****一、命名规约****

1.【强制】所有编程相关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$

2.【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 / DTO / VO / DAO等。  
正例：MarcoPolo / UserDO / XmlService / TcpUdpDeal / TaPromotion 反例：macroPolo / UserDo /XMLService / TCPUDPDeal / TAPromotion

3.【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

4.【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

5.【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。

6.【强制】POJO类中的任何布尔类型的变量，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。

7.【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.alibaba.mpp.util、类名为MessageUtils（此规则参考spring 的框架结构）

8.【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 classResourceObserver;

9.【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。  
正例：接口方法签名：void f();  
接口基础常量表示：String COMPANY = "alibaba";  
反例：接口方法定义：public abstract void f();  
说明：JDK8中接口允许有默认实现，那么这个default方法，是对所有实现类都有价值的默认实现。

10.接口和实现类的命名有两套规则：  
1） 【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，内部的实现类用Impl的后缀与接口区别。  
正例：CacheServiceImpl实现CacheService接口。  
2） 【推荐】 如果是形容能力的接口名称，取对应的形容词做接口名（通常是–able的形式）。  
正例：AbstractTranslator实现 Translatable。

11. 【参考】枚举类名建议带上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：DealStatusEnum；成员名称：SUCCESS / UNKOWN\_REASON。

12.【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。

13.【强制】long或者Long初始赋值时，必须使用大写的L，不能是小写的l，小写容易跟数字1 混淆，造成误解。

二、格式规约（重要）

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：  
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；表示终止右大括号后必须换行。  
2. 【强制】 左括号和后一个字符之间不出现空格；同样，右括号和前一个字符之间也不出现空格。  
3. 【强制】if/for/while/switch/do等保留字与左右括号之间都必须加空格。  
4. 【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。  
5. 【强制】代码块缩进4个空格，如果使用tab缩进，请设置成1个tab为4个空格。

6.【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） 换行时相对上一行缩进4个空格。

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");  
  
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也超过120个字符，逗号后才是换行处 method(args1,args2, args3, ...  
, argsX);  
7. 【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");  
8. 【推荐】没有必要增加若干空格来使某一行的字符与上一行的相应字符对齐。

****三、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。  
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】相同参数类型，相同业务含义，才可以使用Java的可变参数，避免使用Object。  
说明：可变参数必须放置在参数列表的最后。（提倡同学们尽量不用可变参数编程）  
正例：public User getUsers(Stringtype, Integer... ids);

4.【强制】所有的相同类型的包装类对象之间值的比较，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。

5.【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。

6.【强制】定义DO/DTO/VO等POJO类时，不要设定任何属性默认值。  
反例：某业务的DO的gmtCreate默认值为newDate();但是这个属性在数据提取时并没有置入具体值，在更新其它字段时又附带更新了此字段，导致创建时间被修改成当前时间。

7.【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。

8.【推荐】使用索引访问用String的split方法得到的数组时，需做最后一个分隔符后有无内容的检查，否则会有抛IndexOutOfBoundsException的风险。

9.【推荐】类成员与方法访问控制从严：

1） 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。  
2） 工具类不允许有public或default构造方法。  
3） 类非static成员变量并且与子类共享，必须是protected。  
4） 类非static成员变量并且仅在本类使用，必须是private。  
5） 类static成员变量如果仅在本类使用，必须是private。  
6） 若是static成员变量，必须考虑是否为final。  
7） 类成员方法只供类内部调用，必须是private。  
8） 类成员方法只对继承类公开，那么限制为protected。

****四、集合处理****

1. 【强制】Map/Set的key为自定义对象时，必须重写hashCode和equals。  
正例：String重写了hashCode和equals方法，所以我们可以非常愉快地使用String对象作为key来使用。

2.【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。

3.【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。

4.【推荐】高度注意Map类集合K/V能不能存储null值的情况

****五、并发处理****

1.【强制】SimpleDateFormat 是线程不安全的类，一般不要定义为static变量，如果定义为 static，必须加锁，或者使用DateUtils工具类。

2.【强制】并发修改同一记录时，避免更新丢失，要么在应用层加锁，要么在缓存加锁，要么在数据库层使用乐观锁，使用version作为更新依据。说明：如果每次访问冲突概率小于20%，推荐使用乐观锁，否则使用悲观锁。乐观锁的重试次数不得小于3次。

3.【强制】线程池不允许使用Executors去创建，而是通过ThreadPoolExecutor的方式，这样的处理方式让写的同学更加明确线程池的运行规则，规避资源耗尽的风险。  
说明：Executors各个方法的弊端：  
1） newFixedThreadPool和newSingleThreadExecutor: 主要问题是堆积的请求处理队列可能会耗费非常大的内存，甚至OOM。  
2） newCachedThreadPool和newScheduledThreadPool: 主要问题是线程数最大数是Integer.MAX\_VALUE，可能会创建数量非常多的线程，甚至OOM。

4.【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{ publicTimerTaskThread(){ super.setName("TimerTaskThread"); …  
}  
5.【推荐】使用CountDownLatch进行异步转同步操作，每个线程退出前必须调用countDown方法，线程执行代码注意catch异常，确保countDown方法可以执行，避免主线程无法执行至 countDown方法，直到超时才返回结果。说明：注意，子线程抛出异常堆栈，不能在主线程try-catch到。

