data() {
2     return {
3         ...
4         isEdit: false
5     },
6     methods: {
7         //编辑窗口打开供应商
8         editOptionSupplier() {
9             this.isEdit = true // 是编辑窗口
10            this.dialogSupplierVisible = true
11        },
12        optionSupplier(current) { // current 子组件传递的数据
13            console.log('goods', current)
14            // 判断是否编辑
15            if( this.isEdit ) {
```

```
19     this.pojo.supplierName = current.name //供应商名称
20     this.pojo.supplierId = current.id // 供应商 id
21   }else {
22     this.searchMap.supplierName = current.name //供应商名称
23     this.searchMap.supplierId = current.id // 供应商 id
24   }
25   this.setEditable = false // 注意重新赋值 false
26   this.dialogSupplierVisible = false // 关闭窗口
27 },
```

18.3 表单数据提交

当点击新增窗口中的确认按钮时，提交表单数据，后台API服务接口响应新增成功或失败。

18.3.1 EasyMock 添加新增商品模拟接口

- 请求URL : /goods
- 请求方式 : post
- 描述 : 新增商品
- mock.js 配置 :

```
1  {
2   "code": 2000,
3   "flag": true,
4   "message": "新增成功"
5 }
```

18.3.2 Api 调用接口

1. src\api\goods.js 导出的默认对象中，添加调用新增接口的方法

```
1  // 新增
2  add(pojo) {
3    return request({
4      url: '/goods',
5      method: 'post',
6      data: pojo
7    })
8 }
```

2. 在 src\views\goods\index.vue 中的 addData 方法中提交数据，代码如下：

```
1  // 提交新增数据
2  addData(formName) {
3    this.$refs[formName].validate(valid => {
4      if (valid) {
```

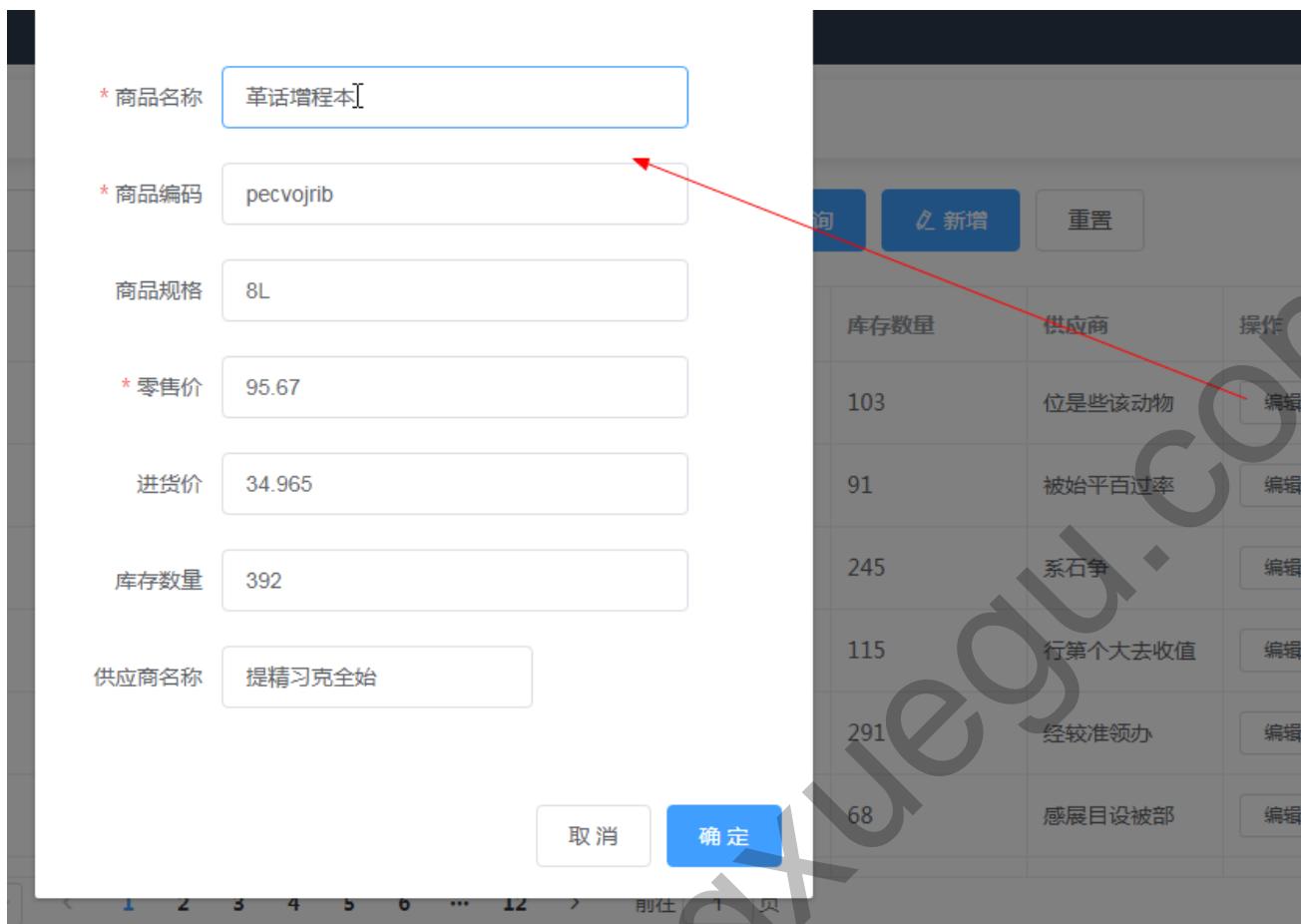
```
5      // 验证通过，提交添加
6      goodsApi.add(this.pojo).then(response => {
7          const resp = response.data
8          if(resp) {
9              this.fetchData()
10             this.dialogFormVisible = false
11         }else{
12             // 验证不通过
13             this.$message({
14                 message: resp.message,
15                 type: 'warning'
16             })
17         }
18     })
19 } else {
20     return false;
21 }
22 })
23 },
```

3. 测试新增功能。

第十九章 商品管理-修改

19.1 需求分析

当点击 编辑 按钮后，弹出编辑商品窗口，并查询出信息渲染。修改后点击确定提交修改数据。



19.2 EasyMock 添加模拟接口

19.2.1 ID 查询数据模拟接口

- 请求URL : /goods/{id}
- 请求方式 : get
- 描述 : 商品 ID 查询数据接口
- mock.js 配置 :

```
1  {
2      "code": 2000,
3      "flag": true,
4      "message": "查询成功",
5      "data": {
6          "id|+1": 10,
7          "name|5": "@cword",
8          "code": "@word",
9          "spec": "@integer(0,9)L", // 11个数字0-9间的数字
10         "retailPrice": "@float(0, 100, 1, 3)", // 0-1000小数,1-3位小数位
11         "purchasePrice": "@float(0, 100, 1, 3)", // 0-1000小数,1-3位小数位
12         "storageNum": "@integer(0,500)", // 4选 其1
13         "supplierName": "@ctitle",
14         "supplierID": "@integer(1,500)"}
```

```
15 }
16 }
```

2.2 提交修改数据接口

- 请求URL : `/goods/{id}`

说明 : id 是修改的那条数据

- 请求方式 : `put`
- 描述 : 商品数据更新
- mock.js 配置 :

```
1 {
2   "code": 2000,
3   "flag": true,
4   'message': "修改成功"
5 }
```

19.3 Api 调用接口

1. `src\api\goods.js` 导出的默认对象中 , 添加ID查询方法 `getById` 和 更新方法 `update`

```
1 // 查询
2 getById(id) {
3   return request({
4     url: `/goods/${id}` , // 反单引号 ``
5     method: 'get'
6   })
7 },
8
9 // 更新
10 update(pojo) {
11   return request({
12     url: `/goods/${pojo.id}` , // 反单引号 ``
13     method: 'put' , // put 方式提交
14     data: pojo
15   })
16 }
```

2. 在 `src\views\goods\index.vue` 中的 `handleEdit` 方法做如下修改 :

```
1 // 打开编辑窗口
2 handleEdit(id) {
3     // 重用打开新增窗口方法, 不要少了 this
4     this.handleAdd()
5     // 查询数据
6     goodsApi.getById(id).then(response => {
7         const resp = response.data
8         if(resp.flag) {
9             this.pojo = resp.data
10        }
11    })
12},
```

3. 测试，打开修改页面是否有数据, 如果报404, 看下是easyMock上URL错了或者goods.js 路径错了
4. 修改对话框中的确认按钮点击事件，`pojo.id === undefined` 值为true是新增，有id说明是更新。

```
1 <el-button type="primary" @click="pojo.id === undefined ? addData('pojoForm'):
updateData('pojoForm')">确定</el-button>
```

5. 提交更新数据

```
1 updateData(formName) {
2     this.$refs[formName].validate(valid => {
3         if (valid) {
4             // 验证通过，提交添加
5             supplierApi.update(this.pojo).then(response => {
6                 const resp = response.data
7                 if(resp.flag) {
8                     this.fetchData()
9                     this.dialogFormVisible = false
10                }else{
11                    // 验证不通过
12                    this.$message({
13                        message: resp.message,
14                        type: 'warning'
15                    })
16                }
17            })
18        } else {
19            return false;
20        }
21    })
22},
```

6. 测试效果

第二十章 商品管理-删除

20.1 需求分析

当点击删除按钮后，弹出提示框，点击确定后，执行删除并刷新列表数据

确认消息弹框参考：<https://element.eleme.cn/#/zh-CN/component/message-box#que-ren-xiao-xi>

20.2 EasyMock 添加模拟接口

- 请求URL：/goods/{id}
- 请求方式：delete
- 描述：根据商品ID删除
- mock.js 配置：

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": "删除成功"
5 }
```

20.3 API 调用接口

1. 在 src\api\goods.js 导出的默认对象中，添加 deleteById 方法

```
1  deleteById(id) {
2    return request({
3      url: `/goods/${id}`， // 反单引号 ``
4      method: 'delete'， // delete 方式提交
5    })
6 }
```

