# JAVA基本编码规范

## 说明

### 声明

本文档内容描述JAVA基本编码规范，凡是在金鑫开发的JAVA程序必须按照此文档规定。

### 为什么要有编码规范

在现代软件开发中，维护工作会占用80%的时间，而且开发者和维护者通常不是同一个程序员，因此，制定统一的代码编码规范显得相当重要。

编码规范提高了软件源代码的可读性，使得软件开发人员能够更加快速和彻底地理解程序源代码。好的代码风格不仅会提高可读性，而且会使代码更健壮，更为重要的是在修改时不容易出错。作为一个软件产品，制定严格的编码规范并在编码过程中严格遵循这些规范就显得更加重要。

鉴于以上原因，特制定本编码规范。本规范适用于金鑫公司的Java语言开发项目。

### 目标

* 为来自不同的项目组或个人提供标准的代码格式。
* 增加易读性。
* 降低维护成本。

## 参考文档

* 《Code Conventions for the Java Programming Language》， Scott Hommel等， Sun Microsystems Inc；
* 《Java编码规范》；

《Java平台最“体贴”的工具——Javadoc》， 王鸿， 赛迪网， 2002-01-28。

## Java文件格式

Java文件的各个部分之间应当使用空行或注释分隔，且每个文件的行数应当尽量控制在2000行以内。

每个Java文件尽量做到只包含一个类或者接口（内部类除外）。

Java文件的书写顺序如下：

* 文件注释；
* 包声明和导入语句；
* 类和接口声明语句；

### 文件注释

Java源文件的文件注释应当包含源文件的文件名、创建者、创建日期和版权信息等信息。

/\*

\* 源文件的文件名

\* 创建者

\* 源文件创建日期

\*

\* 版权信息

\*/

例如：

/\*

\* Example.java

\* 创建者：陈兵

\* 创建日期：2012-04-12

\*

\* 版权所有(C) 2012-2015。金鑫电脑公司。

\* 保留所有权利。

\*/

此处的文档信息不会出现在Java doc文档中。

### 包声明和导入语句

Java源代码的第一个非注释语句应当为package语句。其后为import语句。Import语句中标准包名（例如：JDK的包名）应当在本地包名之前，并且按照字母顺序排列。Import语句应尽量使用完全限定的类名，而不要使用\*。例如：

package hotjava.net.stats;

import java.io.Socket;

import java.util.Observable;

import hotjava.util.Application;

### 类和接口声明语句

下面的表格定义了类和接口声明中各部分出现的顺序：

|  |  |  |
| --- | --- | --- |
| 顺序  /编号 | 名称 | 说明 |
| 1 | 类和接口注释 | 类注释中应当包含以下内容：类的详细说明和@author子句。 |
| 2 | 类和接口声明 | 如果有extends或implements子句则应当放在单独的一行中。 |
| 3 | 类（静态）变量 | 按照public/protected/(package)/private的顺序排列。public的成员变量必须生成文档（Java doc）。protected、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。 |
| 4 | 实例变量 | 按照protected/(package)/private的顺序排列（实例变量不能使用public限定符）。protected、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。 |
| 5 | 构造方法 | 如果有多个构造方法，则应该用递增的方式写，即参数多的写在后面。 |
| 6 | 存取方法 | 类变量的存取方法如果同时包含set和get方法，则它们应当成对放在一起。 |
| 7 | 类方法 | 类方法应当尽量按功能而不是按照范围或可访问性分组。例如一个private的类方法可能放在两个public的类方法之间，目的是使代码的阅读和理解更方便。类方法的书写顺序是先写静态方法，再写实例方法。 |
| 8 | toString方法 | 每个表示实体的类都应当定义toString 方法。toString的内容为可由外界访问到的各实例变量（包括提供了公有的get或set方法的变量）的字符串形式，打印格式规定为：  [⊔[⊔变量1名称⊔:⊔变量1内容⊔]⊔[⊔变量2名称⊔: ⊔变量2内容⊔]⊔…⊔]，其中字符“⊔”为空格符。 |
| 9 | main方法 | 如果定义了main方法，则该方法应当放在类的底部。 |

