编码规范

1 前言

为确保系统源程序可读性，从而增强系统可维护性，java编程人员应具有基本类似的编程风格，兹制定下述Java编程规范，以规范系统Java部分编程。系统继承的其它资源中的源程序也应按此规范作相应修改。

2 适用范围

本文档将作为java编程人员软件开发的编程格式规范。在项目Java部分的编码、测试及维护过程中，要求严格遵守。

3 命名规范

定义这个规范的目的是让项目中所有的文档都看起来像一个人写的，增加可读性，减少项目组中因为换人而带来的损失。

3.1 Package 的命名

Package 的名字应该都是由一个小写单词组成。示例：unipost.trans

3.2 Class 的命名

Class 的名字每个单词必须由大写字母开头而其他字母都小写的单词组成。示例：FileMng

3.3 Class 成员的命名

变量、方法、属性：大小写混排的单词组成，首字母小写

示例： functionName、countNum、size

3.4 Static Final 变量的命名

Static Final常量：大写单词组成，单词之间使用“\_”连接

示例： MAX\_INDEX

3.5 前后台变量名称

前台变量 fg\_变量名

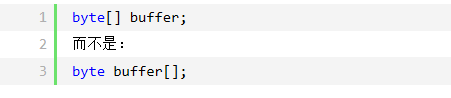
后台变量 bg\_变量名

3.6 参数的命名

参数的名字必须和变量的命名规范一致。

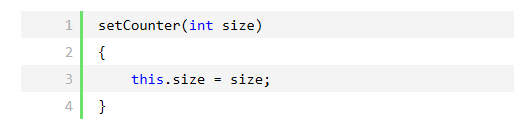
3.7 数组的命名

数组应该总是用下面的方式来命名：



3.8 方法的参数

使用有意义的参数命名，如果可能的话，使用和要赋值的属性一样的名字：



3.9 缩写

某些通用的缩写可以使用，如：

temp 可缩写为 tmp ;

message 可缩写为 msg ;

3.10 标识符命名中应注意的问题

3.10.1 除局部循环变量外变量名禁止取单个字符

对于变量命名，禁止取单个字符（如i、j、k...），建议除了要有具体含义外，还能表明其变量类型、数据类型等，但i、j、k作局部循环变量是允许的。

说明：变量，尤其是局部变量，如果用单个字符表示，很容易敲错（如i写成j），而编译时又检查不出来，有可能为了这个小小的错误而花费大量的查错时间。

3.10.2 不用数字定义名字

除非必要，不要用数字或较奇怪的字符来定义标识符。

示例：如下命名，使人产生疑惑。

void set\_sls00( BYTE sls );

应改为有意义的单词命名

void setUdtMsgSls( BYTE sls );

3.10.3 用正确的反义词组命名

用正确的反义词组命名具有互斥意义的变量或相反动作的函数等。

说明：下面是一些在软件中常用的反义词组。



3.10.4 避免使用

应避免使用\_EXAMPLE\_TEST\_之类以下划线开始和结尾的定义。

4 样式

4.1 Java 文件样式

所有的 Java(\*.java) 文件都必须遵守如下的样式规则

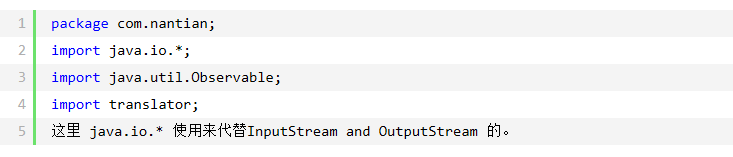
4.1.1 版权信息

版权信息必须在 java 文件的开头，示例：



4.1.2 Package/Imports

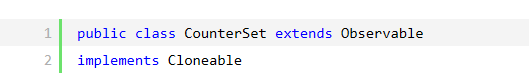
package 行要在 import 行之前，import 中标准的包名要在本地的包名之前，而且按照字母顺序排列。如果 import 行中包含了同一个包中的不同子目录，则建议用 \* 来处理。



