《JAVA 编码规范》

作者：袁慎建

版本：v0.1

二〇一四年一月

版本历史

日期	版本	说明（文档修改描述）	作者
2014	0.1	创建	袁慎建

目 录

1 概述

1.1 目的

在软件的生命周期中，维护的花费通常占很大的比例，且几乎所有的软件，在其整个生命周期中，开发人员和维护人员都不尽相同。编码规范可以改善软件的可读性，使程序员尽快而彻底地理解代码；同时，编码规范还可以提高程序代码的安全性和可维护性，提高软件开发的生产效率，所以，编码规范对于程序员而言至关重要。

为使开发项目中所有的 JAVA 程序代码的风格保持一致，增加代码的可读性，便于维护及内部交流，使 JAVA 程序开发人员养成良好的编码习惯，有必要对 JAVA 程序的代码编码风格做统一的规范约束。本文档定义了我公司软件开发过程中使用的开发语言的编码规范，指导软件开发人员在进行项目开发过程中提高代码质量、统一编码要求。

1.2 适用范围

除客户方另有特别要求外，适用于 JAVA、JSP、Servlet 等项目的开发。

2 JAVA 源文件

2.1 PACKAGE 的组织

Package 是组织相关类的一种比较方便的方法。Package 使我们能够更容易查找和使用类文件，并可以帮助我们在运行程序时更好的访问和控制类数据。

类文件可以很容易的组织到 Package 中，只要把相关的类文件存放到同一个目录下，给该目录取一个与这些类文件的作用相关的名称。如果需要声明程序包，那么每个 JAVA 文件（*.java）都需要在顶部进行 Package 的声明，以反映出包的名称。

例：**package** com.meritit.product.modul.dao;

2.2 JAVA 源文件的内部结构

2.2.1 Package/Import

Package 行要在 import 行之前，import 中标准的包名要在本地的包名之前。如果 import 行中包含了同一个包中的不同子目录，应 import 到某一个指定的类，避免*类型的 import。（导包：**Ctrl+Alt+O**）

例：

```
package com.meritit.product.modul.dao;
import   java.io.InputStream;
import   java.io.OutputStream;
```

```
import   java.io.*;(不提倡，应该避免)
```

2.2.2 Class

所有的 JAVA(*.java) 文件都应遵守如下的样式规则，如果 JAVA 源文件中出现以下相应的部分，应遵循如下的先后顺序。

编码时，即使某个类不是 public 类型的，也要在一个独立的 JAVA 文件（*.java）中声明，避免一个 JAVA 文件包含多个类声明。

如下面命名规则是需要避免的：

```
/**
 * 源码名称：Merit.java
 * 日期：2014-01-15
 * 程序功能：    Merit组织类
 * 版权：CopyRight@2014
 * 作者：meritit.com co.ltd
 */
package com.meritit.product.modul.dao;
/**
 * Merit组织类，封装了Merit公司的各种信息
  */
public class Merit {// ...}
class DataMiningCenter {
   // ...
}
```

需要将 DataMiningCenter 类提取出来放在单独源文件中申明

类的注释一般是用来解释类的，建议使用文档注释。

例：

```
/**
* Class description goes here.
* ...
*/
```

接下来是类的定义，有可能包含了 extends 和 implements。

例：

```
public class CounterSet extends Observable implements Cloneable {
    //...
}
```

2.2.3 Field

public 的成员变量应使用 JavaDoc 注释，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,成员变量要求放置到类的顶部，常量放置成员变量顶部。注释成员变量时，建议使用文档注释。

例：

```
/** classVar1 documentation comment */
public static int classVar1 = 0;
private static int height = 0;
```

2.2.4 Constructor

构造函数应该按照参数数目的递增顺序进行书写。

例：

```
public class Merit {
    private String name;
    public Merit() {
        // ...
    }
    public Merit(String name) {
        this.name = name;
    }
}
```

不提倡使用：

```
public Merit(String n) {
    name = n;
}
```

传入参数名与属性名一致

2.2.5 getter/setter

如果存取方法只进行简单的赋值或取值操作，可以写在一行上，否则不要写在一行上。（Format：Ctrl+Shift+F）

例：

```
/**
 * Set the counters
 * …
 */
public void setPackets(int[] packets) { this.packets = packets; }

/**
 * Get the counters
 */
public int getPackets() {
    packets++;
    return packets;
}
```

```
return ++packets;
```

