**框架开发指南**

目录

[框架开发指南 1](#_Toc17876)

[1. 前提知识 3](#_Toc19313)

[1.1. 前端 3](#_Toc31627)

[1.2. 后端 3](#_Toc15387)

[1.3. 其他 3](#_Toc3267)

[2. 开发指南 3](#_Toc5356)

[2.1. 前端 3](#_Toc14977)

[2.1.1. 目录构成 3](#_Toc19012)

[2.1.2. 代码解释 4](#_Toc10594)

[2.2. 后端 6](#_Toc28934)

[2.2.1. 增删改查 6](#_Toc31377)

[2.2.2. 缓存 7](#_Toc20022)

[2.2.3. 日志 7](#_Toc3429)

[2.2.4. 定时任务 7](#_Toc16558)

[2.2.5. 登录认证 7](#_Toc14019)

[2.2.6. 权限 7](#_Toc15720)

[2.2.7. 配置 8](#_Toc25786)

[2.2.8. 国际化 8](#_Toc6960)

[2.2.9. 扩展 8](#_Toc6142)

[2.3. 场景 8](#_Toc4276)

[2.3.1. 首次登陆 8](#_Toc10788)

[2.3.2. 当日首次登陆后 8](#_Toc8270)

[2.3.3. SSO(CAS、LDAP、OAUTH、企业微信、钉钉) 8](#_Toc20173)

[2.3.4. 登录安全处理（次数、拉黑等） 8](#_Toc18183)

[2.4. 其他 8](#_Toc28454)

[2.4.1. 大量数据处理 8](#_Toc12435)

[2.4.2. 自定义序列化 8](#_Toc14946)

[2.4.3. 特殊类型字段 8](#_Toc93)

[2.4.4. 数据加密 8](#_Toc23330)

[2.4.5. 任务管理 8](#_Toc32740)

[2.4.6. 文件存储 8](#_Toc220)

[2.4.7. 上传下载 9](#_Toc17645)

[2.4.8. 工具类 9](#_Toc9666)

# 前提知识

## 前端

CSS、Javascript（ES6）、HTML

VUE2、VUE3

Antd、ElementUI

## 后端

Java

SpringBoot、Spring Data JPA、Hibernate、Redis、RabbitMQ、HTTP、WebService

## 其他

WebPack、eclipse，IDEA

# 开发指南

## 前端

### 目录构成

resources 根目录

│ application.html 页面框架代码。

│

└─static WEB根目录

│ favicon.ico

│

├─app 分类

│ └─news 模块

│ 存放页面 vue、可以包含 css、js、components

├─assets 存放共通的css、js

├─components 存放共通的components

├─lib 第三方库（vue、echats、ckeditor等第三方库）

│ vue2.7.14.js vue

│

└─sys 分类

├─login 模块

│ login.vue 登录页面的vue

│

└─portal 门户页面的vue

### 代码解释

application.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta name="viewport" content="width=device-width,initial-scale=1">  <link rel="icon" href="/favicon.ico">  <script type="text/javascript" src="/lib/vue2.7.14.js"></script>  <title></title>  <script type='module'>  document.request = {request};  import app from "{url}";  Vue.config.productionTip = false;  new Vue({render: r => r(app)}).$mount("#app");  </script>  </head>  <body>  <div id="app"></div>  </body>  </html> |

{request} 和 {url} 后台自动注入

用户浏览器中输入 <http://server/sys/news/view?fId=1234567890>

服务器返回： 《后台代码：**result.vue("/sys/news/main.vue"， param）；**》

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta name="viewport" content="width=device-width,initial-scale=1">  <link rel="icon" href="/favicon.ico">  <script type="text/javascript" src="/lib/vue2.7.14.js"></script>  <title></title>  <script type='module'>  document.request = **{"fId":"1234567890"}**;  import app from "**/sys/news/main.vue**";  Vue.config.productionTip = false;  new Vue({render: r => r(app)}).$mount("#app");  </script>  </head>  <body>  <div id="app"></div>  </body>  </html> |

