IT 产品线 Java语言编程军规

目录 Table of Contents

IT 产品线 Java语言编程军规 1

[第一章 前言 1](#_TOC_250006)

[第二章 军规应用范围和作用 1](#_TOC_250005)

[第三章 编程军规条例 1](#_TOC_250004)

* 1. [可靠性管理 2](#_TOC_250003)
  2. [可维护性管理 10](#_TOC_250002)
  3. [性能管理 12](#_TOC_250001)
  4. [线程安全管理 14](#_TOC_250000)

1

IT 产品线 Java语言编程军规

# 第一章 前言

分析大量网上事故/ 问题、 缺陷库问题单、 和代码走读问题等根因时， 发现由于低级编码错误所引入的问题是主因。在此将常见问题提炼出来，以军规命名，以说明在开发

过程中必须遵守不能触犯的要求。

# 第二章 军规应用范围和作用

1. 该军规适用于 IT 产品线有关 JAVA 语言开发的所有产品；
2. 军规所有条例各产品部必须严格遵守， 用于网上问题/ 事故的回溯确认， 违反军规条例出现的错误，按照 IT 产品线关于低级错误发文处理。

# 第三章 编程军规条例

军规总览：

军规条例 1 【public 方法参数的合法性检查应由方法本身负责，每个 public 方法必须保证自己的健壮性。】

军规条例 2 【调用方法获取返回结果后必须进行有效性校验， 以确保后续代码在运行过程中不会抛出异常或产生逻辑错误。】

军规条例 3 【在进行数据库操作或 IO 操作时，必须确保资源在使用完毕后得到释放，并且必须确保释放操作在 finally 中进行。】

军规条例 4 【对于 if else if ( 后续可能有多个 else if ) 这种类型的条件判断， 最后必须包含一个 else 分支，避免出现分支遗漏， 造成错误； 每个 switch-case语句都必须保证有 default，避免出现分支遗漏，造成错误。】

军规条例 5 【在使用 Timer 或者 ScheduledThreadPoolExecutor执行周期性任务时， 实现 Runnable 接口必须在方法内捕获异常，避免因为异常抛出导致周期性住务失效，

后续不会继续执行。】

军规条例 6 【确定需要覆盖对象 equals()方法时，必须同吋覆盖 hashCode()方法。】

军规条例 7 【在进行精确计算时( 例如：货币计算) 避免使用 float 和 double, 浮点数计算都是不精确的，必须使用 BigDecimal 或者将浮点数运算转换为整型运算。】

军规条例 8 【确保程序不再持有无用对象的引用，避免程序内存泄露。】军规条例 9 【对象比较必须使用 equals 方法，而不是” ==”。】

军规条例 10 【访问数组、 List 等容器内的元素时，必须首先检查下标是否越界，以杜绝下标越界异常的发生。】

军规条例 11 【将对象存入 HashSe，t 或作为 key 存入 HashMap（或 HashTable）后，

1

IT 产品线 Java语言编程军规

必须确保该对象的 hashcode值不变，避免因为 hashcode值的变化导致不能从容器内删除该对象，进而引起内存泄露的问题。】

军规条例 12 【在编码过程中，必须在适当的位置，以适当的级别打印日志，便于代

码出现问题后进行定位分析。 例如：方法入口，出口以 Debug 打印：异常分支使用 ERROR,

关键参数打印使用 INFO。】

军规条例 13 【避免在程序中使用魔鬼数字，必须用有意义的常量来标识。】

军规条例 14 【将字符串转换为数字时必须处理 NumberFormatException 异常。】军规条例 15 【在进行三个字符串（不包含三个）以上的串联操作时必须使用