2.2.6 Member Method

对于类的具体方法，应将功能相似的方法放置在一起。

例：

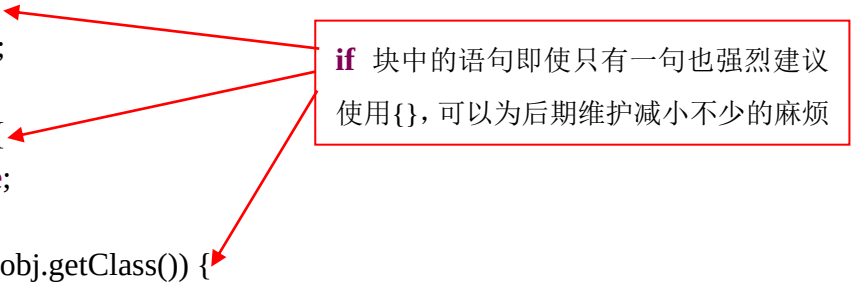
```
/**
 * Method doSomething documentation comment…
 */
public void doSomething() {
    // …
}
```

2.2.7 hashCode/equals

如果有必要覆盖父类的 equals 方法，建议同时覆盖 hashCode 方法，下面是 eclipse 自动生成的一段经典的 equals 和 hashCode 的实现：

```
private int employees;
// ...
@Override
```

```
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + employees;
    return result;
}
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    Merit other = (Merit) obj;
    // ...
}
```



2.2.8 toString

每一个类都应该定义 toString 方法。

例：

```
/**
 * toString method for this class…
 */
public String toString() {
    // …
}
```

2.2.9 clone

如果这个类是可以被克隆的，那么下一步就是 clone 方法：

例：

```
/**
 * clone method for this class…
```

```
 */
public Object clone() {
    // …
}
```

3 命名规则

驼峰命名规则：当变量名或函式名是由一个或多个单字连结在一起，而构成的唯一识别字时，第一个单词以小写字母开始；第二个单词及后面的单词的首字母都采用大写字母。

3.1 Java 源文件的命名

JAVA 源文件名必须和源文件中所定义 **public** 的类的类名相同。

3.2 Package 的命名

Package 名的第一部分应是小写 ASCII 字符，com.meritit 开头，后续内部命名规则指定企业域名、项目/产品名等。采用平台则遵循平台规范。

例：

```
package com.meritit.product.eism.module.dao;
package com.meritit.project.tky.module.dao;
```

3.3 Class 的命名

Class 名应是首字母大写的名词。命名时应该使其简洁而又具有描述性。异常类的命名，应以 Exception 结尾。类的命名遵循驼峰命名规则的基础上将第一个字母大写。

例：

```
public class Set implement ISet {
    // …
};
public class InvalidException extends Exception {
    // …
```

};

3.4 Interface 的命名

Interface 名应是首字母为 I，后跟首写字母大写的名词。Interface 的命名在 Class 的命名基础上在首字母前加上一个 I。

例：

```
public interface ISet {
    double MAX_HEIGHT = 5.5;
    void set();
}
```

interface中：
属性默认是： **public static final**
方法默认是： **public**
建议不要过写过多代码

3.5 Enum 的命名

基本与 Class 的命名规范类似。在满足 Class 命名规则的基础之上，保证开头第一个字母为 “E”， 便于与普通的 Class 区别开。Enum 的命名在 Class 的命名基础上在首字母前加上一个 E。

例：

```
public enum EColor {
    RED, BLUE, BLACK, YELLOW, GREEN;
}
```

3.6 Constant 的命名

常量名的字母应全部大写，不同的单词之间通过下划线进行连接，并且名字组合应该赋予含义。

例：

```
public static final int MIN_WIDTH = 4;
private static final String NAME = "Merit";
private final int MAX_WIDTH = 999;
```

3.7 Variable 的命名

3.7.1 普通变量

普通变量命名遵循驼峰命名规则，建议非实例布尔型变量（即方法变量）建议 is 开头。

例：

```
float minAreaWidth = 0.0F;
double maxAreaWidth = 100.0;
boolean isNew = false;
```

3.7.2 约定变量（建议项）

所谓约定变量，是指那些使用后即可抛弃（throwaway）的临时变量。通常在 while 和 for 等语句中常用的变量， i、j、k、m 和 n 代表整型变量；c、d 和 e 代表字符型变量。

例：

```
int    i = 0;
char c = 'a';
```

3.8 Method 的命名

方法名的第一个单词应该是动词，并且遵循驼峰命名规则。建议布尔类型 is、has 、exits 开头。

例：

```
void findPersonID(int id);
void findEmp(String nameAndAge);
boolean exists(String nameAndAge);
boolean hasUser(String nameAndAge);
boolean isTop(String nameAndAge);
```

