Java™ 编程语言代码规范

1999 年 **4** 月 **20** 日修订版

译：Johnson.Lee

目　　录

1 引言 4
1.1 为什么有代码规范 4
1.2 致谢 4
2 文件名 4
2.1 文件后缀 4
2.2 常用文件名 5
3 文件结构 5
3.1 Java 源文件 5
3.1.1 头注释 5
3.1.2 Package 和 Import 语句 6
3.1.3 类和接口的声明 6
4 缩进 6
4.1 行长 7
4.2 换行 7
5 注释 8
5.1 注释格式实现 9
5.1.1 块注释 9
5.1.2 单行注释 10
5.1.3 尾端注释 10
5.1.4 行末注释 10
5.2 文档注释 11
6 声明 12
6.1 每行声明的数量 12
6.2 初始化 12
6.3 布局 12
6.4 类和接口的声明 13
7 语句 14
7.1 简单语句 14
7.2 复合语句 14
7.3 return 语句 14
7.4 if, if-else, if else-if else 语句 14
7.5 for 语句 15
7.6 while 语句 15
7.7 do-while 语句 16
7.8 switch 语句 16
7.9 try-catch 语句 16
8 空白 17
8.1 空行 17
8.2 空格 17
9 命名约定 18
10 编程实践 19
10.1 提供对实例和类变量的访问 19

10.2 关于类变量和方法 19
10.3 常量 20
10.4 变量赋值 20
10.5 综合练习 20
10.5.1 圆括号 20
10.5.2 返回值 21
10.5.3 条件运算符中‘？’前的表达式 21
10.5.4 特殊的注释 21
11 代码示例 22
11.1 Java 源文件示例 22
附录Ⅰ 版权公告 23

1 引言

1.1 为什么有代码规范

代码规范对程序员很重要的一些原因：

- 一个软件得花费其生命周期的 80%去维护。
- 几乎没有任何软件在整个生命周期中是由其原作者维护的。
- 代码规范提高了软件的可读性，能够让工程师更快、更透彻的了解新的代码。
- 如果你将你的源代码作为产品发布，你需要确保它和你创建的其它产品一样被很好的包装和清洁。

1.2 致谢

此文档反映了源于 Sun Microsystems, Inc. 《Java 语言规范》中描述的 Java 语言编码标准。主要的贡献来自 Peter King, Patrick Naughton, Mike DeMoney, Jonni Kanerva, Kathy Walrath, 和 Scott Hommel。

对此文档的意见请提交给我们的反馈表单。

2 文件名

本节列出了常用的文件名后缀和名称。

2.1 文件后缀

Java 软件使用下列文件后缀：

文件类型	后缀
Java 源文件	.java
Java 字节码文件	.class

2.2 常用文件名

经常使用的文件名包括：

文件名	使用
GNUmakefile	makefile 文件的首选的名称。我们使用 gnumake 构建我们的软件。
README	概述特定的目录内容首选的文件名称

3 文件结构

组成文件的章节应当用空行隔开，而且有一个可选的注释标识每节。
超过 2000 行的文件是非常笨重的，应该避免这种情况。
Java 程序正确格式化的例子，参见 "Java 源文件示例"。

3.1 Java 源文件

每个 Java 源文件包含一个公共类或接口。当私有类和接口与公共类相关时，你可以将它们作为公共类放在同一个文件中。公共类应该是文件中的第一个类或接口。
Java 源文件具有以下顺序：

- 头注释 (参见 "头注释")
- Package 和 Import 语句
- 类和接口声明 (参见 "类和接口的声明")

3.1.1 头注释

所有源文件应该以 c 语言风格的注释开头，列出类的名称、版本信息、日期和版权公告：

```
/*
 * Classname
 *
 * Version information
 *
 * Date
 *
 * Copyright notice
 */
```

3.1.2 Package 和 Import 语句

在大多数 Java 源文件中，第一个非注释行是 package 语句。之后可以是 import，例如：

```
package java.awt;
import java.awt.peer.CanvasPeer;
```

注意：一个独特的包名的首部分应该使用全小写的 ASCII 字符而且应该是现有的顶级域名 com, edu, gov, mil, net, org 之一或是 1981 年的 ISO 标准 3166 中的英文两字母代码标识的国家代码之一。

