**编程规范**

一、编程规约

(一)命名风格

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$

2. 【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。

正例： alibaba / taobao / youku / hangzhou 等国际通用的名称，可视同英文。

反例： DaZhePromotion [打折] / getPingfenByName() [评分] / int 某变量 = 3

3. 【强制】类名使用 UpperCamelCase 风格，必须遵从驼峰形式，但以下情形例外：DO / BO / DTO / VO / AO

正例：MarcoPolo / UserDO / XmlService / TcpUdpDeal / TaPromotion

反例：macroPolo / UserDo / 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，必须遵从驼峰形式。

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

反例： MAX\_COUNT

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】中括号是数组类型的一部分，数组定义如下：String[] args;

反例：使用 String args[]的方式来定义。

8. 【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.open.util、类名为 MessageUtils（此规则参考

spring 的框架结构）

9. 【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。

10. 【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。

11. 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。

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。

(二)常量定义

1. 【强制】不允许任何魔法值（即未经定义的常量）直接出现在代码中。

反例： String key = "Id#taobao\_" + tradeId;

cache.put(key, value);

2. 【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。

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

(三)代码格式

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；表示终止的右大括号后必须换行。

2. 【强制】 左小括号和字符之间不出现空格；同样，右小括号和字符之间也不出现空格。

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：运算符包括赋值运算符=、逻辑运算符&&、加减乘除符号等。

5. 【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。

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） 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，在逗号后换行。

5） 在括号前不要换行，见反例。

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对应位置的字符对齐。

正例：

int a = 3;

long b = 4L;

float c = 5F;

StringBuffer sb = new StringBuffer();

说明：增加 sb 这个变量，如果需要对齐，则给 a、b、c 都要增加几个空格，在变量比较多的情况下，是一种累赘的事情。

9. 【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：没有必要插入多个空行进行隔开。

(四)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。

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(String type, Integer... ids) {...}

4. 【强制】不能使用过时的类或方法。

5. 【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）

6. 【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 范围内的赋值，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。

7. 【强制】序列化类新增属性时，请不要修改 serialVersionUID 字段，避免反序列失败；如果完全不兼容升级，避免反序列化混乱，那么请修改 serialVersionUID 值。

说明：注意 serialVersionUID 不一致会抛出序列化运行时异常。

8. 【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。

9. 【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。

说明：

String str = "a,b,c,,";

String[] ary = str.split(",");

//预期大于 3，结果是 3

System.out.println(ary.length);

10. 【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。

11. 【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。

12. 【推荐】setter 方法中，参数名称与类成员变量名称一致，this.成员名 = 参数名。在

getter/setter 方法中，不要增加业务逻辑，增加排查问题的难度。

13. 【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：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。

反例：

String str = "start";

for (int i = 0; i < 100; i++) {

str = str + "hello";

}

14. 【推荐】final 可以声明类、成员变量、方法、以及本地变量，下列情况使用 final 关键字：

1） 不允许被继承的类，如：String 类。

2） 不允许修改引用的域对象，如：POJO 类的域变量。

3） 不允许运行过程中重新赋值的局部变量。

4） 避免上下文重复使用一个变量，使用 final 描述可以强制重新定义一个变量，方便更好

地进行重构。

15. 【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。

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

思考：如果是一个 private 的方法，想删除就删除，可是一个 public 的 service 方法，或者一个 public 的成员变量，删除一下，不得手心冒点汗吗？变量像自己的小孩，尽量在自己的视线内，变量作用域太大，如果无限制的到处跑，那么你会担心的。

(五)集合处理

1. 【强制】关于 hashCode 和 equals 的处理，遵循如下规则：

1） 只要重写 equals，就必须重写 hashCode。

2） 因为 Set 存储的是不重复的对象，依据 hashCode 和 equals 进行判断，所以 Set 存储的对象必须重写这两个方法。

3） 如果自定义对象做为 Map 的键，那么必须重写 hashCode 和 equals。

说明：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. 【强制】ArrayList的subList结果不可强转成ArrayList，否则会抛出ClassCastException