3.9 方法参数的命名

应该选择有意义的名称作为方法的参数名，遵循驼峰命名规则。如果可能的话，选择和需要赋值的字段一样的名字。建议方法参数如果不可修改，则需增加 final 修饰符。

例：

```
void setCounter(final int size) {
    this.size = size;
}
```

4 样式结构

4.1 整体样式

4.1.1 缩进和对齐

- 缩进

当某行语句在逻辑上比下面的语句高一个层次时，该行下面的语句都要在该行的基础上缩进一个单位。（Format：Ctrl+Shift+F）

例：

```
public void someMethod(String parameterA, String parameterB) {
    int variantA = 0;
    // Sentence1 gose here...;
    if (Condition) {
        // Sentence2 gose here...;
    }
}
```

- 对齐

若干语句在逻辑上属于同一层次时，这些语句应对齐。（Format：Ctrl+Shift+F）

例：

```
public void someMethod(parameterA) {
    int variantA=0;
    // Sentence1 gose here...;
```

```
        if (Conditions) {
            // Sentence2 gose here...;
            // Sentence3 gose here...;
        }
    }
```

4.1.2 行宽

为了和 linux,unix 等字符界面的操作系统兼容，JAVA 代码行应限制在 120 个字符之内，多余部分应换行。建议字符变量定义按照上述规则使用+号换行。（Format：Ctrl+Shift+F）。

例：

variantA = someMethod(longExpression1, longExpression2, longExpression3);（错误×）

应改为：

variantA = someMethod(longExpression1, longExpression2,
longExpression3);（正确√）

4.1.3 断行规则

当一句完整的语句大于 120 个字符时需要断行，断行时，应遵循下面规则（Format：Ctrl+Shift+F）。

- 在逗号后换行

  例：

  variantA = someMethod(longExpression1, longExpression2,
  longExpression3, longExpression4);

- 在操作符前换行

  例：

  longName1 = longName2 * (longName3 + longName4 - longName5)
  + 4 * longName6;

- 换行后，应和断行处的前一层次对齐（Format：Ctrl+Shift+F）

  例：

```
longName1 = longName2 * (longName3 + longName4 - longName5)
+ 4 * longName6;（错误×）
```

应改为：

```
longName1 = longName2 * (longName3 + longName4 - longName5)
                    + 4 * longName6;（正确√）
```

- 换行时尽量选择高层次的地方进行换行（Format：Ctrl+Shift+F）

  例：

```
longName1 = longName2 * (longName3 + longName4
                                    - longName5) + 4 * longName6;  （错误×）
```

应改为：

```
longName1 = longName2 * (longName3 + longName4 - longName5)
                    + 4 * longName6;  （正确√）
```

- 在使用上述的规则换行后对齐时，如果次行的长度大于 120 个字符，应改用两个单位的缩进来代替层次对齐（Format：Ctrl+Shift+F）

  例：

```
private static synchronized void horkingLongMethodName(int anArg,
        Object anotherArg, String yetAnotherArg, Object andStillAnother) {
    // …
}
```

4.1.4 空白的使用

- 空格字符的使用（Format：Ctrl+Shift+F）

1) 关键字和括号()之间要用空格隔开

例：

```
while (Condition1) {
    // Sentence gose here...;
}
if (Condition2) {
    // Sentence gose here...;
}
```

2) （建议）参数列表中逗号的后面应该使用空格

例：

```
public void methodA(parameterA,  parameterB,  parameterC) {
    // Sentence gose here...;
}
```

3) 所有的二元运算符，除了"."，应该使用空格将之与操作数分开

例：

```
longName1 = longName2 * (longName3 + longName4) + 4 * longName5;
```

4) 强制类型转换后应该跟一个空格

例：

```
methodA((byte) parameterA, (Object) parameterB);
```

5) 左括号右边和右括号左边不能有空格

例：

longName1 = longName2 * (longName3 + longName4); （错误×）

应改为：

longName1 = longName2 * (longName3 + longName4); （正确√）

6) 方法名与其参数列表的左括号之间不能有空格

例：

methodA (parameter1, parameter2); （错误×）

应改为：

methodA(parameter1, parameter2); （正确√）

7) 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：

variantA += variantB --;（错误×）

应改为：

variantA += variantB--;（正确√）

- 空白行的使用

空白行将逻辑相关的代码段分隔开，以提高可读性，有如下几种情形：

1) （**建议**）一个源文件的两个片段(section)之间用两个空白行（不强制执行）

例：

用两个空白行将 JAVA 文件顶端的版权说明和下面的内容隔开

```
/**
* Copyright (C) meritit    Co., Ltd.
* …
*/

package com.meritit.product.eismmodule.dao;
```