3.1.3 类和接口的声明

以下表格描述了类和接口的声明部分。他们应该按此顺序出现。参见"Java 源文件示例" 第 19 页的例子，其中包含注释。

	类/接口的声明部分	说明
1	类/接口 文档注释(/**...*/)	参见 "文档注释"
2	class 或 interface 语句	
3	类/接口的实现 注释 (/*...*/), 如果有必要的话	此注释应该包含任何类/接口全范围的信息，是不恰当的类/接口文档注释
4	类的 (static) 变量	首先是 public 的类变量，接着是 protected 的，然后是包级别的（无访问修饰符的），再是 private 的。
5	实例变量	首先是 public 的类变量，接着是 protected 的，然后是包级别的（无访问修饰符的），再是 private 的。
6	构造器	
7	方法	这些方法应该按功能分组，而不是范围或可访问性，例如：一个私有的类方法可以在两个公共的实例方法之间，这样做的目的是为了让阅读和理解代码更容易。

4 缩进

缩进应该以四个空格为单位。对于缩进的准确解释(空格对制表符)未特别指出。必须正确

地设置每 8 个空格为一个制表符而不是 4 个。

4.1 行长

避免行的长度超过 80 个字符，因为很多终端和工具不能很好的处理它们。

注意：文档中使用的示例应该有一个其长度一般不超过 70 个字符的短行。

4.2 换行

当一个表达式不适合在一行时，断行一般根据这些常规原则：

- 在逗号后断行。
- 在操作符前断行。
- 宁可选择高层次断行，而不选择低层次。
- 将新行与表达式同一层次的上一行开头对齐。
- 如果上面的规则会导致混淆代码或代码右边距挤压，只需缩进 8 个空格代替。

以下是一些方法调用断行的例子：

```
someMethod(longExpression1, longExpression2, longExpression3,
       longExpression4, longExpression5);

var = someMethod1(longExpression1,
              someMethod2(longExpression2,
                    longExpression3));
```

以下是两个数字表达式断行的例子，当断行发生在括号表达式以外时，优先选择第一种在一个较高层次上的断行。

```
longName1 = longName2 * (longName3 + longName4 - longName5)
          + 4 * longname6; // 首选

longName1 = longName2 * (longName3 + longName4
                    - longName5) + 4 * longname6; // 避免
```

以下是两个方法声明缩进的示例。第一种是规范的实例。如果采用规范的缩进，第二种会把第二行和第三行移到最右边，所以反而只缩进 8 个空格。

```
//规范的缩进
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
         Object andStillAnother) {
    ...
}
```

```
//为了避免过深的缩进，只缩进 8 个空格
private static synchronized horkingLongMethodName(int anArg,
        Object anotherArg, String yetAnotherArg,
        Object andStillAnother) {
    ...
}
```

对于 if 语句，当规范的（4 个空格）缩进使得主体看起来很困难时，一般使用 8 个空格原则。例如：

```
//不要使用这种缩进
if ((condition1 && condition2)
    || (condition3 && condition4)
    ||!(condition5 && condition6)) { //差的换行
    doSomethingAboutIt();             //容易将此行错过
}

//使用这种缩进代替
if ((condition1 && condition2)
        || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}

//或者使用这种
if ((condition1 && condition2) || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}
```

以下有三种可以接受的方式进行格式化三元表达式：

```
alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongBooleanExpression) ? beta
                                 : gamma;
alpha = (aLongBooleanExpression)
        ? beta
        : gamma;
```

5 注释

Java 程序可以有两种类型的注释：实现注释和文档注释。实现注释在 C++里可以发现，由 /*...*/ 或 // 限定。文档注释（称为 "doc 注释"）为 Java 特有，由/**...*/限

定。使用 javadoc 工具可以将文档注释提取为 HTML 文件。

实现注释意味着注释掉代码或对特定的实现加以注释。文档注释是用来描述代码的规范。供手上也许没有必要拥有源代码的开发者阅读。

注释应该被用来概述代码和提供在代码里不能立竿见影的额外信息。注释应该仅包含与阅读和理解程序有关的信息。例如，关于相应的包是如何构建或者在哪个目录的信息不应该包含在注释中。

