

# 一、若依概述

官网: <https://www.ruoyi.vip/>

RuoYi-Vue是一款基于SpringBoot+Vue的前后端分离极速后台开发框架。

**RuoYi-Vue** 是一个 Java EE 企业级快速开发平台, 基于经典技术组合 (Spring Boot、Spring Security、MyBatis、Jwt、Vue), 内置模块如: 部门管理、角色用户、菜单及按钮授权、数据权限、系统参数、日志管理、代码生成等。在线定时任务配置; 支持集群, 支持多数据源, 支持分布式事务。

## 系统需求

- JDK >= 1.8
- MySQL >= 5.7
- Maven >= 3.0
- Node >= 12
- Redis >= 3

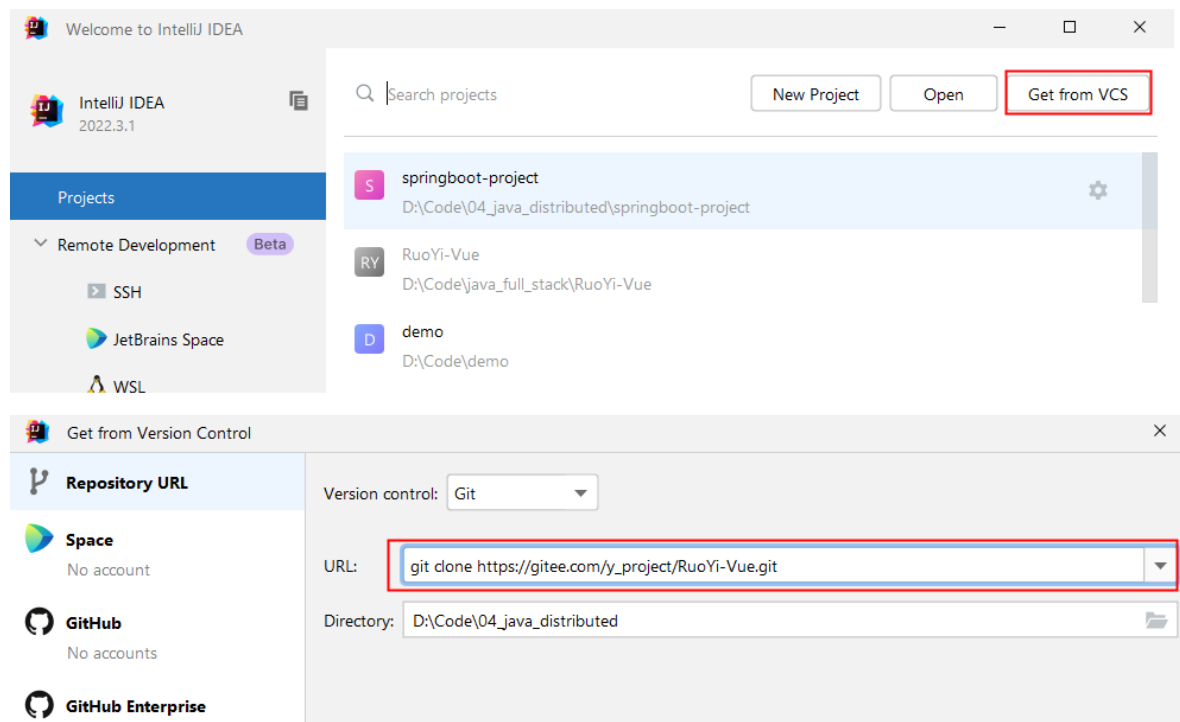
无论将来是否用得到, 都推荐大家熟练掌握若依:

- 开发使用: 提高开发效率, 减少工作量
- 学习使用: 学习优秀开源项目底层的编程思想、设计思路, 提高我们的编程能力

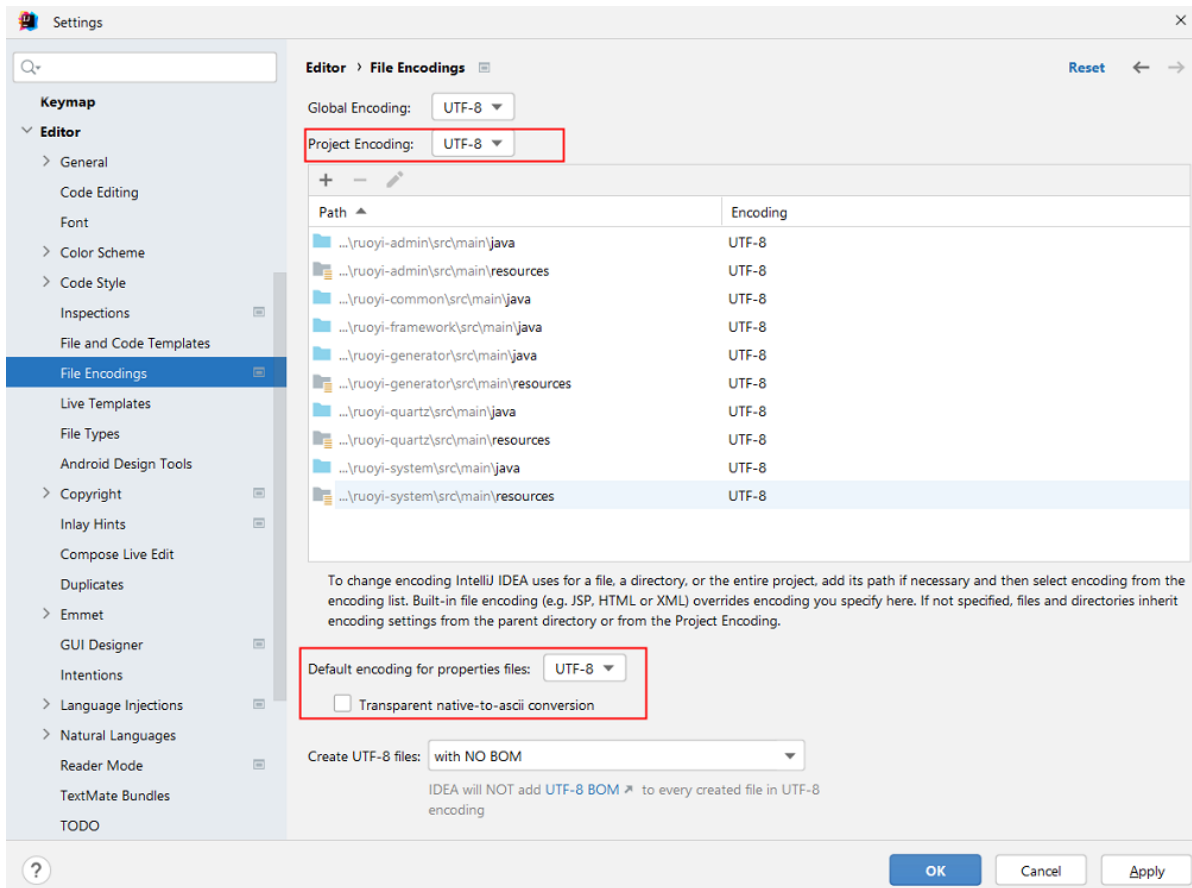
# 二、环境部署

## 下载

推荐使用IDEA直接导入项目代码:



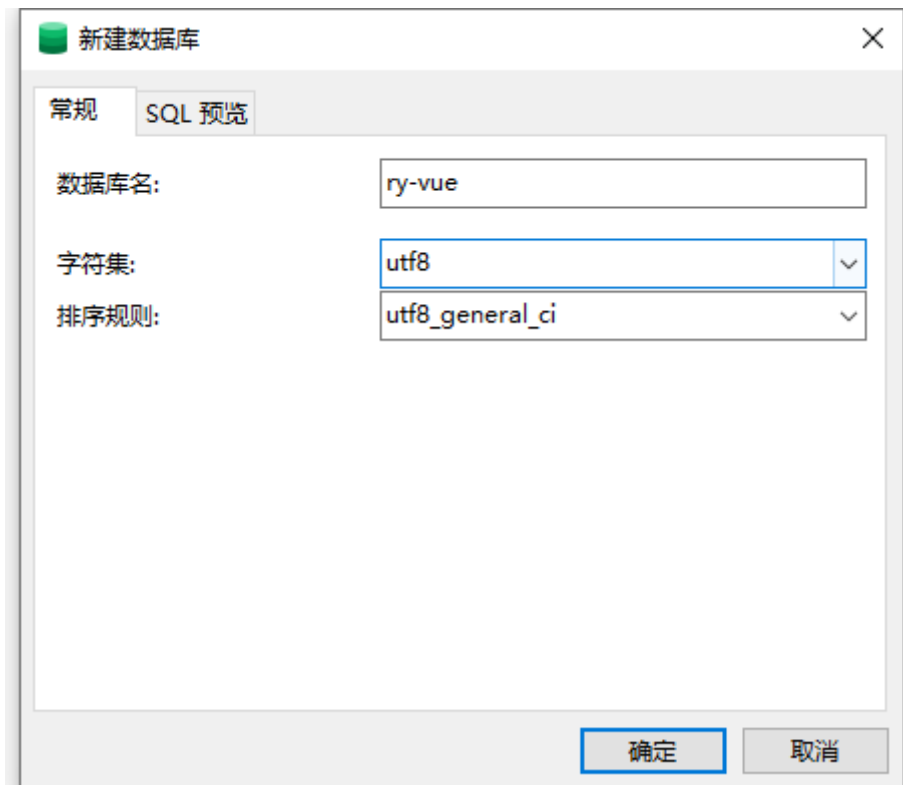
修改项目编码:



## 后端运行

1、创建数据库 ry-vue 并导入数据脚本 ry\_2021xxxx.sql , quartz.sql

- 我使用的是阿里云数据库
- 导入sql
  - 在下载的代码的sql文件夹下面有两个sql脚本；
  - 直接拖进去Navicat执行即可；



## 2、配置数据库MySQL、Redis

- 配置MySQL: ruoyi-admin模块/application-druid.yml

```
# 数据源配置
spring:
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    driverClassName: com.mysql.cj.jdbc.Driver
    druid:
      # 主库数据源
      master:
        url: jdbc:mysql://rm-
2zew01c60a4qd11spqo.mysql.rds.aliyuncs.com:3306/ry-vue?
useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&us
eSSL=true&serverTimezone=GMT%2B8
        username: root
        password: jeb@123456
```

- 配置Redis: ruoyi-admin模块/application.yml
  - 如果使用的是本机的Redis, 并且端口是6379, 那么无需更改配置;
  - **启动本机的Redis;**

## 3、运行ruoyi-admin模块/RuoYiApplication类, 启动SpringBoot项目

## 4、测试: localhost:8080, 出现如下内容, 启动成功

欢迎使用RuoYi 后台管理框架, 当前版本: v3.8.7, 请通过前端地址访问。

# 前端运行

- 1、使用HbuilderX导入ruoyi-ui项目
- 2、进入项目根目录的命令行窗口, 执行npm install 安装依赖
- 3、使用 npm run dev 启动项目
- 4、打开浏览器, 输入: (<http://localhost:80>) 默认账户/密码 admin/admin123)
  - 若能正确展示登录页面, 并能成功登录, 菜单及页面展示正常, 则表明环境搭建成功

# 三、项目介绍

## 文件结构

## 后端结构

```
com.ruoyi
├── common          // 工具类
│   ├── annotation // 自定义注解
│   ├── config      // 全局配置
│   ├── constant    // 通用常量
│   ├── core        // 核心控制
│   └── enums       // 通用枚举
```

```

|       └─ exception                // 通用异常
|       └─ filter                  // 过滤器处理
|       └─ utils                   // 通用类处理
└─ framework                      // 框架核心
    |       └─ aspectj              // 注解实现
    |       └─ config               // 系统配置
    |       └─ datasource            // 数据权限
    |       └─ interceptor           // 拦截器
    |       └─ manager              // 异步处理
    |       └─ security             // 权限控制
    |       └─ web                  // 前端控制
└─ ruoyi-generator                // 代码生成（可移除）
└─ ruoyi-quartz                  // 定时任务（可移除）
└─ ruoyi-system                  // 系统代码
└─ ruoyi-admin                   // 后台服务
└─ ruoyi-xxxxxx                 // 其他模块

```

## 前端结构

```

└─ build                          // 构建相关
└─ bin                           // 执行脚本
└─ public                        // 公共文件
    |   └─ favicon.ico            // favicon图标
    |   └─ index.html             // html模板
    |   └─ robots.txt            // 反爬虫
└─ src                           // 源代码
    |   └─ api                   // 所有请求
    |   └─ assets                // 主题 字体等静态资源
    |   └─ components            // 全局公用组件
    |   └─ directive             // 全局指令
    |   └─ layout                // 布局
    |   └─ plugins               // 通用方法
    |   └─ router                // 路由
    |   └─ store                 // 全局 store管理
    |   └─ utils                 // 全局公用方法
    |   └─ views                 // view
    |   └─ App.vue               // 入口页面
    |   └─ main.js               // 入口 加载组件 初始化等
    |   └─ permission.js         // 权限管理
    |   └─ settings.js           // 系统配置
└─ .editorconfig                 // 编码格式
└─ .env.development              // 开发环境配置
└─ .env.production               // 生产环境配置
└─ .env.staging                  // 测试环境配置
└─ .eslintignore                 // 忽略语法检查
└─ .eslintrc.js                  // eslint 配置项
└─ .gitignore                    // git 忽略项
└─ babel.config.js               // babel.config.js
└─ package.json                  // package.json
└─ vue.config.js                 // vue.config.js

```

## 配置文件

## 核心技术

# 四、验证码功能

## 验证码实现

1、当我们访问<http://localhost:80/>，根据路由规则应该打开的是后台首页，但是打开的是登录页面

- 配置了几个静态的路由
- 其他的都是根据用户权限加载的动态路由

```
// 当访问http://localhost:80/，匹配到这个路由配置【router/index.js】
{
  path: '',
  component: Layout, // 加载Layout组件, import Layout from
  '@/layout/index.vue' 后台首页组件
  redirect: 'index',
  children: [
    {
      path: 'index',
      component: () => import('@/views/index'),
      name: 'Index',
      meta: { title: '首页', icon: 'dashboard', affix: true }
    }
  ]
}

// 在permission.js里面，定义了全局前置路由守卫，如果没有登录，跳转到登录组件
// 通过token判断是否登录
router.beforeEach((to, from, next) => {
  NProgress.start()
  if (getToken()) {
    to.meta.title && store.dispatch('settings/setTitle', to.meta.title)
    /* has token*/
    if (to.path === '/login') {
      next({ path: '/' })
      NProgress.done()
    } else if (whiteList.indexOf(to.path) !== -1) {
      next()
    } else {
      if (store.getters.roles.length === 0) {
        isReLogin.show = true
        // 判断当前用户是否已拉取完user_info信息
        store.dispatch('GetInfo').then(() => {
          isReLogin.show = false
          store.dispatch('GenerateRoutes').then(accessRoutes => {
            // 根据roles权限生成可访问的路由表
            router.addRoutes(accessRoutes) // 动态添加可访问路由表
          })
        })
      }
    }
  } else {
    // 如果没有登录，重定向到登录页
    next('/login')
  }
})
```

```

        next({ ...to, replace: true }) // hack方法 确保addRoutes已完成
    })
    }).catch(err => {
        store.dispatch('Logout').then(() => {
            Message.error(err)
            next({ path: '/' })
        })
    })
} else {
    next()
}
}
} else {
    // 没有token
    if (whitelist.indexOf(to.path) !== -1) {
        // 在免登录白名单，直接进入
        next()
    } else {
        next(`/login?redirect=${encodeURIComponent(to.fullPath)}`) // 否则全部重定向到登录页
        NProgress.done()
    }
}
}
})

```

## 2、验证码：验证码是由后端生成的

- 验证码分为前端验证码和后端验证码
- 若依里面的验证是由后端生成，当打开登录页面就向后台发送请求获取验证码
- 请求验证码的地址：<http://localhost/dev-api/captchaImage>，通过反向代理，最终的请求路径是<http://localhost:8080/captchaImage>
- 根据/captchaImage去ruoyi-admin模块，进行查找【ctrl+shift+f】，输入/captchaImage即可
- 验证码的基本思路：
  - 后端生成一个表达式文本字符串：1+1=?@2
  - 把字符串根据@符号拆分为两部分，1+1=?转换为base64图片，传给前端，前端展示图片
  - 结果2存储在Redis缓存，2min有效期
    - 缓存的key：固定的字符串 + 随机生成一个UUID
    - 缓存的值：2

```

/**
 * 生成验证码
 */
@GetMapping("/captchaImage")
public AjaxResult getCode(HttpServletRequest response) throws IOException {
    // AjaxResult是统一结果集对象
    AjaxResult ajax = AjaxResult.success();
    boolean captchaEnabled = configService.selectCaptchaEnabled(); // 是否开启验证码
    ajax.put("captchaEnabled", captchaEnabled);

    // 如果没有开启验证码，直接返回
}

```

```

    if (!captchaEnabled)
    {
        return ajax;
    }

    // 保存验证码信息
    String uuid = IdUtils.simpleUUID();
    String verifyKey = CacheConstants.CAPTCHA_CODE_KEY + uuid; //
captcha_codes:uuid

    String capStr = null, code = null;
    BufferedImage image = null;

    // 生成验证码
    String captchaType = RuoyiConfig.getCaptchaType(); // 获取验证码类型 math、
char
    if ("math".equals(captchaType)) {
        String capText = captchaProducerMath.createText(); // 创建验证码文本字
        符串
        // 根据@符号拆分
        capStr = capText.substring(0, capText.lastIndexOf("@"));
        code = capText.substring(capText.lastIndexOf("@") + 1);
        image = captchaProducerMath.createImage(capStr); // 把表达式转换为图片
    }
    else if ("char".equals(captchaType))
    {
        capStr = code = captchaProducer.createText();
        image = captchaProducer.createImage(capStr);
    }

    // 把验证码结果放到redis缓存中, 2min有效期
    redisCache.setCacheObject(verifyKey, code, Constants.CAPTCHA_EXPIRATION,
TimeUnit.MINUTES);
    // 转换流信息写出
    FastByteArrayOutputStream os = new FastByteArrayOutputStream();
    try
    {
        ImageIO.write(image, "jpg", os);
    }
    catch (IOException e)
    {
        return AjaxResult.error(e.getMessage());
    }

    ajax.put("uuid", uuid);
    ajax.put("img", Base64.encode(os.toByteArray()));
    return ajax;
}

```

### 3、前端是在哪里发送的获取验证码的请求：

- 在login.vue的created()声明周期函数中，执行了两个方法

```
this.getCode(); // 获取验证码
this.getCookie(); // 获取Cookie里面的数据，勾选记住密码，数据放到Cookie中，从Cookie
里面获取保存的用户名和密码，进行回显
```