2) 两个方法的声明之间使用一个空白行

例：

```
public class Merit {

    private void methodA() {
        // …
    }

    private void methodB() {
        // …
    }
}
```

3) （建议）方法内的局部变量和方法的第一条语句之间使用一个空白行

例：

```
private void methodA() {
        int variantA = 0;
        int variantB = 0;

        variantA = variantB + 10;
        // …
    }
```

4) （建议）块注释或单行注释之前使用一个空白行

例：

```
        if (condition) {

            // This single line comoment gose here…
            variantA = variantB + 10;

            /* block comment gose here... */
            variantA = variantB + 10;
```

```
    }
```

5) （**建议**）一个方法内的两个逻辑段之间应该用一个空白行

例：

```
private int methodA() {
    variantA = methodGet();

    variantB = methodGet();
    return variantA + variantB;
}
```

4.2 注释样式

JAVA 程序有两类注释，实现注释(implementation comments)和文档注释(document comments)。

实现注释，就是使用 /*...*/ 或 // 界定的注释。文档注释，又被称为"doc comments"或" JavaDoc 注释"，是 JAVA 独有的，由 /**...*/ 界定，并且文档注释可以通过 JavaDoc 工具转换成 HTML 文档。

在注释里，应该对设计决策中重要的或者不是显而易见的地方进行说明，但应避免对意思表达已经清晰的语句进行注释。

特别注意，频繁的注释有时反映出代码的低质量。当你觉得被迫要加注释的时候，考虑一下是否可以重写代码，并使其更清晰。

4.2.1 实现注释

- 块注释

块注释通常用于提供对文件，方法，数据结构和算法的描述。块注释被置于每个文件的开始处以及每个方法之前。它们也可以被用于其他地方，比如方法内部。在功能和方法内部的块注释应该和它们所描述的代码具有一样的缩进格式，并且，**建议**块注释之首应该有一个空白行，用于把块注释和代码分割开来。

例：

```
previousSentences;
```

```
/*
 * block comment gose here...
 */
Sentences;
```

- 单行注释

单行注释显示在一行内，并与其后的代码具有一样的缩进层次。如果一个注释不能在一行内写完，应采用块注释，且建议单行注释之前应该有一个空白行。

例：

```
if (condition) {

    // This single line comoment gose here…
    ...
}
```

- 行末注释

行末注释的界定符是"//"，它可以注释掉整行或者一行中的一部分，一般不用于连续多行的注释文本。但是，它可以用来注释掉连续多行的代码段。

例：

```
if (true) {
    // Sentence gose here...;
} else {
    return false; // Explain why here.
}
```

4.2.2 文档注释

置于/**...*/之中的注释称之为文档注释。

文档注释用来描述 Java 的类、接口、构造器、方法以及字段(field)，一个注释对应一个类、接口或成员，该注释应位于声明之前，与被声明的对象有着相同的缩进层次。

在类的声明中，各种类、接口、变量、常量、方法之前都应该有相应注释。关于文档注释中的各种 target 的使用。

- 版权注释

```
/**
```

```
* @Project: ${project_name}
* @Title: ${file_name}
* @Package ${package_name}
* @Description: ${todo}
* @author wangkun wangkun@meritit.com
* @date ${date} ${time}
* @Copyright: ${year}
* @version V1.0
 */
```

- 类注释

```
/**
* @ClassName ${type_name}
* @Description ${todo}
* @author meifeng meifeng@meriti.com
* @date ${date}
* @see Connection#prepareStatement
* @see ResultSet
 */
```

- 方法注释

```
/**
 * @return <code>true</code>
 * @exception SQLException
 *                        if a database access error occurs;
 * @throws SQLTimeoutException
 *                    when the driver has determined that the timeout value that was
 *                    specified by the {@code setQueryTimeout} method has been exceeded
 *                    and has at least attempted to cancel
 * @see Statement#execute
 * @see Statement#getResultSet
 * @see Statement#getUpdateCount
 * @see Statement#getMoreResults
*/
```

4.3 声明

4.3.1 变量的声明

- 一行只声明一个变量

例：

```
int variantA = 0, variantB = 0;  （错误×）
```

应改为：

```
int variantA = 0;
int variantB = 0;  （正确√）
```

- 声明变量时要对其进行初始化，如果是类中的变量，声明时不初始化，**建议**在构造函数中对关键参数初始化（不针对默认初始化的成员变量）。

例：

```
int variantA;  （错误×）
```

应改为：

```
int variantA = 0;  （正确√）
```

```
public class Emp {
    private String name;
    private int age;
    public Emp() {
        name = "merit";
        age = 0;
    }
}（错误×）
```

