Elab后端框架说明书

**文件变更记录**

\*A - 增加 M修订D - 删除

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 变更  版本号 | 日期 | 变更类型（A\*M\*D） | 编写 | 变更摘要 |
| V0.1 | 2016.12.10 16:58:28 | A | 刘海啸 | 初步创建，架构选型，框架搭建，目录结构建立 |
| V0.2 | 2016.12.18 21:55:05 | M | 刘海啸 | MVC框架制定，SQL语句处理 |
| V0.3 | 2016.12.20 18:07:08 | M | 刘海啸 | DB操作封装，多数据源封装，demo编写 |
| V0.4 | 2016.12.28 13:13:19 | M | 刘海啸 | DB操作优化，主要是SQL参数替换，更加简便 |
| V0.5 | 2016.12.30 23:02:25 | M | 刘海啸 | DB操作优化，主要是SQL参数替换的bug修复优化 |
| V0.6 | 2017.01.05 14:22:24 | M | 刘海啸 | 集合工具类，编写框架级别的缓存，框架级别的日志 |
| V0.7 | 2017.01.06 19:02:59 | M | 刘海啸 | 添加日志AOP和缓存AOP |
| V0.8 | 2017.01.09 21:21:53 | M | 刘海啸 | 集成用户，打通APP和service互通 |
| V0.9 | 2017.01.13 02:05:16 | M | 刘海啸 | 异常、缓存、用户、执行时间记录全部打通使用 |
| V1.0 | 2017.01.19 14:35:11 | M | 刘海啸 | 事物bug修复，封版 |
| V1.0.5 | 2017.03.18 10:51:22 | M | 刘海啸 | 开发版本日志，错误提示更加可视化 |
| V1.0.6 | 2017.03.21 10:21:12 | M | 刘海啸 | 加入枚举定义配置文件，所有常量配置化 |
| V1.0.7 | 2017.04.1 11:51:32 | M | 刘海啸 | 修复sql语句使之支持and(… or … or …)语句 |
| V1.0.8 | 2017.04.12 14:31:41 | M | 刘海啸 | 查询支持对象化，自动装箱，入参和出参对象化 |
| V1.0.9 | 2017.05.12 17:45 | M | 刘海啸 | 接入CAT日志系统，框架全面支持全生命周期日志查询 |
| V1.0.9 | 2017.05.22 17:45 | M | 刘海啸 | 优化修改语句，可以任意传是否存在的值 |
| V1.1.0 | 2017.11.02 18:41 | M | 刘海啸 | 查询语句增加排序传惨化 |
| V1.1.1 | 2018.02.24 17:36 | M | 刘凯雄 | 架构升级，集成可视化日 |

### 框架介绍

本框架项目由elab后端技术团队出品，在查看了市面上大部分的java服务端框架，经内部讨论，由于不想与外部对象化的技术一致，但是又不能偏离太多市场上的技术点，故采用spring作为底层框架，结合springMVC和我们自己对jdbc的处理，推出的一款简便的，直观的基于restfull的微服务架构。

#### 1、框架目录结构

Project-service

|—— com.elab.项目

|—— controllers

|—— daos

|—— services

|—— enums

|—— models

|—— utils

以上这些是固定目录，模块再在对应的包下建立模块包！

#### 2、采用技术

框架采用SpringMVC+Spring的技术进行搭建，数据库采用Mysql，数据源采用Druid，db操作使用SpringJDBC；

#### 3、运行方式

项目启动方式采用轻量级启动，采用maven插件jetty来启动，默认8080端口，如需要热加载请看如下配置：

jetty热部署注意事项：

启动事项：jetty:run -Djetty.reload=automatic -Djetty.scanIntervalSeconds=3

当修改了java文件时，在IntelliJ中按：‘Ctrl+Shift+F9’ 将重新编译该java文件，

如果修改了多个java文件，按‘Ctrl+F9’ 可以重新编译整个工程。

访问地址：<http://localhost:8080/elab>

8080端口的来源和elab的来源就是在web.xml中的配置，具体如下：

<connectors>  
 <connector implementation="org.eclipse.jetty.server.nio.SelectChannelConnector">  
 <port>8080</port>  
 <maxIdleTime>60000</maxIdleTime>  
 </connector>  
</connectors>  
<webAppConfig>  
 <contextPath>/elab</contextPath>  
</webAppConfig>