2. 在 src\views\goods\index.vue 中的 handleDelete 方法做如下修改：

```
1  handleDelete(id) {
2    console.log('删除', id)
3    this.$confirm('确认删除这条记录吗？', '提示', {
4      confirmButtonText: '确定',
5      cancelButtonText: '取消',
6      type: 'warning'
7    }).then(() => {
8      // 确认
9      memberApi.deleteById(id).then(response => {
10        const resp = response.data
11        // 提示信息
12        this.$message({
```

```
13     type: resp.flag ? 'success': 'error',
14     message: resp.message
15   });
16   if(resp.flag) {
17     // 删除成功，刷新列表
18     this.fetchData()
19   }
20 }
21 }).catch(() => {
22   // 取消删除，不理会
23 });
24 },
```

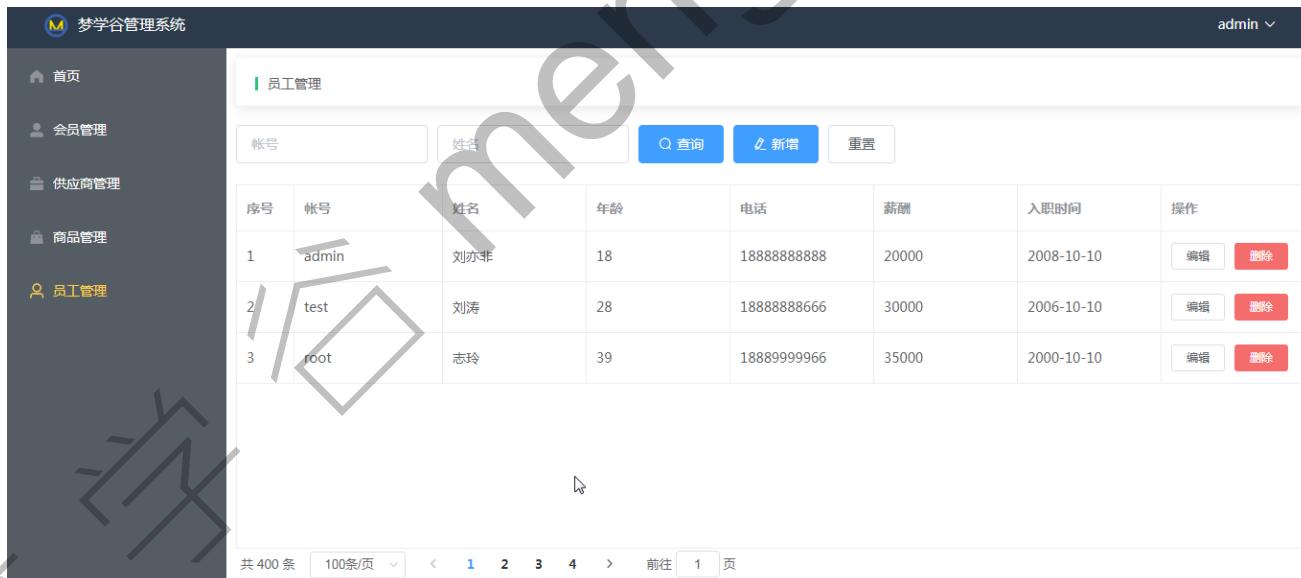
第二十一章 员工管理-练习

1. 员工管理功能由学员自行练习，接口规范按下面的要求

2. 老师已经在 `mock.mengxuegu.com` 上创建好了下面的接口.大家可以直接使用老师的接口地址, 地址如下

<http://mengxuegu.com:7300/mock/5d477ccbfacc296cd6834fe5>

21.1 查询员工列表服务接口



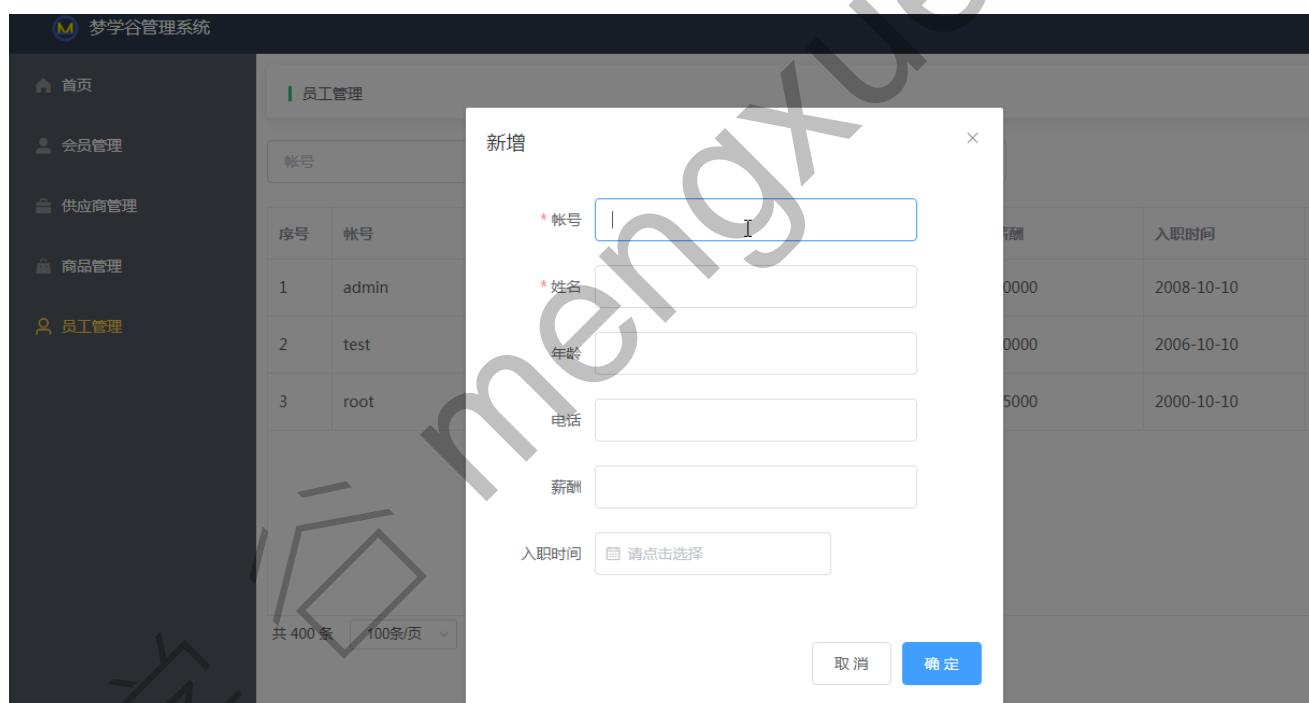
序号	帐号	姓名	年龄	电话	薪酬	入职时间	操作
1	admin	刘亦非	18	18888888888	20000	2008-10-10	<button>编辑</button> <button>删除</button>
2	test	刘涛	28	18888888666	30000	2006-10-10	<button>编辑</button> <button>删除</button>
3	root	志玲	39	18889999966	35000	2000-10-10	<button>编辑</button> <button>删除</button>

- 请求URL : `/staff/list/search/{page}/{size}`
- 请求方式 : `post`
- 描述 : 员工分页数据列表
- mock.js 配置 :

```
1  {
2    "code": 2000,
```

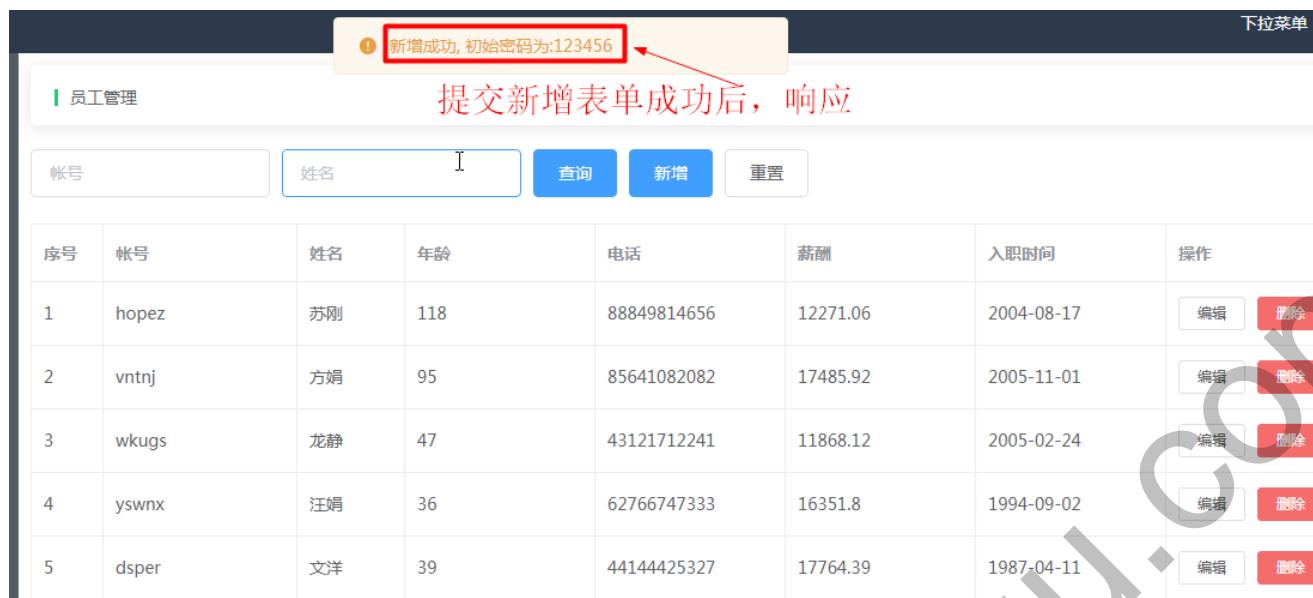
```
3 "flag": true,
4 "message": "查询成功",
5 "data": {
6   "total": "@integer(100,200)", // 查询出来的总记录数
7   "rows|10": [ // 返回当前页的记录数 10 条, 即每页显示 10 条记录
8     "id|+1": 10,
9     "username|5": "@word(1)",
10    "name": "@cname",
11    "age": "@integer(0,120)",
12    "mobile|11": "@integer(0,9)",
13    "salary": "@float(8000, 20000, 1, 2)",
14    "entryDate": "@date"
15  ]
16 }
17 }
```

