# 代码编写规范

（v1.0）

# 目录

# 一 概述

## （1）编写目的

为了健全公司软件源代码的管理.增加源代码的可读性.提高软件开发 效率，特编写些规范。此规范并非源代码编写的一个标准.而是作为一个参考，以培养开发人员良好的编程习惯。

## （2）范围

开发时源代内的命名、注释规范（包括 JSP, C/C++, JAVA）。

# 二 规范内容

## 3.1 源代码标注规范（for c/c++）：

### 3.1.1 缩写规则

▲缩写后•个单词的长度宜为 3•6 个字母。
▲缩写后应能辨认出原意。
▲尽量使用公认的缩写习惯，如：Window 缩写为 Win; SmartPoint 缩写为 sp。

### 3.1.2 命名规则

使用含义鲜明、描述性的英文单词或缩写命名：命名由字母、数 字和下划线构成，一般情况下不得超过 32 个字符；命名建议采用组合词的方式。

#### 3.1.2.1 变量命名规则

般性规则：由单词（第•个单词或其缩写字母全部小写，以后 单词首字母大写，其余的字母小写）构成名词词组，如：int nValue; CString strTemp; CListClrl listUserlnf。;第一个单词（前缀）的 缩写格式原则上以匈牙利命名方式为准（参见衣•:匈牙利命名法）并且原则上不建议用下划线作为单词间的连接符。

| 前缀 | 类型 | 描述 | 实例 |
|---|---|---|---|
| arr | Array | 静态数组 | arrPoinT |
| b | BOOL | 布尔值 | bEnabled |
| by | unsigned char(BYTE) | 8 位无符号字符 | byWeek |
| ch | Char | 8 位字符 | chGrade |
| tch | TCHAR | 如果是_UNICODE,为 16 位字符 | tchName |
| i/n | Int | 整形(其大小依赖于系统) | Renge |
| u | UINT | 无符号整形(其大小依赖于 | nLength |

| | | | |
|---|---|---|---|
| | | 操作系统 | |
| us | unsianed shae | 无符号短整形 | usPort |
| W | WORD | 16 位无符号住 | wParam |
| | LONG | 32 位有符号整形 | offse |
| dw | DWORD | 32 位无符号整形 | dwVolume |
| f | Float | 4 字荧浮点数 | fHigh |
| d | Double | 8 字节浮点数 | dAmount |
| cy | CURRENCY | 8 字节货币类型 | V |
| P | * | 指针 | pDoc |
| lp | FAR* | 远指针 | DDo |
| PSZ | LPSTR | 为 32 位字符中指针 | lpszNamg |
| Csz | LPCSTR | 32 位常量字符中指针 | lpeszNamg |
| c+sz | LPCTSTR | 如果_UNICODE 定义，为 32 位常量字符中指针 | lpctszNomg |
| bstr | BSTR | 带有 4 字节长度前缀的 w_chart 数组 | ntrMessage |
| pbstr | BSTR* | BSTR 收据类型指针 | pbstrMsg |
| 5F | Strin | CString 炎型的字符出 | strMessag |
| och | OLECHAR | OLECHAR 字符类型 | echx |
| OSz | OLECHAR* | OLECHAR 字符中类型 | OszX |
| War | VARIANT | VARIANT 数据类型 | var |
| pvar | VARIANT* | VARIANT 数据类型指针 | pvar |
| 5Q | SAFEARRAY | SAFEARRAY 数据类用 | 50 |
| Psd | SAFEARRAY* | SAFEARRAY 数据类州指针 | psa |
| dlg | CDialog | Windows 对话框 | dlg |
| stc | CStatic | Windows 静态文本控件 | stc |
| edt | Cedit | Windows 维辑框控件 | edt |
| lsb | CListBox | Windows 列表控件 | lsb |
| btn | CButton | Windows 按钳控件 | btn |
| scb | CScrollBar | Windows 滚动条控件 | scb |
| spn | CSpinButterCte | Windowswe 设词技卸控件 | sp |
| tab | CTabCtrl | Windows 选项卡控任 | tab |
| rtf | CRichEditCtrl | Windows 富文木编辑框控件 | rtf |
| sld | CSliderCtrl | Window 泔块控件 | sld |
| list | ClistCtrl | Windows 列表控件 | ist |
| tre | CtreeCtrl | Windows 树形控件 | tre |
| dtc | CDate TimeCtrl | Windows 时间选取器控件 | dtc |

