Version：1.0 **Document Type:制度**

网易 NetEase

Java 编码规范

设计人：

负责人：

审批：

生效日期：

文档控制概要

文档负责人负责将文档(新文档/修改/更新文档)分发给所有审批者进行审阅。只有文档作者和负责人有权更新/修改文档，一切修改请求必须交给文档作者或负责人。

版本变更纪录

日期	版本	说明	草拟/修改
2005.10.20	V1.0	Java 编码规范	IT 内控小组

目录

Java 编码规范

1.0 目的

提高开发效率。

提高源程序的可读性，减少后期维护工作量。

2.0 适用范围

本文档颁布实施后开始的各项目中的所有 Java 程序。

3.0 说明

本规范总则的内容包括：排版、命名规范、注释、控制流程规则、声明、语句变量赋值、符号使用、编程惯例等。

基本要求：编程时强制必须遵守的原则。

建议要求：编程时必须加以考虑的原则。

4.0 基本要求

4.1 排版

4.1.1 包和引入语句(Package and Import Statements)

在多数 Java 源文件中，第一个非注释行是包语句。在它之后可以跟引入语句。例如：

package java.awt;

import java.awt.peer.CanvasPeer;

4.1.2 缩进排版(Indentation)

函数、结构、循环、判断等语句都需要采用缩进，4 个空格或一个 Tab 键常被作为缩进排版的一个单位。

4.1.3 行长度(Line Length)

尽量避免一行的长度超过 80 个字符。

4.2 控制流程规则

4.2.1 条件分支

1 分支的写法

分支语句在两行以上的时候请用{}括起来。

Eg:

```
if ( a == b ) a++;  //正确

if ( a == b ) {    //正确

    a++;     //正确

}          //正确

if ( a == b )  //不允许

    a++;  //不允
```

2 分支的条件表达式

不要在分支的条件表达式中使用赋值操作。

```
int  cnt = count.getCount();  //正确
if ( 1 == cnt ) {
    fileName = "zaiko1.txt";
} else {
    fileName = "zaiko2.txt";
}

int  cnt = 0;
if ( 1 == (cnt = count.getCount()) ) {//不允许
    fileName = "zaiko1.txt";
} else {
    fileName = "zaiko2.txt";
        }
```

3 其他注意点

总是将恒量放在等号/不等号的左边。

如果有用到 else if 语句，通常最好有一个 else 块以用于处理未处理到的其他情况。

4.2.2 Switch 分支

有多个 case 的时候，如果某个不需要 break 时请注释标明。

default case 总应该存在，它应该不被到达，然而如果到达了就会触发一个错误。

如果要创立一个变量，把所有的代码放在块中。

```
例：
switch (funcNo) {
  case 4:
    func4();
```

```
    break;
  case 3:
    func3();
      //No Break
    //这里不需要 break,case 3 需要执行 2 的内容
  case 2:
    {
      v = get_week_number();
      }
    break;
  case 1:
    func1();
    break;
      :
  default:

 }
```

4.2.3 循环

同 if 语句一样，两行以上循环体要用{}括起来。

5.0建议要求

5.1 命名规范

5.1.1 Package 的命名

Package 的名字应该都是由一个小写单词组成。

5.1.2 Class 的命名

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。

5.1.3 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。

5.1.4 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。

5.2 注释

5.2.1 开头注释

所有的源文件都应该在开头有一个注释，其中列出类名、功能、版本信息、日期、作者和版权声明：

```
/*
* 类名
* 功能
* 版本
* 日期
* 作者
* 版权
*/
```

如果对文件进行了修改，应该在文件头中说明修改目的、修改日期、修改人，并变更文件的版本信息；如果修改文件的一部分，则在文件中进行注释即可，并且标识出修改部分的起止位置

……

/*

* 修改目的

* 修改日期

* 修改人

* 版本

*/

……

修改起始

……

……

修改结束

……

5.2.2 类注释

在类定义行的前面一行做如下注释：

```
//*********************************************************
/**
```

```
 * 类的说明和类处理内容

 * @author 作者

 * @version     版本，修改日期，修改原因
*/
//*******************************************************
public class XXXXXXX implements XXX, YYY {
```

1 类的说明和类处理内容

对类的简单说明，并作简单功能描述。

2 作者

程序作者

3 版本，修改日期，修改原因

类的版本、修改日期和修改原因，中间用逗号隔开。

初始版本号 1.00,对于比较大的改动，版本每次增加 0.10，小的变更版本加 0.01。

日期取修改完成的日期。

修改原因要简明清晰。

4 例：

```
//*****************************************************************
/**
 * XXX 和 YYY 的实现子类，主要功能是 XXXXXXX
 *
 * @author Prophet
 * @version     1.00，2005.05.01，新建
 *          1.01，2005.05.06，修改 BUG:数据显示不正确
```

```
 *       1.10，2005.05.07，功能更改:数据处理流程修改
*/
//**************************************************************
public class XXXXXXX implements XXX, YYY {
```

