# 后端方法说明

目录

[后端方法说明 1](#_Toc486434243)

[项目介绍： 3](#_Toc486434244)

[1.BaseDao<T> 4](#_Toc486434245)

[删除实体 5](#_Toc486434246)

[级联删除 5](#_Toc486434247)

[批量删除实体 5](#_Toc486434248)

[通过主键信息获取实体对象 5](#_Toc486434249)

[保存或更新实体, 实体没有主键时保存，否则更新 5](#_Toc486434250)

[获取单个实体，根据HQL查询语句及参数获取 5](#_Toc486434251)

[获取单个实体，根据HQL查询语句及参数获取 5](#_Toc486434252)

[通过Sql语句查询，返回实体对象 6](#_Toc486434253)

[通过Sql语句查询，返回实体对象 6](#_Toc486434254)

[通过Sql语句查询map集合，通过反射封装进实体对象 6](#_Toc486434255)

[通过Sql语句查询map集合，通过反射封装进实体对象 6](#_Toc486434256)

[查询实体列表 6](#_Toc486434257)

[查询实体列表 7](#_Toc486434258)

[通过Sql语句查询map，返回实体对象列表 7](#_Toc486434259)

[通过Sql语句查询map，返回实体对象列表 7](#_Toc486434260)

[通过属性快速查实体对象 7](#_Toc486434261)

[通过属性map查找实体集合 7](#_Toc486434262)

[执行数据库更新操作 7](#_Toc486434263)

[执行数据库更新操作 8](#_Toc486434264)

[不返回记录总数的分页 并根据前端条件判断是否需要多查询一页数据给前端做缓存 8](#_Toc486434265)

[不返回记录总数的分页 并根据前端条件判断是否需要多查询一页数据给前端做缓存 8](#_Toc486434266)

[不返回记录总数的分页 并根据前端条件判断是否需要多查询一页数据给前端做缓存 8](#_Toc486434267)

[执行带输出参数存储过程 9](#_Toc486434268)

[执行带输出参数存储过程 9](#_Toc486434269)

[执行不带输出参数存储过程 9](#_Toc486434270)

[定制输出的存储过程 9](#_Toc486434271)

[获取JdbcTemplate对象 9](#_Toc486434272)

[获取 对象 9](#_Toc486434273)

[获取session对象 10](#_Toc486434274)

[2.BaseService 10](#_Toc486434275)

[保存对象 10](#_Toc486434276)

[通过单个id删除对象 10](#_Toc486434277)

[批量删除实体 10](#_Toc486434278)

[级联删除 10](#_Toc486434279)

[通过id获取到实体对象 11](#_Toc486434280)

[2.附件 11](#_Toc486434281)

[3.字典 13](#_Toc486434282)

[4.消息提醒 15](#_Toc486434283)

## 项目介绍：

项目主要分为五个包，Dao,Domain,Service,Util，Web。

Dao为数据持久层，Domain为实体层，Service为数据处理层，Web中的Conrtoller为控制层，Web中的resourse为项目配置信息，Util为工具包。

一个方法的执行和返回通过实体进行数据传递，由Controller🡪Service🡪Dao；

所有的业务Controller都继承了BaseController，为了统一报错日志的输出。

所有的业务Service继承了BaseService为了简化部分简单的业务数据的增删改查；且实现了类的接口IService。

所有的业务Dao都继承了BaseDao提供了数据持久化和查询的基本方法，且继承了本类的接口IDao。

下面对BaseDao和BaseService进行说明:

BaseService具有：

|  |  |
| --- | --- |
| **save** (T entity) | 保存对象 |
| **delete**(Class<T> entity, Serializable id) | 通过单个id删除对象 |
| **deleteBatch**(Class<T> entity, String ids) | 批量删除实体 |
| **deleteCascade**(Class<T> entity, Serializable id) | 级联删除 |
| **get**(Class<T> clazz, Serializable id) | 通过id获取到实体对象 |
| **findUniqueByProperty**(Class<T> entityClass, String propertyName, Object value) | 通过属性查找单一的参数 |
| **findByProperty**(Class<T> entityClass, String propertyName, Object value) | 通过参数查看对象list |

这些方法。

所以当我们的业务Service继承了BaseService,我们的业务Service 也继承了这些方法，我们在我们的业务Controller直接注入我们的业务Service接口，就可以直接调用它的方法。

