

# Vue3+Koa2 开发全栈 ERP 后台管理系统

## Vue3+Koa2 开发全栈 ERP 后台管理系统

项目背景

系统功能

项目亮点

前端目录结构

  axios 二次封装

  storage 二次封装

后端目录结构

  基于log4js 进行日志封装

    Level

    category (类型)

    Appender

  MongoDB

  登录模块- 用户登录前后台实现

**ORM**

    welcome 组件开发

    Tree-menu 组件开发

    Breadcrumb 面包屑

    扩展 useRoute 与 useRouter 的区别

      基于Token 的身份验证

      token 拦截

  用户管理模块开发

  删除与批量删除功能

  用户新增功能实现

  菜单管理模块-RBAC模型

    一个用户一个或多个角色

    扩展：

      菜单管理-创建与新增功能实现

      菜单管理-编辑与删除功能实现

      菜单管理-菜单列表递归实现

    创建一个二级菜单试试

  角色管理-角色列表开发

    实现UI布局

    角色操作 (创建、编辑、删除)

    角色管理-权限控制

  部门管理-创建、编辑、删除服务端功能实现

    权限中心-按钮权限控制

  审批管理 - 休假申请

    接口字段

    定义表格列头

时间格式化  
休假时长(动态计算)  
休假申请-服务端开发  
待我审批-前后端开发

## 项目背景

现如今前端的边界在持续扩大，企业对于工具化、平台化、全栈化能力要求也越来越高。本系统应用Vite2.0 + Vue3 + ElementPlus+Koa2+Mongo开发一个通用后台管理系统。

## 系统功能

- 用户管理
- 菜单管理
- 角色管理
- 部门管理
- 审批流申请
- 审批流审核

## 项目亮点

- vite 构建 / koa 架构设计
- vue3全家桶 / 日志规范封装
- API统一管理 / 数据库操作技巧
- JWT认证 / 权限控制
- 脚手架开发 / 低代码封装
- 接口文档定义 / models 封装

## 前端目录结构

### manager-fe

```
--dist # 项目打包后自动生成的目录 无需自行创建  
--node-modules  
--public # 公共资源文件
```

```
--src
----api # 项目接口管理文件
----assets # 资源静态文件
----components # 组件
----config # 项目配置 ... mock API
----router # 路由
----store # 状态管理
----utils # 工具函数
----views # 页面结构
----App.vue # 项目工程必备
----main.js # 项目工程必备
--gitignore # 在git提交的时候忽略一些目录跟文件。
--.env.dev # 环境变量
--env.test # 环境变量
--env.prod # 环境变量
--index.html # 入口文件
--package.json # 项目配置文件
--vite.config.js # vite 相关的配置文件
```

## axios 二次封装

为什么我们要二次封装？ 因为我们可以统一进行接口的请求拦截， 响应拦截。 动态配置每个接口的baseURL 以及错误管理。

utils 文件夹新建 request.js

```
const instance = axios.create({ baseURL: config.baseApi, timeout: 8000 });
```

base URL 是接口的地址， 根据环境变量来判断是开发环境还是线上环境， 会根据不同的环境再来更改baseURL。

# storage 二次封装

## localStorage

- 持久化本地存储，除非主动删除，否则不会过期
- 存储大小为5M或更大
- 在所有同源窗口中都是共享的
- 适用于长期登陆，判断用户是否已登陆；长期保存数据，可与vuex同步数据

## sessionStorage

- 同一个会话的页面才能访问，会话结束时会被清除，仅在当前浏览器窗口关闭前有效
- 存储大小为5M或更大
- 不在不同的浏览器窗口中共享
- 适合敏感账号一次性登陆

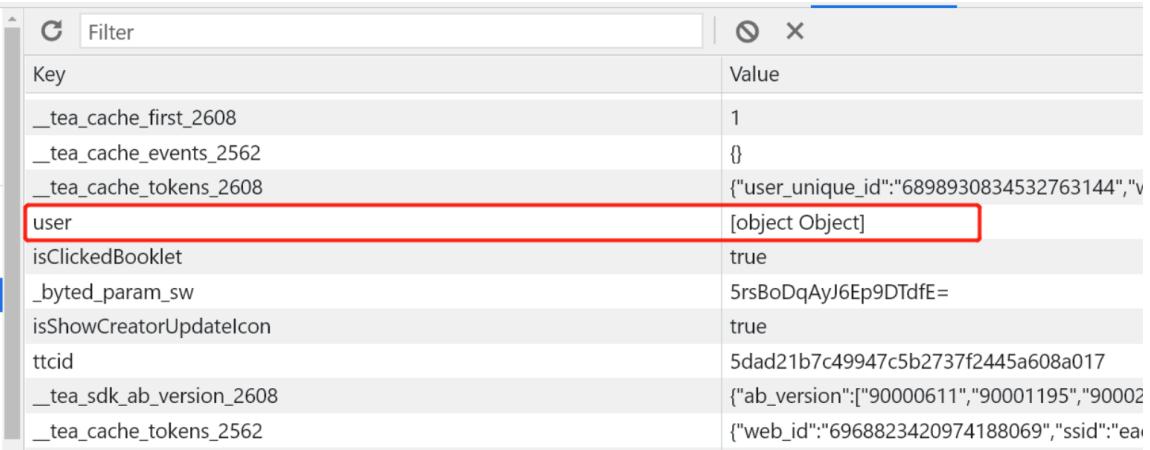
既然存取这么方便为什么我们还要来进行二次封装？

主要是几个问题：

1.localStorage 无法存储引用类型的数据

```
localStorage.setItem("user", {"name": "felix"});
```

