**TonFun快速开发系统架构体系文档**

版本：1.1

|  |  |  |  |
| --- | --- | --- | --- |
| **作者：** | 同丰技术小分队 | **日期：** | 2019/09/24 |
| **审批：** |  | **日期：** |  |

**修改记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **修改内容** | **修改人** | **修改日期** |
| 1.0 | TonFun快速开发系统架构体系文档初始化 | 郭达 | 2019/09/24 |
| 1.1 | 调整框架文件结构、增加资源控制、增加数据权限、增加常见问题、在线预览 | 郭达 | 2019/10/31 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

***目录***

[**一、 快速了解 7**](#_Toc21161)

[1． 简介 7](#_Toc7907)

[2． 主要特性 7](#_Toc26037)

[3． 技术选型 7](#_Toc6446)

[4． 内置功能 8](#_Toc1954)

[5． 更新日志 9](#_Toc32665)

[**二、 环境部署 10**](#_Toc29673)

[1． 准备工作 10](#_Toc20428)

[2． 下载项目 10](#_Toc26973)

[3． 导入项目 12](#_Toc19948)

[4． 运行系统 14](#_Toc8613)

[5． 部署系统 15](#_Toc8892)

[**三、 项目介绍 16**](#_Toc26983)

[1． 文件结构 16](#_Toc15236)

[2． 核心技术 21](#_Toc1792)

[**四、 后台手册 24**](#_Toc646)

[1． 分包 24](#_Toc19383)

[2． 分页实现 25](#_Toc31689)

[3． 事务管理 27](#_Toc19407)

[4． 异常处理 30](#_Toc29053)

[5． 权限与校验 33](#_Toc23524)

[6． 代码生成 34](#_Toc29771)

[7． 枚举使用 37](#_Toc9846)

[8． 热部署使用 39](#_Toc22341)

[9． 缓存使用 42](#_Toc661)

[10． 静态模版使用 43](#_Toc19552)

[11． Spring profile 46](#_Toc15614)

[12． 工具类使用 48](#_Toc11667)

[13． 数据权限 53](#_Toc20280)

[14． 资源控制 54](#_Toc8207)

[**五、 前端手册 56**](#_Toc7272)

[1． 布局 58](#_Toc27188)

[2． 参数使用 59](#_Toc18935)

[3． \_container公用文件使用 59](#_Toc23084)

[4． 弹出框的使用 61](#_Toc14913)

[5． 树形结构渲染 61](#_Toc19025)

[**六、 扩展与高级配置 62**](#_Toc8679)

[1． 重命名项目名称 62](#_Toc31173)

[2． 重命名包名称 64](#_Toc13431)

[3． 更换登录页背景样式 66](#_Toc25329)

[4． 切换Swagger样式效果 66](#_Toc2947)

[5． Mybatis-plus通用枚举 66](#_Toc18109)

[6． 多模块 66](#_Toc17259)

[7． Maven私服 66](#_Toc6847)

[8． 静态资源和模板位置的变更 66](#_Toc25434)

[9． 推荐插件 66](#_Toc14775)

[**七、 常见问题汇总 67**](#_Toc4679)

[1． 放行权限验证 67](#_Toc28942)

[2． 关于 ${ctxPath} 67](#_Toc26884)

[3． Maven下载包慢 68](#_Toc6965)

[4． 事务不生效 69](#_Toc29074)

[5． 关于@layout 70](#_Toc13850)

## 快速了解

### 简介

TonFun是一个Java EE企业级快速开发平台，基于经典技术组合（Spring Boot、Apache Shiro、MyBatis-PlUS、Beetl、Layui），内置模块如：用户管理、角色管理、岗位管理、部门管理、菜单及按钮授权、代码生成等。

### 主要特性

* 强大的一键生成功能（包括控制器、模型、视图、菜单、sql等），简化重复性工作
* 支持按钮及数据权限、让数据更可控
* 基于SpringBoot，简化了大量项目配置和maven依赖，独特的分包方式，代码多而不乱
* 利用beetl模板引擎对前台页面进行封装和拆分，使臃肿的html代码变得简洁，更加易维护
* 对常用js插件进行二次封装，使js代码变得简洁，更加易维护
* 利用ehcache框架对经常调用的查询进行缓存，提升运行速度
* 控制器层统一的异常拦截机制，利用@ControllerAdvice统一对异常拦截，
* 页面统一的js key-value单例模式写法，每个页面生成一个唯一的全局变量，提高js的利用效率，并且有效防止多个人员开发引起的函数名/类名冲突，并且可以更好地去维护代码

### 技术选型

#### 系统环境

* JDK >= 11
* Maven >= 4
* Mysql >= 5.8
* Intellij IDEA >= 2019

#### 主框架

* Spring-Boot >= 2.1.6.RELEASE
* Lombok >= 1.18.8
* Shiro >= 1.4.1
* Ehcache >= 2.10.6

#### 持久层

* Mybatis-plus >= 3.1.2
* PageHelper >= 4.1.3

#### 视图层

* Beetl >= 3.0.10.RELEASE
* Layui >= 2.4.5
* Jquery >= 3.2.1

### 内置功能

* 用户管理： 用户是系统操作者，该功能主要完成系统用户配置。
* 部门管理： 配置系统组织机构（公司、部门、小组），树结构展现支持按钮权限。
* 岗位管理： 配置系统用户所属担任职务，支持数据权限。
* 菜单管理： 配置系统菜单，操作权限，按钮权限标识等。
* 角色管理： 角色菜单权限分配、设置角色按机构进行数据范围权限划分。
* 代码生成： 前后端代码的生成（java、html、xml、sql)
* 假日管理： 配置节假日类型供日历使用。
* 附件管理： 提供多附件上传，同时为了提高办公效率提供在线预览查看。

### 更新日志

* V1.3 提高代码质量、可读性及安全 2019-10-31
  + 框架整体更改为面向接口编程
  + 完善接口可读性
  + 根据原注解提供后端资源控制
  + 前端已知BUG修复
  + 根据岗位提供数据权限
  + 提供在线预览功能
  + 完善快速开发使用手册V1.2
  + 统一表单验证扩展
  + 增加用户登录&退出日志模块
  + 完善代码生成模版
* V1.2 TonFun系统-动态代码生成 2019-09-23
  + 页面提供动态配置代码生成
  + 支持用户多角色
  + 提供岗位功能
  + 采用layui-icon实现可配置菜单图表icon
  + 完善java开发规范
  + 完善统一注解规范
  + 提供框架使用手册
  + 提供工具类使用手册
* V1.1 TonFun系统-权限  2019-08-09
  + 集成shiro权限校验
  + 利用ehcache框架对经常调用的查询进行缓存，提升运行速度
  + 全局统一的异常拦截机制，ControllerAdvice统一对异常拦截
  + 采用MyBatis-Plus3.0强大构造器Wrapper方式进行数据库操作
  + 完善系统的session机制，增加session失效时间的跳转
  + 通过MultipartFile实现上传功能，大大简化上传代码简洁
  + 代码生成可同步生成sql语句INSERT菜单项
* V1.0 TonFun系统-初始化  2019-08-07
  + 提供用户，角色，菜单，部门，代码生成等五大功能
  + 简单可操作，易拓展的代码生成体系
  + 利用beetl模板引擎对前台页面进行封装和拆分,使臃肿的html代码变得简洁
* 从0到1

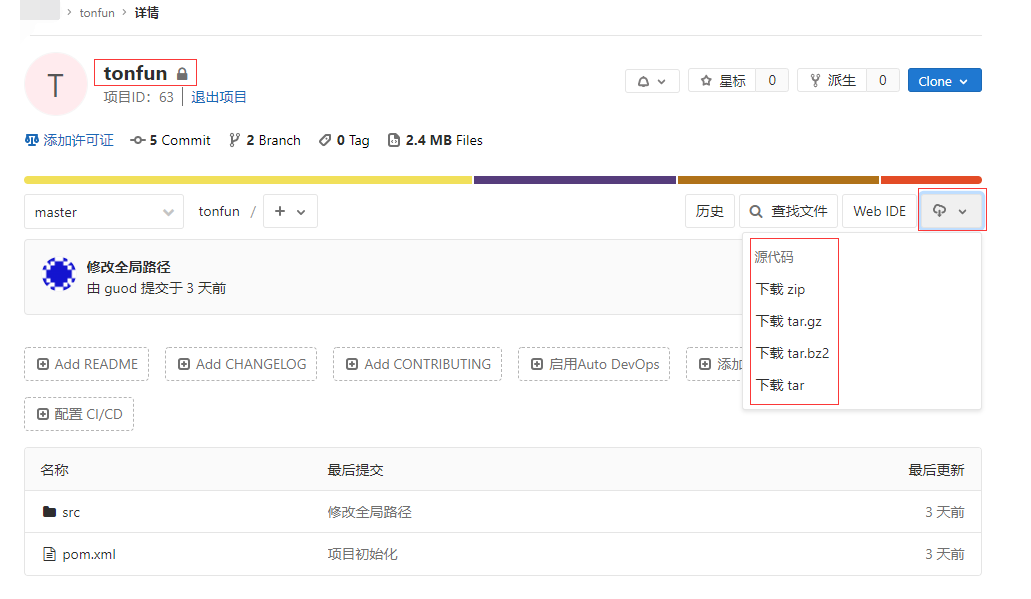
## 环境部署

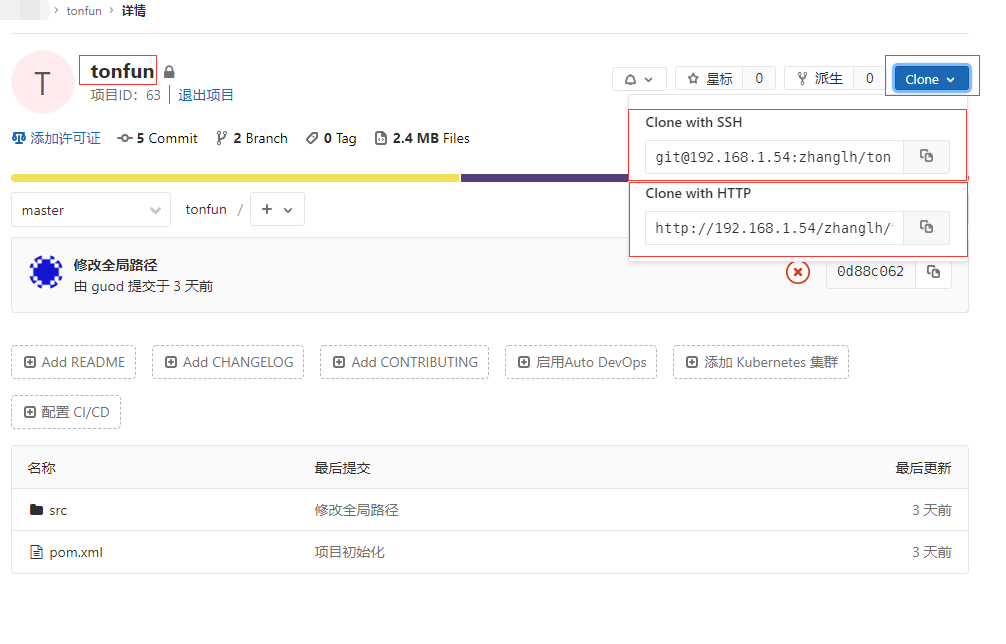
### 准备工作

* JDK >= 11
* Maven >= 4
* Mysql >= 5.8

### 下载项目

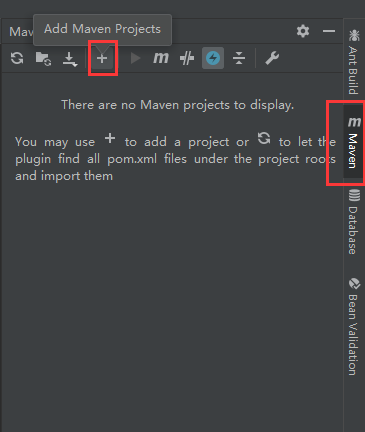
下载项目有N多种方法，在此只提供两种方式经供参考。首先你需要有公司内部得GitLab账号，同时该账号有tonfun项目得一系列权限，才可进行下载。



