目录

[一、 工程规约 1](#_Toc482735154)

[(一) 应用分层 1](#_Toc482735155)

[二、 Java代码规约 2](#_Toc482735156)

[(一) 命名规约 2](#_Toc482735157)

[(二) 常量规约 3](#_Toc482735158)

[(三) 格式规约 3](#_Toc482735159)

[(四) OOP规约 5](#_Toc482735160)

[(五) 集合处理 6](#_Toc482735161)

[(六) 并发处理 6](#_Toc482735162)

[(七) 注释规约 6](#_Toc482735163)

[(八) 其他 7](#_Toc482735164)

[三、 CRM前端代码规约 8](#_Toc482735165)

[(一) 目录结构 8](#_Toc482735166)

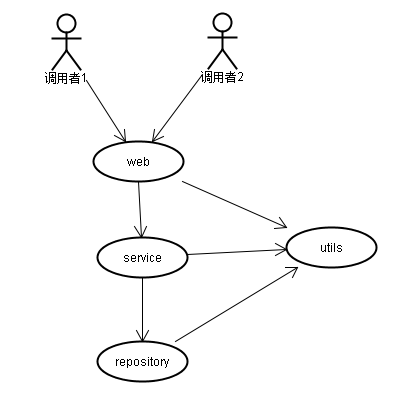
[四、 数据库规约 8](#_Toc482735167)

[(一) 建表规约 8](#_Toc482735168)

[(二) SQL规约 9](#_Toc482735169)

# 工程规约

## 应用分层



1. 【推荐】Web层参数校验，视图封装

Service层业务逻辑

Repository数据仓储层与数据库，mongodb，redis等交互

1. 【参考】分层领域模型规约：  
   •DO（ Data Object） ：与数据库表结构一一对应，通过 DAO 层向上传输数据源对象。  
   •DTO（ Data Transfer Object） ：数据传输对象， Service 和 Manager 向外传输的对象。  
   •BO（ Business Object） ：业务对象。 可以由 Service 层输出的封装业务逻辑的对象。  
   •QUERY：数据查询对象，各层接收上层的查询请求。 注：超过 2 个参数的查询封装，禁止  
   使用 Map 类来传输。  
   •VO（ View Object） ：显示层对象，通常是 Web 向模板渲染引擎层传输的对象。
2. 【强制】依赖于一个二方库群时，必须定义一个统一版本变量， 避免版本号不一致。  
   说明： 依赖 springframework-core,-context,-beans，它们都是同一个版本，可以定义一  
   个变量来保存版本： ${spring.version}，定义依赖的时候，引用该版本。

# Java代码规约

## 命名规约

1. 【强制】类名使用 UpperCamelCase 风格，必须遵从驼峰形式，但以下情形例外： （ 领域模型的相关命名） DO / BO / DTO / VO 等。  
   正例： MarcoPolo / UserDO / XmlService / TcpUdpDeal / TaPromotion  
   反例： macroPolo / UserDo / XMLService / TCPUDPDeal / TAPromotion
2. 【强制】方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase 风格，必须遵从  
   驼峰形式。  
   正例： localValue / getHttpMessage() / inputUserId
3. 【 强制】常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。  
   正例： MAX\_STOCK\_COUNT  
   反例： MAX\_COUNT
4. 【强制】抽象类命名使用 Abstract 或 Base 开头； 异常类命名使用 Exception 结尾； 测试类  
   命名以它要测试的类的名称开始，以 Test 结尾。
5. 【推荐】如果使用到了设计模式，建议在类名中体现出具体模式。  
   说明： 将设计模式体现在名字中，有利于阅读者快速理解架构设计思想。  
   正例： public class OrderFactory;  
   public class LoginProxy;  
   public class ResourceObserver;
6. 【参考】各层命名规约：  
   A) Service/DAO 层方法命名规约  
   1） 获取单个对象的方法用 get 做前缀。  
   2） 获取多个对象的方法用 list 做前缀。  
   3） 获取统计值的方法用 count 做前缀。  
   4） 插入的方法用 save（ 推荐） 或 insert 做前缀。  
   5） 删除的方法用 remove（ 推荐） 或 delete 做前缀。  
   6） 修改的方法用 update 做前缀。  
   B) 领域模型命名规约  
   1） 数据对象： xxxDO， xxx 即为数据表名。  
   2） 数据传输对象： xxxDTO， xxx 为业务领域相关的名称。  
   3） 展示对象： xxxVO， xxx 一般为网页名称。  
   4） POJO 是 DO/DTO/BO/VO 的统称，禁止命名成 xxxPOJO。

