**后端开发规范**

**定制应用开发工作流程规范**

**后端开发工作流程规范**

为了利于项目维护以及规范开发，促进成员之间Code Review的效率，提出以下开发规范，根据约束力强弱， 规约依次分为强制、推荐、参考三大类：

【强制】必须遵守，违反本约定需要进行修改订正；

【推荐】尽量遵守，长期遵守有助于系统稳定性和合作效率的提升；

【参考】充分理解，技术意识的引导，是个人学习、团队沟通、项目合作的方向。

## Java开发规范

(一) 命名风格

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

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

2.【强制】所有编程相关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，纯拼音命名方式更要避免采用。

3. 【强制】类名使用 UpperCamelCase 风格，但以下情形例外：DO / BO / DTO / VO / AO / PO / UID 等。

正例：ForceCode / UserDO / HtmlDTO / XmlService / TcpUdpDeal / TaPromotion

反例：forcecode / UserDo / HTMLDto / XMLService / TCPUDPDeal / TAPromotion

4. 【强制】方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase 风格。

正例：

localValue / getHttpMessage() / inputUserId

5. 【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 / CACHE\_EXPIRED\_TIME

反例：MAX\_COUNT / EXPIRED\_TIME

6.【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。

7. 【强制】类型与中括号紧挨相连来表示数组。

正例：定义整形数组 int[] arrayDemo;

反例：在 main 参数中，使用 String args[]来定义。

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

9. 【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.digiwin.athena.app.util、类名为 MessageUtils（此规则参考 spring 的框架结构）

10.【强制】避免在子父类的成员变量之间、或者不同代码块的局部变量之间采用完全相同的命名使可读性降低。

11.【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。

12.【推荐】为了达到代码自解释的目标，任何自定义编程元素在命名时，使用尽量完整的单词组合来表达。

正例：在 JDK 中，对某个对象引用的 volatile 字段进行原子更新的类名为：AtomicReferenceFieldUpdater。

反例：常见的方法内变量为 int a;的定义方式。

13.【推荐】在常量与变量的命名时，表示类型的名词放在词尾，以提升辨识度。

正例：startTime / workQueue / nameList / TERMINATED\_THREAD\_COUNT

反例：startedAt / QueueOfWork / listName / COUNT\_TERMINATED\_THREAD

14.【推荐】如果模块、接口、类、方法使用了设计模式，在命名时需体现出具体模式。

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理念。

正例：

public class OrderFactory;

public class LoginProxy;

public class ResourceObserver;

15.【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，确定与接口方法相关，并且是整个应用的基础常量。

16.【强制】对于 Service 和 DAO 类，基于 SOA 的理念，暴露出来的服务一定是接口，内部的实现类Impl 的后缀与接口区别。

正例：CacheServiceImpl 实现 CacheService 接口。

17.【参考】枚举类名带上 Enum 后缀，枚举成员名称需要全大写，单词间用下划线隔开。

说明：枚举其实就是特殊的常量类，且构造方法被默认强制是私有。

正例：枚举名字为 ProcessStatusEnum 的成员名称：SUCCESS / UNKNOWN\_REASON。

18.【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，复数结尾，如：listObjects。

3）获取统计值的方法用 count 做前缀。

4）插入的方法用 save/insert 做前缀。

5）删除的方法用 remove/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。

(二) 常量定义

1. 【强制】不允许任何魔法值（即未经预先定义的常量）直接出现在代码中。

2. 【强制】在 long 或者 Long 赋值时，数值后使用大写的 L，不能是小写l，小写容易跟数字混淆，造成误解。

说明：Long a = 2l; 写的是数字的 21，还是 Long 型的 2。

3. 【推荐】不要使用一个常量类维护所有常量，要按常量功能进行归类，分开维护。

说明：大而全的常量类，杂乱无章，使用查找功能才能定位到修改的常量，不利于理解，也不利于维护。

正例：缓存相关常量放在类 CacheConsts 下；系统配置相关常量放在类 ConfigConsts 下。

(三) 代码格式

1. 【强制】如果是大括号内为空，则简洁地写成{}即可，大括号中间无需换行和空格；如果是非空代码块则：

1） 左大括号前不换行。

2） 左大括号后换行。

3） 右大括号前换行。

4） 右大括号后还有 else 等代码则不换行；表示终止的右大括号后必须换行。

2. 【强制】左小括号和右边相邻字符之间不出现空格；右小括号和左边相邻字符之间也不出现空格；而左大括号前需要加空格。详见第 5 条下方正例提示。

反例：if (空格 a == b 空格)

3.【强制】if/for/while/switch/do 等保留字与括号之间都必须加空格。

4. 【强制】任何二目、三目运算符的左右两边都需要加一个空格。

说明：包括赋值运算符=、逻辑运算符&&、加减乘除符号等。

5. 【强制】注释的双斜线与注释内容之间有且仅有一个空格。

正例：

// 这是示例注释，请注意在双斜线之后有一个空格

String commentString = new String();