例：（以警卫中的黑名单为例）

|  |
| --- |
| @Autowired **private** IHmdService **hmdService**;  */\*\* \* 保存黑名单 \** ***@param hmd*** *黑名单 \** ***@return*** *AjaxReturn \*/* @ResponseBody @RequestMapping(**"/saveHmd"**) **public** AjaxReturn saveHmd(@FormModel Hmd hmd) {  **hmdService**.save(hmd);  **return new** AjaxReturn().setSuccess(**true**).setData(hmd.getId()); } |
| */\*\* \* 删除黑名单 \** ***@param id*** *黑名单id \** ***@return*** *AjaxReturn \*/* @ResponseBody @RequestMapping(**"/deleteHmd"**) **public** AjaxReturn deleteHmd(Long id) {  **hmdService**.delete(Hmd.**class**,id);  **return new** AjaxReturn().setSuccess(**true**); } |
| */\*\* \* 通过ids字符串删除多个业务对象 \** ***@param ids*** *业务Id的逗号拼接字符串 \*/* @ResponseBody @RequestMapping(**"/deleteBatch"**) **public** AjaxReturn deleteBatch(String ids){  **hmdService**.deleteBatch(Hmd.class,ids);  **return new** AjaxReturn().setSuccess(**true**); } |
| */\*\*  \*通过id获取对象  \*/* @ResponseBody @RequestMapping(**"/getHmdById"**) **public** AjaxReturn getHmdById(Long id){  **return new** AjaxReturn().setSuccess(**true**).setData(**hmdService**.get(Hmd.**class**,id)); } |
| … |

## 1.BaseDao<T>

BaseDao<T>是后端开发基础Dao,它是一个泛型类，所有的业务Dao都需要继承BaseDao，BaseDao注入了SessionFactory, JdbcTemplate 和NamedParameterJdbcTemplate，并对外提供了公共方法让业务Dao可以获取它们，并调用它们的方法,继承了BaseDao的业务Dao只能保存修改或删除本类泛型的对象。

接口

---------------------------------- 实体基本CRUD操作--------------------------------------

### 删除实体

*/\*\*  
 \** ***@param id*** *实体对象主键  
 \*/***void delete**(Serializable id);

### 级联删除

*/\*\*  
 \* \** ***@param id*** *实体对象主键  
 \*/***void deleteCascade**(Serializable id)

### 批量删除实体

*/\*\*  
 \* \** ***@param ids*** *实体主键集合  
 \*/***void deleteBatch**(String ids)

### 通过主键信息获取实体对象

*/\*\*  
 \** ***@param id*** *主键ID  
 \** ***@return*** *单个实体  
 \*/*

### T **get**(Serializable id)

### 保存或更新实体, 实体没有主键时保存，否则更新

*/\*\*  
 \** ***@param entity*** *实体对象  
 \*/***void saveOrUpdate**(T entity)

---------------------------------- 获取单个实体对象 -------------------------------------

### 获取单个实体，根据HQL查询语句及参数获取

*/\*\** *\* 说明：如果获取部分字段，需要根据部分字段创建新的构造函数  
 \** ***@param queryString*** *查询语句  
 \** ***@param params*** *可选的查询参数，可变参数形式  
 \** ***@return*** *单个实体，如果查询结果有多个，则返回第一个实体  
 \*/*T **getByHql**(CharSequence queryString, Object... params)

### 获取单个实体，根据HQL查询语句及参数获取

*/\*\*  
 \* \* 说明：如果获取部分字段，需要根据部分字段创建新的构造函数  
 \*  
 \** ***@param queryString*** *查询语句  
 \** ***@param params*** *可选的查询参数，map形式  
 \** ***@return*** *单个实体，如果查询结果有多个，则返回第一个实体  
 \*/*T **getByHql**(CharSequence queryString, Map<String, ?> params)

### 通过Sql语句查询，返回实体对象

*/\*\*  
 \**  *\* 说明：该方法必须使用 select \* 的方式获取实体的所有属性，且实体必须受hibernate管理  
 \* 优势：支持级联的关联查询*

*\*  
 \** ***@param sql*** *查询sql  
 \** ***@param params*** *查询参数可变参数形式  
 \** ***@return*** *单个实体  
 \*/*T **getBySql**(CharSequence sql, Object... params)

