Java 语言编码规范

修订记录

日期	修订版本	修改章节	修改描述	作者
2017-02-04	V1.0.0	首次创建	首次创建	乔广

目　录

1.前言

1.1 规范编写的目的和意义

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。

1.2 本规范的适用范围

本规范适用于使用 Java 语言编程的产品和项目。

2.JAVA 源文件的组织结构

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.jumore.jrt;
```

2.2 JAVA 源文件的内部结构

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。

- **版权信息**

版权信息应在 JAVA 文件的开头，其他不需要出现在 JavaDoc 中的信息也可以包含在这里。

例：

```
/**
 * ...
 * Copyright (c) 2017, JUMORE Co.,Ltd. All Rights Reserved.
 * ...
 */
```

- **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，而且按照字母顺序排列。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。

例：

```
package com.jumore.jrt;

import java.io.InputStream;
import java.io.OutputStream;
```

- **Class**

类的注释一般是用来解释类的，建议使用文档注释。

例：

```
/**
 * ClassName: JumorePay
 * Function: 解决聚贸支付业务
 *
 * @author joe
 * @date 2017年2月3日 下午4:10:06
 * @version v1.0.0
 * @see
 */
```

接下来是类的定义，有可能包含了 extends 和 implements。

例：

```
public class JumorePay extends Jumore implements Pay{

}
```

- **Class Fields**

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。注释成员变量时，建议使用文档注释。

例：

```
/**
 * field:成员变量
 */
public int field = 0;
```

- **存取（get、set）方法**

例：

```
/**
 * name 姓名
 *
 * @return the name
 */
public String getName() {
    return name;
}

/**
 * @param name the name to set
 */
public void setName(String name) {
    this.name = name;
}
```

- **构造函数**

构造函数应该按照参数数目的递增顺序进行书写。

例：

```
/**
 * Creates a new instance of JumorePay
 *
 */
public JumorePay() {

}

/**
 * Creates a new instance of JumorePay
 *
 * @param money
 */
public JumorePay(BigDecimal money){

}
```

- **类方法**

  对于类的具体方法，应将功能相似的方法放置在一起。

  例：

```
/**
 * pay:pay for order
 * @param money
 */
public void pay(BigDecimal money){
    // code
}
```

3.命名规则

3.1 JAVA 源文件的命名

JAVA 源文件名必须和源文件中所定义的类的类名相同。

3.2 Package 的命名

Package 名的第一部分应是小写 ASCII 字符，并且是顶级域名之一，通常是 com、edu、gov、mil、net、org 或由 ISO 标准 3166、1981 定义的国家唯一标志码。Package 名的后续部分由各组织内部命名规则决定，内部命名规则指定了各组件的目录名，所属部门名、项目名等。

例：

```
package com.jumore.jrt.pay;
```

3.3 Class/Interface 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。Interface 的命名规则与 Class 相同。

例：

```
public interface Pay {
}
public class JumorePay extends Jumore implements Pay {
}
public class PayException extends Exception {
}
```

3.4 常量的命名

常量名的每个单词的首字母大写，其余单词小写，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;
```

3.5 变量的命名

3.5.1 普通变量

普通变量名的首字母小写，其它每个单词的首字母大写。命名时应该使其简短而又有特定含义，简洁明了的向使用者展示其使用意图。

例：

```
float floatWidth = 0.0;
```

3.5.2 约定变量

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 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’;
```

3.6 方法的命名

方法名的第一个单词应是动词，并且首字母小写，其它每个单词首字母大写。

例：

```
public void sayHello(String name){

}
```

3.7 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，并且首字母小写，其它每个单词首字母大写。如果可能的话，选择和需要赋值的字段一样的名字。

例：

```
public void sayHello(String name){

}
```

4.样式结构

4.1 整体样式

4.1.1 缩进单位

一个缩进单位为四个空格，缩进排版时以缩进一个单位为最小缩进量。不要使用制表符（Tab 键），因为不同的系统对它的解释不尽相同。

4.1.2 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。

例：

```
public void sayHello(String name){
    if (StringUtils.isEmpty(name)) {
        name = “joe”;
    }

    System.out.println(“Hello ” + name);
}
```

- **对齐**

若干语句在逻辑上属于同一层次时，这些语句应对齐。

例：

```
public void sayHello(String name){
    String str = “Hello ”;
    str = str + name;
    System.out.println(str);
}
```

4.1.3 行宽

Java 代码行限制在 140 个字符之内，多余部分应换行。

4.1.4 断行规则(codeformater 辅助)

当一句完整的语句大于 140 个字符时需要断行，断行时，应遵循下面规则。

- **在逗号后换行**

  例：

```
public void sayHello(String name, int age, String address,
        String country){

}
```

- **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**

  错误示例：

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

  应改为：

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

- **换行时尽量选择高层次的地方进行换行**

  错误示例：

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

  应改为：

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

- **在使用上述的规则换行后对齐时，如果次行的长度大于 140 个字符，应改用两个单位的缩进来代替层次对齐**

  例：

```
public void sayHello(String name, int age, String address,
        String country, String longName1,
        String longName2, String longName3){

}
```

4.1.5 空白的使用

- **空格字符的使用**

1) 关键字和括号()之间要用空格隔开

例：

```
while (condition) {
    // code
}

if (condition) {
    // code
}
```

2) 参数列表中逗号的后面应该使用空格

例：

```
public void sayHello(String name, int age, String address){

}
```

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 - longName5)
    + 4 * longName6;
```

