****

**瑞才科技**

**教学管理平台**

**开发规范**

**V1.0**

**瑞才科技**

**2017.3**

文档变更记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **版本号** | **修改点说明** | **变更人** | **变更日期** | **审批人** | **审批日期** |
| V1.0 | 创建 | 陈琨 | 2017-3-15 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

目录

[1. 前言 3](#_Toc477452718)

[2. 编程规约 3](#_Toc477452719)

[2.1 命名规约 3](#_Toc477452720)

[2.2 格式规约 4](#_Toc477452721)

[2.3 OOP规约 5](#_Toc477452722)

[2.4 注释规约 6](#_Toc477452723)

[3. 异常日志规约 7](#_Toc477452724)

[3.1 异常处理规约 7](#_Toc477452725)

[3.2 日志处理规约 8](#_Toc477452726)

[4. 数据库规约 9](#_Toc477452727)

[4.1 建表规约 9](#_Toc477452728)

[4.2 索引规约 10](#_Toc477452729)

[4.3 SQL规约 10](#_Toc477452730)

[4.4 ORM规约 11](#_Toc477452731)

[5. 工程规约 11](#_Toc477452732)

# 前言

瑞才科技教学管理平台的开发需要多团队多人的协同配合，为了提高开发效率、降低协同成本、降低后期维护成本，需要对平台开发过程中的多方面作出规范。

开发过程的不规范会大大降低本平台的交付质量，比如：数据库的表结构和索引设计缺陷可能带来软件上的架构缺陷或性能风险；工程结构混乱导致维护困难；命名和注释不规范会导致平台的迭代更新困难等等。为了提高本平台的交付质量，各位开发者需要认真落实本开发规范。

本手册以Java开发者为中心视角，划分为编程规约、异常日志规约、数据库规约、工程规约、安全规约五大块，再根据内容特征，细分成若干二级子目录。根据约束力强弱及故障敏感性，规约依次分为强制、推荐、参考三大类。对于规约里的内容，“说明”对内容做了引申和解释；“正例”为提倡的编码和实现方式；“反例”为需要提防的雷区，以及真实的错误案例。

# 编程规约

## 命名规约

1. 【强制】代码中的命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束。

反例：\_name / \_\_name / $Object / name\_ / name$ / Object$

1. 【强制】代码中的命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。

说明：正确的英文拼写和语法可以让阅读者易于理解，避免歧义。注意，即使纯拼音命名方式也要避免采用。

反例： DaZhePromotion[打折] / getPingfenByName()[评分]

正例：DiscountPromotion[打折促销]/getScoreByName()[根据姓名获取评分]

1. 【强制】类名使用UpperCamelCase风格，必须遵从驼峰形式

反例：macroPolo/UserDo/XMLService/TCPUDPDeal/TAPromotion

正例：MarcoPolo/UserDO/XmlService/TcpUdpDeal/TaPromotion

1. 【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

1. 【强制】中括号是数组类型的一部分。

反例：String args[];

正例：String[] args;

1. 【强制】POJO 类中布尔类型的变量，都不要加 is，否则部分框架解析会引起序列化错误。

反例：定义为基本数据类型Boolean isSuccess；的属性，它的方法也是 isSuccess()，RPC框架在反向解析的时候，“以为”对应的属性名称是success，导致属性获取不到，进而抛出异常

1. 【强制】杜绝完全不规范的缩写， 避免望文不知义。

反例：AbstractClass“缩写”命名成AbsClass；condition“缩写”命名成condi，此类随意缩写严重降低了代码的可阅读性

1. 【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;

1. 【推荐】接口类中的方法和属性不要加任何修饰符号（public也不要加），保持代码的简洁性，并加上有效的Javadoc注释。尽量不要在接口里定义变量，如果一定要定义变量，肯定是与接口方法相关，并且是整个应用的基础常量。

反例： 接口方法定义： public abstract void f();

## 格式规约

1. 【强制】大括号的使用约定。如果是大括号内为空，则简洁地写成{}即可，不需要换行；如果是非空代码块则：
2. 左大括号前不换行
3. 左大括号后换行
4. 右大括号前换行
5. 右大括号后还有else等代码则不换行；表示终止右大括号后必须换行
6. 【强制】if/for/while/switch/do等保留字与左右括号之间都必须加空格。
7. 【强制】任何运算符左右必须加一个空格。
8. 【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。

1. 【强制】IDE的text file encoding设置为UTF-8;IDE中文件的换行符使用Unix 格式，不要使用windows格式。
2. 【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】避免通过一个类的对象引用访问此类的静态变量或静态方法，无谓增加编译器解析成本，直接用类名来访问即可。
2. 【强制】所有的覆写方法，必须加@Override 注解。

说明：getObject()与get0bject()的问题。一个是字母的 O，一个是数字的0，加@Override可以准确判断是否覆盖成功。另外，如果在抽象类中对方法签名进行修改，其实现类会马上编译报错。

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。

反例：object.equals("test");

反例：object.equals("test");