异常：java.util.RandomAccessSubList cannot be cast to java.util.ArrayList ;

说明：subList 返回的是 ArrayList 的内部类 SubList，并不是 ArrayList ，而是

ArrayList 的一个视图，对于 SubList 子列表的所有操作最终会反映到原列表上。

3. 【强制】 在 subList 场景中，高度注意对原集合元素个数的修改，会导致子列表的遍历、增加、删除均产生 ConcurrentModificationException 异常。

4. 【强制】使用集合转数组的方法，必须使用集合的 toArray(T[] array)，传入的是类型完全

一样的数组，大小就是 list.size()。

说明：使用 toArray 带参方法，入参分配的数组空间不够大时，toArray 方法内部将重新分配内存空间，并返回新数组地址；如果数组元素大于实际所需，下标为[ list.size() ]的数组

元素将被置为 null，其它数组元素保持原值，因此最好将方法入参数组大小定义与集合元素

个数一致。

正例：

List<String> list = new ArrayList<String>(2);

list.add("guan");

list.add("bao");

String[] array = new String[list.size()];

array = list.toArray(array);

反例：直接使用 toArray 无参方法存在问题，此方法返回值只能是 Object[]类，若强转其它

类型数组将出现 ClassCastException 错误。

5. 【强制】使用工具类 Arrays.asList()把数组转换成集合时，不能使用其修改集合相关的方

法，它的 add/remove/clear 方法会抛出 UnsupportedOperationException 异常。

说明：asList 的返回对象是一个 Arrays 内部类，并没有实现集合的修改方法。Arrays.asList

体现的是适配器模式，只是转换接口，后台的数据仍是数组。

String[] str = new String[] { "a", "b" };

List list = Arrays.asList(str);

第一种情况：list.add("c"); 运行时异常。

第二种情况：str[0] = "gujin"; 那么 list.get(0)也会随之修改。

6. 【强制】泛型通配符<? extends T>来接收返回的数据，此写法的泛型集合不能使用 add 方

法，而<? super T>不能使用 get 方法，做为接口调用赋值时易出错。

说明：扩展说一下 PECS(Producer Extends Consumer Super)原则：1）频繁往外读取内容

的，适合用上界 Extends。2）经常往里插入的，适合用下界 Super。

7. 【强制】不要在 foreach 循环里进行元素的 remove/add 操作。remove 元素请使用 Iterator方式，如果并发操作，需要对 Iterator 对象加锁。

正例：

Iterator<String> it = a.iterator();

while (it.hasNext()) {

String temp = it.next();

if (删除元素的条件) {

it.remove();

}

}

反例：

List<String> a = new ArrayList<String>();

a.add("1");

a.add("2");

for (String temp : a) {

if ("1".equals(temp)) {

a.remove(temp);

}

}

说明：以上代码的执行结果肯定会出乎大家的意料，那么试一下把“1”换成“2”，会是同样的结果吗？

8. 【推荐】集合初始化时，指定集合初始值大小。

9. 【推荐】高度注意 Map 类集合 K/V 能不能存储 null 值的情况，如下表格：

10. 【参考】利用 Set 元素唯一的特性，可以快速对一个集合进行去重操作，避免使用 List 的contains 方法进行遍历、对比、去重操作。

(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，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。

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。

8. 【参考】合理处理注释掉的代码。在上方详细说明，而不是简单的注释掉。如果无用，则删除。

说明：代码被注释掉有两种可能性：1）后续会恢复此段代码逻辑。2）永久不用。前者如果没有备注信息，难以知晓注释动机。后者建议直接删掉（代码仓库保存了历史代码）。

9. 【参考】对于注释的要求：第一、能够准确反应设计思想和代码逻辑；第二、能够描述业务含义，使别的程序员能够迅速了解到代码背后的信息。完全没有注释的大段代码对于阅读者形同天书，注释是给自己看的，即使隔很长时间，也能清晰理解当时的思路；注释也是给继任者看的，使其能够快速接替自己的工作。