5.2.3 方法注释

在方法前面的说明注释如下：

```
/**
 * 方法的简要说明和处理内容
 *
 * @param     参数名 参数说明
 * @return     返回值说明
 * @exception 调用这个方法要处理的异常
 */
```

1 方法的简要说明和处理内容

方法的简要说明

2 参数名 参数说明

参数名和参数说明，有多个要写多行，没有可以省略。

3 返回值

方法返回值的描述，没有可以省略。对返回值的意义有必要说明的时候要分多行说明。

4 异常

调用此方法时需要处理的异常。多个要写多行，没有可以省略。

5 例：

```
/**
 * 数字判断。(返回值说明的例子)
 *
 * @return 是不是数字 true:是<BR>
 *             false:不是
 */
public boolean isNumber() {

}

/**
 * 得到字符类型。（返回值说明的例子）
 *
 * @return 字符类型  0:数字<BR>
 *             1:英文字母<BR>
 *             2：其他
 */
public int getCharType() {

}
```

5.2.4 变量的注释

变量的前面要有如下注释:

```
/** 变量说明 */
```

5.3 声明

5.3.1 初始化

尽量在声明局部变量的同时初始化。变量的初始值依赖于某些先前发生的计算时可以不这样做。

5.3.2 布局

1 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。）

2 避免声明的局部变量覆盖上一级声明的变量。

5.3.3 每行声明的变量数量

1 推荐一行一个声明，因为这样以利于写注释。如：

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

要优于，

```
int level, size;
```

2 不要将不同类型变量的声明放在同一行，例如：

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

5.3.4 类的声明

当编写类时，应该遵守以下格式规则：

1 在函数名与其参数列表之前的左括号"("间不要有空格。

2 左大括号"{"位于声明语句同行的末尾。

3 右大括号"}"另起一行，与相应的声明语句对齐，除非是一个空语句，"}"应紧跟在"{"之后。

5.4 语句

5.4.1 简单语句

每行至多包含一条语句，例如：

argv++; // 正确的

argc--; // 正确的

argv++; argc--; // 错误的

5.4.2 复合语句

复合语句是包含在大括号中的语句序列，形如"{ 语句 }"。例如下面各段。

1 被括其中的语句应该较之复合语句缩进一个层次。

2 左大括号"{"应位于复合语句起始行的行尾；右大括号"}"应另起一行并与复合语句首行对齐。

3 大括号可以被用于所有语句，包括单个语句，只要这些语句是诸如 if-else 或 for 控制结构的一部分。

5.4.3 返回语句

一个带返回值的 return 语句不使用小括号"()"。

5.5 变量赋值

5.5.1 避免在一个语句中给多个变量赋相同的值。

```
fooBar.fChar = barFoo.lchar = 'c'; // 错误
```

5.5.2 不要将赋值运算符用在容易与相等关系运算符混淆的地方。

5.5.3 不要使用内嵌(embedded)赋值运算符试图提高运行时的效率，这是编译器的工作。

```
d = (a = b + c) + r; // 错误
```

应该写成

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

5.6 符号使用

5.6.1 圆括号

一般而言，应该在含有多种运算符的表达式中使用圆括号来避免运算符优先级问题。

```
if (a == b && c == d) // 错误

if ((a == b) && (c == d)) // 正确
```

如果一个包含二元运算符的表达式出现在三元运算符"？："的"?"之前，那么应该给表达式添上一对圆括号。例如：

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

5.6.2 大括号

有两种是可以接受的，如下的第一种是最好的：

1 将大括号放置在关键词下方的同列处：

```
if (condition)
{
    ...
    while (condition)
     {
                ...
    }
}
```

2 传统的 UNIX 的括号规则是，首括号与关键词同行，尾括号与关键字同列：

```
if (condition) {
    ...
```

```
        while (condition) {
        ...
        }
    }
```