Vue文件必须遵循以下规范：

顶层必须包含 template、script 标签、可选包含 style 标签

Import 、src、href路径可以使用绝对路径或者相对路径。比如：/assets/slide.js 或 ./play.png

script 中必须仅包含一个 export default { .....}

浏览器解析 Import vue，并向服务器请求vue文件，后台会分析vue文件并返回其中js部分。并将 template 内容注入到 export 中的 template 里面。同时 把style部分通过js增加link到 document.head 中。link 的 href 为 xxxx.vue.css。浏览器请求后台时，后台会返回 vue文件中 style部分。

举例：

Vue代码(/app/hello/main.vue)

|  |
| --- |
| <template>  <h1>Hello {{name}}<h1>  </template>  <style>  H1 { color: red }  </style>  <script>  export default {  name: 'hello-world',  data() {  return {  name: document.request.name?? ""  };  }  }  </script> |

用户请求 /app/hello?name=John 返回如下代码:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta name="viewport" content="width=device-width,initial-scale=1">  <link rel="icon" href="/favicon.ico">  <script type="text/javascript" src="/lib/vue2.7.14.js"></script>  <title></title>  <script type='module'>  document.request = **{"name":"John}**;  import app from "**/app/hello/main.vue**";  Vue.config.productionTip = false;  new Vue({render: r => r(app)}).$mount("#app");  </script>  </head>  <body>  <div id="app"></div>  </body>  </html> |

请求/app/hello/main.vue返回的内容

|  |
| --- |
| addStyle("**/app/hello/main.vue.css**");  export default {  template: `<h1>Hello {{name}}<h1>`,  name: 'hello-world',  data() {  return {  name: document.request.name?? ""  };  }  } |

浏览器执行 addStyle 会在 document.head 增加

<link rel="stylesheet" href="/app/hello/main.vue.css"/>

浏览器请求 /app/hello/main.vue.css 返回的内容

|  |
| --- |
| H1 { color: red } |

最终，浏览器显示 Hello John。

## 后端

### 增删改查

普通三层构造下增删改查需要Controller，Service，Dao和Model。

本框架会根据Model自动生成对应的Controller，Service，Dao的Bean（简称 CSD）。

支持基本的新增，更新，删除，导入，导出，查询操作。

Model：[标准SpringDataJPA的Model的写法。如果需要增加标准的CSD，只需要在Model的类上加上@SimpleModel(url="...")注解。](mailto:标准SpringDataJPA的Model的写法。如果需要增加标准的Controller，Service和Dao，只需要在Model的类上加上@SimpleModel(url=\"...\")注解。)

系统生成的 CSD使用Model的类名命名。比如：Model名是SysTask 则 生成的ServiceBean名为SysTaskService。

代码中使用如下几种方式方式实现自动注入。注意：变量的命名要和 Bean名称一致

|  |
| --- |
| 方式一：（spring 官方不推荐）  @Autowired  SimpleService<SysTask> sysTaskService; // 变量的命名要和 Bean名称一致  方式二：  SimpleService<SysTask> service = applicationContext.getBean("sysTaskService");  方式三：构造方式注入（spring官方推荐）  public class Test {  public Test( SimpleService<SysTask> sysTaskService) {...}  }  可以使用 @Qualifier("sysTaskService") 注解在注入时指定Bean名称 |

如果需要在标准Bean上增加其他方法，可以通过 继承 SimpleService<Model> 实现。如果类名使用标准命名方式，则覆盖框架自动生成的Bean。

|  |
| --- |
| public class SysTaskService extends SimpleService<SysTask> {  } |

框架生成的Bean代码参看 SimpleRepository，SimpleService，SimpleController。

注意点：

Model循环引用会引起toJSONObject 的异常。比如：树形结构的Model。

对于查询结果数据量非常大的处理尽量避免使用 find 查询非分页的数据。可以使用 scroll 方式进行。

生成的接口如下：

|  |  |  |
| --- | --- | --- |
| 接口地址 | 作用 | 接口说明 |
| {url}/load | 根据主键获取单条数据 |  |
| {url}/save | 新增或更新单挑数据 |  |
| {url}/delete | 删除数据（可批量） |  |
| {url}/list | 查询数据（支持分页） |  |
| {url}/export | 导出数据 |  |
| {url}/import | 导入数据 |  |

