# 豆豆[项目介绍](http://doc.ruoyi.vip/" \l "/?id=%e9%a1%b9%e7%9b%ae%e4%bb%8b%e7%bb%8d)

**DouDou**是一个后台管理系统，基于经典技术组合（Spring Boot、Apache Shiro、MyBatis、Thymeleaf）主要目的让开发者注重专注业务，降低技术难度，从而节省人力成本，缩短项目周期，提高软件安全质量。  
账号密码：admin/admin123

## 系统需求

* JDK >= 1.8
* MySQL >= 5.5
* Maven >= 3.0

## [必要配置](http://doc.ruoyi.vip/" \l "/standard/hjbs?id=%e5%bf%85%e8%a6%81%e9%85%8d%e7%bd%ae)

1. 修改数据库连接  
   编辑resources目录下的application-druid.yml  
   url: 服务器地址  
   username: 账号  
   password: 密码
2. 开发环境配置  
   编辑resources目录下的application.yml  
   port: 端口  
   context-path: 部署路径

## [部署系统](http://doc.ruoyi.vip/" \l "/standard/hjbs?id=%e9%83%a8%e7%bd%b2%e7%b3%bb%e7%bb%9f)

bin/package.bat 在项目的目录下执行  
然后会在项目下生成 target文件夹包含 war 或jar （多模块生成在DouDou-admin）

1、jar部署方式  
使用命令行执行：java –jar DouDou.jar 或者执行脚本：bin/run.bat

2、war部署方式  
pom.xml packaging修改为war 放入tomcat服务器webapps

SpringBoot去除内嵌tomcat

<!-- 多模块排除内置tomcat -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- 单应用排除内置tomcat -->

<exclusions>

<exclusion>

<artifactId>spring-boot-starter-tomcat</artifactId>

<groupId>org.springframework.boot</groupId>

</exclusion>

</exclusions>

## [常见问题](http://doc.ruoyi.vip/" \l "/standard/hjbs?id=%e5%b8%b8%e8%a7%81%e9%97%ae%e9%a2%98)

1. 如果使用Mac 需要修改application-druid.yml文件路径profile
2. 如果使用Linux 提示表不存在，设置大小写敏感配置在/etc/my.cnf 添加lower\_case\_table\_names=1，重启MYSQL服务
3. 如果提示当前权限不足，无法写入文件请检查profile是否可读可写，或者无法访问此目录

## [文件结构](http://doc.ruoyi.vip/" \l "/standard/xmjs?id=%e6%96%87%e4%bb%b6%e7%bb%93%e6%9e%84)

com.aaa

├── 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

│ └── json // JSON数据处理

│ └── utils // 通用类处理

│ └── xss // XSS过滤处理

├── framework // 框架核心

│ └── aspectj // 注解实现

│ └── config // 系统配置

│ └── datasource // 数据权限

│ └── manager // 异步处理

│ └── shiro // 权限控制

│ └── util // 通用工具

│ └── web // 前端控制

├── DouDou-generator // 代码生成（可移除）

├── DouDou-quartz // 定时任务（可移除）

├── DouDou-system // 系统代码

├── DouDou-admin // 后台服务

├── DouDou-xxxxxx // 其他模块

## [配置文件](http://doc.ruoyi.vip/" \l "/standard/xmjs?id=%e9%85%8d%e7%bd%ae%e6%96%87%e4%bb%b6)

通用配置 application.yml

# 项目相关配置

DouDou:

# 名称

name: DouDou

# 版本

version: 3.2.0

# 版权年份

copyrightYear: 2019

# 文件上传

profile: D:/profile/

# 获取ip地址开关

addressEnabled: true

# 开发环境配置

server:

# 服务端口

port: 80

servlet:

# 项目contextPath

context-path: /

tomcat:

# tomcat的URI编码

uri-encoding: UTF-8

# tomcat最大线程数，默认为200

max-threads: 800

# Tomcat启动初始化的线程数，默认值25

min-spare-threads: 30

# 日志配置

logging:

level:

com.doudou: debug

org.springframework: WARN

org.spring.springboot.dao: debug

# 用户配置

user:

password:

# 密码错误{maxRetryCount}次锁定10分钟

maxRetryCount: 5

# Spring配置

spring:

# 模板引擎

thymeleaf:

mode: HTML

encoding: utf-8

# 禁用缓存

cache: false

# 资源信息

messages:

# 国际化资源文件路径

basename: i18n/messages

jackson:

time-zone: GMT+8

date-format: yyyy-MM-dd HH:mm:ss

profiles:

active: druid

# 文件上传

servlet:

multipart:

max-file-size: 30MB

max-request-size: 30MB

# 服务模块

devtools:

restart:

# 热部署开关

enabled: true

# MyBatis

mybatis:

# 搜索指定包别名

typeAliasesPackage: com.doudou

# 配置mapper的扫描，找到所有的mapper.xml映射文件

mapperLocations: classpath\*:mapper/\*\*/\*Mapper.xml

# 加载全局的配置文件

configLocation: classpath:mapper/mybatis-config.xml

# PageHelper分页插件

pagehelper:

helperDialect: mysql

reasonable: true

supportMethodsArguments: true

params: count=countSql

# Shiro

shiro:

user:

# 登录地址

loginUrl: /login

# 权限认证失败地址

unauthorizedUrl: /unauth

# 首页地址

indexUrl: /index

# 验证码开关

captchaEnabled: true

# 验证码类型 math 数组计算 char 字符

captchaType: math

cookie:

# 设置Cookie的域名 默认空，即当前访问的域名

domain:

# 设置cookie的有效访问路径

path: /

# 设置HttpOnly属性

httpOnly: true

# 设置Cookie的过期时间，天为单位

maxAge: 30

session:

# Session超时时间（默认30分钟）

expireTime: 30

# 同步session到数据库的周期（默认1分钟）

dbSyncPeriod: 1

# 相隔多久检查一次session的有效性，默认就是10分钟

validationInterval: 10

# 防止XSS攻击

xss:

# 过滤开关

enabled: true

# 排除链接（多个用逗号分隔）

excludes: /system/notice/\*

# 匹配链接

urlPatterns: /system/\*,/monitor/\*,/tool/\*

数据源配置 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://localhost:3306/ry?useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&useSSL=true&serverTimezone=GMT%2B8

username: root

password: password

# 从库数据源

slave:

# 从数据源开关/默认关闭

enabled: false

url:

username:

password:

# 初始连接数

initialSize: 5

# 最小连接池数量

minIdle: 10

# 最大连接池数量

maxActive: 20

# 配置获取连接等待超时的时间

maxWait: 60000

# 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒

timeBetweenEvictionRunsMillis: 60000

# 配置一个连接在池中最小生存的时间，单位是毫秒

minEvictableIdleTimeMillis: 300000

# 配置一个连接在池中最大生存的时间，单位是毫秒

maxEvictableIdleTimeMillis: 900000

# 配置检测连接是否有效

validationQuery: SELECT 1 FROM DUAL

testWhileIdle: true

testOnBorrow: false

testOnReturn: false

webStatFilter:

enabled: true

statViewServlet:

enabled: true

# 设置白名单，不填则允许所有访问

allow:

url-pattern: /monitor/druid/\*

filter:

stat:

enabled: true

# 慢SQL记录

log-slow-sql: true

slow-sql-millis: 1000

merge-sql: true

wall:

config:

multi-statement-allow: true

代码生成配置 generator.yml

# 代码生成

gen:

# 作者

author: DouDou

# 默认生成包路径 system 需改成自己的模块名称 如 system monitor tool

packageName: com.aaa.system

# 自动去除表前缀，默认是true

autoRemovePre: true

# 表前缀(类名不会包含表前缀)

tablePrefix: sys\_

# [核心技术](http://doc.ruoyi.vip/" \l "/standard/xmjs?id=%e6%a0%b8%e5%bf%83%e6%8a%80%e6%9c%af)

## [SpringBoot框架](http://doc.ruoyi.vip/" \l "/standard/xmjs?id=springboot%e6%a1%86%e6%9e%b6)

1、介绍  
Spring Boot是一款开箱即用框架，提供各种默认配置来简化项目配置。让我们的Spring应用变的更轻量化、更快的入门。 在主程序执行main函数就可以运行。你也可以打包你的应用为jar并通过使用java -jar来运行你的Web应用。它遵循"约定优先于配置"的原则， 使用SpringBoot只需很少的配置，大部分的时候直接使用默认的配置即可。可以与Spring Cloud的微服务无缝结合。  
Spring Boot2.0 环境要求必须是jdk8或以上版本，Tomcat8或以上版本

2、优点

* 使编码变得简单： 推荐使用注解。
* 使配置变得简单： 自动配置、快速构建项目、快速集成新技术能力 没有冗余代码生成和XML配置的要求
* 使部署变得简单： 内嵌Tomcat、Jetty、Undertow等web容器，无需以war包形式部署
* 使监控变得简单： 自带项目监控

## [Shiro安全控制](http://doc.ruoyi.vip/" \l "/standard/xmjs?id=shiro%e5%ae%89%e5%85%a8%e6%8e%a7%e5%88%b6)

1、介绍  
Apache Shiro是Java的一个安全框架。Shiro可以帮助我们完成：认证、授权、加密、会话管理、与Web集成、缓存等。其不仅可以用在 JavaSE环境，也可以用在 JavaEE 环境。

2、优点

* 易于理解的 Java Security API
* 简单的身份认证，支持多种数据源
* 对角色的简单的授权，支持细粒度的授权
* 不跟任何的框架或者容器捆绑，可以独立运行

