# 软件编码规范说明书

# 酒店会员管理系统

**版本：v1.0**

**团队：少年先疯队**

**日期：2018.06.08**

# **Java编码规范说明书**

# 1 介绍(Introduction)

## 1.1 为什么要有编码规范(Why Have Code Conventions)

编码规范对于程序员而言尤为重要，有以下几个原因：

- 一个软件的生命周期中，80%的花费在于维护  
- 几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护  
- 编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码  
- 如果你将源码作为产品发布，就需要确任它是否被很好的打包并且清晰无误，一如你已构建的其它任何产品

# 2 文件名(File Names)

这部分列出了常用的文件名及其后缀。

## 2.1 文件后缀(File Suffixes)

Java程序使用下列文件后缀：

|  |  |
| --- | --- |
| 文件类别 | 文件后缀 |
| Java源文件 | .java |
| Java字节码文件 | .class |

## 2.2 常用文件名(Common File Names)

常用的文件名包括：

|  |  |
| --- | --- |
| 文件名 | 用途 |
| GNUmakefile | makefiles的首选文件名。我们采用gnumake来创建（build）软件。 |
| README | 概述特定目录下所含内容的文件的首选文件名 |

# 3 文件组织(File Organization)

一个文件由被空行分割而成的段落以及标识每个段落的可选注释共同组成。超过2000行的程序难以阅读，应该尽量避免。"Java源文件范例"提供了一个布局合理的Java程序范例。

## 3.1 Java源文件(Java Source Files)

每个Java源文件都包含一个单一的公共类或接口。若私有类和接口与一个公共类相关联，可以将它们和公共类放入同一个源文件。公共类必须是这个文件中的第一个类或接口。

Java源文件还遵循以下规则：

- 开头注释（参见"[开头注释](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html" \l "311)"）  
- 包和引入语句（参见"[包和引入语句](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html#312)"）  
- 类和接口声明（参见"[类和接口声明](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html#313)"）

3.1.1 开头注释(Beginning Comments)

所有的源文件都应该在开头有一个C语言风格的注释，其中列出类名、版本信息、日期和版权声明：

/\*

\* Classname

\*

\* Version information

\*

\* Date

\*

\* Copyright notice

\*/

### 3.1.2 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;

### 3.1.3 类和接口声明(Class and Interface Declarations)

下表描述了类和接口声明的各个部分以及它们出现的先后次序。参见"[Java源文件范例](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html" \l "b1)"中一个包含注释的例子。

|  |  |  |
| --- | --- | --- |
|  | 类/接口声明的各部分 | 注解 |
| 1 | 类/接口文档注释(/\*\*……\*/) | 该注释中所需包含的信息，参见"[文档注释](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html#52)" |
| 2 | 类或接口的声明 |  |
| 3 | 类/接口实现的注释(/\*……\*/)如果有必要的话 | 该注释应包含任何有关整个类或接口的信息，而这些信息又不适合作为类/接口文档注释。 |
| 4 | 类的(静态)变量 | 首先是类的公共变量，随后是保护变量，再后是包一级别的变量(没有访问修饰符，access modifier)，最后是私有变量。 |
| 5 | 实例变量 | 首先是公共级别的，随后是保护级别的，再后是包一级别的(没有访问修饰符)，最后是私有级别的。 |
| 6 | 构造器 |  |
| 7 | 方法 | 这些方法应该按功能，而非作用域或访问权限，分组。例如，一个私有的类方法可以置于两个公有的实例方法之间。其目的是为了更便于阅读和理解代码。 |

# 4 缩进排版(Indentation)

4个空格常被作为缩进排版的一个单位。缩进的确切解释并未详细指定(空格 vs. 制表符)。一个制表符等于8个空格(而非4个)。

## 4.1 行长度(Line Length)

尽量避免一行的长度超过80个字符，因为很多终端和工具不能很好处理之。

注意：用于文档中的例子应该使用更短的行长，长度一般不超过70个字符。

## 4.2 换行(Wrapping Lines)

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：

- 在一个逗号后面断开  
- 在一个操作符前面断开  
- 宁可选择较高级别(higher-level)的断开，而非较低级别(lower-level)的断开  
- 新的一行应该与上一行同一级别表达式的开头处对齐  
- 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。

以下是断开方法调用的一些例子：

someMethod(longExpression1, longExpression2, longExpression3,

longExpression4, longExpression5);

var = someMethod1(longExpression1,

someMethod2(longExpression2,

longExpression3));

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; //PREFFER

longName1 = longName2 \* (longName3 + longName4

- longName5) + 4 \* longname6; //AVOID

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

//CONVENTIONAL INDENTATION

someMethod(int anArg, Object anotherArg, String yetAnotherArg,

Object andStillAnother) {

...