21.2 新增员工服务接口



下拉菜单

提交新增表单成功后，响应



序号	帐号	姓名	年龄	电话	薪酬	入职时间	操作
1	hopez	苏刚	118	88849814656	12271.06	2004-08-17	<button>编辑</button> <button>删除</button>
2	vntnj	方娟	95	85641082082	17485.92	2005-11-01	<button>编辑</button> <button>删除</button>
3	wkugs	龙静	47	43121712241	11868.12	2005-02-24	<button>编辑</button> <button>删除</button>
4	yswnx	汪娟	36	62766747333	16351.8	1994-09-02	<button>编辑</button> <button>删除</button>
5	dsper	文洋	39	44144425327	17764.39	1987-04-11	<button>编辑</button> <button>删除</button>

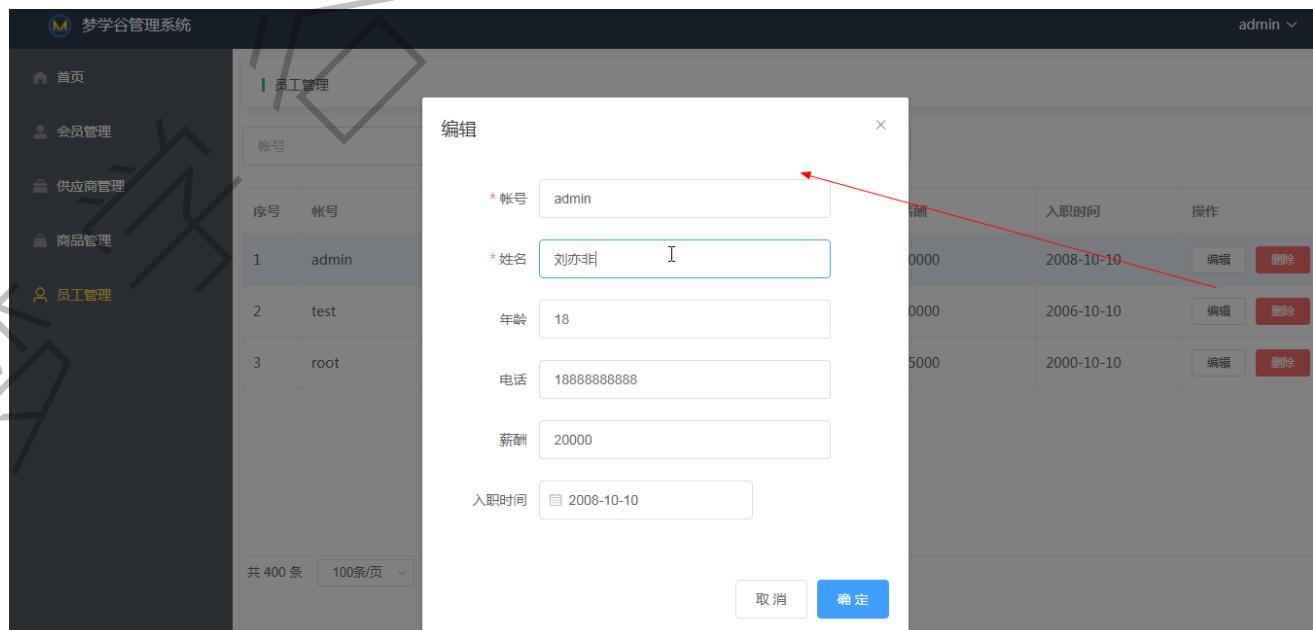
- 请求URL : /staff
- 请求方式 : post
- 描述 : 新增员工
- mock.js 配置 :

```

1  {
2      "code": 2000,
3      "flag": true,
4      "message": "新增成功"
5  }

```

21.3 编辑员工服务接口



The screenshot shows the 'Employee Management' section of the '梦学谷管理系统'. A modal dialog titled '编辑' (Edit) is open, showing fields for updating an employee's information. The fields include: 帐号 (Account) with value 'admin', 姓名 (Name) with value '刘亦非', 年龄 (Age) with value '18', 电话 (Phone) with value '18888888888', 薪酬 (Salary) with value '20000', and 入职时间 (Hire Date) with value '2008-10-10'. The background shows a list of employees with columns: 序号 (ID), 帐号 (Account), 姓名 (Name), 年龄 (Age), 电话 (Phone), 薪酬 (Salary), 入职时间 (Hire Date), and 操作 (Operations). The account 'admin' is selected in the list.

21.3.1 打开编辑窗口回显数据

- 请求URL : /staff/{id}
- 请求方式 : get
- 描述 : 通过员工ID查询数据
- mock.js 配置 :

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": "查询成功",
5    "data": {
6      "id": 10,
7      "username|5": "@word(1)",
8      "name": "@cname",
9      "age": "@integer(0,120)",
10     "mobile|11": "@integer(0,9)",
11     "salary": "@float(8000, 20000, 1, 2)",
12     "entryDate": "@date"
13   }
14 }
```

21.3.2 提交修改数据

- 请求URL : /staff/{id}
- 请求方式 : put
- 描述 : 修改员工信息
- mock.js 配置 :

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": "修改成功"
5 }
```

21.4 删除员工服务接口

需求: 先提示是否确认删除, 确认后才进行删除操作

- 请求URL : /staff/{id}
- 请求方式 : delete
- 描述 : 通过 ID 删除员工
- mock.js 配置 :

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": "删除成功"
5 }
```

第二十二章 修改密码

参考：<https://element.eleme.cn/#/zh-CN/component/form#zi-ding-yi-xiao-yan-gui-ze>

22.1 EasyMock 添加模拟接口

22.2 添加原密码校验接口

输入原密码，失去焦点后异步发送请求校验是否正确，给服务接口传入用户id和密码

- 请求URL： /user/pwd
- 请求方式： post
- 请求参数： userId, password
- 描述： 校验密码是否正确
- mock.js 配置：

```
1  {
2    "code": 2000,
3    "flag": true,
4    "message": "密码正确"
5 }
```

22.3 添加修改密码接口

点击提交，将用户id和新密码提交给后台接口

- 请求URL： /user/pwd
- 请求方式： put
- 请求参数： userId, password
- 描述： 修改密码接口
- mock.js 配置：

```
1  {
2      "code": 2000,
3      "flag": true,
4      "message": "密码修改成功"
5  }
```

22.2 Api 调用接口

新建 api\password.js 文件, 调用服务接口实现如下:

```
1 import request from '@/utils/request'
2
3 export default {
4
5     // 校验密码是否正确
6     checkPwd(userId, password) {
7         return request({
8             url: '/user/pwd',
9             method: 'post',
10            data: {
11                userId,
12                password
13            }
14        })
15    },
16
17     // 修改密码
18     updatePwd(userId, password) {
19         return request({
20             url: '/user/pwd',
21             method: 'put',
22             data: {
23                 userId,
24                 password
25             }
26         })
27     }
28 }
29 }
```

22.3 初始化修改密码组件

1. 打开组件, 添加修改密码表单组件

```
1 <el-dialog title="修改密码" :visible.sync="dialogFormVisible" width="500px">
2     <el-form :model="ruleForm" status-icon :rules="rules"
```

```
3      ref="ruleForm" label-width="100px" style="width: 400px;">>
4      <el-form-item label="原密码" prop="oldPass">
5          <el-input type="password" v-model="ruleForm.oldPass"></el-input>
6      </el-form-item>
7      <el-form-item label="新密码" prop="pass">
8          <el-input type="password" v-model="ruleForm.pass"></el-input>
9      </el-form-item>
10     <el-form-item label="确认密码" prop="checkPass">
11         <el-input type="password" v-model="ruleForm.checkPass" ></el-input>
12     </el-form-item>
13     <el-form-item>
14         <el-button type="primary" @click="submitForm(ruleForm)">提交</el-button>
15         <el-button @click="$refs['ruleForm'].resetFields()">重置</el-button>
16     </el-form-item>
17 </el-form>
18 </el-dialog>
```

2. data 选项中初始化属性值

```
1 data() {
2     // 注意是在 return 上面
3     return {
4         dialogFormVisible: false,
5         ruleForm: {
6             oldPass: '',
7             pass: '',
8             checkPass: ''
9         },
10        rules: {}
11    },
12},
```

3. 在 handleCommand 方法中调用 handlePwd 方法

```
1 methods: {
2     handleCommand(command) {
3         switch (command) {
4             case 'a':
5                 // 打开修改密码窗口
6                 this.handlePwd()
7                 break;
8             case 'b':
9                 // 退出系统
10                this.handleLogout()
11                break;
12            default:
13                break;
14        }
15    },
16    handleLogout() {
17        此处抽取了上面 退出系统 逻辑代码
18    },
19}
```

19 }

4. 添加 handlePwd 方法, 打开修改密码 窗口

```
1 handlePwd() {
2     this.dialogFormVisible = true
3     this.$nextTick(() => {
4         this.$refs['ruleForm'].resetFields()
5     })
6 },
```

5. 表单重置

```
1 <el-button @click="$refs['ruleForm'].resetFields()">重置</el-button>
```

6. 添加 submitForm 方法, 提交修改密码表单

```
1 submitForm(formName) {
2     this.$refs[formName].validate(valid => {
3         if (valid) {
4             // 验证通过，提交添加
5         } else {
6             // 验证不通过
7             return false;
8         }
9     })
10 }
```

22.4 校验密码是否正确

输入原密码，失去焦点后异步发送请求校验是否正确，给服务接口传入用户名和密码

1. 在 data 选项添加 user 获取用户信息 和 rules 属性中添加 校验规则

```
1 data() {
2     // 注意是在 return 上面
3     return {
4         ...
5         // user 获取用户信息
6         user: JSON.parse(localStorage.getItem('mxg-msm-user')),
7         rules: {
8             oldPass: [
9                 { required: true, message: '旧密码不能为空', trigger: 'blur' },
10                { validator: validateOldPass, trigger: 'blur' }
11            ],
12            pass: [
13                { required: true, message: '新密码不能为空', trigger: 'blur' }
```