## 常量规约

1. 【强制】不允许出现任何魔法值（ 即未经定义的常量） 直接出现在代码中。  
   反例： String key="Id#taobao\_"+tradeId；
2. 【推荐】不要使用一个常量类维护所有常量，应该按常量功能进行归类，分开维护。如：缓存  
   相关的常量放在类： CacheConsts 下； 系统配置相关的常量放在类： ConfigConsts 下。  
   说明： 大而全的常量类，非得使用查找功能才能定位到修改的常量，不利于理解和维护。

## 格式规约

1. 【强制】 if/for/while/switch/do 等保留字与左右括号之间都必须加空格。
2. 【强制】任何运算符左右必须加一个空格。  
   说明： 运算符包括赋值运算符=、逻辑运算符&&、加减乘除符号、三目运行符等。
3. 【强制】 缩进采用 4 个空格，禁止使用 tab 字符。  
   说明： 如果使用 tab 缩进，必须设置 1 个 tab 为 4 个空格。 IDEA 设置 tab 为 4 个空格时，  
   请勿勾选 Use tab character；而在 eclipse 中，必须勾选 insert spaces for tabs。  
   正例： （ 涉及 1-5 点）  
   public static void main(String args[]) {  
   // 缩进 4 个空格  
   String say = "hello";  
   // 运算符的左右必须有一个空格  
   int flag = 0;  
   // 关键词 if 与括号之间必须有一个空格，括号内的 f 与左括号， 0 与右括号不需要空格  
   if (flag == 0) {  
   System.out.println(say);  
   }  
   // 左大括号前加空格且不换行；左大括号后换行  
   if (flag == 1) {  
   System.out.println("world");  
   // 右大括号前换行，右大括号后有 else，不用换行} else {  
   System.out.println("ok");  
   // 在右大括号后直接结束，则必须换行  
   }  
   }
4. 【强制】单行字符数限制不超过 120 个，超出需要换行，换行时遵循如下原则：  
   1） 第二行相对第一行缩进 4 个空格，从第三行开始，不再继续缩进，参考示例。  
   2） 运算符与下文一起换行。  
   3） 方法调用的点符号与下文一起换行。  
   4） 在多个参数超长，逗号后进行换行。  
   5） 在括号前不要换行，见反例。  
   正例：  
   StringBuffer sb = new StringBuffer();  
   //超过 120 个字符的情况下，换行缩进 4 个空格，并且方法前的点符号一起换行  
   sb.append("zi").append("xin")...  
   .append("huang")...  
   .append("huang")...  
   .append("huang");  
   反例：  
   StringBuffer sb = new StringBuffer();  
   //超过 120 个字符的情况下，不要在括号前换行  
   sb.append("zi").append("xin")...append  
   ("huang");  
   //参数很多的方法调用可能超过 120 个字符， 不要在逗号前换行  
   method(args1, args2, args3, ...  
   , argsX);
5. 【强制】方法参数在定义和传入时，多个参数逗号后边必须加空格。  
   正例： 下例中实参的"a",后边必须要有一个空格。  
   method("a", "b", "c");
6. 【推荐】方法体内的执行语句组、变量的定义语句组、不同的业务逻辑之间或者不同的语义  
   之间插入一个空行。相同业务逻辑和语义之间不需要插入空行。  
   说明： 没有必要插入多行空格进行隔开。

