# 软件编码规范说明书

## Green Cloud 系统

版本： 1.0
编订：付恩丽
团队：Just do IT!
日期： 2018-6-10

# 目录

# 1. 介绍

## 1.1. 目的

就本团队的 Green Cloud 系统开发做出编码规范，本规范为一套编写高效可靠的代码的标准、约定和指南。它以安全可靠的软件工程原则为基础，使代码易于理解、维护和增强，提高生产效率。同时，将带来更大的一致性，使软件开发团队的效率明显提高。

## 1.2. 范围

本规范应用于采用 J2EE 规范的项目中，所有项目中的 JAVA 代码，JSP 页面均应遵守这个规范。同时，也可作为其它项目的参考。

# 2. Java 代码规范

## 1.3. 注释

注释是软件可读性的具体体现。程序注释量一般占程序编码量的 20%，软件工程要求不少于 20%(20%不是硬性规定)。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。注释将增加代码的清晰度。注释需简洁、明了。注释不应该包括其他的特殊字符。 序言性注释必须存在，在 Java 类和方法的首部都要描述功能、主要参数、特别问题，以提供总概性提示。重要代码处也应该有相应的注释，提示阅读者。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。

### 1.3.1. 注释的类型

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

（1）实现注释(implementation comments)

实现注释是使用/*...*/和//界定的注释。实现注释用以注释代码或者实现细节。

（2）文档注释(document comments)

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。

（3）编写注释的注意事项

编写注释时应该注意：

- 频繁的注释有时反而使代码质量降低。

- 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。
- 注释不应包括诸如制表符和回退符之类的特殊字符。

## 1.3.2. 对注释的约束

（1）文档头部注释

在每个.java 文档的开始部分，应以如下格式加入对文档的注释：

```
/*
 * @(#) ${project_name} ${file_name}      ${date}
 *
 * Copyright ${year} CDSF Corporation, Inc. All rights reserved.
 */
```

${ project_name }　　项目名称

${ file_name }　　文件名称

${date}　　创建时间

${ year }　　年份类、接口注释

（2）类注释

可以被 Javadoc 处理，生成 HTML 文件。以‘/**’开头，‘*/’结尾。类、接口注释不能放在方法或程序块内. 类注释中当包含版本和作者信息:

```
/**
 * class description
 * @author ${user}
 * @createtime ${date} ${time}
 * @version 1.0
 * @see JDK1.6
 **/
```

${user}　作者

${date}　创建日期

${time}　创建时间

（3）方法注释

依据标准 JavaDoc 规范对方法进行注释，以明确该方法功能、作用、各参数含义、返回值以及该方法的创建者等。复杂的算法用/**/在方法内注解出。

参数注释时当注明其取值范围等；返回值当注释出失败、错误、异常时的返回情况；异常当注释出什么情况、什么时候、什么条件下会引发什么样的异常。

方法注释示例：

```
/**
 * 执行查询
 * @author ${user}
 * @creaetime ${date} ${time}
```

```
     * @param sql  标准的  SQL  语句
     * @return ResultSet  结果集，若查询失败则返回  null
     * @throws SQLException  当查询数据库时数据库断开可能引发此异常
     */
    public ResultSet executeQuery(String sql) throws SQLException {
            ……..
    }
```

（4）属性注释

整个注释占据一行，在申明属性的上一行，以‘/**’开头，‘*/’结尾，注释要说明出属性的含义

```
/**
 * 注释说明
 */
int maxCount;
```

（5）行注释

主要用在方法内部，对代码，流程等进行说明。但是整个注释占据一行。例如：

```
……
// 注释
……
```

变量的注释放在代码的同行。例如：

```
……
int m = 4; // 注释
……
```

应对重要的变量加以注释，以说明其含义等；

应对不易理解的分支条件表达式加注释。不易理解的循环，应说明出口条件。过长的方法实现，应将其语句按实现的功能分段加以概括性说明。

（6）常量注释

常量注释与属性注释格式一致。常量必须加以注释，以说明其含义。

## 1.4. 代码组织与风格

### 1.4.1. 基本原则

代码的组织和风格的基本原则是：便于自己的开发，易于与他人的交流。

因个人习惯和编辑器等可以设置和形成自己的风格，但必须前后一致，并符合本规范的基本要求和原则。

### 1.4.2. 缩进

缩进建议以 4 个空格为单位。子功能块当在其父功能块后缩进；当功能块过多而导致缩进过深时当将子功能块提取出来做为子函数。

缩进示例 ：

```
public void methodName() {
    if(some condition) {
        for(…) {
            //some sentences

        }//end for
    }//end if
}
```

