Java语言编程及命名规范

<版本V1.0>

目 录

[修改历史： 3](#_Toc299379746)

[1 前言 4](#_Toc299379747)

[1.1 目的 4](#_Toc299379748)

[1.2 声明 4](#_Toc299379749)

[2 编程规范 5](#_Toc299379750)

[2.1 类定义结构 5](#_Toc299379751)

[2.2 注释 6](#_Toc299379752)

[2.2.1 类文档注释 6](#_Toc299379753)

[2.2.2 方法文档注释 7](#_Toc299379754)

[2.2.3 方法内部注释 7](#_Toc299379755)

[2.2.4 代码修改注释 7](#_Toc299379756)

[2.3 包引入语句声明 8](#_Toc299379757)

[2.4 空白 9](#_Toc299379758)

[2.4.1 空行 9](#_Toc299379759)

[2.4.2 空格 10](#_Toc299379760)

[2.5 代码缩进 11](#_Toc299379761)

[2.5.1 行长度(Line Length) 11](#_Toc299379762)

[2.5.2 换行(Wrapping Lines) 11](#_Toc299379763)

[2.6 变量声明 12](#_Toc299379764)

[2.7 语句 12](#_Toc299379765)

[2.8 其它 13](#_Toc299379766)

[2.8.1 泛型 13](#_Toc299379767)

[2.8.2 嵌套 13](#_Toc299379768)

[2.8.3 代码行数 15](#_Toc299379769)

[3 命名规范 16](#_Toc299379770)

[3.1 包命名 16](#_Toc299379771)

[3.2 类和接口命名 17](#_Toc299379772)

[3.3 变量（含参数）或常量命名 17](#_Toc299379773)

[3.4 方法命名 18](#_Toc299379774)

[3.4.1 数据访问类（DAO）方法命名 18](#_Toc299379775)

[3.4.2 Action层方法命名规范 19](#_Toc299379776)

[3.5 项目命名规范 20](#_Toc299379777)

[3.5.1 项目名 20](#_Toc299379778)

[3.5.2 发布包名 20](#_Toc299379779)

[3.5.3 项目目录结构 20](#_Toc299379780)

[4 附录1. JAVA源程序样本 21](#_Toc299379781)

# 修改历史：

|  |  |  |  |
| --- | --- | --- | --- |
| **时间** | **版本** | **描述** | **作者** |
| 2011-7-25 | 1.0 | 初稿 | xxx |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 前言

## 目的

本规范的主要目的是消除研发团队中不同程序员之间的个性化的东西，以提高代码的可维护性。

据统计，通常在一个软件的生命周期中，维护的工作量占整个软件生命周期全部工作量的70%以上。我们程序员绝大部分时间都用在阅读、修改既存的代码上，而不是开发新的代码。一个规范化的可维护性高的软件系统，在后期维护的时候节省的工作量是非常巨大的。

试想一下，如果每个人在读别人的代码的时候就像是自己开发的代码一样，那是一件多么愉快的事情。

## 声明

本规范中有些规定是有一定的科学性的，但也有些规定并没有十分充足的理由，请不要较真，没有别的，只是为了统一风格，让每个人生产出来的代码看不出来是哪位高手编写出来的。

# 编程规范

## 类定义结构

一个类定义各个部分在类文件（.java）中出现的先后顺序：包名定义、依赖类引入声明、类注释、类定义、静态变量/常量定义、成员变量（非静态）定义、公共方法实现、私有方法实现。对变量/方法，其定义声明优先级是：1）静态→常量→普通成员变量；2）构造函数→共有→保护→私有→getter/setter。例如：

**package** com.pangu.usercenter.sample;

/\*\*

\* ClassA的功能、职责、设计思想介绍，如果必要的话进行使用方法说明。

\*

\* **@author** liufengyu

\* **@newDate** 2011-1-14

\*/

**public** **class** ClassA {

/\*\*

\* 静态变量/常量功能名称（例如：权限平台的url）

\*/

**private** **static** **final** String *XML\_RPC\_SERVER\_URL* = "xmlRpcServerUrl";

//成员变量\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* 成员变量功能名称（例如：公司名称）

\*/

**private** String comName = **null**;

//构造函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* 构造函数

\*

\* **@param** comName 公司名称

\*/

**public** ClassA(String comName) {

**this**.comName = comName;

}

//共有函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* 方法methodA的功能说明

\*

\* **@param** p1 参数名称

\* **@return**

\*/

**public** String methodA(String p1) {

**return** **this**.methodB();

}

//私有函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**private** String methodB() {