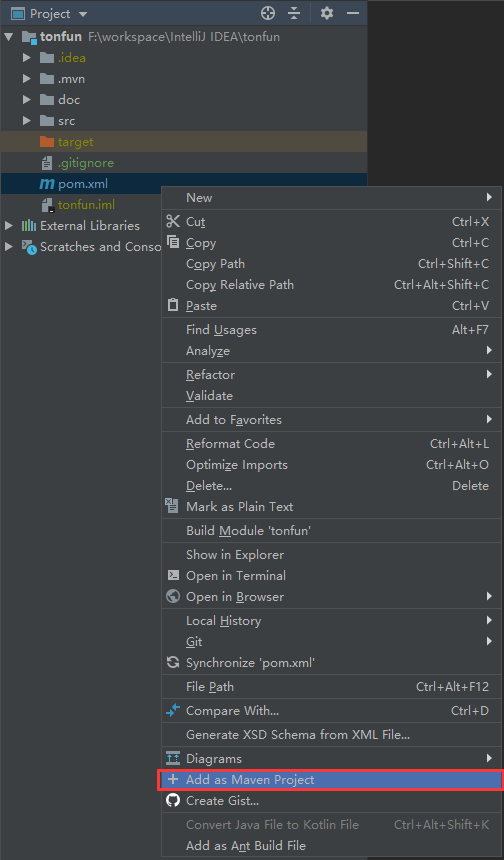


### 导入项目

* 进入IDEA主界面，点击open，并选择下载好的TonFun项目
* 进入IDEA之后，需要添加Maven项目，导入方式有两种。如下
  + 右侧选择Maven，选择 add maven project，弹框中选择该项目的pom.xml文件 ，即可完成导入



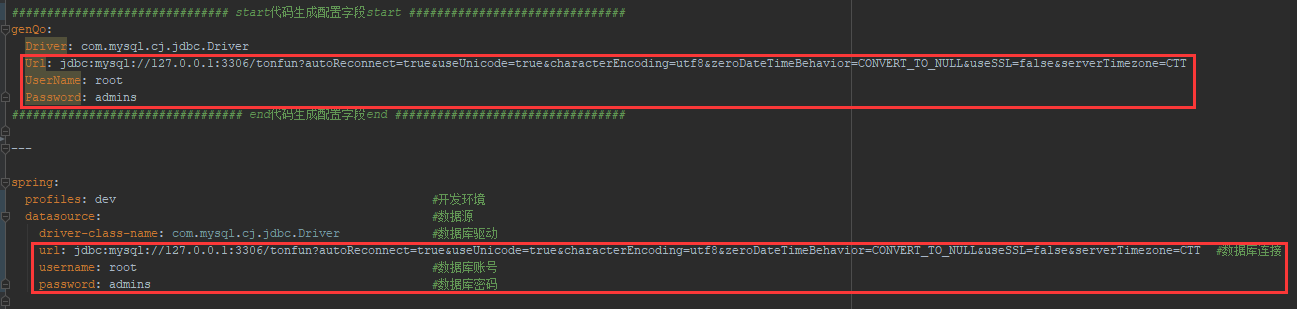
* + 进入IDEA之后，右击pom.xml，选择add as maven project，即可完成导入



