JAVA 编程规范

目　　录

1. 命名规则

1.1 包与类命名

1.1.1 包结构与类名前缀

为了便于管理Java开发的应用，Java开发的应用包命名要求带有前缀com，采用“com.包名.子功能包名”的形式，如com.bcl表示基础工具包。在日常项目应用中积累的公共的内容希望能够形成通用的工具包，供项目组使用，工具包与应用包同级，采用“com.应用包名.子应用包名”的形式来组织项目开发的程序代码。

包名和子功能包名要求具有实际的意义，从单词或缩写上能够看出包的意义，如 “com.bcl”表示基础工具组件的包，纳入公司的基础构件库进行管理，应用的包由项目组自己命名，命名的意义与具体的项目应用的意义相符。

1.1.2 类命名

(1) 类名首字母应该大写，字段、方法以及对象（句柄）的首字母应小写。对于所有标识符，其中包含的所有单词都应紧靠在一起，而且将中间单词的首字母大写。

例如：

ThisIsAClassName

thisIsMethodOrFieldName

若在定义中出现了常数初始化字符，则大写static final基本类型标识符中的所有字母。这样便可标志出它们属于编译期的常数。

Java包（Package）属于一种特殊情况：它们全都是小写字母，即便中间的单词亦是如此。对于域名扩展名称，如com，org，net或者edu等，全部都应小写（这也是Java 1.1 和Java 1.2 的区别之一）。

(2) 一行不要超过 80 个字符，并要注意折行时的写法。下面是例子：

```
someMethod(longExpression1, longExpression2, longExpression3,
        longExpression4, longExpression5);
var = someMethod1(longExpression1,
```

```
someMethod2(longExpression2,
        longExpression3));
```

1.2 变量命名（Attributes/Properties）

在本规则中，变量表示一个类属性（Attribute/Property）或一个类方法中的变量。变量可以是简单数据类型，如整数或浮点数，也可以是一个对象，如客户帐户、操作员等。

本变量命名规则还包括一类特殊的变量：枚举数据常量。

1.2.1 简单数据类型

循环计数器在不影响程序可读性的前提下，可以使用传统的变量命名方式，如i、m、n等。

除计数器以外的简单数据类型的变量由小写字母前缀+大写字母起头的英文单词（或单词缩写）组成。当然，循环计数器也可以使用这种命名方式，如nLoopCounter。

数组在变量名前缀后加数组前缀“a”。

简单数据类型前缀表：

数据类型	变量名前缀	例子
boolean	b	bCustomExist, baCustomExist
char	c	cSymbol, caSymbols
byte	y	yByte, yaBytes
short	s	sHour, saHours
int	n	nLen, nLoopCounter, naLength
long	l	lMemory, laTimes
float	f	fLength, faLength
double	d	dMile, daMiles
注：	byte 与 int 类型的前缀较特殊，使用时应注意	

1.2.2 类实例命名

类实例采用以下两种命名方法：

1．对于可以使用单个单词表示并且该单词与类名一致（不含类名的前缀）的类实例，可以使用全部是小写字母的实例名。例如operator（类WFOperator的实例）、task（类WFTask的实例）、date（类Date的实例）。

2．其它情况下使用以小写字母表示的类名（也可以是类名的缩写）和以大写字母起始的名称组成的标识名。例如strOperatorName、dateStart、

customerList。常用的类名缩写如下表：

类	类名缩写	例子
Exception	e	e, eAllException
SSBusiException	be	be, beLowBalance
SSDbAccess	dba	dba, dbaManagement, dbaCIF
SSLogiException	le	le, leInvalidValue
String	str	strOperatorName, strAddress
WFOperator	op	opCustomerManager, opDepartmentManager

3．类实例数组或集合的命名，可以在前两种情况的基础上用单词的复数形式表示，如tasks、opDepartmentManagers，也可以在变量前面加前缀“some”表示，如someTask、someOperator。