4.2 Class的样式

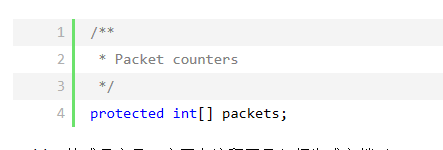
4.2.1 Class的定义

包含了在不同的行的 extends 和 implements



4.2.2 Class Fields

类的成员变量：



public 的成员变量一定要有注释而且必须生成文档（JavaDoc）。

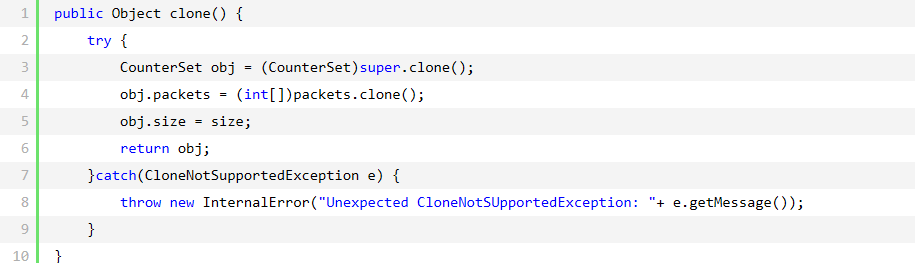
proceted、private和 package 定义的成员变量如果名字含义明确的话，可以没有注释。

4.2.3 构造函数

构造函数，它应该用递增的方式写（比如：参数多的写在后面）。

4.2.4 克隆方法

如果这个类是可以被克隆的：



4.2.5 类成员变量和方法的编写顺序

建议编写顺序为：

public protected private

final static transient

4.2.6 main 方法

如果main(String[]) 方法已经定义了, 那么它应该写在类的底部。

4.3 代码样式

代码应该用 unix 的格式，而不是 windows 的（比如：回车变成回车+换行）

5 注释

5.1 一般情况下

源程序有效注释量必须在20％以上。

说明：注释的原则是有助于对程序的阅读理解，在该加的地方都加了，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁。

5.2 常规注释标记说明

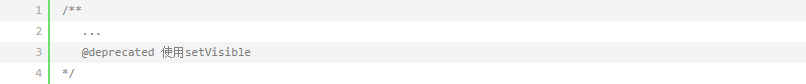
注释起始为“/\*\*…\*/”，注释文档的第一条为总结性语句，可在注释文档中使用HTML的标签语句，但要杜绝使用“HL”“HR”标签。注释全部采用中文，并依据以下标记规范进行书写。

5.2.1 @since

@since 文字：可生成一个“自从”条目，通过其中的“文字”，可说明一项特性是“自从”哪个版本开始引入的。

5.2.2 @deprecated

@deprecated 文字：可增加一条注释，指定特定的类、方法或变量不应继续使用，在这里，deprecated是“不赞成”，“不推荐”之意。利用其中的“文字”，可向用户推荐另一种方法来达到同样的目的。如：



5.2.3 @see

@see 链接：增加一个超链接。

5.2.3.1 若指向类、方法或变量名

可在@see后直接写上类、方法或变量名，书写方法时可省略包、类名，特性会默认为当前包或类。注意：类、方法、变量名间使用“#”分隔。

5.2.3.2 若指向一个具体的URL，则直接书写HTML标签锚。

5.2.3.3 若指向到“seealso（参考）”小节显示出来，使用“”””（双引号）。

5.2.4 @link

@link：可在注释中建立特殊的超链接，令其指向其他类或方法，标记规则同@see。

5.3 类和接口注释说明

类注释必须置于任何一个import语句后面，同时位于class定义的前面。

@author 名字：建立一个“作者”条目。

@version 文字：建立一个“版本”条目。

5.4 方法注释说明

紧靠在每条方法的前面，必须有一个它所描述的那个方法的签名。

@param 变量描述：给“parameters”（参数）小节增添一个条目。

@return 描述：增添一个“returns”（返回值）小节。

@throws 类描述：为方法的“throws”（产生违例）小节增添一个条目。

6 书写格式规范

6.1 代码编写规范

6.1.1 缩进

缩进应该是每行4个空格，在使用不同的源代码管理工具时Tab字符将因为用户设置的不同而扩展为不同的宽度.