### 框架规范

#### dao层的写法以及规范

DAO层采用传统的接口实现类的方式进行，定义接口方法，抛出异常，书写对应的注释；示例：

public interface IHelloDao {

*/\*\*  
 \* 根据参数获取查询结果集  
 \** ***@param*** *params {“username”:”用户名”}  
 \** ***@return*** *\** ***@throws*** *CoreException   
 \*/*List<Map<String, Object>> getList(LinkedHashMap<String, Object> params) throws CoreException;

}

DAO对应的实现类，原则上实现类应该是放在对应模块下的impl文件夹下，如果是一接口一实现，则实现类名应该是去I后首字母大写最后再加上Impl，如果是多实现类则根据具体业务在进行划分，比如操作不同数据库（mysql、mongodb、oracle等等等，对应的实现类名称应该是：HelloDaoMongodbImpl，只是示例），重点是dao实现类中必须要继承BaseDao类，因为在BaseDao类中封装处理了对应的SQL处理，调用哪个XML文件进行SQL解析，以及返回值处理；

对应的增删改查示例如下：

获取数据库中多条结果集，返回的是一个数据集合列表：

返回参数有2中。

第一种是Map的键值对方式；

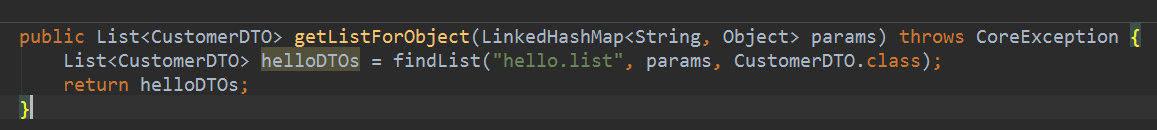
第二种是对象方式，对象命名必须是XXXDTO结尾的javaBean类;

@Component  
public class HelloDaoImpl extends BaseDao implements IHelloDao {

*/\*\*  
 \* 根据参数获取查询结果集  
 \** ***@param*** *params {“username”:”用户名”}  
 \** ***@return*** *\** ***@throws*** *CoreException   
 \*/*

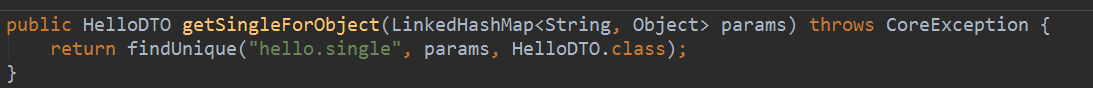
public List<Map<String, Object>> getList(LinkedHashMap<String, Object> params) throws CoreException {  
 List<Map<String, Object>> result = super.findList("hello.list", params);  
 return result;  
 }

}



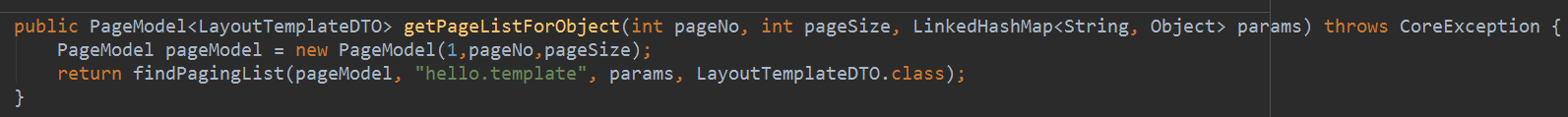
获取数据库中单一结果集，返回的是一行数据：

public Map<String, Object> getSingle(LinkedHashMap<String, Object> params) throws CoreException {  
 Map<String, Object> sqlRow = super.findUnique("hello.single", params);  
 return sqlRow;  
}



获取数据库中多条数据带分页功能：

public PageModel getPageList(int pageNo, int pageSize, LinkedHashMap<String, Object> params) throws CoreException {  
 PageModel pageModel = new PageModel(1,pageNo,pageSize);  
 return super.findPagingList(pageModel, "hello.pagelist", params);  
}