### 1.4.3. 对齐

1、关系密切的行应对齐，对齐包括类型、修饰、名称、参数等各部分对齐。

2、连续赋值时当对齐操作符。

3、每一行的长度不应超过屏幕太多，必要时适当换行，换行后最好以运算符打头，并且以下各行均以该语句首行缩进。

4、当控制或循环中的条件比较长时当换行（操作符前）、对齐并注释各条件。

### 1.4.4. 空行

不得存在无规则的空行，比如说连续十个空行。

适当的增加空行，将逻辑相关的代码段分隔开，以提高可读性。

在下列情况下应该有两行空行：

- 同一文件的不同部分之间；
- 在类，接口以及彼此之间。

在下列情况之间应该有一行空行：

- 方法之间；
- 局部变量和它后边的语句之间；
- 块注释或行注释之前
- 方法内的功能逻辑部分之间
- 不同类别 import 语句之间

### 1.4.5. 空格

- 关键字之后要留空格。诸如 public、protected、private、class、interface、abstracts、extends、implements 等关键字之后至少要留一个空格，否则无法辨析关键字。诸如 if、for、while 等关键字之后应留一个空格再跟左括号“(”，以突出关键字。

- 方法名之后不要留空格，紧跟左括号“(”，以与关键字区别。

- “(”向后紧跟，“(” 、“,” “;” 向前紧跟，紧跟处不留空格。

- 参数列表中逗号之后要留空格，如 Function(x, y, z);

- 如果“;”不是一行的结束符号，其后要留空格。
  例如：
  for (initialization; condition; update)。

- 赋值操作符、比较操作符、算术操作符、逻辑操作符、位域操作符，如“=”、“+=” “>=”、“<=”、“+”、“*”、“%”、“&&”、“||”、“<<”,“^”等二元操作符的前后应当加空格。
  例如：

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

while (d++ = s++) {
   n++;
}

System.out.println("size is " + size + "\n");
```

- 一元操作符如“!”、“~”、“++”、“--”、“&”（地址运算符）等前后不加空格。

- 诸如“［］”、“.”、之类操作符前后不加空格。

12. 强制转型后应该跟一个空格。

- 对于表达式比较长的 for 语句和 if 语句，为了紧凑起见可以适当地去掉一些空格。
  例如：
  if ((a<=b) && (c<=d) … …)

## 1.4.6. 长度

为便于阅读和理解，单个函数的有效代码长度当尽量控制在 100 行以内（不包括注释行），当一个功能模块过大时往往造成阅读困难，因此当使用子函数等将相应功能抽取出来，这也有利于提高代码的重用度。

单个类也不宜过大，当出现此类情况时当将相应功能的代码重构到其他类中，通过组合等方式来调用，建议单个类的长度包括注释行不超过 1500 行。

尽量避免使用大类和长方法。

## 1.5. 声明约定

### 1.5.1. 每行声明变量的数量

推荐一行一个声明，因为这样以利于写注释。

- 推荐使用

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

- 避免使用

```
int level, size;
```

不要将不同类型变量的声明放在同一行，

- 避免使用

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

### 1.5.2. 变量的初始化

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。

### 1.5.3. 声明语句的布局

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。 这不仅降低了代码的可读性，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!
```

```
            ...
        }
        ...
    }
```

### 1.5.4. 类和接口的声明

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。

例如：

```
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

## 1.6. 表达式和语句

### 1.6.1. 一般约定

每行应该只有一条语句。

- 推荐使用

```
argv++;
argc--;
```

- 避免使用

```
argv++; argc--;
```

### 1.6.2. 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。

- 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。

### 1.6.3. if-else, if-else if 语句

- if-else, if-else if 语句，任何情况下，都应该有“{”，“}”，格式如下：