应改为：

```
public class Emp {
    private String name;
    private int age;
    public Emp() {
        name = "merit";
    }
}（正确√）
```

- 临时变量放在其作用域内声明

例：

```
int tempA = 0;
if (condition) {
    tempA = methodA();
    methodB(tempA);
}（错误×）
```

应改为：

```
if (condition) {
    int tempA = 0;
    tempA = methodA();
    methodB(tempA);
}（正确 √）
```

- 声明应集中放在作用域的顶端

例：

```
if (condition) {
        int tempA = 0;
        tempA = methodA();
        int tempB = 0;
        tempB = methodB();
}（错误×）
```

应改为：

```
if (condition) {
        int tempA = 0;
        int tempB = 0;
        tempA = methodA();
        tempB = methodB();
}（正确 √）
```

- 避免声明的局部变量覆盖上一级声明的变量

例：

```
int counter = 0;
if (condition) {
        int counter = 0;
        counter = methodA();
}（错误×）
```

应改为：

```
int counter = 0;
if (condition) {
        int counterTemp = 0;
        counter = methodA();
}（正确 √）
```

4.3.2 类和接口的声明

当编写类和接口时，应该遵守以下规则：

- 在方法名与其参数列表之前的左括号"("间不要有空格
- 左大括号"{"位于声明语句同行的末尾，并与末尾之间留有一个空格
- 右大括号"}"另起一行，与相应的声明语句对齐。如果是一个空语句，"}"应紧跟在"{"之后
- 方法与方法之间以空白行分隔

例：

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

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

        public int emptyMethod() {
        }
}
```

4.4 语句

4.4.1 简单语句

- 每行至多包含一条完整语句

例：

```
variantA++; variantB++;（错误×）
```

应改为：

```
variantA++;
variantB++;（正确√）
```

- 在没有必要的情况下，不要在 return 语句中使用括号

例：

```
return (0);  （错误×）
```

应改成：

```
return 0;  （正确√）
```

4.4.2 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 }"，其编码应有如下基本规则：

- 被括其中的语句应该比复合语句缩进一个层次
- 左大括号"{"应位于复合语句起始行的行尾，并且空一个空格，右大括号"}"应另起一行并与复合语句首行对齐
- 复合语句即使只有一个语句，也要有大括号作为界定
- 每行至多包含一条完整语句

1. 判断语句

例：

if-else 语句建议不超过 3 层，应该具有如下格式：

```
if (condition) {
        // Sentences go here…;
 }
 if (condition) {
        // Sentences go here…;
 } else if (condition) {
        // Sentences go here…;
 } else {
        // Sentences go here…;
 }
```

2. 选择语句

在选择语句中应添加 default 情况，防止不可预知的情况发生。 当一个 case 在没有 break 语句的情况下，它将顺着往下执行。应在 break 语句的位置添加注释。[下面就含注释/* falls through */]

例：

```
switch (condition) {
    case 1:
        // Sentences go here…;

        /* falls through */
    case 2:
       // Sentences go here;
        break;
    case 3:
        // Sentences go here;
        break;
    default:
        // Sentences go here;
        break;
}
```

3. 循环语句

在 for 语句的初始化或更新子句中，如果存在两个以上时，需要在外部定义。同时，应避免使用三个以上子句，从而导致复杂度提高；若确实需要，可以在 for 循环之前放置初始化子句或在 for 循环末尾放置更新子句。

例：

```
for (int i = 0, j = 10, k = 10, m = 50; i < j + k + m; i++, j--, k--, m--) {
        // Sentences go here...;
}（错误×）
```

应改为：

```
 int i = 0;
int j = 100;
int k = 1000;
int m = 500;
for (; i < j + k + m ;) {
        // Sentences go here...;
        i++;
        j--;
        k--;
        m--;
}（正确√）
```

- 一个空的 for 语句和 while 语句只用一行

例：

```
for (initialization; condition; update) {
    // Sentences go here...;
}
```

对应

```
for (initialization; condition; update);

while (condition) {
    // Sentences go here...;
}
```

对应

```
while (condition);
```

4. try-catch 结构语句

例：

```
try {
    // Sentences be catched...;
} catch (Exception e) {
    // Sentences go here...;
}

try {
    // Sentences be catched...;
} finally {
    // Sentences to close the resource,always be executed...;
}

try {
    // Sentences be catched...;
} catch (Exception e) {
    // Sentences go here...;
} finally {
    // Sentences to close the resource,always be executed...;
}
```

5 典型示例

1. 方法的返回值表达应尽量简单。在 return 语句后一般不使用括号，但是如果返回中包含复杂表达式，则要使用括号。

   例：