```
14      ],
15      checkPass: [
16          { required: true, message: '确认密码不能为空', trigger: 'blur' },
17          { validator: validatePass, trigger: 'change' }
18      ],
19  }
20 }
21 },
```

2. 在 data 选项中的 `return {}` 上面添加自定校验

```
1 data() {
2     //校验原密码
3     const validateOldPass = (rule, value, callback) => {
4         // alert(this.user.id)
5         passwordApi.checkPwd(this.user.id, value).then(response => {
6             const resp = response.data
7             if(resp.flag) {
8                 callback()
9             }else{
10                 callback(new Error( resp.message ))
11             }
12         })
13     }
14     // rule 当前校验表单对象
15     const validatePass = (rule, value, callback) => {
16         if(value != this.ruleForm.pass) {
17             callback(new Error('两次输入密码不一致!'));
18         }else {
19             callback()
20         }
21     }
22
23     // 注意是在 return 上面
24     return {
25         ...
26     }
27     ...
28 }
```

22.5 提交新密码

- 在 `submitForm` 中提交修改，修改成功后退出登录，关闭窗口

```
1 submitForm(formName) {
2     this.$refs[formName].validate(valid => {
3         if (valid) {
4             // 验证通过，提交添加
```

```
5         passwordApi.updatePwd(this.userId, this.ruleForm.checkPass)
6         .then(response => {
7             const resp = response.data
8             this.$message({
9                 message: resp.message,
10                type: resp.flag ? 'success': 'warning'
11            })
12            if(resp.flag) {
13                // 修改成功, 清除本地数据, 重新登录
14                this.handleLogout()
15                // 关闭窗口
16                this.dialogFormVisible = false
17            }
18        })
19
20    } else {
21        // 验证不通过
22        return false;
23    }
24}
25}
```

第二十三章 加载效果和异常处理

23.1 Loading 加载数据时显示动效

作用：加载数据时，实现加载中动效，提高用户体验度。

参考：<https://element.eleme.cn/#/zh-CN/component/loading#fu-wu>

1. 按需导入 ElementUI 组件 Loading, Message

```
1 // 按需导入 ElementUI 组件
2 import { Loading, Message } from 'element-ui'
```

2. 加载数据时，打开和关闭动效 对象

注意：loadingInstance 实例采用单例模式创建，防止响应异常时频繁切换路由时加载动效重复创建

```
1 // 加载数据时打开和关闭动效对象
2 const loading = {
3     loadingInstance: null, // Loading实例
4     open: function () { // 打开加载
5         console.log('加载中', this.loadingInstance)
6         if(this.loadingInstance === null) { // 创建单例, 防止切换路由重复加载
7             console.log('创建加载实例..')
8             this.loadingInstance = Loading.service({
9                 text: '拼命加载中',
10                target: '.main', // 效果显示区域
11            })
12        }
13    },
14    close: function () { // 关闭加载
15        if(this.loadingInstance) {
16            Loading.close(this.loadingInstance)
17        }
18    }
19}
```



```
11         background: 'rgba(0, 0, 0, 0.5)' // 加载效果
12     })
13 }
14 },
15 close: function() { // 关闭加载
16     if(this.loadingInstance != null){
17         this.loadingInstance.close()
18         console.log('结束加载')
19     }
20     this.loadingInstance = null // 关闭后实例重新赋值null, 下次让它重新创建
21 }
22 }
```

3. 请求拦截器中打开加载效果, 响应拦截器和异常中关闭加载效果.

```
1 // 请求拦截器
2 request.interceptors.request.use(config => {
3     loading.open() // 打开加载效果
4     return config
5 }, error => {
6     // 出现异常
7     loading.close() // 关闭加载效果
8     return Promise.reject(error);
9 })
10
11 // 响应拦截器
12 request.interceptors.response.use(response =>{
13     loading.close() // 关闭加载效果
14     return response
15 }, error => {
16     loading.close() // 关闭加载效果
17     return Promise.reject(error);
18 })
```

4. mxg-msm\src\utils\request.js 完整代码

```
1 import axios from 'axios'
2 // 按需导入 ElementUI 组件
3 import { Loading, Message } from 'element-ui'
4
5 // 加载数据时打开和关闭动效对象
6 const loading = {
7     loadingInstance : null, // Loading实例
8     open: function () { // 打开加载
9         console.log('加载中', this.loadingInstance)
10        if(this.loadingInstance === null) { // 创建单例, 防止切换路由重复加载
11            console.log('创建加载实例..')
12            this.loadingInstance = Loading.service({
13                text: '拼命加载中',
14                target: '.main', // 效果显示区域
15                background: 'rgba(0, 0, 0, 0.5)' // 加载效果
16            })
17        }
18    },
19    close: function () { // 关闭加载
20        if(this.loadingInstance) {
21            this.loadingInstance.close()
22        }
23    }
24 }
```

```
17      }
18    },
19    close: function() { // 关闭加载
20      if(this.loadingInstance != null){
21        this.loadingInstance.close()
22        console.log('结束加载')
23      }
24      this.loadingInstance = null // 关闭后实例重新赋值null, 下次让它重新创建
25    }
26  }
27
28 const request = axios.create({
29   baseURL: process.env.VUE_APP_BASE_API,
30   timeout: 5000 // 请求超时 , 5000毫秒
31 })
32
33 // 请求拦截器
34 request.interceptors.request.use(config => {
35   loading.open() // 打开加载效果
36   return config
37 }, error => {
38   // 出现异常
39   loading.close() // 关闭加载效果
40   return Promise.reject(error);
41 })
42
43 // 响应拦截器
44 request.interceptors.response.use(response =>{
45   loading.close() // 关闭加载效果
46   return response
47 }, error => {
48   loading.close() // 关闭加载效果
49   return Promise.reject(error);
50 })
51
52 export default request // 导出自定义创建 axios 对象
```

23.2 异常统一处理

请求接口超时，应该EasyMock服务接口不稳定

```
✖ Uncaught (in promise) Error: timeout of 5000ms exceeded
  at e.exports (chunk-vendors.6865ea00.js:7)
  at XMLHttpRequest.h.ontimeout (chunk-vendors.6865ea00.js:27)
```

一般都是在请求后台服务接口时会出现异常，而请求服务接口都是通过 axios 发送的请求，所以可在 axios 响应拦截器进行统一处理。

1. 检查 Message 组件是导入



```
1 import { Loading, Message } from 'element-ui'
```

2. 在 axios 响应拦截器中处理异常，即：mxg-msm\src\views

```
1 // 响应拦截器
2 request.interceptors.response.use(response => {
3   loading.close() // 关闭加载效果
4
5   const resp = response.data
6   // 如果后台响应状态码不是 2000，说明后台服务有异常，统一可在此处处理
7   if(resp.code !== 2000) {
8     Message({
9       message: res.message || '系统异常',
10      type: 'warning',
11      duration: 5 * 1000 // 停留时长
12    })
13  }
14
15  return response
16 }, error => {
17   loading.close() // 关闭加载效果
18   // 当请求接口出错时，进行弹出错误提示，如 404, 500, 请求超时
19   console.log('response error', error.response.status)
20   Message({
21     message: error.message,
22     type: 'error',
23     duration: 5 * 1000
24   })
25   return Promise.reject(error);
26 })
```

第二十四章 Vuex 状态管理

24.1 Vuex 概述

1. 官方文档：<https://vuex.vuejs.org/zh/>

- Vuex 是一个专为 Vue.js 应用程序开发的**状态管理模式**。它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。

2. Vuex简单理解：

- Vue 应用中的每个组件在 data() 中封装着自己数据属性，而这些 data 属性都是私有的，完全隔离的。
- 如果我们希望多个组件都能读取到同一状态数据属性，或者不同组件的行为需要更新同一状态数据属性，
这就需要一个将共享的状态数据属性进行集中式的管理。
- 这就是 Vuex 状态管理所要解决的问题。

24.2 Vuex 入门实战

共享 count 状态数据

24.2.1 初始化 Vuex 工程

1. 解压 `mxg-msm-解手架模板.zip` 到 `D:\StudentProject\VueProject\` 下，重命名为 `vuex-demo`

`mxg-msm-解手架模板.zip` 位于网盘：Vue-CLI 综合实战项目资源\mxg-msm-解手架模板.zip

2. 修改 `package.json` 中 `name` 值为 `vuex-demo`

```
{  
  "name": "vuex-demo",  
  "version": "0.1.0",  
  "private": true,
```

3. 安装依赖和运行

```
1 # 进入工程目录  
2 cd vuex-demo  
3 # 安装依赖  
4 npm install --save vuex  
5 # 运行项目  
6 npm run serve
```

4. 访问 <http://localhost:8080/>

24.2.2 读取状态值 state

24.2.2.1 理解

- 每一个 Vuex 项目的核心就是 `store`（仓库）。`store` 就是一个对象，它包含着你的项目中大部分的状态（state）。
- `state` 是 `store` 对象中的一个选项，是 Vuex 管理的状态对象（共享的数据属性）

24.2.2.2 实操

1. 在 `src` 目录下创建 `store` 目录，`store` 下创建 `index.js` 文件，编码如下：

```
1 import Vue from 'vue'  
2 import Vuex from 'vuex'  
3  
4 // 引入 Vuex 插件  
5 Vue.use(Vuex)  
6  
7 const store = new Vuex.Store({ // 注意V和S都是大写字母  
8   // 存放状态（共享属性）  
9   state: {
```

```
10      count: 1
11    }
12  })
13
14 export default store
```

2. 修改 main.js , 导入和注册 store , 编码如下 :

```
1 import Vue from "vue";
2 import App from "./App.vue";
3 import router from "./router";
4 // 导入的是 src/store/index.js
5 import store from './store'
6
7 Vue.config.productionTip = false;
8
9 new Vue({
10   router,
11   store, // 注册
12   render: h => h(App)
13 }).$mount("#app");
```

3. 组件中读取 state 状态数据 , 修改 src\views\Home.vue , 编码如下 :

