链滴

阿里编码规范

作者：xixiaoming
原文链接：https://ld246.com/article/1542680722020
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

<h2 id="命名规范">命名规范</h2>

【强制】类名使用 UpperCamelCase .不能以下划线或美元符号打头,必须遵从驼峰形式,但以下形例外:(领域模型的相关命名) DO / DTO / VO / DAO 等.

正例: MarcoPolo / UserDO / HtmlDTO / XmlService / TcpUdpDeal / aPromotion
反例: macroPolo / UserDo / HTMLDto / XMLService / TCPUDPDeal / TAPromotion

【强制】抽象类命名使用 Abstract 或 Base 开头;异常类命名使用 Exception 结尾;测试类命名以要测试的类的名称开始,以 Test 结尾.
【强制】中括号是数组类型的一部分,数组定义如下: String[] args;

反例: 即使不会编译出错,请勿使用 String args[];定义

【强制】方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase .尽量是英文语法达,避免使用汉语拼音来命名,严禁使用拼音与英文混合的命名方式.

说明: 正确的英文单词拼写和语法可以让代码阅读者易于理解,避免歧义.
正例: localValue / getHttpMessage() / inputUserId
反例: LocalValue / getPingfenByName() [评分的拼音]

【强制】常量命名应该全部大写,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

【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.</li

反例: 定义为基本数据类型 boolean isSuccess;的属性,它的方法也是 isSuccess(),HSF 框架在反解析的时候,"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.

【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 的架结构)

【强制】杜绝完全不规范的缩写,集团认可的缩写请参考附 2.

反例: < 某业务代码 >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.

【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;

【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.<
p>

接口和实现类的命名有两套规则:

【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 mpl 的后缀与接口区别.

正例: CacheServiceImpl 实现 CacheService 接口.

【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.

【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

【参考】各层命名规约:

Service / DAO 层方法命名规约

获取单个对象的方法用 get 做前缀.

获取多个对象的方法用 list 做前缀.

获取统计值的方法用 count 做前缀.

插入的方法总是用 save (推荐)或 insert 做前缀.

删除的方法总是用 remove (推荐)或 delete 做前缀.

修改的方法总是用 update 做前缀.

领域模型命名规约

数据对象: xxxDO,xxx 即为数据表名.

数据传输对象: xxxDTO,xxx 为业务领域相关的名称.

展示对象: xxxVO,xxx 一般为网页名称.

POJO 是 DO / DTO / BO / VO 的统称,禁止命名成 xxxPOJO .

<h2 id="常量定义">常量定义</h2>

【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;

cache.put(key, value);

【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,成误解.

说明: Long a = 2l;写的是数字的 21,还是 Long 型的 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 Ctrl + F 才定位到修改的常量,不利于理解,也不利于维护.

【推荐】常量的复用层次有五层:跨应用共享常量、应用内共享常量、子工程内共享常量、包内共常量、类内共享常量.

跨应用共享常量:放置在二方库中,通常是 client.jar 中的 const 目录下.

应用内共享常量:放置在一方库的 modules 中的 const 目录下.

反例: 易懂变量也要统一定义成应用内共享常量,两位攻城师在两个类中分别定义了表示"是"的变量

类 A 中:public static final String YES = "yes";
类 B 中:public static final String YES = "y";

A.YES.equals(B.YES),预期是 true,但实际返回为 false,导致产生线上问题.

子工程内部共享常量:即在当前子工程的 const 目录下.

包内共享常量:即在当前包下单独的 const 目录下.

类内共享常量:直接在类内部 private static final 定义.

【推荐】如果变量值仅在一个范围内变化用 Enum 类.如果还带有名称之外的延伸属性,必须使用
num 类,下面正例中的数字就是延伸信息,表示星期几.

正例:
public Enum{ MONDAY(1), TUESDAY(2), WEDNESDAY(3), THURSDAY(4), FRIDAY(5), SATU
DAY(6), SUNDAY(7); }
<h2 id="OOP规约">OOP 规约</h2>

【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,
接用类名来访问即可.
【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
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.

【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: 比如打印方法: void print(String... str);不要写成:void print(Object... str);

【强制】对外暴露的接口签名,原则上不允许修改方法签名,避免对接口调用方产生影响.接口过时
须加 @Deprecated 注解,并清晰地说明采用的新接口或者新服务是什么.
【强制】不能使用过时的类或方法.

说明: java.net.URLDecoder 中的方法 decode(String encodeStr)这个方法已经过时,应该使用
参数 decode(String source, String encode).接口提供方既然明确是过时接口,那么有义务同时提供
的接口;作为调用方来说,有义务去考证过时方法的新实现是什么.