StringBuilder 或 StringBuffer ，禁止使用“ + ”。】

军规条例 16 【根据应用场景选择最适合的容器， 避免因为容器选择不当造成程序性能问题。】

军规条例 17 【必须在进行 I/O 操作时使用缓存。】

军规条例 18 【在程序中必须考虑对象重用，避免创建不必要的垃圾对象。】

军规条例 19 【对多线程访问的变量、方法，必须加锁保护，避免出现多线程并发访问引起的问题。】

军规条例 20 【新起一个线程时，都要使用 Thread.setName“( ” )设置线程名。】

军规条例 21 【线程使用时， 要在代码框架中使用线程池， 避免创建不可复用的线程。禁止在循环中创建新线程，否则会引起 JVM 资源耗尽。】

## 可靠性管理

军规条例 1 【public 方法参数的合法性检查应由方法本身负责， 每个 publi c 方法必须保证自己的健壮性。】

说明： public 方法作为一个模块或功能的对外接口，会接受各种调用者的调用。其中可能会存在传入不合法参数的情况，为了避免方法发生不可预知错误，必须在方法入口处进行参数合法性检查，以确保方法行为可控，保证系统健壮性。同时为了避免出现

过多的参数检查，在设计方法时应该在保证功能的前提下，尽量减少提供 public 方法。错误示例：

public String getDate (Date date)

{

// 缺少入参检验

SimpleDateFormatsf = new SimpleDateFormat( “ yyyy-MM-dHdH:mm:ss” ); return.sf.format(Data.data);

}

正确示例

public String getDate (Date date)

{

if (date == null) //增加入参校验

2

IT 产品线 Java语言编程军规

{

logger.log(ERROR,PROC, ” getDat:ad()atais null ” ); return null;

}

SimpleDateFormatsf = new SimpleDateFormat( “ yyyy-MM-dHdH:mm:ss” ); return.sf.format(Data.data);

}

军规条例 2 【调用方法获取返回结果后必须进行有效性校验, 以确保后续代码 在运行过程中不会抛出异常或产生逻辑错误。】

错误示例：

public void printDate()

{

Date date = new Date();

SimpleDateFormatsf = new SimpleDateFormat("yyyy-MM-dd HH:mm: ss"); String dateStr = sf.format(date);

//缺少对方返回结果的有效性检查System.out.println(dateStr.equals(date.toString()));

}

正确示例:

public void printDate()

{

Date date = new Date();

SimpleDateFormatsf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss"); String dateStr = sf.format(date);

if(dateStr == null)// 增加判空

{

logger.log(ERROR,PROC, ” printDate():getDareteturn null ” ); dateStr = date.toString();

}

System.out.printJn(dateStr.equals(date.toString()));

}

军规条例 3 【在进行数据库操作或 IO 操作时，必须确保资源在使用完毕后得到释放, 并且必须确保释放操作在 finally 中进行。】

说明：数据库操作、 IO 操作等需要使用结束 close()的对象必须 try-catch-finally 的

finally 中 close()，如果有多个 IO 对象需要 close()，需要分对每个对象的 close()方法进行 try-catch，防止一个 IO 对象关闭失败其他 IO 对象都未关闭。

错误示例：

BufferedOutputStream out = null; try

{

FileOutputStreamfos = new File0utputStream(ndest2");

3

IT 产品线 Java语言编程军规

out = new BufferedOutputStream(fos);

... ...

out.close(); // 在关闭操作前代码抛出异常的情况下，关闭操作不能被执行

}

catch (FileNotFoundException e)

{

//TODO

}

catch (IOException e)

{

//TODO

}

正确示例：

BufferedOutputStream out = null; try

{

FileOutputStreamfos = new FileOutputStream("dest2"); out = nev; Buf feredOutputStream(fos);

... ...

}

catch (FileNotFoundException e)

{

// TODO

}

catch (IOException e)

{

//TODO

}

finally

{

//在 finally 块中加入输出流的关闭操作

if(out != null)

{

try

{

}

out.close();

catch (IOException ioe)

{

//TODO

//ioe.printDetail();

}

}

4

IT 产品线 Java语言编程军规

}

军规条例 4 【对于 if elseif ( 后续可能有多个 else if ) 这种类型的条件判断， 最后必须包含一个 else 分支，避免出现分支遗漏造成错误； 每个 switch-case 语句都必须保证有 default ，避免出现分支遗漏，造成错误。】

军规条例 5【在使用 Timer 或者 ScheduleThreadPoolExecutor 执行周期性任务时, 实现

Runnable 接口必须在方法内捕获异常 , 避免因为异常抛出导致周期性任务失效 , 后续不会继续执行。】

错误示例：

Timer timer = new Timer(); timer.schedule(newTimerTask() { @Override

public void run()