1.2.3 枚举数据常量

枚举数据常量由表示枚举类型的前缀和常量名称组成。前缀全部由小写字母组成，名称使用大写字母起头的英文单词。

例如交易类型的枚举常量命名如下：

不明交易类型	requestUnknown
菜单请求	requestMenuPrepare
任务准备	requestTaskPrepare
任务提交	requestTaskProcess
提交	requestReverseProcess
登录	requestLogin
系统应用准备	requestSysAppPrepare
系统应用提交	requestSysAppProcess

1.3 类方法命名

1.3.1 存取类型的类方法命名

对于直接操作类属性的方法，命名使用前缀“get”、“is”和“set”表示存或取类属性，后跟大写字母起头的英文单词。

示例如下：

Field	**Type**	**Gettername**	**Settername**
firstName	String	getFirstName()	setFirstName()
address	SurfaceAddress object	getAddress()	setAddress()
persistent	Boolean	isPersistent()	setPersistent()
customerId	Int	getCustomerId()	setCustomerId()
orderItems	Array of OrderItem objects	getOrderItems()	setOrderItems()

1.3.2 一般类型的类方法命名

使用“动词”+“名词短语”的命名格式，其中动词使用小写字母。例如

invokeProduct()、analyseRiskItem()、openAccount()、refreshOutPutStream()等。

2. 源程序结构

源程序按以下结构进行组织：

程序区	说明
Package 区	使用关键字“package”定义当前类的包
系统 import 区	
用户 import 区	
类说明	
类成员属性定义	
类方法定义	

注意：为了便于使用文档工具自动生成程序文档，程序员必须严格遵守格式说明。

3. Package区

定义类所属的包，例如：

```
package Main;
```

4. 系统import区

定义需要引入的系统资源，例如：

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import java.io.IOException;
import java.util.Vector;
```

5. 用户import区

定义需要引入的本系统内部的包，例如：

```
import SS.*;
import App.*;
```

6. 类说明

类说明使用如下格式：

```
/**
 * <p>类的详细说明</p>
 *
 * @author  类创建者姓名
 * @author  其他作者姓名
 * @version 1.00 9999/99/99 类创建者姓名
 * <p>       9.99 9999/99/99 修改者姓名 修改内容说明</p>
 * <p>       9.99 9999/99/99 修改者姓名 修改内容说明</p>
 * @see     参考类 1
 * @see     参考类 2
 */
```

格式说明：

项目	规则	备注
起始行	/**	该行不允许再有其他内容
说明行	* 说明内容	说明行中对该类的功能进行说明，可以有多个说明行
作者行	* @author 姓名	如果有多名作者，则分写几个作者行，类创建者在第一个作者行
版本行	* @version 1.00 9999/99/99 姓名 * 9.99 9999/99/99 姓名	“@version”为创建版本行。创建版本行的版本号固定为 1.00，日期采用“年/月/日”格式，后跟作者姓名，不止一人时中间以“，”分隔。 “ ”为修改版本行。修改版本行必须紧跟创建版本行，可以有多个修改版本行。修改版本行的版本号使用“主版本号.次版本号”格式，并且在姓名之后说明此次修改涉及的内容，其他要求与创建版本行的要求一致
参考行	* @see 参考内容	列出供参考的类或类的方法与属性（使用格式“类名”列出参考类；使用格式“类名#方法或属性名”列出具体方法或属性名），可以有多个参考行
结束行	*/	该行不允许再有其他内容
描述格式	描述信息	将描述信息使用 括起来

例子：

```
/**
 * <p>工作流管理子系统</p>
 *
 * @author  李昕
 * @author  雷鸣
 * @version 1.00 2001/8/30 李昕
```