## 排版规范

### 页宽

页宽应该设置为120个字符。

### 换行

如果一个语句或表达式过长，则可以分行书写，换行的原则是：

* 在逗号之后换行；
* 在操作符之前换行；
* 在低优先级操作符前而不是在高优先级操作符前换行；
* 换行后的语句应当比原先的语句再缩进一个Tab。

### 缩进

类和接口等的定义、方法的定义、Throws语句、判断语句和循环语句等中的代码都要使用缩进风格。

缩进统一采用Tab，Tab的宽度统一设置为4。

### 空行

使用空行可以提高程序源代码的可读性。

遇到以下一些情况应当使用一个空行分隔：

* 一个源文件的各个部分（包声明语句、包导入语句和类声明）之间；
* 方法之间；
* 方法中的局部变量说明和第一个语句之间；
* 方法中的各个逻辑块之间。

### 空格

遇到以下一些情况时应当在适当的位置加入空格：

* 关键字后跟括号时关键字和括号之间以空格分隔（对于方法的说明和调用，方法名和左括号之间不要加入空格，这样有助于区分关键字和方法名），例如：

while (true) {

...

}

* 在方法的参数列表中在逗号后面应当加入空格；
* 除“.”操作符之外的所有的二元操作符应当将操作符和操作数用空格分开（一元操作符的操作符和操作数之间不要加入空格，如：“--”、“++”等），例如：

a += c + d;

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

while (d++ = s++) {

n++;

}

prints("size is " + foo + "\n");

* for语句的各表达式之间以空格分隔，例如：

for (expr1; expr2; expr3)

* 强制类型转换之后应当加入一个空格，例如：

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

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

+ 1);

### {}对

{}对中的}语句应该单独作为一行。例如：

while (true) {

...

} //正确，}语句单独作为一行

### 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：

method( AParameter ); //错误

method(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。例如：

if ((i) == 42){ //错误，内层括号无意义

### 换行符

换行符统一使用Windows平台格式，即“\r\n”。

## 注释规范

Java程序有两种类型的注释：实现注释（Implementation Comments）和文档注释（Documentation Comments）。

实现注释与C++的注释类似，可以使用“/\* … \*/”或“//”符号界定。

文档注释是Java特有的注释，它以“/\*\* … \*/”符号界定，这类注释可以被javadoc工具提取为HTML格式的文档文件。

实现注释用于注释源代码或特定的实现（供该类源代码的修改者查看和理解源代码）；文档注释是程序的与实现无关的说明书，主要用于使用该类进行开发，但没有该类的源代码的人员使用（供仅有该类的class文件的类使用者查看，以学习该类的使用方法）。

注释应当提供对类的一些概述性的信息，或者无法很容易就能从源代码看出来的额外信息。此外注释只能包含阅读和理解本类相关的一些信息，与本类无关的其他信息不应当出现在本类的注释中，例如：本类相应的包是如何被建立的、放在哪个目录下等信息不应当包含在本类的注释中。

注释中可以写入一些特殊的或者理解起来比较费力的设计决策，但是不要与明显能从现有代码中看出来的信息重复。因为这些冗余的注释信息很容易过时，如果这样做的话，一旦代码有了变动，则这些冗余的注释就需要从注释中删除或者予以更新。

注释不允许放在用“\*”或其他符号框起的大框中，例如：

/\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \* Comments \*

\* \* ………………………………………………….. \*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

注释中不允许包含特殊字符如换页符和退格符等。

***注意：应当视代码的具体情况进行适量的注释，注释的多少从另一个侧面反应出了这段代码的质量。如果一段代码需要加入很多注释才能让阅读者理解这段代码的含义，那就说明这段代码质量较差，此时应当考虑重写这段代码以使其看起来更加清晰。***

### 实现注释

程序中可以出现四种风格的注释：块注释（Block Comments）、单行注释（Single-Line Comments）、拖尾注释（Trailling Comments）和行尾注释（End-Of-Line Comments）。

### 块注释（Block Comments）

块注释用于提供对文件、方法、数据结构和算法的描述。块注释可以用在文件开头或方法开头，也可以用在其他一些地方，例如方法内部。如果用在方法内部，那么块注释应该放在它所描述的代码之前，缩进到与这段代码对齐，并且它与这段代码之间可以加入一个空行。例如：