## OOP规约

1. 【强制】 Object 的 equals 方法容易抛空指针异常，应使用常量或确定有值的对象来调用  
   equals。  
   正例： "test".equals(object);  
   反例： object.equals("test");  
   说明： 推荐使用 java.util.Objects#equals （ JDK7 引入的工具类）
2. 【强制】关于基本数据类型与包装数据类型的使用标准如下：  
   1） 所有的 POJO 类属性必须使用包装数据类型。  
   2） RPC 方法的返回值和参数必须使用包装数据类型。  
   3） 所有的局部变量【 推荐】 使用基本数据类型。  
   说明： POJO 类属性没有初值是提醒使用者在需要使用时，必须自己显式地进行赋值，任何  
   NPE 问题，或者入库检查，都由使用者来保证。  
   正例： 数据库的查询结果可能是 null，因为自动拆箱，用基本数据类型接收有 NPE 风险。  
   反例： 比如显示成交总额涨跌情况，即正负 x%， x 为基本数据类型，调用的 RPC 服务，调用  
   不成功时，返回的是默认值，页面显示： 0%，这是不合理的，应该显示成中划线-。所以包装  
   数据类型的 null 值，能够表示额外的信息， 如：远程调用失败，异常退出。
3. 【推荐】循环体内，字符串的联接方式，使用 StringBuilder 的 append 方法进行扩展。  
   反例：  
   String str = "start";  
   for(int i=0; i<100; i++){  
   str = str + "hello";  
   }  
   说明： 反编译出的字节码文件显示每次循环都会 new 出一个 StringBuilder 对象，然后进行  
   append 操作，最后通过 toString 方法返回 String 对象，造成内存资源浪费。

## 集合处理

1. 【推荐】集合初始化时，尽量指定集合初始值大小。  
   说明： ArrayList 尽量使用 ArrayList(int initialCapacity) 初始化。
2. 【推荐】使用 entrySet 遍历 Map 类集合 KV，而不是 keySet 方式进行遍历。  
   说明： keySet 其实是遍历了 2 次，一次是转为 Iterator 对象，另一次是从 hashMap 中取出  
   key 所对应的 value。而 entrySet 只是遍历了一次就把 key 和 value 都放到了 entry 中，效  
   率更高。如果是 JDK8，使用 Map.foreach 方法。  
   正例： values()返回的是 V 值集合，是一个 list 集合对象； keySet()返回的是 K 值集合，是  
   一个 Set 集合对象； entrySet()返回的是 K-V 值组合集合。

## 并发处理

1. 【强制】创建线程或线程池时请指定有意义的线程名称，方便出错时回溯。  
   正例：  
   public class TimerTaskThread extends Thread {  
   public TimerTaskThread(){  
   super.setName("TimerTaskThread"); ...  
   }
2. 【强制】线程资源必须通过线程池提供，不允许在应用中自行显式创建线程。  
   说明： 使用线程池的好处是减少在创建和销毁线程上所花的时间以及系统资源的开销，解决资  
   源不足的问题。如果不使用线程池，有可能造成系统创建大量同类线程而导致消耗完内存或者  
   “过度切换”的问题。

## 注释规约

1. 【强制】 类、类属性、类方法的注释必须使用 Javadoc 规范，使用/\*\*内容\*/格式，不得使用  
   //xxx 方式。  
   说明： 在 IDE 编辑窗口中， Javadoc 方式会提示相关注释，生成 Javadoc 可以正确输出相应注  
   释； 在 IDE 中，工程调用方法时，不进入方法即可悬浮提示方法、参数、返回值的意义，提高  
   阅读效率
2. 【强制】方法内部单行注释，在被注释语句上方另起一行，使用//注释。方法内部多行注释  
   使用/\* \*/注释，注意与代码对齐。
3. 【推荐】代码修改的同时，注释也要进行相应的修改，尤其是参数、返回值、异常、核心逻辑  
   等的修改。  
   说明： 代码与注释更新不同步，就像路网与导航软件更新不同步一样，如果导航软件严重滞后，  
   就失去了导航的意义。