3、特性  
Authentication身份认证/登录，验证用户是不是拥有相应的身份  
Authorization授权，即验证权限，验证某个已认证的用户是否拥有某个权限，即判断用户是否能做事情 SessionManagement会话管理，即用户登录后就是一次会话，在没有退出之前，它的所有信息都在会话中  
Cryptography加密，保护数据的安全性，如密码加密存储到数据库，而不是明文存储  
Caching缓存，比如用户登录后，其用户信息，拥有的角色/权限不必每次去查，提高效率  
ConcurrencyShiro支持多线程应用的并发验证，即如在一个线程中开启另一个线程，能把权限自动传播过去  
Testing提供测试支持  
RunAs允许一个用户假装为另一个用户（如果他们允许）的身份进行访问  
RememberMe记住我，这是非常常见的功能，即一次登录后，下次再来的话不用登录了

4、架构  
Subject主体，代表了当前的“用户”，这个用户不一定是一个具体的人，与当前应用交互的任何东西都是Subject，如网络爬虫， 机器人等；即一个抽象概念；所有Subject都绑定到SercurityManager，与Subject的所有交互都会委托给SecurityManager；可以把Subject认为是一个门面；SecurityManager才是实际的执行者  
SecurityManage安全管理器；即所有与安全有关的操作都会与SecurityManager交互；且它管理着所有Subject； 可以看出它是Shiro的核心，它负责与后边介绍的其他组件进行交互  
Realm域，Shiro从Realm获取安全数据（如用户，角色，权限），就是说SecurityManager要验证用户身份， 那么它需要从Realm获取相应的用户进行比较以确定用户身份是否合法；也需要从Realm得到用户相应的角色/权限进行验证用户是否能进行操作；可以有1个或多个Realm，我们一般在应用中都需要实现自己的Realm  
SessionManager如果写过Servlet就应该知道Session的概念，Session需要有人去管理它的生命周期，这个组件就是SessionManager  
SessionDAO大家都用过，数据库访问对象，用于会话的CRUD，比如我们想把Session保存到数据库，那么可以实现自己的SessionDAO，也可以写入缓存，以提高性能  
CacheManager缓存控制器，来管理如用户，角色，权限等的缓存的；因为这些数据基本上很少去改变，放到缓存中后可以提高访问的性能

应用代码通过Subject来进行认证和授权，而Subject又委托给SecurityManager； 我们需要给Shrio的SecurityManager注入Realm，从而让SecurityManager能得到合法的用户及其权限进行判断，Shiro不提供维护用户/权限，而是通过Realm让开发人员自己注入。

Shiro不会去维护用户，维护权限；这些需要自己去设计/提供；然后通过响应的接口注入给Shiro即可

## [Thymeleaf模板](http://doc.ruoyi.vip/" \l "/standard/xmjs?id=thymeleaf%e6%a8%a1%e6%9d%bf)

1、介绍  
Thymeleaf是一个用于Web和独立Java环境的模板引擎，能够处理HTML、XML、JavaScript、CSS甚至纯文本。能轻易的与Spring MVC等Web框架进行集成作为Web应用的模板引擎。 与其它模板引擎（比如FreeMaker）相比，Thymeleaf最大的特点是能够直接在浏览器中打开并正确显示模板页面，而不需要启动整个Web应用（更加方便前后端分离，比如方便类似VUE前端设计页面），抛弃JSP吧。 Thymeleaf 3.0是一个完全彻底重构的模板引擎，极大的减少内存占用和提升性能和并发性，避免v2.1版因大量的输出标记的集合产生的资源占用。 Thymeleaf 3.0放弃了大多数面向DOM的处理机制，变成了一个基于事件的模板处理器，它通过处理模板标记或文本并立即生成其输出，甚至在新事件之前响应模板解析器/缓存事件。Thymeleaf是Spring Boot官方的推荐使用模板。

2、优点

* 国际化支持非常简单
* 语法简单，功能强大。内置大量常用功能，使用非常方便
* 可以很好的和Spring集成
* 静态html嵌入标签属性，浏览器可以直接打开模板文件，便于前后端联调
* Spring Boot 官方推荐，用户群广

# 后台手册

## 分页实现

前端基于Bootstrap的轻量级表格插件 BootstrapTable

后端分页组件使用Mybatis分页插件 PageHelper

分页实现流程

1、前端调用封装好的方法$.table.init，传入后台url。

var options = {

url: prefix + "/list",

columns: [{

field: 'id',

title: '主键'

},

{

field: 'name',

title: '名称'

}]

};

$.table.init(options);

2、自定义查询条件参数（特殊情况提前设置查询条件下使用）

var options = {

url: prefix + "/list",

queryParams: queryParams,

columns: [{

field: 'id',

title: '主键'

},

{

field: 'name',

title: '名称'

}]

};

$.table.init(options);

function queryParams(params) {

var search = $.table.queryParams(params);

search.userName = $("#userName").val();

return search;

}

3、后台实现查询逻辑，调用startPage()方法即可自动完成服务端分页。

@PostMapping("/list")

@ResponseBody

public TableDataInfo list(User user)

{

startPage(); // 此方法配合前端完成自动分页

List<User> list = userService.selectUserList(user);

return getDataTable(list);

}

常见坑点1：selectPostById莫名其妙的分页。例如下面这段代码

startPage();

List<User> list;

if(user != null){

list = userService.selectUserList(user);

} else {

list = new ArrayList<User>();

}

Post post = postService.selectPostById(1L);

return getDataTable(list);

原因分析：这种情况下由于user存在null的情况，就会导致PageHelper生产了一个分页参数，但是没有被消费，这个参数就会一直保留在这个线程上。 当这个线程再次被使用时，就可能导致不该分页的方法去消费这个分页参数，这就产生了莫名其妙的分页。

上面这个代码，应该写成下面这个样子才能保证安全。

List<User> list;

if(user != null){

startPage();

list = userService.selectUserList(user);

} else {

list = new ArrayList<User>();

}

Post post = postService.selectPostById(1L);

return getDataTable(list);

常见坑点2：添加了startPage方法。也没有正常分页。例如下面这段代码

startPage();

Post post = postService.selectPostById(1L);

List<User> list = userService.selectUserList(user);

return getDataTable(list);

原因分析：只对该语句以后的第一个查询（Select）语句得到的数据进行分页。

上面这个代码，应该写成下面这个样子才能正常分页。

Post post = postService.selectPostById(1L);

startPage();

List<User> list = userService.selectUserList(user);

return getDataTable(list);

注意：如果改为其他数据库需修改配置application.yml helperDialect=你的数据库

导入导出

在实际开发中经常需要使用导入导出功能来加快数据的操作。在项目中可以使用注解来完成此项功能。 在需要被导入导出的实体类属性添加@Excel注解，目前支持参数如下：

参数 类型 默认值 描述

name String 空 导出到Excel中的名字

dateFormat String 空 日期格式, 如: yyyy-MM-dd

readConverterExp String 空 读取内容转表达式 (如: 0=男,1=女,2=未知)

height String 14 导出时在excel中每个列的高度 单位为字符

width String 16 导出时在excel中每个列的宽 单位为字符

suffix String 空 文字后缀,如% 90 变成90%

defaultValue String 空 当值为空时,字段的默认值

prompt String 空 提示信息

combo String Null 设置只能选择不能输入的列内容

targetAttr String 空 另一个类中的属性名称,支持多级获取,以小数点隔开

type Enum Type.ALL 字段类型（0：导出导入；1：仅导出；2：仅导入）

## 导出实现流程

1、前端调用封装好的方法$.table.init，传入后台exportUrl

var options = {

exportUrl: prefix + "/export",

columns: [{

field: 'id',

title: '主键'

},

{

field: 'name',

title: '名称'

}]

};

$.table.init(options);

2、添加导出按钮事件

<a class="btn btn-warning" onclick="$.table.exportExcel()">

<i class="fa fa-download"></i> 导出

</a>

3、在实体变量上添加@Excel注解

@Excel(name = "用户序号", prompt = "用户编号")

private Long userId;

@Excel(name = "用户名称")

private String userName;

@Excel(name = "用户性别", readConverterExp = "0=男,1=女,2=未知")

private String sex;

@Excel(name = "最后登陆时间", width = 30, dateFormat = "yyyy-MM-dd HH:mm:ss")

private Date loginDate;

4、在Controller添加导出方法

@PostMapping("/export")

@ResponseBody

public AjaxResult export(User user)

{

List<User> list = userService.selectUserList(user);

ExcelUtil<User> util = new ExcelUtil<User>(User.class);

return util.exportExcel(list, "用户数据");

}

## 导入实现流程

1、前端调用封装好的方法$.table.init，传入后台importUrl。

var options = {

importUrl: prefix + "/importData",

columns: [{

field: 'id',

title: '主键'

},

{

field: 'name',

title: '名称'

}]

};

$.table.init(options);

2、添加导入按钮事件

<a class="btn btn-info" onclick="$.table.importExcel()">

<i class="fa fa-upload"></i> 导入

</a>

3、添加导入前端代码，form默认id为importForm，也可指定importExcel(id)

<form id="importForm" enctype="multipart/form-data" class="mt20 mb10" style="display: none;">

<div class="col-xs-offset-1">

<input type="file" id="file" name="file"/>

<div class="mt10 pt5">

<input type="checkbox" id="updateSupport" name="updateSupport" title="如果登录账户已经存在，更新这条数据。"> 是否更新已经存在的用户数据

&nbsp; <a onclick="$.table.importTemplate()" class="btn btn-default btn-xs"><i class="fa fa-file-excel-o"></i> 下载模板</a>

</div>

<font color="red" class="pull-left mt10">

提示：仅允许导入“xls”或“xlsx”格式文件！

</font>

</div>

</form>

4、在实体变量上添加@Excel注解，默认为导出导入，也可以单独设置仅导入Type.IMPORT

@Excel(name = "用户序号")

private Long id;

@Excel(name = "部门编号", type = Type.IMPORT)

private Long deptId;

@Excel(name = "用户名称")

private String userName;

5、在Controller添加导入方法，updateSupport属性为是否存在则覆盖（可选）

@PostMapping("/importData")

@ResponseBody

public AjaxResult importData(MultipartFile file, boolean updateSupport) throws Exception