- getCode方法的实现：

```
getCode() {
  // axios.get('').then(), getCodeImg就是axios的封装
  getCodeImg().then(res => { // 请求成功之后的处理
    // 复制给data中的变量
    this.captchaEnabled = res.captchaEnabled === undefined ? true :
res.captchaEnabled;
    if (this.captchaEnabled) {
      this.codeUrl = "data:image/gif;base64," + res.img;
      this.loginForm.uuid = res.uuid;
    }
  });
}
```

- axios的封装：

```
import { getCodeImg } from "@/api/login";

// 去api/login.js这个模块化文件
// 获取验证码
export function getCodeImg() {
  // request就是对axios({})的封装
  return request({
    url: '/captchaImage',
    headers: {
      isToken: false
    },
    method: 'get',
    timeout: 20000
  })
}

// 去找 '@/utils/request'
// 在request.js里面创建axios示例，设置请求拦截器和响应拦截器
```

## 前端请求路径解析

<http://localhost/dev-api/captchaImage>，Vue用来获取验证码图片，验证码是由后端生成的，那么为什么端口是80呢？

- 前端端口：80
- 后端端口：8080

因为使用了前端反向代理，url路径请求的前端，进行代理之后，映射到后端，解决跨域；

- 前端跨域解决
- 推荐大家还是后端解决跨域

vue.config.js反向代理配置：



```
// 我们通过npm run dev 启动的开发环境
devServer: {
  host: '0.0.0.0', // 本机
  port: port, // 80
  open: true, // 开启代理
  proxy: { // 代理设置
    // /dev-api 当发送请求是http://localhost:80/dev-api, 才会走代理, 访问后端请求
    [process.env.VUE_APP_BASE_API]: {
      target: `http://localhost:8080`,
      changeOrigin: true,
      pathRewrite: {
        ['^' + process.env.VUE_APP_BASE_API]: '' // /dev-api重写为 ''
      }
    },
    // http://localhost/dev-api/captchaImage
    // http://localhost:8080/dev-api/captchaImage
    // http://localhost:8080/captchaImage
  }
}
```

vue项目环境快速切换: <https://blog.csdn.net/shaogaiyue9745602/article/details/127883774>

## 五、登录功能

### 前端实现

#### 1、当我们点击登录按钮，发送请求给后端

- 当我们点击登录按钮，触发handleLogin事件函数

```
<el-button :loading="loading"
  size="medium"
  type="primary"
  style="width:100%;"
  @click.native.prevent="handleLogin"
>
  <span v-if="!loading">登 录</span>
  <span v-else>登 录 中...</span>
</el-button>
```

- 在handleLogin()中存储Cookie、移除Cookie、提交给后端请求

this.\$store.dispatch: <https://www.python100.com/html/84811.html>

this.\$store.dispatch是VueX框架中用来触发actions的方法

```
handleLogin() {
  this.$refs.loginForm.validate(valid => {
    if (valid) { // 校验通过
      this.loading = true;
      if (this.loginForm.rememberMe) { // 如果勾选了记住密码，把输入的账号、
        密码、记住密码放到Cookie中，30天有效期
        Cookies.set("username", this.loginForm.username, { expires:
        30 });
      }
    }
  });
}
```

```

        Cookies.set("password", encrypt(this.loginForm.password), {
expires: 30 });
        Cookies.set('rememberMe', this.loginForm.rememberMe, {
expires: 30 });
    } else { // 没有勾选记住密码, 把Cookie里面的都移除掉
        Cookies.remove("username");
        Cookies.remove("password");
        Cookies.remove('rememberMe');
    }
    // 发送请求给后端服务器
    // this.$store.dispatch是Vuex框架中用来触发actions的方法
    this.$store.dispatch("Login", this.loginForm).then(() => {
        this.$router.push({ path: this.redirect || "/" }).catch(()=>
{}});

    }).catch(() => {
        this.loading = false;
        if (this.captchaEnabled) {
            this.getCode();
        }
    });
    }
    });
}

```

```

// Vuex actions
actions: {
    // 登录
    Login({ commit }, userInfo) {
        const username = userInfo.username.trim()
        const password = userInfo.password
        const code = userInfo.code
        const uuid = userInfo.uuid
        return new Promise((resolve, reject) => {
            login(username, password, code, uuid).then(res => {
                setToken(res.token)
                commit('SET_TOKEN', res.token)
                resolve()
            }).catch(error => {
                reject(error)
            })
        })
    },
}

```

```

// api/login.js
export function login(username, password, code, uuid) {
    const data = {
        username,
        password,
        code,
        uuid
    }
    return request({
        url: '/login',

```

```

headers: {
    isToken: false,
    repeatSubmit: false
},
method: 'post',
data: data // axios中put和post发送的json数据
})
}

```

## 后端实现

1、根据/login，找到对应Controller里面的方法

- 控制层: SysLoginController 类 login()

```

@PostMapping("/login")
public AjaxResult login(@RequestBody LoginBody loginBody) {
    AjaxResult ajax = AjaxResult.success();
    // 生成令牌
    String token = loginService.login(loginBody.getUsername(),
    loginBody.getPassword(), loginBody.getCode(),
    loginBody.getUuid());
    ajax.put(Constants.TOKEN, token);
    return ajax;
}

```

- 业务逻辑层:
  - 校验验证码: 验证码在内存中, 验证速度快; 如果验证码不通过, 不去做数据库操作;
  - 登录前置校验: 判断用户名、密码格式是否正确
  - 用户验证: 判断账号和密码是否正确, 使用Spring Security来进行身份认证和权限鉴定

```

public String login(String username, String password, String code, String
uuid) {
    // 验证码校验
    validateCaptcha(username, code, uuid);
    // 登录前置校验
    loginPreCheck(username, password);
    // 用户验证
    Authentication authentication = null;
    try {
        // 使用Spring Security进行身份认证, 如果用户名和密码正确, 则没有异常; 否则抛出
        异常
        UsernamePasswordAuthenticationToken authenticationToken = new
        UsernamePasswordAuthenticationToken(username, password);
        AuthenticationContextHolder.setContext(authenticationToken);
        // 该方法会去调用UserDetailsServiceImpl.loadUserByUsername
        authentication =
        authenticationManager.authenticate(authenticationToken);
    } catch (Exception e) {
        if (e instanceof BadCredentialsException)
        {

        }

        AsyncManager.me().execute(AsyncFactory.recordLogininfor(username,
        Constants.LOGIN_FAIL, MessageUtils.message("user.password.not.match")));
    }
}

```

```

        throw new UserPasswordNotMatchException();
    }
    else
    {

        AsyncManager.me().execute(AsyncFactory.recordLogininfor(username,
        Constants.LOGIN_FAIL, e.getMessage()));
        throw new ServiceException(e.getMessage());
    }
} finally {
    AuthenticationContextHolder.clearContext();
}
// 异步任务管理器：记录登录操作日志
AsyncManager.me().execute(AsyncFactory.recordLogininfor(username,
Constants.LOGIN_SUCCESS, MessageUtils.message("user.login.success")));
LoginUser loginUser = (LoginUser) authentication.getPrincipal(); // 获取
登录成功的用户对象
recordLoginInfo(loginUser.getUserId()); // 记录登录信息 【最后一次登录的时间】
// 生成token
return tokenService.createToken(loginUser);
}

```

生成的token里面是包含了用户的信息，以后就使用token进行身份验证；

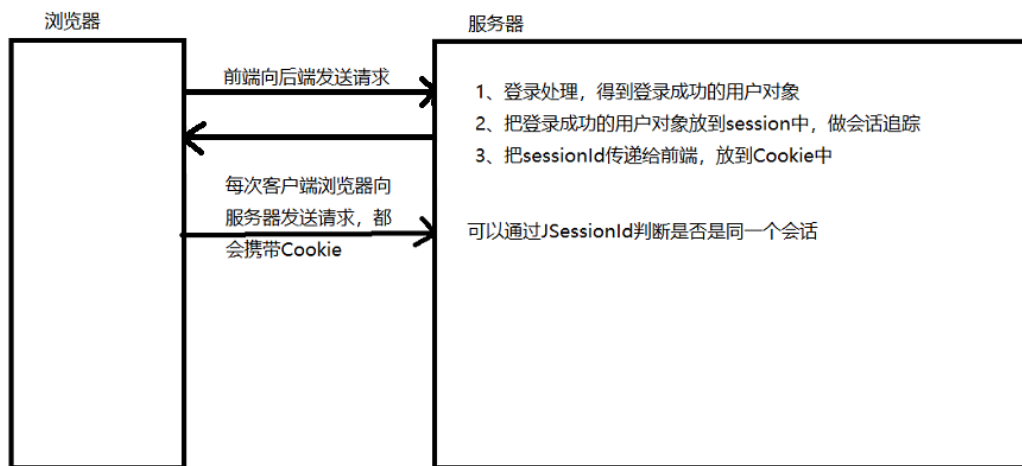
## Token

Token是服务端生成的一串字符串，以作客户端进行请求的一个令牌，**当第一次登录后，服务器生成一个Token便将此Token返回给客户端，以后客户端只需带上这个Token前来请求数据即可，无需再次带上用户名和密码**