### 通过Sql语句查询，返回实体对象

*/\*\*  
 \* \* 说明：该方法必须使用 select \* 的方式获取实体的所有属性，且实体必须受hibernate管理  
 \* 优势：支持级联的关联查询  
 \*  
 \** ***@param sql*** *查询sql  
 \** ***@param params*** *查询参数map形式  
 \** ***@return*** *单个实体  
 \*/*T **getBySql**(CharSequence sql, Map<String, ?> params)

### 通过Sql语句查询map集合，通过反射封装进实体对象

*/\*\*  
 \* 优势：支持查询部分属性，封装进实体  
 \*  
 \** ***@param sql*** *查询sql  
 \** ***@param params*** *查询参数可变参数形式  
 \** ***@return*** *单个实体  
 \*/*T **getBySqlNative**(CharSequence sql, Object... params)

### 通过Sql语句查询map集合，通过反射封装进实体对象

*/\*\*  
 \* 优势：支持查询部分属性，封装进实体  
 \*  
 \** ***@param sql*** *查询sql  
 \** ***@param params*** *查询参数map形式  
 \** ***@return*** *单个实体  
 \*/*T **getBySqlNative**(CharSequence sql, Map<String, ?> params)

### 查询实体列表

*/\*\*  
 \** ***@param queryString*** *查询语句  
 \** ***@param params*** *可选的查询参数，可变参数形式  
 \** ***@return*** *实体列表  
 \*/*

---------------------------------- 获取实体集合对象-------------------------------------List<T> **findListByHql**(CharSequence queryString, Object... params)

### 查询实体列表

*/\*\*  
 \** ***@param queryString*** *查询语句  
 \** ***@param params*** *可选的查询参数，map形式  
 \** ***@return*** *实体列表  
 \*/*List<T> **findListByHql**(CharSequence queryString, Map<String, ?> params)

### 通过Sql语句查询map，返回实体对象列表

*/\*\*  
 \** ***@param sql*** *查询sql  
 \** ***@param params*** *查询参数可变参数形式  
 \** ***@return*** *实体列表  
 \*/*List<T> **getListBySql**(CharSequence sql, Object... params)

### 通过Sql语句查询map，返回实体对象列表

*/\*\*  
 \** ***@param sql*** *查询sql  
 \** ***@param params*** *查询参数map形式  
 \** ***@return*** *实体列表  
 \*/*List<T> **getListBySql**(CharSequence sql, Map<String, ?> params)

### 通过属性快速查实体对象

*/\*\*  
 \** ***@param name*** *实体的属性名  
 \** ***@param value*** *属性值  
 \** ***@return*** *实体列表  
 \*/*List<T> **findByProperty**(String name, Object value)

### 通过属性map查找实体集合

*/\*\*  
 \** ***@param conditionMap*** *实体的属性集合，属性名：属性值  
 \** ***@return*** *实体列表  
 \*/*List<T> **findByProperty**(Map<String, ?> conditionMap)

--------------------------------- 执行更新语句 -------------------------------------

### 执行数据库更新操作

*/\*\*  
 \** ***@param sql*** *sql语句  
 \** ***@return*** *更新的记录条数  
 \*/***int executeSqlUpdate**(CharSequence sql, Object... args)

### 执行数据库更新操作

*/\*\*  
 \** ***@param hql*** *hql语句  
 \** ***@return*** *更新的记录条数  
 \*/***int executeHqlUpdate**(CharSequence hql, Object... args)

------------------------------------- 快速分页查询----------------------------------------

### 不返回记录总数的分页 并根据前端条件判断是否需要多查询一页数据给前端做缓存

*/\*\*  
 \** ***@param queryString*** *原生sql语句  
 \** ***@param params*** *sql的参数  
 \** ***@param pageParam*** *分页查询参数  
 \** ***@return*** *FastPagination  
 \*/***FastPagination** **cacheNextTotalPaginationSql**(CharSequence queryString, List params, Map pageParam)