{

ExcelUtil<SysUser> util = new ExcelUtil<SysUser>(SysUser.class);

List<SysUser> userList = util.importExcel(file.getInputStream());

String operName = ShiroUtils.getSysUser().getLoginName();

String message = userService.importUser(userList, updateSupport, operName);

return AjaxResult.success(message);

}

## 上传下载

首先创建一张上传文件的表，例如：

drop table if exists sys\_file\_info;

create table sys\_file\_info (

file\_id int(11) not null auto\_increment comment '文件id',

file\_name varchar(50) default '' comment '文件名称',

file\_path varchar(255) default '' comment '文件路径',

primary key (file\_id)

) engine=innodb auto\_increment=1 default charset=utf8 comment = '文件信息表';

上传实现流程

1、代码生成sys\_file\_info表相关代码并覆盖到对应目录。

2、参考示例修改代码。

<input id="filePath" name="filePath" class="form-control" type="file">

function submitHandler() {

if ($.validate.form()) {

uploadFile();

}

}

function uploadFile() {

var formData = new FormData();

if ($('#filePath')[0].files[0] == null) {

$.modal.alertWarning("请先选择文件路径");

return false;

}

formData.append('fileName', $("#fileName").val());

formData.append('file', $('#filePath')[0].files[0]);

$.ajax({

url: prefix + "/add",

type: 'post',

cache: false,

data: formData,

processData: false,

contentType: false,

dataType: "json",

success: function(result) {

$.operate.successCallback(result);

}

});

}

3、在FileInfoController添加对应上传方法

@PostMapping("/add")

@ResponseBody

public AjaxResult addSave(@RequestParam("file") MultipartFile file, FileInfo fileInfo) throws IOException

{

// 上传文件路径

String filePath = DouDouConfig.getUploadPath();

// 上传并返回新文件名称

String fileName = FileUploadUtils.upload(filePath, file);

fileInfo.setFilePath(fileName);

return toAjax(fileInfoService.insertFileInfo(fileInfo));

}

4、上传成功后需要预览可以对该属性格式化处理

{

field : 'filePath',

title: '文件预览',

formatter: function(value, row, index) {

return '<a href="javascript:downloadFile(' + row.fileId + ')"><img style="width:30;height:30px;" src="/profile/upload/' + row.filePath + '"/></a>';

}

},

如需对文件格式控制，设置application.yml中的multipart属性

# 文件上传

servlet:

multipart:

# 单个文件大小

max-file-size: 10MB

# 设置总上传的文件大小

max-request-size: 20MB

注意：如果只是单纯的上传一张图片没有其他参数可以使用通用方法 /common/upload

请求处理方法 com.doudou.web.controller.common.CommonController

## 下载实现流程

1、参考示例代码。

function downloadFile(fileId){

window.location.href = ctx + "system/fileInfo/downloadFile/" + fileId;

}

2、在Controller添加对应上传方法

@GetMapping("/downloadFile/{fileId}")

public void downloadFile(@PathVariable("fileId") Integer fileId, HttpServletResponse response,

HttpServletRequest request) throws Exception

{

FileInfo sysFile = fileInfoService.selectFileInfoById(fileId);

String filePath = sysFile.getFilePath();

String realFileName = sysFile.getFileName() + filePath.substring(filePath.indexOf("."));

String path = DouDouConfig.getUploadPath() + sysFile.getFilePath();

response.setCharacterEncoding("utf-8");

response.setContentType("multipart/form-data");

response.setHeader("Content-Disposition",

"attachment;fileName=" + FileUtils.setFileDownloadHeader(request, realFileName));

FileUtils.writeBytes(path, response.getOutputStream());

}

## 事务管理

新建的Spring Boot项目中，一般都会引用spring-boot-starter或者spring-boot-starter-web，而这两个起步依赖中都已经包含了对于spring-boot-starter-jdbc或spring-boot-starter-data-jpa的依赖。 当我们使用了这两个依赖的时候，框架会自动默认分别注入DataSourceTransactionManager或JpaTransactionManager。 所以我们不需要任何额外配置就可以用@Transactional注解进行事务的使用。

@Transactional注解只能应用到public可见度的方法上，可以被应用于接口定义和接口方法，方法会覆盖类上面声明的事务。

例如用户新增需要插入用户表、用户与岗位关联表、用户与角色关联表，如果插入成功，那么一起成功，如果中间有一条出现异常，那么回滚之前的所有操作， 这样可以防止出现脏数据，就可以使用事务让它实现回退。

做法非常简单，我们只需要在方法或类添加@Transactional注解即可。

@Transactional

public int insertUser(User user)

{

// 新增用户信息

int rows = userMapper.insertUser(user);

// 新增用户岗位关联

insertUserPost(user);

// 新增用户与角色管理

insertUserRole(user);

return rows;

}

常见坑点1：遇到非检测异常时，事务开启，也无法回滚。 例如下面这段代码，用户依旧增加成功，并没有因为后面遇到检测异常而回滚！！

@Transactional

public int insertUser(User user) throws Exception

{

// 新增用户信息

int rows = userMapper.insertUser(user);

// 新增用户岗位关联

insertUserPost(user);

// 新增用户与角色管理

insertUserRole(user);

// 模拟抛出SQLException异常

boolean flag = true;

if (flag)

{

throw new SQLException("发生异常了..");

}

return rows;

}

原因分析：因为Spring的默认的事务规则是遇到运行异常（RuntimeException）和程序错误（Error）才会回滚。如果想针对非检测异常进行事务回滚，可以在@Transactional 注解里使用 rollbackFor 属性明确指定异常。

例如下面这样，就可以正常回滚：

@Transactional(rollbackFor = Exception.class)

public int insertUser(User user) throws Exception

{

// 新增用户信息

int rows = userMapper.insertUser(user);

// 新增用户岗位关联

insertUserPost(user);

// 新增用户与角色管理

insertUserRole(user);

// 模拟抛出SQLException异常

boolean flag = true;

if (flag)

{

throw new SQLException("发生异常了..");

}

return rows;

}

常见坑点2： 在业务层捕捉异常后，发现事务不生效。 这是许多新手都会犯的一个错误，在业务层手工捕捉并处理了异常，你都把异常“吃”掉了，Spring自然不知道这里有错，更不会主动去回滚数据。

例如：下面这段代码直接导致用户新增的事务回滚没有生效。

@Transactional

public int insertUser(User user) throws Exception

{

// 新增用户信息

int rows = userMapper.insertUser(user);

// 新增用户岗位关联

insertUserPost(user);

// 新增用户与角色管理

insertUserRole(user);

// 模拟抛出SQLException异常

boolean flag = true;

if (flag)

{

try

{

// 谨慎：尽量不要在业务层捕捉异常并处理

throw new SQLException("发生异常了..");

}

catch (Exception e)

{

e.printStackTrace();

}

}

return rows;

}

推荐做法：在业务层统一抛出异常，然后在控制层统一处理。

@Transactional

public int insertUser(User user) throws Exception

{

// 新增用户信息

int rows = userMapper.insertUser(user);

// 新增用户岗位关联

insertUserPost(user);

// 新增用户与角色管理

insertUserRole(user);

// 模拟抛出SQLException异常

boolean flag = true;

if (flag)

{

throw new RuntimeException("发生异常了..");

}

return rows;

}

Transactional注解的常用属性表：

属性 说明

propagation 事务的传播行为，默认值为 REQUIRED。

isolation 事务的隔离度，默认值采用 DEFAULT

timeout 事务的超时时间，默认值为-1，不超时。如果设置了超时时间(单位秒)，那么如果超过该时间限制了但事务还没有完成，则自动回滚事务。

read-only 指定事务是否为只读事务，默认值为 false；为了忽略那些不需要事务的方法，比如读取数据，可以设置 read-only 为 true。

rollbackFor 用于指定能够触发事务回滚的异常类型，如果有多个异常类型需要指定，各类型之间可以通过逗号分隔。{xxx1.class, xxx2.class,……}

noRollbackFor 抛出 no-rollback-for 指定的异常类型，不回滚事务。{xxx1.class, xxx2.class,……}

....

事务的传播机制是指如果在开始当前事务之前，一个事务上下文已经存在，此时有若干选项可以指定一个事务性方法的执行行为。 即:在执行一个@Transactinal注解标注的方法时，开启了事务；当该方法还在执行中时，另一个人也触发了该方法；那么此时怎么算事务呢，这时就可以通过事务的传播机制来指定处理方式。

TransactionDefinition传播行为的常量：



## 异常处理

通常一个web框架中，有大量需要处理的异常。比如业务异常，权限不足等等。前端通过弹出提示信息的方式告诉用户出了什么错误。 通常情况下我们用try.....catch.... 对异常进行捕捉处理，但是在实际项目中对业务模块进行异常捕捉，会造成代码重复和繁杂， 我们希望代码中只有业务相关的操作，所有的异常我们单独设立一个类来处理它。全局异常就是对框架所有异常进行统一管理。 我们在可能发生异常的方法里throw抛给控制器。然后由全局异常处理器对异常进行统一处理。 如此，我们的Controller中的方法就可以很简洁了。

所谓全局异常处理器就是使用@ControllerAdvice注解。示例如下：

1、统一返回实体定义

package com.doudou.common.core.domain;

import java.util.HashMap;

/\*\*

\* 操作消息提醒

\*

\* @author DouDou

\*/

public class AjaxResult extends HashMap<String, Object>

{

private static final long serialVersionUID = 1L;

/\*\*

\* 返回错误消息

\*

\* @param code 错误码

\* @param msg 内容

\* @return 错误消息

\*/

public static AjaxResult error(String msg)

{

AjaxResult json = new AjaxResult();

json.put("msg", msg);

json.put("code", 500);

return json;

}

/\*\*

\* 返回成功消息

\*

\* @param msg 内容

\* @return 成功消息

\*/

public static AjaxResult success(String msg)

{

AjaxResult json = new AjaxResult();

json.put("msg", msg);

json.put("code", 0);

return json;