```
1 <template>
2   <div>
3     count: {{ $store.state.count }}
4   </div>
5 </template>
6
7 <script>
8 </script>
```

4. 访问 <http://localhost:8080/#/> 页面显示 count: 1

24.2.3 改变状态值 mutation

24.2.3.1 理解

1. 在 store 的 mutations 选项中定义方法 , 才可以改变状态值。
2. 在通过 \$store.commit('mutationName') 触发状态值的改变

24.2.3.2 实操

1. 修改 store/index.js , 在 store 中添加 mutations 选项 , 编码如下 :

```
1 import Vue from 'vue'
2 import Vuex from 'vuex'
3
4 // 引入 Vuex 插件
5 Vue.use(Vuex)
```



```
6
7 const store = new Vuex.Store({ // 注意V和S都是大写字母
8   // 存放状态（共享属性）
9   state: {
10     count: 1
11   },
12   // 改变 state 状态
13   mutations: {
14     increment(state) {
15       state.count ++
16     },
17     decrement(state) {
18       state.count --
19     }
20   }
21 })
22
23 export default store
```

2. 修改 src\views\Home.vue , 调用 mutations 中 increment、decrement 方法

```
1 <template>
2   <div>
3     count: {{ $store.state.count }}
4     <button @click="addCount">加法</button>
5     <button @click="decrement">减法</button>
6   </div>
7 </template>
8
9 <script>
10 export default {
11   methods: {
12     addCount() {
13       // 获取状态值
14       console.log(this.$store.state.count)
15       // 通过commit调用mutations中的increment改变状态值
16       this.$store.commit('increment')
17     },
18     decrement() {
19       // 通过commit调用mutations中的decrement改变状态值
20       this.$store.commit('decrement')
21     }
22   },
23 } </script>
```

3. 访问：点击 加法 按钮，控制台和页面显示数字变化

24.2.4 不同组件共享状态值

24.2.4.1 疑问：

- 在About组件中，是否可以读取到在Home.vue中改变的状态值？

下面进行测试

24.2.4.2 实操

- 修改src\views\About.vue，获取状态值count

```
1 <template>
2   <div>
3     About: {{ $store.state.count }}
4   </div>
5 </template>
```

- 访问：点击Home页面的加法按钮，访问About页面是否获取到最新状态值

24.2.5 提交载荷

你可以向`$store.commit`传入额外的参数，即mutation的载荷(payload)：

- 修改src\store下的index.js

```
1 // ...
2 mutations: {
3   increment(state, n){ // n为载荷
4     state.count += n
5   }
6 }
7 // ...
```

- 修改views\Home.vue组件

```
1 // ...
2 <script>
3 export default {
4   methods: {
5     addCount() {
6       // 获取状态值
7       console.log(this.$store.state.count)
8       // 通过commit调用mutations中的increment改变状态值
9       // this.$store.commit('increment')
10      this.$store.commit('increment', 10) // 提交载荷
11    }
12  }
13  // ...
14 },
15 }
16 </script>
```

24.2.6 Action

24.2.6.1 理解

Action 类似于 mutation , 不同在于 :

- Action 提交的是 mutation , 而不是在组件中直接变更状态 , 通过它间接更新 state。
- 在组件中通过 this.\$store.dispatch('actionName') 触发状态值间接改变
- Action 也支持载荷
- Action 可以包含任意异步操作。

24.2.6.2 实操

1. 修改 store/index.js , 增加 actions 选项

```
1 const store = new Vuex.Store({ // 注意V和S都是大写字母
2   // 存放状态 ( 共享属性 )
3   state: {
4     count: 1
5   },
6   // 改变 state 状态
7   mutations: {
8     increment(state, n) { // n 为载荷
9       // state.count ++
10      state.count += n
11    },
12     decrement(state) {
13       state.count --
14     }
15   },
16   actions: {
17     add(context, n) {
18       // 触发 mutations 中的 increment 改变 state
19       context.commit('increment', n)
20     },
21     decrement({commit, state}) { // 按需传值
22       commit('decrement')
23     }
24   }
25 })
```

2. 修改 views\Home.vue, 触发 action 改变值

```
1 <script>
2 export default {
3   methods: {
```

```
4  addCount() {
5    // 获取状态值
6    console.log(this.$store.state.count)
7    // 通过commit 调用 mutations 中的 increment 改变状态值
8    // this.$store.commit('increment')
9    // this.$store.commit('increment', 10) // 提交载荷
10   // 触发 actions 中的 add 改变状态值
11   this.$store.dispatch('add', 10)
12 },
13 decrement(){
14   // this.$store.commit('decrement')
15   this.$store.dispatch('decrement')
16 }
17 },
18 }
19 </script>
```

24.2.7 派生属性 getter

24.2.7.1 理解

- 有时候我们需要从 store 中的 state 中派生出一些状态。

例如：基于上面代码，增加一个 desc 属性，当 count 值小于 50，则 desc 值为 吃饭；大于等于 50 小于 100，则desc 值为 睡觉；大于100，则 desc 值为 打豆豆。这时我们就需要用到 getter 为我们解决。

- getter 其实就类似于计算属性(get)的对象.
- 组件中读取 \$store.getters.xxx

24.2.7.2 实操

1. 修改 store\index.js ,增加 getters 选项

注意：getters 中接受 state 作为其第一个参数，也可以接受其他 getter 作为第二个参数

```
1 const store = new Vuex.Store({ // 注意V 和 S都是大写字母
2   ...
3
4   //派生属性
5   getters: {
6     desc(state) {
7       if(state.count < 50) {
8         return '吃饭'
9       }else if(state.count < 100) {
10        return '睡觉'
11      }else {
12        return '打豆豆'
13      }
14    }
15  }
```

16 })

2. 修改 views\Home.vue, 显示派生属性的值

```
1 <template>
2 <div>
3   count: {{ $store.state.count }}
4   <button @click="addCount">加法</button>
5   <button @click="decrement">减法</button>
6   派生属性desc: {{ $store.getters.desc }}
7 </div>
8 </template>
```

3. 访问：点击 Home 页面的 触发改变 按钮，当 count 新增到 61 , desc 是否会显示为 睡觉

24.3 Module模块化项目结构

24.3.1 Module

由于使用单一状态树，应用的所有状态会集中到一个比较大的对象。

当应用变得非常复杂时，store 对象就有可能变得相当臃肿。

为了解决以上问题，Vuex 允许我们将 store 分割成模块（module）。

每个模块拥有自己的 state、mutation、action、getter 等，参见以下代码模型

```
1 const moduleA = {
2   state: { ... },
3   mutations: { ... },
4   actions: { ... },
5   getters: { ... }
6 }
7
8 const moduleB = {
9   state: { ... },
10  mutations: { ... },
11  actions: { ... }
12 }
13
14 const store = new Vuex.Store({
15   modules: {
16     a: moduleA,
17     b: moduleB
18   }
19 })
20
21 store.state.a // -> moduleA 的状态
22 store.state.b // -> moduleB 的状态
```



24.3.2 工程模块化进行改造

1. 修改 store\index.js

```
1 import Vue from 'vue'
2 import Vuex from 'vuex'
3
4 // 引入 Vuex 插件
5 Vue.use(Vuex)
6
7 const home = {
8     // 存放状态 ( 共享属性 )
9     state: {
10         count: 1
11     },
12     // 派生属性
13     getters: {
14         desc(state) {
15             if(state.count < 50) {
16                 return '吃饭'
17             }else if(state.count < 100) {
18                 return '睡觉'
19             }else {
20                 return '打豆豆'
21             }
22         }
23     },
24     // 改变 state 状态
25     mutations: {
26         increment(state, n) { // n 为载荷
27             // state.count ++
28             state.count += n
29         },
30         decrement(state) {
31             state.count --
32         }
33     },
34     actions: {
35         add(context) {
36             // 触发 mutations 中的 increment 改变 state
37             context.commit('increment', 10)
38         },
39         decrement({commit, state}) { // 按需传值
40             commit('decrement')
41         }
42     }
43 }
44
45 const store = new Vuex.Store({ // 注意V 和 S都是大写字母
46     modules: {
47         home // home: home
```

```
48      }
49  })
50
51  export default store
```

2. 修改 Home.vue

```
1  <template>
2    <div>
3      <!--修改部分-->
4      count: {{ $store.state.home.count }}
5      <button @click="addCount">加法</button>
6      <button @click="decrement">减法</button>
7      派生属性desc: {{ $store.getters.desc }}
8    </div>
9  </template>
10
11 <script>
12 export default {
13   methods: {
14     addCount() {
15       console.log(this.$store.state.home.count)
16       this.$store.dispatch('add', 10)
17     },
18     decrement(){
19       this.$store.dispatch('decrement')
20     }
21   },
22 }
23 </script>
```

3. 修改 About.vue

```
1  <template>
2    <div>
3      About: {{ $store.state.home.count }}
4    </div>
5  </template>
```

24.3.3 标准项目结构

24.3.3.1 参考

如果所有的状态都写在一个 js 中，这个 js 必定会很臃肿，Vuex 并不限制你的代码结构。但是它建议你按以下代码结构来构建项目结构：

```
1  |--- index.html
2  |--- main.js
```



```
3   ┌── api
4   |   └── ... # 抽取出API请求
5   ┌── components
6   |   └── App.vue
7   ┌── ...
8   └── store
9     ├── index.js    # 我们组装模块并导出 store 的地方
10    ├── actions.js  # 根级别的 action
11    ├── mutations.js # 根级别的 mutation
12    └── modules
13      ├── cart.js    # 购物车模块
14      └── products.js # 产品模块
```

24.3.3.2 重构项目结构

1. 在 store下创建 modules 目录，该目录下创建 home.js

```
1  const state = {
2    count: 1
3  }
4
5  const getters = {
6    desc(state){
7      if(state.count < 50){
8        return '吃饭'
9      }else if(state.count < 100){
10        return '睡觉'
11      }else {
12        return '打豆豆'
13      }
14    }
15  }
16
17 const mutations = {
18  increment(state, n){ // n 为载荷
19    // state.count ++
20    state.count += n
21  },
22  decrement(state) {
23    state.count --
24  }
25 }
26
27 const actions = {
28  add(context){
29    // 触发 mutations 中的 increment 改变 state
30    context.commit('increment', 10)
31  },
32  decrement({commit, state}){
33    commit('decrement')
34  }
35 }
36 export default {
```