当我们的项目复杂，或者多个系统模块都需要往localStorage存数据，而localStorage中存的方式是键值对的形式，就很容易导致命名冲突的问题。



The screenshot shows the Chrome DevTools Storage panel for the URL <https://juejin.cn>. The left sidebar lists various storage types: Session Storage, Local Storage, IndexedDB, Web SQL, and Cookies. The Local Storage section is selected and highlighted in blue. The main area displays the contents of localStorage as a table:

Key	Value
_tea_cache_first_2608	1
_tea_cache_events_2562	{}
_tea_cache_tokens_2608	{"user_unique_id": "6898930834532763144", "v
user	[object Object]
isClickedBooklet	true
_byted_param_sw	5rsBoDqAyJ6Ep9DTdffE=
isShowCreatorUpdateIcon	true
ttcid	5dad21b7c49947c5b2737f2445a608a017
_tea_sdk_ab_version_2608	{"ab_version": ["90000611", "90001195", "90002
_tea_cache_tokens_2562	{"web_id": "6968823420974188069", "ssid": "ea

## 对象字面量模式

对象字面量模式可以认为是包含一组键值对的对象，每一对键和值由冒号分隔，键也可以是代码新的命名空间。

又回到关键问题上了怎么让localStorage 支持存储引用类型的数据

那怎么给这个对象扩展呢？

```
localStorage.setItem("manager", '{"name": "Felix"}');
```

1. 取，序列化，通过JSON.parse 转化成一个对象，在给这个对象进行扩展。JSON.stringify() 把对象转化成字符串（反序列化）

```
localStorage.setItem("manager", '{"name": "老李"}');
JSON.parse(localStorage.getItem("manager"));
var obj = JSON.parse(localStorage.getItem("manager"));
obj.age = 30
localStorage.setItem("manager", JSON.stringify(obj));
```

## 后端目录结构

<https://koa.bootcss.com/#>

全局安装脚手架工具

```
npm install -g koa-generator
```

### App.js

是项目的根文件，注意它并不是入口文件。

**bin -> www**

项目启动服务的入口文件。

**models** 存放mongodb的模型

**public** 静态资源根目录

**utils** 工具函数封装

**Logs** 存放日志

**routes** 目录

index.js 一级路由

users.js 二级路由 router.prefix() 将路径中公共的部分先抽离出来，你也可以理解为路由前缀部分。

## 基于log4js 进行日志封装

log4js 是 Node.js 日志处理中的数一数二的模块。

- 日志分级
- 日志分类

- 日志落盘

log4js 最简单的用法：

```
var log4js = require("log4js");
var logger = log4js.getLogger();
logger.level = "debug";
logger.debug("Some debug messages");
```

调用 .getLogger() 可以获得 log4js 的 Logger 实例，这个实例的用法与 console 是一致的，可以调用.debug（也有.info、.error 等方法）来输出日志。

## Level

log4js 的日志分为九个等级，各个级别的名字和权重如下：

```
{
  ALL: new Level(Number.MIN_VALUE, "ALL"), // output all log
  TRACE: new Level(5000, "TRACE"),
  DEBUG: new Level(10000, "DEBUG"),
  INFO: new Level(20000, "INFO"),
  WARN: new Level(30000, "WARN"),
  ERROR: new Level(40000, "ERROR"),
  FATAL: new Level(50000, "FATAL"), // high or equal level log
  MARK: new Level(9007199254740992, "MARK"),
  // 2^53 OFF: new Level(Number.MAX_VALUE, "OFF") // not output
}
```

## category (类型)

设置一个 Logger 实例的类型，按照另外一个维度来区分日志。

```
var log4js = require("log4js");
var logger = log4js.getLogger("global-test.js");
logger.level = "debug";
logger.debug("Some debug messages");
```

在通过 getLogger 获取 Logger 实例时，唯一可以传的一个参数就是 "global-test.js"，通过这个参数来指定 Logger 实例属于哪个类别。

# Appender

日志有了级别和类别，解决了日志在入口处定级和分类问题，而在 log4js 中，日志的出口问题（即日志输出到哪里）就由 Appender 来解决。

默认的appender

```
// log4js.js
defaultConfig = {
  appenders: [
    { type: "console" }
  ]
}
```

设置自己的 appender

通过log4js.configure来设置我们想要的 appender。

```
var log4js = require("log4js");
log4js.configure({
  //必选
  appenders: { out: { type: "file", filename: "test.log" } },
  //必选 getLogger 参数为空时，默认使用该分类
  categories: { default: { appenders: ["out"], level: "error" } },
});
var logger = log4js.getLogger("global-test.js");
logger.level = "debug";
logger.debug("Some debug messages");
```