}

}

2、定义登录异常定义

package com.doudou.common.exception;

/\*\*

\* 登录异常

\*

\* @author DouDou

\*/

public class LoginException extends RuntimeException

{

private static final long serialVersionUID = 1L;

protected final String message;

public LoginException(String message)

{

this.message = message;

}

@Override

public String getMessage()

{

return message;

}

}

3、基于@ControllerAdvice注解的Controller层的全局异常统一处理

package com.doudou.framework.web.exception;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.bind.annotation.RestControllerAdvice;

import com.doudou.common.core.domain.AjaxResult;

import com.doudou.common.exception.LoginException;

/\*\*

\* 全局异常处理器

\*

\* @author DouDou

\*/

@RestControllerAdvice

public class GlobalExceptionHandler

{

private static final Logger log = LoggerFactory.getLogger(GlobalExceptionHandler.class);

/\*\*

\* 登录异常

\*/

@ExceptionHandler(LoginException.class)

public AjaxResult loginException(LoginException e)

{

log.error(e.getMessage(), e);

return AjaxResult.error(e.getMessage());

}

}

4、测试访问请求

@Controller

public class SysIndexController

{

/\*\*

\* 首页方法

\*/

@GetMapping("/index")

public String index(ModelMap mmap)

{

/\*\*

\* 模拟用户未登录，抛出业务逻辑异常

\*/

SysUser user = ShiroUtils.getSysUser();

if (StringUtils.isNull(user))

{

throw new LoginException("用户未登录，无法访问请求。");

}

mmap.put("user", user);

return "index";

}

}

根据上面代码含义，当我们未登录访问/index时就会发生LoginException业务逻辑异常，按照我们之前的全局异常配置以及统一返回实体实例化，访问后会出现AjaxResult格式JSON数据， 下面我们运行项目访问查看效果。

界面输出内容如下所示：

{

"msg": "用户未登录，无法访问请求。",

"code": 500

}

对于一些特殊情况，如接口需要返回json，页面请求返回html可以使用如下方法：

@ExceptionHandler(LoginException.class)

public Object loginException(HttpServletRequest request, LoginException e)

{

log.error(e.getMessage(), e);

if (ServletUtils.isAjaxRequest(request))

{

return AjaxResult.error(e.getMessage());

}

else

{

return new ModelAndView("/error/500");

}

}

若依系统的全局异常处理器GlobalExceptionHandler

注意：如果全部异常处理返回json，那么可以使用@RestControllerAdvice代替@ControllerAdvice，这样在方法上就可以不需要添加@ResponseBody。

## 系统日志

在实际开发中，对于某些关键业务，我们通常需要记录该操作的内容，一个操作调一次记录方法，每次还得去收集参数等等，会造成大量代码重复。 我们希望代码中只有业务相关的操作，在项目中使用注解来完成此项功能。

在需要被记录日志的controller方法上添加@Log注解，使用方法如下：

@Log(title = "用户管理", businessType = BusinessType.INSERT)

支持参数如下：

参数 类型 默认值 描述

title String 空 操作模块

businessType BusinessType OTHER 操作功能（OTHER其他 INSERT新增 UPDATE修改 DELETE删除 GRANT授权 EXPORT导出 IMPORT导入 FORCE强退 GENCODE生成代码 CLEAN清空数据）

operatorType OperatorType MANAGE 操作人类别（OTHER其他 MANAGE后台用户 MOBILE手机端用户）

isSaveRequestData boolean true 是否保存请求的参数

关于自定义操作功能使用流程

1、在BusinessType中新增业务操作类型如:

/\*\*

\* 测试

\*/

TEST,

2、在sys\_dict\_data字典数据表中初始化操作业务类型

insert into sys\_dict\_data values(25, 10, '测试', '10', 'sys\_oper\_type', '', 'primary', 'N', '0', 'admin', '2018-03-16 11-33-00', 'ry', '2018-03-16 11-33-00', '测试操作');

3、在Controller中使用注解

@Log(title = "测试标题", businessType = BusinessType.TEST)

逻辑实现代码 com.doudou.framework.aspectj.LogAspect

查询操作详细记录可以登录系统（系统管理-操作日志）

## 数据权限

在实际开发中，需要设置用户只能查看哪些部门的数据，这种情况一般称为数据权限。

例如对于销售，财务的数据，它们是非常敏感的，因此要求对数据权限进行控制， 对于基于集团性的应用系统而言，就更多需要控制好各自公司的数据了。如设置只能看本公司、或者本部门的数据，对于特殊的领导，可能需要跨部门的数据， 因此程序不能硬编码那个领导该访问哪些数据，需要进行后台的权限和数据权限的控制。

默认系统管理员admin拥有所有数据权限（userId=1），默认角色拥有所有数据权限（如不需要数据权限不用设置数据权限操作）

关于数据权限使用流程

支持参数如下：

参数 类型 默认值 描述

tableAlias String 空 表的别名

1、在（系统管理-角色管理）设置需要数据权限的角色

2、在需要数据权限控制方法上添加@DataScope注解，其中u用来表示表的别名

@DataScope(tableAlias = "u")

3、在mybatis查询底部标签添加数据范围过滤

<!-- 数据范围过滤 -->

${params.dataScope}

用户管理（未过滤数据权限的情况）：

select u.user\_id, u.dept\_id, u.login\_name, u.user\_name, u.email

, u.phonenumber, u.password, u.sex, u.avatar, u.salt

, u.status, u.del\_flag, u.login\_ip, u.login\_date, u.create\_by

, u.create\_time, u.remark, d.dept\_name

from sys\_user u

left join sys\_dept d on u.dept\_id = d.dept\_id

where u.del\_flag = '0'

用户管理（已过滤数据权限的情况）：

select u.user\_id, u.dept\_id, u.login\_name, u.user\_name, u.email

, u.phonenumber, u.password, u.sex, u.avatar, u.salt

, u.status, u.del\_flag, u.login\_ip, u.login\_date, u.create\_by

, u.create\_time, u.remark, d.dept\_name

from sys\_user u

left join sys\_dept d on u.dept\_id = d.dept\_id

where u.del\_flag = '0'

and u.dept\_id in (

select dept\_id

from sys\_role\_dept

where role\_id = 2

)

结果很明显，我们多了如下语句。通过角色部门表（sys\_role\_dept）完成了数据权限过滤

and u.dept\_id in (

select dept\_id

from sys\_role\_dept

where role\_id = 2

)

逻辑实现代码 com.doudou.framework.aspectj.DataScopeAspect

## 多数据源

在实际开发中，经常可能遇到在一个应用中可能需要访问多个数据库的情况

在需要切换数据源Service或Mapper方法上添加@DataSource注解

@DataSource(value = DataSourceType.MASTER)，其中value用来表示数据源名称

关于多数据源使用流程（如果有多个，可以参考slave添加）

支持参数如下：

参数 类型 默认值 描述

value DataSourceType DataSourceType.MASTER 主库

1、在application-druid.yml配置从库数据源

# 从库数据源

slave:

# 从数据源开关/默认关闭

enabled: true

url: jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&useSSL=true&serverTimezone=GMT%2B8

username: root

password: password

2、在DataSourceType类添加数据源枚举

/\*\*

\* 从库

\*/

SLAVE

3、在DruidConfig配置读取数据源

@Bean

@ConfigurationProperties("spring.datasource.druid.slave")

@ConditionalOnProperty(prefix = "spring.datasource.druid.slave", name = "enabled", havingValue = "true")

public DataSource slaveDataSource(DruidProperties druidProperties)

{

DruidDataSource dataSource = DruidDataSourceBuilder.create().build();

return druidProperties.dataSource(dataSource);

}

4、在DruidConfig类dataSource方法添加数据源

targetDataSources.put(DataSourceType.SLAVE.name(), slaveDataSource);

5、在需要使用多数据源方法上添加@DataSource注解，其中value用来表示数据源

@DataSource(value = DataSourceType.SLAVE)

public List<SysUser> selectUserList(SysUser user)

{

return userMapper.selectUserList(user);

}

对于特殊情况可以通过DynamicDataSourceContextHolder手动实现数据源切换

public List<SysUser> selectUserList(SysUser user)

{

DynamicDataSourceContextHolder.setDataSourceType(DataSourceType.SLAVE.name());

List<SysUser> userList = userMapper.selectUserList(user);

DynamicDataSourceContextHolder.clearDataSourceType();

return userList;

}

逻辑实现代码 com.doudou.framework.aspectj.DataSourceAspect

注意：目前配置了一个从库，默认关闭状态。如果不需要多数据源不用做任何配置。

另外可新增多个从库。支持不同数据源（Mysql、Oracle、SQLServer）

## 代码生成

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。

关于代码生成使用流程

1、修改代码生成配置

单应用编辑resources目录下的application.yml

多模块编辑DouDou-generator中的resources目录下的generator.yml

author: # 开发者姓名，生成到类注释上

packageName: # 默认生成包路径

autoRemovePre: # 是否自动去除表前缀

tablePrefix: # 表前缀

2、新建数据库表结构（需要表注释）

drop table if exists sys\_student;

create table sys\_student (

student\_id int(11) auto\_increment comment '学生编号',

student\_name varchar(30) default '' comment '学生名称',

student\_age int(3) default null comment '学生年龄',

primary key (student\_id)

) engine=innodb auto\_increment=1 comment = '学生信息表';

3、登录系统（系统工具 -> 代码生成） 找到sys\_student表，点击生成代码会得到一个DouDou.zip

执行sql文件，覆盖文件到对应目录即可

多模块所有代码生成的相关业务逻辑代码在DouDou-generator模块，可以自行调整或剔除

## 定时任务

在实际项目开发中Web应用有一类不可缺少的，那就是定时任务。 定时任务的场景可以说非常广泛，比如某些视频网站，购买会员后，每天会给会员送成长值，每月会给会员送一些电影券； 比如在保证最终一致性的场景中，往往利用定时任务调度进行一些比对工作；比如一些定时需要生成的报表、邮件；比如一些需要定时清理数据的任务等。 所以我们提供方便友好的web界面，实现动态管理任务，可以达到动态控制定时任务启动、暂停、重启、删除、添加、修改等操作，极大地方便了开发过程。