}

//INDENT 8 SPACES TO AVOID VERY DEEP INDENTS

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，因为常规缩进(4个空格)会使语句体看起来比较费劲。比如：

//DON’T USE THIS INDENTATION

if ((condition1 && condition2)

|| (condition3 && condition4)

||!(condition5 && condition6)) { //BAD WRAPS

doSomethingAboutIt(); //MAKE THIS LINE EASY TO MISS

}

//USE THIS INDENTATION INSTEAD

if ((condition1 && condition2)

|| (condition3 && condition4)

||!(condition5 && condition6)) {

doSomethingAboutIt();

}

//OR USE THIS

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 注释(Comments)

Java程序有两类注释：实现注释(implementation comments)和文档注释(document comments)。实现注释是那些在C++中见过的，使用/\*...\*/和//界定的注释。文档注释(被称为"doc comments")是Java独有的，并由/\*\*...\*/界定。文档注释可以通过javadoc工具转换成HTML文件。

实现注释用以注释代码或者实现细节。文档注释从实现自由(implementation-free)的角度描述代码的规范。它可以被那些手头没有源码的开发人员读懂。

注释应被用来给出代码的总括，并提供代码自身没有提供的附加信息。注释应该仅包含与阅读和理解程序有关的信息。例如，相应的包如何被建立或位于哪个目录下之类的信息不应包括在注释中。

在注释里，对设计决策中重要的或者不是显而易见的地方进行说明是可以的，但应避免提供代码中己清晰表达出来的重复信息。多余的的注释很容易过时。通常应避免那些代码更新就可能过时的注释。

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，考虑一下重写代码使其更清晰。

注释不应写在用星号或其他字符画出来的大框里。注释不应包括诸如制表符和回退符之类的特殊字符。

## 5.1 实现注释的格式(Implementation Comment Formats)

程序可以有4种实现注释的风格：块(block)、单行(single-line)、尾端(trailing)和行末(end-of-line)。

### 5.1.1 块注释(Block Comments)

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。

块注释之首应该有一个空行，用于把块注释和代码分割开来，比如：

/\*

\* Here is a block comment.

\*/

块注释可以以/\*-开头，这样indent(1)就可以将之识别为一个代码块的开始，而不会重排它。

/\*-

\* Here is a block comment with some very special

\* formatting that I want indent(1) to ignore.

\*

\* one

\* two

\* three

\*/

注意：如果你不使用indent(1)，就不必在代码中使用/\*-，或为他人可能对你的代码运行indent(1)作让步。

参见"[文档注释](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html" \l "52)"

### 5.1.2 单行注释(Single-Line Comments)

短注释可以显示在一行内，并与其后的代码具有一样的缩进层级。如果一个注释不能在一行内写完，就该采用块注释(参见"[块注释](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html" \l "511)")。单行注释之前应该有一个空行。以下是一个Java代码中单行注释的例子：

if (condition) {

/\* Handle the condition. \*/

...

}

### 5.1.3 尾端注释(Trailing Comments)

极短的注释可以与它们所要描述的代码位于同一行，但是应该有足够的空白来分开代码和注释。若有多个短注释出现于大段代码中，它们应该具有相同的缩进。

以下是一个Java代码中尾端注释的例子：

if (a == 2) {

return TRUE; /\* special case \*/

} else {

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

}

### 5.1.4 行末注释(End-Of-Line Comments)

注释界定符"//"，可以注释掉整行或者一行中的一部分。它一般不用于连续多行的注释文本；然而，它可以用来注释掉连续多行的代码段。以下是所有三种风格的例子：

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(Documentation Comments)

注意：此处描述的注释格式之范例，参见"[Java源文件范例](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html" \l "b1)"

若想了解更多，参见"How to Write Doc Comments for Javadoc"，其中包含了有关文档注释标记的信息(@return, @param, @see)：

<http://java.sun.com/javadoc/writingdoccomments/index.html>

若想了解更多有关文档注释和javadoc的详细资料，参见javadoc的主页：

<http://java.sun.com/javadoc/index.html>

文档注释描述Java的类、接口、构造器，方法，以及字段(field)。每个文档注释都会被置于注释定界符/\*\*...\*/之中，一个注释对应一个类、接口或成员。该注释应位于声明之前：

/\*\*

\* The Example class provides ...