### 不返回记录总数的分页 并根据前端条件判断是否需要多查询一页数据给前端做缓存

*/\*\*  
 \** ***@param queryString*** *原生sql语句  
 \** ***@param params*** *sql的参数  
 \** ***@param pageIndex*** *当前页  
 \** ***@param pageSize*** *一页数据行数  
 \** ***@param onePage*** *查一页还是查两页（多的一页给前端做缓存用）  
 \** ***@param oldPage*** *历史页 可用于判断是向前发呢还是向后翻页以便于判断多取一页数据时是向前多取还是向后多取  
 \** ***@return*** *FastPagination  
 \*/***FastPagination** **cacheNextPagePaginationSql**(CharSequence queryString, List params, **int** pageIndex, **int** pageSize, Boolean onePage, **int** oldPage)

### 不返回记录总数的分页 并根据前端条件判断是否需要多查询一页数据给前端做缓存

*/\*\*  
 \** ***@param queryString*** *hql语句  
 \** ***@param pageIndex*** *当前页  
 \** ***@param pageSize*** *一页数据行数  
 \** ***@param onePage*** *查一页还是查两页（多的一页给前端做缓存用）  
 \** ***@param oldPage*** *历史页 可用于判断是向前发呢还是向后翻页以便于判断多取一页数据时是向前多取还是向后多取  
 \** ***@param params*** *可变参数  
 \** ***@return*** *FastPagination  
 \*/*FastPagination **cacheNextPagePaginationHql**(CharSequence queryString, **int** pageIndex, **int** pageSize, Boolean onePage, **int** oldPage, Object... params)

------------------------------------- 调用存储过程----------------------------------------

### 执行带输出参数存储过程

*/\*\*  
 \** ***@param callSql*** *存储过程名称  
 \** ***@param inParameters*** *输入参数  
 \** ***@return*** *返回值  
 \*/*String **prepareCallAndReturn**(**final** CharSequence callSql, **final** Object... inParameters)

### 执行带输出参数存储过程

*/\*\*  
 \** ***@param callSql*** *存储过程名称  
 \** ***@param inParameters*** *输入参数  
 \** ***@return*** *返回值  
 \*/*

List **prepareCallAndReturnList**(**final** CharSequence callSql, **final** Object... inParameters)

### 执行不带输出参数存储过程

*/\*\*  
 \** ***@param callSql*** *存储过程名称  
 \** ***@param inParameters*** *输入参数  
 \*/***void prepareCallNoReturn**(**final** CharSequence callSql, **final** Object... inParameters)

### 定制输出的存储过程

*/\*\*  
 \* \** ***@param callSql*** *存储过程名称  
 \** ***@param inParameters*** *输入参数  
 \** ***@param outTypeList*** *输出类型  
 \** ***@return*** *输出List  
 \*/*List<Object> **prepareCallAndReturnCustom**(**final** CharSequence callSql, **final**

List<Object> inParameters, **final** List<Integer> outTypeList)

------------------------------------- 其他方法----------------------------------------

### 获取JdbcTemplate对象

*/\*\*  
 \* \*/*

JdbcTemplate **getJdbcTemplate**()

### 获取NamedParameterJdbcTemplate对象

*/\*\*  
 \* \*/*

NamedParameterJdbcTemplate **getNpJdbcTemplate**()

### 获取session对象

*/\*\*  
 \* \*/*

Session **getSession**()

## 2.BaseService

数据服务层的类都要继承BaseService,因为BaseService为我们提供了一些通用的服务层接口。包括实体的增删改查

### 保存对象

*/\*\*  
 \* \** ***@param entity*** *\** ***@param <T>*** *\*/*<T> **void** save(T entity)

### 通过单个id删除对象

*/\*\*  
 \** ***@param entity*** *\** ***@param id*** *\** ***@param <T>*** *\*/*<T> **void** delete(Class<T> entity, Serializable id)

### 批量删除实体

*/\*\*  
 \*   
 \** ***@param entity*** *\** ***@param ids*** *\** ***@param <T>*** *\*/*<T> **void** deleteBatch(Class<T> entity, String ids)

### 级联删除

*/\*\*  
 \*   
 \** ***@param entity*** *\** ***@param id*** *\** ***@param <T>*** *\*/*<T> **void** deleteCascade(Class<T> entity, Serializable id)

### 通过id获取到实体对象

*/\*\*  
 \*   
 \** ***@param clazz*** *\** ***@param id*** *\** ***@param <T>*** *\** ***@return*** *\*/*<T> T get(Class<T> clazz, Serializable id)