关于定时任务使用流程

1、后台添加定时任务处理类，并且添加对应Bean注解（@Component或@Service）。如下：

/\*\*

\* 定时任务调度测试

\*

\* @author DouDou

\*/

@Component("ryTask")

public class RyTask

{

public void ryParams(String params)

{

System.out.println("执行有参方法：" + params);

}

public void ryNoParams()

{

System.out.println("执行无参方法");

}

}

2、前端新建定时任务信息（系统监控 -> 定时任务）

任务名称：对应后台bean注解value名称

任务组名：对应的定时任务组名

方法名称：对应后台任务方法名称 方法参数：对应后台任务方法名称值，没有可不填 执行表达式：可查询官方cron表达式介绍

执行策略：定时任务自定义执行策略

并发执行：是否需要多个任务间同时执行

状态：是否启动定时任务

备注：定时任务描述信息

3、点击执行一次，测试定时任务是否正常及调度日志是否正确记录，如正常执行表示任务配置成功。

执行策略详解：

立即执行（所有misfire的任务会马上执行）打个比方，如果9点misfire了，在10：15系统恢复之后，9点，10点的misfire会马上执行

执行一次（会合并部分的misfire，正常执行下一个周期的任务）假设9，10的任务都misfire了，系统在10：15分起来了。只会执行一次misfire，下次正点执行。

放弃执行(所有的misfire不管，执行下一个周期的任务)

常用表达式例子:

表达式 说明

0 0 2 1 \* ? \* 表示在每月的1日的凌晨2点调整任务

0 15 10 ? \* MON-FRI 表示周一到周五每天上午10:15执行作业

0 15 10 ? 6L 2002-2006 表示2002-2006年的每个月的最后一个星期五上午10:15执行作

0 0 10,14,16 \* \* ? 每天上午10点，下午2点，4点

0 0/30 9-17 \* \* ? 朝九晚五工作时间内每半小时

0 0 12 ? \* WED 表示每个星期三中午12点

0 0 12 \* \* ? 每天中午12点触发

0 15 10 ? \* \* 每天上午10:15触发

0 15 10 \* \* ? 每天上午10:15触发

0 15 10 \* \* ? \* 每天上午10:15触发

0 15 10 \* \* ? 2005 2005年的每天上午10:15触发

0 \* 14 \* \* ? 在每天下午2点到下午2:59期间的每1分钟触发

0 0/5 14 \* \* ? 在每天下午2点到下午2:55期间的每5分钟触发

0 0/5 14,18 \* \* ? 在每天下午2点到2:55期间和下午6点到6:55期间的每5分钟触发

0 0-5 14 \* \* ? 在每天下午2点到下午2:05期间的每1分钟触发

0 10,44 14 ? 3 WED 每年三月的星期三的下午2:10和2:44触发

0 15 10 ? \* MON-FRI 周一至周五的上午10:15触发

0 15 10 15 \* ? 每月15日上午10:15触发

0 15 10 L \* ? 每月最后一日的上午10:15触发

0 15 10 ? \* 6L 每月的最后一个星期五上午10:15触发

0 15 10 ? \* 6L 2002-2005 2002年至2005年的每月的最后一个星期五上午10:15触发

0 15 10 ? \* 6#3 每月的第三个星期五上午10:15触发

多模块所有定时任务的相关业务逻辑代码在DouDou-quartz模块，可以自行调整或剔除

注意：不同数据源定时任务都有对应脚本，Oracle、Mysql已经有了，其他的可自行下载执行

## 系统接口

在现在的开发过程中还有很大一部分公司都是以口口相传的方式来进行前后端的联调，而接口文档很大一部分都只停留在了说说而已的地步，或者写了代码再写文档。 还有一点就是文档的修改，定义好的接口并不是一成不变的，可能在开发过程中文档修改不止一次的变化，这个时候就会很难受了。 只要不是强制性要求，没人会愿意写这东西，而且在写的过程中，一个字母的错误就会导致联调时候的很大麻烦，但是通过Swagger，我们可以省略了这一步，而且文档出错率近乎于零， 只要你在写代码的时候，稍加几个注解，文档自动生成。

1、在控制层Controller中添加注解来描述接口信息如:

@Api("参数配置")

@Controller

@RequestMapping("/system/config")

public class ConfigController

2、在方法中配置接口的标题信息

@ApiOperation("查询参数列表")

@ResponseBody

public TableDataInfo list(Config config)

{

startPage();

List<Config> list = configService.selectConfigList(config);

return getDataTable(list);

}

3、在系统工具-系统接口测试相关接口

注意：SwaggerConfig可以指定根据注解或者包名扫描具体的API

API详细说明

作用范围 API 使用位置

协议集描述 @Api 用于controller类上

对象属性 @ApiModelProperty 用在出入参数对象的字段上

协议描述 @ApiOperation 用在controller的方法上

Response集 @ApiResponses 用在controller的方法上

Response @ApiResponse 用在 @ApiResponses里边

非对象参数集 @ApiImplicitParams 用在controller的方法上

非对象参数描述 @ApiImplicitParam 用在@ApiImplicitParams的方法里边

描述返回对象的意义 @ApiModel 用在返回对象类上

api标记，用在类上，说明该类的作用。可以标记一个Controller类做为swagger 文档资源，使用方式：

@Api(value = "/user", description = "用户管理")

与Controller注解并列使用。 属性配置：

属性名称 备注

value url的路径值

tags 如果设置这个值、value的值会被覆盖

description 对api资源的描述

basePath 基本路径可以不配置

position 如果配置多个Api 想改变显示的顺序位置

produces For example, "application/json, application/xml"

consumes For example, "application/json, application/xml"

protocols Possible values: http, https, ws, wss.

authorizations 高级特性认证时配置

hidden 配置为true 将在文档中隐藏

ApiOperation标记，用在方法上，说明方法的作用，每一个url资源的定义,使用方式：

@ApiOperation("获取用户信息")

与Controller中的方法并列使用，属性配置：

属性名称 备注

value url的路径值

tags 如果设置这个值、value的值会被覆盖

description 对api资源的描述

basePath 基本路径可以不配置

position 如果配置多个Api 想改变显示的顺序位置

produces For example, "application/json, application/xml"

consumes For example, "application/json, application/xml"

protocols Possible values: http, https, ws, wss.

authorizations 高级特性认证时配置

hidden 配置为true将在文档中隐藏

response 返回的对象

responseContainer 这些对象是有效的 "List", "Set" or "Map".，其他无效

httpMethod "GET", "HEAD", "POST", "PUT", "DELETE", "OPTIONS" and "PATCH"

code http的状态码 默认 200

extensions 扩展属性

ApiParam标记，请求属性，使用方式：

public TableDataInfo list(@ApiParam(value = "查询用户列表", required = true)User user)

与Controller中的方法并列使用，属性配置：

属性名称 备注

name 属性名称

value 属性值

defaultValue 默认属性值

allowableValues 可以不配置

required 是否属性必填

access 不过多描述

allowMultiple 默认为false

hidden 隐藏该属性

example 举例子

ApiResponse标记，响应配置，使用方式：

@ApiResponse(code = 400, message = "查询用户失败")

与Controller中的方法并列使用，属性配置：

属性名称 备注

code http的状态码

message 描述

response 默认响应类 Void

reference 参考ApiOperation中配置

responseHeaders 参考 ResponseHeader 属性配置说明

responseContainer 参考ApiOperation中配置

ApiResponses标记，响应集配置，使用方式:

@ApiResponses({ @ApiResponse(code = 400, message = "无效的用户") })

与Controller中的方法并列使用，属性配置：

属性名称 备注

value 多个ApiResponse配置

ResponseHeader标记，响应头设置，使用方法

@ResponseHeader(name="head",description="响应头设计")

与Controller中的方法并列使用，属性配置：

属性名称 备注

name 响应头名称

description 描述

response 默认响应类 void

responseContainer 参考ApiOperation中配置

# 前台手册

## [前端组件](http://doc.ruoyi.vip/" \l "/standard/qdsc?id=%e5%89%8d%e7%ab%af%e7%bb%84%e4%bb%b6)

若依封装了一些常用的JS组件方法。

| **名称** | **代码** | **介绍** |
| --- | --- | --- |
| 表格 | $.table | 表格封装处理 |
| 表格树 | $.treeTable | 表格树封装处理 |
| 表单 | $.form | 表单封装处理 |
| 弹出层 | $.modal | 弹出层封装处理 |
| 操作 | $.operate | 操作封装处理 |
| 校验 | $.validate | 校验封装处理 |
| 树插件 | $.tree | 树插件封装处理 |
| 通用方法 | $.common | 通用方法封装处理 |