1. 【强制】构造方法里面禁止加入任何业务逻辑，如果有初始化逻辑，请放在 init 方法中。
2. 【推荐】为POJO类编写toString方法。使用 IDE 的中工具： source> generate toString时，如果继承了另一个POJO类，注意在前面加一下super.toString。
3. 【推荐】当一个类有多个构造方法，或者多个同名方法，这些方法应该按顺序放置在一起，便于阅读。
4. 【推荐】类内方法定义顺序依次是：公有方法或保护方法 > 私有方法 > getter/setter方法。

说明：公有方法是类的调用者和维护者最关心的方法，首屏展示最好；保护方法虽然只是子类关心，也可能是“模板设计模式”下的核心方法；而私有方法外部一般不需要特别关心，是一个黑盒实现；因为方法信息价值较低，所有Service和DAO的getter/setter方法放在类体最后。

1. 【推荐】循环体内，字符串的连接方式，使用StringBuilder的append方法进行扩展。

说明：在循环中使用“+”进行字符串拼接的代码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。

1. 【推荐】下列情况， 声明成 final 会更有提示性：
2. 不需要重新赋值的变量，包括类属性、局部变量。
3. 对象参数前加 final，表示不允许修改引用的指向。
4. 类方法确定不允许被重写。
5. 【推荐】类成员与方法访问控制从严：
6. 如果不允许外部直接通过new来创建对象，那么构造方法必须是 private。
7. 类非static成员变量并且仅在本类使用，必须是private。
8. 若是static成员变量，必须考虑是否为final。
9. 类成员方法只供类内部调用，必须是private。
10. 类成员方法只对继承类公开，那么限制为protected。

说明：任何类、方法、参数、变量，严控访问范围。过宽泛的访问范围，不利于模块解耦。

## 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。

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。

1. 【强制】所有的类都必须添加创建者信息。
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。
4. 【参考】好的命名、代码结构是自解释的，注释力求精简准确、表达到位。避免出现注释的一个极端：过多过滥的注释，代码的逻辑一旦修改，修改注释是相当大的负担。

反例：

// put elephant into fridge

put(elephant, fridge);

方法名 put，加上两个有意义的变量名elephant和fridge，已经说明了这是在干什么，语义清晰的代码不需要额外的注释。

# 异常日志规约

## 异常处理规约

1. 【强制】Java类库中定义的一类RuntimeException可以通过预先检查进行规避，而不应该通过catch来处理，比如：IndexOutOfBoundsException， NullPointerException等等。

正例： if (obj != null) {...}

1. 【强制】异常不要用来做流程控制，条件控制，因为异常的处理效率比条件分支低。
2. 【强制】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。
3. 【强制】最外层的业务使用者，必须处理异常，将其转化为用户可以理解的

内容。

1. 【强制】不能在finally块中使用return，finally块中的return返回后方法结束执行，不会再执行try块中的return语句。
2. 【推荐】建议使用try-with-resources方式使用资源和流对象，保证资源和流的及时关闭，防止资源泄露。
3. 【推荐】方法的返回值可以为 null，不强制返回空集合，或者空对象等，必须添加注释充分说明什么情况下会返回null值。调用方需要进行null判断防止NPE 问题。
4. 【推荐】防止 NPE，是程序员的基本修养，注意 NPE 产生的场景：
5. 返回类型为包装数据类型，有可能是 null，返回 int 值时注意判空。

反例： public int f() { return Integer 对象}; 如果为 null，自动解箱抛 NPE。

1. 数据库的查询结果可能为 null。
2. 集合里的元素即使isNotEmpty，取出的数据元素也可能为null。
3. 级联调用obj.getA().getB().getC()；一连串调用，易产生NPE。
4. 【推荐】避免直接使用RuntimeException 抛出，更不允许抛出Exception 或者Throwable，应使用有业务含义的自定义异常。推荐业界已定义过的自定义异常，如：DAOException / ServiceException 等。
5. 【推荐】避免出现重复的代码（Don’t Repeat Yourself），即DRY原则。

说明：随意复制和粘贴代码，必然会导致代码的重复，在以后需要修改时，需要修改所有的副本，容易遗漏。必要时抽取共性方法，或者抽象公共类，甚至是共用模块。

## 日志处理规约

1. 【强制】应用中不可直接使用日志系统（Log4j、Logback）中的 API，而应依赖使用日志框架SLF4J中的API，使用门面模式的日志框架，有利于维护和各个类的日志处理方式统一。
2. 【强制】日志文件推荐至少保存15天，因为有些异常具备以“周”为频次发生的特点。
3. 【强制】对trace/debug/info级别的日志输出，必须使用条件输出形式或者使用占位符的方式。
4. 【强制】避免重复打印日志，浪费磁盘空间，务必在log4j.xml中设置 additivity=false。
5. 【强制】日志中的异常信息应该包括两类信息：案发现场信息和异常堆栈信息。
6. 【推荐】谨慎地记录日志。生产环境禁止输出debug日志； 有选择地输出 info日志； 如果使用warn来记录刚上线时的业务行为信息，一定要注意日志输出量的问题，避免把服务器磁盘撑爆，并记得及时删除这些观察日志。