{

// 业务代码，其中可能抛出异常

}

},delay, period);

正确示例：

Timer timer = new Timer (); timer.schedule(newTimerTask()

{

@0verride public void run()

{

try

{

}

// 业务代码

catch ( Exception e)// 增加异常捕获处理

{

// process

}

}

}, delay, period);

军规条例 6 【确定霈要覆写对象 equals()方法则必须同时覆写 hashCode()方法。】说明： equals 和 hashCode方法是对象在 hash 容器内高效工作的基础，正确的覆写

这两个方法才能保证在 hash 容器内查找对象的正确性，同时一个好的 hashCode方法能

大幅提升 hash 容器效率。示例：

... ...

private static final int HASHGODE\_RESULT = 17;

... ...

public classPoolEntry

5

IT 产品线 Java语言编程军规

{

private String value;

public boolean: equals(Object obj ){ if (obj == this) {

return true;

}

if (obj instanceof PoolEntry){ PoolEntry temp = (PoolEntry) obj;

if (value i=null && value.equals(temp.value) ) { return true;

}

return false;

}

return false;

}

public int hashGode(){

int result = HASHCODE\_RESULT;

//此处 37 是计算 hashcode使用的因子result = 37 \* result + value.hashCode(); return result;

}

}

军规条例 7 【在进行精确计算时( 例如: 货币计算) 避免使用 float 和 double ，浮点数计算都是不精确的，必须使用 BigDecimal 或将浮点数运算转换为整型运算。】

说明：浮点运算在一个范围很广的值域上提供了很好的近似，但是它不能产生精确

的结果。二进制浮点对于精度计算是非常不适合的，因为它不可能将 0.1——或者 10 的其它任何次负幂精确表示为一个长度有限的二进制小数。

错误示例:

public static void main(String[] args)

{

System.out.println(2.00 - 1.10);

}

输出结果:

0.9

正确示例:

public static void main(String[] args)

{

BigDecimal valuel = new BigDecimal( “ 2.00 ” ); BigDecimal value2 = new BigDecinmal("1,10"); System.out.println(valuel.subtract(value2));

}

输出结果:

6

IT 产品线 Java语言编程军规

0.9

军规条例 8 【确保程序不再持有无用对象的引用，避免程序内存泄露。】

说明：java 与 C/C++ 最大的区别之一就是 java 通过 GC（垃圾收集器) 自动回收内存，但是这并不意味着 java 代码中不存在内存泄露的问题。 Java中的内存泄露更准确的提法是“无意识的引用保留”， GC 会在程序运行过程中对每一个对象进行检查，如果当前程序中不在引用此对象，则此对象被标识为垃圾对象，可以被回收。但是如果程序中保留

了对无用对象的引用则会造成 GC 无法检测出垃圾对象， 进而无法回收垃圾对象的内存。错误示例：

public classStack

{

private Object[] elements; private int size = 0;

//堆栈数组默认长度

private static final int DEFAULT\_INITIAL\_CAPACITY = 16; public Stack()

{

elements= new Object[DEFAULT\_INITIAL\_CAPACITY];

}

public void push(Object e)

{

ensureCapacity(); elements[size++] = e;

}

public Object pop()

{

if (size == 0){

throw new EmptyStackException ();

}

/ \*

* 以下代码造成了内存泄露，对象出栈后仍保留了对此对象 \* 的引用，当出栈对象变成无用对象后，由于堆栈还持有引用，所以对象
* 不会被 GC 回收。

\*/

return elements[--size];

}

/\*\*

* 确保数组空间长度大于当前堆栈中元素个数，否则扩容

\*/

private void ensureCapacity()

{

if (elements.length== size)

{

elements= Arrays. copyOf (elements,2 \* size + 1);

7

IT 产品线 Java语言编程军规

}

}

}

正确示例：

public classStack

{

......

public Object pop()

{

if (size == 0)

{

throw new EmptyStackException ();

}

Object ret = elements[--size];

elements[size] = null; //清除堆栈持有的对象引用return ret;

}

... ...