## [通用方法](http://doc.ruoyi.vip/" \l "/standard/qdsc?id=%e9%80%9a%e7%94%a8%e6%96%b9%e6%b3%95)

支持属性

| **方法** | **参数** | **介绍** |
| --- | --- | --- |
| $.table.init(); | options（选项参数） | 初始化表格参数 |
| $.table.search(); | formId（表单ID） | 搜索-默认第一个form |
| $.table.exportExcel(); | formId（表单ID） | 导出-默认第一个form |
| $.table.importExcel(); | formId（表单ID） | 导入-默认importForm |
| $.table.importTemplate(); | formId（表单ID） | 摸版下载 |
| $.table.refresh(); | 无 | 刷新表格 |
| $.table.selectColumns(); | column（查询列值） | 查询表格指定列值 |
| $.table.selectFirstColumns(); | 无 | 查询表格首列值 |
| $.table.destroy(); | tableId（表格ID） | 销毁表格-默认options.id |
| $.table.serialNumber(); | index（序号） | 序列号生成 |
| $.table.dropdownToggle(); | value（内容） | 下拉按钮切换 |
| $.table.showColumn(); | column（列值） | 显示表格特定的列 |
| $.table.hideColumn(); | column（列值） | 隐藏表格特定的列 |
| $.table.tooltip(); | value（内容）, length（截取长度） | 超出指定长度浮动提示 |
| $.table.selectDictLabel(); | datas（字典列表）, value（当前值） | 回显数据字典 |
| $.treeTable.init(); | options（选项参数） | 初始化表格树参数 |
| $.treeTable.search(); | formId（表单ID） | 搜索-默认第一个form |
| $.treeTable.refresh(); | 无 | 刷新表格树 |
| $.form.reset(); | formId（表单ID） | 表单重置 |
| $.form.selectCheckeds(); | name（name名称） | 获取选中复选框项 |
| $.form.selectSelects(); | name（id名称） | 获取选中下拉框项 |
| $.modal.icon | type（图标类型） | 显示图标 |
| $.modal.msg | content（内容）, type（图标类型） | 消息提示 |
| $.modal.msgError(); | content（内容） | 错误消息 |
| $.modal.msgSuccess(); | content（内容） | 成功消息 |
| $.modal.msgWarning(); | content（内容） | 警告消息 |
| $.modal.alert | content（内容）, type（图标类型） | 消息提示 |
| $.modal.msgReload | msg（消息）, type（图标类型） | 消息提示并刷新父窗体 |
| $.modal.alertError(); | content（内容） | 错误提示 |
| $.modal.alertSuccess(); | content（内容） | 成功提示 |
| $.modal.alertWarning(); | content（内容） | 警告提示 |
| $.modal.close(); | 无 | 关闭窗体 |
| $.modal.closeAll | 无 | 关闭全部窗体 |
| $.modal.confirm(); | content（内容）, callBack（回调函数） | 确认窗体 |
| $.modal.open(); | title, url, width, height, callBack（回调函数） | 弹出层指定宽度 |
| $.modal.openOptions(); | options（选项参数） | 弹出层指定参数选项 |
| $.modal.openFull(); | title, url, width, height | 弹出层全屏 |
| $.modal.openTab | title（标题）, url（地址） | 选卡页方式打开 |
| $.modal.disable | 无 | 禁用按钮 |
| $.modal.enable | 无 | 启用按钮 |
| $.modal.loading(); | message（提示消息） | 打开遮罩层 |
| $.modal.closeLoading(); | 无 | 关闭遮罩层 |
| $.modal.reload(); | 无 | 重新加载 |
| $.operate.submit(); | url, type, dataType, data | 提交数据 |
| $.operate.post(); | url（地址）, data（数据） | post方式请求提交数据 |
| $.operate.get(); | url（地址） | get请求传输数据 |
| $.operate.detail(); | id（数据ID） | 详细信息 |
| $.operate.remove(); | id（数据ID） | 删除信息 |
| $.operate.removeAll(); | 无 | 批量删除信息 |
| $.operate.clean(); | 无 | 清空信息 |
| $.operate.add(); | id（数据ID） | 添加信息 |
| $.operate.addTab(); | id（数据ID） | 添加信息（选项卡方式） |
| $.operate.addFull(); | id（数据ID） | 添加信息 全屏 |
| $.operate.addUrl(); | id（数据ID） | 添加访问地址 |
| $.operate.edit(); | id（数据ID） | 修改信息 |
| $.operate.editTab(); | id（数据ID） | 修改信息（选项卡方式） |
| $.operate.editFull(); | id（数据ID） | 修改信息 全屏 |
| $.operate.editUrl(); | id（数据ID） | 修改访问地址 |
| $.operate.save(); | url（地址）, data（数据） | 保存信息 |
| $.operate.saveTab(); | url（地址）, data（数据） | 保存选项卡信息 |
| $.operate.ajaxSuccess(); | result（返回结果） | 保存结果弹出msg刷新table表格 |
| $.operate.saveSuccess(); | result（返回结果） | 保存结果提示msg |
| $.operate.successCallback(); | result（返回结果） | 成功回调执行事件（静默更新） |
| $.operate.successTabCallback(); | result（返回结果） | 选项卡成功回调执行事件（静默更新） |
| $.validate.unique(); | value（返回标识） | 判断返回标识是否唯一 |
| $.validate.form(); | formId（表单ID） | 表单验证-默认第一个form |
| $.tree.init(); | options（选项参数） | 初始化树结构 |
| $.tree.searchNode(); | 无 | 搜索节点 |
| $.tree.selectByIdName(); | treeId, treeName, node | 根据Id和Name选中指定节点 |
| $.tree.showAllNode(); | nodes（全部节点数据） | 显示所有节点 |
| $.tree.hideAllNode(); | nodes（全部节点数据） | 隐藏所有节点 |
| $.tree.showParent(); | treeNode（节点数据） | 显示所有父节点 |
| $.tree.showChildren(); | treeNode（节点数据） | 显示所有孩子节点 |
| $.tree.updateNodes(); | nodeList（全部节点数据） | 更新节点状态 |
| $.tree.getCheckedNodes(); | column（列值） | 获取当前被勾选集合 |
| $.tree.notAllowParents(); | \_tree（树对象） | 不允许根父节点选择 |
| $.tree.toggleSearch(); | 无 | 隐藏/显示搜索栏 |
| $.tree.collapse(); | 无 | 树折叠 |
| $.tree.expand(); | 无 | 树展开 |
| $.common.isEmpty(); | value（值） | 判断字符串是否为空 |
| $.common.isNotEmpty(); | value（值） | 判断一个字符串是否为非空串 |
| $.common.nullToStr(); | value（值） | 空对象转字符串 |
| $.common.visible(); | value（值） | 是否显示数据 为空默认为显示 |
| $.common.trim(); | value（值） | 空格截取 |
| $.common.random(); | min（最小）, max（最大） | 指定随机数返回 |
| $.common.startWith(); | value（值）, start（开始值） | 判断字符串是否是以start开头 |
| $.common.endWith(); | value（值）, end（结束值） | 判断字符串是否是以end结尾 |

## [表格使用](http://doc.ruoyi.vip/" \l "/standard/qdsc?id=%e8%a1%a8%e6%a0%bc%e4%bd%bf%e7%94%a8)

表格组件基于bootstrap table组件进行封装，轻松实现数据表格。

* 表格初始化 $.table.init

表的各项(Table options )

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| url | String | Null | 请求后台的URL |
| uniqueId | String | Null | 指定唯一列属性 配合删除/修改使用 未指定则使用表格行首列 |
| createUrl | String | Null | 新增URL 配合使用 $.operate.add()，$.operate.addTab() |
| updateUrl | String | Null | 修改URL 配合使用 $.operate.edit()，$.operate.editTab() |
| removeUrl | String | Null | 删除URL 配合使用 $.operate.remove() |
| exportUrl | String | Null | 导出URL 配合使用 $.table.exportExcel() |
| importUrl | String | Null | 导入URL 配合使用 $.table.importExcel() |
| detailUrl | String | Null | 详细URL 配合使用 $.operate.detail() |
| cleanUrl | String | Null | 清空URL 配合使用 $.operate.clean() |
| importTemplateUrl | String | Null | 摸版URL 配合使用 $.table.importTemplate() |
| height | String | undefined | 表格的高度 |
| striped | String | false | 是否显示行间隔色 |
| sortName | String | Null | 排序列名称 |
| sortOrder | String | Null | 排序方式 asc 或者 desc |
| pagination | Boolean | true | 默认为true表格的底部工具栏会显示分页条，设为false不显示 |
| pageSize | int | 10 | 每页的记录行数（\*） |
| id | String | bootstrap-table | 表格ID属性 |
| toolbar | String | toolbar | 表格工具栏ID属性 |
| escape | Boolean | false | 是否转义HTML字符串 |
| showFooter | Boolean | false | 默认为false隐藏表尾，设为true显示 |
| sidePagination | String | server | server启用服务端分页client客户端分页 |
| search | Boolean | true | 默认为true显示搜索框功能，设为false隐藏 |
| showSearch | Boolean | true | 默认为true显示检索信息，设为false隐藏 |
| showPageGo | Boolean | false | 默认为false不显示跳转页，设为true显示 |
| showRefresh | Boolean | true | 默认为true显示刷新按钮，设为false隐藏 |
| showColumns | Boolean | true | 默认为true显示某列下拉菜单，设为false隐藏 |
| showToggle | Boolean | true | 默认为true显示视图切换按钮，设为false隐藏 |
| showExport | Boolean | true | 默认为true显示导出文件按钮，设为false隐藏 |
| clickToSelect | Boolean | false | 默认为false不启用点击选中行，设为true启用 |
| rememberSelected | Boolean | false | 默认为false不启用翻页记住前面的选择，设为true启用 |
| fixedColumns | Boolean | false | 默认为false禁用冻结列，设为true启用冻结列（左侧） |
| fixedNumber | int | 0 | 冻结列的个数，当fixedColumns设为true有效（左侧） |
| rightFixedColumns | Boolean | false | 默认为false禁用冻结列，设为true启用冻结列（右侧） |
| rightFixedNumber | int | 0 | 冻结列的个数，当fixedColumns设为true有效（右侧） |
| params | Array | Null | 当请求数据时，你可以通过修改queryParams向服务器发送参数 |
| columns | Array | Null | 默认空数组，在JS里面定义 参考列的各项(Column options ) |
| responseHandler | Array | Null | 在加载服务器发送来的数据之前，处理数据的格式 |
| onLoadSuccess | Array | Null | 当所有数据被加载时触发处理函数 |

列的各项(Column options )