\*/

public class Example { ...

注意顶层(top-level)的类和接口是不缩进的，而其成员是缩进的。描述类和接口的文档注释的第一行(/\*\*)不需缩进；随后的文档注释每行都缩进1格(使星号纵向对齐)。成员，包括构造函数在内，其文档注释的第一行缩进4格，随后每行都缩进5格。

若你想给出有关类、接口、变量或方法的信息，而这些信息又不适合写在文档中，则可使用实现块注释(见5.1.1)或紧跟在声明后面的单行注释(见5.1.2)。例如，有关一个类实现的细节，应放入紧跟在类声明后面的实现块注释中，而不是放在文档注释中。

文档注释不能放在一个方法或构造器的定义块中，因为Java会将位于文档注释之后的第一个声明与其相关联。

# 6 声明(Declarations)

## 6.1 每行声明变量的数量(Number Per Line)

推荐一行一个声明，因为这样以利于写注释。亦即，

int level; // indentation level

int size; // size of table

要优于，

int level, size;

不要将不同类型变量的声明放在同一行，例如：

int foo, fooarray[]; //WRONG!

注意：上面的例子中，在类型和标识符之间放了一个空格，另一种被允许的替代方式是使用制表符：

int level; // indentation level

int size; // size of table

Object currentEntry; // currently selected table entry

## 6.2 初始化(Initialization)

尽量在声明局部变量的同时初始化。唯一不这么做的理由是变量的初始值依赖于某些先前发生的计算。

## 6.3 布局(Placement)

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。）不要在首次用到该变量时才声明之。这会把注意力不集中的程序员搞糊涂，同时会妨碍代码在该作用域内的可移植性。

void myMethod() {

int int1 = 0; // beginning of method block

if (condition) {

int int2 = 0; // beginning of "if" block

...

}

}

该规则的一个例外是for循环的索引变量

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

避免声明的局部变量覆盖上一级声明的变量。例如，不要在内部代码块中声明相同的变量名：

int count;

...

myMethod() {

if (condition) {

int count = 0; // AVOID!

...

}

...

}

## 6.4 类和接口的声明(Class and Interface Declarations)

当编写类和接口是，应该遵守以下格式规则：

- 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  
- 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

class Sample extends Object {

int ivar1;

int ivar2;

Sample(int i, int j) {

ivar1 = i;

ivar2 = j;

}

int emptyMethod() {}

...

}

- 方法与方法之间以空行分隔

# 7 语句(Statements)

## 7.1 简单语句(Simple Statements)

每行至多包含一条语句，例如：

argv++; // Correct

argc--; // Correct

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

## 7.2 复合语句(Compound Statements)

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。

- 被括其中的语句应该较之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；右大括号"}"应另起一行并与复合语句首行对齐。  
- 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。这样便于添加语句而无需担心由于忘了加括号而引入bug。

## 7.3 返回语句(return Statements)

一个带返回值的return语句不使用小括号"()"，除非它们以某种方式使返回值更为显见。例如：

return;

return myDisk.size();

return (size ? size : defaultSize);

## 7.4 if，if-else，if else-if else语句(if, if-else, if else-if else Statements)

if-else语句应该具有如下格式：

if (condition) {

statements;

}

if (condition) {

statements;

} else {

statements;

}

if (condition) {

statements;

} else if (condition) {

statements;

} else{

statements;

}

注意：if语句总是用"{"和"}"括起来，避免使用如下容易引起错误的格式：

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

statement;

## 7.5 for语句(for Statements)

一个for语句应该具有如下格式：

for (initialization; condition; update) {

statements;

}

一个空的for语句(所有工作都在初始化，条件判断，更新子句中完成）应该具有如下格式：

for (initialization; condition; update);

当在for语句的初始化或更新子句中使用逗号时，避免因使用三个以上变量，而导致复杂度提高。若需要，可以在for循环之前(为初始化子句)或for循环末尾(为更新子句)使用单独的语句。

## 7.6 while语句(while Statements)

一个while语句应该具有如下格式

while (condition) {

statements;

}

一个空的while语句应该具有如下格式：

while (condition);

## 7.7 do-while语句(do-while Statements)

一个do-while语句应该具有如下格式：

do {

statements;

} while (condition);

## 7.8 switch语句(switch Statements)

一个switch语句应该具有如下格式：

switch (condition) {

case ABC:

statements;