}

军规条例 9 【对象比较必须使用叫 equals 而不是“==”】

说明：在 Java 中，有两种方式检查两个数据是否相等：通过使用 ==操作符，或者使用对象的 equals 方法。原子类型(int, char 等) 不是对象，因此他们只能使用==操作符，对象更复杂些， ==操作符检查两个引用是否指向同一个对象， 而 equals方法则实现更专门的相等性检查。

错误示例：

long time = System.currentTimeMillis(); Date today = new Date (time);

Date now = new Date (time);

System.out.println (now == today);//错误的对象比较结果：

false

正确示例:

long time = System.currentTimeMillis(); Date today = new Date(time);

Date now = new Date(time);

System.out.println(now.equals(today));//正确的对象比较结果：

True

军规条例 10 【访问数组、 List 等容器内的元素时，必须首先检查下标是否越界，杜绝下标越界异常的发生。】

8

IT 产品线 Java语言编程军规

错误示例：

public void checkArray(String name)

{

//获取一个数组对象

String[] clds = ContentService.queryByName(name); if (null != clds)

{

//只是考虑到 clds有可能为 null 的情况，但是 cids 完全有可能是长度的数组， 因此 clds[0] 有可能数组下标越界

String cid = clds[0]; cid.toCharArray();

}

}

正确示例：

public void checkArray(String name)

{

//获取一个数组对象

String[] elds = ContentService.queryByName(name); if (null != elds && clds.length > 0)

{

String cid=clds[0]; cid.toCharArray();

}

}

军规条例 11 【将对象存入 HashSet, 或作为 key 存入 HashlMap ( 或 HashTable ) 后， 必须确保该对象的 bashcode 值不变， 避免因为 hashcode 值变化导致不能从容器内删除该对象，进而引起内存泄露的问题。】

说明：对于 Hash 容器(HashMap，HashSet等) 而言， 对象的 hashcode至关重要， 在 hash

容 器内查找该对象完全依赖此值。 如果一个对象存入 Hash容器后 hashcode随即发生变化， 结果就是无法在容器内找到该对象，进而不能删除该对象，最终导致内存泄露。

错误示例：

public int hashCode()

{

int result = address.hashCode()r;eturn result;

}

......

HashSet<Email> set= new HashSet<Email>(); Email email = new Email("huawei.com"); set.add(email);

......

email.address= “ silong.com

......

set.remove(mail);

//”修;改地址值，导致 hashcode值变化



9

IT 产品线 Java语言编程军规

正确示例：

public int hashCode()

{

int result = address.hashCode()r;eturn result;

}

......

HashSet<Email> set= new HashSet<Ernail> (); Email email = new Email ( “ huawei.com ” ); set.add(email);

......//没有修改 address值，保证了 hashcode值不变set.remove(mail);

......

## 可维护性管理

军规条例 12 【在编码过程中必须在适当的位置， 以适当的级别打印日志， 便于出现问题后进行定位分析。例如：方法入口，出口以 Debug 打印; 异常分支使用 ERROR , 关键参数打印使用 INFO 。】

说明：在编码过程中必须选择合适的日志打印级别，打印信息必须是有用的 , 与本函数

相关的信息，严禁打印无关信息等。例如：在方法的入口出口处，方法关键信息处，异常处理处，条件判断发现错误处等等，都应该打印日志。这样一旦出现问题可以获取充分的定位信息，快速解决问题，增加程序可靠性。

军规条例 13 【避免在程序中使用魔鬼数字，必须用有意义的常量来标识。】

说明：魔鬼数字不会带来程序逻辑的错误，它主要影响代码可读性，读者看到的数字无法理解其含义，从而难以理解程序的意图。当程序中出现的魔鬼数字量过多时，代码的可维护性将会急剧下降，代码变得难以修改，并容易引入错误。

错误示例：

......

int itemCount = EXIST\_ITEM\_COUNT;// 这里可以认为不是魔鬼数字，但应该有注释

int itemSize = ITEM\_UNIT\_SIZE;// 这可以认为不是魔鬼数字，但应该有注释

//1024 是魔鬼数字

if （（ storageManager.getCapacity() -itemCount \* itemSize) < 1024）

{

storageManager.expandCapacityBy(512);//512 是魔鬼数字

}

......

正确示例：

......