10. 【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。

11. 【参考】特殊注释标记，请注明标记人与标记时间。注意及时处理这些标记，通过标记扫描，经常清理此类标记。线上故障有时候就是来源于这些标记处的代码。

(八)其它

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，使用 System.nanoTime()的方式。在 JDK8 中，

针对统计时间等场景，推荐使用 Instant 类。

2. 【推荐】不要在视图模板中加入任何复杂的逻辑。

说明：根据 MVC 理论，视图的职责是展示，不要抢模型和控制器的活。

3. 【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。

8. 【推荐】对于“明确停止使用的代码和配置”，如方法、变量、类、配置文件、动态配置属性等要坚决从程序中清理出去，避免造成过多垃圾。

二、异常日志

(一)异常处理

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。

说明：无法通过预检查的异常除外，如在解析一个外部传来的字符串形式数字时，通过 catch

NumberFormatException 来实现。

正例：if (obj != null) {...}

反例：try { obj.method() } catch (NullPointerException e) {...}

2. 【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。

3. 【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。

4. 【强制】捕获异常是为了处理它，不要捕获了却什么都不处理而抛弃之，如果不想处理它，请将该异常抛给它的调用者。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。

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。

6. 【强制】finally 块必须对资源对象、流对象进行关闭，有异常也要做 try-catch。

说明：如果 JDK7 及以上，可以使用 try-with-resources 方式。

7. 【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。

8. 【强制】捕获异常与抛异常，必须是完全匹配，或者捕获异常是抛异常的父类。

说明：如果预期对方抛的是绣球，实际接到的是铅球，就会产生意外情况。

9. 【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。

说明：本手册明确防止 NPE 是调用者的责任。即使被调用方法返回空集合或者空对象，对调用者来说，也并非高枕无忧，必须考虑到远程调用失败、序列化失败、运行时异常等场景返回null 的情况。

10. 【推荐】防止 NPE，是程序员的基本修养，注意 NPE 产生的场景：

1）返回类型为基本数据类型，return 包装数据类型的对象时，自动拆箱有可能产生 NPE。

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

2） 数据库的查询结果可能为 null。

3） 集合里的元素即使 isNotEmpty，取出的数据元素也可能为 null。

4） 远程调用返回对象时，一律要求进行空指针判断，防止 NPE。

5） 对于 Session 中获取的数据，建议 NPE 检查，避免空指针。

6） 级联调用 obj.getA().getB().getC()；一连串调用，易产生 NPE。

正例：使用 JDK8 的 Optional 类来防止 NPE 问题。

11. 【推荐】定义时区分 unchecked / checked 异常，避免直接抛出 new RuntimeException()，

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。

12. 【参考】在代码中使用“抛异常”还是“返回错误码”，对于公司外的 http/api 开放接口必须使用“错误码”；而应用内部推荐异常抛出；跨应用间 RPC 调用优先考虑使用 Result 方式，封装 isSuccess()方法、“错误码”、“错误简短信息”。

说明：关于 RPC 方法返回方式使用 Result 方式的理由：

1）使用抛异常返回方式，调用方如果没有捕获到就会产生运行时错误。

2）如果不加栈信息，只是 new 自定义异常，加入自己的理解的 error message，对于调用

端解决问题的帮助不会太多。如果加了栈信息，在频繁调用出错的情况下，数据序列化和传输的性能损耗也是问题。

13. 【参考】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。

正例：一个类中有多个 public 方法，都需要进行数行相同的参数校验操作，这个时候请抽取：

private boolean checkParam(DTO dto) {...}

(二)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);

logger.debug("Processing trade with id: {} symbol : {} ", id, symbol);

三、MySQL 数据库

(一)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）。

说明：任何字段如果为非负数，必须是 unsigned。

正例：表达逻辑删除的字段名 is\_deleted，1 表示删除，0 表示未删除。

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，请参考 MySQL 官方保留字。

5. 【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。

说明：pk\_ 即 primary key；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。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”。