# MongoDB

什么是MongoDB？

- MongoDB 诞生在2007年是由C++语言编写的，是一个基于**分布式文件存储**的开源数据库系统。
- MongoDB 特点是将数据存储为一个文档，数据结构由键值(key=>value)对组成。
- MongoDB 文档类似于 JSON 对象。
- 字段值可以包含其他文档，数组及文档数组。

**MongoDB优点？**

- MongoDB 是一个面向文档存储的数据库，操作起来比较简单和容易。
- Mongo支持丰富的查询表达式。查询指令使用**JSON**形式的标记，可轻易查询文档中内嵌的对象及数组。
- 你可以通过本地或者网络创建数据镜像，这使得MongoDB有更强的扩展性。
- MongoDB允许在服务端执行脚本，可以用Javascript编写某个函数，直接在服务端执行，也可以把函数的定义存

储在服务端，下次直接调用即可。

## 排序

在MongoDB中使用 `sort()` 方法对数据进行排序，`sort()` 方法通过参数来指定排序的字段，并使用1和-1来指定排序方式，1为升序，-1为降序；

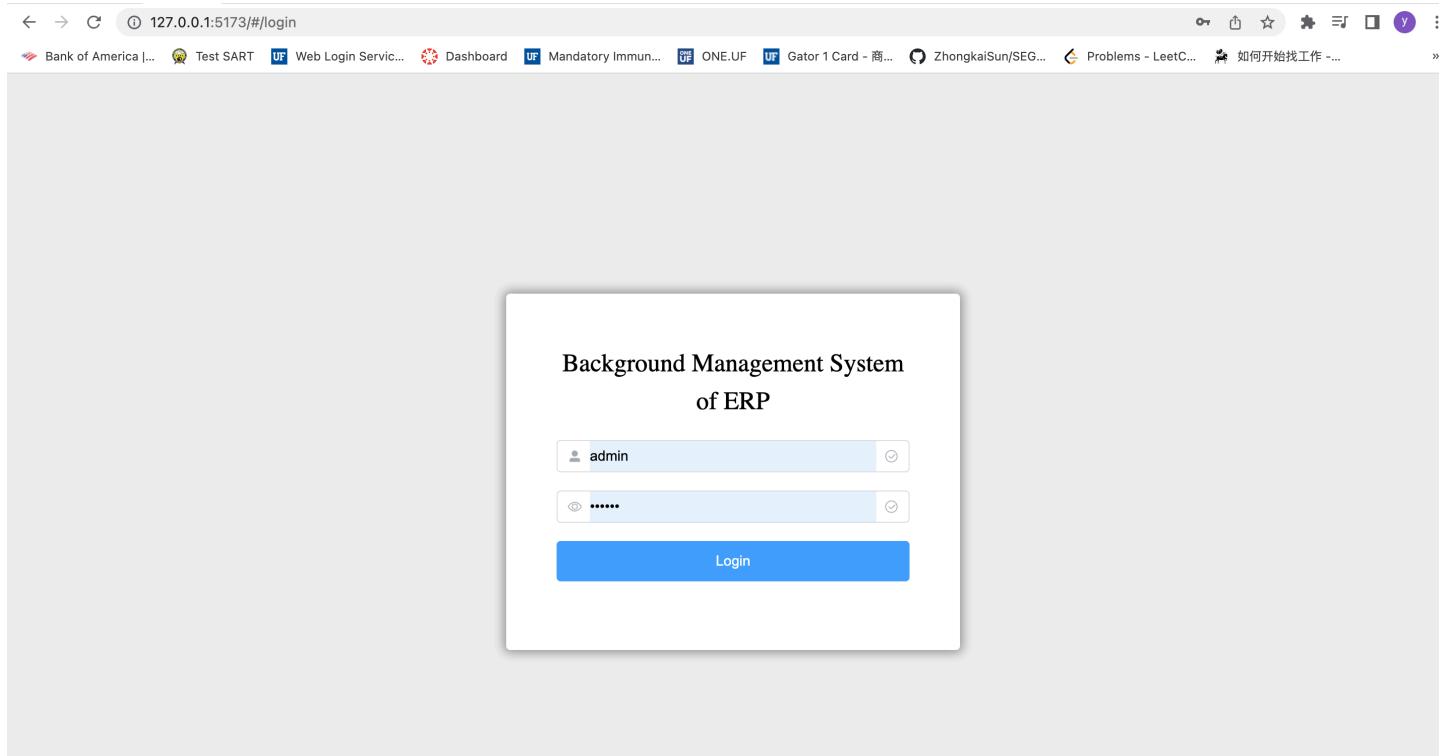
```
db.collection.find().sort({KEY:1})
```

## 索引

- 索引通常能够极大的提高查询的效率，如果没有索引，MongoDB在读取数据时必须扫描集合中的每个文件并选取那些符合查询条件的记录。
- MongoDB使用 `createIndex()` 方法来创建索引，语法如下；

```
db.collection.createIndex(keys, options)
# background: 建索引过程会阻塞其它数据库操作，设置为true表示后台创建，默认为false
# unique: 设置为true表示创建唯一索引
# name: 指定索引名称，如果没有指定会自动生成。
```