讨论平凡或差异性的设计决策是恰当的，但要避免重复的信息出现在（并清除）的代码中。这是很容易让冗余的注释过时。一般情况下，避免任何注释随着代码的改进而过时。

注意：注释的频率有时候反映了代码的质量较差。当你感到有必要添加注释的时候，考虑重写代码使之更清晰。

注释不应该括在用星号或其它字符画的大框内。
注释不应该包含特殊字符，如换页和退格符。

5.1 注释格式实现

程序可以有四种风格的实现注释：块注释、单行注释、尾端注释和行末注释

5.1.1 块注释

块注释用于提供对文件、方法数据结构和算法的描述。块注释可以用在每个文件的开头和方法之前，也可以用在其它的地方，比如方法里面，函数或方法里的块注释应该和它所描述的代码的缩进层次相同。

在块注释之前，应该用一空行将它和其余的代码隔开。

```
/*
 * 这里是一个块注释.
 */
```

块注释可以以 /*- 开始，作为根据缩进来识别的块注释的开头，不应该被重新格式化。例如：

```
/*-
 * 这里是一个具有一些比较特别的格式的块注释，
 * 我想用缩进(1)来忽略。
 *
 *    one
 *       two
 *          three
 */
```

注意： 如果你不使用缩进，你不必在你的代码里使用 /*- 或其它任何让步的可能性，别人可能会缩进你的代码。
参见 "文档注释"。

5.1.2 单行注释

简短的注释可以出现在和代码缩进同一层次的后面的单行中。如果注释不能写在一行，应该使用块注释格式。（参见 第 5.1.1 节）. 当选注释前应该有一空行。下面是 Java 代码的单行注释示例（另请参见 "文档注释"）:

```
if (condition) {
    /* 处理条件 */
    ...
}
```

5.1.3 尾端注释

很短的注释可以出现在它们所描述的代码所在的行。但是应该把它们移得足够远以和语句隔开。如果不只一个短注释出现在代码块中，应该和代码一样缩进。

以下是 Java 代码尾注释的示例：

```
if (a == 2) {
    return TRUE;            /* 特殊情况 */
} else {
    return isPrime(a);      /* 仅仅适用于奇数 */
}
```

5.1.4 行末注释

注释限定符 // 能够注释掉一整行或是一行的某部分。尽管行末注释能够注释连续的多行文本，但不应该用于注释连续的代码行；以下是三种风格的示例：

```
    if (foo > 1) {
        // Do a double-flip.
        ...
    }
    else {
        return false;          // Explain why here.
    }
    //if (bar > 1) {
```

```
//    // Do a triple-flip.
//    ...
//}
//else {
//    return false;
//}
```

5.2 文档注释

注意：参见 "Java 源文件示例" 中对注释格式的描述。

详情，参见"How to Write Doc Comments for Javadoc"，这里面包含了有关文档注释标签(@return, @param, @see)的信息：

http://java.sun.com/javadoc/writingdoccomments

关于文档注释和 javadoc 的详情，参见 javadoc 的主页：

http://java.sun.com/javadoc/

文档注释是用来描述 Java 类、接口、构造器、方法和字段。每个文档注释都被包含在注释限定符/**...*/之内，每个类 、接口或类成员一个注释。 这种注释应该出现在声明之前：

```
/**
 * The Example class provides ...
 */
public class Example { ...
```

注意最上层的类和接口不要缩进，只有它们作为成员时才缩进。类和接口的文档注释的第一行(/**)不缩进；随后的注释行都有一个空格的缩进(为了让星号垂直对齐)。成员，包括构造器，第一行注释有四个空格，随后的注释行有五个空格。

如果你需要提供有关类、接口、变量或方法的信息，采用文档注释不太合适，直接在声明后面使用块注释，（参见第 5.1.1 节）或单行注释（参见第 5.1.2 节）。 例如，关于一个类的实现细节应该在一个实现块注释的类声明之后，而不是在类的文档注释中。