```
return ((x >= 0) ? x : y);
```

2. 比较对象用 equals() 方法代替操作符“==”。特别是不能用“==”去比较字符串类型的变量。

例：

```
String strA = "abc";
String strB = "bcd";
if (strA == strB) {
    // Sentence goes here…
}
```

应改为：

```
String strA = "abc";
String strB = "bcd";
if (strA.equals(strB)) {
    // Sentence goes here…
}
```

3. if、while 等语句中的条件判断部分，应将常量（如果有,建议常量单独定义，不直接使用数字）写在“==”运算符的左边，而把变量写在右边。

例：

```
final   String MAX=”3”;
 if (MAX.equals(var)) {
      // Sentence goes here...
 }
 while (MAX == var ) {
     // Sentence goes here...
 }
```

4. hasCode、equals、compareTo，项目方便建议采用 Apache Commons Lang 实现方式,实体类建议这三个方法实现，另外建议参考 eclipseIDE 的默认实现方式

例：

- **反射方式**

```
@Override
 public int hashCode() {
     return HashCodeBuilder.reflectionHashCode(this);
 }
@Override
```

```
 public boolean equals(Object obj) {
        return EqualsBuilder.reflectionEquals(this, obj);
 }
@Override
public int compareTo(Object obj) {
    return CompareToBuilder.reflectionCompare(this, obj);
 }
```

- 定制方式

```
    @Override
    public int hashCode() {
        return new HashCodeBuilder(17, 37)
                .append(name)
                .append(age)
                .append(bar)
                .toHashCode();
    }
@Override
public boolean equals(Object obj) {
    boolean flag = false;
    if (obj != null && Foo.class.isAssignableFrom(obj.getClass())) {
        Foo f = (Foo) obj;
        flag = new EqualsBuilder().append(name, f.getName())
                .append(age, f.getAge()).append(bar, f.getBar()).isEquals();
    }
    return flag;
 }
@Override
public int compareTo(Object obj) {
    int flag = -1;
    if (obj != null && Foo.class.isAssignableFrom(obj.getClass())) {
        Foo f = (Foo) obj;
        flag = new CompareToBuilder().append(name, getName())
                .append(age, f.getAge()).append(bar, f.getBar())
                .toComparison();
    }
    return flag;
}
```

5. 异常捕获 try-catch-finally 的编写

```
try {
```

```
        try {
            // Sentences be catched...;
        } finally {
            // Sentences to close the resource,always be executed...;
        }
    } catch (Exception e) {
        // Sentences go here...;
    }
```

6 尾声

6.1 作者简介

基础实践篇到此已经接近尾声，本文档作者袁慎建，生于 1990 年 6 月 12 日，老家在湖北省黄石市阳信县军垦农场河头屋村 91#，2009 到 2013 年就读于长安大学材料学院，大学官方专业是“材料成型与控制工程”，大学期间长期处于迷茫期，由于没有确切找到自己的兴趣，和大多数从农村走向大城市的孩纸一样把主要精力花在了文化课的学习（大一大二的时候学习成绩很好的噢），课余时间常常泡图书馆看课外书，主要有历史，修身养性之书、文学名著等，以弥补大学前的大脑空白，但由于空白面积太大，200 多本书籍（孩纸，别多想了，木有不良书籍的）都没有给我留下深刻的印象（大学期间英语四、六级、计算机二级都是一次通过的噢）。大三了，开始不习惯大学里面的教育方式了，开始不上课专门看书，考试只求不挂科，直到大四，鼓起勇气在两个毫不相关的专业进行跨界，转行开始学习 Java 软件开发，实践证明，自己对这方面很有兴趣，一直积极主动地去学习相关知识和技术，并动手实践，现在我经过从行外到行内的过度，中间一直没有停止学习，也经常总结，谈不上深刻，希望能够给那些正在起步的 Java 爱好者一起启发，本人在 CSDN 上的博客访问量不大，一直在分享，也希望以后的学习之路得到大牛的指点，希望和大家一起学习交流。

6.2 特别声明

本文档仅代表个人意见，存在错误或遗漏欢迎指正和交流，文档仅供学习。

6.3 学习交流

我的 CSDN 博客主页：http://my.csdn.net/ysjian_pingcx

我的邮箱：ysjian_pingcx@126.com

我的 QQ：646633781

我的微信：ysjian_pingcx

我的特立风：18192235667

我的说明：平常心

6.4 关于 JavaDoc

6.4.1 常用的 JavaDoc 标记

JavaDoc 标记是插入文档注释中的特殊标记。JavaDoc 标记由“@”、标记、专用注释引用组成。

JavaDoc 主要有以下标记：

*@author （用于类和接口，标明开发该类模块或接口的作者）

*@version （用于类和接口，标明该模块的版本）

*@param （用于方法和构造函数，说明方法中的某个参数）

*@return （用于方法，说明方法的返回值）

*@exception （用于方法，说明方法可能抛出的异常,同@throws）

*@see （对类、属性、方法的说明，参考转向，也就是相关主题）

*@since （说明引进该类、方法或属性的起始版本）

*@deprecated （说明某类或方法不被推荐使用,以及相应的替代类或替代方法）

关于 JavaDoc 标记使用的更详细信息请参考下面链接：

http://java.sun.com/j2se/1.4.1/docs/tooldocs/windows/javadoc.html#javadoctags

@author 和@version

@author 标记用于指明类或接口的作者。在缺省情况下 JavaDoc 工具将其忽略，但命令行开关-author 可以修改这项功能，使其包含的信息被输出。

语法：@author 作者名。

@author 可以多次使用，以指明多个作者，生成的文档中每个作者之间使用逗号“，”分隔。

@version 标记用于指明类或接口的版本。

语法：@version 版本号。

例：