| | | | |
|---|---|---|---|
| Prg | CProgressCtp | Windows 进度指示器控件 | prg |
| cbo | CComboBox | Windows 组合柜控件 | cbo |
| ils | CTmaqeLis | Windows 刚像列表 | ils |
| tlb | CToolBarCtr | Windows 工具条控件 | +lb |
| mnu | Cmenu | Wir daws 单控件 | mnu |
| sta | C StatusBar | Windows状态条控件 | sta |
| wnd | CWnd | Windows 窗口 | wnd |
| cx | X | Windows X坐标 | cx |
| cy | y | Windows Y坐标 | cy |
| pt | C point | Windows点阵 | hWnd |
| size | C size | Windows尺寸 | size |
| rect/rc | Crect | Windows矩形 | rect |
| pen | C pen | Windows画笔 | pen |
| br | C brush | Windows刷子 | br |
| fnt | C font | Windows字体 | fnt |
| cr | COLORREF | Windows颜色值RGB | cr |
| PMp | Temp | 临时变量 | PMp |
| h | Handle | Windows对象句柄 | hWnd |
| i pfn | Callback | 指向CALLBACK函数的远指针 | I pfn Abort |
| it/iter | Iterator | 迭代子 | it |
| vect | std:: vector | 标准模板库 | vect |
| map | std: map | 标准模板库 | map |

| | | | |
|---|---|---|---|
| deque | std：deque | 标准模板库 | deque |
| list | std：list | 标准模板库 | list |
| set | std：set | 标准模板库 | Set |
| multiset | std：multiset | 标准模板库 | multiset |
| multimap | std：multimap | 标准模板库 | multimap |

表一：匈牙利命名法

▲指针类型的变量，必须加上前缀"p"，如:int* pnValue;
▲全局(静态)变量的命名规则同第一条的规定，并且必须加上前缀"g_"。
▲类成员(静态)变量的命名规则同第一条的规定，并且必须加上前缀"m_"。
▲模块(函数/过程)内部的重要变量命名遵循第一条的规定。
▲模块(函数/过程)内部的静态变量命名遵循第一条的规定，并且加上前缀"s_"。
▲其余变量在不损害程序易读易懂原则的基础上酌情处理。这些变量通常是模块内的临时变量，如循环控制变量、数组的下标等。

3.1.2.2 常变量命名规则

(1)宏(#define)或者常量修饰符(const):大写字母组成的名词词组，
单词间用下划线作为连接符,如:#define INVALID_CLOSE 0;
const double INVALID_CLOSE = double(0);。
(2)枚举类型(enum):此时的命名规则同"变量命名的一般性规则"，
可参考 MFC 或 ADO 等的命名方式。

3.1.2.3 函数命名规则

单词构成的动宾词组，每个单词首字母大写，其余字母小写。如: void DrawLine ( void );。

3.1.2.4 类命名规则

(1)类的原型，是前缀"C"加单词构成的动宾词组，每个单词首字母
大写，其余字母小写。如:class CStockPacket; class CPacket;。
(2)类的实例变量，采用小写首字母缩写的方式定义，如:CStockPacket sp;.
(3)在对类的成员变量和成员函数定义时，要加以分类，并界定相应的存取权限

| 分类描述 | 存取权限(依实际应用而定) |
|---|---|
| //Constructors | Public |
| //Attributes | Private |
| //Operations(对于 com 组件是//Methods) | Public |
| // Overridables | Protected |
| // Implementation | Private |

表二:类成员的分类描述

#### 3.1.2.5 结构命名规则

(1)结构的原型采用组合词的方式定义，并且全部采用大写字符的形式。如:

```
typedef struct tagStockTrans{
DWORD dwOpen;
...
} STOCKTRANS;
```

(2)结构的实例变量，采用小写首字母缩写的方式定义，如: STOCKTRANS st;。

### 3.1.3 语句构造规则

#### 3.1.3.1 一般性的原则

简单而直接，尽量避免出现难懂的技巧型代码。

#### 3.1.3.2 技巧型代码

为追求效率而出现的技巧型代码，必须加上足够详细的注释。

#### 3.1.3.3 代码规则

对以下的规则不做硬性规定，可酌情考虑:

(1)一个函数/过程的代码行数控制在 60 行(A4 打印纸可打印的行数)以内。
尽量避免复杂的测试条件。

(2)避免使用过深的(3 级以上)循环或条件嵌套，必要时可采用 goto 语句。

(3)循环或条件中的语句块控制在 60 行以内。

(4)利用括号使逻辑表达式或算术表达式的运算次序清晰直观。

### 3.1.4 书写格式规则

#### 3.1.4.1 语句

一行一条语句，赋值语句可例外，如:i=0,j= 1, k= 2;。

#### 3.1.4.2 层次结构

(1)"{"与"}"各占一行。

(2)"{"所在的列与其前一行语句对齐;"}"所在的列与对应的"{"对齐。

(3)"{"与"}"之间的语句相对缩进一个 Tab 键(设为 4 个空格字符)。

(4)注释与相关的代码首列对齐。

#### 3.1.4.3 空行和空格

(1)一组相关的函数/过程间以一个空行分隔;组间两个空行。

(2)行内注释(//)部分与语句间以空格或 Tab 分隔，数量酌情。

(3)多个函数参数或测试条件间用一个空格分隔

### 3.1.5 注释规则

#### 3.1.5.1 序言性注释

在模块开始处简要描述模块的功能、主要算法、接口特点、重要数据等的解释性说明。

(1)源程序(.cpp,.h etc)文件的开始处的格式(行宽 80 列以内):

```
// 文件名称:OptimizeObj.cpp
//
// Version xxx.xx.xx
//
// C opyright(c) 1999-2001 APEX International (ShangHai) Inc.
//
```

//创建人:XXX
// 创建日期:YYYY/MM/DD// 描述:
//
//修改人: XXX
//修改日期:YYYY/MM/DD
// 修改原因:
//描述:
//----------------------------------------
重要函数/过程的开始处的格式(行宽 80 列以内):
//----------------------------------------
//类 属:<如果是类的成员，此处填类的名称;否则忽略>// 函数名称:DemoFunc
//参数:int nParam1-参数一
// LPCSTR IpszParam2-参数二
// int* pnParam3 - [out]参数三
// 返回类型:BOOL
// TRUE- 成功
// FALSE-失败
// 功能描述:
//全局变量:
//调用模块:
// 备注:pnParam3 需要调用者释放内存
//创建人: XXX
// 创建日期:YYYY/MM/DD
//描述:
//修改人: XXX
// 修改日期:YYYY/MM/DD
//修改原因:

#### 3.1.5.2 数据说明注释

函数/过程中重要的变量必须加上注释，一行定义一个变量，注释放在行尾。

#### 3.1.5.3 插入性注释

在程序中间与一段代码有关的解释性说明。

(1)对于大段的功能相关代码的注释从行首开始，与前面的代码段分隔一行。

(2)对于小段(三四行以内)代码的注释放在语句末或下一行。

## 3.2 源代码标注规范(for JSP)

在编写 asp 程序时，主要用到 HTML、javascript、vbscript 等语言，下面将以 vbscript 为例详细说明编写规范，， javascript 语言规范请参考 vbscript。

### 3.2.1 编码约定概述

编码约定是帮助您使用 Microsoft Visual Basic Scripting Edition 编写代码的一些建议。编码约定包含以下内容:

对象、变量和过程的命名规则注释约定

• 文本格式和缩进指南

使用一致的编码约定的主要原因是使脚本或脚本集的结构和编码样式标准化，这样代码

易于阅读和理解。使用好的编码约定可以使源代码明白、易读、准确，更加直观且与其他语言约定保持一致。

### 3.2.2 常数命名规则

VBScript 的早期版本不允许创建用户自定义常数。如果要使用常数，则常数以变量的方式实现，且全部字母大写以和其他变量区分。常数名中的多个单词用下划线)分隔。例如:

USER_LIST_MAX NEW LINE

这种标识常数的方法依旧可行，但您还可以选择其他方案，用 Const 语句创建真正的常数。这个约定使用大小写混合的格式，并以“con”作为常数名的前缀。例如:conYourOwnConstant

### 3.2.3 变量命名规则

为提高易读和一致性，请在 VBScript 代码中使用以下变量命名规则:

子类型 前缀 示例

Boolean bln blnFound

Byte byt bytRasterData

Date (Time) dPM dPMStart

Double dbl dblTolerance

@wj504732875

Error err errOrderNum

Integer int intQuantity

Long lng lngDistance

Object obj objCurrent

Single sng sngAverage

String str strFirstName

### 3.2.4 变量作用域

变量应定义在尽量小的作用域中。VBScript 变量的作用域如下所示:

作用域，声明变量处，可见性，过程级.事件、函数或子过程.在声明变量的过程中可见。

Scrip 级 HPML 页面的 HEAD 部分,任何过在脚本的所有过程中可见。

### 3.2.5 变量作用域前缀

随着脚本代码长度的增加，有必要快速区分变量的作用域。在类型前缀前面添加一个单字符前缀可以实现这一点，而不致使变量名过长。

作用域 前缀 示例

过程级 无 dblVelocity

Script 级 s_blnCalcInProgress

### 3.2.6 描述性变量名和过程名

变量名或过程名的主体应使用大小写混合格式，并且尽量完整地描述其目的。另外，过程名应以动词开始，例如 InitNameArray 或 CloseDialog。

对于经常使用的或较长的名称，推荐使用标准缩写以使名称保持在适当的长度内。通常多于 32 个字符的变量名会变得难以阅读。使用缩写时，应确保在整个脚本中保持一致。例

如，在一个脚本或脚本集中随意切换 Cnt 和 Count 将造成混乱。

### 3.2.7 对象命名规则

下表列出了 VBScript 中可能用到的对象命名规则(推荐):

| 对象类型 | 前缀 | 示例 |
|---|---|---|
| 3D 面板 | pnl | pnlGroup |
| 动画按钮 | ani | aniMailBox |
| 复选框 | CHK | chkReadOnly |
| 组合框,下拉列表框 | cbo | cboEnglish |
| 命令按钮 | cmd | cmdExit |
| 公共对话框 | dlg | dlgFileOpen |
| 框架 | fra | fraLanguage |
| 水平滚动条 | hsb | hsbVolume |
| 图像 | img | imgIcon |
| 标签 | lbl | lb1HelPMessage |
| 直线 | lin | linVertical |
| 列表框 | lst | lstPolicyCodes |
| 旋钮 | spn | spnPages |
| 文本框 | txt | txtLastName |
| 垂直滚动条 | vsb | vsbRate |
| 滑块 | sld | sldScale |

### 3.2.8 代码注释约定

所有过程的开始部分都应有描述其功能的简要注释。这些注释并不描述细节信息(如何实现功能)，这是因为细节有时要频繁更改。这样就可以避免不必要的注释维护工作以及错误的注释。细节信息由代码本身及必要的内部注释来描述。

当传递给过程的参数的用途不明显，或过程对参数的取值范围有要求时，应加以说明。如果过程改变了函数和变量的返回值(特别是通过参数引用来改变)，也应在过程的开始部分描述该返回值。过程开始部分的注释应包含以下区段标题。相关样例,请参阅后面的“格式化代码”部分。

区段标题 注释内容目的 过程的功能(不是实现功能的方法)。假设其状态影响此过程的外部变量、控件或其他元素的列表。

效果 过程对每个外部变量、控件或其他元素的影响

效果的列表。输入,每个日的不明显的参数的解释。每个参数都应占据单独一行并有其内部注释。返回,返回值的解释。

请记住以下几点:

·每个重要的变量声明都应有内部注释，描述变量的用途。

应清楚地命名变量、控件和过程，仅在说明复杂细节时需要内部注释。

·应在脚本的开始部分包含描述该脚木的概述，列举对象、过程、运算

法则、对话框和其他系统从属物。有时一段描述运算法则的假码是很有用的。

### 3.2.9 格式化代码

应尽可能多地保留屏幕空间，但仍允许用代码格式反映逻辑结构和嵌套。以下为儿点提示:

·标准嵌套块应缩进 4 个空格。

，过程的概述注释应缩讲 1 个空格。

·慨述注释后的最高层语句应缩进 4 个空格，每一层嵌套块再缩进 4 个空格。

在每个 asp 文件的开头需要按一下规范编码:

```
'=========
' 文件名称:OptimizeObj.asp
' Version xxx.xx.xx
' C opyright(c) 1999-2001 APEXInternational (ShangHai) Inc.
' 创建人:xxX
' 创建口期:YYYY/MM/DD' 描述:
' 修改人:XXX
修改日期:YYYY/MM/DD' 修改原因:' 描述:
----- =================
```