正例：tiger\_task / tiger\_reader / mpp\_config

11. 【推荐】库名与应用名称尽量一致。

12. 【推荐】如果修改字段含义或对字段表示的状态追加时，需要及时更新字段注释。

13. 【推荐】字段允许适当冗余，以提高查询性能，但必须考虑数据一致。冗余字段应遵循：

1）不是频繁修改的字段。

2）不是 varchar 超长字段，更不能是 text 字段。

正例：商品类目名称使用频率高，字段长度短，名称基本一成不变，可在相关联的表中冗余存储类目名称，避免关联查询。

14. 【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：如下表，其中无符号值可以避免误存负数，且扩大了表示范围。

对象 年龄区间 类型 表示范围

人 150 岁之内 unsigned tinyint 无符号值：0 到 255

龟 数百岁 unsigned smallint 无符号值：0 到 65535

恐龙化石 数千万年 unsigned int 无符号值：0 到约 42.9 亿

太阳 约 50 亿年 unsigned bigint 无符号值：0 到约 10 的 19 次方

(二)索引规约

1. 【强制】业务上具有唯一特性的字段，即使是多个字段的组合，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，数据类型必须绝对一致；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即可。

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。

6. 【推荐】利用覆盖索引来进行查询操作，避免回表。

说明：如果一本书需要知道第 11 章是什么标题，会翻开第 11 章对应的那一页吗？目录浏览一下就好，这个目录就是起到覆盖索引的作用。

正例：能够建立索引的种类：主键索引、唯一索引、普通索引，而覆盖索引是一种查询的一种效果，用 explain 的结果，extra 列会出现：using index。

7. 【推荐】利用延迟关联或者子查询优化超多分页场景。

说明：MySQL 并不是跳过 offset 行，而是取 offset+N 行，然后返回放弃前 offset 行，返回N 行，那当 offset 特别大的时候，效率就非常的低下，要么控制返回的总页数，要么对超过特定阈值的页数进行 SQL 改写。

正例：先快速定位需要获取的 id 段，然后再关联：

SELECT a.\* FROM 表 1 a, (select id from 表 1 where 条件 LIMIT 100000,20 ) b where a.id=b.id

8. 【推荐】建组合索引的时候，区分度最高的在最左边。

正例：如果 where a=? and b=? ，a 列的几乎接近于唯一值，那么只需要单建 idx\_a 索引即

可。

说明：存在非等号和等号混合判断条件时，在建索引时，请把等号条件的列前置。如：where a>? and b=? 那么即使 a 的区分度更高，也必须把 b 放在索引的最前列。

10. 【推荐】防止因字段类型不同造成的隐式转换，导致索引失效。

11. 【参考】创建索引时避免有如下极端误解：

1）宁滥勿缺。误认为一个查询就需要建一个索引。

2）宁缺勿滥。误认为索引会消耗空间、严重拖慢更新和新增速度。

3）抵制惟一索引。误认为业务的惟一性一律需要在应用层通过“先查后插”方式解决。

(三)SQL 语句

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。

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 之外的不重复行数，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】不得使用外键与级联，一切外键概念必须在应用层解决。

说明：（概念解释）学生表中的 student\_id 是主键，那么成绩表中的 student\_id 则为外键。

如果更新学生表中的 student\_id，同时触发成绩表中的 student\_id 更新，则为级联更新。

外键与级联更新适用于单机低并发，不适合分布式、高并发集群；级联更新是强阻塞，存在数据库更新风暴的风险；外键影响数据库的插入速度。

7. 【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。

8. 【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。

(四)ORM 映射

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。

2. 【强制】sql.xml 配置参数使用：#{}，#param# 不要使用${} 此种方式容易出现 SQL 注入。

3. 【强制】iBATIS 自带的 queryForList(String statementName,int start,int size)不推荐使用。

说明：其实现方式是在数据库取到statementName对应的SQL语句的所有记录，再通过subList

取 start,size 的子集合。

正例：Map<String, Object> map = new HashMap<String, Object>();

map.put("start", start);

map.put("size", size);

4. 【强制】更新数据表记录时，必须同时更新记录对应的 gmt\_modified 字段值为当前时间。

5. 【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。