文档注释不应该定位在一个构造器或方法的定义块内，因为 Java 将文档注释与注释后第一个声明关联的。

6 声明

6.1 每行声明的数量

建议每行一个声明，因为它支持注释，换句话说，

```
int level; // indentation level
int size;  // size of table
```

与下面声明相比，宁愿选择上面的声明方式，

```
int level, size;
```

不要把不同的放在同一行。例如：

```
int foo, fooarray[]; //WRONG!
```

注意：以上的例子中，在类型和标识符之间有一个空格。另一种可接受的选择是使用水平制表符，例如：

```
int    level;           // indentation level
int    size;            // size of table
Object currentEntry;    // currently selected table entry
```

6.2 初始化

尝试在声明局部变量时，初始化它们，唯一不初始化局部变量的理由是局部变量的初始值首先取决于一些计算。

6.3 布局

块的开头只放声明。(块是由大括号包围的任何代码)。不要等要用到的时候才声明变量。它可以混淆粗心的程序员而且在一定范围内阻碍了代码的可移植性。

```
void myMethod() {
    int int1 = 0;         // beginning of method block

    if (condition) {
        int int2 = 0;     // beginning of "if" block
        ...
```

```
    }
}
```

一个例个的规则是 for 循环索引，可以在 Java 中如下声明：

```
for (int i = 0; i < maxLoops; i++) { ... }
```

避免局部声明隐藏更高级别的声明。例如，不要在块内声明相同的变量名：

```
int count;
...
myMethod() {
    if (condition) {
    int count = 0;     // AVOID!
        ...
    }
    ...
}
```

6.4 类和接口的声明

当编写 Java 类和接口时，应该遵循下面的格式规范：

- 方法名和其参数列表起始的括号“(”之间没有空格
- 起始大括号“{”作为声明语句出现在与声明同一行的末尾
- 结束大括号“}”自起一行缩进到与其起始语句相匹配，除非是空语句，那么“}”应该紧跟在“{”之后。

```
class Sample extends Object {
    int ivar1;
    int ivar2;

    Sample(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int emptyMethod() {}

    ...
}
```

- 方法之间用一空行隔开

7 语句

7.1 简单语句

每行最多包含一条语句。例如：

```
argv++;       // Correct
argc--;       // Correct
argv++; argc--;     // AVOID!
```

7.2 复合语句

复合语句是一些包含一系列包在大括号“`{ statements }`”内的语句。见如下示例：

- 被括起的语句比复合语句缩进一个以上层次。
- 左大括号应该在复合语句开始行的末尾；右大括号应该另起一行并缩进到复合语句起始的位置。
- 大括号用于括起所有的语句，即使是一条语句，当它们作为控制结构的一部分时，例如 `if-else` 或 `for` 语句。这使得在忘记加括号的情况下增加语句而不引入意外的错误更容易。

7.3 return 语句

一个有值的 `return` 语句不应该使用小括号，除非能使其返回值在某些方面更突出。例如：

```
return;

return myDisk.size();

return (size ? size : defaultSize);
```

7.4 if, if-else, if else-if else 语句

`if-else` 语句类应该有如下形式：

```
if (condition) {
    statements;
}
```

```
if (condition) {
    statements;
} else {
    statements;
}

if (condition) {
    statements;
} else if (condition) {
    statements;
} else {
    statements;
}
```

注意：if 语句总是使用大括号 {}。避免如下易错的形式：

```
if (condition) //AVOID! THIS OMITS THE BRACES {}!
    statement;
```

7.5 for 语句

一个 for 语句应该有如下形式：

```
for (initialization; condition; update) {
    statements;
}
```

当在 for 语句的初始化或更新从句中使用逗号时，避免使用三个以上变量的复杂情况。如果有必要，在 for 循环之前（为初始化从句）或循环之后（为更新从句）使用分开的语句。

7.6 while 语句

一个 while 语句应该有如下形式：

```
while (condition) {
    statements;
}
```

一个空 while 语句应该有如下形式：

```
while (condition);
```

7.7 do-while 语句

一个 do-while 语句应该有如下形式：

```
do {
    statements;
} while (condition);
```

7.8 switch 语句

一个 switch 语句应该有如下形式：

```
switch (condition) {
case ABC:
    statements;
    /* falls through */

case DEF:
    statements;
    break;

case XYZ:
    statements;
    break;

default:
    statements;
    break;
}
```