【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 .
/li>

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)

【强制】所有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,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,会在堆上产生,并不会复用已有对象,这是一个大坑,推荐作用 equals 方法进行判断.

【强制】关于基本数据类型与包装数据类型的使用标准如下:

所有的 POJO 类属性使用包装数据类型.

RPC 方法的返回值和参数必须使用包装数据类型.

所有的局部变量使用基本数据类型.

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: 供销平台交易总额涨跌 HSF 服务,返回值为基本数据类型,调用不成功时,返回的是默认值,页显示:0%,这是不合理的.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.

【强制】定义 DO/DTO/VO 等 POJO 类时,不要加任何属性默认值.

反例: 某业务的 DO 的 gmtCreate 默认值为 new Date();但是这个属性在数据提取时并没有置入体值,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.

【强制】序列化类修改时,请不要修改 serialVersionUID 字段,避免反序列失败;如果完全不兼容升,避免反序列化混乱,那么请修改 serialVersionUID 值.

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

【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.
【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.
【强制】POJO 类必须写 toString 方法.使用工具类 source> generate toString 时,如果继了另一个 POJO 类,注意在前面加一下 super.toString.

说明: 在方法执行抛出异常时,可以直接调用 POJO 的 toString()方法打印其属性值,便于排查问题.

【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,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;因为方法息价值较低,所有 Service 和 DAO 的 getter/setter 方法放在类体最后.

【推荐】本类调用加 this,父类调用加 super,不管是变量还是方法.这样有利于阅读者清晰知道这方法是本类定义,还是从父类继承.
【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 get/set 方法中,量不要增加业务逻辑,增加排查问题难度.

反例:
public Integer getData(){ if(true) {return data + 100;} else {return data - 100;} }

【推荐】字符串的联接方式,推荐使用 StringBuffer 和 StringBuilder 的方法进行扩展:

正例:
StringBuffer s = new StringBuffer(); s.append("guan").append("bao");
 如果确定没有线程安全问题,或者是方法内局部变量,推荐使用 StringBuilder,在能够提前预估目字符串的长度时,可以指定 StringBuilder 初始容量值,避免自动扩容带来的数据拷贝开销与频繁内存分产生的内存碎片.

【推荐】final 可提高程序响应效率,声明成 final 的情况:

不需要重新赋值的变量,包括类属性、局部变量.

对象参数前加 final,表示不允许修改引用的指向.

类方法确定不会被重写.

说明: final 修饰变量表示此变量是不可变的,在多线程读写场景中是线程安全的.

【推荐】慎用 Object 的 clone 方法来拷贝对象.

说明: 对象的 clone 方法默认是浅拷贝,若想实现深拷贝需要重写 clone 方法实现属性对象的拷贝.
/p>

【推荐】类成员与方法访问控制从严:

如果不允许外部直接通过 new 来生成对象,那么构造方法必须 private.

工具类不允许有 public 或 default 构造方法.

类的非 static 成员变量,必须 private.

类的 static 成员变量如果只是类内部使用,也必须是 private.

若是 static 成员变量,必须考虑是否为 final.

类成员方法只供类内部调用,必须 private.

如果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.思考:如果是个 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,么你会担心的.

【推荐】方法的多个参数顺序为:输入参数在前,输出参数在后;代码里先出现的参数在前,后出现的数在后.

反例:

public Object f(Object obj) {... ; return obj;} //obj 既是输入参数,又是返回对象,尽量避免.
public String f(Long id, String str){ return str + id;} //str 和 id 的使用先后尽量与参数顺序保一致.

<h2 id="集合处理">集合处理</h2>

【强制】Map/Set 的 key 为自定义对象时,必须重写 hashCode 和 equals.

说明: 为什么要重写 hashCode()
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.

【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.

【强制】在 subList 场景中,高度注意对原列表的修改,会导致子列表的遍历、增加、删除均产生 ConcurrentModificationException 异常.

说明: 抽查表明,九成的开发同学对此知识点都有错误的认知.

【强制】使用集合转数组的方法,必须使用集合的 toArray(T[] a),传入的是类型完全一样的数组,大就是 list.size().

说明: 直接使用 toArray 无参方法存在问题,此方法返回值只能是 Object[]类,若强转其它类型数将出现 ClassCastException 错误.使用 toArray 带参方法,入参分配的数组空间不够大时,toArray 方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,多余的数组元素将被置为 null因此最好将方法入参数组大小定义与集合元素个数一致.

【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.

【强制】泛型通配符来接收返回的数据,此写法的泛型集合不能使用 add 方法.

说明: 苹果装箱后返回一个对象,此对象就不能往里加任何水果,包括苹果.

【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.