```
if (condition) {
    statements;
} else if (condition) {
    statements;
} else {
    statements;
}
```

- 推荐使用

```
if (condition){
    statement;
}
```

- 避免使用

```
if (condition)
    statement;
```

### 1.6.4. for 语句

- for 语句格式如下：

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

- 如果语句为空，则格式如下：

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

注意：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。

### 1.6.5. while 语句

- while 语句格式如下：

```
while (condition) {
    statements;
}
```

- 如果语句为空，则格式如下：

```
while (condition);
```

### 1.6.6. do-while 语句

- do-while 语句格式如下：

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

### 1.6.7. switch 语句

- switch 语句，格式如下：

```
switch (condition) {
    case ABC: statements; /* falls through */
    case DEF: statements; break;
    case XYZ: statements; break;
    default: statements;break;
}
```

注意：

- 每个 switch 里都应包含 default 子语句。
- 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 */。

### 1.6.8. try-catch 语句

- try-catch 语句格式如下：

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
}
```

- try-catch-finally 语句格式如下：

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
} finally {
    statements;
 }
```

## 1.7. 命名规范

### 1.7.1. 命名原则

本章定义了标识符（包括文件名、接口、类、方法、参数、变量等的名称）的命名通用性原则。

（1）原则一：充分表意

标识符应当直观且可以拼读，可望文知意，不必进行“解码”。 标识符最好采用英文单词或其组合，便于记忆和阅读。切忌使用汉语拼音来命名。程序中的英文单词一般不会太复杂，用词应当准确。例如不要把

CurrentValue 写成 NowValue。 标识符的长度应当符合“min-length && max-information”原则。 在表示出必要的信息的前提下，标识的命名应该尽量的简短。例如，例如标识最大值的变量名命名为成 maxValue，而不推荐命名为 maxValueUntilOverflow。 单字符的名字允许使用，常见的如 i, j, k, m, n, x, y, z 等，它们通常可用作函数内的局部变量，如循环计数器等。

用正确的反义词组命名具有互斥意义的变量或相反动作的函数等。

例如：

```
int minValue, maxValue;
int SetValue(…);
int GetValue(…);
```

单词的缩写应谨慎使用。在使用缩写的同时，应该保留一个标准缩写的列表，并且在使用时保持一致。 尽量避免名字中出现数字编号，如 Value1、Value2 等，除非逻辑上的确需要编号。为了防止某一软件库中的一些标识符和其它软件库中的冲突，可以为各种标识符加上能反映软件性质的前缀。

（2）原则二：避免混淆

- 程序中不要出现仅靠大小写区分的相似的标识符。

例如：

```
int x, X; // 变量 x 与 X 容易混淆
void foo(int x); // 函数 foo 与 FOO 容易混淆
void FOO(float x);
```

- 程序中尽量不要出现代表特定含义的、标识符完全相同的局部变量和全局变量，尽管两者的作用域不同而不会发生语法错误，但会使人误解。

（3）原则三：使用正确的词性

- 变量的名字应当使用“名词”或者“形容词＋名词”。

例如：

```
float value;
float oldValue;
float newValue;
```

- 全局函数的名字应当使用“动词”或者“动词＋名词”（动宾词组）。
- 类的成员函数应当只使用“动词”，被省略掉的名词就是对象本身。

例如：

```
DrawBox(); // 全局函数
Box.Draw(); // 类的成员函数
```

## 1.7.2. 对命名的约定

（1）Package 的命名

- 包名必须全部小写。
- 标识符用点号分隔开来。
- 包名中用点号分隔后的最后一个单词应该代表该包下所有类的大致作用。
- 包名用机构(公司)的 Internet 保留域名“反向”后开头 。
- 包命名层次结构：反写域名.项目名称.功能模块.功能描述

例如：公司网站地址：www.cdsf.com

则包名：com.cdsf.cims.interf.kingsand

表示如下含义：

社区信息化系统的一个接口，并且指定的是与谁(金沙公司)的接口。

这里假定 kingsand 代表金沙公司的英文名称。

（2）类 、接口

- 类的名字应该使用名词。
- 每个单词第一个字母应该大写。避免使用单词的缩写，除非它的缩写已经广为人知，如：每个人都知道 HTTP(Hypertext Transfer Protocol)代表的是 WWW 服务程序所用的协议。

例如：

