TIMO技术文档

小懒虫

目　录

项目介绍

一、模块介绍

1. admin 后台管理模块

   该模块用于开发后台系统，也是最主要的模块。

2. boot 引导模块——用于项目启动和打包

   该模块只要存放着项目主入口类和配置文件，用于集成其他模块。可以将其改名成前台模块，用于开发前台系统，不建议将主入口类和配置文件放到后台模块中！

3. core 核心模块——用于存放通用代码

4. devtools 开发中心模块——方便项目开发

   该模块是一个相对独立的模块，打包时可以将其依赖去掉，减少项目部署后的体积。

二、目录结构

```
├─admin      --后台管理模块
│    │
│    ├─java
│    │    ├─com.linln.admin.core-------------------后台模块核心模块
│    │    │    ├─config----------配置文件
│    │    │    ├─constant--------常量文件
│    │    │    ├─enums----------枚举类
│    │    │    ├─log-------------日志管理相关类
│    │    │    ├─shiro-----------权限管理相关类
│    │    │    ├─thymeleaf-------模板操作相关类
│    │    │    └─utils------------后台模块工具目录
│    │    └─com.linln.admin.system----------------后台模块系统模块
│    │         ├─controller--------控制器
│    │         ├─domain----------实体类
│    │         ├─repository--------dao层
│    │         ├─service-----------服务层
│    │         └─validator-------------验证类
│    │
│    └─resources
│         ├─static--------------------静态资源目录
│         └─templates-----------------前端模板目录
│              ├─common----------------公共模板目录
│              │    ├─module.html---------------功能模块
│              │    └─template.html--------------页面模板
│              └─system------------------系统模板目录
│
├─boot       --引导模块
│    │
```

```
│   ├─java
│   │   └─com.linln.BootApplication----------------启动项目入口
│   │
│   └─resources
│       └─application.yml--------------------------项目配置文件
│
├─core      --核心模块
│
└─devtools  --开发中心模块
```

前端手册

项目前端使用了layui框架，页面组件文档可参考官网：https://www.layui.com/

图标集成了FontAwesome，提供更多的选择，官网：https://fontawesome.com/v4.7.0/icons/

页面开发

1.数据列表—表格

```
<table class="layui-table timo-table">
    <thead>
    <tr>
        <th>字段th>
        ......
    tr>
    thead>
    <tbody>
    <tr>
        <td>数据td>
        ......
    tr>
    ......
    tbody>
table>
```

2.数据列表—搜索

```
<div class="pull-left layui-form-pane timo-search-box">
    <div class="layui-inline">
        <label class="layui-form-label">标题label>
        <div class="layui-input-block">
            [[表单控件]]
        div>
    div>
    .......
    <div class="layui-inline">
        <button class="layui-btn timo-search-btn">
            <i class="fa fa-search">i>
        button>
    div>
div>
```

异步请求

1.GET异步请求

触发事件class类为 `.ajax-get` ，可设置触发前提示信息！

```
<a class="ajax-get" data-msg='提示信息' href="地址">GETa>
```

2.提交表单

触发事件class类为 `.ajax-submit` ，必须使用
包裹

```
<form action="地址">
    <button class="ajax-submit">提交buttom>
form>
```

3.iframe弹出层

触发事件class类为 `.open-popup` ，必须设置url地址

```
<button class="open-popup" data-title="标题" data-url="url地址" data-size="宽,高">
弹出button>
```

4.列表多选操作

触发事件class类为 `.open-popup-param` ，必须设置url地址，触发后将选择的列表ID组作为参数传递

```
<button class="open-popup-param" data-title="标题"  data-url="url地址"  data-size
="宽,高">
按钮button>
```

视图模板

模板布局

通用模板

```
<html xmlns:th="http://www.thymeleaf.org">
<head th:replace="/common/template :: header(~{::title},~{::link},~{::style})">
head>
<body class="layui-layout-body">

    <script th:replace="/common/template :: script">script>
body>
html>
```

在模板页面中直接引用该模型，参数如下：

- ~{::title}

模板片段

分页片段

在页面中直接引入配合控制层使用，控制器需传入 `page` 变量

```
<div th:replace="/common/fragment :: page">div>
```

日志展示片段

日志展示片段一般在详细页面中使用，用于展示该条记录的日志信息，需要传入实体对象才可获取该条记录日志信息。

```
<div th:replace="/common/fragment :: log(${实体对象})">div>
```