**return** **null**;

}

//getter/setter\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** String getComName() {

**return** comName;

}

**public** **void** setComName(String comName) {

**this**.comName = comName;

}

}

## 注释

### 类文档注释

类文档注释描述该类实现的基本功能、职责、设计思想等，如果必要的话还需要对该类的使用方法进行说明。该注释可以自动生成javadoc文档。如：

/\*\*

\* 该类实现了xxxx, 主要包括的功能有xxx。

\* 说明：使用该类，需要先初始化xxx,同时用户需要调用xxx以释放资源。

\*/

### 方法文档注释

方法文档注释描述该类方法实现的基本功能及采用的算法等。同时，对方法的输入、输出及异常等进行说明。另外，如果该方法分配了资源没有释放或者对类成员数据结构进行修改，线程安全等方面，也需要在文档中进行描述。该注释可以自动生成javadoc文档。

下列情况可以不用方法文档注释：

* getter/setter方法
* 私有方法（私有方法如果要加注释的话，可以用“//”行注释）
* 实现了接口中方法的方法（因为接口方法中已经有注释了）

### 方法内部注释

* 功能相对独立的代码片段（代码片段的行数以不超过10行为佳，一行也可以）前面需要进行注释，描述该代码片段的功能；
* 方法内部变量的注释请写在变量的后面；

### 代码修改注释

我们在修改别人开发的不太熟悉的而且比较重要的代码的时候，请不要直接修改代码，免得代码本人要看代码的时候不知道做了什么修改。尽管我们有版本管理工具可以查看历史代码，但是毕竟不太方便，所以这时候请为修改的代码加上注释。注释示例如下：

//2011.1.17 by liufy insert start 修改原因

//...插入的代码

//2011.1.17 by liufy insert end

//2011.1.17 by liufy delete start 修改原因

//注释掉的原代码

//2011.1.17 by liufy delete end

//2011.1.17 by liufy update start 修改原因

//注释掉的原代码

//自己插入的新代码

//2011.1.17 by liufy update end

## 包引入语句声明

包引入语句指定该类所中依赖的外部类。需要注意：

* 仅仅引入本类依赖的类，对于本类不需要的类绝对不能引入；
* 引入的类需要含包名路径的全名称限定，不能用**\***，如*java.io.\*;*
* 对JDK默认引入的类,绝不要额外声明，如*java.lang.Integer*;
* 引入的所有类按照包路径名进行分类，同时JDK系统类、第三方类引入声明在前面，项目/模块自定义的类引入声明在后面。

例如：

import java.io.InputStream;

import java.io.OutputStream;

import java.io.IOException;

import java.net.Socket;

import java.util.Enumeration;

**技巧：**使用eclipse修改java代码的时候，在每次代码保存之前，习惯性的按一下【ctrl + shift + o】快捷键，这样既可以删除不必要的引用，又可以自动按照上述规范进行格式化。

## 空白

### 空行

空行将逻辑相关的代码段分隔开，以提高可读性。

下列情况应该总是使用一个空行：

* 两个方法之间
* 方法内的局部变量和方法的第一条语句之间
* 块注释或单行注释之前
* 一个方法内的两个逻辑段之间，用以提高可读性
* 类的成员变量之间请不要留空行，但是逻辑上独立的变量组之间可以有一行空行

**注意：**空行一定不要随便乱加，因为空行是为了提高代码的逻辑性、可读性的，太多的空行会是代码显得不紧凑，没有逻辑性，可读性降低。

### 空格

下列情况应该使用空格：

* 一个紧跟着括号的关键字应该被空格分开，例如：

**while** (**true**) {

//...