```
Class HelloWorld;
Interface Apple;
```

（3）方法

- 第一个单词一般是动词。
- 第一个字母是小写，但是中间单词的第一个字母是大写。

  例如：

  addUserInfo(); //
- 如果方法返回一个成员变量的值，方法名一般为 get+成员变量名。

  例如：

  getName(); //表示获得成员变量 name 的值
- 如若返回的值是 boolean 变量，一般以 is 作为前缀。

  例如：

  isFirst(); //表示该方法返回值是一个 boolean 型
- 如果方法修改一个成员变量的值，方法名一般为：set + 成员变量名。

  例如：

  setName(); //表示修改成员变量 name 的值
- 如果方法是增加对象信息，则方法名一般为：add + 对象名。

  例如：

  addUserInfo(); //增加用户信息
- 如果方法是删除对象信息，则方法名一般为：del + 对象名。

  例如：

  delUserInfo(UserInfo userInfo); //删除用户信息,其中参数是 UserInfo 类的对象
- 如果方法是根据某个条件删除对象信息，则方法名一般为：del + 对象名 + By+条件字段。

  例如：

  delUserInfoByID(String ID); //删除指定 ID 的用户信息,其中参数为用户 ID
- 如果方法是修改对象信息，则方法名一般为：update + 对象名。

  例如：

  updateUserInfo(UserInfo userInfo); //更新用户信息
- 如果方法是根据某个条件修改对象信息，则方法名一般为：update + 对象名 + By+条件字段。

例如：

updateUserInfoByID(String ID);　　//修改指定 ID 的用户信息,其中参数为用户 ID

- 如果方法是查询对象信息，则方法名一般为：find + 对象名

例如：

findUserInfo();　　//查询所有用户信息

- 如果方法是根据某个条件查询对象信息，则方法名一般为：get + 对象名 + By + 条件字段。

例如：

getUserInfoByID(String ID);　　//查询指定 ID 的用户信息，其中参数为用户 ID

### （4）变量

- 第一个字母小写，中间单词的第一个字母大写。
- 不要用_或&作为第一个字母。
- 尽量使用短而且具有意义的单词。
- 单字符的变量名一般只用于生命期非常短暂的变量。
- 单字符的变量 i, j, k, m, n 一般用于 integers；c, d, e 一般用于 characters。
- 如果变量是集合类的对象，则请参见：集合类变量的命名
- 命名组件采用匈牙利命名法，所有前缀均应遵循同一个组件名称缩写列表。

例如：

String myName;
int i;
int n;
char c;
btnNew;
(btn 是 Button 的缩写)

### （5）常量

- 所有常量名均全部大写，单词间以‘_’隔开。

例如：

int MAX_VALUE;

- Static Final 常量的名字应该都大写，并且指出完整含义。

例如：

public final static int MAX_CONNECT_NUM = 100;
表示：最大连接数

### （6）数组的命名

- 数组应该总是用下面的方式来命名：byte[] buffer; 而不是：byte buffer[];

### （7）集合类变量的命名

- 集合类变量的命名必须以结果集中内容的对象名+该集合类的名字(或缩写)组成。

例如：

Vector　userInfoVec
表示结果集是一个 Vector 其中存放的对象是 UserInfo(用户信息)

ArrayList　userInfoAList
表示结果集是一个 ArrayList 其中存放的对象是 UserInfo(用户信息)

**注意：**如果结果集中存放的不是 UserInfo 的对象，而是该对象的字段，命名规则不变。

## 1.8. 编程惯例

### 1.8.1. 提供对实例以及类变量的访问控制

- 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(get)，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，那么把类的实例变量声明为公有是合适的。

### 1.8.2. 引用类变量和类方法

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。

- 推荐使用

```
classMethod();
AClass.classMethod();
```

- 避免使用

```
anObject.classMethod();
```

### 1.8.3. 变量赋值

避免在一个语句中给多个变量赋相同的值。它很难读懂。

- 推荐使用

```
aObj.fChar = 'c';
bObj.lChar = 'c';
```

- 避免使用

```
aObj.fChar = bObj.lChar = 'c';
```

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。

- 推荐使用