使用token机制的身份验证方法，在服务器端不需要存储用户的登录记录

**大概的流程：**

- 客户端使用用户名和密码请求登录(服务器--后端)
- 服务端收到请求，验证用户名和密码
- 验证成功后，服务端(后台)会生成一个token，然后把这个token发送给客户端(前端)
- 客户端收到token后把它存储起来，可以放在cookie或者Local Storage（本地存储）里【Vuex】
- 客户端每次向服务端发送请求的时候都需要带上服务端发给的token(客户端--前端)
- 服务端收到请求，然后去验证客户端请求里面带着token，如果验证成功，就向客户端返回请求的数据
- 这个token必须在每次请求时传递给服务器，应该保存在请求头里，另外服务器要支持CORS(跨域资源共享)的策略



会话：从客户端打开浏览器开始访问服务器，到服务器给客户端响应数据，客户端继续访问服务器，\*\*直到客户端关闭浏览器\*\*，整个过程称为一次客户端和服务端之间的会话。

如果我们使用axios发送异步请求，那么就没有这种会话追踪；

对于服务器来说，每次异步请求都是一个新的会话

## 扩展

若依管理系统也实现了注册功能，只是隐藏起来了而已；

前端login.vue：

```

<!-- register设置为true，显示注册 -->
<div style="float: right;" v-if="register">
  <router-link class="link-type" :to="'/register'">立即注册</router-link>
</div>

```

后端：系统管理 -> 参数设置 -> 账号自主-是否开启用户注册功能，参数键值改为true即可；

## 六、后台主页面

### 后台主页结构

1、当请求登录成功，跳转到 `/`，就是后台首页

```

this.$store.dispatch("Login", this.loginForm).then(() => {
  // 登录成功，路由跳转到/
  this.$router.push({ path: this.redirect || "/" }).catch(()=>{});
}).catch(() => {
  this.loading = false;
  if (this.captchaEnabled) {
    this.getCode();
  }
});

```

2、后台首页组件 `Layout`，在嵌套路由里面展示index

```

// 当访问http://localhost:80/，匹配到这个路由配置【router/index.js】
{
  path: '',
  component: Layout, // 加载Layout组件，import Layout from '@/layout/index.vue'
}

```

后台首页组件

```

    redirect: 'index', // 重定向到index组件
    children: [ // 嵌套路由
      {
        path: 'index',
        component: () => import('@views/index'),
        name: 'Index',
        meta: { title: '首页', icon: 'dashboard', affix: true }
      }
    ]
  }
}

```

```

<template>
  <div :class="classObj" class="app-wrapper" :style="{ '--current-color':
theme}">
    <!-- 如果是移动设备，进行折叠侧边栏菜单 -->
    <div v-if="device==='mobile'&&sidebar.opened" class="drawer-bg"
@click="handleClickOutside" />

    <!-- 自定义的sidebar组件，里面封装了侧边栏菜单 -->
    <sidebar v-if="!sidebar.hide" class="sidebar-container" />
    <!-- 右侧主区域：导航菜单、标签页区域、主区域、右侧panel -->
    <div :class="{hasTagsView:needTagsView,sidebarHide:sidebar.hide}"
class="main-container">
      <div :class="{ 'fixed-header':fixedHeader}">
        <!-- 导航菜单 -->
        <navbar />
        <!-- 标签页区域 -->
        <tags-view v-if="needTagsView" />
      </div>
      <!-- 主区域 -->
      <app-main />
      <!-- 右侧panel -->
      <right-panel>
        <settings />
      </right-panel>
    </div>
  </div>
</template>

```

### 3、在SideBar组件中，去后台查询动态菜单；


- 在打开后台的首页，发送了两个请求：
  - <http://localhost/dev-api/getInfo>：获取当前用户的角色信息、权限信息
  - <http://localhost/dev-api/getRouters>：获取动态菜单
- getInfo的返回值：

```
{
  "msg": "操作成功",
  "code": 200,
  "permissions": [ // 权限信息
    "*:*:~*"
  ],
  "roles": [ // 角色信息
    "admin"
  ],
  "user": { // 用户信息
  }
}
```

- getRouters的返回值：查询当前用户的权限，符合树形结构的数据；
  - 后端通过数据库查询得到的都是列表；
  - 怎么得到的这样的数据结构呢？后台处理数据得到的

## 权限五表

1. sys\_user：用户信息表，任何系统里面都包含用户，存储用户的基本信息；

字段	索引	外键	检查	触发器	选项	注释	SQL 预览					
名						类型	长度	小数点	不是 null	虚拟	键	注释
user_id						bigint			<input checked="" type="checkbox"/>	<input type="checkbox"/>	 1	用户ID
dept_id						bigint			<input type="checkbox"/>	<input type="checkbox"/>		部门ID
user_name						varchar	30		<input checked="" type="checkbox"/>	<input type="checkbox"/>		用户账号
nick_name						varchar	30		<input checked="" type="checkbox"/>	<input type="checkbox"/>		用户昵称
user_type						varchar	2		<input type="checkbox"/>	<input type="checkbox"/>		用户类型 (00系统用户)
email						varchar	50		<input type="checkbox"/>	<input type="checkbox"/>		用户邮箱
phonenumber						varchar	11		<input type="checkbox"/>	<input type="checkbox"/>		手机号码
sex						char	1		<input type="checkbox"/>	<input type="checkbox"/>		用户性别 (0男 1女 2未知)
avatar						varchar	100		<input type="checkbox"/>	<input type="checkbox"/>		头像地址
password						varchar	100		<input type="checkbox"/>	<input type="checkbox"/>		密码
status						char	1		<input type="checkbox"/>	<input type="checkbox"/>		帐号状态 (0正常 1停用)
del_flag						char	1		<input type="checkbox"/>	<input type="checkbox"/>		删除标志 (0代表存在 2代表删除)
login_ip						varchar	128		<input type="checkbox"/>	<input type="checkbox"/>		最后登录IP
login_date						datetime			<input type="checkbox"/>	<input type="checkbox"/>		最后登录时间
create_by						varchar	64		<input type="checkbox"/>	<input type="checkbox"/>		创建者
create_time						datetime			<input type="checkbox"/>	<input type="checkbox"/>		创建时间
update_by						varchar	64		<input type="checkbox"/>	<input type="checkbox"/>		更新者
update_time						datetime			<input type="checkbox"/>	<input type="checkbox"/>		更新时间
remark						varchar	500		<input type="checkbox"/>	<input type="checkbox"/>		备注

- 从用户表角度看：一个用户可以拥有多个角色，比如赵老师，她即是班主任，又是就业老师
- 从角色表角度看：一个角色可以被多个用户拥有，比如张三和李四都是市场人员
- 所以用户表和角色表是多对多，肯定存在中间表

2. sys\_user\_role：用户和角色关联表


字段	索引	外键	检查	触发器	选项	注释	SQL 预览					
名						类型	长度	小数点	不是 null	虚拟	键	注释
user_id						bigint			<input checked="" type="checkbox"/>	<input type="checkbox"/>	 1	用户ID
role_id						bigint			<input checked="" type="checkbox"/>	<input type="checkbox"/>	 2	角色ID

3. sys\_role：角色信息表，在系统中，会有角色划分；比如OA系统，有总经理角色、人事角色、部门经理角色、员工角色...，不同的角色拥有不同的权限，也就是说登录系统之后，可以看到的菜单和可以进行的操作是不一样的；比如总经理可以看到员工信息、部门信息、财务报表、资产....，但是财务人员只能看到员工信息、财务报表...

- 从角色表角度看：一个角色可以拥有多个权限
- 从权限表角度看：一个权限可以被多个角色拥有
- 所以角色表和权限表是多对多，肯定存在中间表

字段	索引	外键	检查	触发器	选项	注释	SQL 预览					
名					类型		长度	小数点	不是 null	虚拟	键	注释
role_id					bigint				<input checked="" type="checkbox"/>	<input type="checkbox"/>	1	角色ID
role_name					varchar		30		<input checked="" type="checkbox"/>	<input type="checkbox"/>		角色名称
role_key					varchar		100		<input checked="" type="checkbox"/>	<input type="checkbox"/>		角色权限字符串
role_sort					int				<input checked="" type="checkbox"/>	<input type="checkbox"/>		显示顺序
data_scope					char		1		<input type="checkbox"/>	<input type="checkbox"/>		数据范围（1：全部数据权限 2：自定数据权限 3：本部门及以下所有权限）
menu_check_strictly					tinyint		1		<input checked="" type="checkbox"/>	<input type="checkbox"/>		菜单树选择项是否关联显示
dept_check_strictly					tinyint		1		<input type="checkbox"/>	<input type="checkbox"/>		部门树选择项是否关联显示
status					char		1		<input checked="" type="checkbox"/>	<input type="checkbox"/>		角色状态（0正常 1停用）
del_flag					char		1		<input type="checkbox"/>	<input type="checkbox"/>		删除标志（0代表存在 2代表删除）
create_by					varchar		64		<input type="checkbox"/>	<input type="checkbox"/>		创建者
create_time					datetime				<input type="checkbox"/>	<input type="checkbox"/>		创建时间
update_by					varchar		64		<input type="checkbox"/>	<input type="checkbox"/>		更新者
update_time					datetime				<input type="checkbox"/>	<input type="checkbox"/>		更新时间
remark					varchar		500		<input type="checkbox"/>	<input type="checkbox"/>		备注

#### 4. sys\_role\_menu: 角色和权限关联表

字段	索引	外键	检查	触发器	选项	注释	SQL 预览					
名						类型	长度	小数点	不是 null	虚拟	键	注释
▶ role_id						bigint			<input checked="" type="checkbox"/>	<input type="checkbox"/>	 1	角色ID
menu_id						bigint			<input checked="" type="checkbox"/>	<input type="checkbox"/>	 2	菜单ID