往数据库中新增一条记录：

public int insert(LinkedHashMap<String, Object> params) throws CoreException {  
 int id = this.executeInsert("hello.insert", params);  
 return id;  
}

根据对应的参数修改数据库中的记录：

public int update(LinkedHashMap<String, Object> params) throws CoreException {  
 int count = this.executeUpdate("hello.update", params);  
 return count;  
}

根据对应的参数参数数据库中的一条数据（**物理删除**）

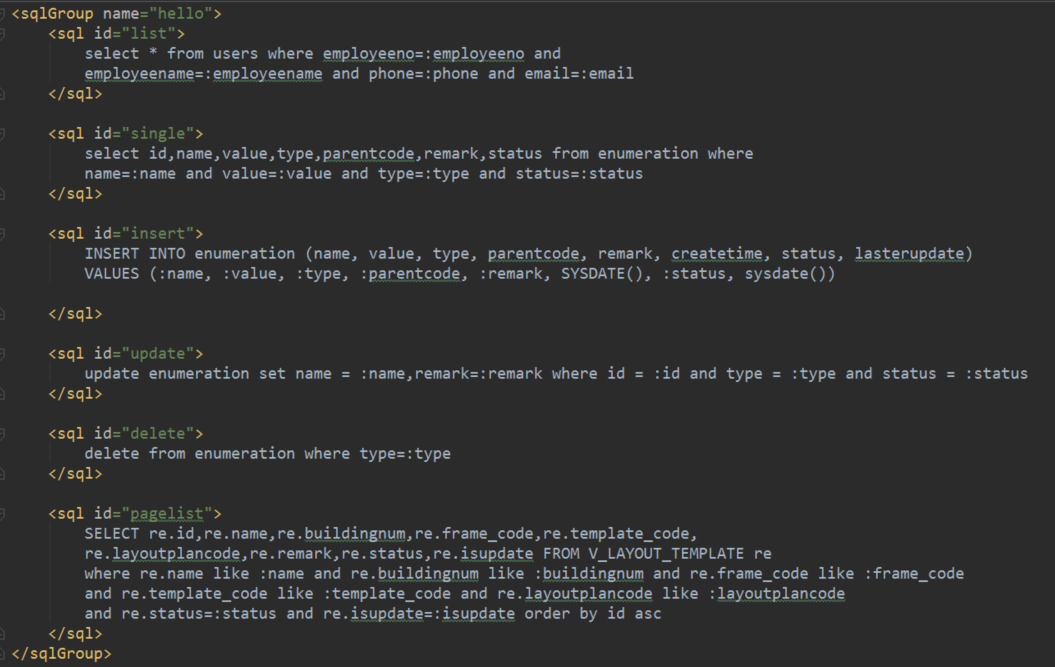
public int delete(LinkedHashMap<String, Object> params) throws CoreException {  
 int count = this.executeDelete("hello.delete", params);  
 return count;  
}

hello.list和hello.single和hello.pagelist和hello.insert和hello.update和hello.delete，这几个字符串的来源是在来源如下hello-sql.xml中,标记为红色的字体必须是一模一样；

原则上hello.XXX中的hello应该是hello-sql.xml，举例：user.detail，那么对应的sql文件名称应该是user-sql.xml（必须遵守）；

hello来源于如下xml中的sqlGroup中的属性name，

list和single和pagelist和insert和update和delete来源于如下xml中的SQLGroup下的子属性sql元素中的id属性，在BaseDao中会去resource目录下的sql目录下寻找hello-sql.xml文件中的对应的sql语句；



#### xxx-sql.xml 规范

##### **查询（select）**

1、所有where后面不能换行，有内容必须紧跟着；例如：select a from table where b=1

2、<号需要用转义符号替换 &lt; 因为这个小于号和xml中的尖括号冲突了

3、所有子查询的语句，如果有可变条件查询，则子查询中的可变条件必须是要有值，不能为null

4、如果涉及多张表多字段展示，但是又是多必要条件查询，则最好将该查询语句写成视图，用视图代替复杂SQL，这样出错几率小

5、查询参数如有多个，则条件查询的参数必须按照顺序来传值，例如：select a from table where b=:b and c=:c and d=:d ,参数顺序必须是b,c,d