```
37 // 存放状态（共享属性）  
38 state,  
39 //派生属性  
40 getters,  
41 // 改变 state 状态  
42 mutations,  
43 actions  
44 }
```

2. 修改 store\index.js, 导入 ./modules/home.js

```
1 import Vue from 'vue'  
2 import Vuex from 'vuex'  
3 // 导入 Module  
4 import home from './modules/home'  
5 // 引入 Vuex 插件  
6 Vue.use(Vuex)  
7 const store = new Vuex.Store({ // 注意V和S都是大写字母  
8   modules: {  
9     home // home: home  
10   }  
11 })  
12 export default store
```

3. 正常访问, 与重构前一样

第二十五章 会员管理系统-Vuex版

25.1 初始化项目

1. 复制 mxg-msm 工程为 mxg-msm-vuex
2. 安装 Vuex

```
1 # 进入工程目录  
2 cd mxg-msm-vuex  
3 # 安装依赖  
4 npm install --save vuex  
5 # 启动项目  
6 npm run serve
```

25.2 登陆

25.2.1 登录与 token 状态管理



1. 在 src\utils\ 目录下创建 auth.js , 封装 token 和 用户信息工具模块

```
1 const TOKEN_KEY = 'mxg-msm-token'
2 const USER_KEY = 'mxg-msm-user'
3
4 // 获取 token
5 export function getToken() {
6     return localStorage.getItem(TOKEN_KEY)
7 }
8 // 保存 token
9 export function setToken(token) {
10    return localStorage.setItem(TOKEN_KEY, token)
11 }
12 // 获取用户信息
13 export function getUser() {
14     return JSON.parse(localStorage.getItem(USER_KEY))
15 }
16 // 保存用户信息
17 export function setUser(user) {
18     localStorage.setItem(USER_KEY, JSON.stringify(user))
19 }
20 // 移除用户信息
21 export function removeToken() {
22     localStorage.removeItem(TOKEN_KEY)
23     localStorage.removeItem(USER_KEY)
24 }
```

2. 在 src 目录下新建 store\modules 目录 , modules 下创建 user.js

```
1 import { login, getUserInfo, logout } from '@/api/login'
2 import { getToken, setToken, setUser, getUser, removeToken } from '@/utils/auth'
3
4 const user = {
5     state: {
6         token: null,
7         user: null
8     },
9
10    mutations: {
11        SET_TOKEN(state, token) {
12            state.token = token
13            setToken(token)
14        },
15        SET_USER(state, user) {
16            state.user = user
17        }
18    },
19
20    actions: {
21        // 登录
22    }
23}
```



```
22     Login({ commit }, form) {
23         // 提交表单给后台进行验证是否正确
24         // resolve 触发成功处理，reject 触发异常处理
25         return new Promise((resolve, reject) => {
26             login(form.username.trim(), form.password).then(response => {
27                 const resp = response.data
28                 commit('SET_TOKEN', resp.data.token)
29                 resolve(resp)
30             }).catch(error => {
31                 reject(error)
32             })
33         })
34     },
35 }
36 }
37
38 export default user
```

3. 在 src\store 目录下创建 index.js

```
1 import Vue from 'vue'
2 import Vuex from 'vuex'
3 import user from './modules/user'
4
5 Vue.use(Vuex)
6
7 const store = new Vuex.Store({
8     modules: {
9         user
10    }
11})
12 export default store
```

4. 修改 src 下的 main.js，导入和注册 store

```
1 ... .省略
2
3 // 导入
4 import store from './store'
5
6 Vue.use(ElementUI);
7 import './permission'
8
9 new Vue({
10     router,
11     store, // 注册
12     render: h => h(App)
13 }).$mount("#app");
```



5. 重构登录组件 views\login\index.vue

```
1 . . .
2     methods: {
3         submitForm(formName) {
4             this.$refs[formName].validate(valid => {
5                 if (valid) {
6                     this.$store.dispatch('Login', this.form).then(response => {
7                         if(response.flag) {
8                             // 前往首页
9                             this.$router.push('/')
10                        }else {
11                            this.$message({
12                                message: response.message,
13                                type: 'warning'
14                            })
15                        }
16                    })
17                }else{
18                    console.log('验证失败')
19                    return false
20                }
21            })
22        }
23    . . .
```

6. 测试登录是否正常进入，并观察浏览器中localStorage是否有值

25.2.2 解决刷新页面回到登录页面

当前在 permission.js 路由拦截中是通过浏览器 localStorage 获取 token 值，但现在使用的Vuex状态管理，那我们希望通过 store 来获取 token 状态值.

1. 在 src\permission.js 中将获取 token 方式替换为从 store 状态中获取， 如下：

注意要 import 导入 store

```
1 . . .
2
3 import store from './store'
4
5 router.beforeEach((to, from , next) => {
6     // 1. 获取token
7     // const token = localStorage.getItem('mxg-msm-token')
8     const token = store.state.user.token
9
10    . . .
```

2. 重新访问登录 <http://localhost:8888/#/login>，登录后一样可以进入首页



3. 刷新首页，发现会回到登录页面。因为刷新后token状态值还原初始值 null，所以又会要求重新登录
4. 解决刷新页面重新登录的问题，在 src\store\modules\user.js 中 token 初始值为 getToken()

```
1 import {getToken, setToken, setUser, getUser, removeToken} from '@/utils/auth'
2 import { login, getUserInfo, logout} from '@/api/login'
3
4 const user = {
5   state: {
6     token: getToken(), // getToken() 作为token初始值，解决刷新页面之后token为null
7     user: null
8   },
9
10  ...
11  ...
```

25.3 获取用户信息

之前获取用户信息是在登录中一起获取的，其实可以在权限拦截器（ permission.js ）中进行获取就行。

1. 修改 src\store\modules\user.js，在 actions 中添加 GetUserInfo 方法

```
1 import {getToken, setToken, setUser, getUser, removeToken} from '@/utils/auth'
2 import { login, getUserInfo, logout} from '@/api/login'
3
4 const user = {
5   state: {
6     token: getToken(), // getToken() 作为token初始值，解决刷新页面之后token为null
7     user: getUser()
8   },
9
10  mutations: {
11    SET_TOKEN(state, token) {
12      state.token = token
13      setToken(token)
14    },
15    SET_USER(state, user) {
16      state.user = user
17      setUser(user)
18    }
19  },
20  actions: {
21    // 登录获取token
22    Login({commit}, form) {
23      // resolve 触发成功处理, reject 触发异常处理
24      return new Promise((resolve, reject) => {
25        login(form.username.trim(), form.password).then(response => {
26          const resp = response.data // 获取到的就是响应的data数据
27        })
28      })
29    }
30  }
31}
```



```
29         commit('SET_TOKEN', resp.data.token)
30         // 通过组件已经将token更新成功
31         resolve(resp)
32     }).catch(error => {
33         reject(error)
34     })
35
36 }
37
38 },
39
40 // 通过token获取用户信息
41 GetUserInfo({commit, state}) {
42
43     return new Promise((resolve, reject) => {
44         getUserInfo(state.token).then(response => {
45             const respUser = response.data
46             commit('SET_USER', respUser.data)
47             resolve(respUser)
48         }).catch(error => {
49             reject(error)
50         })
51     })
52
53 }
54 }
55
56
57 }
58
59 export default user
```

2. 修改 src\permission.js , 从 store 中获取用户数据

```
1 import router from './router'
2 import store from './store'
3
4 router.beforeEach((to, from, next) => {
5     // 1. 获取token
6     // const token = localStorage.getItem('mxg-msm-token')
7     const token = store.state.user.token
8     console.log('token', token)
9
10    if(token){
11        // 1.1 如果没有获取到 ,
12        // 要访问非登录页面，则不让访问，加到登录页面 /login
13        if(to.path !== '/login'){
14            next({path: '/login'})
15        }else {
16            // 请求登录页面 /login
17            next()
```

```
18      }
19  }else {
20      // 1.2 获取到token,
21      // 1.2.1 请求路由 /login , 那就去目标路由
22      if(to.path === '/login') {
23          next()
24      }else {
25          // 1.2.2 请求路由非登录页面 , 先在本地查看是否有用户信息 ,
26          // const userInfo = localStorage.getItem('mxg-msm-user')
27          const userInfo = store.state.user.userInfo
28          console.log('userInfo', userInfo)
29          if(userInfo) {
30              // 本地获取到 , 则直接让它去目标路由
31              next()
32          }else {
33              console.log('获取用户信息')
34              // 如果本地没有用户信息 , 就通过token去获取用户信息 ,
35              store.dispatch('GetUserInfo').then(response => {
36                  if(response.flag) {
37                      next()
38                  }else {
39                      next({path: '/login'})
40                  }
41              }).catch(error => {
42                  })
43          }
44      }
45  }
46 }
47 }
48 }
49 }
50 })
```

3. 测试 , 当登录后 , 查看localStorage 中有用户数据和 token

25.4 退出

1. 修改 src\store\modules\user.js , 在 actions 中添加 Logout 方法

```
1 actions: {
2     ..
3     ,
4     // 退出
5     Logout({ commit, state }) {
6         return new Promise((resolve, reject) => {
7             logout(state.token).then(response => {
8                 const resp = response.data
9                 commit('SET_TOKEN', '')
```

```
11         commit('SET_USER', {})
12         removeToken()
13         resolve(resp)
14     }).catch(error => {
15         reject(error)
16     })
17 }
18 }
19 }
```

2. 修改 src\components\AppHeader\index.vue 头部组件,

- 修改 data 选项中的 user 属性值

```
1 data() {
2     //...
3
4     return {
5         // user: JSON.parse(localStorage.getItem('mxg-msm-user')),
6         user: this.$store.state.user.user,
7     }
}
```

- 修改 methods 选项中的 handleLogout 方法