# 登录模块- 用户登录前后台实现



页面结构独立并不复用上一页的结构，我们并没有注册按钮。原因是当新页上加入的时候账号密码没有系统模块分配的，并不支持注册。

## 连接数据库

当用户点击登录(携带账号密码)请求过来的时候在服务端需要去数据库中查询是否有当前用户，密码是否正确，最后把查询的结果返回给客户端。

在manager-server 中新建一个config-> index.js (配置文件)

```
module.exports = {  
    URL: "mongodb://127.0.0.1:27017/manager"  
}
```

- mongodb 这是一个固定的前缀。
- 127.0.0.1:27017 这个是mongodb 服务部署在本地的端口。
- manager 这个跟你在 mongodb 中创建的数据库名称一一对应。
- 那这个固定的格式怎么来的呢？这是 mongoose 的一个语法。

## mongoose

mongoose是 nodejs 提供连接 mongodb 的一个库，它对mongodb一些原生方法进行了封装以及优化。

Mongoose就是对node环境中MongoDB数据库操作的封装，一个对象模型工具，它将数据库中的数据转换为JavaScript对象以供我们在应用中使用

## Schema

- 数据库连接成功我们再来讲个Mongoose 的Schema功能。
- Mongoose的一切都始于一个Schema。
- 每个schema映射到MongoDB的集合(collection)和定义该集合(collection)中的文档的形式。
- 简单说 Schemas 定义了文档和属性的结构。

models -> userSchema.js

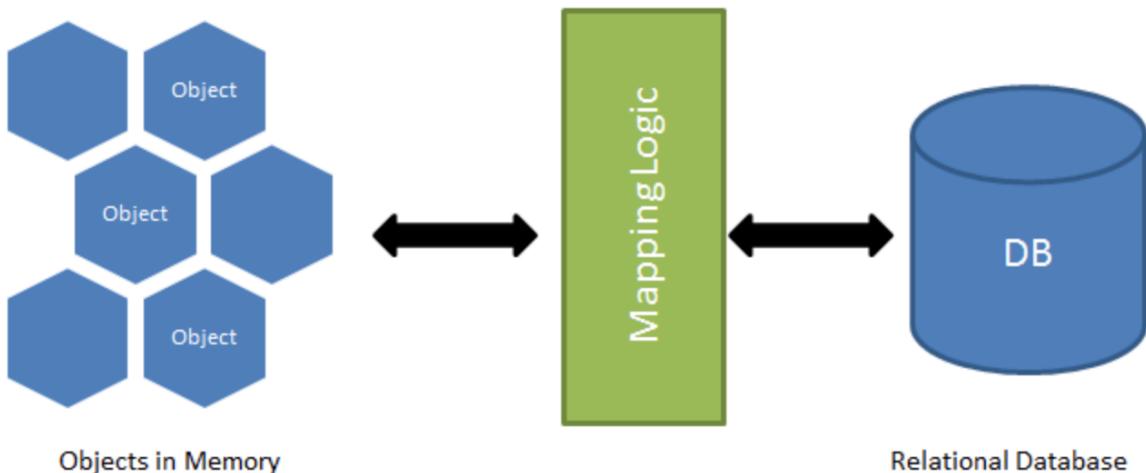
首先Mongoose不是MongoDB的驱动，而是ODM (类似于关系型数据库的 ORM , ORM ODM 的设计思想是一样的，不过一种是针对关系型数据库 一种是针对文档型数据库)

## ORM

面向对象编程把所有实体看成对象 (object) ， 关系型数据库则是采用实体之间的关系 (relation) 连接数据。

很早就有人提出，关系也可以用对象表达，这样的话，就能使用面向对象编程，来操作关系型数据库。

## O/R Mapping



ORM优点:

- 数据模型在一个地方定义，易于更新与维护。
- 有现成的工具，很多功能可以自动完成，如数据预处理、事务等。
- 迫使你使用MVC架构，ORM是天然的Model。
- 基于ORM的业务代码简单，代码量少，语义性好，易理解。
- 不必编写性能不佳的SQL语句。

ORM缺点:

- ORM库通常都不是轻量级工具，是有一定学习曲线的。
- 对于复杂查询，ORM要么无法表达，要么性能不如原生SQL。
- 无法定制一些特殊SQL。

## welcome 组件开发

侧边栏组件 菜单组件 用户模块

## Tree-menu 组件开发

在一个公司组织架构上是一个三角形的，站在顶层的人能支配的资源/权限更多，然后就是逐层递减，所以在管理系统中也是一样，不同角色的员工能看到的菜单是不一样的，能点击的按钮也是不一样的。