6、in查询方式，需要使用List的方式进行替换；代码如下：

Select \* from table where name in (:name)

具体调用的service中：

List name = Arrays.asList(“数组”);

Map.put(“name”, name);

7、自定义排序写法采用参数替换法orderby，传参map中传：

Select id from table g order by :orderby

LinkedHashMap paramsMap = new LinkedHashMap();

paramsMap.put("orderby","g.followCreated desc");

8、自定义分组写法采用参数替换法groupby，传参map中传

Select id from table g group by :groupby

LinkedHashMap paramsMap = new LinkedHashMap();

paramsMap.put("groupby","g.mobile");

##### **新增（insert）**

1、传参的顺序必须和你insert中的语句字段顺序保持一致，并且个数也需要保持一致；例如：insert into table(a,b,c,d) values(:a,:b,:c,:d)

2、传参内容不需要关心是否有值，不管有没有值，按照顺序直接传进去即可；

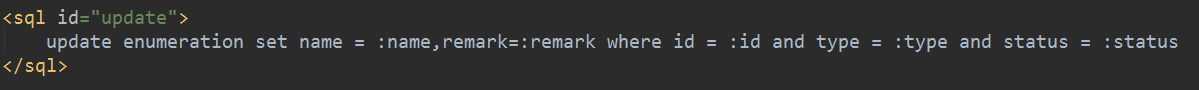
##### 修改**（update）**

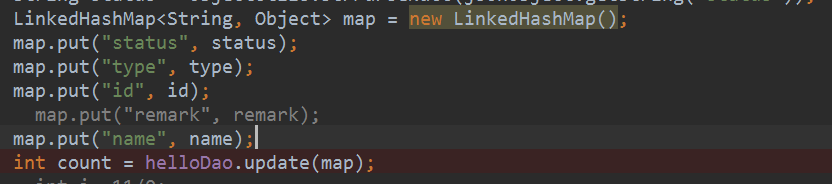
1、where 后面的内容不能用in和大于（>）小于（&lt;）之类的特殊用法，只支持普通的等号（=）；

2、基于遵守1的前提下，传参内容不需要关心是否有值，不管有没有值，不需要按照顺序传值，随便传值；例如：update table set a=:a,d=:d,c=:c where b=:b；

3、update语句的where后面必须带有一个条件，否则程序会直接报错

4、如果update中的参数值不想修改，但是语句想用同一条（比如：update table set name=:name, sex=:sex, status=:status where id=:id）,该语句中参数是name、sex、status、id，我可以直接只修改其中的status字段，直接传map对应的key是status，就可以修改，其他2个（name和sex）不传入map中即可达到统一update语句多次重复使用。代码如下：





##### 删除（**delete**）

1、介于删除操作的危险性，故不替换任何null的参数，sql中有多少条件，则传值必须传相应的多少参数；例如：delete from table where a=:a

#### Service层的写法以及规范

采用一接口多实现，并且传参的类型为String格式的json字符串，返回参数的类型也是String格式的json字符串，所有service层的注解，只能使用@Service，不能用其他的类注解。例子：



##### 多数据源支持

Service是支持多数据源的，如果需要多数据源操作，则在对应的方法上面加上注解@DataSource，写上对应的数据库名称；前提是在一个方法内部不能同时操作2个库，如有这种多库操作的需求，则需要写上单独方法；