/\*

\* 代码段注释。

\*/

代码段

### 单行注释（Single-Line Comments）

简短的注释应当写在一行中并且缩进到与所注释的代码对齐。如果在一行中无法写完注释，则应当使用块注释的方法进行注释。单行注释应当写在它所描述的代码之前，缩进到与这段代码对齐。例如：

if (condition) {

/\* 代码段注释 \*/

代码段

}

或

if (condition) {

// 代码段注释

代码段

}

### 拖尾注释（Trailling Comments）

特别简短的注释也可以与所要描述的代码段出现在同一行中，但这段注释应该被移动到能够与所注释的语句完全分离的位置，如果一大块代码中都出现了拖尾注释，则这些注释也应当缩进到能够相互对齐的位置。例如：

if (a == 2) {

return TRUE; /\* special case \*/

} else {

return isPrime(a); /\* works only for odd a \*/

}

或

if (a == 2) {

return TRUE; // special case

} else {

return isPrime(a); // works only for odd a

}

### 行尾注释（End-Of-Line Comments）

注释分隔符“//”可以用于注释一整行或者一行中的一部分。如果注释内容是文字，则注释多行时不要使用该注释分隔符；如果注释内容为程序代码，则可以使用该注释分隔符。例如：

if (foo > 1) {

// Do a double-flip.

...

}

else{

return false; // Explain why here.

}

//if (bar > 1) {

//

// // Do a triple-flip.

// ...

//}

//else{

// return false;

//}

### 文档注释

文档注释用于描述Java类、接口、构造方法、方法和域等元素。注释被放在注释分隔符“/\*\* … \*/”之间，每个元素应当只有一个文档注释，文档注释应当出现在这些元素之前。例如：

/\*\*

\* The Example class provides ...

\*/

public class Example { ...

对于以上元素中不需要提供给类用户的一些注释信息，例如类的实现方面的信息，可以使用实现注释的方法进行注释。

文档注释不允许放在构造方法和或方法定义块的内部，因为javadoc在处理文档注释时是把文档注释与该注释后的第一段声明代码块相关联的，这样生成的HTML文档会出现错误。

类或接口注释应当至少包含以下内容：类的功能描述、作者。并可以适当加入可参考的相关类等信息。例如：

/\*\*

\* The Example class provides ...

\*

\* @author 作者

\* @see 可参考的相关类

\*/

public class Example {

...

}

方法注释应当包含以下内容：方法的功能描述、参数描述、返回值、异常描述。例如：

/\*\*

\* The example method provides ...

\*

\* @param int param1 参数1描述

\* @param MyObject param2 参数2描述

\* @return String 返回值描述

\* @exception MyException1 抛出的异常1描述

\* @exception MyException2 抛出的异常2描述

\*/

public String example(int param1, MyObject param2)

throws MyException1, MyException2 {

...

}

以下是Java编程中常用的javadoc标签：

|  |  |  |
| --- | --- | --- |
| 标签 | 说明 | 适用范围 |
| @author 作者 | 指定生成文档中的“Author”项 | 概述、包、类、接口 |
| @version 版本号 | 添加“Version”标题 | 概述、包、类、接口 |
| @param 类型 参数名 描述 | 描述参数 | 构造方法、方法 |
| @return 类型 描述 | 描述返回值 | 方法 |
| @deprecated 描述 | 添加注释，表明不推荐使用该API | 包、类、接口、域、构造方法、方法 |
| @throws 异常类 描述 | 指定方法抛出的异常，在生成的文档中出现在“Throws”项中 | 构造方法、方法 |
| @exception 异常类 描述 | 同@throws | 构造方法、方法 |
| @since 版本号 | 为文档增加“Since”标题 | 概述、包、类、接口、域、构造方法、方法 |
| @see 参考主题 | 添加"See Also"标题，其中有指向参考主题的链接或者文本项，@see有三种使用方法：  （1）@see "string" 为"string"添加文本项，不产生链接；  （2）@see Label 使用HTML标记产生链接；  （3）@see package.class#member Label 使用Java语言的名字package.class#member产生链接。 | 概述、包、类、接口、域、构造方法、方法 |
| {@link 包名.类名#成员 标签} | 插入指向包名.类名#成员的内嵌链接 | 概述、包、类、接口、域、构造方法、方法 |
| {@value} 域值 | 指定静态域的在生成文档中显示的常量值 | 域 |