```
 * <p>      1.01 2001/9/3  李昕，雷鸣 优化 invokeProduct()方法
 * <p>      1.02 2001/9/10 李昕       修改登录错误
 * @see     SSBusiException
 * @see     SSLogiException
 */
```

7. 类成员属性定义

类成员属性定义分为说明与定义两部分。

说明部分有三种方式，见下表：

方式	规则	备注
简单说明方式	/** 说明内容 */ 属性定义	对于功能比较单一的类属性，使用简单方式进行说明。 说明行以“/**”起头，以“*/”结束。 定义紧跟在说明后，另起一行。
详细说明方式	/** * 说明内容 * … */ 属性定义	对于功能复杂，需要详细说明的类成员属性，使用详细方式进行解释。 详细说明部分第一行必须是“/** ”，不能有其它内容；中间的说明行以“ * ”起头，后跟说明内容；结束行必须是“ */”，也不能有其它内容。 定义紧跟在说明后，另起一行。
枚举类型说明方式	/** 说明内容 */ 属性定义	对于定义的枚举类型（包括常量）的类属性，其定义放在说明部分之后，处于同一行上。 对于枚举类型，说明内容的格式为“枚举类型-内容说明”。

简单说明方式的例子：

```
/** 提示信息 */
private String strMsg = null;
```

详细说明方式的例子：

```
/**
 * 详细的补充信息，此属性在缺省方式下不填，
 * 需要时通过 addDetail(String strDetailMessage)方法补充
 */
private String strDetail = null;
```

枚举类型（常量）说明方式的例子：

```
/** 标记-格式 */       public static final String tagMessageFormat = "MF";
/** 标记-交易类型 */   public static final String tagRequestType = "RT";
/** 标记-工作流 ID */  public static final String tagWorkflowId = "WF";
/** 标记-子任务 ID */  public static final String tagSubtaskId = "ST";
/** 标记-操作员 ID */  public static final String tagOperatorId = "OP";
/** 标记-口令 */       public static final String tagPassword = "PW";
```

8. 类方法定义

类方法定义有方法说明与方法实现两部分，方法实现部分必须紧跟在方法说明部分下一行。

8.1 类方法说明

类方法说明使用如下格式：

```
/**
 * 类方法的详细使用说明
 *
 * @param  参数 1 参数 1 的使用说明
 * @param  参数 2 参数 2 的使用说明
 * @param  参数 3 参数 3 的使用说明
 * @return 返回结果的说明
 * @throws 异常类型.错误代码 注明从此类方法中抛出异常的说明
 * @throws 异常类型 注明原来从此类方法的被调用方法中抛出的异常
 * @see    参考类 1
 * @see    参考类 2#类方法或类属性
 */
```

格式说明：

项目	类型	规则	备注
起始行	必选	/**	该行不允许再有其它内容
说明行	必选	* 说明内容	说明行中对该方法的功能进行说明，可以有多个说明行
参数行	可选	* @param 参数 参数说明	一行对一个参数进行说明，可以有多个参数行。 如果该方法没有任何参数，此项可省略。
返回行	可选	* @return 返回结果说明	对方法的返回结果进行详细说明，只允许有一个返回行，但是可以折行。 如果该方法没有返回，此项可省略。
直接引发行	可选	* @throws 异常.代码 抛出说明 *	一行对一类异常的一种错误码进行说明，可以有多个直接引发行。代码使用系统预定义的代码常量，不能直接写数字。 列出该方法直接抛出的异常，并说明在何种情况下抛出该异常。
间接引发行	可选	* @throws 异常 说明抛出者	一行对一类异常进行说明，可以有多个间接引发行。 列出该方法调用的方法可能抛出的异常。
参考行	可选	* @see 参考内容	列出供参考的类或类的方法与属性（使用格式“类名”列出参考类；使用格式“类名#方法或属性名”列出具体方法或属性名），可以有多个参考行

项目	类型	规则	备注
结束行	可选	*/	该行不允许再有其他内容

例子

```
/**
 * 调用相关产品的指定方法
 *
 * @param  request          请求参数
 * @param  ac               环境参数
 * @param  strProductMethod 产品方法名
 * @return 应用处理结果
 * @throws SSLogiException.codeNoEnoughParam 没有合适的方法或参
 *                                           数不足时抛出异常
 * @throws SqlException     MainScheduler.CallProduct()抛出
 * @see    AppContext
 * @see    AppOutput#form
 * @see    SSLogiException
 * @see    SqlException
 */
public abstract AppOutput invokeProduct(HttpServletRequest req,
                                        AppContext ac,
                                        String strProductMethod)
                                        throws SSLogiException,
                                               SqlException;
```

8.2 类方法实现部分编程规范

8.2.1 程序注释的形式

本系统中，程序注释分以下三种形式：

- 详细说明式：

较详细地解释后续处理的功能、算法等内容，可以有多行说明。