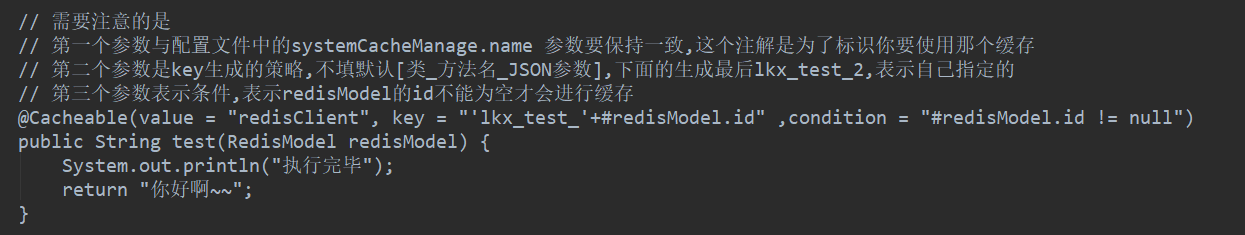
##### 缓存支持

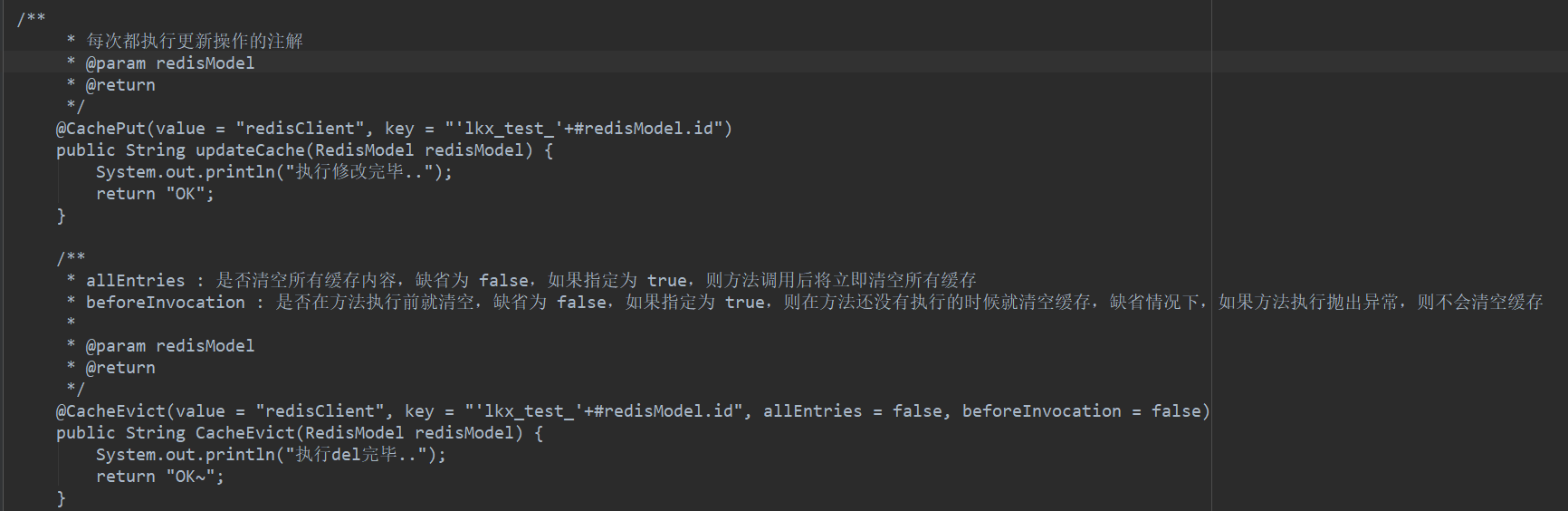
Service是支持方法级别的缓存的，目前Redis，如果对某个查询需要做缓存处理，扩展了Spring的缓存使用，使之可以用spring的注解来支持我们需要的缓存；

@Cacheable：加入缓存的注解

@CachePut：更新缓存内容的注解

@CacheEvict：删除缓存内容的注解





##### 异常支持

目前异常信息支持Service和controller捕获，好处是即使程序出问题了，也可以优雅的对调用者提供比较友好的错误信息；注意用法：

service的实现类中在每个方法的上面加上注解，@ExceptionHandle(username =”人名”, ModuleName = 类名.class, Throwable = true)，service层所有的异常信息原则上是全部需要对外抛出的， Throwable=true，这个在service层中是必须需要写成true，供controller层去解析并且捕获；