每当一个 case 穿透（没有包含 break 语句），通常在 break 语句处增加一段注释。例如使用/* falls through */注释。

每个 switch 语句应该包含一个 default case.default case 中的 break 是多余的，但是如果以后增加了另一个 case 它会预防 case 穿透的错误。

7.9 try-catch 语句

一个 try-catch 语句应该有如下格式：

```
try {
    statements;
} catch (ExceptionClass e) {
```

```
    statements;
}
```

一个try-catch语句后可能跟着finally，无论try语句块有没有成功完成它都会执行。

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
} finally {
    statements;
}
```

8 空白

8.1 空行

空行通过隔断逻辑上相关的代码，增强了代码的可读性。

如下情况应该使用两个空行：

- 源文件中节与节之间
- 类与接口定义之间

如下情况应该使用一个空行：

- 方法之间
- 方法内的局部变量与方法中的第一个语句之间
- 块注释或单行注释之前
- 方法内的逻辑部分之间为了提高可读性

8.2 空格

如下情况应该使用空格：

- 关键词后的小括号应该用一个空格隔开。例如：

  ```
  while (true) {
      ...
  }
  ```

  注意空格不应该用在方法名和其左小括号之间。这样可以帮助区分关键字和方法调

用。

- 参数列表的逗号后应该有空格。
- 除二进制操作符外的所有操作符都应该用空格将其与操作数隔开。一元运算符绝不应用空格将其与操作数隔开，比如一元减号、自增符（“++”）和自减符（“—”）。例如：

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

while (d++ = s++) {
    n++;
}
printSize("size is " + foo + "\n");
```

- for 语句中的表达式应该用空格隔开，例如：

```
for (expr1; expr2; expr3)
```

- 强制转换后应该紧跟一个空格。例如：

```
myMethod((byte) aNum, (Object) x);
myMethod((int) (cp + 5), ((int) (i + 3)) + 1);
```

9 命名规范

命名规范通过增强程序的可读性，使用程序更容易理解。它还可以提供有关标识符的功能信息，例如，无论是常量、包或类都能对理解代码很有帮助。

标识符类型	命名规则	示例
包	一个唯一的包名前缀总是以小写 ASCII 字母，而且是顶级域名之一，目前有 com, edu, gov, mil, net, org 或是 ISO 标准 3166, 1981 中的英文两字母国家标识码。 包名的后续部分根据不同组织的内部命名约定。这些规范可能指定某些目录名称组件是子公司、部门、项目、设备或登录名。	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese
类	类名应该是名词，每个内部单词首字母大写。尽量保持类名简单且具描述性，尽量避免缩略语和简称（除非缩写比较长的形式更为广泛的使用，例如 URL	class Raster; class ImageSprite;

	或 HTML）	
接口	接口名应和类名一样大写。	`interface RasterDelegate;` `interface Storing;`

方法	方法应该是动词，第一个字母小写，每个内部单词的首字母大写。	`run();`
		`runFast();`
		`getBackground();`

变量	除变量外，所有实例、类及类的常量均采用第一个字母小写，内部单词的首字母大写。变量名不应以下划线或美元符开头，尽管它们是被允许的。
	变量名应该简短且有意义。变量名的选择应该便于记忆，并设计得能顾名思义。除临时 “一次性”的变量外，避免用一个字符的变量名。临时变量通用的名称是整型的 i、j、k、m 和 n；字符型的 c、d、e。
		`int i;`
		`char c;`
		`float myWidth;`

常量	类的常量及 ANSI 常量的变量名声明应该全部大写，单词之间用下划线（“_”）隔开。（为了方便调试，应避免 ANSI 常量。）	`static final int MIN_WIDTH = 4;`
		`static final int MAX_WIDTH = 999;`
		`static final int GET_THE_CPU = 1;`

10 编程实践

10.1 提供对实例和类变量的访问

没有很好的理由，不要让实例或类变量为 public。通常情况下，实例变量并不需要明确设置或类似方法调用的形式频繁获取。

10.2 关于类变量和方法

避免使用对象去访问一个类（静态）的变量或方法，应使用类名来访问。例如：

```
classMethod();          //OK
AClass.classMethod();   //OK
anObject.classMethod(); //AVOID!
```