下列代码符合 VBScript 函数编码规范。

```
==== ============
=二=====二
' 函数名称:DemoFunc
' 参数:nParaml-参数一
lpszParam2 参数二
,
pnParam3 参数三
' 返回:' 功能描述:' 备注:
,
' 创建人:xxx
' 创建口期:YYYY/MM/DD' 描述:
' 修改人:xxx
' 修改日期:YYYY/MM/DD 修改原因:
Function intFindUser(strUserList(), strTargetUser)
Dim i' Loop counter.
Dim blnFound' 找到目标标志 intFindUser = -1
i=0' 初始化循环计数器
Do While i<= Ubound(strUserList)and Not blnFound If strUserList(i)=
strTargetUser Then
blnFound = True' 将标志设置为 True
intFindUser =i' 将返回值设置成循环计数 End If
i=i+1' 递增循环计数器 Loop
End Function
```

## 3.3 源代码标注规范(for Java)

### 3.3.1 命名规范

Package 的命名

Package 的名字应该都是由一个小写单词组成。 Class 的命名

(1)Class 的名字必须由大写字母开头而其他字母都小写的单词组成

(2)Class 变量的命名

变量的名字必须用一个小写字母开头。后面的单词用大写字母开头。

(3)Static Final 变量的命名

Static Final 变量的名字应该都大写，并且指出完整含义。

(4)参数的命名

参数的名字必须和变量的命名规范一致。

(5)数组的命名

数组应该总是用下面的方式来命名:

byte[] buffer;

而不是:byte buffer[];

(6)方法的参数使用有意义的参数命名，如果可能的话，使用和要赋值的字段一样的名字:

```
setCounter(int size){
this.size= size;
```

### 3.3.2 Java 文件的样式规则

所有的 Java(*.java)文件都必须遵守如下的样式规则

版权信息

版权信息必须在 java 文件的开头，比如:

```
/**
* Copyright 2000 Shanghai XXX Co. Ltd.* All right reserved.*/
```

其他不需要出现在 javadoc 的信息也可以包含在这里。

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，则应该用*来处理。

```
package hotlava.net.stats;
import java.io.*;
import java.util.Observable;
import hotlava.util.Application;
```

这里 java.io.*使用来代替 InputStream and OutputStream 的。

Class

接下来的是类的注释，一般是用来解释类的。

```
/**
* A class representing a set of packet and byte counters* It is observable to allow it to be watched, but only* reports changes when the current set is complete*/
```

接下来是类定义，包含了在不同的行的 extends 和 implements

```
public class CounterSet
extends Observable
```

Class Frelds

接下来是类的成员变量:

```
/**
* Packet counters*/
protected int[] packets;
```

public 的成员变量必须生成文档(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。

存取方法

接下来是类变量的存取的方法。它只是简单的用来将类的变量赋值获取值的话，可以简单的写在一行上。

```
/**
* Get the counters
* @return an array containing the statistical data. This array has been* freshly allocated and can be modified by the caller,*/
public int[]getPackets(){return copyArray(packets,offset);} public int[]getBytes(){ return copyArray(bytes, offset);}
public int[]getPackets(){ return packets; }
public void setPackets(int[]packets){ this.packets = packets;}
```

其它的方法不要写在一行上.

构造函数

接下来是构造函数，它应该用递增的方式写(比如:参数多的写在后面)。访问类型("public"，"private"等.)和任何"static"，"final"或

"synchronized"应该在一行中，并且方法和参数另写一行，这样可以使方法和参数更易读。

```
public
CounterSet(int size){ this,size = size;
```

克隆方法

如果这个类是可以被克隆的，那么下一步就是 clone 方法:

```
public
Object clone() { try {
CounterSet obj = (CounterSet)super.clone(); obj.packets = (int[])packets.clone(); obj.size = size; return obj;
}catch(CloneNotSupportedException e){
throw new InternalError("Unexpected CloneNotSUpportedException:"+ e.gePMessage());
```

类方法

下面开始写类的方法:

```
/**
* Set the packet counters
* (such as when restoring from a database)*/
protected final
void setArray(int[] rl, int[] r2, int[] r3, int[] r4) throws
```

```
IllegalArgumentException{
    //
    // Ensure the arrays are of equal size//
    if (r1.length != r2.length ||r1.length !=r3.length ||r1.length != r4. length)
    throw new IllegalArgumentException("Arrays must be of the same size");
    System.arraycopy (r1,  0,  r3,  0，r1.length); System.arraycopy (r2,  0，r4，
0，r1.length);

    toString 方法
    无论如何，每一个类都应该定义 toString 方法:
    public
    String toString() {
    String retval= "CounterSet:";
    for (int i = 0; i < data.length(); i++) { retval += data.bytes.toString(); retval
+= data.packets.toString();
    return retval;
    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.
```

## 3.3.3 代码编写格式

(1)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)

(2)文档化:必须用 javadoc 来为类生成文档。不仅因为它是标准，这也是被各种 java 编译器都认可的方法。使用@author 标记是不被推荐的，因为代码不应该是被个人拥有的。

(3)缩进:缩进应该是每行 2 个空格.不要在源文件中保存 Tab 字符.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 2 个空格，然后用 Format|Tabs to Spaces 菜单将 Tab 转换为空格。

(4)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 2 个字符.