### 缓存

本框架支持二级缓存。使用方式参考spring cache。

\*\*\* 修改为使用 JetCache 实现 \*\*\*

### 日志

本框架使用logback作为日志输出组件。提供了写入数据的Appender。

应用日志存储到数据库 或者 ES或者MongoDB（默认）待开发

AppLog.log(data);// 附加记录 node，time，thread等

例：

|  |
| --- |
| AppLog.log(new LoginLog("192.168.10.98", "john.smith", "windows")); |

会按照 如下格式写入数据库

|  |
| --- |
| {  node: "server01",  time: "2024/01/01 12:12:23.182",  thread: "http-nio-thread-11",  ip: "192.168.10.98",  user: "john.smith",  os: "windows"  } |

### 定时任务

QTask（待稳定性验证）

### 登录认证

支持Ajax认证，

扩展可支持 Token认证（OAuth，LtpaToken），LDAP认证

### 权限

访问控制上使用RBAC方式进行控制。用户可以关联组织架构的员工。

ModuleSecurity中可以配置访问控制，动态权限，过滤器。

数据过滤：采用可见对象集配置。

#### 设计理念

权限分为：系统管理员，分级管理员，模块管理员，模块分级管理员 四个大维度

系统管理员可以访问系统所有操作，可以访问系统所有数据。

分级管理员可以访问系统所有操作，可以访问指定范围内的所有数据。

模块管理员可以访问指定模块的所有操作，可以访问该模块下的所有数据。

模块分级管理员可以访问指定模块的所有操作，可以访问该模块下指定范围内的的所有数据。

普通授权：系统管理员设定某人，某角色的权限

分级授权过程：

系统管理员指定分级管理员（给分级管理员设置管辖范围），分级管理员给管辖范围的人员设置角色和权限。

系统管理员指定模块管理员，模块管理员指定模块分级管理员。模块分级管理员给管辖范围的人员设置角色和权限。

设计方式：

权限信息表：权限名称，类型（权限 AUTH、角色 ROLE），所属模块，所属分类，范围

类型：权限是系统内置的权限。比如：新增，删除，管理。权限有所属模块。没有所属分类和范围

角色是用户定义的。比如：归档人，调度员。角色没有硕鼠模块吗，有所属分类和范围

赋权信息表：组织对象，权限角色，范围。

关于数据权限和组织穿透策略。

数据的可修改，可阅读保存在对应的数据权限表中。

数据权限表：数据的主键，策略器，参数，操作（读取，删除，更新，下载，有限下载，作废，回收。。。）

策略器：

ALL（无参数） - 所有人

ONE（指定组织、人）-- 指定人或者组织的单层

PASS（指定组织）-- 指定组织及其所有下层

策略器可以定制：比如：SUBPASS（可穿透子公司，不可穿透分公司）

操作：对文档的各种操作。（仅范围，业务上需根据数据状态等判断是否可以进行该操作）

比如：还在审批中的文档，无法回收。

#### 访问控制

完成

#### 数据过滤

待开发

### 配置

使用 AppConfig 进行配置数据的读写。

|  |
| --- |
| @Component  public class SysTaskConfig extends AppConfig {  public String getModelName() {  return “SysTask”;  }  private vars  public get/set  }  支持数据类型：数字，布尔，字符串，日期及数组  使用方法  public class SysTaskController {  @Autowired  Private SysTaskConfig sysTaskConfig;  Public void reset() {  sysTaskConfig.refresh();  sysTask.setXXX();  sysTask.Commit();  }  } |

### 国际化

本框架从 messages.properties 或者 messages.{lang}.properties 读取资源。

Spring启动后，会从数据库获取用户自定义的数据

### 扩展

使用 Hibernate 标准的增删改扩展

## 场景

### 首次登陆

### 当日首次登陆后

### SSO(CAS、LDAP、OAUTH、企业微信、钉钉)

### 登录安全处理（次数、拉黑等）

## 其他

### 限流熔断

完成

### 大量数据处理

完成 （scroll）

### 自定义序列化

完成待验证

### 特殊类型字段

完成（properties，jsonObject，jsonArray）

### 数据加密

完成（hibernate 特殊字段）

### 任务管理

完成

### 文件存储

完成待验证

### 上传下载

实验完成，待整合

### 工具类

前端：excel填充（上传、复制黏贴），大文件上传

后端：按需