/\* falls through \*/

case DEF:

statements;

break;

case XYZ:

statements;

break;

default:

statements;

break;

}

每当一个case顺着往下执行时(因为没有break语句)，通常应在break语句的位置添加注释。上面的示例代码中就包含注释/\* falls through \*/。

## 7.9 try-catch语句(try-catch Statements)

一个try-catch语句应该具有如下格式：

try {

statements;

} catch (ExceptionClass e) {

statements;

}

一个try-catch语句后面也可能跟着一个finally语句，不论try代码块是否顺利执行完，它都会被执行。

try {

statements;

} catch (ExceptionClass e) {

statements;

} finally {

statements;

}

# 8 空白(White Space)

## 8.1 空行(Blank Lines)

空行将逻辑相关的代码段分隔开，以提高可读性。

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  
- 类声明和接口声明之间

下列情况应该总是使用一个空行：

- 两个方法之间  
- 方法内的局部变量和方法的第一条语句之间  
- 块注释（参见"[5.1.1](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html" \l "511)"）或单行注释（参见"[5.1.2](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html#512)"）之前  
- 一个方法内的两个逻辑段之间，用以提高可读性

## 8.2 空格(Blank Spaces)

下列情况应该使用空格：

- 一个紧跟着括号的关键字应该被空格分开，例如：

while (true) {

...

}

注意：空格不应该置于方法名与其左括号之间。这将有助于区分关键字和方法调用。  
- 空白应该位于参数列表中逗号的后面  
- 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");

- for语句中的表达式应该被空格分开，例如：

for (expr1; expr2; expr3)

- 强制转型后应该跟一个空格，例如：

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

myMethod((int) (cp + 5), ((int) (i + 3)) + 1);

# 9 命名规范(Naming Conventions)

命名规范使程序更易读，从而更易于理解。它们也可以提供一些有关标识符功能的信息，以助于理解代码，例如，不论它是一个常量，包，还是类。

|  |  |  |
| --- | --- | --- |
| 标识符类型 | 命名规则 | 例子 |
| 包(Packages) | 一个唯一包名的前缀总是全部小写的ASCII字母并且是一个顶级域名，通常是com，edu，gov，mil，net，org，或1981年ISO 3166标准所指定的标识国家的英文双字符代码。包名的后续部分根据不同机构各自内部的命名规范而不尽相同。这类命名规范可能以特定目录名的组成来区分部门(department)，项目(project)，机器(machine)，或注册名(login names)。 | com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese |
| 类(Classes) | 命名规则：类名是个一名词，采用大小写混合的方式，每个单词的首字母大写。尽量使你的类名简洁而富于描述。使用完整单词，避免缩写词(除非该缩写词被更广泛使用，像URL，HTML) | class Raster; class ImageSprite; |
| 接口(Interfaces) | 命名规则：大小写规则与类名相似 | interface RasterDelegate; interface Storing; |
| 方法(Methods) | 方法名是一个动词，采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。 | run(); runFast(); getBackground(); |
| 变量(Variables) | 除了变量名外，所有实例，包括类，类常量，均采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。变量名不应以下划线或美元符号开头，尽管这在语法上是允许的。 变量名应简短且富于描述。变量名的选用应该易于记忆，即，能够指出其用途。尽量避免单个字符的变量名，除非是一次性的临时变量。临时变量通常被取名为i，j，k，m和n，它们一般用于整型；c，d，e，它们一般用于字符型。 | char c; int i; float myWidth; |
| 实例变量(Instance Variables) | 大小写规则和变量名相似，除了前面需要一个下划线 | int \_employeeId; String \_name; Customer \_customer; |
| 常量(Constants) | 类常量和ANSI常量的声明，应该全部大写，单词间用下划线隔开。(尽量避免ANSI常量，容易引起错误) | static final int MIN\_WIDTH = 4; static final int MAX\_WIDTH = 999; static final int GET\_THE\_CPU = 1; |

# 10 编程惯例(Programming Practices)

## 10.1 提供对实例以及类变量的访问控制(Providing Access to Instance and Class Variables)

若没有足够理由，不要把实例或类变量声明为公有。通常，实例变量无需显式的设置(set)和获取(gotten)，通常这作为方法调用的边缘效应 (side effect)而产生。

一个具有公有实例变量的恰当例子，是类仅作为数据结构，没有行为。亦即，若你要使用一个结构(struct)而非一个类(如果java支持结构的话)，那么把类的实例变量声明为公有是合适的。

## 10.2 引用类变量和类方法(Referring to Class Variables and Methods)

避免用一个对象访问一个类的静态变量和方法。应该用类名替代。例如：

classMethod(); //OK

AClass.classMethod(); //OK

anObject.classMethod(); //AVOID!