### 运行系统

#### 执行doc/TonFun数据库.sql脚本，初始化tonfun数据库环境

#### 打开application.yml配置文件，修改数据连接、账号、密码，改为您所连接数据库的配置



#### 如需修改服务器端口



#### 执行TonfunApplication类中的main方法，即可运行TonFun系统

#### 打开浏览器，输入localhost，即可访问到TonFun的登录页面，默认登录账号密码:

#### tonfun/123456

### 部署系统

### 目前TonFun支持两种打包方式，即jar包和war包

#### 打包之前增加pom.xml中的packaging节点，jar 或者 war

#### 

#### 采用Maven进行打包

#### 

### 通过命令行也可进行打包clean package -Dmaven.test.skip=true

#### 命令执行成功后，在target目录下即可看到打包好的文件

#### 

### 提示：若打的包为jar包，可通过java -jar tonfun-1.2.jar 来启动系统

## 项目介绍

### 文件结构

#### 1、简洁文件结构

#### │src.main

├─ java

│ └─ com.example.tofun

│ └─ config · · · · · · · · · // 全局配置

│ │ └─ constant · · · · · · · · // 常量配置

│ │ └─ generator · · · · · · · · // 代码生成配置

│ │ │ └─ actionr · · · · · · · // Mybatis生成策略

│ │ │ └─ engine · · · · · · · // TonFun生成策略

│ │ └─ properties · · · · · · · · // 系统配置

│ │ └─ web · · · · · · · · · // 前端控制

│ └─ core · · · · · · · · · · // 核心控制

│ │ └─ annotation · · · · · · · // 自定义注解

│ │ └─ aop · · · · · · · · · // 切面配置

│ │ └─ base · · · · · · · · · // 基础配置

│ │ └─ cache · · · · · · · · · // 缓存配置

│ │ └─ exception · · · · · · · · // 通用异常

│ │ └─ factory · · · · · · · · // 通用工厂

│ │ └─ listener · · · · · · · · // 监听器

│ │ └─ node · · · · · · · · ·// 通用节点

│ │ └─ shiro · · · · · · · · ·// 权限控制

│ │ └─ state · · · · · · · · ·// 通用状态

│ │ └─ util · · · · · · · · ·// 工具类处理

│ └─ modular · · · · · · · · ·// 业务模块

├─ resources

│ └─ lib · · · · · · · ····// 外部jar包

│ └─ templates · · · · · · · ···// 生成代码静态模版

└─ webapp

└─ static · · · · · · · ····// 页面静态资源js、css等

│ └─ common · · · · · · · ··// 公共业务模块静态资源

│ └─ modular · · · · · · · ··// 业务模块静态资源

└─ templates · · · · · · · ···// 页面Beetl模版

└─ common · · · · · · · ··// 公共业务模块页面

└─ modular · · · · · · · ··// 业务模块页面

#### 2、清晰文件结构

│src.main

├─ java

│ └─ com.example.tofun

│ └─ config · · · · · · · · · // 全局配置

│ │ └─ constant · · · · · · · · // 常量配置

│ │ │ └─ CacheConsts.java · · · · · // 缓存名称常量配置

│ │ │ └─ CacheKeyConsts.java · · · · // 缓存标识前缀常量配置

│ │ │ └─ ConfigConsts.java · · · · · // 系统常量配置

│ │ └─ generator · · · · · · · · // 代码生成配置

│ │ │ └─ action · · · · · · · // Mybatis生成策略

│ │ │ │ └─ config · · · · · · // Mybatis

│ │ │ │ │ └─ AbstractGeneratorConfig.java · // Mybatis-plus抽象配置

│ │ │ │ │ └─ WebGeneratorConfig.java · // Mybatis-plus代码生成配置

│ │ │ │ └─ model · · · · · · // 模型

│ │ │ │ └─ GenQo.java · · · · // 生成代码参数

│ │ │ │ └─ IsMenu.java · · · · // 生成菜单枚举

│ │ │ └─ engine · · · · · · · // TonFun生成策略

│ │ │ └─ base · · · · · · · // TonFun基础生成

│ │ │ │ └─ AbstractTemplateEngine.java · // TonFun模版生成父类

│ │ │ │ └─ SimpleTemplateEngine.java · // TonFun通用模版生成引擎

│ │ │ │ └─ TonFunTemplateEngine.java · // TonFun项目模版生成引擎

│ │ │ └─ config · · · · · · // TonFun生成配置

│ │ │ └─ ControllerConfig.java · · // Controller模版生成配置（无效）

│ │ │ └─ MapperConfig.java · · · // Mapper模版生成配置（无效）

│ │ │ └─ PageConfig.java · · · // 页面模版生成配置

│ │ │ └─ ServiceConfig.java · · · // Service模版生成配置（无效）

│ │ │ └─ SqlConfig.java · · · · // SQL模版生成配置

│ │ └─ properties · · · · · · · · // 系统配置

│ │ │ └─ BeetlProperties .java · · · · · // Beetl配置项

│ │ │ └─ TonFunProerties.java · · · · // TonFun配置

│ │ └─ web · · · · · · · · · // 前端控制

│ │ └─ BeetlConfig.java · · · · · // Beetl配置初始化控制

│ │ └─ BeetlConfiguration.java · · · · // Beetl全局工具类控制

│ │ └─ CoresConfig.java · · · · · // 跨域控制

│ │ └─ PageHelperConfig.java · · · · // 分页控制

│ │ └─ ShiroConfig.java · · · · · // 权限控制

│ │ └─ TonFunUserFilter.java · · · · // TonFun访问控制

│ └─ core · · · · · · · · · · // 项目核心运行机制

│ │ └─ annotation · · · · · · · // 自定义注解

│ │ │ └─ DataScope.java · · · · · · // 数据权限注解

│ │ │ └─ Permission.java · · · · · · // 资源空值注解

│ │ └─ aop · · · · · · · · · // 切面配置

│ │ │ └─ DataScopeAspect.java · · · · // 数据权限注解实现类

│ │ │ └─ GlobalExceptionHandler.java · · · // 全局异常拦截器

│ │ │ └─ PermissionAop.java · · · · · // 资源控制注解实现类

│ │ └─ base · · · · · · · · · // 基础配置

│ │ │ └─ BaseController.java · · · · · // 基础Controller类

│ │ │ └─ BaseResult.java · · · · · · // 返回结果封装

│ │ │ └─ BaseResultPage.java · · · · · // 分页返回结果封装

│ │ │ └─ ErrorEnum.java · · · · · · // 错误码枚举类

│ │ └─ cache · · · · · · · · · // 缓存配置

│ │ │ └─ EhCacheConfig.java · · · · · // Ehcache配置

│ │ └─ exception · · · · · · · · // 通用异常

│ │ │ └─ BusinessException.java · · · · // 异常处理封装类

│ │ │ └─ ServiceException.java · · · · // 异常处理类

│ │ └─ factory · · · · · · · · // 通用工厂

│ │ │ └─ CacheFactory.java · · · · // 常用生产接口

│ │ │ └─ ICacheFactory.java · · · · // 常用生产工厂接口

│ │ └─ listener · · · · · · · · // 监听器

│ │ │ └─ ConfigListener.java · · · · · // ServletContext监听器

│ │ └─ node · · · · · · · · · // 通用节点

│ │ │ └─ MenuNode.java · · · · · // 菜单节点

│ │ │ └─ ZTreeNode.java · · · · · · // Ztree节点

│ │ └─ shiro · · · · · · · · · // 权限控制

│ │ │ └─ service · · · · · · · // 权限服务类

│ │ │ │ └─ impl · · · · · · · // 权限实现类

│ │ │ │ │ └─ PermissionCheckServiceImpl.java // 权限检查实现类

│ │ │ │ │ └─ UserAuthServiceImpl.java · // 权限认证服务实现类

│ │ │ │ └─ PermissionCheckService.java · · // 权限检查

│ │ │ │ └─ UserAuthService.java · · · // 权限认证服务类

│ │ │ └─ ShiroDbRealm.java · · · · · // 权限认证

│ │ │ └─ ShiroKit.java · · · · · · // 权限工具处理

│ │ │ └─ ShiroUser.java · · · · · · // 权限用户

│ │ └─ state · · · · · · · · · // 状态枚举

│ │ │ └─ ErrorEnum.java · · · · · · // 错误码枚举类

│ │ │ └─ LogSucceedEnum.java · · · · // 成功日志枚举类

│ │ │ └─ LogTypeEnum.java · · · · · // 日志类型枚举类

│ │ │ └─ TypeStatesEnum.java · · · · // 公共状态枚举类

│ │ └─ util · · · · · · · · · // 工具类处理

│ │ └─ CacheUtil.java · · · · · · // 缓存工具类

│ │ └─ DateTimeUtil.java · · · · · // 日期时间工具

│ │ └─ FileUtil.java · · · · · · // 文件上传工具类

│ │ └─ HttpContext.java · · · · · // 请求上下文工具类

│ │ └─ SpringContextUtils.java · · · · // ApplicationContext工具类

│ │ └─ ToolUtil.java · · · · · · // 通用工具类

│ └─ modular · · · · · · · · · // 业务模块

│ │ └─ system · · · · · · · · // 系统管理模块

│ └─ TonfunApplication.java · · · · · · // 项目启动类

├─ resources

│ └─ lib · · · · · · · · · · · // 外部jar

│ │ └─ aspose.slides-19.3.jar · · · · · · // PPT解析包

│ │ └─ aspose-cells-8.5.2.jar · · · · · · // Excel解析包

│ │ └─ aspose-words-14.9.0-jdk16.jar · · · · · // words解析包

│ └─ templates · · · · · · · · · · // 生成代码静态模版

│ │ └─ controller.java.btl · · · · · · · // 控制器模版

│ │ └─ entity.java.btl · · · · · · · · // 实体模版

│ │ └─ mapper.java.btl · · · · · · · · // Mapper接口模版

│ │ └─ mapper.xml.btl · · · · · · · · // Xml模版

│ │ └─ men\_sql.sql.btl · · · · · · · · // SQL模版

│ │ └─ page.html.btl · · · · · · · · // 首页模版

│ │ └─ page.js.btl · · · · · · · · · // 首页JS模版

│ │ └─ page\_add.html.btl · · · · · · · // 添加模版

│ │ └─ page\_add.js.btl · · · · · · · · // 添加JS模版

│ │ └─ page\_detail.html.btl · · · · · · · // 详情模版

│ │ └─ page\_detail.js.btl · · · · · · · // 详情JS模版

│ │ └─ page\_edit.html.btl · · · · · · · // 编辑模版

│ │ └─ page\_edit.js.btl · · · · · · · · // 编辑JS模版

│ │ └─ service.java.btl · · · · · · · · // 服务类模版

│ │ └─ serviceImpl.java.btl · · · · · · · // 服务实现类模版

│ └─ application.yml · · · · · · · · · // 项目配置

│ └─ banner.txt · · · · · · · · · · // 启动LOGO配置

│ └─ ehcache.xml · · · · · · · · · // 缓存配置

│ └─ license.xml · · · · · · · · · // office解析key值

└─ webapp

└─ static · · · · · · · · · · · // 页面静态资源js、css等

│ └─ common · · · · · · · · · // 公共业务模块静态资源

│ └─ css · · · · · · · · · // 通用css样式

│ └─ images · · · · · · · · // 通用图片

│ └─ file · · · · · · · · // 文件缩略图

│ └─ portrait · · · · · · · // 用户头像

│ └─ js · · · · · · · · · // 通用js

│ └─ plugins · · · · · · · · // 公共插件

│ └─ ax · · · · · · · · // Jquery中aJax插件

│ └─ city-picker · · · · · · · // Layui城市插件

│ └─ dropdown · · · · · · · // Jquery下拉菜单插件

│ └─ dtree · · · · · · · · // Layui树形插件dtree

│ └─ font · · · · · · · · // 公用字体包

│ └─ formSelects · · · · · · // Layui多选框下拉插件

│ └─ func · · · · · · · · // TonFun弹框简化封装插件

│ └─ iconPicker · · · · · · · // Layui图标插件

│ └─ jquery · · · · · · · // Jquer插件

│ └─ layui · · · · · · · · // Layui样式插件

│ └─ notice · · · · · · · // Jquery消息通知插件

│ └─ step-lay · · · · · · · // Layui分步表单插件

│ └─ tableSelect · · · · · · // Layui下拉表格选择器

│ └─ theme · · · · · · · // TonFun主题样式

│ └─ treetable-lay · · · · · · // Layui树形表格treeTable

│ └─ webuploader · · · · · · // layui上传插件

│ └─ ztree · · · · · · · · // Jquery树插件

│ └─ modular · · · · · · · · · // 业务模块

│ └─ frame · · · · · · · · // 框架模块

│ └─ system · · · · · · · · // 系统管理模块

└─ templates · · · · · · · · · · // 页面Beetl模版

└─ common · · · · · · · · · // 公共业务模块页面

│ └─ \_body.html · · · · · · · // 内容页

│ └─ \_container.html · · · · · · · // 公共引用页

│ └─ \_footer.html · · · · · · · // 底部页

│ └─ \_header.html · · · · · · · // 顶部页

│ └─ \_sidebar.html · · · · · · · // 左侧栏页

│ └─ loading.html · · · · · · · // 页面加载loading页

│ └─ tree\_dlg.html · · · · · · · // 弹框tree页

└─ modular · · · · · · · · · // 业务模块

│ └─ frame · · · · · · · · // 框架模块

│ │ └─ console.html · · · · · · // 项目介绍日志页

│ │ └─ message.html · · · · · · // 消息页

│ │ └─ password.html · · · · · · // 修改密码页

│ │ └─ theme.html · · · · · · // 主题页

│ │ └─ user\_info.html · · · · · · // 用户信息页

│ │ └─ welcome.html · · · · · · // 欢迎页

│ └─ system · · · · · · · · // 系统管理模块

└─ 404.html · · · · · · · · · // 404错误页

└─ index.html · · · · · · · · · // 首页

└─ login.html · · · · · · · · · // 登录页

### 核心技术

#### 1、SpringBoot框架

#### 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

#### 优点

#### 编码简单： 推荐使用注解

#### 配置简单： 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

#### 部署简单： 内嵌Tomcat、Jetty、Undertow等web容器，无需以war包形式部署

#### 2、Shiro安全控制

#### 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

#### 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

#### 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

#### 架构

#### Subjec： 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

#### SessionDAODAO： 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

#### 3、Beetl模版

#### 介绍

Beetl目前版本是3.0.10RELEASE,相对于其他java模板引擎，具有功能齐全，语法直观,性能超高，以及编写的模板容易维护等特点。使得开发和维护模板有很好的体验。是新一代的模板引擎。

#### 优点

#### 功能完备： 作为主流模板引擎，Beetl具有相当多的功能和其他模板引擎不具备的功能。适用于各种应用场景，从对响应速度有很高要求的大网站到功能繁多的CMS管理系统都适合。Beetl本身还具有很多独特功能来完成模板编写和维护，这是其他模板引擎所不具有的。

#### 非常简单： 类似Javascript语法和习俗，只要半小时就能通过半学半猜完全掌握用法。拒绝其他模板引擎那种非人性化的语法和习俗。同时也能支持html 标签，使得开发CMS系统比较容易

#### 超高的性能： Beetl 远超过主流java模板引擎性能(引擎性能5-6倍于FreeMarker，2倍于JSP），而且消耗较低的CPU。

#### 易于整合： Beetl能很容易的与各种web框架整合，如Spring MVC，JFinal，Struts，Nutz，Jodd，Servlet等。

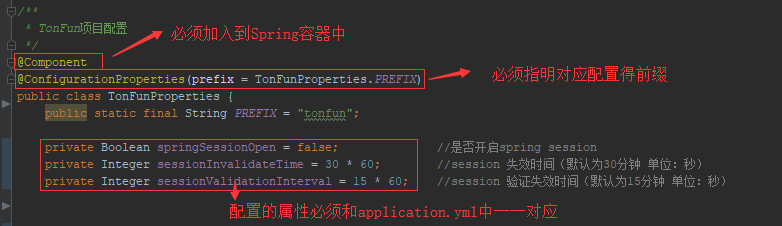
#### 扩展和个性化： Beetl支持自定义方法，格式化函数，虚拟属性，标签，和HTML标签. 同时Beetl也支持自定义占位符和控制语句起始符号也支持使用者完全可以打造适合自己的工具包。模板引擎可以个性化定制，可以扩展为脚本引擎，规则引擎，能定制引擎从而实现高级功能。

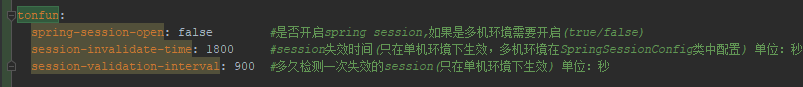
## 后台手册

### 分包

在日常开发中，业务模块的包结构划分一般划分为三个config、core、modular

* config包存放整个模块的配置类，因为项目基于spring boot开发，大部分的spring配置都换成了java bean方式的配置，所以单独分一个包来存放配置， config 包中除了存放配置类，还 有一些以 Properties 结尾的类，这些类的作用是启动应用的时候把 application.yml 中的配 置映射到类的属性上，使用时需要注意以下几点



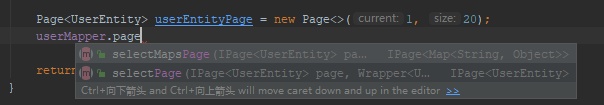


* core包存放当前模块所运行的一些 核心机制，例如全局的异常拦截器，日志AOP，权限的AOP，项目初始化后的监听器，工具类等，还可以存放一些对某些框架的扩展，例如对beetl模板的扩展配置和工具类，对flowable的扩展类，Shiro的一些拓展类等等
* modular存放按业务划分的业务代码，若本模块中包含多个模块业务，则在modular中建立多个业务包，在具体的业务包下再建立entity、controller、service、dao、dto、vo这几个包，其中vo为前后端传输数据所用的属性封装，dto为对返回结果的属性封装。

### 分页实现

#### 简单查询分页

#### 如果查询结果为单表查询，例如查询用户列表，则可以调用Mybatis-Plus自动生成得Mapper中的selectPage()或者selectMapPage()方法，Page类的构造函数中第一个参数为当前查询第几页，第二个参数为每页的记录数。案例如下:



#### 如果不想分页参数传来传去，则可以应用PageHelper插件在Controller中调用默认Page对象。并启动PageHelper.startPage(第几页，条数);执行sql语句查询前一行，错位无效。同时查询结果交由BaseResultPage进行封装。案例如下

#### /\*\*

#### \* 获取用户列表

#### \*/

#### @RequestMapping(value = "/list")

@ResponseBody

public BaseResultPage<UserEntity> list() {

Page page = BaseResultPage.defaultPage();