##### 方法体中的注意事项

LinkedHashMap中的参数的顺序必须和sql中需要替换的参数顺序必须保持一致；返回值必须是对象封装成json字符串；

#### Controller层的写法以及规范

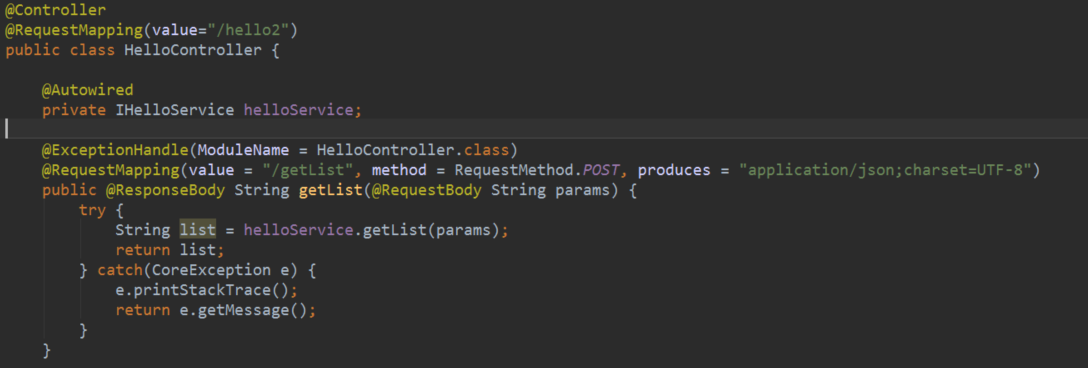
在controller的类上加上@Controller注解；

类级别的@RequestMapping注解必须加，value的值为具体模块名称，统一小写；

@ExceptionHandle：代表捕获Service层往外抛出的一切异常；

方法级别的@RequestMapping注解必须加，其中produces属性必须加上，并且要指明数据传输的格式和编码；

具体如下图所示：



#### 其他配置文件说明

##### global-config.xml

文件定义，全局配置文件：

配置框架的日志，开发环境、测试环境、堡垒环境、生产环境对应日志的存储媒介配置；以及对应的日志级别设置；

配置框架的缓存，开发环境、测试环境、堡垒环境、生产环境对应日志的存储媒介配置；存储内容中还是存储文件中，还是存储数据库中等等；

系统参数配置，采用map对象的形式，体现在key和value的存储方式上；

alreadyAdded\_BeiXuan对应在global-config.xml中配置的key;配置如下：

<global version="0.1">  
 <settings>

<property name="alreadyAdded\_BeiXuan" value="已经加入备选"/>

</settings>

</global>

获取如下：

根据key获取对应的值：ConfigManager.*getProperty*("alreadyAdded\_BeiXuan")

##### database.properties

数据库连接池参数配置，支持多数据源配置；

##### applicationContext-datasource.xml

框架数据源配置，扫描包配置

##### spring-mvc.xml

框架注解扫描配置；拦截器配置；其他自定义拦截器配置等等

##### enumeration.xml

和后台管理系统遥相呼应，原则上后台管理系统中的配置项有多少，这个文件里面的内容就有多少。提供公用方法获取对应的key的值；代码中获取对应的值：

10001对应在enumeration.xml中的配置如下：

<enumerations>  
 <enumeration type="10001" remark="小区环境">  
 <value name="游泳池" val="1"></value>  
 <value name="跑道" val="2"></value>  
 <value name="网球馆" val="3"></value>  
 <value name="幼儿园" val="4"></value>  
 <value name="便利店" val="5"></value>  
</enumeration>

获取如下：

获取key下的中文：ConfigEnumeration. getName("10001", "1");

得到的值：游泳池

##### 日志记录

日志记录和正常的log4j用法一样，只不过封装了一个固定的类，支持字符替换，以下两种写法均可，**原则上只记录重要逻辑业务日志**；

现有日志不变，还是便于开发定位问题，接入CAT日志系统，主要用于生产环境中的日志查看，问题分析，性能瓶颈分析用。不影响框架本身。

日志记录方式更新，采用log4J的方式，把对应的慢查询、info信息和error异常信息都单独区分，把log日志信息集成到CAT可视化平台，供查错误和定位问题，正常log4J的使用即可；