//表示 lkb 单位容量

private static final int ONE KB = 1024;

//存储管理器扩大容最时使用的大小

10

IT 产品线 Java语言编程军规

private static final int EXPAND\_SIZE = 512; public void calculate()

{

......

int itemCount = EXIST\_ITEM\_COUNT; //已有数据数量 itemSize = ITEM UNIT SIZE; //每个数据的单位大小

if((storageManager.getCapacity()-itemCount \* itemSize) < ONE\_KB)

{

storageManager.expandCapacityBy(EXPAND\_SIZE);

}

.....

}

军规条例 14 【将字符串转换为数字时必须处理 NumberFormatExcepation 异常。】错误示例：

public Integer getlnteger(String number)

{

//如果 number 格式非法，例如： “ 910a”

//会抛出 NumberFormatException return Integer.valueOf(number);

}

正确示例：

public Integer getlnteger {String number).

{

try

{

}

return Integer.valueOf(number);

catch (NumberFormatException e)

{

//Process

}

}

## 性能管理

军规条例 15 【在进行三个字符串( 不包含三个) 以上的串联操作时必须使

StringBuilder 或 StringBuffer, 禁止使用”+”。】

说明：java 中字符串是不可变对象( immutable), 在进行字符串串联操作( 字符串+)时会生成临时对象。这些对象没有任何意义，会增加 JVM 垃圾收集的负担。如果串联字符串比较多会严重影响程序性能。 StringBiiffer 和 StringBuilder 功能完全一致，唯一的区别

11

IT 产品线 Java语言编程军规

只是 StringBuffer 中的每个方法都是线程安全的，在无需考虑线程同步的场景使用

StringBuilder 性能更高。错误示例：

String string = "test";

for (int i = 0; i < 50000; i++)

{

string += "test";

}

耗时： 8515ms

正确示例：

StringBuilder sb= new StringBuilder ( “ test ” ); for {int i = 0; i < 50000; i++ )

{

sb.append("test");

}

String string = sb.toString();

耗时： 16ms

军规条例 16 【根据应用场景选择最适合的容器， 避免因为容器选择不当造成程序性能问题】

说明：

ArrayList ：

* + 1. ArrayList 内部是使用 Object 数组来实现，随机访问速度很快。
    2. 向 ArrayList 内添加元素如果遇到内部数组需要扩容，则需要内存拷贝，速度慢。
    3. 在 ArrayList 内插入，删除元素涉及内存拷贝，速度很慢。

LinkedList:

1. LinkedList 内部使用链表实现，随机访问其中的元素很慢。
2. 在 LinkedList 内插入删除元素速度很快。

总结：如果需要频繁的随机访问容器中的数据，不需要频繁的对容器中的数据进行 修改或者移动，那么考虑使用 ArrayList; 如果你不需超频繁的随机访问容器中的数据， 需要频繁的对容器中的数据进行修改或者移动，那么考虑使用 LinkedList ;

HashSet:

Set 是集合类，该集合不能有“重复”对象存在。 HashSet将持有对象映射到在哈希表中，可快速存取对象。由于使用了 Hash 算法对对象的 hashCode方法强依赖。HashMap: Map 是一组 key-value ( 键值对）集合，其中的 key ( 键）不能重复。 HashMap 用 key 对象生成 hashcode然后映射到 Entry<K,V> [] 数组（键值对数组） 中, 其 get(Object

key)最佳时间复杂度为 O(1), 最坏则为 O (n), 因此高效的实现 key 对象的 hashCode方法对 HashMap的性能至关重要。在存在多线程并发访 HashMap的场景下必须考虑加/ 解锁。

军规条例 17 【必须在进行 I/O 操作时使用缓存。】

说明：不使用缓冲的 I/O 操作会频繁的调用操作系统底层函数访问磁盘，使用缓冲机制能带来显著的性能提升。

错误示例：

12

IT 产品线 Java语言编程军规

FileInputS.tre.am fis = new FilelnputStream(“srcl”） ;

// 没有使用缓存

FileOutputStream fos = new FileOutputStream（” destl"）; int read = -1;

while((read = fis.read()) != -1)

{

fos.write(read);

}

耗时:194985ms

正确示例:

FileInputStream fis = newFileInputStream(“src2”); FileOutputStream fos = new FileOutputStream(“dest2");