## 命名规范

遵循命名规范可以使得源代码的可读性更强。

### 包的命名

包的命名应当使用全部小写的ASCII字符命名。命名应当以顶级域名（如com、edu、org、gov、mil、net等）或ISO标准规定的两个英文字母的国家名（ISO 3166，1981，如cn）开始。包名中接下来的部分的命名应当根据组织内部的命名规范命名，这些命名规范可以是部门名、项目名、机器名、注册名、路径名或者功能分类名等。例如：

com.sun.eng

com.jxinol.wlinfo

edu.ctbu.cs.bovik.cheese

包的命名应该都是小写字母，单词之间用“.”分开。所有的JAVA文件必须建立在com.jinxinol.包下。例如：

package com. jinxinol.dcd.system;

### 类的命名

类的命名应该使用名词，如果有多个单词，则每个单词的首字母大写而其余字母一律小写。类名应当是简单的描述性的词语，应当使用完整的单词而尽量避免只取首字母的缩写词或任何的缩写形式（除非该缩写词比完整形式的词语使用更广泛，例如URL、HTML等，对于这些缩写词，也应当采用首字母大写而其余字母小写的形式）。例如：

class Raster;

class ImageSprite;

class HttpLink;

JUnit测试类名应该总是以“Test”单词开始，JUnit测试桩类名应该总是以“Fixture”单词结尾，JUnit测试套件类应该总是命名为“AllTests”。而其他类的类名应该总是避免使用这这些单词。

### 接口的命名

接口名应当使用形容词，其余规则与类名相似。例如：

interface Runnable;

interface Serializable;

### 常量的命名

声明类常量的变量名应当全部大写，并且各个单词之间以下划线符号“\_”分隔。例如：

static final int MIN\_WIDTH = 4;

static final int MAX\_WIDTH = 999;

static final int GET\_THE\_CPU = 1;

### 变量的命名

除类常量之外，所有的变量，包括实例变量、类变量等的命名都是以首字母小写（如果包含多个单词，则其余各个单词的首字母大写）的方式命名。尽管变量名中可以出现数字、下划线“\_”和“$”符号，但是变量名只能以字母开头而不能以这些字符开头。

变量的命名应当是简短且意义明确的。变量名应当易于记忆，也就是说，应当指出使用它的意图。例如：

float myWidth;

除了一些临时用完就抛弃的变量外，变量的命名应当尽量避免使用单字符变量或任何形式的缩写命名变量。在使用临时的单字符变量时，一般使用i、j、k、m和n做整型变量，而以c、d和e做字符型变量。例如：

int i;

char c;

### 方法的命名

方法名应该使用动词，且首字母小写，如果方法名包含多个单词，则其余各个单词的首字母大写。例如：

run();

runFast();

getBackground();

方法的参数应当使用有意义的名字命名，如果可能的话，使用和要赋值的域一样的名字。例如：

public void setSize(int size) {

this.size = size;

}

## 语句书写规范

### 简单语句

每一个简单语句应当写在一行中，例如：

argv++; // 正确

argv++;argc++; // 错误

### 复合语句

书写复合语句时应当遵循下面的规则：

* 花括号对“{}”内的语句应当比外层语句缩进1个Tab；
* 左花括号“{”应当放在复合语句开始行的尾部；右花括号“}”应当单独位于一行中，并且与复合语句的开始缩进至同一位置；
* 如果一个语句是控制结构如if-else、for等的一部分，则不论是单条语句还是多条语句，都应该放在花括号对“{}”中。这样当向该控制结构中加入新的语句时，可以避免由于忘记加括号而出现错误。

例如：

if (i == 2) {

x = 2;

}

### return语句

除非加上能使返回值看起来更加明显，否则return语句的返回值不要放在括号中：

return; //正确

return myObject.getSize(); //正确

return (size > 0 ? 0 : 1); //正确

return (result); //错误

### if，if-else，if-else if-else语句

if-else类型的语句应当使用下面的形式书写：

if (condition) {

statements;

}

if (condition) {

statements;

} else {

statements;

}

if (condition) {

statements;

} else if ( condition) {

statements;

} else {

statements;