```
/**
 * 说明内容
 * ......
 */
程序行.....
```

- 简单说明式：

简要说明后续处理，注释内容在一行内写完。

```
/** 说明内容 */
程序行.....
```

- 行末说明式：

紧接在一个程序行最右边，对该行程序进行简要说明，注释内容在一行内写完。

```
程序行……;  //注释内容
```

8.2.2 局部变量

原则上，一行中只允许定义一种类型的同一种用途的变量。局部变量定义必须在同一行定义语句后面加以注释，使用行末说明式的注释方法。一个变量只能用于一种用途。例如用于循环计数的变量就不要再用作记录函数的返回值。

变量注释举例如下：

```
int i, n;                          //循环计数器
int nPromptMsgLength;              //提示信息长度
WFOperator operator;               //当前操作员
WFOperator opDepartmentManager;    //当前部门主管
```

8.2.3 语句块

完成一种功能的一组语句组成一个语句块。语句块之间应该使用空行适当分隔。在语句块前面必须加注释，详细说明其后的语句块的功能。注释采用详细说明或简单说明的注释形式。

```
/**
 * 功能说明一
 */
语句
语句
……
                （空行以分隔两个功能的语句块）
/**
 * 功能说明二
 */
语句
语句
……
```

在语句块的注释中，必须包括详细设计文档中的所有伪码，以标明此段程序是在实现哪部分详细设计。

8.2.4 条件判断 if…else…

每一个条件判断语句前面使用详细说明或简单说明形式进行注释，在每一个分支内或者使用详细/简单说明形式进行注释，或者在条件分支语句后面以行末说明形式进行注释。

例如在条件分支内进行注释：

```
/**
 * 判断说明
 */
if (...){
    /**
     * 处理说明
     */
    … …
}
else{
    /**
     * 处理说明
     */
    … …
}
```

或者在条件分支语句末进行注释：

```
/**
 * 判断说明
 */
if (...){ //处理说明
    … …
}
else{ //处理说明
    … …
}
```

8.2.5 条件判断 switch…case…default…

与if式的判断语句类似，在每一个条件判断语句前面使用详细说明或简单说明形式进行注释，在每一个分支内或者使用详细/简单说明形式进行注释，或者在条件分支语句后面以行末说明形式进行注释。

例如在条件分支内进行注释：

```
/**
 * 判断说明
 */
switch (...){
    case ...:
        /**
         * 处理说明
         */
        … …
        break;
```

```
            … …
        default:
            /**
             * 处理说明
             */
            … …
    }
```

或者在条件分支语句末进行注释：

```
    /**
     * 判断说明
     */
    switch (...){
        case …: //处理说明
            … …
            break;
            … …
        default: //处理说明
            … …
    }
```

8.2.6 循环控制语句

循环语句块前必须使用详细说明或简单说明形式进行注释说明，在循环体内部的每一条循环控制语句或者使用详细/简单说明形式进行注释，或者使用行末说明形式进行注释说明。

例如：

```
    /**
     * 循环处理说明
     */
    for (...;...;...){
        … …
        break; //退出循环说明（如何种条件下退出循环）
        … …
        continue; //循环控制条件说明
        … …
    }
```

8.2.7 方法调用

在每一个方法调用前使用详细说明或简单说明形式进行注释，解释调用该方法的目的。

例如：

```
    /**
```

```
     * 使用得到的方法名调用合适的应用功能
     */
    output = invokeProduct(request, ac, strMethodName);
```

8.2.8 编程风格的要求

1．大括号（“{”与“}”）的使用

类定义中使用起止括号均左边齐头的大括号，例如：

```
public Class Datetime
{
    …
}
```

类成员方法的实现部分，可以选择以下两种方式中的一种：起止括号均左边齐头或者起始括号在同一程序行尾部，终止括号左边齐头的格式，例如：