```
/**
 * @author ysjian
 * @author ysjian
 * @version 1.0
 */
```

@param、@return 和@exception

这三个标记都是用于方法说明。

@param 标记用于描述方法的参数。

语法：@param 参数名 参数的描述。

每一个@param 只能描述方法的一个参数，所以，如果方法需要多个参数，就需要多次使用@param 来描述。

@return 标记用于描述方法的返回值。

语法：@return 返回值描述。

一个方法中只能用一个@return。

@exception 标记用于说明方法可能抛出的异常，同@throws。

语法：@exception 异常类 抛出异常的原因。

一个方法可以有多个@exception 标记。

例：

```
/**
 * @param param1
 *                  description of param1
 * @param param2
 *                  description of param2
 * @return true or false
 * @exception java.lang.Exception
 *                      throw when switch is 1
 * @exception NullPointerException
 *                      throw when parameter is null
 */
```

@see

该标记用于参考转向。

语法：

@see 类名

@see #方法名或属性名

@see 类名#方法名或属性名

@see HTML 链接字

关于类名，如果 JAVA 源文件中的 import 语句包含了该类，可以只写出类名；若没有包含，则需要写出类全名（如 java.lang.String）。对于方法或属性，如果没有指定类名，则默认为当前类。对于方法，还需要写出方法名及其参数类型，没有参数的，需要写一对括号。

例：

```
/**
 * @see String
 * @see java.lang.String The String Class
 * @see java.lang.StringBuffer#str
 * @see #str
 * @see #str()
 * @see #main(String[])
 * @see Object#toString()
 * @see <a href=”http://…/somepage.html”>some page</a>
 */
```

@since

该标记用于**说明引进某个类、方法或属性的起始版本。**

语法：@since 版本号

其中，对“版本号”没有特殊的格式要求。@since 标记可用于包、类、接口、方法、属性的文档注释里。表示它描述的修改或特性从“版本号”所描述的版本开始就存在了。

例：

```
/**
 * @since 1.2
 */
```

@deprecated

该标记用于指出某个类或方法不被推荐使用，以及相应的替代类或替代方法。

语法：@deprecated　deprecate-Description

@deprecated 标记可用于包、类、接口、方法、属性的文档注释里。Deprecate-Description 里，首先要说明该类或者方法从什么时候（什么版本）起不再使用，其次要说明用哪个类或者方法可以替代它。可以使用@see 或@link 指明替代类或替代方法。

例：

```
/**
 * @deprecated As of JDK 1.1, replaced by {@link setPrice(int)}
 */
```

又例：