反例:
List a = new ArrayList(); a.add("1"); a.add("2"); for (String temp : a) { if ("1".equals(tem)) { a.remove(temp); } }
说明: 这个例子的执行结果会出乎大家的意料,那么试一下把"1"换成"2",会是同样的结果吗?
参考: ConcurrentModificationException 异常析
反例:
Iterator it = a.iterator(); while(it.hasNext(){ String temp = it.next(); if(删除元素的条件){ it.remove(); } }

【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.

【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 合对象;entrySet()返回的是 K-V 值组合集合.

【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:

集合类 | Key| Value| Super| 说明
---|
Hashtable| 不允许为 null| 不允许为 null| Dictionary| 线程安全
ConcurrentHashMap| 不允许为 null| 不允许为 null| AbstractMap| 线程局部安全
TreeMap| 不允许为 null| 允许为 null| AbstractMap| 线程不安全
HashMap| 允许为 null| 允许为 null| AbstractMap| 线程不安全

反例: 前期抽样表明近八成的同学认为 ConcurrentHashMap 是可以置入 null 值.在"美杜莎"的批量翻译场景中,子线程分发时,出现置入 null 值的情况,但主线程没有捕获到此异常,导致排查困难.
【参考】合理利用好集合的有序性(sort)和稳定性(order),避免集合的无序性(unsort)和不稳定性(uorder)带来的负面影响.

说明: 稳定性指集合每次遍历的元素次序是一定的.有序性是指遍历的结果是按某种比较规则依次

列的.如:ArrayList 是 order/unsort;HashMap 是 unorder/unsort;TreeSet 是 order/sort.

【参考】利用 Set 元素唯一的特性,可以快速对另一个集合进行去重操作,避免使用 List 的 contain 方法进行遍历去重操作.

<h2 id="并发处理">并发处理</h2>

【强制】静态 Utils 或单例必须是线程安全的.
【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./p>

【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,须加锁,或者使用 DateUtils 工具类.

正例: 注意线程安全,使用 DateUtils.亦推荐如下处理:
private static final ThreadLocal df = new ThreadLocal() { @Override protected DateFor at initialValue() { return new SimpleDateFormat("yyyy-MM-dd"); } };

【强制】高并发时,同步调用应该去考量锁的性能损耗.能用无锁数据结构,就不要用锁;能锁区块,就要锁整个方法体;能用对象锁,就不要用类锁.
【强制】对多个资源、数据库表、对象同时加锁时,需要保持一致的加锁顺序,否则可能会造成死锁

说明: 线程一需要对表 A、B、C 依次全部加锁后才可以进行更新操作,那么线程二的加锁顺序也须是 A、B、C,否则可能出现死锁.

【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于 3 次.
正例: 集团很多业务使用 TairManager 方法:incr(namespace, lockKey, 1, 0, expireTime); 判断回步长是否为 1,实现分布式锁.

【强制】多线程并行处理定时任务时,Timer 运行多个 TimeTask 时,只要其中之一没有捕获抛出异常,其它任务便会自动终止运行,使用 ScheduledExecutorService 则没有这个问题.

反例: 阿里云平台产品技术部,域名更新具体产品信息保存到 tair,Timer 产生了 RunTimeExcetion 异常后,定时任务不再执行,通过检查日志发现原因,改为 ScheduledExecutorService 方式.

【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:

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.

newCachedThreadPool 和 newScheduledThreadPool:

主要问题是线程数最大数是 Integer.MAX_VALUE,可能会造成创建数量非常多的线程,甚至 OOM

【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:

正例:
public class TimerTaskThread extends Thread { public TimerTaskThread(){ super.setN me("TimerTaskThread"); ... } ... }

【推荐】使用 CountDownLatch 进行同步转异步操作,每个线程退出前必须调用 countDown 方 ,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 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.
反例: 在"马可波罗平台"的翻译同步转异步多线程时,由于翻译过程抛出异常,导致 countDown 方 失败,经常超时才返回.

【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 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:AtomicInteger count = n w AtomicInteger(); count.addAndGet(1); 参考内存模型:深入理解 java 内存模型

说明: count++ 操作如果是 JDK8,推荐使用 LongAdder 对象,比 AtomicLong 性能更好(减少乐 锁的重试次数).

【参考】注意 HashMap 的扩容死链,导致 CPU 飙升的问题.

反例: "采购直达"业务 P2 故障,原因是依赖的开源软件中对 HashMap 有并发写,导致 get 时死循环 耗尽了 CPU 资源,参考开源框架误用 HashMap 导致的线上问题分析

【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 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 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 变量.