目前的做法是一刀切，把菜单都写死，贴在页面上。所以正确的做法是当用户登录进入到首页我们就要发起请求根据用户的角色展示不同的菜单/按钮。

那这也意味着菜单需要动态遍历。

递归拼接树形列表，先把第一层菜单便利出来，再把第二层菜单便利出来，依此类推。

这样我们就可以服务端设计好返回对象的结构

## Breadcrumb 面包屑

顶部导航栏显示当前页面的路径，并且能快速返回之前的任意页面。

那这个功能在Element UI 中有一个专门的组件 Breadcrumb

## 扩展 useRoute 与 useRouter 的区别

### useRoute()

route是一个跳转的路由对象，每一个路由都会有一个route对象，是一个局部的对象，可以获取对应的name,path,params,query等：

● matched: 数组 只今当前匹配的数组由所有今的所有此段所对应的配置参数对象

- name: 当前路径的名字，如果没有使用具名路径，则名字为空。
  - params: 对象，包含路由中的动态片段和全匹配片段的键值对
  - path: 字符串，等于当前路由对象的路径，会被解析为绝对路径，如 “/home/news”
  - query: 对象，包含路由中查询参数的键值对。
    - 例如，对于 /home/news/detail/01?favorite=yes，会得到\$route.query.favorite => 'yes'

## useRouter()

router是VueRouter的一个对象，通过Vue.use(VueRouter) 和 VueRouter 构造函数得到一个 router 的实例对象，这个对象中是一个全局的对象，他包含了所有的路由包含了许多关键的对象和属性。

举例：history对象

`router.push({path:'home'})`;本质是向history栈中添加一个路由，在我们看来是切换路由，但本质是在添加一个history记录。

## 基于Token 的身份验证

jwt

## jsonwebtoken

<https://www.npmjs.com/package/jsonwebtoken>

引用 jsonwebtoken，设置你要加密的数据，传入秘钥。

```
var jwt = require('jsonwebtoken');
var token = jwt.sign({ foo: 'bar' }, 'shhhhh');
```

```
jwt.sign({  
  exp: Math.floor(Date.now() / 1000) + (60 * 60),  
  data: 'foobar'  
, 'secret');
```

## token 拦截

verify方法添加回调，回调里面有两个参数，error 表示错误，decoded 是解码之后的 Token 数据。

但这里的问题是在我们后台接口中绝大多数都需要做token认证，单个认证是很简单，但是我们不能给每个接口都去调用一次verify方法吧？

所以我们可以这样做：这里使用一个 jsonwebtoken 的上游工具 koa-jwt。koa-jwt 是基于jsonwebtoken 封装的一个更上游的工具，使用它做 token 拦截非常的简单。

## 用户管理模块开发

	User ID	Name	Email	Role	State	Create Time	Last Login	Operations
<input type="checkbox"/>	100001	admin	admin@mashibing.com	admin	On the job	11/17/2022, 8:32:06 AM	11/17/2022, 8:32:06 AM	<a href="#">Edit</a> <a href="#">Delete</a>
<input type="checkbox"/>	100002	James	james@mashibing.com	user	On the job	11/17/2022, 8:32:06 AM	11/17/2022, 8:32:06 AM	<a href="#">Edit</a> <a href="#">Delete</a>
<input type="checkbox"/>	100005	felix	111@mashibing.com	admin	On the job	3/16/2023, 11:24:58 AM	3/16/2023, 11:24:58 AM	<a href="#">Edit</a> <a href="#">Delete</a>
<input type="checkbox"/>	100009	ffff	eeerrrr@mashibing.com	user	On the job	3/16/2023, 11:24:58 AM	3/16/2023, 11:24:58 AM	<a href="#">Edit</a> <a href="#">Delete</a>
<input type="checkbox"/>	100010	felix3	zhangyinghuan@ufl.edu@mashibing.com	admin	On the job	3/16/2023, 11:24:58 AM	3/16/2023, 11:24:58 AM	<a href="#">Edit</a> <a href="#">Delete</a>
<input type="checkbox"/>	100026	lee	12344@mashibing.com@mashibing.com	user	On the job	4/1/2023, 8:42:17 PM	4/1/2023, 8:42:17 PM	<a href="#">Edit</a> <a href="#">Delete</a>
<input type="checkbox"/>	100030	tes	zhangying@mashibing.com	user	On the job	4/1/2023, 9:12:17 PM	4/1/2023, 9:12:17 PM	<a href="#">Edit</a> <a href="#">Delete</a>

## 删除与批量删除功能

先来看下删除的接口，如果做单个删除的操作只需要传入userId, 如果删除多个就把userId 组装在数组中实现批量删除。

先定义接口

```
export function userDelete(params) {
  return request.post("/users/delete", params, true);
}
```

绑定事件

```
<el-button type="danger" size="small" @click="handleDelete">删除</el-button>

async function handleDelete(){
}
```

```
import { ElMessage } from 'element-plus'

async function handleDelete(row){
  await userDelete({
    userIds:[row.userId]
  });

  ElMessage.success("删除成功");
  getUserList();
}


