**框架使用手册**

## 几个基础类

1. IBaseDao<T>：基础数据访问接口，提供数据库基础访问，如：单表的增删改查操作。一般情况下如果表操作只是简单的增删改查操作，那么我们不需要写DAO类；如果有基础操作之外的数据库操作，建议新增一个DAO类或者接口，不需要继承此接口。

insert(T t):新增一条数据

updateByPrimaryKey(T t):根据主键修改数据

delete(T t):以t对象的非NULL字段为条件删除数据

selectOne(T t):以t对象的非NULL字段为条件查询一条数据，如果查询结果有多条会报错，一般根据主键获取数据

select(T t):以t对象的非NULL字段为条件查询数据

通过IBaseDao进行数据库操作时，如果数据表包含主键，则必须在对应的T类通过@Id注解标记对应的主键字段，可标记多个字段为主键

1. IBaseService<T>：基础服务接口，提供基本的数据操作服务，如：单表的增删改查操作。要求所有的服务类都必须实现此接口
2. BaseService<T>：基础服务类，IBaseService的实现类，提供基本的数据操作服务实现。此类包含一个数据层操作对象字段@Autowired protected IBaseDao<T> dao，常规数据操作可以通过dao操作; 如果有新增的服务类，有三种选择，一是继承BaseService类，二是在服务类中添加BaseService字段，三是实现IBaseService接口；建议所有服务类继承自BaseService类，下面给出的示例为第二种方式：

@Service

**public** **class** TestService **implements** IBaseService<TestModel> {

@Autowired **private** BaseService<TestModel> svc;

@Override

**public** **long** add(TestModel t) {

**return** svc.add(t);

}

}

## 代码结构

1. api:所有的接口位于此目录下，通过不同的角色和模块命名接口类，如：api/admin/User.java
2. service:所有的服务类位于此目录下，且后缀必须为Service，如：service/BaseService.java。所有的服务类必须实现IBaseService<T>接口，建议继承自BaseService类
3. dao:所有的数据访问类位于此目录下，且后缀必须为Dao，如dao/UserDao.java。
4. model:所有对系统外部提供的数据的对象位于此目录下且后缀必须为Model，此目录提供对外(系统外部，不包含在工程内的数据访问)的接口数据，如：为接口的请求参数和返回数据必须以model对象化并位于此目录下，model示例：model/UserModel.java
5. model/validation:数据的数据校验服务位于此目录下且后缀必须为Validation，此目下的校验针对model目下的实体类进行校验，如：model/validation/UserValidation.java
6. domain:所有和数据表对应的实体对象位于此目录下且类名必须与数据表对应，实体字段必须与数据表的字段对应，一般进行数据的增删改操作时用domain进行操作
7. resources:

## 异常处理

当系统出现异常时，我们需要统一进行异常处理，返回统一的异常格式，并且记录异常时的状态，框架提供ApplicationException异常类，所有异常应抛出此异常。ApplicationException可以记录异常的状态码、异常消息、异常时的数据以及导致此异常的上一个异常信息。使用方式：

throw new ApplicationException(int code, Object data, Object... params);

throw new ApplicationException(Exception e, Object data);

throw new ApplicationException(int code, Object data, Exception e, Object... params);

其中Object... params表示消息参数。

ApplicationException将根据code状态码从error\_message.properties文件内(classes目录下)获取code对应的状态消息，状态消息可以用%s等作为占位符，然后将占位符的内容替换为params参数，如：

10010=%s您好，您的%s信息错误，请补充完整

抛出异常: throw new ApplicationException(10010, Object data, "张三", "银行");

输出结果: {status:10010, statusText:"张三您好，您的银行信息错误，请补充完整"}

## 全局消息码

0:未知错误，可能原因：参数输入不正确(在GET参数时，如果参数类型不正确会导致此错误)、系统内部错误

1:运行正常

2:参数校验错误

## Kafka的使用

## Quartz的使用

## Redis的使用

## 代码规范

## 接口文档编写

接口文档采用swagger2根据在代码添加的注解自动生成，生成后可通过/{projectName}/swagger-ui.html访问。

1. 接口类

@Api:在所有的接口类上必须添加此注解，swagger会根据此注解生成接口文档，无注解不会生成。

1. 接口方法

@ApiOperation("接口说明"):在所有接口方法上必须添加此注解，并描述清楚接口功能。

@ApiResponses({@ApiResponse(code=状态码,message="状态消息"),...}):在所有接口方法上应该添加此注解，并描述清楚对应状态码信息。

1. 接口参数

接口参数根据参数传递方式不同所使用的注解也不同。常用的注解有@ApiParam、@RequestParam、@RequestBody、@Valid。

@ApiParam("参数描述")/@ApiParam(name="参数名称,一般为英文，与实际参数名称对应", value="参数描述"):一般用第一种使用方式，所有参数必须使用。

@RequestBody:以json格式传参时使用，一般用在POST请求，且正常情况下只能有一个此注解。

@RequestParam:在通过URL传参时使用，GET、POST都可以，如果POST请求为json格式请求时。

@Valid:参数校验。

1. 参数/结果对象(类)

当我们的接口接口参数或者返回对象为一个实体类时，需要标注对象说明。

@ApiModel("测试对象"): 此注解标注在实体类上面，参数为实体类说明，一般可以不需要此注解。

@ApiModelProperty(required=true,value="用户性别, 0-女;1-男",allowableValues="0,1")：此注解标注在实体类的字段上面，用于描述字段信息：required参数如果没有设置则默认为false,value参数描述字段的含义及备注信息,allowableValues参数表示可能的值，一般用于枚举。

ApiModelProperty注解还有更多的参数，需要我们自己慢慢探索。

1. 示例代码

Api示例

@Api

@RestController

@RequestMapping("test")

public class TestApi {

@PostMapping("swagger")

@ApiOperation(value="swagger 文档生成示例")

@ApiResponses({@ApiResponse(code=10010,message="用户令牌已过期")})

public OutputModel<List<TestModel>> swagger(

@ApiParam(name="参数名称,一般为英文，与实际参数名称对应", value="参数描述") @RequestBody @Valid TestModel mod

, @ApiParam("参数描述") @RequestParam String stringParam) {

TestModel model = new TestModel();

List<TestModel> list = new ArrayList<TestModel>();

list.add(model);

return new OutputModel<List<TestModel>>(0, list);

}

}

Model示例

@ApiModel(value="测试对象")

public class TestModel {

/\*\*

\* 用户名称

\*/

@NotBlank

@ApiModelProperty(required=true, value="用户名称")

private String username;

@ApiModelProperty(value="用户性别, 0-女;1-男", allowableValues="0,1")

private int gender;

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public int getGender() {

return gender;

}

public void setGender(int gender) {

this.gender = gender;

}

}