```
/**
 * 方法原型部分较短时，使用下面的格式
 */
public double getBalance() throws SSLogiException{
    …
    if(dBalance < dAmount){    //如果余额小于本次发生额，提示错误
        …
    }
}

/**
 * 方法原型部分较长时，使用下面的格式
 */
public double getBalance(String strItemName, int nItemType)
        throws SSException, SQLException
{
    …
    if(dBalance < dAmount){    //如果余额小于本次发生额，提示错误
        …
    }
}
```

其他位置一律使用起始括号在同一程序行尾部，终止括号左边齐头的格式，例如：

```
    if(dBalance < dAmount){    //如果余额小于本次发生额，提示错误
        …
    }
    else{        //余额大于或等于本次发生额时
        …
    }
```

2. 缩进格式

每一层嵌套向右侧缩进 4 个空格。以下列出了各种情况下的嵌套格式，请参考。

函数定义：

```
public double getBalance() throws SSLogiException{
    …
}
```

条件语句 if…else…：

```
if(dBalance < dAmount){    //如果余额小于本次发生额，提示错误
    …
}
```

或：

```
if(dBalance < dAmount){    //如果余额不足，提示错误
    …
}
else{                      //否则（余额足），正常支取
    …
}
```

或

```
if(dBalance < dAmount){    //如果余额不足，提示错误
    …
}
else if(bValid == true){   //余额足，但账户不正常，提示错误
    …
}
else{                      //余额足，且账户正常，正常支取
    …
}
```

条件分支语句 switch…case…default…：

```
switch(nSex){
    case sexMale:    //男士
        …
        break;
    case sexFemale:  //女士
        …
        break;
    default:         //其他，提示错误
        …
}
```

循环语句 for：

```
/**
 * 对所有的账户进行处理
 */
for(i = 0; i < nAccountNum; i++){
    …
}
```

循环语句 while：

```
/**
 * 对所有的账户进行处理
 */
i = 0;
while(i < nAccountNum){
    …
    i++;
}
```

循环语句 do…while：

```
/**
 * 至少有一个账户，对所有的账户进行处理
 */
i = 0;
do{
    …
    i++;
} while(i < nAccountNum)
```

3． 空格的使用

等号左右必须各有一个空格：

```
strName = null;
```

双目运算符左右必须各有一个空格：

```
strFullName = strFirstName + strLastName;
```

标点符号后面必须跟一个空格

标点符号包括“,”、“;”等，下面列出几个例子。

一行定义多个变量时，“,”后跟空格：