如果你使用 UltrEdit 作为你的 Java 源代码编辑器的话，你可以通过如下操作来禁止保存Tab字符, 方法是通过 UltrEdit中先设定 Tab 使用的长度室4个空格，然后用 Format|Tabs to Spaces 菜单将 Tab 转换为空格。

case语句下的情况处理语句也要遵从语句缩进要求。

6.1.2 页宽

页宽应该设置为80字符. 源代码一般不会超过这个宽度, 并导致无法完整显示, 但这一设置也可以灵活调整. 在任何情况下, 超长的语句应该在一个逗号或者一个操作符后折行. 一条语句折行后, 应该比原来的语句再缩进4个字符。

示例：

permCountMsg.Head.Len = NO7\_TO\_STAT\_PERM\_COUNT\_LEN

+ STAT\_SIZE\_PER\_FRAM \* sizeof( \_UL );

actTaskTable[FrameID \* STAT\_TASK\_CHECK\_NUMBER + index].Occupied

= statPole[index].occupied;

6.1.3 空行

相对独立的程序块之间、变量说明之后必须加空行。

6.1.4 空格的使用

在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格；进行非对等操作时，如果是关系密切的立即操作符（如－>），后不应加空格。

说明：采用这种松散方式编写代码的目的是使代码更加清晰。

左括号和后一个字符之间不应该出现空格, 同样, 右括号和前一个字符之间也不应该出现空格. 下面的例子说明括号和空格的错误及正确使用:

CallProc( AParameter ); // 错误

CallProc(AParameter); // 正确

6.1.5 {}的用法

程序块的分界符（如大括号‘{’和‘}’）应各独占一行并且位于同一列，同时与引用它们的语句左对齐。在函数体的开始、类的定义、结构的定义、枚举的定义以及if、for、do、while、switch、case语句中的程序都要采用如上的缩进方式。

6.1.6 if等语句写法

if、for、do、while、case、switch、default、try、catch等语句自占一行，“{”必须回行编写，且if、for、do、while等语句的执行语句部分必须用{}括起回行编写，若有执行语句只有一条，{}可缺省。

示例：如下例子不符合规范

if (pUserCR == NULL) return;

应如下书写：

if (pUserCR == NULL)

{

return;

}

6.1.7 循环、判断等语句

循环、判断等语句中若有较长的表达式或语句，则要进行适当的划分，长表达式要在低优先级操作符处划分新行，操作符放在新行之首。

6.1.8 参数划分

若方法中的参数较长，则要进行适当的划分。

6.1.9 一行只写一条语句

不允许把多个短语句写在一行中，即一行只写一条语句。

示例：如下例子不符合规范。

rect.length = 0; rect.width = 0;

应如下书写

rect.length = 0;

rect.width = 0;

6.2 变量编写规范

6.2.1 公共变量

6.2.1.1 去掉没必要的公共变量。

说明：公共变量是增大模块间耦合的原因之一，故应减少没必要的公共变量以降低模块间的耦合度

6.2.1.2 仔细定义并明确公共变量的含义、作用、取值范围及公共变量间的关系。

说明：在对变量声明的同时，应对其含义、作用及取值范围进行注释说明，同时若有必要还应说明与其它变量的关系。

6.2.1.3 当向公共变量传递数据时，要十分小心，防止赋与不合理的值或越界等现象发生。

说明：对公共变量赋值时，若有必要应进行合法性检查，以提高代码的可靠性、稳定性。

6.2.2 局部变量

6.2.2.1 防止局部变量与公共变量同名。

说明：若使用了较好的命名规则，那么此问题可自动消除。

6.3 程序编写规范

6.3.1 exit()

exit 除了在 main 中可以被调用外，其他的地方不应该调用。因为这样做不给任何代码机会来截获退出。一个类似后台服务的程序不应该因为某一个库模块决定了要退出就退出。