```
1 handleLogout() {
2     this.$store.dispatch('Logout').then(response => {
3         if(response.flag) {
4             // 退出成功
5             // 回到登录页面
6             this.$router.push('/login')
7         }else {
8             this.$message({
9                 message: resp.message,
10                type: 'warning',
11                duration: 500 // 弹出停留时间
12            });
13        }
14    })
15},
```

3. 测试 右上角显示名字正常, 退出正常

第二十六 阿里云服务器Nginx部署项目

注意：下面使用的阿里云服务器是 CentOS 7.5 版本教学

为什么选择Nginx

Nginx 是一个高性能的 Web 和反向代理服务器, 它具有有很多非常优越的特性:

作为 Web 服务器 : 相比 Apache , Nginx 使用更少的资源 , 支持更多的并发连接 , 体现更高的效率 , 这点使 Nginx 尤其受到虚拟主机提供商的欢迎。能够支持高达 50,000 个并发连接数的响应. 更好的处理静态文件

Nginx 安装非常的简单 , 配置文件 非常简洁 , Bugs非常少的服务器: Nginx 启动特别容易 , 并且几乎可以做到 7*24不间断运行 , 即使运行数个月也不需要重新启动。

26.1 打包项目

1. 在项目根目录下 , 执行打包命令

```
1 npm run build
```

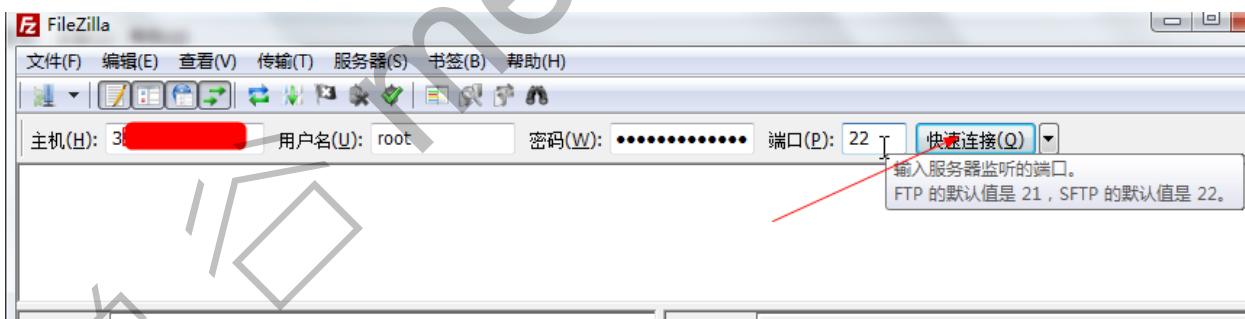
2. dist 重命名为 mxg-mms

26.2 上传安装包到服务器

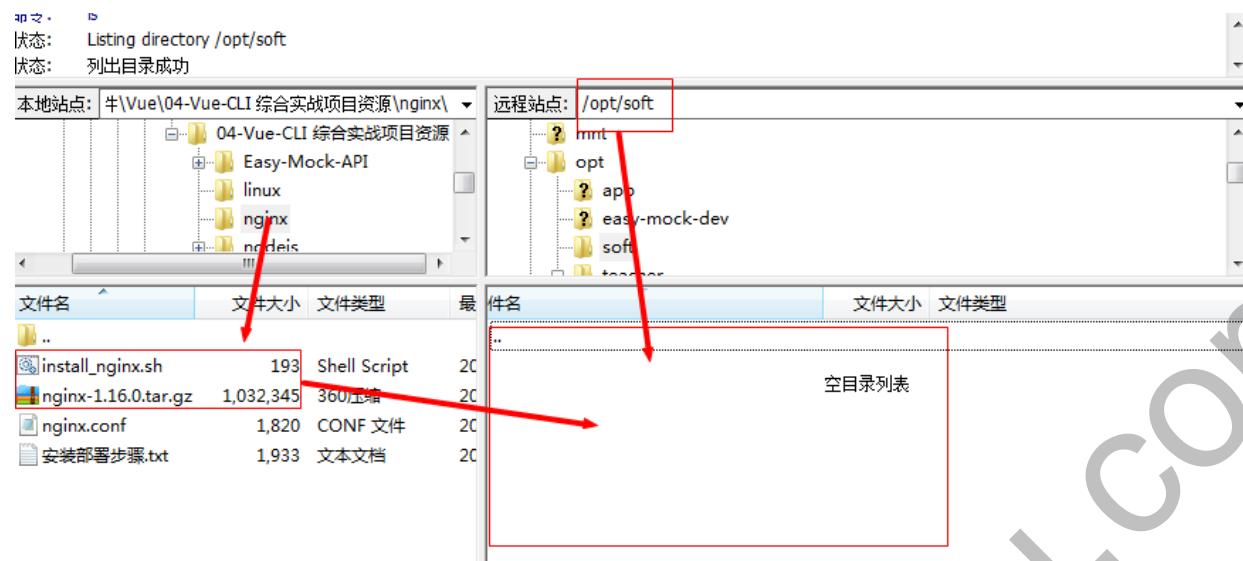
- 使用 filezilla 工具连接服务器

工具位于 : Vue-CLI 综合实战项目资源\linux客户端\FileZilla_3.7.3_win32.zip

解压 , 打开 filezilla.exe 即可。



- 创建 /opt/soft/
- 上传 install_nginx.sh 和 nginx-1.16.0.tar.gz 到 /opt/soft/



26.3 查看卸载操作

- 使用 SecureCRT 连接服务器

工具位于：Vue-CLI 综合实战项目资源\linux客户端\SecureCRT.zip
解压，打开 SecureCRT.exe 即可。

- 清屏快捷键 Ctrl + l(小写的L)
- 查看已经安装软件包

```
1 rpm -qa
```

- 卸载已安装的软件

```
1 rpm -e 软件名
```

26.4 安装 nginx 依赖包

安装时有提示输入，直接输入 y 按回车即可

- 安装 gcc，编译 nginx

```
1 yum install gcc-c++
```

- 安装 pcre，nginx 使用它解析正则表达式

```
1 yum install -y pcre pcre-devel
```

- 安装 zlib，提供了很多种压缩和解压方式，nginx 使用它对 http 包内容进行解压

```
1 yum install -y zlib zlib-devel
```

- 安装 openssl , nginx 支持http和https协议需要依赖它

```
1 yum install -y openssl openssl-devel
```

26.5 赋予脚本执行权限, 执行脚本

- 查看 /opt/soft/install_nginx.sh 脚本内容

注意：脚本内容中的换行符 Unix状态下的， windows转换Unix风格: 编辑 -> 档案格式转换 -> 转为XXX格式

参考: Vue-CLI 综合实战项目资源\nginx\install_nginx.sh

```
1 cat /opt/soft/install_nginx.sh
2
3 #!/bin/bash
4 cd /opt/soft
5 tar -zxf nginx-1.16.0.tar.gz
6 cd nginx-1.16.0/
7 ./configure \
8 --prefix=/usr/local/nginx \
9 --with-http_stub_status_module \
10 --with-http_ssl_module
11 make && make install
```

- 为所有者 (u) 增加脚本执行权限(+v)

```
1 chmod u+x install_nginx.sh
```

- 执行脚本

```
1 ./install_nginx.sh
```

26.6 启动与停止 nginx

- 进入安装目录

```
1 cd /usr/local/nginx/sbin/
```

- 启动 nginx

```
1 ./nginx
```

- 查看启动的 nginx 进程

```
1 ps -ef|grep nginx
```

- 结束进程

```
1 kill -9 进程号
```

- 检测 nginx 语法是否正确

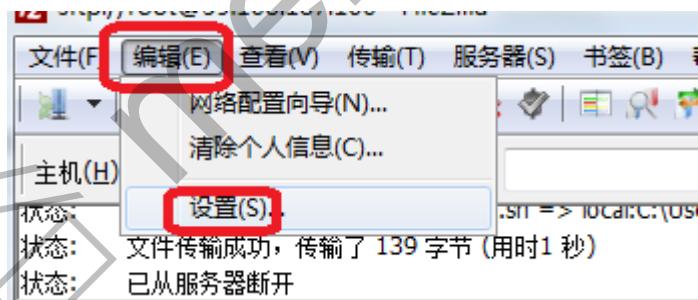
```
1 ./nginx -t
```

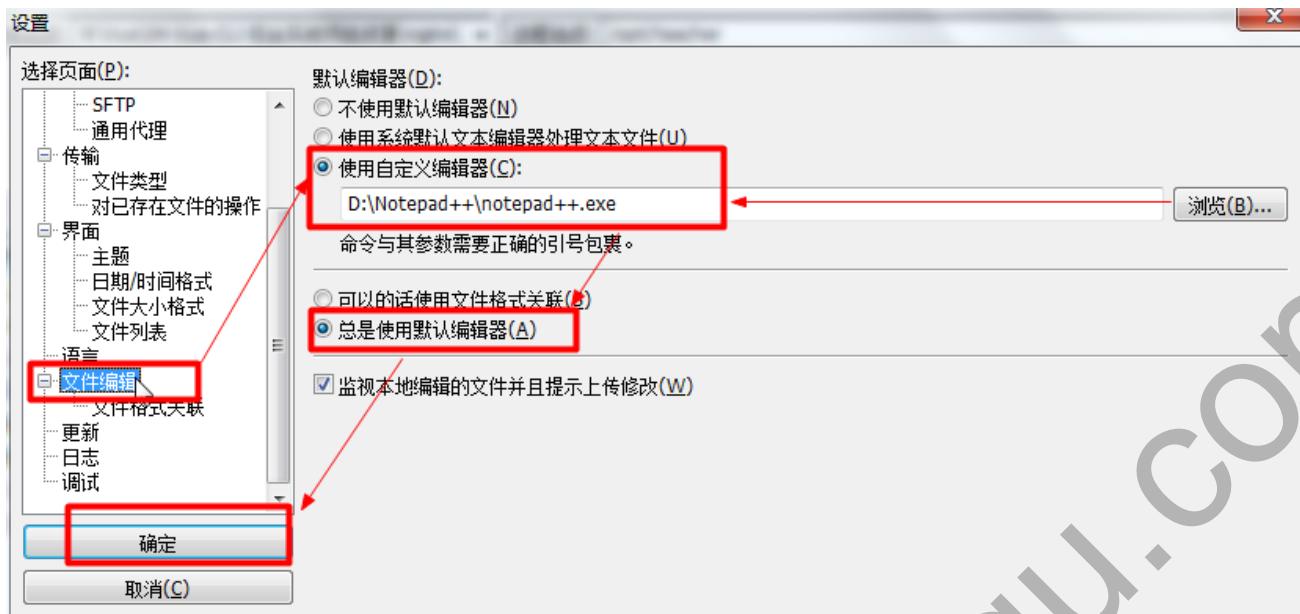
- 重新加载配置|重启|退出(正常关闭)|停止(强制关闭)