自定义方法

> 项目自定义的Thymeleaf方法，在项目 `com.linln.admin.core.thymeleaf.utility` 目录下

1. 字典提取方法

字典提取方法

1.获取选项集合

获取字典类型为 `键值对` 的字典数据，返回数据为Map集合

```
#dicts.value(字典标识)
```

2.获取选项编码的值

获取字典类型为 `键值对` 的字典数据，返回数据为字符串值

```
#dicts.keyValue(字典标识,选项编码)
```

3.获取数据状态

传入数据的状态字段值，获取状态字典对应信息

```
#dicts.dataStatus(状态)
```

自定义标签

> 项目自定义的Thymeleaf标签，在项目 `com.linln.admin.core.thymeleaf.attribute` 目录下，如果需要dialect约束，可引入 `xmlns:mo="https://gitee.com/aun/Timo"`

生成下拉列表—数组和集合

根据 `数据和集合` 变量生成对应的下拉列表选择器

```
<select  mo:list="数组或集合变量" >select>
```

功能属性：

1. mo-selected 默认选中下拉选项，传入值（可选）
2. mo-empty 空值选项名称（可选）

生成下拉列表—字典标识

根据 `字典标识` 变量生成对应的下拉列表选择器

```
<select  mo:dict="字典标识" >select>
```

功能属性：

1. mo-selected 默认选中下拉选项，传入值（可选）
2. mo-empty 空值选项名称（可选）

获取当前用户信息

根据 `用户字段` 获取用户信息

```
<div mo:user="用户字段">div>
```

后台手册

使用分页

控制器定义

在Controller中返回分页对象Page变量名为 `page` ，下面代码为字典列表实例！

```
public String index(Model model, Dict dict){

    // 创建匹配器，进行动态查询匹配
    ExampleMatcher matcher = ExampleMatcher.matching().
            withMatcher("title", match -> match.contains());

    // 获取字典列表
    Example<Dict> example = TimoExample.of(dict, matcher);
    Page<Dict> list = dictService.getPageList(example);

    // 封装数据
    model.addAttribute("list",list.getContent());
    model.addAttribute("page",list);
    return "/system/dict/index";
}
```

service定义

在业务层中可通过PageSort创建一个分页请求对象，也可自行创建分页请求对象，下面代码为字典业务层实例！

```
public Page<Dict> getPageList(Example<Dict> example) {
    // 创建分页对象
    PageRequest page = PageSort.pageRequest();
    return dictRepository.findAll(example, page);
}
```

视图模板定义

```
<div th:replace="/common/fragment :: page">div>
```

字典使用

字典介绍

本项目字典系统分为三种类型：字符类型、键值对类型、枚举类型。

- 字符类型：一个字典标识对应一个字典值。
- 键值对类型：可以设置多条键值数据，通过key获取对应的值。
- 枚举类型：保存枚举类的路径，通过反射获取枚举类中的数据。

使用字典

在模板中使用字典，请转到 字典提取方法

文件上传

图片上传

已内置图片上传接口 `/upload/image` ，上传字段名为 `image` ，成功后返回上传文件JOSN数据。

特性：

- 图片上传后保存其MD5值及SHA1值，可以进行部分特殊的使用，但会损失部分性能。
- 通过SHA1值判断图片是否存在服务器中，如果存在将不再保存新上传的图片！

文件上传

文件上传使用spring mvc内置的MultipartFile上传组件，配合FileUpload工具类使用，可以通过配置文件 `project.static-path-pattern` 指定文件的保存路径。

示例代码如下：

```
@ResponseBody
public ResultVo uploadFile(@RequestParam("xxx") MultipartFile multipartFile) th
rows IOException {

    // 创建Upload实体对象
    Upload upload = FileUpload.getFile(multipartFile, "/xxx");

    // 保存文件到指定路径
    multipartFile.transferTo(FileUpload.getDestFile(upload));

    // 保存文件上传信息
    uploadService.save(upload);
    return ResultVoUtil.success(upload);
}
```

行为日志

注解介绍

> 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，行为日志通过APO注解方式实现，只需将标注在方法上即可实现日志记录。

注解对象介绍——@ActionLog

1. name属性——日志名称，也可以在Action行为类中添加
2. message属性——日志消息，也可以在Action行为类中添加
3. key属性——行为key，在Action行为类中定义
4. action属性——Action行为类