6.3.2 异常

申明的错误应该抛出一个RuntimeException或者派生的异常。申明的错误主要指前提条件违例、处理流程违例的情况。对于功能性的分支建议采用返回值的方式。

异常建议根据模块结构，采用逐级处理的方式，并打印（或者记录在日志中）。同一模块层次的异常，按功能可由某一模块集中处理。

6.3.3 垃圾收集

JAVA使用成熟的后台垃圾收集技术来代替引用计数。但是这样会导致一个问题：你必须在使用完对象的实例以后进行清场工作（将对象置为NULL后，引用计数自动－1）。

6.3.4 final 类

绝对不要因为性能的原因将类定义为 final 的（除非程序的框架要求）

如果一个类还没有准备好被继承，最好在注释中注明，而不要将它定义为 final 的。这是因为没有人可以保证会不会由于什么原因需要继承。

6.3.5 访问类的成员变量

大部分的类成员变量应该定义为 private 的来防止继承类使用他们。

7 编程技巧

7.1 一般性原则

7.1.1 检查所有参数输入的有效性。

7.1.2 检查参数输入

检查所有非参数输入的有效性，如数据文件、公共变量等。

说明：输入主要有两种：一种是参数输入；另一种是全局变量、数据文件的输入，即非参数输入。在使用输入之前，应进行必要的检查。

7.1.3 类名应准确描述类的功能。

7.1.4 避免强制返回值类型

除非必要，最好不要把与返回值类型不同的变量，以编译系统默认的转换方式或强制的转换方式作为返回值返回。

7.1.5 让调用点显得易懂、容易理解。

7.1.6 减少数据类型转换

在填写参数时，应尽量减少没有必要的默认数据类型转换或强制数据类型转换。

说明：因为数据类型转换或多或少存在危险。

7.1.7 防止程序中的垃圾代码。

说明：程序中的垃圾代码不仅占用额外的空间，而且还常常影响程序的功能与性能，很可能给程序的测试、维护等造成不必要的麻烦。

7.1.8 减少递归调用。

说明：递归调用影响程序的可理解性；递归调用一般都占用较多的系统资源（如栈空间）；递归调用对程序的测试有一定影响。故除非为某些算法或功能的实现方便，应减少没必要的递归调用。

7.1.9 使用数据流图

仔细分析模块的功能及性能需求，并进一步细分，同时若有必要画出有关数据流图。

说明：根据模块的功能图或/及数据流图映射出结构是常用方法之一。

7.1.10 避免使用BOOL参数。

说明：原因有二，其一是BOOL参数值无意义，TURE/FALSE的含义是非常模糊的，在调用时很难知道该参数到底传达的是什么意思；其二是BOOL参数值不利于扩充。还有NULL也是一个无意义的单词。

7.2 开发过程中的技巧

7.2.1 byte 数组转换到 characters

为了将 byte 数组转换到 characters，你可以这么做：

"Hello world!".getBytes();

7.2.2 Utility 类

Utility 类（仅仅提供方法的类）应该被申明为抽象的来防止被继承或被初始化。

7.2.3 初始化数组

下面的代码是一种很好的初始化数组的方法：

objectArguments = new Object[] { arguments };

7.2.4 枚举类型

JAVA 对枚举的支持不好。

这种技术实现了RED, GREEN, BLUE 等可以象其他语言的枚举类型一样使用的常量。 他们可以用 == 操作符来比较。

但是这样使用有一个缺陷：如果一个用户用这样的方法来创建颜色 BLACK

new Colour(0,0,0) ，那么这就是另外一个对象。==操作符就会产生错误。她的 equal() 方法仍然有效。由于这个原因，这个技术的缺陷最好注明在文档中，或者只在自己的包中使用。

7.2.5 Swing

避免使用 AWT 组件

混合使用 AWT 和 Swing 组件

如果要将 AWT 组件和 Swing 组件混合起来使用的话，请小心使用。实际上，尽量不要将他们混合起来使用。

滚动的 AWT 组件

AWT 组件绝对不要用 JscrollPane 类来实现滚动。滚动 AWT 组件的时候一定要用 AWT ScrollPane 组件来实现。

避免在 InternalFrame 组件中使用 AWT 组件

尽量不要这么做，要不然会出现不可预料的后果。

7.2.6 Z-Order 问题

AWT 组件总是显示在 Swing 组件之上。当使用包含 AWT 组件的 POP-UP 菜单的时候要小心，尽量不要这样使用。

7.2.7 不必要的对象构造

不要在循环中构造和释放对象

7.2.8 synchronized 关键字

避免太多的使用 synchronized 关键字

避免不必要的使用关键字synchronized，应该在必要的时候再使用它，这是一个避免死锁的好方法。

Borland Jbulider 不喜欢 synchronized 这个关键字，如果你的断点设在这些关键字的作用域内的话，调试的时候你会发现的断点会到处乱跳，让你不知所措。除非必须，尽量不要使用。