```
 /**
  * @deprecated As of JDK 1.1, replaced by setBounds
  * @see #setBounds(int, int, int, int)
  */
```

6.4.2 JavaDoc 命令的使用

用法：

javadoc [options] [packagenames] [sourcefiles] [classnames] [@files]

主要选项：

-public	显示 public 类和成员及其注释
-protected	显示 protected/public 类和成员（缺省）及其注释
-package	显示 package/protected/public 类和成员及其注释
-private	显示所有类和成员及其注释
-d <directory>	输出文件的目标目录
-doclet <class>	指定生成输出文档使用的 doclet
-docletpath <path>	指定 doclet 类的搜索路径
-sourcepath <pathlist>	指定源文件的搜索路径
-classpath <pathlist>	指定用户类的搜索路径
-version	显示@version 段包含的内容

-author 显示@author 段包含的内容

-splitindex 将索引分为每个字母对应一个文件

-windowtitle <text> 文档的浏览器窗口标题

-verbose 是否显示 javadoc 运行的详细信息

-charset <charset> 设置生成 HTML 文档的字符集

关于 JavaDoc 工具如何使用的更多详细信息请参考下面链接：

http://java.sun.com/j2se/1.4.1/docs/tooldocs/windows/javadoc.html#options

JavaDoc 可以对单个文件或者一个 package 生成文档，有如下两个命令：

javadoc samples\sample.java

javadoc samplespackage

第一个命令对一个单独的 java 文件 sample.java 生成 HTML 文档；第二个命令对一个 package 包 samplespackage 生成 HTML 文档。

1）**-public、-protected、-package、-private**

这四个选项，只需要任选其一即可。-public 在生成的文档中只包含 public 类型的类或者方法及其注释内容，-protected 选项生成 protected、public 的类和成员及其注释内容，-package 选项生成 package、protected、public 的类和成员及其注释内容，-private 选项生成所有的类和成员及其注释内容。

例：

javadoc -public sample.java

2）**-d <targetpath>**

选项允许你定义输出目录。如果不用 -d 定义输出目录，生成的文档文件会放在当前目录下。targetpath 可以是相对路径，也可以是绝对路径。

例：

javadoc -d doc\ samplespackage

3）**-doclet <class>**

指定一个类，该类用于启动生成 java 文档的 doclet，该 doclet 指定了输出的内容和格式。如果没有指定 doclet，将采用默认的 doclet。

例：

javadoc -doclet com.sun.tools.doclets.mif.MIFDoclet

4）**-sourcepath <pathlist> 和 -classpath <pathlist>**

-sourcepath 指定源文件的搜索路径，多个路径间用“；”分隔。注意，只有当向 javadoc 传入 package 名时，才需要-sourcepath。如果未指定-sourcepath，javadoc 会把 -classpath 作为搜索路径，如果-classpath 也没有指定，则当前路径会被当作搜索路径。

例：

javadoc -sourcepath C:\user1\src;C:\user2\src com.mypackage

-classpath 指定 javadoc 搜索文档中引用类的类路径，多个路径间用“；”分隔。如果未指定-classpath，javadoc 把-sourcepath 指定的路径作为引用类的搜索路径。

例：

javadoc -classpath \user\lib -sourcepath \user\src com.mypackage

5）**-version 和 -author**

-version 和 -author 用于控制生成文档时是否包含@version 和@author 指定的内容。不加这两个参数的情况下，生成的文档中不包含版本和作者信息。

例：

javadoc -classpath \user\lib -version -author com.mypackage

6）**-windowtitle 和-charset**

-windowtitle 标记指定添加到 HTML 标记<title>中的文字。这些文字将显示在浏览器的标题栏。

例：

javadoc -windowtitle "help of myPackage" com.mypackage

-charset 指定生成的 HTML 页的字符集。比如，如果希望生成 HTML 页正确显示中文，就需要指定这个标记。

例：

javadoc -charset "gb2312 " mypackage

7）**-verbose**

指定是否显示 javadoc 运行时的详细信息。

例：

javadoc -classpath \user\lib -verbose com.mypackage

8） **JavaDoc 工具组合示例**

例：

JavaDoc -public -charset "Shift_JIS" -windowtitle "Fujitsu XML Enc Wrapper" -doctitle "Wrapper Interfaces OF Fujitsu XML Enc System [By FNST]" -d ./Docs/ -verbose–classpath G:/working/library;c:/systemlib -sourcepath G:/working com.meritit.xmldsig.enc.wrapper com. meritit.xmldsig.enc.wrapper.types

6.5 参考资料

- 《Code Conventions for the Java™ Programming Language》[Sun 标准 JAVA 编码规范]
  http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html
- 《JAVA 程序编码规范》[IBM DeveloperWorks 网站资料]
  http://www-900.ibm.com/developerWorks/cn/java/java_standard/index.shtml
- 《JAVA 核心技术（卷 1）》 [Sun 公司核心技术丛书]
  http://download.csdn.net/detail/ysjian_pingcx/6908441
- 《Effective Java》中文版 第 2 版 [Sun 公司核心技术丛书]
  http://download.csdn.net/detail/ysjian_pingcx/6844135