//增加缓存使用

BufferedOutputStream out = new BufferedOutputStream（ fos); byte [] buffer = new byte [512];

while (fis.read(buffer) != -1){ out.write(buffer);

}

耗 时:125ms

军规条例 18 【在程序中必须考虑对象重用，避免创建不必要的垃圾对象。】

说明： Java中除了基本类型外一切皆对象。一个大型系统中肯定存在大量对象互相协作完成各种功能。但是我们必须确保每一个对象是必须的，不是多余的垃圾对象。如果存在大量垃圾对象会增大 GC 负载，对程序性能造成严重影响。例如：程序响应请求时间明显变长，程序吞吐率降低等。

错误示例一：

List<Record> instancelnfoList = new ArrayList<Record>();

//查询数据库获取符合条件的记录列表

instanceInfoList = selectFromDB();//产生垃圾 ArrayList 对象

修改后：

//查询数据库获取符合条件的记录列表List<Record> instancelnfoList = selectFromDB();

错误示例二：

String str = new String（” stringette");每//

次都创建一个对象



修改后：

String str = “stringette"; //JVM 会确保在整个 JVM 范围内重用

13

IT 产品线 Java语言编程军规

## 线程安全管理

军规条例 19 【对多线程访问的变量、方法，必须加锁保护，避免出现多线程并发访问引起的问题。】

说明：目前在实际的产品中多线程的使用非常普遍，且问题较多，因此在进行编码 实现的时候必须重点考虑多线程并发的问题。 多线程并发问题的实质是多个线程间对共

享数据进行同步的问题。 根据 JavaLanguage Specification中对 Java内存模型的定义，JVM

中存在一个主内存 (JavaHeap Memory), Java 中所有变量都储存在主存中，对于所有线程都是共享的。 每条线程都有自己的工作内存 ( Working Memory) ，工作内存中保存的是主存中某些变量的拷贝，线程对所有变量的操作都是在工作内存中进行，线程之间无法 相互直接访问，变量传递均需要通过主存完成。

根据上述内存模型的定义， 要在多个线程间安全的同步共享数据就必须使用锁机制， 将某线程中更新的数据从其工作内存中刷新至主内存， 并确保其他线程从主内存获取此数据更新后的值再使用。

错误示例：

public classTest

{

private int count = 0;

//没有加锁进行同步处理public int increment()

{

return ++count;

}

public static void main(String... strings)

{

final Testtst = new Test(); new Thread(new Runnable ()

{

@Override public void run()

{

for (;;)

{

System.out.println(Thread.currentThread().getName() + " " + tst.increment());

}

}

}, "Thread-l").start(); new Thread{

new Runnable()

{

@Override . public void run()

{

14

IT 产品线 Java语言编程军规

for (;;)

{

System.out.println(Thread.currentThread().getName()+" "+ tst.increment());

}

}

}, "Thread-2").start();

}

}

结果：两线程中打印同一个变量，变最值存在跳变 : Thread-2 1234896

Thread-1 134853

正确示例：

public classTest

{

private int count = 0;

//增加方法同步

public synchronized int increment()

{

return ++count:;

}

public static void main{String... strings)

{

final Testtst = new Test(); new Thread(newRunnable ()

{

@Override public void run()

{

for (;;)

{

System.out.println(Thread.currentThread().getName()+" "+ tst.increment());

}

}

},"Thread-1") .start();

new Thread(newRunnable ()

{

@Override public void run ()

{

for (;;)

{

15

IT 产品线 Java语言编程军规

System.out.println(Thread.currentThread().getName()+ " " + tst.increment ());

}

}

}, "Thread-2").start ();

}

}

结果：两个线程中打印同一个变量，变量值不存在跳变。

Thread-2 95437

Thread-I 95438

军规条例 20 【每新起一个线程，都必须设置线程名。】

说明：这样在打印日志的时候就会把线程名称打印出来，如果不设置，线程名称默认为

Thread-1 这样类型的，日志区分不出来那个线程打印的。

军规条例 21 【线程使用时， 要在代码框架中使用线程池， 避免创建不可复用的线程。禁止在循环中创建新线程，否则会引起 JVM 资源耗尽。】

16