7.3 程序效率

7.3.1 注意代码的效率

编程时要经常注意代码的效率。

说明：代码效率分为全局效率、局部效率、时间效率及空间效率。全局效率是站在整个系统的角度上的系统效率；局部效率是站在模块或函数角度上的效率；时间效率是程序处理输入任务所需的时间长短；空间效率是程序所需内存空间，如机器代码空间大小、数据空间大小、栈空间大小等。

7.3.2 提高代码效率

在保证系统的正确性、稳定性、可读性及可测性的前提下，提高代码效率。

说明：不能一味地追求代码效率，而对软件的正确性、稳定性、可读性及可测性造成影响。

7.3.3 局部效率应为全局效率服务

局部效率应为全局效率服务，不能因为提高局部效率而对全局效率造成影响。

7.3.4 循环体内工作量最小化。

说明：应仔细考虑循环体内的语句是否可以放在循环体之外，使循环体内工作量最小，从而提高程序的时间效率。

7.3.5 仔细分析有关算法，并进行优化。

7.3.6 改进输入方式

仔细考查、分析系统及模块处理输入（如事务、消息等）的方式，并加以改进。

7.3.7 提高调用不频繁的代码效率要慎重

不应花过多的时间拼命地提高调用不很频繁的代码效率。

说明：对代码优化可提高效率，但若考虑不周很有可能引起严重后果。

7.3.8 提高空间效率

在保证程序质量的前提下，通过压缩代码量、去掉不必要代码以及减少不必要的局部和全局变量，来提高空间效率。

说明：这种方式对提高空间效率可起到一定作用，但往往不能解决根本问题

7.3.9 循环的位置

在多重循环中，应将最忙的循环放在最内层。

说明：减少CPU切入循环层的次数。

7.3.10 尽量减少循环嵌套层次。

7.3.11 避免循环体内含判断语句

避免循环体内含判断语句，应将循环语句置于判断语句的代码块之中。

说明：目的是减少判断次数。循环体中的判断语句是否可以移到循环体外，要视程序的具体情况而言，一般情况，与循环变量无关的判断语句可以移到循环体外，而有关的则不可以。

7.3.12 不要一味追求紧凑的代码。

说明：因为紧凑的代码并不代表高效的机器码

8 性能

在写代码的时候，从头至尾都应该考虑性能问题。这不是说时间都应该浪费在优化代码上，而是我们时刻应该提醒自己要注意代码的效率。比如：如果没有时间来实现一个高效的算法，那么我们应该在文档中记录下来，以便在以后有空的时候再来实现她。

不是所有的人都同意在写代码的时候应该优化性能这个观点的，他们认为性能优化的问题应该在项目的后期再去考虑，也就是在程序的轮廓已经实现了以后。

8.1 可移植性

为了保证系统的可移植性，需要注意以下几点：

8.1.1 换行

如果需要换行的话，尽量用 println 来代替在字符串中使用"\n"。

你不要这样：

System.out.print("Hello,world!\n");

要这样：

System.out.println("Hello,world!");

或者你构造一个带换行符的字符串，至少要象这样：

String newline = System.getProperty("line.separator");

System.out.print ("Hello world" + newline);

8.1.2 PrintStream

PrintStream已经被不赞成（deprecated）使用，用PrintWrite来代替。

8.2 可测性

8.2.1 调测

在同一项目组或产品组内，要有一套统一的为集成测试与系统联调准备的调测开关及相应打印方法，并且要有详细的说明。

说明：本规则是针对项目组或产品组的。

8.2.2 调测信息串格式

在同一项目组或产品组内，调测打印出的信息串的格式要有统一的形式。信息串中至少要有所在模块名（或源文件名）。

说明：统一的调测信息格式便于集成测试。

8.2.3 在编程中注意单元测试

编程的同时要为单元测试选择恰当的测试点，并仔细构造测试代码、测试用例，同时给出明确的注释说明。测试代码部分应作为（模块中的）一个子模块，以方便测试代码在模块中的安装与拆卸（通过调测开关）。