```
1 ./nginx -s reload|reopen|quit|stop
```

26.7 配置 nginx.conf 和上传项目

26.7.1 filezilla 设置默认文件编辑器





26.7.2 详解 nginx.config 配置

查看当前服务器所占用的端口号

```
1 netstat -ntlp
```

进入 nginx 安装目录 /usr/local/nginx/conf/，打开 nginx.config 配置文件

参考: Vue-CLI 综合实战项目资源\nginx\nginx.conf

```
1 # 启动的过程数量
2 worker_processes 1;
3
4 events {
5     # 单个进程的并发量
6     worker_connections 1024; # 总并发量 = 进程数量 * 单个进程的并发量
7 }
8
9 http {
10    include mime.types;
11    default_type application/octet-stream;
12
13    #access_log logs/access.log main;
14
15    sendfile on;
16    #tcp_nopush on;
17
18    #keepalive_timeout 0;
19    keepalive_timeout 65; # 连接服务器超时时长, 65秒
20
21    #gzip on;
```

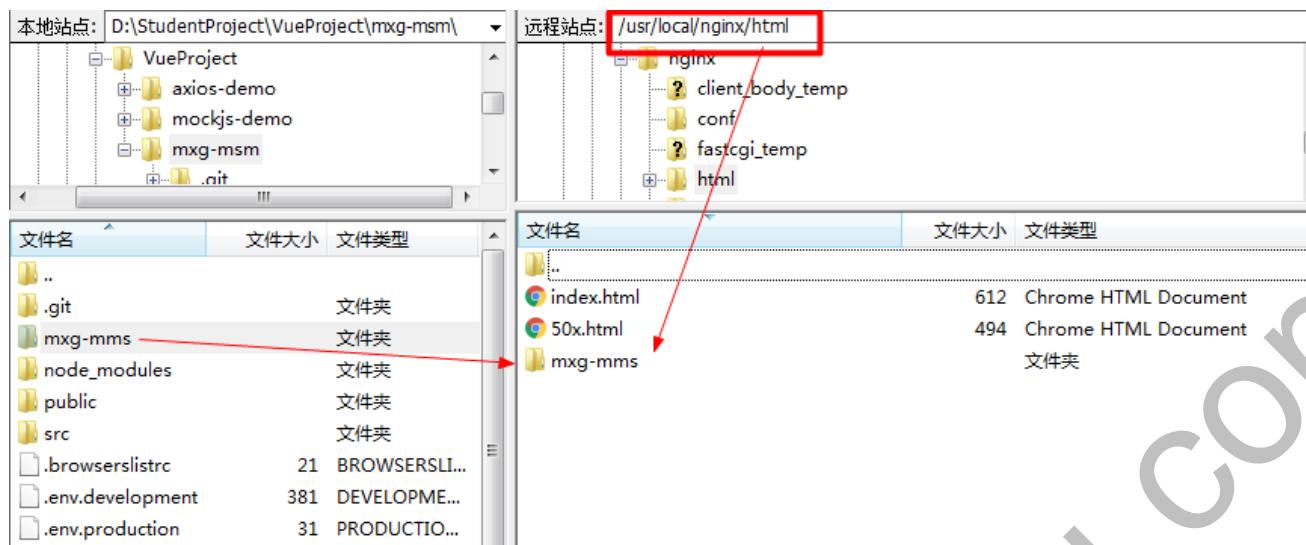
```
22
23 server { # 一个虚拟主机配置, 多个虚拟机配置多个 server 即可
24   listen 8888; # 端口号
25   server_name vue.mengxuegu.com; # 域名解析
26
27   location / { # 配置默认访问页
28     #root html; # 网站根目录
29     root /usr/local/nginx/html/mxg-mms;
30     index index.html index.htm; # 首页
31   }
32
33   location /pro-api {
34     # 代理转发后台服务接口
35     proxy_pass http://mengxuegu.com:7300/mock/5d477ccbfacc296cd6834fe5;
36   }
37
38   #error_page 404      /404.html;
39
40   # redirect server error pages to the static page /50x.html
41   # 配置错误页面
42   error_page 500 502 503 504 /50x.html;
43   location = /50x.html {
44     root html;
45   }
46
47 }
48
49 }
```

26.7.3 重新加载配置

```
1 ./nginx -s reload
```

26.7.4 打包项目上传

使用 filezilla 将 mxg-mms 打包目录上传到 /usr/local/nginx/html



26.8 开放端口号 8888 - 阿里云服务器

登录,进入控制台: <https://account.aliyun.com/>

谷歌浏览器限制 6666 端口 , 所以不要配置 6666 端口

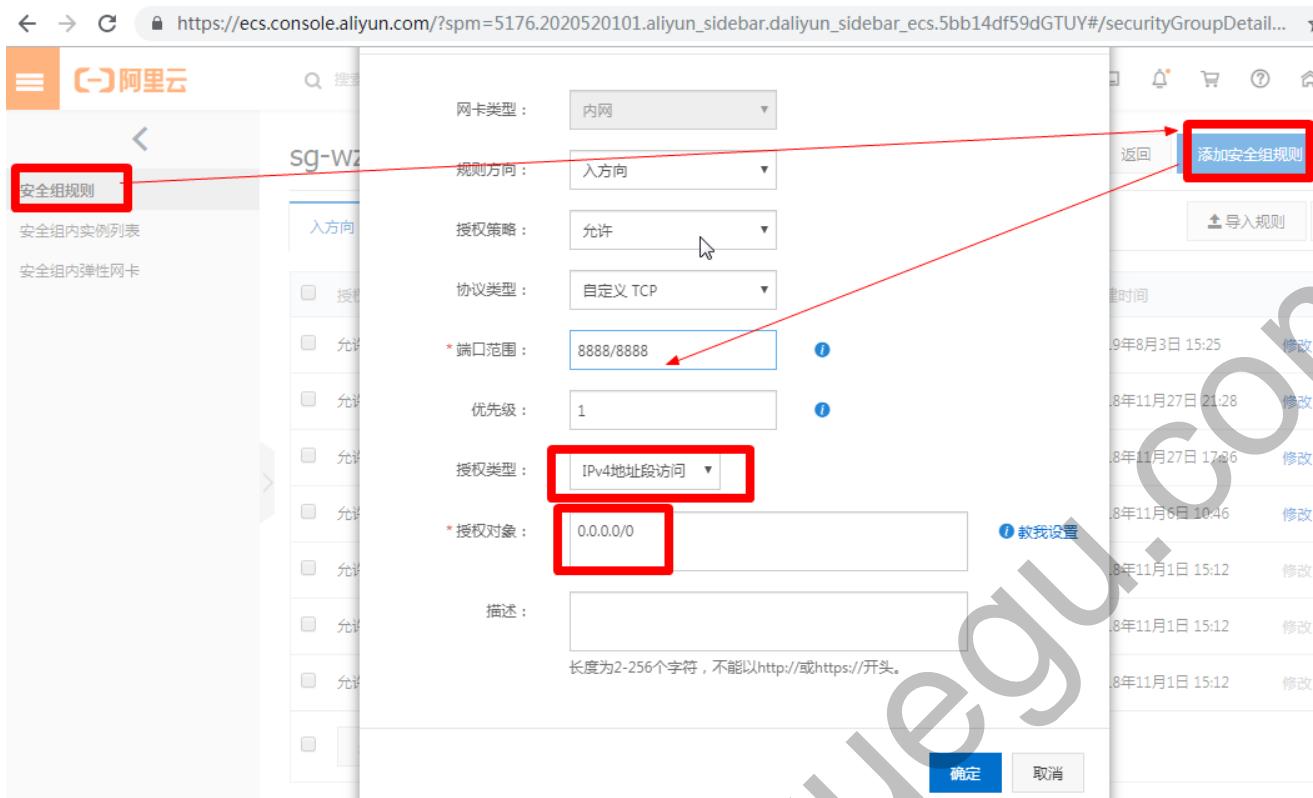


阿里云

产品与服务 > 云服务器 ECS

安全组列表

安全组ID/名称	标签	相关实例数	网络类型(全部)	安全组类型	创建时间	描述	操作
sg-wz9halq97qzw772274w0	vpc-wz9xuzjz7ctly2uxfd4pa	1	专有网络	普通安全组	15:11	System created security group...	修改 克隆 管理规则 配置规则 管理弹性网卡



26.9 二级域名绑定相同IP不同端口

二级域名 vue 与端口绑定



域名列表 进入域名解析列表>> 域名简介

您可能感兴趣的域名

mxg.cn szu.com 0bt.com jl.com jn.com qd.com ts.com kd.com qn.com jp.com tb.com
yb.com rz.com bz.com ob.com

全部域名 急需续费域名 急需赎回域名 未实名认证域名 预登记域名

域名： 域名类型： 全部 域名分组： 全部 注册日期：
到期日期：
域名 域名类型 (1) 域名状态 域名分组 注册日期 到期日期 操作
 mengxuegu.com gTLD 正常 未分组 2018-11-06 09:40:56 2021-11-06 09:40:56 续费 | 解析 | 安全锁 (1) | 备注 | 管理

映射 `vue.mengxuegu.com` 与 8080 端口, 注意记录类型 隐性 U R L



<http://vue.mengxuegu.com/> 即可访问到 mengxuegu.com:8888 的那个项目