```

## 用户新增功能实现

静态布局 填充表单

Add a New User

×

\* User Name

\* Email  @mail.com

\* Mobile

Role

State

\* RoleList

Department

级联选择器



\$in

语法:

```
{ field: { $in: [<value1>, <value2>, ... <valueN> ] } }
```

如果 field持有一个数组，那么\$in 运算符选择包含数组的文档，field该数组包含至少一个与指定数组中的值匹配的元素（例如，、等）

语法:

```
{ $or: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }
```

\$or 运算符对一个或多个 数组执行逻辑运算，并选择满足至少一个。

## 菜单管理模块-RBAC模型

RBAC权限模型 (Role-Based Access Control) 即：基于角色的权限控制。

- 用户：系统接口及访问的操作者
- 权限：能够访问某接口或者做某操作的授权资格
- 角色：具有一类相同操作权限的用户的总称

RBAC权限模型核心授权逻辑如下

- 某用户是什么角色？
- 某角色具有什么权限？
- 通过角色的权限推导用户的权限

**一个用户一个或多个角色**

但是在实际的应用系统中，一个用户一个角色远远满足不了需求。如果我们希望一个用户既担任销售角色、又暂时担任副总角色。该怎么做呢？为了增加系统设计的适用性，我们通常设计：

- 一个用户有一个或多个角色
- 一个角色包含多个用户
- 一个角色有多种权限
- 一个权限属于多个角色

## 扩展：

**页面访问权限：**所有系统都是由一个个的页面组成，页面再组成模块，用户是否能看到这个页面的菜单、是否能进入这个页面就称为页面访问权限。

**操作权限：**用户在操作系统中的任何动作、交互都需要有操作权限，如增删改查等。比如：某个按钮，某个超链接用户是否可以点击，是否应该看见的权限。

**数据权限：**数据权限比较好理解，就是某个用户能够访问和操作哪些数据。

## 树形数据与懒加载

支持树类型的数据的显示。当 row 中包含 children 字段时，被视为树形数据。渲染嵌套数据需要 prop 的 row-key。此外，子行数据可以异步加载。设置 Table 的 lazy 属性为 true 与加载函数 load。通过指定 row 中的 hasChildren 字段来指定哪些行是包含子节点。children 与 hasChildren 都可以通过 tree-props 配置。

## 菜单管理-创建与新增功能实现

The screenshot shows a user interface for managing menus. On the left is a dark sidebar with a user profile picture and the name "manager". Below the profile are several menu items: "系统管理", "用户管理", "菜单管理", "角色管理", "部门管理", "审批管理", "待我审批", and "休假申请".

The main content area has a header with search and reset buttons. Below the header is a table titled "Create" for adding new menu items. The table columns are: Menu Name, Icon, Menu Type, Menu Code, Router Path, Menu State, and Operations.

Menu Name	Icon	Menu Type	Menu Code	Router Path	Menu State	Operations
系统管理	Grid	Menu	/system	Normal	Add Edit Delete	
用户管理		Menu	/system/user	Normal	Add Edit Delete	
新增	Button	user-add		Normal	Add Edit Delete	
删除	Button	user-delete		Normal	Add Edit Delete	
编辑	Button	user-edit		Normal	Add Edit Delete	
批量删除	Button	user-all-delete		Normal	Add Edit Delete	
菜单管理		Menu	/system/menu	Normal	Add Edit Delete	
角色管理		Menu	/system/role	Normal	Add Edit Delete	
部门管理		Menu	/system/dept	Normal	Add Edit Delete	
审批管理	Promotion	Menu	/audit	Normal	Add Edit Delete	
待我审批		Menu		Normal	Add Edit Delete	
休假申请		Menu	/approval/application	Normal	Add Edit Delete	

## 菜单管理-编辑与删除功能实现

## Edite Menu's/Button's Information

X

Parent Menu

系统管理



If no selection, create a level 1 menu directly

Menu Type

Menu

Button

\* Menu Name

用户管理

Menu Icon

Please Input Menu Icon Path

Router Path

/system/user

Menu State

Normal

Disable

Cancel

Conform

## 菜单管理-菜单列表递归实现

创建一级菜单

## New Menu/Button

X

Parent Menu

Selection



If no selection, create a level 1 menu directly

Menu Type

Menu

Button

\* Menu Name

Please Input Menu Name

Menu Icon

Please Input Menu Icon Path

Router Path

Please Input Router Path

Menu State

Normal

Disable

Cancel

Conform

## 创建一个二级菜单试试

发现一个问题就是用户管理是系统管理下的二级菜单，但是在表格中没有显示关联关系。

系统管理	Grid	菜单	/system	正常	<button>新增</button>	<button>编辑</button>	<button>删除</button>
审批管理	Promotion	菜单	/audit	正常	<button>新增</button>	<button>编辑</button>	<button>删除</button>
用户管理		菜单	/system/user	正常	<button>新增</button>	<button>编辑</button>	<button>删除</button>

同时我们还发现了，在数据库中用户管理的parentId 字段并没有存储上级菜单的"\_id"