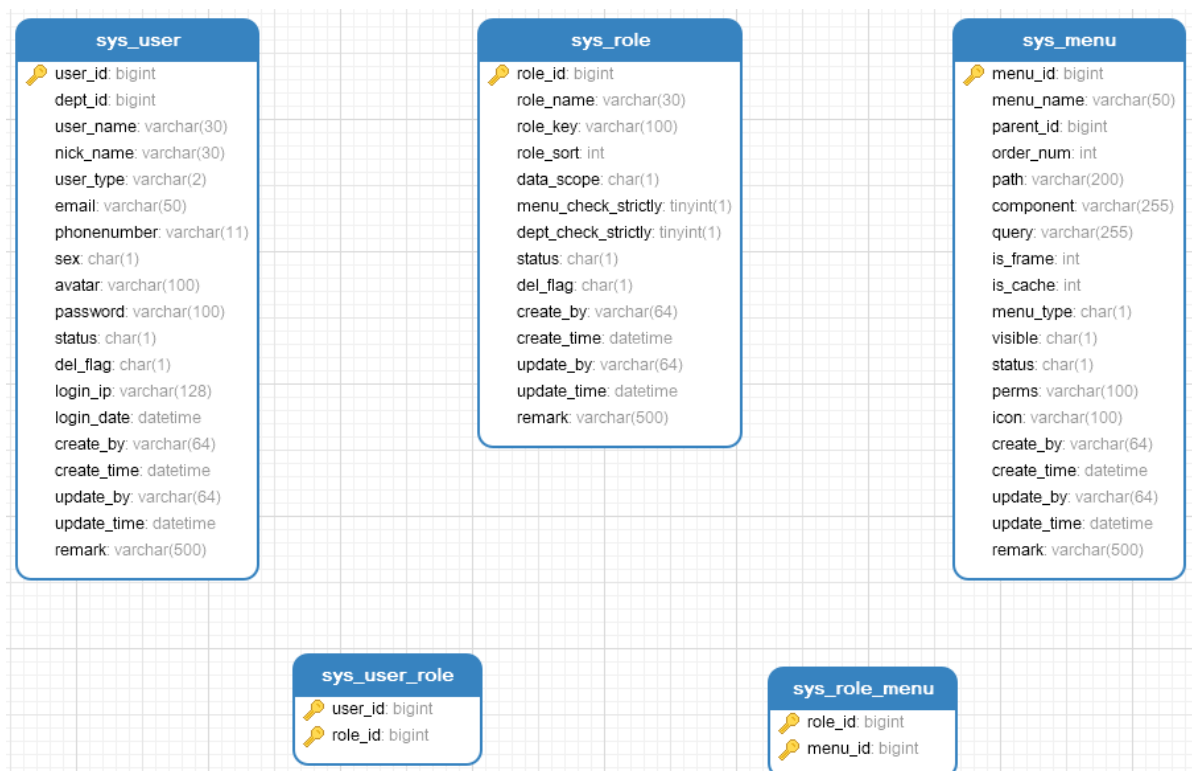
5. `sys_menu`: 菜单权限表, 不同的角色看到的菜单是不同的

字段	索引	外键	检查	触发器	选项	注释	SQL 预览					
名						类型	长度	小数点	不是 null	虚拟	键	注释
menu_id						bigint			<input checked="" type="checkbox"/>	<input type="checkbox"/>	 1	菜单ID
menu_name						varchar	50		<input checked="" type="checkbox"/>	<input type="checkbox"/>		菜单名称
parent_id						bigint			<input type="checkbox"/>	<input type="checkbox"/>		父菜单ID
order_num						int			<input type="checkbox"/>	<input type="checkbox"/>		显示顺序
path						varchar	200		<input type="checkbox"/>	<input type="checkbox"/>		路由地址
component						varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		组件路径
query						varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		路由参数
is_frame						int			<input type="checkbox"/>	<input type="checkbox"/>		是否为外链 (0是 1否)
is_cache						int			<input type="checkbox"/>	<input type="checkbox"/>		是否缓存 (0缓存 1不缓存)
menu_type						char	1		<input type="checkbox"/>	<input type="checkbox"/>		菜单类型 (M目录 C菜单 F按钮)
visible						char	1		<input type="checkbox"/>	<input type="checkbox"/>		菜单状态 (0显示 1隐藏)
status						char	1		<input type="checkbox"/>	<input type="checkbox"/>		菜单状态 (0正常 1停用)
perms						varchar	100		<input type="checkbox"/>	<input type="checkbox"/>		权限标识
icon						varchar	100		<input type="checkbox"/>	<input type="checkbox"/>		菜单图标
create_by						varchar	64		<input type="checkbox"/>	<input type="checkbox"/>		创建者
create_time						datetime			<input type="checkbox"/>	<input type="checkbox"/>		创建时间
update_by						varchar	64		<input type="checkbox"/>	<input type="checkbox"/>		更新者
update_time						datetime			<input type="checkbox"/>	<input type="checkbox"/>		更新时间
remark						varchar	500		<input type="checkbox"/>	<input type="checkbox"/>		备注

生成动态菜单

结合权限验证框架使用

### 五张表总览：





# 获取用户角色和权限

<http://localhost/dev-api/getInfo>: 获取当前用户的角色信息、权限信息

- Controller

```
@GetMapping("getInfo")
public AjaxResult getInfo() {
    // 从Spring Security中获取当前登陆的用户
    SysUser user = SecurityUtils.getLoginUser().getUser();
    // 角色集合
    Set<String> roles = permissionService.getRolePermission(user);
    // 权限集合
    Set<String> permissions = permissionService.getMenuPermission(user);
    // 数据封装
    AjaxResult ajax = AjaxResult.success();
    ajax.put("user", user);
    ajax.put("roles", roles);
    ajax.put("permissions", permissions);
    return ajax;
}
```

- Service

```
public Set<String> getRolePermission(SysUser user) {
    Set<String> roles = new HashSet<String>();
    // 管理员拥有所有权限
    if (user.isAdmin()) {
        roles.add("admin");
    }
    else { // 不是管理员，查询当前用户的角色列表

        roles.addAll(roleService.selectRolePermissionByUserId(user.getUserId()));
    }
    return roles;
}

public Set<String> getMenuPermission(SysUser user) {
    Set<String> perms = new HashSet<String>();
    // 管理员拥有所有权限
    if (user.isAdmin()) {
        perms.add("*:*:"); // 权限规则，拥有所有的权限
    }
    else {
        // 如果不是管理员，先获取当前用户的角色，然后根据角色id查询权限列表
        List<SysRole> roles = user.getRoles();
        if (!CollectionUtils.isEmpty(roles)) {
            // 多角色设置permissions属性，以便数据权限匹配权限
            for (SysRole role : roles) {
                Set<String> rolePerms =
                    menuService.selectMenuPermsByRoleId(role.getRoleId());
                role.setPermissions(rolePerms);
                perms.addAll(rolePerms);
            }
        }
    }
}
```

```

        else
        {

perms.addAll(menuService.selectMenuPermsByUserId(user.getUserId()));

        }
    }
    return perms;
}

```

## 获取动态菜单数据

<http://localhost/dev-api/getRouters>: 获取动态菜单

- Controller

```

/**
 * 获取路由信息：就是当前登录的用户可以看到哪些目录和菜单
 *
 * @return 路由信息
 */
@GetMapping("getRouters")
public AjaxResult getRouters() {
    // 获取当前用户的id
    Long userId = SecurityUtils.getUserId();
    // 获取树形菜单数据
    List<SysMenu> menus = menuService.selectMenuTreeByUserId(userId);
    return AjaxResult.success(menuService.buildMenus(menus));
}

```

- Service

```

/**
 * 根据用户ID查询菜单
 *
 * @param userId 用户名称
 * @return 菜单列表
 */
@Override
public List<SysMenu> selectMenuTreeByUserId(Long userId)
{
    List<SysMenu> menus = null;
    // 如果是管理员，查询所有的菜单数据
    if (SecurityUtils.isAdmin(userId)) {
        menus = menuMapper.selectMenuTreeAll();
    }
    else {
        // 如果不是管理员，根据id查询菜单数据
        menus = menuMapper.selectMenuTreeByUserId(userId);
    }
    // 构建树形数据【用到了递归调用】
    return getChildPerms(menus, 0);
}

```

# 七、用户管理

## 用户查询

### 前端实现

用户组件: `system/user/index.vue`

在index.vue组件的created()生命周期函数里面，调用两个方法：分页查询用户数据、查询所有的部门数据

```
created() {
  // 分页查询用户列表
  this.getList();

  // 查询所有部门的树形数据
  this.getDeptTree();
  this.getConfigKey("sys.user.initPassword").then(response => {
    this.initPassword = response.msg;
  });
},

/** 查询用户列表 */
getList() {
  this.loading = true;
  listUser(this.addDateRange(this.queryParams, this.dateRange)).then(response
=> {
    this.userList = response.rows;
    this.total = response.total;
    this.loading = false;
  })
},

/** 查询部门下拉树结构 */
getDeptTree() {
  deptTreeSelect().then(response => {
    this.deptOptions = response.data;
  });
},
```

### 后端实现

1、查询用户Controller层实现

```

/**
 * 获取用户列表 【分页带条件查询】
 * @Params SysUser user 封装了查询条件的用户对象
 * @return TableDataInfo 表格数据的结果集封装，和AjaxResult类似
 * @PreAuthorize 标识所需要的权限
 */
@PreAuthorize("@ss.hasPermi('system:user:list')")
@GetMapping("/list")
public TableDataInfo list(SysUser user) {
    startPage();
    List<SysUser> list = userService.selectUserList(user);
    return getDataTable(list);
}