## 其他

1. 【强制】获取当前毫秒数 System.currentTimeMillis(); 而不是 new Date().getTime();  
   说明： 如果想获取更加精确的纳秒级时间值，用 System.nanoTime()。在 JDK8 中，针对统计  
   时间等场景，推荐使用 Instant 类。
2. 【推荐】 任何数据结构的构造或初始化，都应指定大小，避免数据结构无限增长吃光内存
3. 【强制】不要捕获 Java 类库中定义的继承自 RuntimeException 的运行时异常类，如：  
   IndexOutOfBoundsException / NullPointerException，这类异常由程序员预检查  
   来规避，保证程序健壮性。  
   正例： if(obj != null) {...}  
   反例： try { obj.method() } catch(NullPointerException e){...}
4. 【强制】对大段代码进行 try-catch，这是不负责任的表现。 catch 时请分清稳定代码和非稳  
   定代码，稳定代码指的是无论如何不会出错的代码。对于非稳定代码的 catch 尽可能进行区分  
   异常类型，再做对应的异常处理。
5. 【强制】有 try 块放到了事务代码中， catch 异常后，如果需要回滚事务，一定要注意手动回  
   滚事务。

# CRM前端代码规约

## 目录结构

1. 【强制】

Src/main/webapp

/static/静态资源

/WEB-INFO/jsp/业务页面

/ CommonFunction.tld

1. 【强制】crm项目是富客户端应用，所有页面都由div拼接而成，为了避免ID或者js、css选择器冲突，js方法等需要选择被引用的都必须添加${t}后缀,${t}是后台返回的时间戳
2. 【强制】页面所有ajax必须用base.js里面封装的ajax
3. 【强制】页面所有弹窗必须用base.js里面封装的opendialog
4. 【强制】所有后台接口跳转页面必须和页面文件名一致方面查询

# 数据库规约

## 建表规约

1. 【强制】小数类型为 decimal，禁止使用 float 和 double。  
   说明： float 和 double 在存储的时候，存在精度损失的问题，很可能在值的比较时，得到不  
   正确的结果。如果存储的数据范围超过 decimal 的范围，建议将数据拆成整数和小数分开存储。
2. 【推荐】表必备三字段： id, gmt\_create, gmt\_modified。  
   说明： 其中 id 必为主键，类型为 unsigned bigint、单表时自增、步长为 1。 gmt\_create,  
   gmt\_modified 的类型均为 date\_time 类型。

## SQL规约

1. 【强制】不要使用 count(列名)或 count(常量)来替代 count(\*)， count(\*)就是 SQL92 定义  
   的标准统计行数的语法，跟数据库无关，跟 NULL 和非 NULL 无关。  
   说明： count(\*)会统计值为 NULL 的行，而 count(列名)不会统计此列为 NULL 值的行。
2. 【强制】 count(distinct col) 计算该列除 NULL 之外的不重复数量。注意 count(distinct  
   col1, col2) 如果其中一列全为 NULL，那么即使另一列有不同的值，也返回为 0。
3. 【强制】当某一列的值全是 NULL 时， count(col)的返回结果为 0，但 sum(col)的返回结果为  
   NULL，因此使用 sum()时需注意 NPE 问题。  
   正例： 可以使用如下方式来避免 sum 的 NPE 问题： SELECT IF(ISNULL(SUM(g)),0,SUM(g))  
   FROM table;
4. 【强制】使用 ISNULL()来判断是否为 NULL 值。注意： NULL 与任何值的直接比较都为 NULL。  
   说明：  
   1） NULL<>NULL 的返回结果是 NULL， 而不是 false。  
   2） NULL=NULL 的返回结果是 NULL， 而不是 true。  
   3） NULL<>1 的返回结果是 NULL，而不是 true。
5. 【强制】 在代码中写分页查询逻辑时，若 count 为 0 应直接返回，避免执行后面的分页语句。
6. 【强制】数据订正时，删除和修改记录时，要先 select，避免出现误删除，确认无误才能执  
   行更新语句。
7. 【参考】 @Transactional 事务不要滥用。事务会影响数据库的 QPS，另外使用事务的地方需  
   要考虑各方面的回滚方案，包括缓存回滚、搜索引擎回滚、消息补偿、统计修正等。