### 通过属性查找单一的参数

*/\*\*  
 \** ***@param entityClass*** *\** ***@param propertyName*** *\** ***@param value*** *\** ***@param <T>*** *\** ***@return*** *\*/*<T> T findUniqueByProperty(Class<T> entityClass, String propertyName, Object value)

### 通过参数查看对象list

*/\*\*  
 \** ***@param entityClass*** *\** ***@param propertyName*** *\** ***@param value*** *\** ***@param <T>*** *\** ***@return*** *\*/*<T> List<T> findByProperty(Class<T> entityClass, String propertyName, Object value)

### 通过参数查看对象list

<T> List<T> getListBySqlT(Class<T> entityClass,CharSequence sql, Map<String, ?> params)

## 3.BaseController

所有的业务控制层都要继承BaseController，BaseController为我们抛出了大部分项目开发中遇到的异常，并进行日志记录。且为提供了当前登录人的信息和当前登录人组织机构的信息，为业务BaseController提供查询信息提供方便。

### 获取当前登录人的信息

SysUser **getLoginUser**(HttpSession session)

### 获取当前登录人组织机构的信息

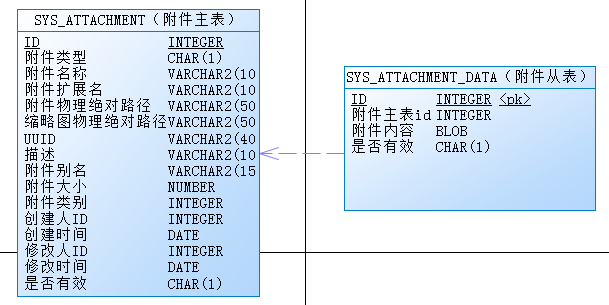
SysOrgan **getLoginOrgan**(HttpSession session)

## 2.附件（Attachment）

附件：

附件的存储在数据库中由两张表组成,分别是SYS\_ATTACHMENT和SYS\_ATTACHMENT\_DATA,在java实体中对应Attachment实体和AttachmentData实体，当我们只将附件保存到文件系统中时，我们只需要使用主表即可，如果我们需要将保存到数据库中，我们才会使用到我们的附件从表。

这里要介绍的是，当我们一次性上传多张附件时，在我们的附件主表中将会生成多条记录，但是它们具有相同的UUID，我们可以通过UUID将我们上传的附件展示，附件的类型我们可以通过附件类别字段进行区分。



基础平台已经提供了一整套从前端上传附件和展示附件的功能，若项目中有其他附件需求可以结合附件的API自定义操作。

### 保存Attachment实体

*/\*\* \** ***@param attachment*** *实体  
 \** ***@return*** *id  
 \*/*Long **saveAttachment**(Attachment attachment);

### 获取Attachment实体

*/\*\*  
 \** ***@param id*** *附件id  
 \*/*Attachment **getAttachmentById**(Long id);

### 删除文件

*/\*  
 \** ***@param idstr*** *附件ids  
 \*/***void delAttachment**(String idstr);

### 根据map获取附件列表

*/\*  
 \** ***@param map*** *查询条件  
\** ***@return*** *附件实体列表  
\*/*List<Attachment> **getAttachmentList**(Map map);

### 根据uuid获取附件列表

*/\*  
 \** ***@param uuid*** *附件id  
 \** ***@return*** *附件实体列表  
\*/*List<Attachment> **getAttachmentList**(String uuid);

### 重新生成附件缩略图路径

*/\*  
 \** ***@param id*** *附件id  
 \** ***@param thPath*** *路径  
 \*/***void updateThPath**(Long id, String thPath);

### 保存AttachmentData实体

*/\*\*  
 \** ***@param attachmentData*** *实体  
 \*/***void saveAttachmentData**(AttachmentData attachmentData);

### 获取AttachmentData实体

*/\*\*  
 \** ***@param attachmentId*** *附件id  
 \** ***@return*** *AttachmentData  
 \*/*AttachmentData **getDataByAttachmentId**(Long attachmentId);

### 删除AttachmentData实体

*/\*\*  
 \* \** ***@param attachmentId*** *附件id  
 \*/***void delDataByAttachmentId**(Long attachmentId);