```

## 2、查询用户Service层实现

```

/**
 * 根据条件分页查询用户列表
 *
 * @param user 用户信息
 * @return 用户信息集合信息
 */
@Override
@DataScope(deptAlias = "d", userAlias = "u")
public List<SysUser> selectUserList(SysUser user)
{
    return userMapper.selectUserList(user);
}

```

## 3、查询部门的Controller层实现

```

/**
 * 获取部门树列表
 */
@PreAuthorize("@ss.hasPermi('system:user:list')")
@GetMapping("/deptTree")
public AjaxResult deptTree(SysDept dept) {
    return success(deptService.selectDeptTreeList(dept));
}

```

## 4、查询部门的Service层实现

```

/**
 * 查询部门树结构信息
 *
 * @param dept 部门信息
 * @return 部门树信息集合
 */
@Override
public List<TreeSelect> selectDeptTreeList(SysDept dept) {
    List<SysDept> depts = SpringUtils.getAopProxy(this).selectDeptList(dept);
    return buildDeptTreeSelect(depts);
}

```

## 分页实现

如果自己在写分页带条件查询代码业务的时候，还是可以按照以前的写法写【Mybatis PageHelper】；

```
public AjaxResult list(@RequestParam(value = "pageNum", required = false,
    defaultValue = "1") Integer pageNum
    @RequestParam(value = "pageSize", required = false,
    defaultValue = "1") Integer pageSize
    SysUser sysUser
    )
```

若依对分页进行了进一步的封装：

```
@GetMapping("/list")
public TableDataInfo list(SysUser user) { // 在这里没有获取pageNum和pageSize
    startPage();
    List<SysUser> list = userService.selectUserList(user);
    return getDataTable(list);
}

// startPage(); 开启分页，封装了通过Servlet获取pageNum和pageSize
protected void startPage() {
    PageUtils.startPage();
}

/**
 * 设置请求分页数据
 */
public static void startPage() {
    PageDomain pageDomain = TableSupport.buildPageRequest(); // 通过请求参数构建分页
    实体对象
    Integer pageNum = pageDomain.getPageNum();
    Integer pageSize = pageDomain.getPageSize();
    String orderBy = SqlUtil.escapeOrderBySql(pageDomain.getOrderBy());
    Boolean reasonable = pageDomain.getReasonable();
    PageHelper.startPage(pageNum, pageSize, orderBy).setReasonable(reasonable);
}

public static PageDomain buildPageRequest() {
    return getPageDomain();
}

/**
 * 封装分页对象
 */
public static PageDomain getPageDomain()
{
    PageDomain pageDomain = new PageDomain();
    // 从Servlet请求中获取pageNum
    pageDomain.setPageNum(Convert.toInt(ServletUtils.getParameter(PAGE_NUM),
    1));
    // 从Servlet请求中获取PageSize
```

```

    pageDomain.setPageSize(Convert.toInt(ServletUtils.getParameter(PAGE_SIZE),
10));
    pageDomain.setOrderByColumn(ServletUtils.getParameter(ORDER_BY_COLUMN));
    pageDomain.setIsAsc(ServletUtils.getParameter(IS_ASC));
    pageDomain.setReasonable(ServletUtils.getParameterToBool(REASONABLE));
    return pageDomain;
}

```

## 请求部门员工数据

当我们点击部门列表项的时候，查询部门对应的员工数据

```

<el-tree
    :data="deptOptions"
    :props="defaultProps"
    :expand-on-click-node="false"
    :filter-node-method="filterNode"
    ref="tree"
    node-key="id"
    default-expand-all
    highlight-current
    @node-click="handleNodeClick"
/>

```

```

// handleNodeClick只是把部门id封装了请求参数中，执行的还是查询所有员工列表的方法
this.queryParams.deptId = data.id;

```

## 用户新增

### 前端实现

#### 1、当点击新增按钮的时候，执行handleAdd

- 显示对话框Dialog
- 预加载数据（部门列表数据、岗位列表数据、角色数据）

```

/** 新增按钮操作 */
handleAdd() {
    this.reset(); // 重置对话框表单数据
    // 新增：获取岗位数据和角色列表数据
    // 修改：获取岗位数据和角色列表数据，获取当前修改用户的岗位和角色
    getUser().then(response => {
        // 将查询得到的数据复制给data中的变量，做数据动态显示
        this.postOptions = response.posts;
        this.roleOptions = response.roles;
        this.open = true; // 显示对话框
        this.title = "添加用户"; // 修改对话框标题
        this.form.password = this.initPassword; // 设置初始密码
    });
}

// /api/system/user 查询用户详细

```

```
export function getUser(userId) { // 如果新增，不传递id；如果修改，肯定要传递修改用户id
    return request({
        url: '/system/user/' + parseStrEmpty(userId),
        method: 'get'
    })
}
```

## 2、当我们点击对话框的提交按钮的时候，执行submitForm

```
/** 提交按钮 */
submitForm: function() {
    // 校验数据
    this.$refs["form"].validate(valid => {
        if (valid) {
            // 表单数据包userId，那么就是修改
            if (this.form.userId !== undefined) {
                updateUser(this.form).then(response => {
                    this.$modal.msgSuccess("修改成功");
                    this.open = false;
                    this.getList();
                });
            } else { // 新增
                addUser(this.form).then(response => {
                    this.$modal.msgSuccess("新增成功");
                    this.open = false;
                    this.getList();
                });
            }
        }
    });
},

export function addUser(data) {
    return request({
        url: '/system/user',
        method: 'post',
        data: data
    })
}
```

## 后端实现

### 1、点击新增按钮，查询所有岗位数据和角色数据的后台代码

```
/**
 * 根据用户编号获取详细信息
 * /: 点击新增按钮，查询所有岗位数据、角色数据
 * /userId: 点击修改按钮，查询所有岗位数据、角色数据以及修改用户的角色和岗位
 */
@PreAuthorize("@ss.hasPermi('system:user:query')")
@GetMapping(value = { "/",("/{userId}" })
public AjaxResult getInfo(@PathVariable(value = "userId", required = false) Long
userId) {
```

```

userService.checkUserDataScope(userId); // 检查id是否正确
AjaxResult ajax = AjaxResult.success();
List<SysRole> roles = roleService.selectRoleAll(); // 查询所有的角色
// 如果传递进来的userId是管理员，那么就把所有的角色数据都放在结果集里面；如果不是管理员，
把管理员角色数据去掉；
ajax.put("roles", SysUser.isAdmin(userId) ? roles : roles.stream().filter(r
-> !r.isAdmin()).collect(Collectors.toList()));
// 查询所有的岗位，并放到结果集里面
ajax.put("posts", postService.selectPostAll());
// 修改
if (StringUtils.isNotBlank(userId)) {
    SysUser sysUser = userService.selectUserById(userId);
    ajax.put(AjaxResult.DATA_TAG, sysUser);
    ajax.put("postIds", postService.selectPostListByUserId(userId));
    ajax.put("roleIds",
sysUser.getRoles().stream().map(SysRole::getRoleId).collect(Collectors.toList()
));
}
return ajax;
}

```

2、点击新增弹窗的提交按钮，把数据存储到数据库中

```

/**
 * 新增用户
 */
@PreAuthorize("@ss.hasPermi('system:user:add')")
@Log(title = "用户管理", businessType = BusinessType.INSERT)
@PostMapping
public AjaxResult add(@Validated @RequestBody SysUser user) {
    // 检查数据的唯一性：用户名、手机号、邮箱
    if (!userService.checkUserNameUnique(user)) {
        return error("新增用户'" + user.getUserName() + "'失败，登录账号已存在");
    } else if (StringUtils.isNotEmpty(user.getPhonenumber()) &&
!userService.checkPhoneUnique(user)) {
        return error("新增用户'" + user.getUserName() + "'失败，手机号码已存在");
    } else if (StringUtils.isNotEmpty(user.getEmail()) &&
!userService.checkEmailUnique(user)) {
        return error("新增用户'" + user.getUserName() + "'失败，邮箱账号已存在");
    }
    user.setCreateBy(getUsername()); // 设置谁创建的
    user.setPassword(SecurityUtils.encryptPassword(user.getPassword())); // 密码加
    密
    return toAjax(userService.insertUser(user));
}

```

## 用户修改

整体思路和用户新增类似；



# 前端实现

## 1、当点击修改按钮的时候，执行handleUpdate

- 显示对话框Dialog
- 预加载数据（部门列表数据、岗位列表数据、角色数据），当前修改的用户的数据

```
/** 修改按钮操作 */
handleUpdate(row) {
  this.reset();
  // row.userId 点击当前行后面的修改链接
  // this.ids 勾选复选框，点击修改按钮
  const userId = row.userId || this.ids;
  // 查询所有的岗位和角色数据，并查询当前用户所拥有的角色和岗位
  getUser(userId).then(response => {
    this.form = response.data; // 把查询得到的用户数据给data的form变量，做数据回显
    this.postOptions = response.posts;
    this.roleOptions = response.roles;
    this.$set(this.form, "postIds", response.postIds);
    this.$set(this.form, "roleIds", response.roleIds);
    this.open = true;
    this.title = "修改用户";
    this.form.password = "";
  });
},
```