5.7 编程惯例

5.7.1 换行

当一个表达式无法容纳在一行内时，可以依据如下一般规则断开之：

1 在一个逗号后面断开。

2 在一个操作符前面断开。

3 宁可选择较高级别(higher-level)的断开，而非较低级别(lower-level)的断开。

4 新的一行应该与上一行同一级别表达式的开头处对齐。

5 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。

✧ *以下是两个断开算术表达式的例子。前者更好，因为断开处位于括号表达式的外边，这是个较高级别的断开。*

```
longName1 = longName2 * (longName3 + longName4 - longName5)
                                + 4 * longname6; //使用这种缩进方式

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

✧ *以下是两个缩进方法声明的例子。前者是常规情形。后者若使用常规的缩进方式将会使第二行和第三行移得很靠右，所以代之以缩进 8 个空格*

```
//传统的缩进方式
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
                Object andStillAnother) {
  ...
 }
//利用 8 个连续空格避免过渡的缩进
private static synchronized horkingLongMethodName(int anArg,
        Object anotherArg, String yetAnotherArg,
        Object andStillAnother) {
  ...
 }
```

✧ if 语句的换行通常使用 8 个空格的规则。比如：

```
//不要使用这种缩进方式
if ((condition1 && condition2)
    || (condition3 && condition4)
        ||!(condition5 && condition6)) { //错误的换行方式，没有进行缩进
        doSomethingAboutIt(); //条件与此句对齐，造成阅读程序时很可能漏过此句
    }
//应该使用这种缩进方式
if ((condition1 && condition2)
        || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
```

```
    }
//或这样的缩进方式也可以
if ((condition1 && condition2) || (condition3 && condition4)
                    ||!(condition5 && condition6)) {
    doSomethingAboutIt();
    }
```

✧ 这里有三种可行的方法用于处理三元运算表达式：

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

5.7.2 空格

1 一个紧跟着括号的关键字应该被空格分开，例如：

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

2 空格不应该置于函数名与其左括号之间。这将有助于区分关键字和函数调用。

3 空白应该位于参数列表中逗号的后面。

4 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因该加空格，比如：负号("-")、自增("++")和自减("--")。例如：

```
a += c + d;
a = ( a + b ) / ( c * d );
while ( d++ = s++ ) {
    n++;
}
printSize( "size is " + foo + "\n" );
```

5 for 语句中的表达式应该被空格分开，例如：

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

6 强制转型后应该跟一个空格，例如：

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

5.7.3 空行

1 下列情况应该总是使用两个空行：

- ✓ 一个源文件的两个片段(section)之间
- ✓ 类声明声明之间

2 下列情况应该总是使用一个空行：

- ✓ 两个函数之间
- ✓ 函数内的局部变量和函数的第一条语句之间
- ✓ 块注释或单行注释之前
- ✓ 一个函数内的两个逻辑段之间，用以提高可读性

5.7.4 数组变量的定义规则

在类型后面加[]来定义数组。

Eg:

int[] a; //正确

int a[]; //不允许

5.7.5 类成员规则

1 类的字段必须在方法之前，内部类写在类的最后。

2 类的字段的顺序

- ✓ 修饰符的编写顺序为

  public(private protected) static final int(long double) value = 0;

- ✓ 常量最先定义。一般字段先写 public,其次 protected,默认,最后 private。
- ✓ 有多个修饰符的情况，从修饰符多的开始编码。

Eg:

```
public static final int ARRAY_MAX_LENGTH = 1000;
public static final int ARRAY_MIN_LENGTH = 100;
protected static final int ARRAY_DEFAULT_LENGTH = 500;

public static String[] accessArray = null;
```

public String bookName = null;

protected int arrayLength = 600;

int defaultValue = 0;

private String privateString = null;

3 类的方法的顺序

- ✓ 修饰符的编写顺序为

  public(private protected) static final int(long double) methodName() {}

- ✓ 构造器最先写。
- ✓ 其后是静态方法
- ✓ 一般方法先写 public(getter setter 在 public 区的最后),其次 protected,默认,最后 private。

5.7.6 变量定义的作用域

1 减少类成员变量，对只在某个方法中使用的变量，要把它定义成方法中的变量

2 在方法中定义的变量作用域要尽量小。

Eg:

```
for( int i = 0; i < 10; i ++ ) { //正确写法
}
int i; //不允许做法，这样 i 的作用域变大了。
for( i = 0; i < 10; i ++ ) { //不允许做法
}
```

3 避免无意义的作用域。

Eg:

```
{               //正确这个作用域有意义
    int i = 0;
    i++;
}       //正确这个作用域有意义
int i = 0;
{               //不允许这个作用域无意义
    i ++;
}       //不允许这个作用域无意义
```

5.8 项目编码标准

各项目组应根据各项目具体情况制定适合项目开发的统一的编码标准，但是对于基本要求中的各要素必须要做到。

6.0代码范例

```
/*
 * @(#)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...
    }
}
```