### 上传附件

*/\*\*  
 \** ***@param alias*** *别名  
 \** ***@param description*** *描述  
 \** ***@param uuid*** *uuid  
 \** ***@param fjlbNo*** *附件类别分类  
 \** ***@param thumbFlag*** *缩略图是否生成标志  
 \** ***@param ifUnValid*** *缩略图是否生成标志 @return str  
 \*/*String **upload**(MultipartFile multipartFile, String alias,  
 String description, String uuid, String fjlbNo, **boolean** thumbFlag, **boolean** ifUnValid,String fj\_type);

### 获取附件类型为图片的实体列表

*/\*\*  
 \** ***@param uuid*** *附件id  
 \** ***@return*** *list  
 \*/*List<Attachment> **getLzPhotoList**(String uuid);

### 获取缩略图

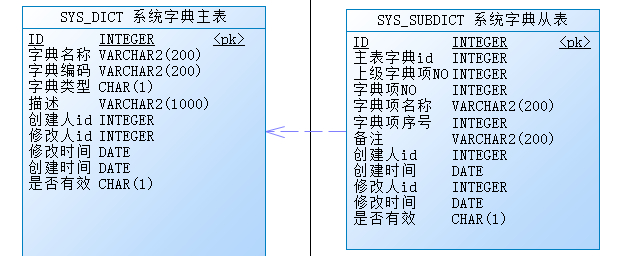
*/\*\*  
 \** ***@param id*** *附件id  
 \** ***@param thPath*** *缩略图路径  
 \** ***@param response*** *响应  
 \*/*

**void getThumbnail**(Long id, String thPath, HttpServletResponse response);

### 

## 3.字典（SysDict）

基础平台中的字典表包含了主表SYS\_DICT和字表SYS\_SUBDICT，对应java实体中的SysDict和SysSubDict,主表中保存了字典的名称和字典Code,从表中保存了字典项的NO。



我们可以通过基础平台后台对字典项的维护，对字典的增删改查。其它个性需求可结合字典后端接口个性操作。

### 保存字典实体

*/\*\*  
 \** ***@param sysDict*** *字典实体  
 \** ***@param rjspath*** *文件路径  
 \*/***void saveDict**(SysDict sysDict, String rjspath)

### 获取字典列表

*/\*\*  
 \** ***@param map*** *查询条件:dictName dictCode dictType  
 \** ***@return*** *分页信息  
 \*/*FastPagination **getDictList**(Map map);

### 获取字典实体

*/\*\*  
 \** ***@param id*** *字典id  
 \** ***@return*** *ar  
 \*/*SysDict **getDictById**(Long id);

### 删除字典

*/\*\*  
 \** ***@param dictId*** *字典id  
 \*/***void deleteDict**(Long dictId);

### 通过字典code查询字典list

*/\*\*  
 \** ***@param dictcode*** *字典code  
 \*/*List<Map<String, Object>> **getDictListByCode**(String dictcode);

### 获得下级字典ID通过字典Code和字典No

*\** ***@param dictCode*** *\** ***@param dictNo*** *\** ***@return*** *\*/***public** Integer **getSubdictIDByCodeAndNO**(String dictCode, String dictNo);

### 通过字典CODE和下级字典name获得字典NO

*/\*\*  
 \** ***@param dictCode*** *上级字典CODE  
 \** ***@param subDictName*** *字典名字  
 \** ***@return*** *字典NO  
 \*/*

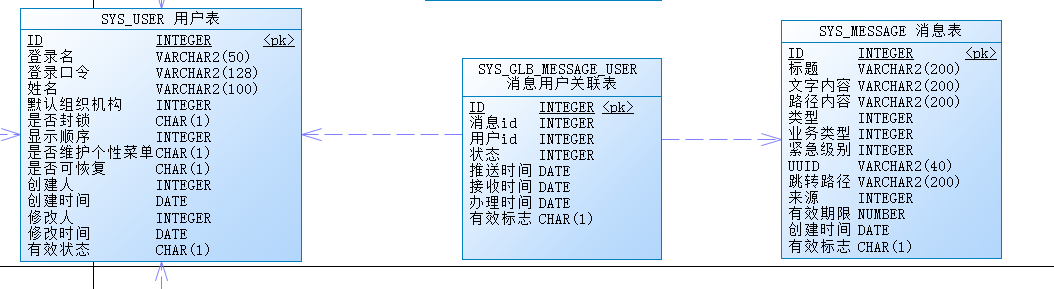
Integer **getCodeByName**(String dictCode, String subDictName);