10.3 常量

数值常量不应该直接编码，除了-1、0 和 1，它们能够作为 for 循环的计数器。

10.4 变量赋值

避免在一条语句中为多个变量赋同一个值，这样读起来很困难。例如：

```
fooBar.fChar = barFoo.lchar = 'c';   //AVOID!
```

不要在易与等号运算混淆的地方使用赋值运算符。例如：

```
if (c++ = d++) { // AVOID! (Java disallows)
    ...
}
```

应该这样写：

```
if ((c++ = d++) != 0) {
    ...
}
```

不要企图使用嵌入的赋值语句改善运行时性能，这是编译器的事情。例如：

```
d = (a = b + c) + r;    // AVOID!
```

应该这样写：

```
a = b + c;
d = a + r;
```

10.5 综合练习

10.5.1 圆括号

在包含有混合运算符的表达式中自由的使用括号避免操作符优先级问题通常是一个好主意。即使运算符的优先级你看起来很清楚，但你不应该假定其他程序员像你一样了解其优先级。

```
if (a == b && c == d)      // AVOID!
if ((a == b) && (c == d))  // RIGHT
```

10.5.2 返回值

尽量使你的程序结构迎合你的意图，例如：

```
if (booleanExpression) {
    return true;
} else {
    return false;
}
```

应该这样写：

```
return booleanExpression;
```

同样的写法，

```
if (condition) {
    return x;
}
return y;
```

应该这样写：

```
return (condition ? x : y);
```

10.5.3 条件运算符中‘？’前的表达式

如果包含二元运算符的表达式出现在三元运算符的 ? 前面，应该用括号将它括起来。例如：

```
(x >= 0) ? x : -x;
```

10.5.4 特殊的注释

在注释中使用 XXX 标识一些东西虽然是臆想的，但是也凑效。使用 FIXME 来标识一些东西虽然是臆想的，但有点蹩脚。

11 代码示例

11.1 Java 源文件示例

下面的示例演示了如何格式化包含单一的公共类的 Java 源文件。接口的格式化与类相同。有关更多信息，请参见“类和接口的声明和文档注释”

```
/*
 * @(#)Blah.java       1.82 99/03/18
 *
 * Copyright (c) 1994-1999 Sun Microsystems, Inc.
 * 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information of
 * Sun  Microsystems, Inc. ("Confidential Information"). You shall not
 * disclose such Confidential Information and shall use it only in
 * accordance with the terms of the license agreement you entered into
 * with Sun.
 */


package java.blah;

import java.blah.blahdy.BlahBlah;

/**
 * Class description goes here.
 *
 * @version   1.82 18 Mar 1999
 * @author    Firstname Lastname
 */
public class Blah extends SomeClass {
    /* A class implementation comment can go here. */

    /** classVar1 documentation comment */
    public static int classVar1;

    /**
     * classVar2 documentation comment that happens to be
     * more than one line long
     */
```

```
    private static Object classVar2;

    /** instanceVar1 documentation comment */
    public Object instanceVar1;

    /** instanceVar2 documentation comment */
    protected int instanceVar2;

    /** instanceVar3 documentation comment */
    private Object[] instanceVar3;

    /**
     * ...constructor Blah documentation comment...
     */
    public Blah() {
        // ...implementation goes here...
    }

    /**
     * ...method doSomething documentation comment...
     */
    public void doSomething() {
        // ...implementation goes here...
    }

     /**
      * ...method doSomethingElse documentation comment...
      * @param someParam description
      */
    public void doSomethingElse(Object someParam) {
        // ...implementation goes here...
    }
}
```

附录 I　版权公告

如果您修改的方式改变此文档的含义，例如，以符合自己公司的编码规范，请使用此归属行：

改编自《JAVA™ 编程语言代码规范》。Copyright 1995-1999 Sun Microsysytems, Inc. All rights reserved.

无论哪种方式，请包含超文本链接或其它参考 Java 代码规范网站：

http://java.sun.com/docs/codeconv/