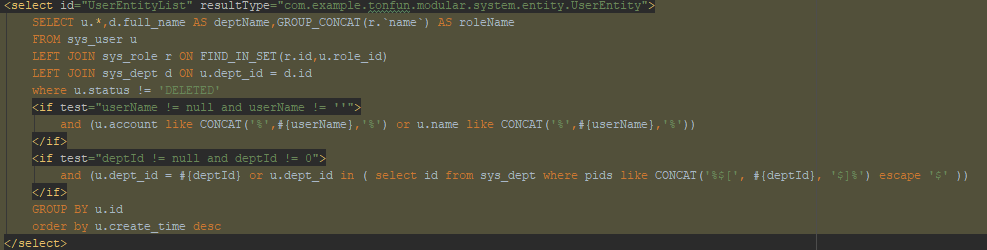
PageHelper.startPage((int) page.getCurrent(), (int) page.getSize());

List<UserEntity> userEntities = userService.selectUser();

PageInfo pageInfo = new PageInfo<>(userEntities);

return new BaseResultPage().createPage(pageInfo);

}



#### 复杂查询分页

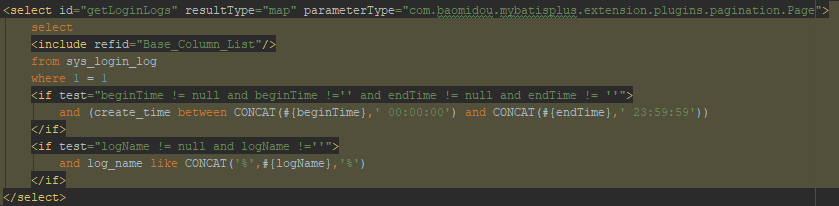
#### 若查询结果是关联多个表得操作，则需要用自定义Mapper，此时得分页操作也很简单,只需要给Mapper的第一个参数设置为 Page 对象即可，我们可以看到在mybatis接口的第一个参数中,传递了Page对象,当mybatis执行此方法的时候，会被mybatis-plus的分页插件自动拦截到，并且把分页查询的结果返回到这个Page对象中。案例如下

#### List<Map<String, Object>> getLoginLogs(@Param("page") Page page,

#### @Param("beginTime") String beginTime,

#### @Param("endTime") String endTime,

#### @Param("logName") String logName);



#### PageHelper同样得方式也支持复杂查询，请前往”简单查询分页”②查看，案例如下

#### /\*\*

#### \* 获取用户列表

#### \*/

#### @RequestMapping(value = "/list")

@ResponseBody

public BaseResultPage<UserEntity> list() {

Page page = BaseResultPage.defaultPage();

PageHelper.startPage((int) page.getCurrent(), (int) page.getSize());

List<UserEntity> userEntities = userService.selectUser();

PageInfo pageInfo = new PageInfo<>(userEntities);

return new BaseResultPage().createPage(pageInfo);

}

### 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。

例如用户新增需要插入用户表、用户与角色关联表、用户与部门关联表，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。

#### /\*\*

#### \* 新增用户

\*/

@Transactional

public int insertUser(UserEntity userEntity){

// 新增用户信息

int rows = userMapper.insertUser(userEntity);

// 新增用户角色关联

insertUserAndRole(userEntity);

// 新增用户部门关联

insertUserAndDept(userEntity);

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：

|  |  |
| --- | --- |
| **常量** | **含义** |
| PROPAGATION\_REQUIRED | 如果当前存在事务，则加入该事务；如果当前没有事务，则创建一个新的事务。这是默认值。 |
| PROPAGATION\_REQUIRES\_NEW | 创建一个新的事务，如果当前存在事务，则把当前事务挂起。 |
| PROPAGATION\_SUPPORTS | 如果当前存在事务，则加入该事务；如果当前没有事务，则以非事务的方式继续运行。 |
| PROPAGATION\_NOT\_SUPPORTED | 以非事务方式运行，如果当前存在事务，则把当前事务挂起。 |
| PROPAGATION\_NEVER | 以非事务方式运行，如果当前存在事务，则抛出异常。 |
| PROPAGATION\_MANDATORY | 如果当前存在事务，则加入该事务；如果当前没有事务，则抛出异常。 |
| PROPAGATION\_NESTED | 如果当前存在事务，则创建一个事务作为当前事务的嵌套事务来运行；如果当前没有事务，则该取值等价于TransactionDefinition.PROPAGATION\_REQUIRED。 |

### 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。

#### 统一返回实体定义

@Data

public class BaseResult <T extends Object> implements Serializable {

public boolean success; //调用是否成功

private T content; //结果

private String message; //信息

private Integer code; //错误码

#### /\*\*

#### \* 返回错误消息

\*/

public BaseResult<T> success(String message) {

this.setSuccess(true);

this.setMessage(message);

this.setCode(200);

return this;

}

#### /\*\*

#### \* 返回成功消息

\*/

public BaseResult<T> failure(ErrorEnum e) {

this.setSuccess(false);

this.setMessage(e.getMessage());

this.setCode(e.getCode());

return this;

}

}

@CrossOrigin(maxAge = 3600)

@RequestMapping(produces = "application/json")

public class BaseController {

protected static BaseResult SUCCESS = new BaseResult(true, 200, "请求成功");

}

#### 通用异常

/\*\*

\* @Author: GD

\* @Description: 异常处理类

\* @Date: 2019/8/2 10:33

\*/

public class ServiceException extends RuntimeException {

private ErrorEnum errorEnum;

public ServiceException(ErrorEnum errorEnum) {

super(errorEnum.getMessage());

this.errorEnum = errorEnum;

}

}

/\*\*

\* @Author: GD

\* @Description: 异常处理封装类

\* @Date: 2019/8/2 10:37

\*/

public class BusinessException extends ServiceException {

public BusinessException(ErrorEnum errorEnum) {

super(errorEnum);

}

}

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

#### 我们在 GlobalExceptionHandler 类中做统一异常拦截处 理， GlobalExceptionHandler 类中可以拦截所有控制器执行过程中抛出的异常，若需要拦截 其他包下的异常可以参考 SessionInterceptor 这个类中AOP的写法，来拦截其他特定包的异 常。统一异常拦截的写法注意一下几点

#### 

#### 可能会认为利用异常拦截这种机制，把业务逻辑的错误都用业务异常抛出进入aop的执行器，对性能会有所影响，经过笔者的调研和测试，频繁的抛出异常和try catch不会有性能损耗，主要的性能损耗在catch方法内部，并且在catch内，记录日志比较占用大部分的时间所以，如果是系统特别注重性能等问题，可以把业务异常分为两类，一类是较为频繁抛出的业务异常，一类是较少出现次数的业务异常，第一类异常可以再@ExceptionHandler中不做日志记录，只进行简单的返回操作，第二类可以着重做异常处理，并做结果返回。

### 权限与校验

**1、用户，角色与资源**

#### 用户、角色和资源（或者说权限），这三者的关系是用户对应角色 ，角色对应资源，菜单和所有的按钮都可以看做是资源(或权限 )，把某一个角色赋予相应的资源，那么该角色就会有访问该资源的权限，否则不展示，当角色绑定资源后还需要给用户赋予角色才可以让登录的用户访问相关菜单或按钮。后期会完善后台进行资源控制。

#### 一句话概括: 用户对应角色，角色对应资源

**2、前端页面对权限资源的显示**

#### 在前端页面中，如果增删改查等按钮受权限控制，则我们需要对资源进行一个权限检查，如果 有该资源的权限，才能让该按钮显示，通过 beetl 的 shiro注册方法 即可完成该项的检查。

#### 

#### 其中 shiro.hasPermission() 起到了权限检查的作用，如果有该资源对应的权限，则被检查 的资源显示，若没有该资源的权限，则按钮不显示

### 代码生成

#### TonFun开发三部曲 -> 1.建表 2.代码生成 3.适配业务

#### 下面以一个“订单业务”为例，实战演练如何用TonFun编写简单的增删改查业务

#### 第一步：建表

#### 

DROP TABLE IF EXISTS `buy\_order`;

CREATE TABLE `buy\_order` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT '主键ID',

`name` varchar(255) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '商品名称',

`user\_name` varchar(255) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '用户名称',

`user\_phone` varchar(255) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '用户手机号',

`create\_time` datetime(0) NULL DEFAULT NULL COMMENT '下单时间',

PRIMARY KEY (`id`) USING BTREE

) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8\_general\_ci COMMENT = '订单记录' ROW\_FORMAT = Dynamic;

SET FOREIGN\_KEY\_CHECKS = 1;

#### 第二步：代码生成

#### 登录管理系统，打开代码生成页面，填写如下内容，注意看红线部分内容

#### 

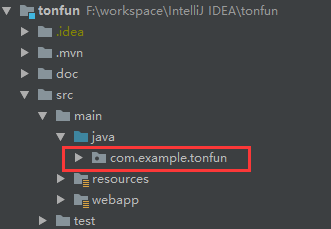
#### 生成代码之后热部署会帮你重新启动项目或是手动重启一下管理系统，生成的代码才可以生效!

#### 下面详细讲解代码生成使用

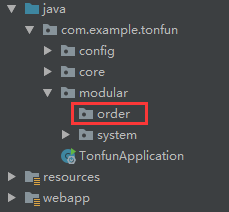
#### 作者： 填写代码生成出的注释上的作者

#### 父级菜单： 此项的选择会影响生成sql添加菜单项的切入点，生成出的sql文件执行后可自动 增加到sys\_menu菜单项，省去手动添加菜单的繁琐

#### 包名称： 一般不需要修改，如下图



#### 模块名称： 对应代码中modular包下的模块名称，如下图，若模块名称填order，则生成出的业务代码会生成到order包下



#### 表前缀： 填写此项会自动移除生成Entity，Controller，Mapper和Service类的名称中包含的重复前缀，例如生成订单记录业务代码时，填写部buy\_，则生成的实体中不会包含Buy前缀名称，若不填写，则生成的实体类为BuyOrder

#### 表描述： 生成业务代码中得名称

#### 选择表： 选择即为生成该表所对应的实体，dao，service等类

#### 生成文件： 选择后生成相应的控制器，实体，service，dao代码等等a

#### 第三部：适配业务

#### 再编写业务代码前还需要给角色分配相关的菜单权限，配置完需再次重新登录才可以把新增的业务显示到菜单上。

#### 

#### 至此，TonFun的开发流程介绍完毕！祝君顺利~！

### 枚举使用

#### 在TonFun中，枚举一般分两类，一种是状态枚举，一种是异常枚举，状态枚举的作用是枚举状态，列出状态的所有值。如下

#### /\*\*

#### \* @Author: GD

#### \* @Description: 用户状态

#### \* @Date: 2019/8/1 15:15

#### \*/

@Getter

public enum TypeStates() {

OK("ENABLE", "启用"),

PROHIBIT("PROHIBIT", "禁用"),

FREEZED("LOCKED", "冻结"),

DELETED("DELETED", "被删除");

String code;

String message;

TypeStates(String code, String message) {

this.code = code;

this.message = message;

}

public static String getDescription(String value) {

if (value == null) {

return "";

} else {

for (TypeStates ms : TypeStates.values()) {

if (ms.getCode().equals(value))

return ms.getMessage();

}

return "";

}

}

}

#### 使用方法如下

TypeStates.OK.getCode()