6.【参考】ThreadLocal无法解决共享对象的更新问题，ThreadLocal对象建议使用static修饰。这个变量是针对一个线程内所有操作共有的，所以设置为静态变量，所有此类实例共享此静态变量 ，也就是说在类第一次被使用时装载，只分配一块存储空间，所有此类的对象(只要是这个线程内定义的)都可以操控这个变量。

7.【参考】volatile解决多线程内存不可见问题。对于一写多读，是可以解决变量同步问题，但是如果多写，同样无法解决线程安全问题。如果想取回count++数据，使用如下类实现：  
AtomicIntegercount = new AtomicInteger(); count.addAndGet(1); count++操作如果是  
JDK8，推荐使用LongAdder对象，比AtomicLong性能更好（减少乐观锁的重试次数）。

****六、控制语句****

1.【强制】在一个switch块内，每个case要么通过break/return来终止，要么注释说明程序将继续执行到哪一个case为止；在一个switch块内，都必须包含一个default语句并且放在最后，即使它什么代码也没有。

2.强制】在if/else/for/while/do语句中必须使用大括号，即使只有一行代码，避免使用下面的形式：if (condition) statements;

3.【推荐】循环体中的语句要考量性能，以下操作尽量移至循环体外处理，如定义对象、变量、  
获取数据库连接，进行不必要的try-catch操作（这个try-catch是否可以移至循环体外）。

****七、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

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。

3.【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。

****八、异常处理****

1.【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){…}

2.【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。

3.【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。

****九、日志规约****

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。  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
private static final Logger logger =LoggerFactory.getLogger(Abc.class);

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】应用中的扩展日志（如打点、临时监控、访问日志等）命名方式：  
appName\_logType\_logName.log。logType:日志类型，推荐分类有stats/desc/monitor/visit 等；logName:日志描述。这种命名的好处：通过文件名就可知道日志文件属于什么应用，什么类型，什么目的，也有利于归类查找。  
正例：mppserver应用中单独监控时区转换异常，如： mppserver\_monitor\_timeZoneConvert.log  
说明：推荐对日志进行分类，错误日志和业务日志尽量分开存放，便于开发人员查看，也便于通过日志对系统进行及时监控。

4. 【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。  
说明：logger.debug("Processingtrade with id: " + id + " symbol: " + symbol); 如果日志级别是warn，上述日志不会打印，但是会执行字符串拼接操作，如果symbol是对象，会执行toString()方法，浪费了系统资源，执行了上述操作，最终日志却没有打印。  
正例：（条件）  
if (logger.isDebugEnabled()) {  
logger.debug("Processing trade with id: " +id + " symbol: " + symbol);  
}  
正例：（占位符）  
logger.debug("Processing trade with id: {} andsymbol : {} ", id, symbol);

5. 【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。  
正例：<loggername="com.taobao.ecrm.member.config" additivity="false">

6. 【强制】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。如果不处理，那么往上抛。  
正例：logger.error(各类参数或者对象toString +"\_" + e.getMessage(), e);

7. 输出的POJO类必须重写toString方法，否则只输出此对象的hashCode值（地址值），没啥参考意义。

****十、MYSQL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，数据类型是unsigned tinyint  
（ 1表示是，0表示否），此规则同样适用于odps建表。  
说明：任何字段如果为非负数，必须是unsigned。  
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。数据库字段名的修改代价很大，因为无法进行预发布，所以字段名称需要慎重考虑。  
正例：getter\_admin，task\_config，level3\_name 反例：GetterAdmin，taskConfig，level\_3\_name  
3. 【强制】表名不使用复数名词。  
说明：表名应该仅仅表示表里面的实体内容，不应该表示实体数量，对应于DO类名也是单数形式，符合表达习惯。  
4. 【强制】禁用保留字，如desc、range、match、delayed等，参考官方保留字。  
5. 【强制】唯一索引名为uk\_字段名；普通索引名则为idx\_字段名。  
说明：uk\_即 unique key；idx\_ 即index的简称。  
6. 【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。  
7. 【强制】如果存储的字符串长度几乎相等，使用CHAR定长字符串类型。  
8. 【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。  
9. 【强制】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；分表时改为从  
TDDL Sequence取值，确保分表之间的全局唯一。gmt\_create,gmt\_modified的类型均为 date\_time类型。  
10.【推荐】表的命名最好是加上“业务名称\_表的作用”，避免上云梯后，再与其它业务表关联时有混淆。  
正例：tiger\_task/ tiger\_reader / mpp\_config

十一、MYSQL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。  
说明：不要以为唯一索引影响了insert速度，这个速度损耗可以忽略，但提高查找速度是明显的；另外，即使在应用层做了非常完善的校验和控制，只要没有唯一索引，根据墨菲定律，必然有脏数据产生。  
2. 【强制】超过三个表禁止join。需要join的字段，数据类型保持绝对一致；多表关联查询时，保证被关联的字段需要有索引。  
说明：即使双表join也要注意表索引、SQL性能。  
3. 【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。  
说明：索引的长度与区分度是一对矛盾体，一般对字符串类型数据，长度为20的索引，区分度会高达90%以上，可以使用count(distinct left(列名, 索引长度))/count(\*)的区分度来确定。  
4. 【强制】页面搜索严禁左模糊或者全模糊，如果需要请走搜索引擎来解决。  
说明：索引文件具有B-Tree的最左前缀匹配特性，如果左边的值未确定，那么无法使用此索引。  
5. 【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。  
正例：where a=? and b=? order by c; 索引：a\_b\_c  
反例：索引中有范围查找，那么索引有序性无法利用，如：WHERE a>10 ORDER BY b; 索引a\_b 无法排序。