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| radio | Boolean | false | 默认false不显示radio（单选按钮），设为true则显示，radio宽度是固定的 |
| checkbox | Boolean | false | 默认false不显示checkbox（复选框），设为true则显示，checkbox的每列宽度已固定 |
| field | String | Null | 是每列的字段名，不是表头所显示的名字，通过这个字段名可以给其赋值，相当于key，表内唯一 |
| title | String | Null | 这个是表头所显示的名字，不唯一，如果你喜欢，可以把所有表头都设为相同的名字 |
| titleTooltip | String | true | 当悬浮在某控件上，出现提示 - 参考 Bootstrap 提示工具（Tooltip）插件 |
| class | boolean | false | 表格列样式 |
| rowspan | Number | true | 每格所占的行数 |
| colspan | Number | true | 每格所占的列数 |
| align | String | true | 每格内数据的对齐方式，有：left（靠左）、right（靠右）、center（居中） |
| halign | String | true | table header（表头）的对齐方式，有：left（靠左）、right（靠右）、center（居中） |
| falign | String | true | table footer（表脚，的对齐方式，有：left（靠左）、right（靠右）、center（居中） |
| valign | String | true | 每格数据的对齐方式，有：top（靠上）、middle（居中）、bottom（靠下） |
| width | Number | Null | 每列的宽度。如果没有自定义宽度自适应 |
| sortable | Boolean | false | 默认false就默认显示，设为true则会被排序 |
| order | String | asc | 默认的排序方式为"asc（升序）"，也可以设为"desc（降序）"。 |
| visible | Boolean | true | 默认为true显示该列，设为false则隐藏该列 |
| cardVisible | Boolean | true | 默认为true显示该列，设为false则隐藏。 |
| switchable | Boolean | true | 默认为true显示该列，设为false则禁用列项目的选项卡。 |
| clickToSelect | Boolean | true | 默认true不响应，设为false则当点击此行的某处时，不会自动选中此行的checkbox（复选框）或radiobox（单选按钮） |
| formatter | Function | Null | 某格的数据转换函数，需要三个参数： -value： field（字段名） -row：行的数据 -index：行的（索引）index |
| footerFormatter | Function | Null | 某格的数据转换函数，需要一个参数： -data： 所有行数据的数组 函数需要返回（return）footer某格内所要显示的字符串的格式 |
| events | Object | Null | 当某格使用formatter函数时，事件监听会响应，需要四个参数： -event：-value：字段名 -row：行数据 -index：此行的index |
| sorter | Function | Null | 自定义的排序函数，实现本地排序，需要两个参数： - a：第一个字段名 - b：第二个字段名 |
| sortName | String | Null | 排序列名称 |
| cellStyle | Function | Null | 对某列中显示样式改变 |
| searchable | Boolean | true | 默认true，表示此列数据可被查询 |
| searchFormatter | Boolean | true | 默认true，可使用格式化的数据查询 |
| escape | Boolean | false | 是否转义HTML字符串 |

* 表单搜索 $.table.search

<a onclick="$.table.search();">搜索</a>

* 表格数据导出 $.table.exportExcel

<a onclick="$.table.exportExcel();">导出</a>

* 数据模板下载 $.table.importTemplate

<a onclick="$.table.importTemplate();">下载模板</a>

* 表格数据导入 $.table.importExcel
* <a onclick="$.table.importExcel();">导入</a>
* <form id="importForm" enctype="multipart/form-data" class="mt20 mb10" style="display: none;">
* <div class="col-xs-offset-1">
* <input type="file" id="file" name="file"/>
* <div class="mt10 pt5">
* <input type="checkbox" id="updateSupport" name="updateSupport" title="如果登录账户已经存在，更新这条数据。"> 是否更新已经存在的用户数据
* &nbsp; <a onclick="$.table.importTemplate()" class="btn btn-default btn-xs"><i class="fa fa-file-excel-o"></i> 下载模板</a>
* </div>
* <font color="red" class="pull-left mt10">
* 提示：仅允许导入“xls”或“xlsx”格式文件！
* </font>
* </div>

</form>

* 表格销毁 $.table.destroy

<a onclick="$.table.destroy();">销毁</a>

* 表格数据刷新 $.table.refresh

<a onclick="$.table.refresh();">刷新</a>

* 选择表格行具体列 $.table.selectColumns

var loginName = $.table.selectColumns("loginName");

* 选择表格行首列 $.table.selectFirstColumns

var firstColumn = $.table.selectFirstColumns();

* 显示表格特定的列 $.table.showColumn

$.table.showColumn("userName");

* 隐藏表格特定的列 $.table.hideColumn

$.table.hideColumn("userName");

* 序列号生成 $.table.serialNumber
* {
* title: "序号",
* formatter: function (value, row, index) {
* return $.table.serialNumber(index);
* }

},

* 超出指定长度浮动提示 $.table.tooltip
* {
* field: 'remark',
* title: '备注',
* align: 'center',
* formatter: function(value, row, index) {
* return $.table.tooltip(value);
* }

},

* 回显数据字典 $.table.selectDictLabel
* var datas = [[${@dict.getType('sys\_common\_status')}]];
* {
* field: 'status',
* title: '用户状态',
* align: 'center',
* formatter: function(value, row, index) {
* return $.table.selectDictLabel(datas, value);
* }

},

* 下拉按钮切换 $.table.dropdownToggle
* formatter: function(value, row, index) {
* var actions = [];
* actions.push('<a class="' + editFlag + '" href="#" onclick="$.operate.edit(\'' + row.deptId + '\')"><i class="fa fa-edit"></i>编辑</a>');
* actions.push('<a class="' + removeFlag + '" href="#" onclick="$.operate.remove(\'' + row.deptId + '\')"><i class="fa fa-trash"></i>删除</a>');
* actions.push('<a class="' + addFlag + '" href="#" onclick="$.operate.add(\'' + row.deptId + '\')"><i class="fa fa-plus"></i>添加下级部门</a>');
* return $.table.dropdownToggle(actions.join(''));

}

## [弹层使用](http://doc.ruoyi.vip/" \l "/standard/qdsc?id=%e5%bc%b9%e5%b1%82%e4%bd%bf%e7%94%a8)

弹层组件目前基于layer组件进行封装,提供了弹出、消息、提示、确认、遮罩处理等功能。

* 提供成功、警告和错误等反馈信息
* $.modal.msg("默认反馈");
* $.modal.msgError("错误反馈");
* $.modal.msgSuccess("成功反馈");

$.modal.msgWarning("警告反馈");

* 提供成功、警告和错误等提示信息
* $.modal.alert("默认提示");
* $.modal.alertError("错误提示");
* $.modal.alertSuccess("成功提示");
* $.modal.alertWarning("警告提示");

$.modal.confirm("确认信息", function() {});

* 提供弹出层信息
* // title 标题 url 请求链接 width 宽度 height 高度 options 选项
* $.modal.open(title, url, width, height);
* $.modal.openTab(title, url);
* $.modal.openOptions(options);
* $.modal.openFull(title, url, width, height);
* $.modal.close();
* $.modal.closeAll();

$.modal.reload();

* 提供遮罩层信息
* $.modal.loading("正在导出数据，请稍后...");

$.modal.closeLoading();

## [权限使用](http://doc.ruoyi.vip/" \l "/standard/qdsc?id=%e6%9d%83%e9%99%90%e4%bd%bf%e7%94%a8)

使用thymeleaf模板整合了shiro标签 - 界面可以直接使用。（此处简单介绍两个，更多请参考官方文档）

<a href="#" shiro:hasPermission="system:user:add">包含权限字符串才能看到</a>

<a href="#" shiro:hasRole="admin">管理员才能看到</a>

如果需要在JS中使用权限，使用封装方法

var addFlag = [[${@permission.hasPermi('system:user:add')}]];

<a class="btn btn-success btn-xs ' + editFlag + '">包含权限字符串才能看到</a>

## [字典使用](http://doc.ruoyi.vip/" \l "/standard/qdsc?id=%e5%ad%97%e5%85%b8%e4%bd%bf%e7%94%a8)

配置好相关的数据字典信息即可正常使用（系统管理-字典管理）

<select name="status" th:with="type=${@dict.getType('sys\_normal\_disable')}">

<option value="">所有</option>

<option th:each="dict : ${type}" th:text="${dict.dictLabel}" th:value="${dict.dictValue}">

</option>

</select>

如果在想Table表格数据使用字典，使用formatter格式化

// 获取数据字典数据

var datas = [[${@dict.getType('sys\_normal\_disable')}]];

// 格式化数据字典

formatter: function(value, row, index) {

return $.table.selectDictLabel(datas, value);

}

## [参数使用](http://doc.ruoyi.vip/" \l "/standard/qdsc?id=%e5%8f%82%e6%95%b0%e4%bd%bf%e7%94%a8)

配置好相关的参数信息即可正常使用（系统管理-参数管理）

<body th:classappend="${@config.getKey('sys.index.skinName')}">

如果需要在JS中使用参数，使用封装方法

var skinName = [[${@config.getKey('sys.index.skinName')}]];

$("#id").val(skinName);

# 常见问题

## [如何不登录直接访问](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e5%a6%82%e4%bd%95%e4%b8%8d%e7%99%bb%e5%bd%95%e7%9b%b4%e6%8e%a5%e8%ae%bf%e9%97%ae)

在 ShiroConfig 中设置filterChainDefinitionMap 配置url=anon

/admins/\*\*=anon # 表示该 uri 可以匿名访问

/admins/\*\*=auth # 表示该 uri 需要认证才能访问

/admins/\*\*=authcBasic # 表示该 uri 需要 httpBasic 认证

/admins/\*\*=perms[user:add:\*] # 表示该 uri 需要认证用户拥有 user:add:\* 权限才能访问

/admins/\*\*=port[8080] # 表示该 uri 需要使用 8080 端口

/admins/\*\*=roles[admin] # 表示该 uri 需要认证用户拥有 admin 角色才能访问

/admins/\*\*=ssl # 表示该 uri 需要使用 https 协议

/admins/\*\*=user # 表示该 uri 需要认证或通过记住我认证才能访问

/logout=logout # 表示注销,可以当作固定配置

注意：

anon，authcBasic，authc，user 是认证过滤器。

perms，roles，ssl，rest，port 是授权过滤器。

## [如何使用多数据源](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e5%a6%82%e4%bd%95%e4%bd%bf%e7%94%a8%e5%a4%9a%e6%95%b0%e6%8d%ae%e6%ba%90)