#### 异常枚举的作用是枚举所有出现的业务异常，例如

#### /\*\*

#### \* Created by GD on 2018/5/2

#### \*/

@Getter

public enum ErrorEnum() {

ERROR\_ILLEGAL\_PARAMS(101, "参数非法"),  
 CANT\_OPERATION\_ADMIN(102, "不能操作系统管理员"),  
 UPLOAD\_ERROR(103, "上传图片出错"),  
     TOKEN\_EXPIRED(104, "token过期"),  
     TOKEN\_ERROR(105, "token验证失败"),  
    NO\_PRIVILEGES(106, "没有该操作权限"),  
     ERROR\_SYSTEM(999, "系统异常"),

private Integer code;

private String message;

private ErrorEnum(Integer code, String message) {

this.code = code;

this.message = message;

}

}

#### 使用方法如下

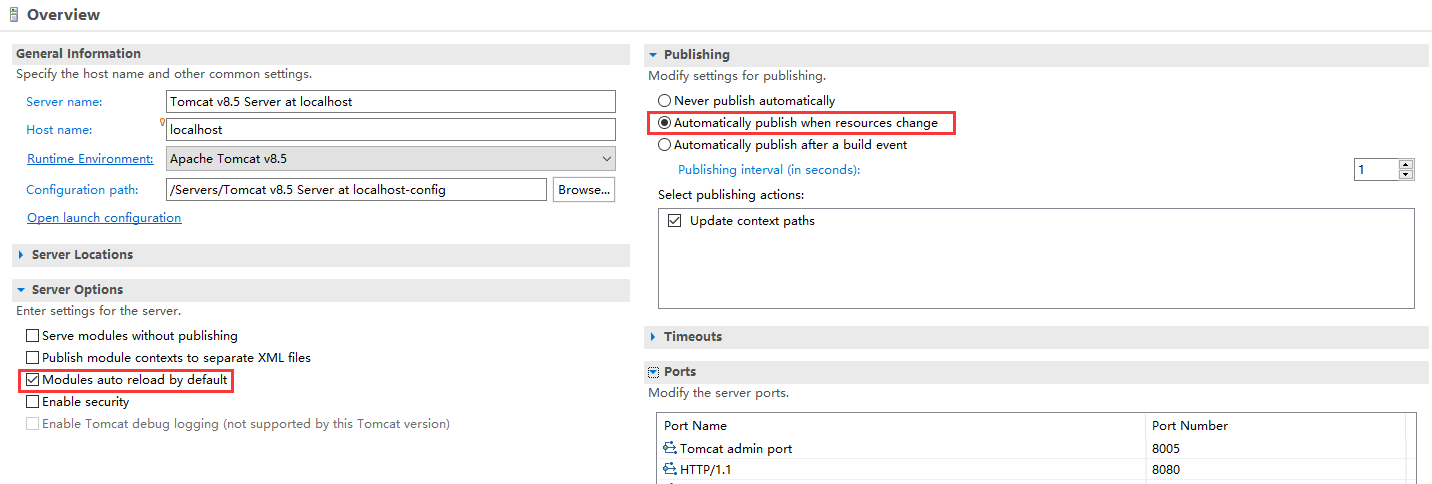
return new BaseResult(ErrorEnum.ERROR\_SYSTEM);

#### 使用枚举可以方便维护一些状态的值和管理所有的业务异常，所以在有状态或者新的业务异常的时候推荐写到枚举里。

### 热部署使用

#### 热部署的两种情况

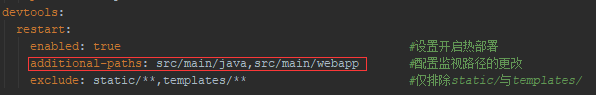
1. 重新加载html
   * 1. 如果是eclipse修改html保存后可以自动替换，如果不能请检查server配置



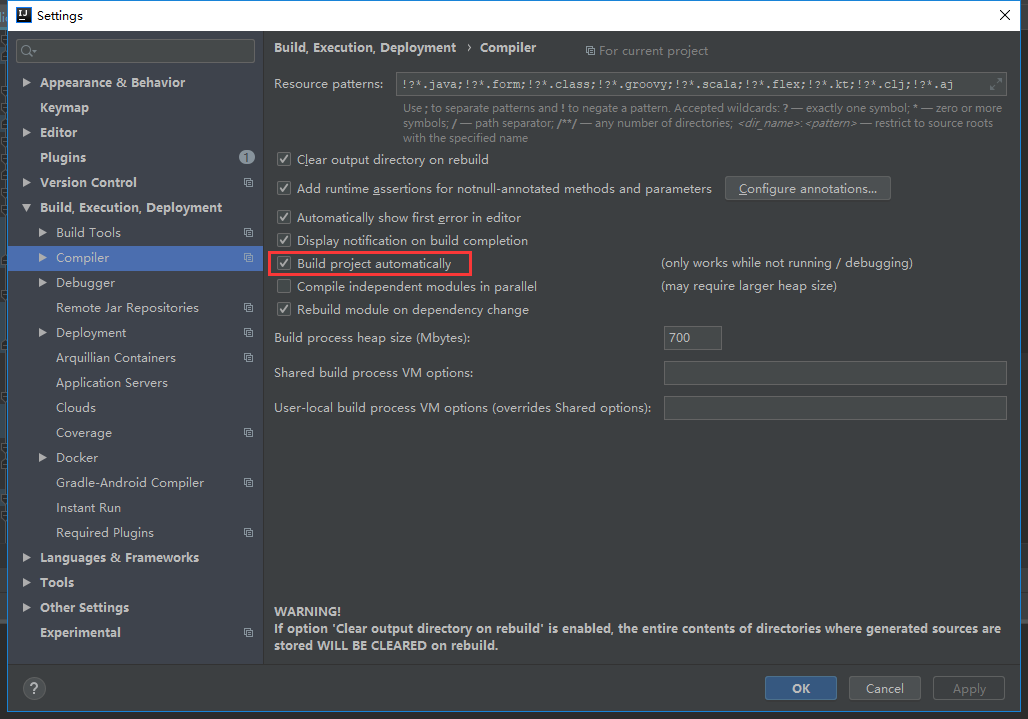
* + 1. 如果是IDEA，可以修改html后点击如下按钮，或是按快捷键CTRL+F9即可刷新



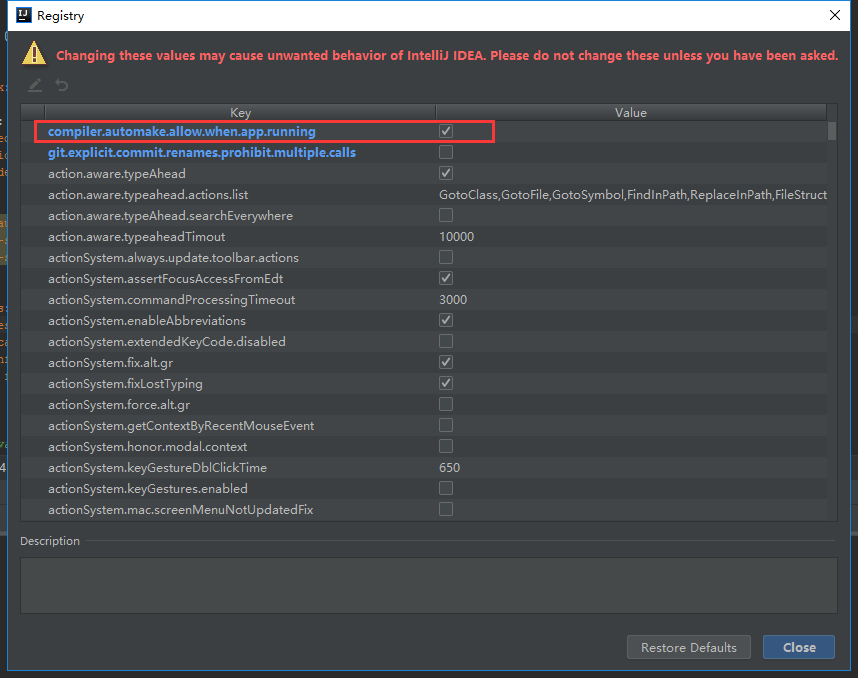
1. 重新加载Java类
   * 1. 如果是eclipse，在application.yml中找到配置spring.devtools.restart.enabled改为true即可
     2. 如果是IDEA中:
        1. **第一步：** 请先修改spring.devtools.restart.enabled=true
        2. **第二步：** 需要修改application.yml中additional-paths添加上相应的项目名称，如下所示:



* + - 1. **第三步：** 如下idea配置，打上对勾



* + - 1. **第四步：** 按下 Shift+Ctrl+Alt+/，选择Registry。进去之后，找到如下图所示的选项，打勾



### 缓存使用

#### 在TonFun中使用缓存的地方不多，主要在CacheFactory的查询中用了缓存，在CacheFactory有高频调用的查询，所以在这些方法上加了缓存，搜索加上缓存后还要注意在修改了相关数据的时候要删除缓存，否则可能导致数据的不一致，在TonFun中默认用的是Ehcache缓存，并配合了spring cache使用，用spring cache的好处就是，spring cache是缓存的抽象，如果想换为redis缓存，则不用修改代码，改一下配置即可实现，下面介绍两种操作缓存的方法。

#### 用工具类操作

#### 在core中封装了一些常用的操作Ehcache缓存的工具类CacheUtil ，此类采用静态方法调用的方式，可以添加、获取、删除缓存、用法非常简单。方法如下：

//添加缓存，第一个为缓存的名称，是ehcache.xml中<cache>节点NAME，key为缓存的键，value为缓存的值

#### public static void put(String cacheName, Object key, Object value)

//获取某个缓存名称中的某个键值对应的缓存

#### public static <T> T get(String cacheName, Object key)

#### //获取某个缓存的所有key

public static List getKeys(String cacheName)

//删除某个key对应的缓存

#### public static void remove(String cacheName, Object key)

//删除某个缓存名称下的所有缓存

#### public static void removeAll(String cacheName)

#### 用Spring cache操作缓存

#### 利用spring cache来操作缓存，可以很方便的在redis和ehcache之间切换缓存实现，利用spring cache 的缓存注解，加到方法之上可以很方便的缓存方法的结果，如果参数对应的键值存在了缓存，则下一次走这个方法则会直接返回缓存的结果，spring cache提供了4个注解来操作缓存.