<h2 id="控制语句">控制语句</h2>

【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 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 码也没有.
【强制】在 if/for/while/switch 语句中尽量使用大括号,即使只有一行代码,避免使用下面的形式:if condition) statements;
【推荐】推荐尽量少用 else,如下写法:

if(condition){ ... return obj; } // 接着写 else 的业务逻辑代码;
说明: 如果使用要 if-else if-else 方式表达逻辑,【强制】请勿超过 3 层.

【推荐】除常用的 get 方法外,不要在条件语句中执行复杂的语句,以提高可读性.

正例:

//伪代码如下 InputStream stream = file.open(fileName, "w"); if (stream != null) { ... }</
>
反例:
if (File.open(fileName, "w") != null)) { ... }

【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
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).
【推荐】接口入参保护,这种场景常见的是用于做批量操作的接口.

反例: 某年支付宝发生的一个严重故障,就是一个用户批量查询的接口,API 文档上有说最多查多少个
但接口实现上没做任何保护,导致调用方传了一个 1000 的用户 id 数组过来后,查询信息后,内存爆了.</
>

【参考】方法中需要进行参数校验的场景:

调用频次低的方法.

执行时间开销很大的方法,参数校验时间几乎可以忽略不计,但如果因为参数错误导致中间执行回退
或者错误,那得不偿失.

需要极高稳定性和可用性的方法.

对外提供的开放接口,不管是 HSF/API/HTTP 接口.

【参考】方法中不需要参数校验的场景:

极有可能被循环调用的方法,不建议对参数进行校验.但在方法说明里必须注明外部参数检查.

底层的方法调用频度都比较高,一般不校验.毕竟是像纯净水过滤的最后一道,参数错误不太可能到
层才会暴露问题.一般 DAO 层与 Service 层都在同一个应用中,部署在同一台服务器中,所以 DAO 的
数校验,可以省略.

被声明成 private 只会被自己代码所调用的方法,如果能够确定调用方法的代码传入参数已经做过
查或者肯定不会有问题,此时可以不校验参数.

<h2 id="注释规约">注释规约</h2>

【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 I
E 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.

【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

明外,还必须指出该方法做什么事情,实现什么功能.

说明: 如有实现和调用注意事项,请一并说明.

【强制】所有的类都必须添加创建者信息.

说明: 未来集团会统一 IDE 开发模板.

【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.
【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.
【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、关键字,保持英文原即可.

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.

【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.

【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,就失了导航的意义.

【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,语义清晰的码不需要额外的注释.

【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,经常清理类标记.线上故障有时候就是来源于这些标记处的代码.

待办事宜(TODO):([标记人,标记时间,[预计处理时间])

表示需要实现,但目前还未实现的功能.这实际上是一个 javadoc 的标签,目前的 javadoc 还没有实,但已经被广泛使用.只能应用于类,接口和方法(因为它是一个 javadoc 标签).

错误,不能工作(FIXME):([标记人,标记时间,[预计处理时间])

在注释中用 FIXME 标记某代码是错误的,而且不能工作,需要及时纠正的情况.
<h2 id="其他">其他</h2>

【强制】在使用正则表达式时,利用好其预编译功能,可以有效加快正则匹配速度.

说明: 不要在方法体内定义:Pattern pattern = Pattern.compile(规则);

【强制】避免用 Apache Beanutils 进行属性的 copy.

说明: Apache BeanUtils 性能较差,可以使用其他方案比如 Spring BeanUtils, Cglib BeanCopier

【强制】velocity 调用 POJO 类的属性时,建议直接使用属性名取值即可,模板引擎会自动按规范
用 POJO 的 getXxx(),如果是 boolean 基本数据类型变量(注意,boolean 命名不需要加 is 前缀),会自
调用 isXxx()方法.

说明: 注意如果是 Boolean 包装类对象,优先调用 getXxx()的方法.

【强制】尽量不要在 vm 中加入变量声明、逻辑运算符,更不要在 vm 模板中加入任何复杂的逻辑.
/li>
【强制】后台输送给页面的变量必须加 $!{var}——中间的感叹号.

说明: 如果 var=null 或者不存在,那么 ${var}会直接显示在页面上.

【强制】注意 Math.random() 这个方法返回是 double 类型,注意取值范围 0≤x<1(能够取到
值,注意除零异常),如果想获取整数类型的随机数,不要将 x 放大 10 的若干倍然后取整,直接使用 Rand
m 对象的 nextInt 或者 nextLong 方法.
【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.

【推荐】任何数据结构的使用都应限制大小.

说明: 这点很难完全做到,但很多次的故障都是因为数据结构自增长,结果造成内存被吃光.

【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
从程序中清理出去,避免造成过多垃圾.清理这类垃圾代码是技术气场,不要有这样的观念:"不做不错,多
多错".