## 10.3 常量(Constants)

位于for循环中作为计数器值的数字常量，除了-1,0和1之外，不应被直接写入代码。

## 10.4 变量赋值(Variable Assignments)

避免在一个语句中给多个变量赋相同的值。它很难读懂。例如：

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

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。例如：

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

...

}

应该写成

if ((c++ = d++) != 0) {

...

}

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。例如：

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

应该写成

a = b + c;

d = a + r;

## 10.5 其它惯例(Miscellaneous Practices)

### 10.5.1 圆括号(Parentheses)

一般而言，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，是个好方法。即使运算符的优先级对你而言可能很清楚，但对其他人未必如此。你不能假设别的程序员和你一样清楚运算符的优先级。

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

if ((a == b) && (c == d)) // RIGHT

### 10.5.2 返回值(Returning Values)

设法让你的程序结构符合目的。例如：

if (booleanExpression) {

return true;

} else {

return false;

}

应该代之以如下方法：

return booleanExpression;

类似地：

if (condition) {

return x;

}

return y;

应该写做：

return (condition ? x : y);

### 10.5.3 条件运算符"?"前的表达式(Expressions before '?' in the Conditional Operator)

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;

### 10.5.4 特殊注释(Special Comments)

在注释中使用XXX来标识某些未实现(bogus)的但可以工作(works)的内容。用FIXME来标识某些假的和错误的内容。

# 11 代码范例(Code Examples)

## 11.1 Java源文件范例(Java Source File Example)

下面的例子，展示了如何合理布局一个包含单一公共类的Java源程序。接口的布局与其相似。更多信息参见"[类和接口声明](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html" \l "313)"以及"[文挡注释](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html#52)"。

/\*

\* @(#)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...

}

}

# HTML编码规范说明书

# 1 前言

HTML 作为描述网页结构的超文本标记语言，在百度一直有着广泛的应用。本文档的目标是使 HTML 代码风格保持一致，容易被理解和被维护。

# 2 代码风格

## 2.1 缩进与换行

[强制] 使用 4 个空格做为一个缩进层级，不允许使用 2 个空格 或 tab 字符。

示例：

<ul>

<li>first</li>

<li>second</li>

</ul>

[建议] 每行不得超过 120 个字符。

解释：

过长的代码不容易阅读与维护。但是考虑到 HTML 的特殊性，不做硬性要求。

## 2.2 命名

[强制] class 必须单词全字母小写，单词间以 - 分隔。

[强制] class 必须代表相应模块或部件的内容或功能，不得以样式信息进行命名。

示例：

<!-- good -->

<div class="sidebar"></div>

<!-- bad -->

<div class="left"></div>

[强制] 元素 id 必须保证页面唯一。

解释：

同一个页面中，不同的元素包含相同的 id，不符合 id 的属性含义。并且使用 document.getElementById 时可能导致难以追查的问题。

[建议] id 建议单词全字母小写，单词间以 - 分隔。同项目必须保持风格一致。

[建议] id、class 命名，在避免冲突并描述清楚的前提下尽可能短。

示例：

<!-- good -->

<div id="nav"></div>

<!-- bad -->

<div id="navigation"></div>

<!-- good -->

<p class="comment"></p>

<!-- bad -->

<p class="com"></p>

<!-- good -->

<span class="author"></span>

<!-- bad -->

<span class="red"></span>

[强制] 禁止为了 hook 脚本，创建无样式信息的 class。

解释：

不允许 class 只用于让 JavaScript 选择某些元素，class 应该具有明确的语义和样式。否则容易导致 CSS class 泛滥。

使用 id、属性选择作为 hook 是更好的方式。

[强制] 同一页面，应避免使用相同的 name 与 id。

解释：

IE 浏览器会混淆元素的 id 和 name 属性， document.getElementById 可能获得不期望的元素。所以在对元素的 id 与name 属性的命名需要非常小心。

一个比较好的实践是，为 id 和 name 使用不同的命名法。

示例：

<input name="foo">

<div id="foo"></div>

<script>

// IE6 将显示 INPUT

alert(document.getElementById('foo').tagName);