4) 强制类型转换后应该跟一个空格

例：

```
sayHello(name, (int) age, (String) address);
```

5) 左括号右边和右括号左边不能有空格

错误示例：

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

应改为：

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

6) 方法名与其参数列表的左括号之间不能有空格

错误示例：

```
sayHello (name, age, address);
```

应改为：

```
sayHello(name, age, address);
```

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("--")

错误示例：

```
var --;
```

应改为：

```
var--;
```

- **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) 一个源文件的两个片段(section)之间用两个空白行

例：

```
/**
 * Copyright (c) 2017, JUMORE Co.,Ltd. All Rights Reserved.
 */


package com.jumore.jrt;
```

2) 两个类声明或接口声明之间使用两个空白行

例：

```
public class JumorePay{
    // code
}


public class JumoreOrder{
    // code
}
```

3) 两个方法的声明之间使用一个空白行

例：

```
public class JumorePay{
    public void pay() {
        // code
    }

    public void jumore() {
        // code
    }
}
```

4) 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：

```
public class JumorePay{
    public void pay() {
        int a = 0;
        int b = 0;

        varC = a + b;
    }
}
```

5) 块注释或单行注释之前使用一个空白行

例：

```
public class JumorePay{
    public void pay() {
        if (condition) {

            /* comment */
            varA = varB + varC;
        }
    }
}
```

6) 一个方法内的两个逻辑段之间应该用一个空白行

例：

```
public class JumorePay{
    public String pay() {
        varA = payStep1();
        varB = payStep2();

        return varA + varB;
    }
}
```

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，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
 */
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，且单行注释之前应该有一个空白行。

例：

```
if(condition){

    /* comment */
    ...
}
```

- 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。

- 文件注释

例：

```
/**
 * Project Name:com.jumore
 * File Name:JumorePay.java
 * Package Name:com.jumore.jrt
 * Copyright (c) 2017, JUMORE Co.,Ltd. All Rights Reserved.
 *
 * @author joe
 * @date 2017年2月6日 下午1:06:55
 */
```

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。

- 类注释

  例：

```
/**
 * Function: jumore支付业务处理类
 *
 * @author joe
 * @date 2017年2月6日 下午1:06:55
 * @version 1.0.0
 * @see Pay
 */
public class JumorePay {
    // cede
}
```

- 字段，变量注释

  例：

```
/**
 * money： 支付金额
 */
private BigDecimal money;

/* 支付账户 */
private String account;
```

- 构造方法注释
  例：

```
/**
 * Creates a new instance of JumorePay
 *
 * @param money 支付金额
 */
public JumorePay(BigDecimal money) {
    // code
}
```

- 方法注释
  例：

```
/**
 * pay:支付方法
 *
 * @author joe
 * @date 2017年2月6日 下午1:25:09
 * @param money    支付金额
 * @param account  支付账户
 * @return 支付是否成功
 */
public boolean pay(BigDecimal money, String account){
    // code
    return true;
}
```

- getter 注释
  例：

```
/**
 * money 支付金额
 *
 * @return the money
 */
public BigDecimal getMoney() {
    return money;
}
```

- setter 注释

  例：

```
/**
 * @param money the money to set
 */
public void setMoney(BigDecimal money) {
    this.money = money;
}
```

4.3 声明

4.3.1 变量的声明

- 变量声明，名称缩写规则：

  1 将类型首字母小写，整个单词作为变量名

  例：

```
String string = “hello”;
```

  2 取类型单词首字母小写作为变量名

  例：

```
JumorePay jp = new JumorePay();
```

- 一行只声明一个变量

  错误示例：

```
int a = 0, b = 0;
```

应改为：

```
int a = 0;
int b = 0;
```

- 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，在构造函数中一定要初始化

  错误示例：

```
int a;
```

应改为：

```
int a = 0;
```

- 临时变量放在其作用域内声明

  错误示例：

```
int varA = 0;

if (condition) {
    varA = methodA();
    methodB(varB);
}
```

应改为：

```
if (condition) {
    int varA = 0;

    varA = methodA();
    methodB(varB);
}
```

- 声明应集中放在作用域的顶端

  错误示例：

```
if (condition) {
    int varA = 0;

    varA = methodA();

    int varB = 0;

    methodB(varB);
}
```

应改为：

```
if (condition) {
    int varA = 0;
    int varB = 0;

    varA = methodA();
    methodB(varB);
}
```

- 避免声明的局部变量覆盖上一级声明的变量
  错误示例：

```
int counter = 0;

if (condition) {
    int counter = 0;

    counter = methodA();
}
```

  应改为：

```
int counter = 0;

if (condition) {
    int cnt = 0;

    cnt = methodA();
}
```

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

4.4 语句

4.4.1 简单语句

- 每行至多包含一条完整语句
  错误示例：

```
varA++; varB++;
```

  应改为：

```
varA++;
varB++;
```

- 在没有必要的情况下，不要在 return 语句中使用括号
  错误示例：

```
return (0);
```

  应改成：

```
return 0;
```

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 (condition) {
    // code
}


if (condition) {
    // code
} else if (condition) {
    // code
} else {
    // code
}
```

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 A:
        // code
        break;
    case B:
        // code
        /* falls through */
    default:
        // code
        break;
}
```

3. 循环语句

例：

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


    while (condition) {
        Sentences;
    }


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

4. Try-Catch 结构语句

例：

```
try {
    Sentences;
} catch (Exception e) {
    Sentences;
}


try {
    Sentences;
} catch (Exception e) {
    Sentences;
} finally {
    Sentences;
}
```