}

***注意：任何情况下，if语句都应当使用花括号对“{}”括起来，而不要使用下面的形式：***

if (condition) //错误，没有使用花括号对{}

statement;

### for语句

for语句应当使用下面的形式书写：

for (initialization; condition; update) {

statements;

}

如果for语句的循环语句为空语句（所有的工作已经在初始化表达式、条件表达式和迭代表达式中完成），则可以使用下面的形式书写：

for (initialization; condition; update) ;

如果需要在for循环的初始化表达式和迭代表达式中使用逗号操作符时，表达式中变量定义和执行操作的数目不要超过三个。如果必须这样做时，可以把待初始化的变量放在for循环之前，而把要迭代的变量放在循环语句的末尾。

### while语句

while语句应当使用下面的形式书写：

while (condition) {

statements;

}

空while语句使用下面的形式书写：

while (condition) ;

### do-while语句

do-while语句应当使用下面的形式书写：

do {

statements;

} while (condition) ;

### switch语句

switch语句应当使用下面的形式书写：

switch (condition) {

case ABC:

statements;

/\* falls through \*/

case DEF:

statements;

break;

case XYZ:

statements;

break;

default:

statements;

}

每次当一个case语句没有break语句时，要在break语句的位置加入相应的注释，如上例中的注释/\* falls through \*/。每个switch语句都要有default语句。当向switch语句中加入新的case语句时，需要加在default语句之前。

### try-catch语句

try-catch语句的书写形式如下：

try {

statements;

} catch (ExceptionClass e) {

statements;

}

带finally子句的try-catch语句的书写形式如下：

try {

statements;

} catch (ExceptionClass e) {

statements;

} finally {

statements;

}

### 数组声明语句

数组声明时应该以如下形式书写：

String[] strs = new String[2];

而不要写成如下形式：

String strs[] = new String[2];

## 样例文件

以下是样例文件Example.java的内容：

/\*

\* Example.java

\* 创建者：陈兵

\* 创建日期：2012-07-11

\*

\* 版权所有(C) 2015-2015。金鑫电脑公司。

\* 保留所有权利。

\*/

package com.jinxinol.wlinfo;

import java.io.File;

/\*\*

\* Example类的描述可以放在这里。

\*

\* @author 史永锋

\*/

public class Example {

/\*

\* Example类的实现方面的说明可以放在这里

\*/

/\*\*

\* 自定义的常量。

\*/

public final static int MY\_CONSTANT = 1;

/\*\*

\* 类变量1。

\*/

protected static int count = 0;

/\*\*

\* 类变量2

\*/

private static int size = 1;

/\*\*

\* 实例变量2。

\*/

protected File testFile2 = null;

/\*\*

\* 实例变量3。

\*/

private File testFile3 = null;

/\*\*

\* 构造方法1。

\*/

public Example() {

}

/\*\*

\* 构造方法2。

\*

\* @param param1 参数1。

\*/

public Example(String param1) {

}

/\*\*

\* 构造方法3。

\*

\* @param param1 参数1。

\* @param param2 参数2。

\*/

public Example(String param1, File param2) {

}

/\*\*

\* 普通方法。

\*

\* @param param1 参数1含义描述。

\* @param param2 参数2含义描述。

\* @return 返回值含义描述。

\* @throws IndexOutOfBoundsException 出现该异常的原因及处理方法等的描述。

\* @throws NullPointerException 出现该异常的原因及处理方法等的描述。

\*/

public int method(String param1, String param2)

throws IndexOutOfBoundsException, NullPointerException {

return 0;

}

/\*\*

\* 域testFile3的set方法。

\*

\* @param testFile3 待指定的域testFile3的值。

\*/

public void setTestFile3(File testFile3) {

this.testFile3 = testFile3;

}

/\*\*

\* 域testFile3的get方法。

\*

\* @return 域testFile3。

\*/

public File getTestFile3() {

return testFile3;

}

/\*\*

\* 返回该对象的字符串形式。

\*

\* @return 该对象的字符串形式。

\* @todo Implement this java.lang.Object method

\*/

public String toString() {

return "";