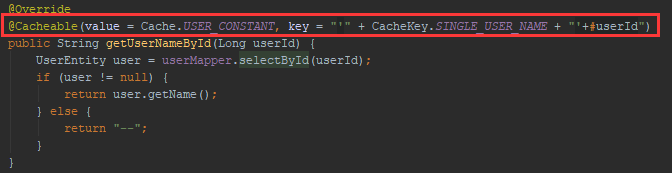
#### @Cacheable： 表明在调用方法之前，首先应该在缓存中查找方法的返回值，如果这 个值能够找到，则会返回缓存的值，否则执行该方法，并将返回值放到缓存中，一般 在数据库查询( select )之后调用这个注解

#### @CachePut： 表明在方法调用前不会检查缓存，方法始终都会被调用，调用之后把 结果放到缓存中，一般在数据库操作插入数据( save )的时候调用

#### @CacheEvict： 表明spring会清除一个或多个缓存，一般在数据库更新或者删除数据时候调用(update或delete)

#### @Caching： 分组的注解，可以同时应用多个其他缓存注解，可以相同类型或者不同类型

一般在用这些注解的时候，我们需要填写两个参数，一个是 value代表缓存的名称，一个是key代表缓存的键值。如下图所示，键值key一般包含两部分组成，一部分是键的标识例如上图中 的 CacheKey.SINGLE\_USER\_NAME，一部分是参数(一般是参数的值)例如上图中的#userId



### 静态模版使用

#### 静态模版可减少服务端压力，提高页面性能，可跨平台（可以兼容不同后端技术），比较灵活改变页面数据，那为什么采用Beetl作为本项目的静态模版？可参考第三章核心技术。静态模版核心配置如下

@Configuration

public class BeetlConfig {

@Autowired

BeetlProperties beetlProperties;

@Bean(initMethod = "init")

public BeetlGroupUtilConfiguration getBeetlGroupUtilConfiguration() {

BeetlConfiguration beetlConfiguration = new BeetlConfiguration();

beetlConfiguration.setResourceLoader(new lasspathResourceLoader(BeetlConfig.class.getClassLoader(), beetlProperties.getPrefix()));

beetlConfiguration.setConfigProperties(beetlProperties.getProperties());

return beetlConfiguration;

}

@Bean

public BeetlSpringViewResolver getBeetlSpringViewResolver() {

BeetlSpringViewResolver beetlSpringViewResolver = new BeetlSpringViewResolver();

beetlSpringViewResolver.setSuffix(".html");

beetlSpringViewResolver.setContentType("text/html;charset=UTF-8");

beetlSpringViewResolver.setOrder(0);

return beetlSpringViewResolver;

}

}

#### Suffix：页面后缀名

#### ContentType： 发送信息至服务器时的内容编码类型a

/\*\*

\* beetl拓展配置,绑定一些工具类,方便在模板中直接调用

\*/

public class BeetlConfiguration extends BeetlGroupUtilConfiguration {

@Override

public void initOther() {

//全局共享变量

Map<String, Object> shared = new HashMap<>();

shared.put("systemName", *DEFAULT\_SYSTEM\_NAME*);

shared.put("welcomeTip", *DEFAULT\_WELCOME\_TIP*);

groupTemplate.setSharedVars(shared);

//全局共享方法

groupTemplate.registerFunctionPackage("shiro", new ShiroKit());

groupTemplate.registerFunctionPackage("tool", new ToolUtil());

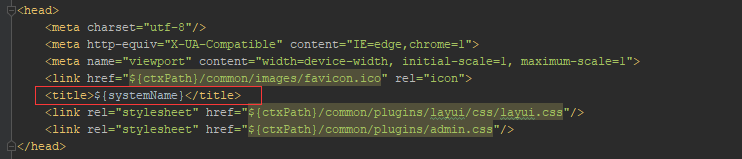
}

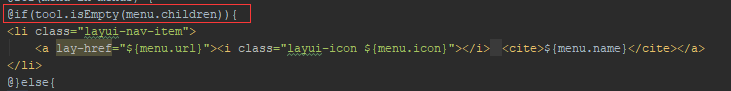
}

#### setSharedVars： 前段共享变量，可根据自己得需求添加

#### registerFunctionPackage： 前端共享方法，可自行定义

#### 变量与方法实战使用如下：





/\*\*

\* beetl配置(如果需要配置别的配置可参照这个形式自己添加)

\*/

@Configuration

@ConfigurationProperties(prefix = BeetlProperties.*BEETLCONF\_PREFIX*)

public class BeetlProperties {

public static final String *BEETLCONF\_PREFIX* = "beetl";

private String delimiterStatementStart;

private String delimiterStatementEnd;

private String resourceTagroot;

private String resourceTagsuffix;

private String resourceAutoCheck;

@Value("${spring.mvc.view.prefix}")

private String prefix;

public Properties getProperties() {

Properties properties = new Properties();

if (ToolUtil.isNotEmpty(delimiterStatementStart)) {

if (delimiterStatementStart.startsWith("\\")) {

delimiterStatementStart = delimiterStatementStart.substring(1);

}

properties.setProperty("DELIMITER\_STATEMENT\_START", delimiterStatementStart);

}

if (ToolUtil.isNotEmpty(delimiterStatementEnd)) {

properties.setProperty("DELIMITER\_STATEMENT\_END", delimiterStatementEnd);

} else {

properties.setProperty("DELIMITER\_STATEMENT\_END", "null");

}

if (ToolUtil.isNotEmpty(resourceTagroot)) {

properties.setProperty("RESOURCE.tagRoot", resourceTagroot);

}

if (ToolUtil.isNotEmpty(resourceTagsuffix)) {

properties.setProperty("RESOURCE.tagSuffix", resourceTagsuffix);

}

if (ToolUtil.isNotEmpty(resourceAutoCheck)) {

properties.setProperty("RESOURCE.autoCheck", resourceAutoCheck);

}

return properties;

}

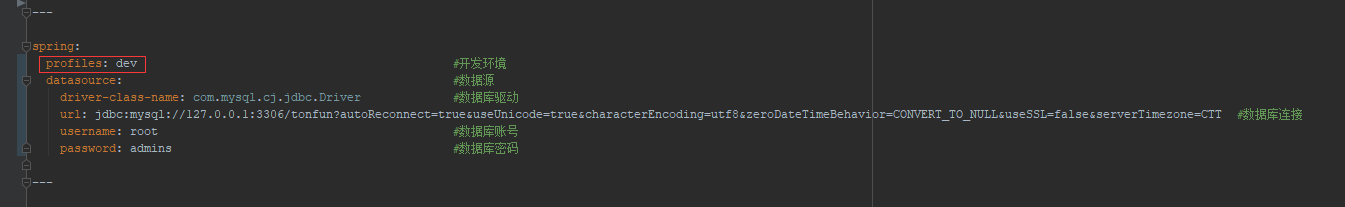
}

### Spring profile

### 在实际的生产环境中，往往存在多个环境，例如开发环境(dev)，测试环境(test)，生产环境 (pro)，并且不同环境的数据库和日志记录等配置的都不相同，为了每次发布不同环境的包时，不来回的修改这些配置，特引入了spring profile，引入之后，我们只需要把所有环境的配置都预先列出来，在每次发布不同环境的包的时候，只需要选择当前激活的是哪个环境的配置即可快速切换配置。在yml配置中，我们用 --- 来切分不同profile的配置，如下:

### 

### 在分割线的下边我们就可以配置不同环境的配置了， profile 可以配置多个，只需要 用 spring.profiles 来标记当前节段的 profile 的名字即可

并用 spring.profiles.active 来激活当前的profile配置即可





### --- 把配置切分成了多个节段，其中第一节是所有profile共有的配置，例如下图中的配置中的这一大段

### 

### 第一节段 --- 下方的配置则是不同的profile的配置

### 工具类使用

### CacheUtil： 缓存工具类

/\*\*

\* 添加缓存

\*

\* @param cacheName 缓存名称,ehcache.xml中<cache>节点NAME

\* @param key 缓存键

\* @param value 缓存值

\*/

public static void put(String cacheName, Object key, Object value)；

/\*\*

\* 获取某个缓存名称中的某个缓存键对应的缓存

\*

\* @param cacheName 缓存名称

\* @param key 缓存键

\* @param <T>

\* @return 返回缓存名称对应缓存值

\*/

public static <T> T get(String cacheName, Object key)；

/\*\*

\* 获取某个缓存名称中的所有key

\*

\* @param cacheName 缓存名称

\* @return 返回该缓存中的所有缓存键

\*/

public static List getKeys(String cacheName)；

/\*\*

\* 删除某个缓存名称中的缓存键对应的缓存

\*

\* @param cacheName

\* @param key

\*/

public static void remove(String cacheName, Object key)；

/\*\*

\* 删除某个缓存名称下的所有缓存

\* @param cacheName 缓存名称

\*/

public static void removeAll(String cacheName)；

### SpringContextUtil： Spring上下文工具类

/\*\*

\* 实现ApplicationContext的方法

\* @param applicationContext 对象

\*/

public void setApplicationContext(ApplicationContext applicationContext) throws BeansException；

/\*\*

\* 获取ApplicationContext对象

\* @return 返回ApplicationContext对象

\*/

public static ApplicationContext getApplicationContext()；

/\*\*

\* 根据bean的beanName来查找对象

\*/

public static <T> T getBean(String beanName)；

/\*\*

\* 根据bean的class来查找对象

\*/

public static <T> T getBean(Class<T> requiredType)；

### ToolUtil： 公共工具类

/\*\*

\* 校验有元素（空、null、""、.length=0）

\*

\* @param o 要检测的容器

\* @return 返回boolean类型true或false，有元素为true，无元素为false

\*

\* 例如："abcdefghijk" -> true

\* 例如："" -> false

\*/

public static boolean isNotEmpty(Object o);

/\*\*

\* 多参数校验无元素（空、null、""、.length=0）

\*

\* @param os 要检测的容器

\* @return 返回boolean类型true或false，有元素为false，无元素为true

\*

\* 例如："abcdefghijk","123456","!@#$%^" -> false

\* 例如："abcdefghijk","","!@#$%^" -> true

\*/

public static boolean isOneEmpty(Object... os)；

/\*\*

\* 校验无元素（空、null、""、.length=0）

\*

\* @param o 要检测的容器

\* @return 返回boolean类型true或false，有元素为false，无元素为true

\*

\* 例如："abcdefghijk" -> false

\* 例如："" -> true

\*/

public static boolean isEmpty(Object o)；

/\*\*

\* 首字母变小写

\*

\* @param str 字符串

\* @return 返回首字母变小写的字符串

\*

\* 例如：HELLO\_WORLD -> hELLO\_WORLD

\*/

public static String firstCharToLowerCase(String str)

/\*\*

\* 首字母大写

\*

\* @param val 字符串

\* @return 返回首字母变大写的字符串

\*

\* 例如：hello\_world -> Hello\_world

\*/

public static String firstCharToUpperCase(String val)；

/\*\*

\* 格式化文本

\*

\* @param template 文本模板,被替换的部分用{}表示

\* @param values 参数值

\* @return 格式化后的文本

\*

\* 例如：\{}\{}.html,aaaa,bbbb -> \aaaa\bbbb.html

\*/

public static String format(String template, Object... values)；

/\*\*

\* 截取字符串

\*

\* @param string 字符串

\* @param fromIndex 开始的索引(包含)

\* @param toIndex 结束索引(不包含)

\* @return 返回截取后的字符串

\*

\* 例如：abcdefghijk,2,0 -> ab

\* 例如：abcdefghijk,2,3 -> c

\* 例如：abcdefghijk,2,-3 -> cdefgh

\*/

public static String sub(String string, int fromIndex, int toIndex)；

/\*\*

\* 将下划线方式命名的字符串转换为驼峰式

\*

\* @param name 转换前的下划线大写方式命名的字符串

\* @return 转换后的驼峰式命名的字符串

\*

\* 例如：hello\_world->helloWorld

\* 例如：HELLO\_WORLD->helloWorld

\*/

public static String toCamelCase(String name)；

/\*\*

\* 剔除指定前缀

\*

\* @param str 字符串

\* @param prefix 前缀

\* @return 返回剔除指定前缀得字符串,若前缀不是preffix,返回原字符串

\*

\* 例如：sys\_user,sys\_ -> user

\* 例如：sys\_user,aaa\_ -> sys\_user

\*/

public static String removePrefix(String str, String prefix)；

/\*\*

\* 获取随机生成“盐”