### 通过字典CODE和下级字典NO获得下级字典的Name

*/\*\*  
 \** ***@param dictCode*** *上级字典CODE  
 \** ***@param subDictNo*** *下级字典No  
 \** ***@return*** *下级字典的Name  
 \*/*String **getNameByCode**(String dictCode, Integer subDictNo);

## 4.消息提醒（SysMessage）

消息提醒主要是用来保存系统消息提醒信息，消息提醒设计到消息的生产者和消息的查看。消息提醒在系统中主要由SYS\_GLB\_MESSAGE\_USER 和SYS\_MESSAGE两张表组成。



### 生成消息提醒接口

*/\*\*  
 \** ***@param ids*** *逗号拼接的字符串  
 \** ***@param idsType*** *指明ids类型 1：角色ids 2：用户ids  
 \** ***@param title*** *消息标题  
 \** ***@param textContent*** *文字内容  
 \** ***@param pathContent*** *路径内容  
 \** ***@param type*** *消息类型（字典项 1：阅读告知型 2：操作引导型）  
 \** ***@param ywlx*** *业务类型（字典项 由具体系统确定 1：预警提醒、2：异常提醒）  
 \** ***@param emgLevel*** *紧急级别（字典项 1：紧急 2：重要 3：一般）  
 \** ***@param uuid*** *uuid  
 \** ***@param ywcs*** *业务参数  
 \** ***@param skipPath*** *跳转路径  
 \** ***@param source*** *消息来源（字典项 由具体系统确定）  
 \** ***@param validTime*** *消息有效期限  
 \*/***void generateMsg**(String ids, String idsType, String title, String textContent,  
 String pathContent, String type, String ywlx, String emgLevel,  
 String uuid, String ywcs, String skipPath, String source, Double validTime);

### 获取消息列表

*/\*\*  
 \** ***@param map****：查询条件（userId 用户id、ywlx 业务类型、type 消息类型、emgLevel 紧急级别）  
 \** ***@return*** *分页信息  
 \*/*

FastPagination **getMsgList**(Map map);

### 更新消息用户关联表 状态、接收时间

*/\*\*  
 \** ***@param msgId*** *消息id  
 \** ***@param userId*** *用户id  
 \** ***@param status*** *状态  
 \** ***@param receiveTime*** *接收时间  
 \*/***void updateMsgUser**(Long msgId, Long userId, Integer status, Date receiveTime);

### 获取首页提醒总数及列表信息

*/\*\*  
 \** ***@param userId*** *用户id  
 \** ***@return*** *map  
 \*/*Map<String, Object> **getSyMsgList**(Long userId);

### 根据用户ID查询消息提醒未读信息

*/\*\*  
 \** ***@param id*** *\** ***@return*** *\*/*List<Map<String, Object>> **getSysMessageNoReadListByUserId**(Long id);

### 根据用户ID查询所有消息提醒信息

*/\*\*  
 \** ***@param map*** *\** ***@param id*** *\** ***@return*** *\*/* FastPagination **getSysMessageListByUserId**(Map map,Long id);

## 5.工具类

### 日期工具类（DateUtils）

### 文件工具类（RxFileUtils）

### 字符串工具类（RxStringUtils）

### 文件压缩下载类（ZipUtil）

### 反射工具类（ReflectionUtils）

### 将数字转化为财务形式的人民币大写（NumberToRMBUtil）

### 数据处理工具类OracleUtilsOracle

### Session工具类（SessionUtils）

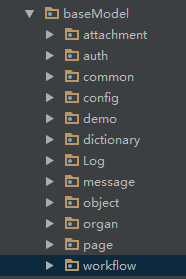
### 下划线和驼峰互转工具类（CamelCaseUtils）

### 枚举类（EnumUtil）

工具类：

对象工具类

图为基础模型的架构图，可以看到主要分为12个模块。



attachment 附件相关操作

auth 权限

config 配置文件

dictionary 字典

Log日志

message消息提醒

page 页面

organ 组织机构

workflow 流程

**Controller 层接口**

消息