}

/\*\*

\* 类的主函数。

\*/

public static void main(String[] args) {

}

}

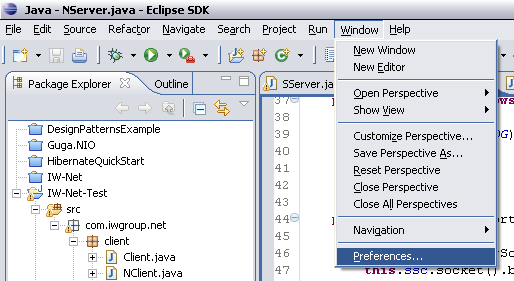
## 代码格式化模板文件使用方法

对于代码的格式化的问题，公司已为两种IDE Eclipse制作了模板文件，以方便大家工作。现讲述Eclipse中的使用方法：

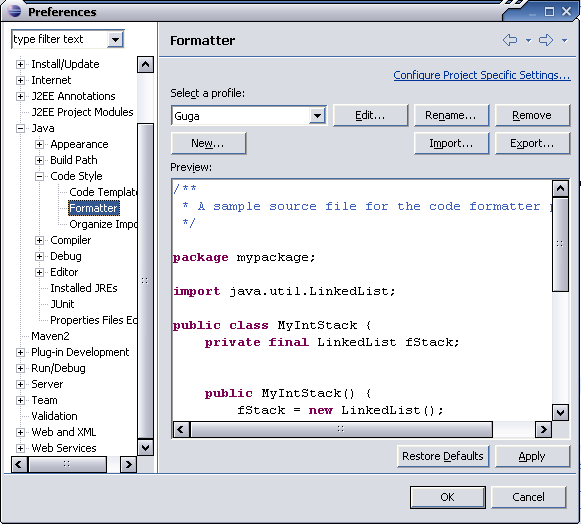
### Eclipse

使用模板文件JXCodeStyleEclipse-V3.0.xml，方法如下：

在eclipse窗口中选择菜单“window / preferences”打开preferences对话框。



在对话框左侧的列表树中选择“Java / Code Style / Formatter”。



选择对话框右侧的“Import”按钮，打开文件选择对话框，选择模板文件“JXCodeStyleEclipse.xml”，单击打开即可。



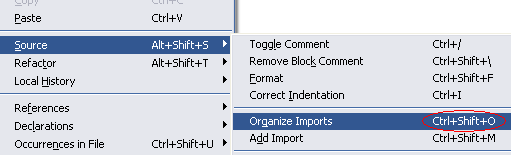
## IDE格式化使用方法

### Eclipse格式化使用方法

Eclipse的格式化分为代码中import部分的格式化和代码的格式化。

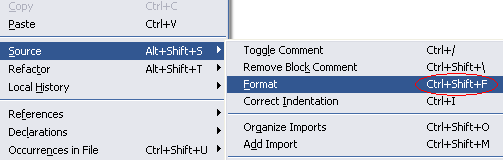
在代码编辑页面中单击右键弹出菜单，选择“Source / Organize Imports”即执行import的格式化，他将import部分按照格式化模板中的格式进行重新排序。

或在代码编辑页面中按组合键“Ctrl + Shift + O”，执行相同操作。

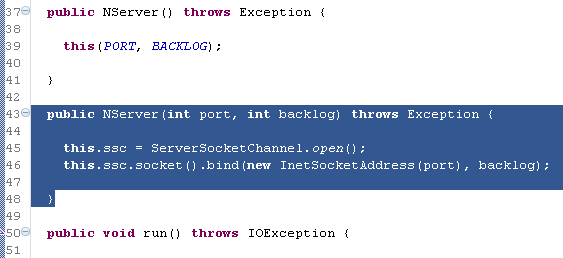


在代码编辑页面中单击右键弹出菜单，选择“Source / Format”即执行代码的格式化操作，他将代码按照格式化模板中的格式进行重新安排。

或在代码编辑页面中按组合键“Ctrl + Shift + F”，执行相同操作。



**提示1：在执行代码格式化操作时，如果选择了一段代码，则格式化操作只针对这段被选择的代码进行，否则格式化整个文件中的代码。**



**提示2：也可以一次对整个工程中的所有文件或者某个包中的文件执行import格式化和format格式化，在Package Explore视图中右键单击想要进行格式化的工程，或工程中的源码目录，或某个具体的包，在弹出菜单中选择“Source / Format”或“Source / Organize Import”，即可执行批量的import格式化和代码格式化。**