```
int i, j;    //循环变量
```

在for循环中，“;”后跟空格：

```
for(i = 0; i < nAccountNum; i++)…
```

在有多个入口参数的函数调用中，“,”后跟一个空格：

```
strCustomId = getCustomId(nCustomType, strCustomName)
```

9. 编写类和方法时的一些约定

- 对于自己创建的每一个类，都考虑置入一个main()，其中包含了用于测试那个类的代码。为使用一个项目中的类，没必要删除测试代码。若进行了任何形式的改动，可方便地返回测试。这些代码也可作为如何使用类的一个示例使用。
- 应将方法设计成简要的、功能性单元，用它描述和实现一个不连续的类接口部分。理想情况下，方法应简明扼要。若长度很大，可考虑通过某种方式将其分割成较短的几个方法。这样做也便于类内代码的重复使用（有时，方法必须非常大，但它们仍应只做同样的一件事情）。
- 使类尽可能短小精悍，而且只解决一个特定的问题。下面是对类设计的一些建议：

一个复杂的开关语句考虑采用“多形”机制；数量众多的方法涉及类型差别极大的操作时，考虑用几个类来分别实现；许多成员变量在特征上有很大的差别时，考虑使用几个类。

- 让一切东西都尽可能地“私有”——private。使库的某一部分“公共化”（一个方法、类或者一个字段等），就永远不能把它拿出。若强行拿出，就可能破坏其他人现有的代码，使他们不得不重新编写和设计。若只公布自己必须公布的，就可放心大胆地改变其他任何东西。在多线程环境中，隐私是特别重要的一个因素——只有private字段才能在非同步使用的情况下受到保护。
- 任何时候只要发现类与类之间结合得非常紧密，就需要考虑是否采用内部类，从而改善编码及维护工作。
- 尽可能细致地加上注释，并用javadoc注释文档语法生成自己的程序文档。
- 当客户程序员用完对象以后，若你的类要求进行任何清除工作，可考虑将清除代码置于一个良好定义的方法里，采用类似于cleanup()这样的名字，明确表明自己的用途。除此以外，可在类内放置一个boolean（布尔）标记，指出对象是否已被清除。在类的finalize()方法里，请确定对象已被清除，并已丢弃了从RuntimeException继承的一个类（如果还没有的话），从而指

出一个编程错误。在采取像这样的方案之前，请确定finalize()能够在自己的系统中工作（可能需要调用System.runFinalizersOnExit(true)，从而确保这一行为）。

- 在一个特定的作用域内，若一个对象必须清除（非由垃圾收集机制处理），请采用下述方法：初始化对象；若成功，则立即进入一个含有finally从句的try块，开始清除工作。
- 若在初始化过程中需要覆盖（取消）finalize()，请记住调用super.finalize()（若Object属于直接超类，则无此必要）。在对finalize()进行覆盖的过程中，对super.finalize()的调用应属于最后一个行动，而不应是第一个行动，这样可确保在需要基础类组件的时候它们依然有效。
- 创建大小固定的对象集合时，请将它们传输至一个数组（若准备从一个方法里返回这个集合，更应如此操作）。这样一来，人们就可享受到数组在编译期进行类型检查的好处。此外，为使用它们，数组的接收者也许并不需要将对象“造型”到数组里。
- 尽量使用interfaces接口，不要使用abstract抽象类。若已知某样东西准备成为一个基础类，那么第一个选择应是将其变成一个interface（接口）。只有在不得不使用方法定义或者成员变量的时候，才需要将其变成一个abstract（抽象）类。接口主要描述了客户希望做什么事情，而一个类则致力于（或允许）具体的实施细节。
- 在现成类的基础上创建新类时，请首先选择“新建”或“创作”。只有自己的设计要求必须继承时，才应考虑这方面的问题。若在本来允许新建的场合使用了继承，则整个设计会变得没有必要的复杂。
- 为避免编程时遇到麻烦，请保证在自己类路径指到的任何地方，每个名字都仅对应一个类。否则，编译器可能先找到同名的另一个类，并报告出错消息。若怀疑自己碰到了类路径问题，请试试在类路径的每一个起点搜索一下同名的.class文件。

10. 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。程序中类的描述要求符合Javadoc的规范，如下面所示：

```
/**
 * <P>向缓冲池中增加一个属性和相应的字符串值</P>
 *
 * @return int
 * @param attribute java.lang.String
 * @param data java.lang.String
 * @exception java.lang.Exception
 */
```

描述信息使用 括起来，返回参数、传入参数、异常处理进行申明。

11. 附录

11.1 本系统常用单词表

单词	词意	建议的缩写
Analysis	分析	
Class	类	
Client	客户端	
Component	组件，业务组件	
Customer	客户	
Date	日期	
Department	部门，处，室	
DepartmentManager	部门主管	dpm
Id	标识	
Length	长度	len
Manager	主管	
Message	消息，信息	msg
Method	方法，操作，成员函数	
Object	对象	
Offset	偏移量	off
Operator	操作员，柜员	op
Password	口令	
Phase	阶段	
Product	产品，金融产品	
ProductPackage	产品包	ppk
Prompt	提示	
Risk	风险	
Subtask	子任务	
Symbol	标识符，记号	
Task	任务	
Time	时间	
Workdate	工作日	wdt
Workflow	工作流	wfl

11.2 参考资料

[1] Scott W. Ambler.《Writing Robust Java Code - The AmbySoft Inc. Coding Standards for Java》.v17.01d.

[2] Silver Siu.《The Java Coding Standard》.Version 1.1.

[3] Sun Microsystems, Inc.《How to Write Doc Comments for the Javadoc(TM) Tool》.