</script>

## 2.3 标签

[强制] 标签名必须使用小写字母。

示例：

<!-- good -->

<p>Hello StyleGuide!</p>

<!-- bad -->

<P>Hello StyleGuide!</P>

[强制] 对于无需自闭合的标签，不允许自闭合。

解释：

常见无需自闭合标签有 input、br、img、hr 等。

示例：

<!-- good -->

<input type="text" name="title">

<!-- bad -->

<input type="text" name="title" />

[强制] 对 HTML5 中规定允许省略的闭合标签，不允许省略闭合标签。

解释：

对代码体积要求非常严苛的场景，可以例外。比如：第三方页面使用的投放系统。

示例：

<!-- good -->

<ul>

<li>first</li>

<li>second</li>

</ul>

<!-- bad -->

<ul>

<li>first

<li>second

</ul>

[强制] 标签使用必须符合标签嵌套规则。

解释：

比如 div 不得置于 p 中，tbody 必须置于 table 中。

详细的标签嵌套规则参见[HTML DTD](http://www.cs.tut.fi/~jkorpela/html5.dtd)中的 Elements 定义部分。

[建议] HTML 标签的使用应该遵循标签的语义。

解释：

下面是常见标签语义

p - 段落

h1,h2,h3,h4,h5,h6 - 层级标题

strong,em - 强调

ins - 插入

del - 删除

abbr - 缩写

code - 代码标识

cite - 引述来源作品的标题

q - 引用

blockquote - 一段或长篇引用

ul - 无序列表

ol - 有序列表

dl,dt,dd - 定义列表

示例：

<!-- good -->

<p>Esprima serves as an important <strong>building block</strong> for some JavaScript language tools.</p>

<!-- bad -->

<div>Esprima serves as an important <span class="strong">building block</span> for some JavaScript language tools.</div>

[建议] 在 CSS 可以实现相同需求的情况下不得使用表格进行布局。

解释：

在兼容性允许的情况下应尽量保持语义正确性。对网格对齐和拉伸性有严格要求的场景允许例外，如多列复杂表单。

[建议] 标签的使用应尽量简洁，减少不必要的标签。

示例：

<!-- good -->

<img class="avatar" src="image.png">

<!-- bad -->

<span class="avatar">

<img src="image.png">

</span>

2.4 属性

[强制] 属性名必须使用小写字母。

示例：

<!-- good -->

<table cellspacing="0">...</table>

<!-- bad -->

<table cellSpacing="0">...</table>

[强制] 属性值必须用双引号包围。

解释：

不允许使用单引号，不允许不使用引号。

示例：

<!-- good -->

<script src="esl.js"></script>

<!-- bad -->

<script src='esl.js'></script>

<script src=esl.js></script>

[建议] 布尔类型的属性，建议不添加属性值。

示例：

<input type="text" disabled>

<input type="checkbox" value="1" checked>

[建议] 自定义属性建议以 xxx- 为前缀，推荐使用 data-。

解释：

使用前缀有助于区分自定义属性和标准定义的属性。

示例：

<ol data-ui-type="Select"></ol>

# 3 通用

## 3.1 DOCTYPE

[强制] 使用 HTML5 的 doctype 来启用标准模式，建议使用大写的 DOCTYPE。

示例：

<!DOCTYPE html>

[建议] 启用 IE Edge 模式。

示例：

<meta http-equiv="X-UA-Compatible" content="IE=Edge">

[建议] 在 html 标签上设置正确的 lang 属性。

解释：

有助于提高页面的可访问性，如：让语音合成工具确定其所应该采用的发音，令翻译工具确定其翻译语言等。

示例：

<html lang="zh-CN">

## 3.2 编码

[强制] 页面必须使用精简形式，明确指定字符编码。指定字符编码的 meta 必须是 head 的第一个直接子元素。

解释：

见 [HTML5 Charset能用吗](http://www.qianduan.net/html5-charset-can-it.html) 一文。

示例：

<html>

<head>

<meta charset="UTF-8">

......

</head>

<body>

......