12 | "updateTime": ISODate("2023-02-24T08:04:39.448+0000"),  
12 | "v": NumberInt(0)

```

14 }
15 {
16     "_id" : ObjectId("63f874edee40d9588a4684d2"),
17     "menuType" : NumberInt(1),
18     "menuState" : NumberInt(1),
19     "menuName" : "审批管理",
20     "path" : "/audit",
21     "icon" : "Promotion",
22     "parentId" : [
23
24     ],
25     "createTime" : ISODate("2023-02-24T08:10:02.651+0000"),
26     "updateTime" : ISODate("2023-02-24T08:10:02.651+0000"),
27     "__v" : NumberInt(0)
28 }
29 {
30     "_id" : ObjectId("63f877dce40d9588a4684df"),
31     "menuType" : NumberInt(1),
32     "menuState" : NumberInt(1),
33     "menuName" : "用户管理",
34     "path" : "/system/user",
35     "parentId" : [
36
37     ],
38     "createTime" : ISODate("2023-02-24T08:10:02.651+0000"),
39     "updateTime" : ISODate("2023-02-24T08:10:02.651+0000"),
40     "__v" : NumberInt(0)
41 }
42

```

## 删除接口问题

```

async function handleDelete(row){
    await menuOperate({
        _id:[row._id],
        action:"delete"
    });

    ElMessage.success("删除成功");
    getMenuList();
}

```

需要传递正确的id 给到服务端。

这个时候我们已经能够在数据库中通过parentId 设计正确的菜单父子（上下级）关系。

但是这还不够，我们需要把这种关系传递到前端去，Table表格支持树型的数据的显示，是根据 children 来指定哪些行是包含子节点。

换句话说最终我们的目的，是要在服务端设计好返回对象的结构：

```
{ .... 当前菜单信息 (一级菜单) children:[ ... 当前子菜单的信息 (二级菜单) ] }
```

定义一个treeMenu 方法来完成这个事情

menuList 是所有菜单的列表，它在数据库中是平级关系，我们首先要做就是把第一级菜单遍历出来，在把第二级菜单遍历出来，以此类推。

```
router.get("/list", async (ctx)=>{
  const {menuName, menuState} = ctx.request.query;
  const params = {};
  if(menuName)params.menuName = menuName;
  if(menuState)params.menuState = menuState;
  let menuList = await Menu.find(params);
  //menuList 是所有菜单的列表 null 是否为一级菜单的凭证 [] 最终返回给客户端的数据对象
  treeMenu(menuList, null, []);
  ctx.body = util.success(menuList);
});

//递归拼接树形列表
function treeMenu(menuList, _id, list){
  for(let i = 0; i<menuList.length; i++){
    }
}
```

参数定义：

menuList 是所有菜单的列表

null 是否为一级菜单的凭证

[] 最终返回给客户端的数据对象

```
function treeMenu(menuList, id, list){}
```

```

function treeMenu(menuList, id, list){
  for(let i = 0; i<menuList.length; i++){
    const item = menuList[i];
    if(item.parentId.slice().pop() == null){
      list.push(item._doc);
    }
  }
}

```

## 为什么我们要用slice ?

因为如果直接用pop 提取item.parentId 数组中的值会改变原有数组的结构，所以需要通过slice方法克隆一个数组对象处理进行操作。

接着递归去处理二级或多级菜单

```

router.get("/list", async (ctx)=>{
  const {menuName, menuState} = ctx.request.query;
  const params = {};
  if(menuName)params.menuName = menuName;
  if(menuState)params.menuState = menuState;
  let menuList = await Menu.find(params);
  //menuList 是所有菜单的列表 null 是否为一级菜单的凭证 [] 最终返回给客户端的数据对象
  const result = treeMenu(menuList, null, []);
  ctx.body = util.success(result);
});

//递归拼接树形列表
function treeMenu(menuList, id, list){
  for(let i = 0; i<menuList.length; i++){
    const item = menuList[i];
    if(String(item.parentId.slice().pop()) == String(id)){
      list.push(item._doc);
    }
  }

  list.map(item => {
    item.children = [];
    treeMenu(menuList, item._id, item.children);
  })
  return list;
}

```

```
<el-menu-item v-else-if="menu.menuType == 1" :index="menu.path">{{menu.menuName}}</el-
```

menu-item>

## 问题解决

# 角色管理-角色列表开发

## 实现UI布局

Role	Remark	Permission List	Create Time	Operations
产品经理		休假申请,用户管理,菜单管理,角色管理,部门管理	3/29/2023, 11:41:34 AM	Edit Set Permission Delete
java	待我审批		3/29/2023, 3:22:33 PM	Edit Set Permission Delete
测试			3/29/2023, 3:22:33 PM	Edit Set Permission Delete
行政		用户管理,菜单管理,角色管理,部门管理,休假申请	3/29/2023, 3:22:33 PM	Edit Set Permission Delete
前台			3/29/2023, 3:22:33 PM	Edit Set Permission Delete
绩效			3/29/2023, 3:22:33 PM	Edit Set Permission Delete
销售			3/29/2023, 3:22:33 PM	Edit Set Permission Delete
管理员		用户管理,菜单管理,角色管理,部门管理,待我审批,休假申请	3/29/2023, 3:22:33 PM	Edit Set Permission Delete
财务			3/29/2023, 3:22:33 PM	Edit Set Permission Delete
售后			3/29/2023, 3:22:33 PM	Edit Set Permission Delete

## 角色操作（创建、编辑、删除）

## New Role

X

\* Role Name

Please Input Role Name

Remark

Please Input Remark

Cancel

Conform

Permission Set

Role Name [产品经理]