1. 在 resources 目录下修改application-druid.yml
2. # 从库数据源
3. slave:
4. # 开启从库
5. enabled: true
6. url: 数据源
7. username: 用户名

password: 密码

1. 在Service实现中添加DataSource注解
2. @DataSource(value = DataSourceType.SLAVE)
3. public List<User> selectUserList()
4. {
5. return mapper.selectUserList();

}

## [日期插件精确到时分秒](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e6%97%a5%e6%9c%9f%e6%8f%92%e4%bb%b6%e7%b2%be%e7%a1%ae%e5%88%b0%e6%97%b6%e5%88%86%e7%a7%92)

datetimepicker日期控件可以设置format

$('.input-group.date').datetimepicker({

format: 'yyyy-mm-dd hh:ii:ss',

autoclose: true,

minView: 0,

minuteStep:1

});

laydate日期控件可以设置common.js 配置type=datetime

layui.use('laydate', function() {

var laydate = layui.laydate;

var startDate = laydate.render({

elem: '#startTime',

max: $('#endTime').val(),

theme: 'molv',

trigger: 'click',

type : 'datetime',

done: function(value, date) {

// 结束时间大于开始时间

if (value !== '') {

endDate.config.min.year = date.year;

endDate.config.min.month = date.month - 1;

endDate.config.min.date = date.date;

} else {

endDate.config.min.year = '';

endDate.config.min.month = '';

endDate.config.min.date = '';

}

}

});

var endDate = laydate.render({

elem: '#endTime',

min: $('#startTime').val(),

theme: 'molv',

trigger: 'click',

type : 'datetime',

done: function(value, date) {

// 开始时间小于结束时间

if (value !== '') {

startDate.config.max.year = date.year;

startDate.config.max.month = date.month - 1;

startDate.config.max.date = date.date;

} else {

startDate.config.max.year = '';

startDate.config.max.month = '';

startDate.config.max.date = '';

}

}

});

});

## [代码生成不显示新建表](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e4%bb%a3%e7%a0%81%e7%94%9f%e6%88%90%e4%b8%8d%e6%98%be%e7%a4%ba%e6%96%b0%e5%bb%ba%e8%a1%a8)

默认条件需要表注释，特殊情况可在GenMapper.xml去除table\_comment条件

<select id="selectTableByName" parameterType="String" resultMap="TableInfoResult">

<include refid="selectGenVo"/>

where table\_comment <> '' and table\_schema = (select database())

</select>

## [提示您没有数据的权限](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e6%8f%90%e7%a4%ba%e6%82%a8%e6%b2%a1%e6%9c%89%e6%95%b0%e6%8d%ae%e7%9a%84%e6%9d%83%e9%99%90)

这种情况都属于权限标识配置不对在菜单管理配置好权限标识（菜单&按钮）

1. 确认此用户是否已经配置角色
2. 确认此角色是否已经配置菜单权限
3. 确认此菜单权限标识是否和后台代码一致

如参数管理  
后台配置@RequiresPermissions("system:config:view")对应参数管理权限标识为system:config:view

注：如需要角色权限，配置角色权限字符 使用@RequiresRoles("admin")

## [富文本编辑器文件上传](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e5%af%8c%e6%96%87%e6%9c%ac%e7%bc%96%e8%be%91%e5%99%a8%e6%96%87%e4%bb%b6%e4%b8%8a%e4%bc%a0)

富文本控件采用的summernote，图片上传处理需要设置callbacks函数

$('.summernote').summernote({

height : '220px',

lang : 'zh-CN',

callbacks: {

onImageUpload: function(files, editor, $editable) {

var formData = new FormData();

formData.append("file", files[0]);

$.ajax({

type: "POST",

url: ctx + "common/upload",

data: data,

cache: false,

contentType: false,

processData: false,

dataType: 'json',

success: function(result) {

if (result.code == web\_status.SUCCESS) {

$(obj).summernote('editor.insertImage', result.url, result.fileName);

} else {

$.modal.alertError(result.msg);

}

},

error: function(error) {

$.modal.alertWarning("图片上传失败。");

}

});

}

}

});

## [如何创建新的菜单页签](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e5%a6%82%e4%bd%95%e5%88%9b%e5%bb%ba%e6%96%b0%e7%9a%84%e8%8f%9c%e5%8d%95%e9%a1%b5%e7%ad%be)

建新新的页签有以下两种方式（js&html）

// 方式1 js创建

function dept() {

var url = ctx + "system/dept";

createMenuItem(url, "部门管理");

}

// 方式2 html创建

<a class="menuItem" href="/system/dept">部门管理</a>

## [表格数据进行汇总统计](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e8%a1%a8%e6%a0%bc%e6%95%b0%e6%8d%ae%e8%bf%9b%e8%a1%8c%e6%b1%87%e6%80%bb%e7%bb%9f%e8%ae%a1)

对于某些数据需要对金额，数量等进行汇总，可以配置showFootertrue表示尾部统计

// options 选项中添加尾部统计

showFooter: true,

// columns 中添加

{

field : 'money',

title : '金额',

sortable: true,

footerFormatter:function (value) {

return "总金额：" + value;

}

},

## [如何去除数据监控广告](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e5%a6%82%e4%bd%95%e5%8e%bb%e9%99%a4%e6%95%b0%e6%8d%ae%e7%9b%91%e6%8e%a7%e5%b9%bf%e5%91%8a)

服务监控中使用的Driud，默认底部有阿里的广告。如果是一个商业项目这个是很不雅也是不允许的

1. 找到本地maven库中的对应的druid-1.1.xx.jar文件，用压缩包软件打开
2. 找到support/http/resource/js/common.js, 打开找到 buildFooter 方法
3. this.buildFooter();
4. buildFooter : function() {
5. var html ='此处省略一些相关JS代码';
6. $(document.body).append(html);

},

1. 删除此函数和及初始方法后覆盖文件
2. 重启项目后，广告就会消失了

## [如何支持多类型数据库](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e5%a6%82%e4%bd%95%e6%94%af%e6%8c%81%e5%a4%9a%e7%b1%bb%e5%9e%8b%e6%95%b0%e6%8d%ae%e5%ba%93)

对于某些特殊需要支持不同数据库，参考以下支持oracle mysql配置

<!--oracle驱动-->

<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc6</artifactId>

<version>11.2.0.3</version>

</dependency>

# 数据源配置

spring:

datasource:

type: com.alibaba.druid.pool.DruidDataSource

druid:

# 主库数据源

master:

url: jdbc:mysql://127.0.0.1:3306/ry?useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&useSSL=true&serverTimezone=GMT%2B8

username: root

password: password

# 从库数据源

slave:

# 从数据源开关/默认关闭

enabled: true

url: jdbc:oracle:thin:@127.0.0.1:1521:oracle

username: root

password: password

## [如何实现翻页保留选择](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e5%a6%82%e4%bd%95%e5%ae%9e%e7%8e%b0%e7%bf%bb%e9%a1%b5%e4%bf%9d%e7%95%99%e9%80%89%e6%8b%a9)

1. 配置checkbox选项field属性为state
2. {
3. field: 'state',
4. checkbox: true

},

1. 表格选项options添加rememberSelected

rememberSelected: true,

## [如何自定义查询条件参数](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e5%a6%82%e4%bd%95%e8%87%aa%e5%ae%9a%e4%b9%89%e6%9f%a5%e8%af%a2%e6%9d%a1%e4%bb%b6%e5%8f%82%e6%95%b0)

1、在options中添加queryParams参数

var options = {

url: prefix + "/list",

queryParams: queryParams,

columns: [{

field: 'id',

title: '主键'

},

{

field: 'name',

title: '名称'

}]

};

$.table.init(options);

2、在当前页添加queryParams方法设置自定义查询条件如userName

function queryParams(params) {

var search = $.table.queryParams(params);

search.userName = $("#userName").val();

return search;

}

请求后台参数为：pageSize、pageNum、searchValue、orderByColumn、isAsc、userName

## [如何降低mysql驱动版本](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e5%a6%82%e4%bd%95%e9%99%8d%e4%bd%8emysql%e9%a9%b1%e5%8a%a8%e7%89%88%e6%9c%ac)

1、在pom.xml中properties新增节点如：

<mysql.version>6.0.6</mysql.version>

2、单应用可以不添加，多模块需要在dependencyManagement声明依赖

<!-- Mysql驱动包 -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>${mysql.version}</version>

</dependency>

注意：如果是6以下的版本需要修改application-druid.yml中driverClassName  
com.mysql.jdbc.Driver 是 mysql-connector-java 5中的  
com.mysql.cj.jdbc.Driver 是 mysql-connector-java 6中的

## [如何汉化系统接口Swagger](http://doc.ruoyi.vip/" \l "/standard/cjwt?id=%e5%a6%82%e4%bd%95%e6%b1%89%e5%8c%96%e7%b3%bb%e7%bb%9f%e6%8e%a5%e5%8f%a3swagger)

想必很多小伙伴都曾经使用过Swagger，但是打开UI界面是纯英文的界面并不太友好，作为国人还是习惯中文界面。

1. 找到m2/repository/io/springfox/springfox-swagger-ui/x.x.x/springfox-swagger-ui-x.x.x.jar
2. 修改对应springfox-swagger-ui-x.x.x.jar包内resources目录下swagger-ui.html，添加如下JS代码
3. <!-- 选择中文版 -->
4. <script src='webjars/springfox-swagger-ui/lang/translator.js' type='text/javascript'></script>

<script src='webjars/springfox-swagger-ui/lang/zh-cn.js' type='text/javascript'></script>

1. 本地修改结束后，在覆盖压缩包文件重启就实现汉化了