}

**注意：**空格不应该置于方法名与其左括号之间。这将有助于区分关键字和方法调用。

* 方法参数的逗号后面和下一个参数之间要有一个空格
* 所有的二元运算符，除了"."，应该使用空格将之与操作数分开。一元操作符和操作数之间不应该加空格，比如：负号("-")、自增("++")和自减("--")。例如：

**int** a = c + d;

a = (a + b) / (c \* d);

**while** (d++ == s++) {

n++;

}

* for 语句中的表达式应该被空格分开，例如：

**for** (expr1; expr2; expr3)

* 强制转型后应该跟一个空格，例如：

myMethod((byte) aNum, (Object) x);

myMethod((int) (cp + 5), ((int) (i + 3)) + 1);

## 代码缩进

### 行长度(Line Length)

尽量避免一行的长度超过 80个字符（这个规则可以适量放松，但是最长别超过100个字符，再长的话就很难读了），因为很多终端和工具不能很好处理之。

注意：用于文档中的例子应该使用更短的行长，长度一般不超过 70个字符。

### 换行(Wrapping Lines)

当一个表达式无法容纳在一行内时，可以依据如下一下规则断开：

* 在一个逗号后面断开
* 在一个操作符前面断开
* 宁可选择较高级别(higher-level)的断开，而非较低级别(lower-level)的断开
* 新的一行应该与上一行同一级别表达式的开头处对齐
* 如果以上规则导致你的代码混乱或者使你的代码都堆挤在右边，那就代之以缩进8个空格。

**技巧：**【2.4.1】和【2.5】两节的代码规范可以通过eclipse的【ctrl + shift + f】快捷键来很方便的进行格式化。通过该快捷键可以对部分代码或者整个java文件进行格式化。

## 变量声明

* 一行只声明一个变量，因为这样以利于写注释；
* 尽量在声明局部变量的同时初始化；
* 只在代码块的开始处声明变量（一个块是指任何被包含在大括号"{"和"}"中间的代码。），不要在首次用到该变量时才声明之，这会把注意力不集中的程序员搞糊涂，同时会妨碍代码在该作用域内的可移植性；

例如：

**void** myMethod() {

**int** int1 = 0; // 方法代码块的开始

**if** (**true**) {

**int** int2 = 0; // "if"代码块的开始

//...

}

}

例外：

**for** (**int** i = 0; i < maxLoops; i++) {

//...

}

## 语句

* 请不要省略代码块的“{”和“}”，并且“{”前不要换行

**正确：**

**if** (**true**) {

**return** 0;

}

**错误：**

1. **if** (**true**)

**return** 0;

1. **if** (**true**) **return** 0;
2. **if** (**true**)

{

**return** 0;

}

## 其它

### 泛型

对于集合框架类，我们必须严格指定集合元素的类型。例如：

List<String> names = **new** ArrayList<String>();

Map<String, Integer> map = **new** HashMap<String, Integer>();

如果，集合对象的元素是混合元素（如String, Integer, Float等），可以将元素类型指定为Object。例如：

Map<Object, Object> map = **new** HashMap< Object, Object >();

### 嵌套

为了提高程序的可读性，每个方法的嵌套层数不要超过3层。

减少嵌套的常用方法有：

* IF语句改造

1. **改造前：**

**if** (condition1()) {

//...异常处理

} **else** {

//...正常处理

}

**return**;

**改造后：**

**if** (condition1()) {

//...异常处理

**return**;

}

//...正常处理

**return**;

1. **改造前：**

**if** (condition1()) {

**if** (condition2()) {

//...

}

}

**return**;

**改造后：**

**if** (condition1()) {

**return**;

}

**if** (condition2()) {

**return**;

}

//...