保存行为

```
@ActionLog(name = "日志名称", message = "提示：${字段}", action = SaveAction.class)
```

标注在控制器保存方法上即可记录保存信息，日志内容格式为：添加/更新+message，可以使用${字段}占位符。标注的方法中参数对象必须添加 `Object entity` 字段。

状态更改行为

```
@ActionLog(name = "日志名称", action = StatusAction.class)
```

标注在控制器 `status` 方法上，标注的方法上必须有 `String param` 和 `List idList` 参数才可记录状态的改变日志。

自定义行为

可参考其他行为类（待续）

导入导出

实体类上注解

需要导入导出功能，首先需要在对应的实体类中加入注解@Excel，可以表标注在类和方法上，类上则作为文件名称，方法上则作为标题名称。

注解对象介绍——@Excel

1. value属性——字段标题名称或文件名称
2. type属性——excel操作类型ExcelType，可指定不同的操作
3. dict属性——字段字典标识，用于导入导出时进行字典转换（只支持导出操作）
4. joinField属性——关联操作实体对象字段名称，用于获取关联数据（只支持导出操作）

工具类使用、

1. 生成模板

```
ExcelUtil.genTemplate(Class<?> entity)
```

```
ExcelUtil.genTemplate(Class<?> entity, String sheetTitle)
```

2. 导入操作

```
ExcelUtil.importExcel(Class<T> entity, InputStream inputStream)
```

3. 导出操作

```
ExcelUtil.exportExcel(Class<?> entity, List<T> list)
```

```
ExcelUtil.exportExcel(Class<?> entity, List<T> list, String sheetTitle)
```

动态查询

Jpa内置查询

jpa内置多种动态查询，常用的有：Example实例查询、Specification复杂查询。

使用文章推荐：https://blog.csdn.net/qq_30054997/article/details/79420141

QuerySpec查询

QuerySpec是基于Specification复杂查询封装的一套查询工具，语法类似于Example实例查询，dao层需要继承JpaSpecificationExecutor接口。示例如下：

```
public Page<User> getPageList(User user, Integer pageIndex, Integer pageSize) {
    // 创建分页对象
    Sort sort = new Sort(Sort.Direction.ASC, "createDate");
    PageRequest page = PageRequest.of(pageIndex-1, pageSize, sort);

    QuerySpec querySpec = QuerySpec.matching()
            .withMatcher("nickname", QuerySpec.LIKE)
            .withIgnorePaths("password");

    return userRepository.findAll(QuerySpec.of(user, querySpec), page);
}
```

1. static matching() ----------创建QuerySpec实例
2. withMatcher("字段名", 查询规则) ----------添加查询匹配器
3. withMatcherIn("字段名", 值列表) ----------添加In查询方式匹配器
4. withMatcherBetween("字段名", 值1, 值2) ----------添加Between查询方式匹配器
5. withIgnorePaths("忽略字段"...) ----------设置需要忽略查询的字段
6. static of(实体对象 ,QuerySpec实例) ----------获取Specification复杂查询实例

注意：QuerySpec查询不支持关联的字段，必须忽略关联的字段！

配置信息

TIMO配置

常用配置

配置项	默认值	说明
project.captcha-open	false	是否开启登录验证码
project.remember-me-timeout	7	cookie记住登录信息时间，默认7天（天）
project.global-session-timeout	1800	Session会话超时时间，默认30分钟（秒）
project.file-upload-path	./upload	文件上传路径，默认在项目根目录upload下
project.static-path-pattern	/upload/**	上传文件静态访问路径

XSS防护

配置项	默认值	说明
project.xss-enabled	true	是否开启XSS防护
project.xss-url-patterns	/*	拦截规则，可通过“,”隔开多个
project.xss-excludes	静态文件地址	忽略规则，可通过“,”隔开多个

工具类

ToolUtil

SpringContextUtil

FormBeanUtil

ToolUtil

通用方法工具类—通用方法工具类

```
com.linln.core.utils.ToolUtil
```

1. getRandomString(int length)——获取随机位数的字符串

> @param length 随机位数

2. lowerFirst(String word)——首字母转小写

> @param word 单词

3. upperFirst(String word)——首字母转大写

> @param word 单词

4.getProjectPath()——获取项目不同模式下的根路径

5.getFileSuffix(String fileName)——获取文件后缀名

> @param fileName 文件名

6.enumToMap(Class enumClass)——将枚举转成List集合

> @param enumClass 枚举类

7.根据枚举code获取枚举对象——根据枚举code获取枚举对象

> @param enumClass 枚举类
> @param code code值

SpringContextUtil

获取Spring的ApplicationContext对象工具，可以用静态方法的方式获取spring容器中的bean

```
com.linln.core.utils.SpringContextUtil
```