\*

\* @param length 生成随即“盐”的长度

\* @return 生成对应长度的随机数

\*

\* 例如：6 -> 7m6ulo

\*/

public static String getRandomString(int length)；

/\*\*

\* 密码加密

\*

\* @param credentials 密码

\* @param saltSource 密码盐

\* @return 生成后返回SimpleHash格式的32位MD5值

\*

\* 例如：123465,7m6ulo -> c17c7f07bf50c48552701409a4e82da4

\*/

public static String md5(String credentials, String saltSource)；

/\*\*

\* 获取文件后缀名

\*

\* @param fileWholeName 转换前的下划线大写方式命名的字符串

\* @return 截取后返回后缀名

\*

\* 例如：test.xml -> xml

\*/

public static String getFileSuffix(String fileWholeName)；

### 数据权限

为了不同的人员可以看到只有与他相关的数据，特提供数据权限功能。可以防止数据泄漏问题。相关代码有

* DataScope：自定义注解
* DataScopeAspect：自定义注解实现类
* BaseEntity：公共实体参数

/\*\* 添加到服务器实现类中 \*/

@DataScope(deptAlias = "d", userAlias = "u")

/\*\* XML增加${user.params.dataScope} \*/

<select id="UserEntityList" resultType="com.example.tonfun.modular.system.entity.UserEntity">

SELECT u.\*,d.full\_name AS deptName,GROUP\_CONCAT(r.`name`) AS roleName

FROM sys\_user u

LEFT JOIN sys\_role r ON FIND\_IN\_SET(r.id,u.role\_id)

LEFT JOIN sys\_dept d ON u.dept\_id = d.id

WHERE u.status != 'DELETED'

<if test="user.name != null and user.name != ''">

ADD (u.account LIKE CONCAT('%',#{user.name},'%') )

</if>

<if test="user.deptId != null and user.deptId != 0">

AND (u.dept\_id = #{user.deptId} OR u.dept\_id in ( SELECT id FROM sys\_dept where pids LIKE CONCAT('%$[', #{user.deptId}, '$]%') ESCAPE '$' ))

</if>

<!-- 数据范围过滤 -->

${user.params.dataScope}

GROUP BY u.id

ORDER BY u.create\_time EDSC

</select>

注：其中@DataScope(deptAlias = "d", userAlias = "u")中得d和u是在xml创建得关联表别名

### 资源控制

为了防止通过URL地址进行操作数据库数据，特提供资源权限控制功能。具体实现及使用如下

* 自定义注解

@Inherited

@Retention(RetentionPolicy.*RUNTIME*)

@Target({ElementType.*METHOD*})

public @interface Permission {

/\*\*

\* <p>角色英文名称</p>

\* <p>使用注解时加上这个值表示限制只有某个角色的才可以访问对应的资源</p>

\* <p>常用在某些资源限制只有超级管理员角色才可访问</p>

\*/

String[] value() default {};

}

* 注解实现类

@Aspect

@Component

@Order(200)

public class PermissionAop {

@Autowired

private PermissionCheckService check;

@Pointcut(value = "@annotation(com.example.tonfun.core.annotation.Permission)")

private void cutPermission() {}

@Around("cutPermission()")

public Object doPermission(ProceedingJoinPoint point) throws Throwable {

MethodSignature ms = (MethodSignature) point.getSignature();

Method method = ms.getMethod();

Permission permission = method.getAnnotation(Permission.class);

Object[] permissions = permission.value();

if (permissions.length == 0) {

//检查全体角色

boolean result = check.checkAll();

if (result)

return point.proceed();

else

throw new NoPermissionException();

} else {

//检查指定角色

boolean result = check.check(permissions);

if (result)

return point.proceed();

else

throw new NoPermissionException();

}

}

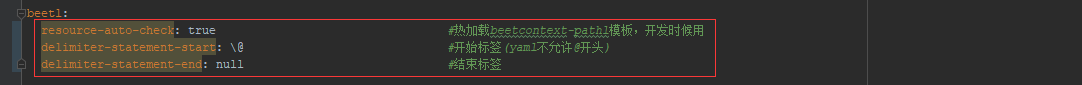
}

* 注解使用,如多角色可进行逗号拼接进行处理

@Permission({ConfigConsts.*ADMIN\_NAME*})

## 前端手册

TonFun前端采用了beetl模板引擎，beetl包含语法简洁，速度快，文档全，社区活跃等众多优点，默认情况beetl支持<% %>,本系统采用更简便的语法以@开头。

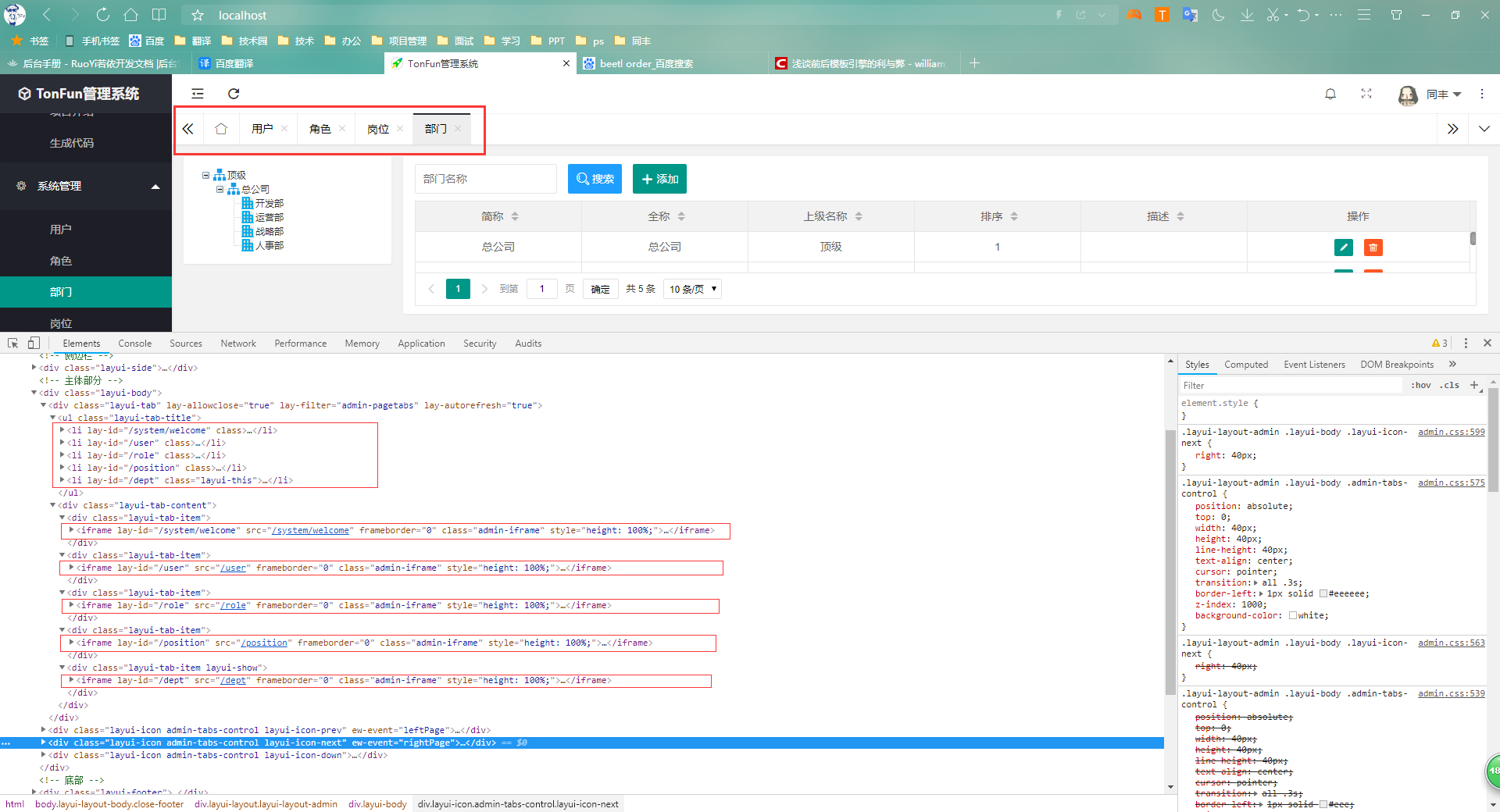


### 布局

#### 在用户登录页面后进入的是index.html页面，这个页面加载了整个后台管理系统的框架，我们可以看到index.html源代码中把整个页面分为了三部分，左侧菜单栏，右侧页面和右侧主题栏部分，其实就是利用beetl的@include把整个大的复杂的页面细化了，这样好维护

#### 

#### 左侧菜单和右侧主题栏部分在用户登录后会一直不变，除非刷新浏览器页面，动态变化的是页面右侧这部分，我们打开4个标签页，并打开浏览器F12调试



#### 新建和切换标签，页面的地址不会变化，变化的是页面右侧的 iframe 这部分 下面我们分析一下右侧页面的组成，我们打开菜单管理页面，查看他的代码

@layout("/common/\_container.html",{plugins:["ztree"],js:["/modular/system/dept/dept.js"]}){

XXXX等html代码...

@}

#### 整个页面被@layout所包围，@layout是beetl的引用布局(具体用法文档可以查看beetl的官方文档)，TonFun中内置了/common/\_container.html这样一个布局，可以把 /common/\_container.html 理解为一个html的抽象封装，我们每个页面都继承自这个模板，默认包含了一系列通用的js、css引用等，这样写即简化了我们的开发和维护，又使我们的代码简洁有序，在 /common/\_container.html 中的 ${layoutContent} 就代表我们每个页面不同的html。

### 参数使用

@layout("/common/\_container.html",{title:"xxx",plugins:["ztree"],css:["/modular/xxx/xxx.css"],js:["/modular/xxx/dept.js"]}){

XXXX等html代码...

@}

* **Titile：** 标签页名称
* **Plugins：** 参数标识
* **Css：** 引入该页面所对应的CSS页
* **Js：** 引入该页面所对应的js页

### \_container公用文件使用

如需扩展请自行根据规则进行添加

* Css的两种引用方式第一种参数形式、第二种为数组形式

@/\* 其他插件css \*/

@if(isNotEmpty(plugins)){

@if(array.contain(plugins,"ztree")){

<link href="${ctxPath}/common/plugins/ztree/zTreeStyle.css" rel="stylesheet" type="text/css" />

@}

@}

@/\* 其他业务用的css \*/

@if(isNotEmpty(css)){

@for(item in css){

@if(strutil.startWith(item,"/")){

@item = ctxPath + item;

@}

<link href="${item}" rel="stylesheet" type="text/css" />

@}

@}

* Js的两种引用方式第一种参数形式、第二种为数组形式

@/\* 其他插件js \*/

@if(isNotEmpty(plugins)){

@if(array.contain(plugins,"ztree")){

<script src="${ctxPath}/common/plugins/jquery/jquery-3.2.1.min.js"></script>

<script src="${ctxPath}/common/plugins/ztree/jquery.ztree.all.min.js"></script>

@}

@if(array.contain(plugins,"media")){

<script src="${ctxPath}/common/plugins/jquery/jquery.min.js"></script>

<script src="${ctxPath}/common/plugins/jquery/jquery.media.pdf.js"></script>

@}

@}

@/\* js和业务用的js数组 \*/

@if(isNotEmpty(js)){

@for(item in js){

@if(strutil.startWith(item,"/")){

@item = ctxPath + item;

@}

<script src="${item}"></script>

@}

@}

### 弹出框的使用

/\*\* 封装简化弹出框 area可不填 \*/

func.open({

title: '添加XXX',

area: ['300px', '400px'],

content: Feng.ctxPath + '/XXX/XXX\_add',

tableId: XXX.tableId

});