</body>

</html>

[建议] HTML 文件使用无 BOM 的 UTF-8 编码。

解释：

UTF-8 编码具有更广泛的适应性。BOM 在使用程序或工具处理文件时可能造成不必要的干扰。

## 3.3 CSS 和 JavaScript 引入

[强制] 引入 CSS 时必须指明 rel="stylesheet"。

示例：

<link rel="stylesheet" href="page.css">

[建议] 引入 CSS 和 JavaScript 时无须指明 type 属性。

解释：

text/css 和 text/javascript 是 type 的默认值。

[建议] 展现定义放置于外部 CSS 中，行为定义放置于外部 JavaScript 中。

解释：

结构-样式-行为的代码分离，对于提高代码的可阅读性和维护性都有好处。

[建议] 在 head 中引入页面需要的所有 CSS 资源。

解释：

在页面渲染的过程中，新的CSS可能导致元素的样式重新计算和绘制，页面闪烁。

[建议] JavaScript 应当放在页面末尾，或采用异步加载。

解释：

将 script 放在页面中间将阻断页面的渲染。出于性能方面的考虑，如非必要，请遵守此条建议。

示例：

<body>

<!-- a lot of elements -->

<script src="init-behavior.js"></script>

</body>

[建议] 移动环境或只针对现代浏览器设计的 Web 应用，如果引用外部资源的 URL 协议部分与页面相同，建议省略协议前缀。

解释：

使用 protocol-relative URL 引入 CSS，在 IE7/8 下，会发两次请求。是否使用 protocol-relative URL 应充分考虑页面针对的环境。

示例：

<script src="//s1.bdstatic.com/cache/static/jquery-1.10.2.min\_f2fb5194.js"></script>

# 4 head

## 4.1 title

[强制] 页面必须包含 title 标签声明标题。

[强制] title 必须作为 head 的直接子元素，并紧随 charset 声明之后。

解释：

title 中如果包含 ASCII 之外的字符，浏览器需要知道字符编码类型才能进行解码，否则可能导致乱码。

示例：

<head>

<meta charset="UTF-8">

<title>页面标题</title>

</head>

## 4.2 favicon

[强制] 保证 favicon 可访问。

解释：

在未指定 favicon 时，大多数浏览器会请求 Web Server 根目录下的 favicon.ico 。为了保证 favicon 可访问，避免 404，必须遵循以下两种方法之一：

在 Web Server 根目录放置 favicon.ico 文件。

使用 link 指定 favicon。

示例：

<link rel="shortcut icon" href="path/to/favicon.ico">

## 4.3 viewport

[建议] 若页面欲对移动设备友好，需指定页面的 viewport。

解释：

viewport meta tag 可以设置可视区域的宽度和初始缩放大小，避免在移动设备上出现页面展示不正常。

比如，在页面宽度小于 980px 时，若需 iOS 设备友好，应当设置 viewport 的 width 值来适应你的页面宽度。同时因为不同移动设备分辨率不同，在设置时，应当使用 device-width 和 device-height 变量。

另外，为了使 viewport 正常工作，在页面内容样式布局设计上也要做相应调整，如避免绝对定位等。关于 viewport 的更多介绍，可以参见 [Safari Web Content Guide的介绍](https://developer.apple.com/library/mac/documentation/AppleApplications/Reference/SafariWebContent/UsingtheViewport/UsingtheViewport.html#//apple_ref/doc/uid/TP40006509-SW26)

# 5 图片

[强制] 禁止 img 的 src 取值为空。延迟加载的图片也要增加默认的 src。

解释：

src 取值为空，会导致部分浏览器重新加载一次当前页面，参考：<https://developer.yahoo.com/performance/rules.html#emptysrc>

[建议] 避免为 img 添加不必要的 title 属性。

解释：

多余的 title 影响看图体验，并且增加了页面尺寸。

[建议] 为重要图片添加 alt 属性。

解释：

可以提高图片加载失败时的用户体验。

[建议] 添加 width 和 height 属性，以避免页面抖动。

[建议] 有下载需求的图片采用 img 标签实现，无下载需求的图片采用 CSS 背景图实现。

解释：

产品 logo、用户头像、用户产生的图片等有潜在下载需求的图片，以 img 形式实现，能方便用户下载。

无下载需求的图片，比如：icon、背景、代码使用的图片等，尽可能采用 CSS 背景图实现。

# 6 表单

## 6.1 控件标题

[强制] 有文本标题的控件必须使用 label 标签将其与其标题相关联。

解释：

有两种方式：

将控件置于 label 内。

label 的 for 属性指向控件的 id。

推荐使用第一种，减少不必要的 id。如果 DOM 结构不允许直接嵌套，则应使用第二种。

示例：

<label><input type="checkbox" name="confirm" value="on"> 我已确认上述条款</label>

<label for="username">用户名：</label> <input type="textbox" name="username" id="username">