说明：为单元测试而准备。

8.2.4 测试准备

在进行集成测试/系统联调之前，要构造好测试环境、测试项目及测试用例，同时仔细分析并优化测试用例，以提高测试效率。

说明：好的测试用例应尽可能模拟出程序所遇到的边界值、各种复杂环境及一些极端情况等。

8.2.5 测试手段

在软件系统中设置与取消有关测试手段，不能对软件实现的功能等产生影响。

说明：即有测试代码的软件和关掉测试代码的软件，在功能行为上应一致。

8.2.6 调测开关

用调测开关来切换软件的DEBUG版和正式版，而不要同时存在正式版本和DEBUG版本的不同源文件，以减少维护的难度。

8.2.7 调试与测试

在编写代码之前，应预先设计好程序调试与测试的方法和手段，并设计好各种调测开关及相应测试代码如打印函数等。

说明：程序的调试与测试是软件生存周期中很重要的一个阶段，如何对软件进行较全面、高率的测试并尽可能地找出软件中的错误就成为很关键的问题。因此在编写源代码之前，除了要有一套比较完善的测试计划外，还应设计出一系列代码测试手段，为单元测试、集成测试及系统联调提供方便。

8.2.8 调测开关的级别和类型

调测开关应分为不同级别和类型。

说明：调测开关的设置及分类应从以下几方面考虑：针对模块或系统某部分代码的调测；针对模块或系统某功能的调测；出于某种其它目的，如对性能、容量等的测试。这样做便于软件功能的调测，并且便于模块的单元测试、系统联调等。

8.2.9 编写防错程序

编写防错程序，然后在处理错误之后可用断言宣布发生错误。

示例：假如某模块收到通信链路上的消息，则应对消息的合法性进行检查，若消息类别不是通信协议中规定的，则应进行出错处理

9 质量保证

9.1 代码质量保证优先原则

（1）正确性，指程序要实现设计要求的功能。

（2）稳定性、安全性，指程序稳定、可靠、安全。

（3）可测试性，指程序要具有良好的可测试性。

（4）规范/可读性，指程序书写风格、命名规则等要符合规范。

（5）全局效率，指软件系统的整体效率。

（6）局部效率，指某个模块/子模块/函数的本身效率。

（7）个人表达方式/个人方便性，指个人编程习惯。

9.2 打开的文件要关闭

程序中申请的（为打开文件而使用的）文件句柄，在过程/函数退出之前要关闭。

9.3 一致性检查

系统运行之初，要对加载到系统中的数据进行一致性检查。

说明：使用不一致的数据，容易使系统进入混乱状态和不可知状态。

9.4 switch语句必须有default分支。

9.5 其他质量问题

使用第三方提供的软件开发工具包或控件时，要注意以下几点：

（1）充分了解应用接口、使用环境及使用时注意事项。

（2）不能过分相信其正确性。

（3）除非必要，不要使用不熟悉的第三方工具包与控件。

说明：使用工具包与控件，可加快程序开发速度，节省时间，但使用之前一定对它有较充分的了解，同时第三方工具包与控件也有可能存在问题。

资源文件（多语言版本支持），如果资源是对语言敏感的，应让该资源与源代码文件脱离

10 代码编辑、编译、审查

10.1 统一编译环境

在产品软件（项目组）中，要统一编译环境，相同的JDK版本

10.2 代码走读及评审

通过代码走读及审查方式对代码进行检查。

（1）代码走读主要是对程序的编程风格如注释、命名等以及编程时易出错的内容进行检查，可由开发人员自己或开发人员交叉的方式进行。

（2）代码审查主要是对程序实现的功能及程序的稳定性、安全性、可靠性等进行检查及评审，可通过自审、交叉审核或指定部门抽查等方式进行。

10.3 QA代码抽查

测试部测试产品之前，QA应对代码进行抽查及评审。

10.4 软件系统目录

合理地设计软件系统目录，方便开发人员使用。

说明：方便、合理的软件系统目录，可提高工作效率。目录构造的原则是方便有关源程序的存储、查询、编译、链接等工作，同时目录中还应具有工作目录----所有的编译、链接等工作应在此目录中进行，工具目录----有关文件编辑器、文件查找等工具可存放在此目录中。