Select

Permission

- ▼  系统管理
- ▼  用户管理
  - 新增
  - 删除
  - 编辑
  - 批量删除
- ▼  菜单管理
  - 编辑
  - 删除
- ▼  角色管理
  - 新增
  - 删除
- ▼  部门管理
  - 新增
  - 删除
- ▼  审批管理
  - ▼  待我审批
    - 等待
    - 已经处理
  - ▼  休假申请
    - 查询

## 角色管理-权限控制

关注当点击设置权限需要显示的对话框：

```
<el-button type="warning" size="small" @click="handlePermission(scope.row)">设置权限</el-button>
```

绑定事件

事件处理程序

```
function handlePermission(row){
```

```
}
```

需要设置对应的响应式对象。

```
const permissionDialogVisible = ref(false);
const currentRoleName = ref('');
let permissionForm = reactive({});

function handlePermission(row){
    currentRoleName.value = row.roleName;
    permissionDialogVisible.value = true;
}
```

## Tree 树形控件

树节点可以在初始化阶段被设置为展开和选中。

分别通过 default-expanded-keys 和 default-checked-keys 设置默认展开和默认选中的节点。

# 部门管理-创建、编辑、删除服务端功能实现

前端部门管理展示的数据是树形结构的，而我们在数据库存储的部门数据是平级结构。所以这里需要跟菜单管理一样在服务端不仅是数据查询还有数据拼接的操作，再返回到客户端来。

编写 departmentSchema.js

编写router department.js

注册路由 app.js

编写路由，先完成创建/编辑/删除功能



Dept Name: Please Input Department

Search Reset

Create				
Department Name	Leader	Update Time	Create Time	Operations
ByteDance	admin	12/1/2022, 3:53:37 AM	12/1/2022, 3:53:37 AM	Edit Delete
研发部	felix	3/29/2023, 9:34:26 PM	3/29/2023, 9:34:26 PM	Edit Delete
前端	lee	3/29/2023, 9:34:26 PM	3/29/2023, 9:34:26 PM	Edit Delete
行政部	admin	3/29/2023, 9:34:26 PM	3/29/2023, 9:34:26 PM	Edit Delete
财务部	felix	4/1/2023, 9:12:17 PM	4/1/2023, 9:12:17 PM	Edit Delete

### Edit Department's Information

\* Superior: Selection

\* Dept Name: ByteDance

\* Leader: admin

Email: admin@mashibing.com

Cancel Conform

## New Department

X

\* Superior

Selection



\* Dept Name

ByteDance

\* Leader

admin



Email

admin@mashibing.com

Cancel

Conform

## 权限中心-按钮权限控制

在前端保存后台接口返回的权限数据，再自定义一个指令 v-permission 然后判断当前用户是否有此按钮的权限，有就保留按钮，没有就删除。

## 审批管理 - 休假申请

### 接口字段

applicant 申请人

type 表示休假类型的变量，例如病假、年假、调休等。

startDate 休假起始时间。

endDate 休假截止时间。

reasons 休假原因

order 订单号

approvers 申请人的直接上级或人力资源部门/财务部

currentApprover 当前审批人

auditFlows 审批流程

logs 审批日志

## 定义表格列头

```
//定义表格列头
const columns = reactive([
    {label:"订单号", prop:"order", width:180},
    {
        label:"申请时间",
        prop:"startDate",
        formatter(row, column, value){
            let time = new Date(value)
            const options = { year: 'numeric', month: '2-digit', day: '2-digit' };
            return time.toLocaleDateString(options);
        }
    },
    {label:"休假时长", prop:"leaveTime"},
    {
        label:"休假类型",
        prop:"type",
        formatter(row, column, value){
            return {
                1:"事假",
                2:"病假",
                3:"年假",
                4:"调休"
            }[value];
        }
    },
    {label:"请假原因", prop:"reasons"},
    {label:"审批人", prop:"approvers"},
    {label:"当前审批人", prop:"currentApprover"},
    {
        label:"审批状态",
        prop:"approveState",
        formatter(row, column, value){
            return {
                1:"作废",
                2:"审批拒绝",
                3:"待审批",
                4:"审批通过"
            }[value];
        }
    },
]);

```

## 时间格式化

```
formatter(row, column, value){
  let time = new Date(value)
  const options = { year: 'numeric', month: '2-digit', day: '2-digit' };
  return time.toLocaleDateString(options);
}
```

## 休假时长(动态计算)

```
//data select
function handleDateChange(){
  let {startDate, endDate} = leaveForm;
  if(startDate && endDate){
    if(startDate > endDate){
      ElMessage.error("请假起始时间不能大于休假时间");
      leaveForm.startDate = '';
      return;
    }
    //计算请假时长
    leaveForm.leaveTime = (endDate - startDate)/(24*60*60*1000) + "天"
  }
}
```

## 休假申请-服务端开发

新建leaveSchama

新建路由 router/leave.js

注册路由

编写 list 后端接口

前端需要合并分页对象

## 待我审批-前后端开发

To be continue....