## 2、当我们点击对话框的提交按钮的时候，执行submitForm

```
/** 提交按钮 */
submitForm: function() {
  // 校验数据
  this.$refs["form"].validate(valid => {
    if (valid) {
      // 表单数据包userId，那么就是修改
      if (this.form.userId !== undefined) {
        updateUser(this.form).then(response => {
          this.$modal.msgSuccess("修改成功");
          this.open = false;
          this.getList();
        });
      } else { // 新增
        addUser(this.form).then(response => {
          this.$modal.msgSuccess("新增成功");
          this.open = false;
          this.getList();
        });
      }
    }
  });
},

// 修改用户
export function updateUser(data) {
  return request({
    url: '/system/user',
```

```

        method: 'put',
        data: data
    })
}

```

## 后端实现

1、点击修改按钮，查询所有岗位数据、角色数据以及当前要修改用户的信息的后台代码

```

@PreAuthorize("@ss.hasPermi('system:user:query')")
@GetMapping(value = { "/",("/{userId}" })
public AjaxResult getInfo(@PathVariable(value = "userId", required = false) Long
userId) {
    userService.checkUserDataScope(userId); // 检查id是否正确
    AjaxResult ajax = AjaxResult.success();
    List<SysRole> roles = roleService.selectRoleAll(); // 查询所有的角色
    // 如果传递进来的userId是管理员，那么就把所有的角色数据都放在结果集里面；如果不是管理员，
    把管理员角色数据去掉；
    ajax.put("roles", SysUser.isAdmin(userId) ? roles : roles.stream().filter(r
-> !r.isAdmin()).collect(Collectors.toList()));
    // 查询所有的岗位，并放到结果集里面
    ajax.put("posts", postService.selectPostAll());
    // 修改
    if (StringUtils.isNotBlank(userId)) {
        SysUser sysUser = userService.selectUserById(userId); // 根据id查询得到用户
        数据
        ajax.put(AjaxResult.DATA_TAG, sysUser);
        ajax.put("postIds", postService.selectPostListByUserId(userId)); // 当前用
        户的岗位id
        ajax.put("roleIds",
        sysUser.getRoles().stream().map(SysRole::getRoleId).collect(Collectors.toList()
        ); // 当前用户的角色id
    }
    return ajax;
}

```

2、点击修改弹窗的提交按钮，把更新数据库中的数据

- Controller层

```

/**
 * 修改用户
 */
@PreAuthorize("@ss.hasPermi('system:user:edit')")
@Log(title = "用户管理", businessType = BusinessType.UPDATE)
@PutMapping
public AjaxResult edit(@Validated @RequestBody SysUser user)
{
    userService.checkUserAllowed(user);
    userService.checkUserDataScope(user.getUserId());
    if (!userService.checkUserNameUnique(user)) {
        return error("修改用户'" + user.getUserName() + "'失败，登录账号已存在");
    } else if (StringUtils.isNotEmpty(user.getPhonenumber()) &&
    !userService.checkPhoneUnique(user)) {

```

```

        return error("修改用户'" + user.getUserName() + "'失败，手机号码已存在");
    } else if (StringUtils.isNotEmpty(user.getEmail()) &&
!userService.checkEmailUnique(user)) {
        return error("修改用户'" + user.getUserName() + "'失败，邮箱账号已存在");
    }
    user.setUpdateBy(getUsername()); // 更新这条数据的人
    return toAjax(userService.updateUser(user));
}

```

- Service层：关于用户的角色和岗位，是全部先删除，再新增；

```

/**
 * 修改保存用户信息
 *
 * @param user 用户信息
 * @return 结果
 */
@Override
@Transactional
public int updateUser(SysUser user)
{
    Long userId = user.getUserId();
    // 删除用户与角色关联
    userRoleMapper.deleteUserRoleByUserId(userId);
    // 新增用户与角色管理
    insertUserRole(user);
    // 删除用户与岗位关联
    userPostMapper.deleteUserPostByUserId(userId);
    // 新增用户与岗位管理
    insertUserPost(user);
    return userMapper.updateUser(user);
}

```

## 用户删除

删除分为单条数据删除和批量删除；

## 前端实现

点击每行数据后面删除链接，是单条删除；勾选复选框，点击删除按钮是批量删除；但是执行的都是 handleDelete

```

/** 删除按钮操作 */
handleDelete(row) {
    const userIds = row.userId || this.ids;
    // 弹出提示
    this.$modal.confirm('是否确认删除用户编号为"' + userIds + '"的数据项? ').then(function() {
        return delUser(userIds); // 删除操作
    }).then(() => {
        this.getList();
        this.$modal.msgSuccess("删除成功");
    }).catch(() => {});
},

```

```
// 删除用户
export function delUser(userId) {
  return request({
    url: '/system/user/' + userId,
    method: 'delete'
  })
}
```

## 后端实现

### 1、Controller层实现

```
/**
 * 删除用户
 */
@PreAuthorize("@ss.hasPermi('system:user:remove')")
@Log(title = "用户管理", businessType = BusinessType.DELETE)
@DeleteMapping("/{userIds}")
public AjaxResult remove(@PathVariable Long[] userIds) {
  // 判断当前登录的用户id是否存在删除列表中
  if (ArrayUtils.contains(userIds, getUserId())) { // getUserId() 获取当前登录的用户id
    return error("当前用户不能删除");
  }
  return toAjax(userService.deleteUserByIds(userIds));
}
```

### 2、Service层实现

```
/**
 * 批量删除用户信息
 *
 * @param userIds 需要删除的用户ID
 * @return 结果
 */
@Override
@Transactional
public int deleteUserByIds(Long[] userIds) {
  for (Long userId : userIds) {
    checkUserAllowed(new SysUser(userId)); // 超级管理员不能删除
    checkUserDataScope(userId); // 检查id是否存在
  }
  // 删除用户与角色关联
  userRoleMapper.deleteUserRole(userId);
  // 删除用户与岗位关联
  userPostMapper.deleteUserPost(userId);
  return userMapper.deleteUserByIds(userId); // 逻辑删除
}
```

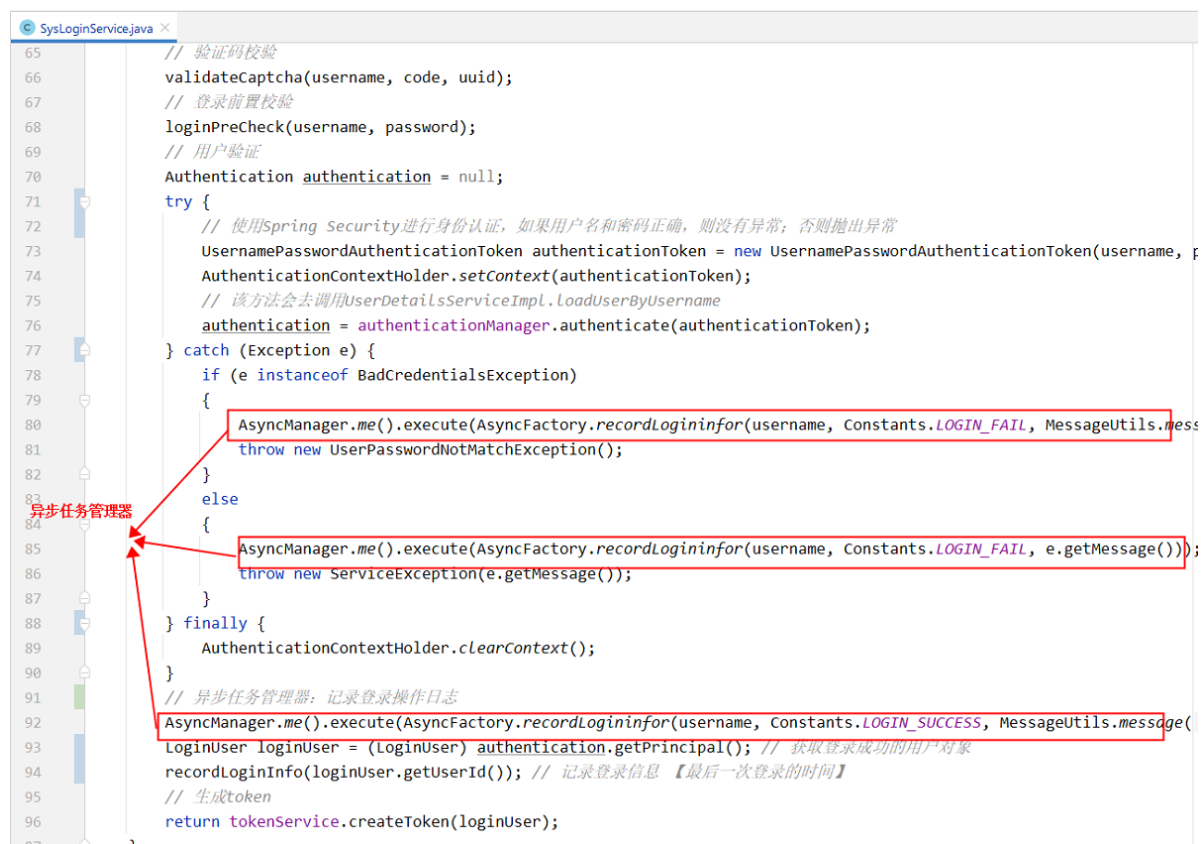
## 八、异步任务管理器

大家在面试的过程中，经常会问到线程和线程池；

**线程和线程池面试题：**什么是线程、线程的生命周期、创建线程的几种方式、什么线程池、为什么要使用线程池、线程池的生命周期、线程池的分类...

如果有人问到，是否使用过线程池？可以说使用**线程池实现异步日志记录**；

- 异步任务管理器：底层使用的就是调度线程池，用来实现异步日志记录；



AsyncManager：异步任务管理器，用来做异步日志记录；

- 调用AsyncManager.me() 得到 AsyncManager的单例对象
- 当调用execute(TimerTask task)，用来执行线程任务
  - TimerTask实现了Runnable接口，run()可以定义线程要执行的任务
  - execute()底层使用调度线程池来进行线程任务处理

```
private ScheduledExecutorService executor =
    SpringUtils.getBean("scheduledExecutorService");

public void execute(TimerTask task) {
    executor.schedule(task, OPERATE_DELAY_TIME, TimeUnit.MILLISECONDS);
}
```