6. 【强制】在进行类型强制转换时，右括号与强制转换值之间不需要任何空格隔开。

正例：

long first = 1000000000000L;

int second = (int)first + 2;

7. 【推荐】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）方法调用中的多个参数需要换行时，在逗号后进行。

5）在括号前不要换行。

8. 【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 args1，后边必须要有一个空格。

method(args1, args2, args3);

9.【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。

10.【推荐】单个方法的总行数不超过 80 行。

正例：代码逻辑分清红花和绿叶，个性和共性，绿叶逻辑单独出来成为额外方法，使主干代码更加清晰；共性逻辑抽取成为共性方法，便于复用和维护。

11.【推荐】不同逻辑、不同语义、不同业务的代码之间插入一个空行分隔开来以提升可读性。

说明：任何情形，没有必要插入多个空行进行隔开。

12.【推荐】 传参规范：根据阿里传参规范

对于所传的参数不超过5个不要封成对象，以JSON格式得到对应的参数值

以JSON格式获取参数

JSONObject parameter = EAIRequest.build(messageBody).getJsonObject()

.getJSONObject("std\_data").getJSONObject("parameter");

当参数超过5个允许使用对象,入参反序列化

EAIRequest request = EAIRequest.build(messageBody);

List<?> outsourcingStockInInfoCreateList =request.getToList("outsourcing\_stock\_info", ?.class);

（四）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】所有的抽象方法（包括接口中的方法）必须要用 Javadoc 注释、除了返回值、参数、

异常说明外，还必须指出该方法做什么事情，实现什么功能。

说明：对子类的实现要求，或者调用注意事项，请一并说明。

3. 【强制】所有的类都必须添加创建者和创建日期。

说明：在设置模板时，注意 IDEA 的@author 为`${USER}`，而 eclipse 的@author 为`${user}`，大小写有

区别，而日期的设置统一为 yyyy/MM/dd 的格式。

正例：

/\*\*

\* @author xxx

\* @date xxxx/xx/xx

\*/

4. 【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。

5. 【强制】所有的枚举类型字段必须要有注释，说明每个数据项的用途

6. 【推荐】与其“半吊子”英文来注释，不如用中文注释把问题说清楚。专有名词与关键字保持英文原文即可。

反例：“TCP 连接超时”解释成“传输控制协议连接超时”，理解反而费脑筋。

7. 【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。

1. 【推荐】在类中删除未使用的任何字段和方法；在方法中删除未使用的任何参数声明与内部变量。

(四) 面向对象规范

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】所有的重写方法，必须加@Override 注解。

反例：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。

3，【强制】不能使用过时的类或方法。

说明：本条已在java10中得到禁止。

4，【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）Objects.equals(args1, args2);

5，【强制】所有的相同类型的包装类型对象之间值的比较，全部使用 equals 方法比较。

说明：对于 Integer var=?在-128 至 127 之间的赋值，Integer 对象是在 IntegerCache.cache 产生，会复用已有对象，这个区间内的 Integer 值可以直接使用==进行判断，但是这个区间之外的所有数据，都会在堆上产生，并不会复用已有对象，这是一个大坑，

推荐使用 equals 方法进行判断。

## Redis开发规范

1. 【强制】键（Key）命名规范：

使用统一的命名规范，如使用冒号（:）作为分隔符表示层级关系。

避免使用过长或过短的键名，以节省内存空间和提高查询性能。

命名应当按照以下规范：APPID:业务KEY1:业务KEY1:XXX

正例： “KCF:UserInfo:donglh”:“用户信息” ;

反例： “donglh”:“用户信息”;

1. 【强制】数据结构选择：

根据数据的特性和操作需求选择适当的数据结构，如字符串（string）、哈希（hash）、列表（list）、集合（set）和有序集合（sorted set）等。

考虑数据的读写频率、排序需求和查询模式来选择合适的数据结构。

1. 【推荐】内存优化：

使用合适的数据结构和数据压缩技术来减少内存占用。

避免存储大的字符串或数据块，以防止内存碎片化和性能下降。

考虑使用分片或集群来分散数据存储和提高可用性。

1. 【推荐】缓存策略：

根据数据的访问模式和更新频率来制定合适的缓存策略。

设置适当的过期时间（TTL）来控制缓存的有效期。

考虑使用缓存穿透、缓存击穿和缓存雪崩等问题的预防措施。

1. 【强制】事务处理：

使用Redis的事务功能来保证一系列操作的原子性。

谨慎使用事务，避免长时间阻塞Redis的主线程。

1. 【强制】安全性：

配置适当的访问控制，限制对Redis的访问权限。

避免将敏感信息存储在Redis中，或者对敏感信息进行加密。

1. 【推荐】监控和性能优化：

定期监视Redis的性能指标，如内存使用、命令执行时间和连接数等。

根据监控结果来优化Redis的配置和数据结构设计。