1.getApplicationContext()——获取applicationContext对象

2.getBean(String name)——通过name获取 Bean

3.getBean(Class clazz)——通过class获取Bean

4.getBean(Class clazz)——通过class获取Bean

5.getBean(String name,Class clazz)——通过name,以及Clazz返回指定的Bean

6.getEnvironmentProperty(String key)——获取配置文件配置项的值

> @param key 配置项key

FormBeanUtil

复制表单验证对象数据—忽略部分字段

```
com.linln.core.utils.FormBeanUtil
```

1.copyProperties(Object source, Object target)——复制对象

> @param source 源对象
> @param target 目标对象

2.copyProperties(Object source, Object target, String... ignoreProperties)——复制对象，自定义忽略Bean属性

> @param source 源对象
> @param target 目标对象

常见问题

1.导入时方法报错

如果导入项目后get/set等方法报错不存在，请在您的IDE编辑器中安装 `Lombok` 插件，安装教程请参考使用手册。

```
// 判断验证码是否正确
ProjectProperties properties = SpringContextUtil.getBean(ProjectProperties.class)
if (properties.isCaptchaOpen()) {
    Session session = SecurityUtils.getSubject().getSession();
    String code = (String) session.getAttribute( key: "captcha");
    if (StringUtils.isEmpty(captcha) || StringUtils.isEmpty(code)
            || !captcha.toUpperCase().equals(code.toUpperCase())) {
        throw new ResultException(ResultEnum.USER_CAPTCHA_ERROR);
    }
    session.removeAttribute( key: "captcha");
}
```

2.单元测试出错

由于本项目将入口文件放在boot模块中，如果进行单元测试时提示：@SpringBootConfiguration入口文件不存在，请将测试文件移至boot模块下。

```
14:07:16.111 [main] DEBUG org.springframework.core.io.support.PathMatchingResourcePatternResolver - Looking for matching resources in
14:07:16.115 [main] DEBUG org.springframework.core.io.support.PathMatchingResourcePatternResolver - Looking for matching resources in
14:07:16.127 [main] DEBUG org.springframework.core.io.support.PathMatchingResourcePatternResolver - Looking for matching resources in
14:07:16.144 [main] DEBUG org.springframework.core.io.support.PathMatchingResourcePatternResolver - Looking for matching resources in
14:07:16.185 [main] DEBUG org.springframework.core.io.support.PathMatchingResourcePatternResolver - Resolved location pattern [classp
java.lang.IllegalStateException: Unable to find a @SpringBootConfiguration, you need to use @ContextConfiguration or @SpringBootTest(

    at org.springframework.util.Assert.state(Assert.java:73)
    at org.springframework.boot.test.context.SpringBootTestContextBootstrapper.getOrFindConfigurationClasses(SpringBootTestContextBoo
    at org.springframework.boot.test.context.SpringBootTestContextBootstrapper.processMergedContextConfiguration(SpringBootTestContex
    at org.springframework.test.context.support.AbstractTestContextBootstrapper.buildMergedContextConfiguration(AbstractTestContextBo
    at org.springframework.test.context.support.AbstractTestContextBootstrapper.buildDefaultMergedContextConfiguration(AbstractTestCo
    at org.springframework.test.context.support.AbstractTestContextBootstrapper.buildMergedContextConfiguration(AbstractTestContextBo
    at org.springframework.test.context.support.AbstractTestContextBootstrapper.buildTestContext(AbstractTestContextBootstrapper.java
    at org.springframework.boot.test.context.SpringBootTestContextBootstrapper.buildTestContext(SpringBootTestContextBootstrapper.jav
    at org.springframework.test.context.TestContextManager.<init>(TestContextManager.java:139)
    at org.springframework.test.context.TestContextManager.<init>(TestContextManager.java:124)
    at org.springframework.test.context.junit4.SpringJUnit4ClassRunner.createTestContextManager(SpringJUnit4ClassRunner.java:151)
    at org.springframework.test.context.junit4.SpringJUnit4ClassRunner.<init>(SpringJUnit4ClassRunner.java:142)
```