这么做的目的是为了日志记录和业务解耦，日志实现统一处理；

## 九、代码生成

代码生成文档：<https://doc.ruoyi.vip/ruoyi/document/htsc.html#%E4%BB%A3%E7%A0%81%E7%94%9F%E6%88%90>

大部分项目里其实有很多代码都是重复的，几乎每个基础模块的代码都有增删改查的功能，而这些功能都是大同小异，如果这些功能都要自己去写，将会大大浪费我们的精力降低效率。所以这种重复性的代码可以使用代码生成。

# 默认配置

单应用在 `resources` 目录下的 `application.yml`，多模块 `ruoyi-generator` 中的 `resources` 目录下的 `generator.yml`，可以自己根据实际情况调整默认配置。

```
# 代码生成
gen:
  # 开发者姓名，生成到类注释上
  author: ruoyi
  # 默认生成包路径 system 需改成自己的模块名称 如 system monitor tool
  packageName: com.ruoyi.system
  # 自动去除表前缀，默认是false
  autoRemovePre: false
  # 表前缀（生成类名不会包含表前缀，多个用逗号分隔）
  tablePrefix: sys_
```

# 单表结构

1、创建表，大家在创建表的时候，最好添加一些注释

表的字段不要使用数据库关键字；

```
drop table if exists sys_student;
create table sys_student (
  student_id          int(11)          auto_increment    comment '编号',
  student_name        varchar(30)       default ''         comment '学生名称',
  student_age         int(3)            default null        comment '年龄',
  student_hobby        varchar(30)       default ''         comment '爱好（0代码 1音乐
2电影）',
  student_sex         char(1)           default '0'         comment '性别（0男 1女 2未
知）',
  student_status      char(1)           default '0'         comment '状态（0正常 1停
用）',
  student_birthday    datetime          comment '生日',
  primary key (student_id)
) engine=innodb auto_increment=1 comment = '学生信息表';
```

2、登录系统，系统工具 -> 代码生成 -> 导入 -> 选择表，点击确定

3、点击编辑，修改配置：基本信息、字段信息、生成信息

基本信息

字段信息

生成信息

\* 表名称

sys\_student

\* 表描述

学生信息表

\* 实体类名称

SysStudent

\* 作者

jimbo

备注

提交

返回

基本信息

字段信息

生成信息

序号	字段列名	字段描述	物理类型	Java类型	java属性	插入	编辑	列表	查询	查询方式	必填	显示类型	字典类型
1	student_id	编号	int	Long	studentId	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	=	<input type="checkbox"/>	文本框	请选择
2	student_name	学生名称	varchar(30)	String	studentName	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	LIKE	<input type="checkbox"/>	文本框	请选择
3	student_age	年龄	int	Integer	studentAge	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	=	<input type="checkbox"/>	文本框	请选择
4	student_hobby	爱好 (0代码 1非)	varchar(30)	String	studentHobby	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	=	<input type="checkbox"/>	文本框	请选择
5	student_sex	性别 (0男 1女)	char(1)	String	studentSex	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	=	<input type="checkbox"/>	下拉框	用户性别
6	student_status	状态 (0正常 1非)	char(1)	String	studentStatus	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	=	<input type="checkbox"/>	单选框	系统开关
7	student_birthday	生日	datetime	Date	studentBirthday	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	=	<input type="checkbox"/>	日期控件	请选择

可以修改

提交

返回

查询就是页面顶部的搜索条件

下拉框、单选框、复选框都需要字典

基本信息

字段信息

生成信息

\* 生成模板

单表 (增删改查)

前端类型

Vue2 Element UI 模版

\* 生成包路径

com.ruoyi.system

\* 生成模块名

system

\* 生成业务名

student

\* 生成功能名

学生管理

生成代码方式

☒ zip压缩包

☐ 自定义路径

上级菜单

系统管理

提交

返回

3、生成代码，下载到本地

4、解压代码

- main: Java代码 (controller、service、mapper、domain...)
- vue: 前端代码
- sql: 菜单sql

5、把代码合并到项目中

- 前端合并: 复制api、views, 在HBuilderX中粘贴到src;
- 后端合并: 复制java、resources, 在IDEA中粘贴到main;
- sql导入

6、启动前端和后端

自动生成的页面, 无法自动生成单选框、复选框、下拉框;

推荐: 先全部使用文本框, 然后自己改

7、在菜单管理中, 修改新增菜单的图标、顺序

自定义字典

1. 字典管理 -> 新增

添加字典类型

×

\* 字典名称

爱好分组

\* 字典类型

sys\_hobby

状态

正常

停用

备注

请输入内容

确定

取消

2. 新增字典数据：点击字典类型链接，进入字典数据页面，新增即可

字典名称 爱好分组

字典标签 请输入字典标签

状态 数据状态

Q 新增

Q 重置

+ 新增

✎ 修改

🗑 删除

📄 导出

✕ 关闭

<input type="checkbox"/>	字典编码	字典标签	字典键值	字典排序	状态	备注	创建时间	操作
<input type="checkbox"/>	100	足球	0	0	正常		2024-02-20 23:58:46	<a href="#">✎ 修改</a> <a href="#">🗑 删除</a>
<input type="checkbox"/>	101	篮球	1	1	正常		2024-02-20 23:59:01	<a href="#">✎ 修改</a> <a href="#">🗑 删除</a>
<input type="checkbox"/>	102	编码	2	2	正常		2024-02-20 23:59:18	<a href="#">✎ 修改</a> <a href="#">🗑 删除</a>

共 3 条

10条/页

< 1 >

前往 1 页

# 树表结构

## 新建数据库表结构（树表）

```
drop table if exists sys_product;
create table sys_product (
  product_id      bigint(20)      not null auto_increment      comment '产品id',
  parent_id       bigint(20)      default 0                    comment '父产品id',
  product_name    varchar(30)     default ''                    comment '产品名称',
  order_num       int(4)           default 0                    comment '显示顺序',
  status          char(1)          default '0'                  comment '产品状态(0正常 1停用)',
  primary key (product_id)
) engine=innodb auto_increment=1 comment = '产品表';
```

基本信息和字段信息设置，和单表的类似的，区别点：生成信息的设置



基本信息

字段信息

生成信息

\* 生成模板

树表 (增删改查)

生成包路径

com.ruoyi.system

\* 生成业务名

product

生成代码方式

zip压缩包

自定义路径

前端类型

Vue2 Element UI 模板

\* 生成模块名

system

\* 生成功能名

产品管理

上级菜单

系统管理

×

其他信息

树编码字段

product\_id: 产品id

树父编码字段

parent\_id: 父产品id

树名称字段

product\_name: 产品名称

提交

返回

# 主子表结构

一般不用

## 新建数据库表结构 (主子表)

```
-- -----
-- 客户表
-- -----

drop table if exists sys_customer;
create table sys_customer (
    customer_id          bigint(20)          not null auto_increment    comment '客户id',
    customer_name        varchar(30)          default ''                comment '客户姓名',
    phonenumber          varchar(11)          default ''                comment '手机号码',
    sex                  varchar(20)          default null              comment '客户性别',
    birthday             datetime              comment '客户生日',
    remark               varchar(500)         default null              comment '客户描述',
    primary key (customer_id)
) engine=innodb auto_increment=1 comment = '客户表';

-- -----
-- 商品表
-- -----

drop table if exists sys_goods;
create table sys_goods (
    goods_id             bigint(20)          not null auto_increment    comment '商品id',
    customer_id          bigint(20)          not null                  comment '客户id',
    name                 varchar(30)          default ''                comment '商品名称',
    weight               int(5)               default null              comment '商品重量',
    price                decimal(6,2)        default null              comment '商品价格',
    date                datetime              comment '商品时间',
    type                 char(1)              default null              comment '商品种类',
    primary key (goods_id)
```

```
) engine=innodb auto_increment=1 comment = '商品表';
```

基本信息

字段信息

生成信息

\* 生成模板

主子表 (增删改查)

\* 生成包路径

com.ruoyi.system

\* 生成业务名

customer

生成代码方式

☒ zip压缩包 ☐ 自定义路径

前端类型

Vue2 Element UI 模版

\* 生成模块名

system

\* 生成功能名

客户管理

上级菜单

业务管理

关联信息

关联子表的表名

sys\_goods: 商品表

子表关联的外键名

customer\_id: 客户id

提交

返回

## 十、基于若依开发

1、常规的操作（CRUD）：照葫芦画瓢

2、非常规操作，参照官方文档，比如导入

- <https://doc.ruoyi.vip/ruoyi-vue/document/htsc.html#%E5%AF%BC%E5%85%A5%E5%AE%9%E7%8E%B0%E6%B5%81%E7%A8%8B>

后端手册：<https://doc.ruoyi.vip/ruoyi-vue/document/htsc.html#%E5%90%8E%E5%8F%B0%E6%89%8B%E5%86%8C>