(5){}对

{}中的语句应该单独作为一行.例如，下面的第 1 行是错误的，第 2 行是正确的:

```
if (i>0) { i ++ };// 错误，{和}在同一行
if (i>0){
i ++
}; //正确，{单独作为一行
```

}语句永远单独作为一行.

}语句应该缩进到与其相对应的{那一行相对齐的位置。

(6)括号

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

不要在语句中使用无意义的括号.括号只应该为达到某种目的而出现在源代码中。下面的例子说明错误和正确的用法:

if ((I) =42) { //错误-括号毫无意义

if (I == 42) or (J== 42) { //正确-的确需要括号

(7)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有机会来截获退出。一个类似后台服务地程序不应该因为某一个库模块决定了要退出就退出。

(8)异常:申明的错误应该抛出一个 RuntimeException 或者派生的异常。顶层的 main()函数应该截获所有的异常，并且打印(或者记录在日志中)在屏幕

(9)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。比如一个 perl 的程序员可能这么写:

```
FileOutputStream fos = new FileOutputStream(projectFile);
project.save(fos,"IDE Project File");
...
```

除非输出流一出作用域就关闭，非引用计数的程序语言，比如 JAVA，是不能自动完成变量的清场工作的。必须象下面一样写:

```
implements Cloneable
public
Object clone()
try {
ThisClass obj = (ThisClass)super.clone(); obj.field1 = (int[])field1.clone();
obj.field2 = field2; return obj;
} catch(CloneNotSupportedException e) {
throw new InternalError("Unexpected CloneNotSUpportedException:"+
e.gePMessage());
```

(10)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，最好在类文档中注明，而不要将她定义为 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她。

(11)问类的成员变量

大部分的类成员变量应该定义为 protected 的来防止继承类使用他们。注意，要用"int[]packets"，而不是"int packets[]"，后一种永远也不要用。

```
public void setPackets(int[] packets) { this.packets = packets;}
CounterSet(int size)
this.size = size;
```

(12)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();
```

(13)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。

(14)初始化

下面的代码是一种很好的初始化数组的方法:

objectArguments = new Object[]{ arguments };

(14)枚举类型

JAVA 对枚举的支持不好，但是下面的代码是一种很有用的模板:

class Colour {

public static final ColourBLACK = new Colour(0, 0, 0); public static final ColourRED = new Colour(0xFF, 0, 0); public static final Colour GREEN = new Colour(0, 0xFF, 0); public static final Colour BLUE = new Colour(0, 0, 0xFF);

public static final Colour WHITE = new Colour(0xFF, 0xFF, 0xFF);

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。

### 3.3.4 Swing

(1)避免使用 AWT 组件

(2)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。

(3)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。

(4)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。

(5)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。

### 3.3.5 调试

调试在软件开发中是一个很重要的部分，存在软件生命周期的各个部分中。调试能够用配置开、关是最基本的。

很常用的一种调试方法就是用一个 PrintStream 类成员,在没有定义调试流的时候就为 null，类要定义一个 debug 方法来设置调试用的流。

### 3.3.6 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。

（1）不必要的对象构造

（2）不要在循环中构造和释放对象

（3）使用 StringBuffer 对象

在处理 String 的时候要尽量使用 StringBuffer 类，StringBuffer 类是构成 String

类的基础。String 类将 StringBuffer 类封装了起来，(以花费更多时间为代价)为开发人员提供了一个安全的接口。当我们在构造字符串的时候，我们应该用 StringBuffer 来实现大部分的工作，当工作完成后将 StringBuffer 对象再转换为需要的 String 对象。比如:如果有一个字符串必须不断地在其后添加许多字符来完成构造，那么我们应该使用 StringBuffer 对象和她的 append()方法。和释放对象的 CPU 时间。

（4）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她，这是一个避免死锁的好方法。

### 3.3.7 可移植性

（1）synchronized

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。

（2）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.
println("Hello world"+ newline);
```

（3）PrintStream

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