/\*\* 原生弹出框 \*/

layer.open({

type: 2,

title: '添加XXX',

area: ['300px', '400px'],

content: Feng.ctxPath + '/XXX/XXX\_add',

end: function () {

layer.closeAll('tips');

admin.getTempData('formOk') && table.reload(param.tableId);

}

});

### 树形结构渲染

* 弹框树、左侧树

满足ZTreeNode结构同时加入createParent。防止空数据展示空白

* 树形列表

通过treetable.render方式进行调用初始化表格的列,同时满足

// 渲染表格 树形列表

return treetable.render({

elem: '#' + id,

url: Feng.ctxPath + '/XXX/XXX',

where: data,

page: false,

height: "full-158",

cellMinWidth: 100,

cols: Menu.initColumn(),

treeColIndex: 2,

treeSpid: "0",

treeIdName: 'id',

treePidName: 'pid',

treeDefaultClose: true,

treeLinkage: false

});

Menu.initTable(Menu.tableId); //初始化

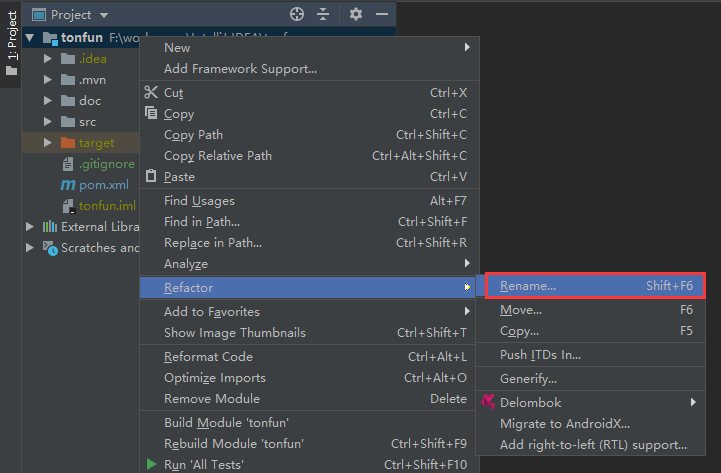
## 扩展与高级配置

### 重命名项目名称

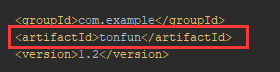
#### 第一步：修改文件名称



#### 第二步：右击项目，点refactor->Rename



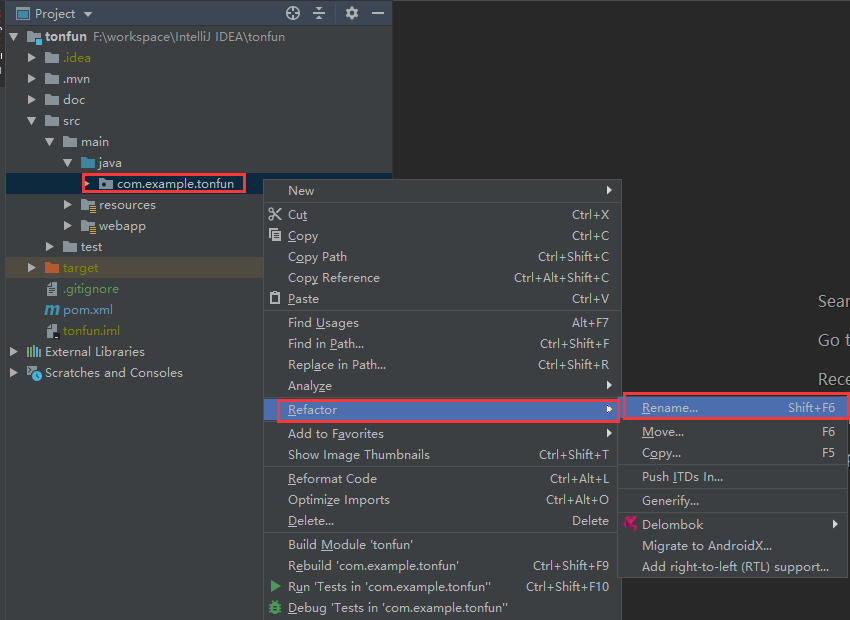
#### 第三步： 修改POM.xml中的artifactId改为project



### 重命名包名称

#### 下面以把com.example.tonfun改为com.tonfun.project为例

#### 选择com.example.tonfun包，仍然为右键 refactor->Rename



#### 弹出对话框选择，有两个选项，我们选 Rename Package，修改名字后点Refactor

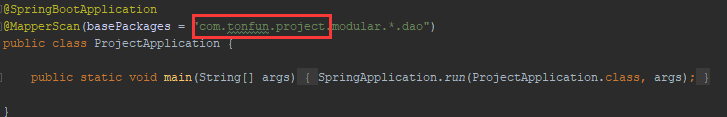
#### Rename Package： 也就是重命名所有模块中 com.example.tonfun的包

#### Rename Directory： 只重命名当前模块的 com.example.tonfun包

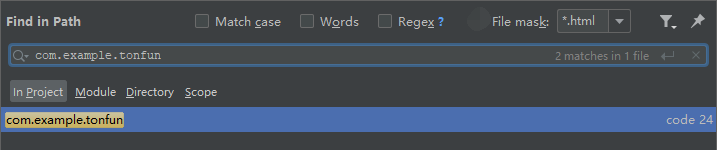
#### 把所有包扫描的地方改为com.tonfun.project，首先改application.yml中的相关配置



#### 修改启动类ProjectApplication



#### 最后Ctrl+Shift+F输入com.example.tonfun把系统中相关得名称改为com.tonfun.project



### 更换登录页背景样式

### 切换Swagger样式效果

### Mybatis-plus通用枚举

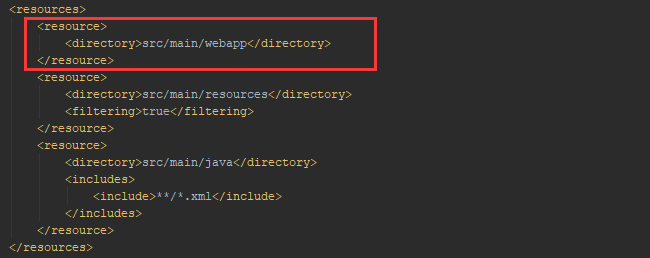
### 多模块

### Maven私服

### 静态资源和模板位置的变更

#### 由于SpringBoot默认是把静态资源文件css，js等放到 resources/static目录的，默认把前端模板文件放到 resources/templates目录，笔者认为前端面页面还是按maven的思想放到webapp目录比较分层清晰，所以做了一个变动，主要变动如下:

#### pom.xml中增加



#### application.yml中增加



#### 若想变动资源和模板的位置修改这两个配置即可

### 推荐插件

为了减少与代码无关的操作及处理，特准备部分插件供开发者使用。

安装方法：打开IDEA，进入File -> Settings -> Plugins -> Browse Repositories，输入mybatisx 搜索并安装。

* **Mybatisx：**Java与XML调回跳转、Mapper方法自动生成XML
* **Lombok：**减少getter、setter等方法的编写
* **Translation：**翻译插件

## 常见问题汇总

### 放行权限验证

#### 在日常开发中，我们可能需要放过某个接口的权限验证，即用户不用登录就可以访问接口.在ShiroConfig类中，找到shiroFilter()这个方法，配置上需要过滤得接口，注意加到最上面，这个Map是有顺序的，可以用通配符。

### 

### 关于 ${ctxPath}

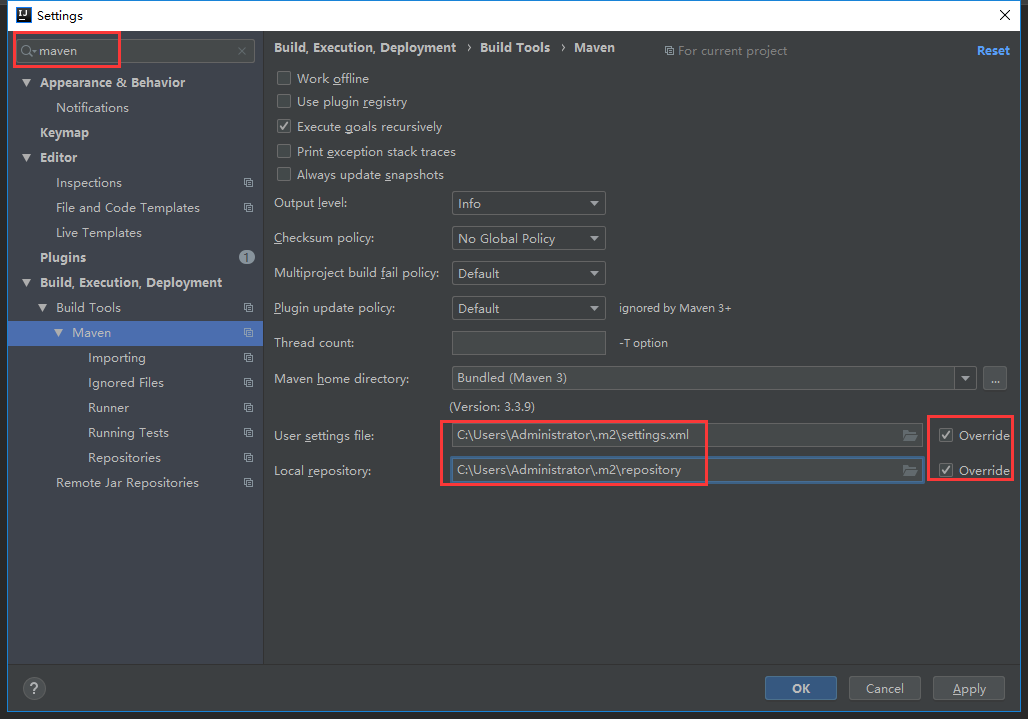
#### 这个变量在哪里定义的?这个是beetl自带的具体请看beetl文档

#### Beetl默认提供了WebRender用于帮助web集成开发，所有内置的集成均基于此方法。



### Maven下载包慢

### 检查maven安装目录下的settings.xml是否配置了阿里云镜像



<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">

<mirrors>

<mirror>

<id>alimaven</id>

<name>aliyun maven</name>

<url>http://maven.aliyun.com/nexus/content/groups/public/</url>

<mirrorOf>central</mirrorOf>

</mirror>

</mirrors>

</settings>

### 事务不生效

#### 常见问题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！！

#### /\*\*

#### \* 新增用户

\*/

@Transactional

public int insertUser(UserEntity userEntity) throws Exception{

// 新增用户信息

int rows = userMapper.insertUser(userEntity);

// 新增用户角色关联

insertUserAndRole(userEntity);

// 新增用户部门关联

insertUserAndDept(userEntity);

// 模拟抛出SQLException异常

if (true)

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)

#### 常见问题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。如下面这段代码直接导致用户新增的事务回滚没有生效。

#### 推荐做法：在业务层统一抛出异常，然后在控制层统一处理。

#### /\*\*

#### \* 新增用户

\*/

@Transactional

public int insertUser(UserEntity userEntity) throws Exception{

// 新增用户信息

int rows = userMapper.insertUser(userEntity);

// 新增用户角色关联

insertUserAndRole(userEntity);

// 新增用户部门关联

insertUserAndDept(userEntity);

// 模拟抛出SQLException异常

if (true) {

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;

}

### 关于@layout

layout关键字是Beetl模版所提供得页面引入方法，同include的使用。