# 数据库规约

## 建表规约

1. 【强制】表达是与否概念的字段，必须使用is\_xxx的方式命名
2. 【强制】表名、字段名必须使用小写字母或数字，禁止出现数字开头，禁止两个下划线中间只出现数字。

反例：GetterAdmin，taskConfig，level\_3\_name

正例：getter\_admin，task\_config，level3\_name

1. 【强制】禁用保留字，如desc、range、match、delayed等。
2. 【强制】表名不使用复数名词。
3. 【强制】主键索引名为 pk\_字段名； 唯一索引名为 uk\_字段名； 普通索引名则为 idx\_字段名。
4. 【强制】varchar是可变长字符串，不预先分配存储空间，长度不要超过5000，如果存储长度大于此值，定义字段类型为text，独立出来一张表， 用主键来对应，避免影响其它字段索引效率。
5. 【参考】合适的字符存储长度，不但节约数据库表空间、节约索引存储，更重要的是提升检索速度。

正例：无符号值可以避免误存负数，且扩大了表示范围。

## 索引规约

1. 【强制】业务上具有唯一特性的字段，即使是组合字段，也必须建成唯一索引。
2. 【强制】在varchar字段上建立索引时，必须指定索引长度，没必要对全字段建立索引，根据实际文本区分度决定索引长度即可。
3. 【推荐】如果有order by的场景，请注意利用索引的有序性。order by最后的字段是组合索引的一部分，并且放在索引组合顺序的最后，避免出现file\_sort的情况，影响查询性能。

## SQL规约

1. 【强制】不要使用count(列名)或count(常量)来替代count(\*)，count(\*)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。
2. 【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。
3. 【强制】不得使用外键与级联，一切外键概念必须在应用层解决。
4. 【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。
5. 【推荐】in操作能避免则避免，若实在避免不了，需要仔细评估in后边的集合元素数量，控制在1000个之内。
6. 【参考】如果有全球化需要，所有的字符存储与表示，均以utf-8编码，此时字符计数使用SELECT CHARACTER\_LENGTH()。
7. 【参考】TRUNCATE TABLE比DELETE速度快，且使用的系统和事务日志资源少，但TRUNCATE无事务且不触发trigger，有可能造成事故，故不建议在开发代码中使用此语句。

说明：TRUNCATE TABLE在功能上与不带WHERE子句的DELETE语句相同。

## ORM规约

1. 【强制】在表查询中，一律不要使用\*作为查询的字段列表，需要哪些字段必须明确写明。

说明：1）增加查询分析器解析成本。2）增减字段容易与resultMap配置不一致。

1. 【强制】POJO类的Boolean属性不能加is，而数据库字段必须加is\_，要求在resultMap中进行字段与属性之间的映射。

说明：在<resultMap>中增加映射，是必须的。在MyBatis Generator生成的代码中，需要进行适当的修改。

1. 【强制】不允许直接拿HashMap与Hashtable作为查询结果集的输出。
2. 【推荐】执行SQL时，尽量不要更新无改动的字段，一是易出错；二是效率低；三是增加binlog存储。
3. 【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。

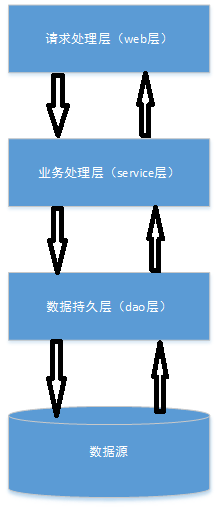
# 工程规约

1. 【强制】包名统一使用小写，点分隔符之间有且仅有一个自然语义的英语单词。包名统一使用单数形式，但是类名如果有复数含义，类名可以使用复数形式。

正例：应用工具类包名为 com.ruicai.util、类名为 MessageUtils（此规则参考

spring的框架结构）

1. 【强制】应用分层如下：



1. Web层：主要是对访问控制进行转发，各类基本参数校验，或者不复用的业务简单处理等
2. Service层：相对具体的业务逻辑服务层
3. DAO层：数据访问层，与底层Oracle进行数据交互
4. 【强制】在DAO层，产生的异常类型有很多，无法用细粒度的异常进行 catch，使用catch(Exception e)方式，并throw new DAOException(e)，不需要打印日志，因为日志在Service层一定需要捕获并打到日志文件中去。在Service层出现异常时，必须记录出错日志到磁盘，尽可能带上参数信息。Web层绝不应该继续往上抛异常，因为已经处于顶层，无继续处理异常的方式
5. 【参考】分层领域模型规约：
6. DO（DataObject）：与数据库表结构一一对应，通过DAO层向上传输数据源对象
7. DTO（Data Transfer Object）：数据传输对象，Service向外传输的对象
8. BO（Business Object）：业务对象。可以由Service层输出的封装业务逻辑的对象
9. 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
10. 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