**return**;

* 将嵌套的代码提取成一个方法

### 代码行数

* 方法的代码行数越少越好，每个方法的最大代码行数不要超过100行
* 每个java文件的最大代码行数不要超过2000行

# 命名规范

## 包命名

* 包名全部小写
* 包名必须是范围、分层或功能性的名词，com.comp.usercenter.action
* 系统包名详细规范：【根包名】 + 【项目组ID】 +【子系统ID】 + 【分层ID】 + 【模块ID（可选）】 + 【内部细分（可选）】

举例如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **根包名** | **子系统ID** | **分层ID** | **模块ID** | **内部细分** |
| com.comp.bf.usercenter | admin | action | mp | vo |
| service | vo |
| content | vo |
| … | vo |
| bo | mp | impl |
| service | impl |
| content | impl |
| … | impl |
| persistence | dao | mp | impl |
| service | impl |
| content | impl |
| … | impl |
| po | mp |  |
| service |  |
| content |  |
| … |  |

## 类和接口命名

* 必须是一个有意义的英文名词，做到见名知其意
* 尽量少用或者不用缩写，但是一些熟知的缩写（如URL、WAP、DB等）和**《数据词典.xlsx》**文档中规定的缩写必须使用
* 采用大小写混合的方式，首字母必须大写，类名中不能出现“\_”、“-”等字符
* 分层下的类或接口必须以“分层ID”结尾。例如：ContentPO、ContentDAO、ContentBO、ContentAddAction
* “vo”包中的类必须以“VO”结尾；“impl”包中的类必须在实现的接口后面加上“Impl”。例如：ContentListVO、ContentBOImpl
* 类或接口名中不推荐出现全部大写的缩写（除了本规范中已经规定好的），也应该改成首字母大写的方式。例如：MpCpPO、WapDownloadAction等

## 变量（含参数）或常量命名

* 所有变量或常量名必须是名词，要能够见名知意。**《数据词典.xlsx》**文档中有的名词，则必须使用该文档中规定的名词进行缩写
* 所有的常量必須大写，如果含多个名词中间用下划线（‘\_’）隔开， （如ERR\_OK, ERR\_CONNECT, ERR\_PERMISSION, BUF\_SIZE）
* 变量的首字母必须小写，如果包含多个单词，则第二个（含）以后的每个单词首字母必须大写
* 变量中不能含有下划线（‘\_’）等别的符号分割，包括PO中与数据库字段对用的变量中也不能有

## 方法命名

* 方法名应该是一个动词或动名词，要能够见名知意。如login、 verifyUser、 checkOrder、 register、 doFinished、 processRequest、 handle、 notifySuccessed、generateAuthCode等
* 所有的方法名首字母必须小写，如果包含多个单词，则第二个（含）以后的每个单词首字母必须大写
* 方法命中不能含有下划线（‘\_’）等别的符号分割

### 数据访问类（DAO）方法命名

下面是采用Hibernate作为持久层框架时的方法名命名规范：

* 插入或更新方法

命名规范：“save”

举例：save(MpChPO mpChPo)

* 删除方法

命名规范：“delete”

举例：delete(MpChPO mpChPo)

* 主键加载（使用二级缓存，如果数据不存在，则抛出异常）

命名规范：“loadByID”

举例：loadByID(MpChPO mpChPo)

* 主键查询（不使用二级缓存，如果数据不存在，则返回null）

命名规范：“getByID”

举例：getByID(MpChPO mpChPo)

* 全表查询

命名规范：“getAll”

* 检索多条数据

命名规范：“getList” [+ “4” + 【用途】]

* 检索分页数据

命名规范：“getList4Page”

* 检索记录条数

命名规范：“getCount”

### Action层方法命名规范

* 页面显示

命名规范：“open”

* 查询

命名规范：“doSearch”

* 按钮动作

命名规范：“do” + 【按钮功能】

## 项目命名规范

### 项目名

命名规范：【项目组ID】 + “-” + 【项目ID（全小写）】

参考：项目组ID和项目ID参考《搜索项目列表.xlsx》

### 发布包名

命名规范：“pg-” + 【项目组ID】 + “-” + 【项目ID】 + “\_” +【版本号】

举例：pg-bf-usercenter\_1.0.1.war

### 项目目录结构

表格 3.5‑2 项目目录结构举例

|  |  |  |  |
| --- | --- | --- | --- |
| 项目组ID | 项目ID | 子文件夹 | 说明 |
| bf | usercenter | branch | 项目分支代码存放文件夹 |
| tag | 项目发布包存放文件夹 |
| trunk | 项目主分支代码存放文件夹，trunk下面是maven项目 |

# 附录1. JAVA源程序样本

待补充