## 6.2 按钮

[强制] 使用 button 元素时必须指明 type 属性值。

解释：

button 元素的默认 type 为 submit，如果被置于 form 元素中，点击后将导致表单提交。为显示区分其作用方便理解，必须给出 type 属性。

示例：

<button type="submit">提交</button>

<button type="button">取消</button>

[建议] 尽量不要使用按钮类元素的 name 属性。

解释：

由于浏览器兼容性问题，使用按钮的 name 属性会带来许多难以发现的问题。具体情况可参考[此文](http://w3help.org/zh-cn/causes/CM2001)。

## 6.3 可访问性 (A11Y)

[建议] 负责主要功能的按钮在 DOM 中的顺序应靠前。

解释：

负责主要功能的按钮应相对靠前，以提高可访问性。如果在 CSS 中指定了 float: right 则可能导致视觉上主按钮在前，而 DOM 中主按钮靠后的情况。

示例：

<!-- good -->

<style>

.buttons .button-group {

float: right;

}

</style>

<div class="buttons">

<div class="button-group">

<button type="submit">提交</button>

<button type="button">取消</button>

</div>

</div>

<!-- bad -->

<style>

.buttons button {

float: right;

}

</style>

<div class="buttons">

<button type="button">取消</button>

<button type="submit">提交</button>

</div>

[建议] 当使用 JavaScript 进行表单提交时，如果条件允许，应使原生提交功能正常工作。

解释：

当浏览器 JS 运行错误或关闭 JS 时，提交功能将无法工作。如果正确指定了 form 元素的 action 属性和表单控件的 name属性时，提交仍可继续进行。

示例：

<form action="/login" method="post">

<p><input name="username" type="text" placeholder="用户名"></p>

<p><input name="password" type="password" placeholder="密码"></p>

</form>

[建议] 在针对移动设备开发的页面时，根据内容类型指定输入框的 type 属性。

解释：

根据内容类型指定输入框类型，能获得能友好的输入体验。

示例：

<input type="date">

# 7 多媒体

[建议] 当在现代浏览器中使用 audio 以及 video 标签来播放音频、视频时，应当注意格式。

解释：

音频应尽可能覆盖到如下格式：

MP3

WAV

Ogg

视频应尽可能覆盖到如下格式：

MP4

WebM

Ogg

[建议] 在支持 HTML5 的浏览器中优先使用 audio 和 video 标签来定义音视频元素。

[建议] 使用退化到插件的方式来对多浏览器进行支持。

示例：

<audio controls>

<source src="audio.mp3" type="audio/mpeg">

<source src="audio.ogg" type="audio/ogg">

<object width="100" height="50" data="audio.mp3">

<embed width="100" height="50" src="audio.swf">

</object>

</audio>

<video width="100" height="50" controls>

<source src="video.mp4" type="video/mp4">

<source src="video.ogg" type="video/ogg">

<object width="100" height="50" data="video.mp4">

<embed width="100" height="50" src="video.swf">

</object>

</video>

[建议] 只在必要的时候开启音视频的自动播放。

[建议] 在 object 标签内部提供指示浏览器不支持该标签的说明。

示例：

<object width="100" height="50" data="something.swf">DO NOT SUPPORT THIS TAG</object>

# 8 模板中的 HTML

[建议] 模板代码的缩进优先保证 HTML 代码的缩进规则。

示例：

<!-- good -->

{if $display == true}

<div>

<ul>

{foreach $item\_list as $item}

<li>{$item.name}<li>

{/foreach}

</ul>

</div>

{/if}

<!-- bad -->

{if $display == true}

<div>

<ul>

{foreach $item\_list as $item}

<li>{$item.name}<li>

{/foreach}

</ul>

</div>

{/if}

[建议] 模板代码应以保证 HTML 单个标签语法的正确性为基本原则。

示例：

<!-- good -->

<li class="{if $item.type\_id == $current\_type}focus{/if}">{ $item.type\_name }</li>

<!-- bad -->

<li {if $item.type\_id == $current\_type} class="focus"{/if}>{ $item.type\_name }</li>

[建议] 在循环处理模板数据构造表格时，若要求每行输出固定的个数，建议先将数据分组，之后再循环输出。

示例：

<!-- good -->

<table>

{foreach $item\_list as $item\_group}

<tr>

{foreach $item\_group as $item}

<td>{ $item.name }</td>

{/foreach}

<tr>

{/foreach}

</table>

<!-- bad -->

<table>

<tr>

{foreach $item\_list as $item}

<td>{ $item.name }</td>

{if $item@iteration is div by 5}

</tr>

<tr>

{/if}

{/foreach}

</tr>

</table>