```
if ((c++ = d++) != 0) {
    ...
}
```

- 避免使用

```
if (c++ = d++) {
    ...
```

```
}
```

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。

- 推荐使用

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

- 避免使用

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

### 1.8.4. 其它惯例

（1）圆括号

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。

- 推荐使用

```
if ((a == b) && (c == d))
```

- 避免使用

```
if (a == b && c == d)
```

（2）返回值

设法让你的程序结构符合目的。

- 推荐使用

```
return booleanExpression;
```

- 避免使用

```
if (booleanExpression) {
    return true;
} else {
    return false;
}
```

- 推荐使用

```
return (condition ? x : y);
```

- 避免使用

```
if (condition) {
    return x;
}
return y;
```

（3）条件运算符"?"前的表达式

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。

- 推荐使用

  (x >= 0) ? x : -x;
- 避免使用

  x >= 0 ? x : -x;

## 1.9. 错误处理和异常事件

通常的思想是只对错误采用异常处理：逻辑和编程错误，设置错误，被破坏的数据，资源耗尽，等等。通常的法则是系统在正常状态下以及无重载和硬件失效状态下，不应产生任何异常。异常处理时可以采用适当的日志机制来报告异常，包括异常发生的时刻。不要使用异常实现来控制程序流程结构。

## 1.10. 可移植性

- 尽量不要使用已经被标为不赞成使用的(deprecated)类或方法。
- 如果需要换行的话，尽量用 println 来代替在字符串中使用"\n"。
- 用 java.io.`File.separator` 常量代替路径中的“/”或“\” 。
- 用 java.io.`File`.pathSeptarator 常量代替路径中的“:”或“;”。

## 1.11. 参考文献


- How to Write Doc Comments for the Javadoc. Tool JavadocTool@sun.com, Sun Microsystems Inc. 2000
- Java. Coding Style Guide Achut Reddy, Server Management Tools Group, Sun Microsystems, Inc. May. 30, 2000
- Referrence: http://java.sun.com/j2se/javadoc/writingdoccomments/index.html


# 3. HTML 规范(参考 W3C 部分规范)

## 3.1. HTML 一般需要遵循以下格式

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<head>
```

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>some title</title>
<link rel="stylesheet" href="some.css" type="text/css" />
<script type="text/javascript" src="some.js"></script>
</head>
注意：必须指定一个有意义的<title>，严禁出现“Untitled”或“未命名”之类的<title>。

## 3.2. 注释

<!-- <b>被注释语句</b> --> : HTML 注释,用户在 html 的源码里可见。
<%-- <%=user.getName()%> --%> : Scriptless 注释，用户在 html 源码里不可见。

一般不使用 html 注释，除非是有必要让最终用户看到的内容。

## 3.3. form 属于域的 maxlength

对于 text 类型的输入域，必须根据数据库字段的长度设置相应的 maxlength，例如数据库类型是 VARCHAR(64)，那么 maxlength 是 32（因为中文浏览器对于中文也认为是一个字符）。

## 3.4. <form>表单 name 属性

该表单所需收集的信息的作用或动作+From，例如：fileUploadForm 或 sendMailForm

## 3.5. 含 frame 页面

<frameset>中<frame>的 name 属性命名的格式是①xxx._②xxx_③xxx
①xxx 部分用来标识当前页面隶属于整个系统中的哪一功能模块。
如：属于 ebwebmail 则被表示为 ebwebmail，其它情况依次类推。
②xxx 部分标识当前页面所要完成的功能。
如：完成用户登录的功能则被标识为 login，其它情况依次类推。
③xxx 部分用来用来表示页面在浏览器窗口所处的位置。
处于浏览器窗口的顶部则标识为 top，其它情况依次类推。
例如：ebwebmail_inbox_top.jsp

## 3.6. Width 和 height 的写法也有统一的规范

一般情况下只有一列的表格，width 写在<table> 的标签内，只有一行的表格，height 写在 <table> 的标签内，多行多列的表格，width 和 height 写在第一行或者第一列的 <td> 标签内。总之遵循一条原则：不出现

多于一个的控制同一个单元格大小的 height 和 width, 保证任何一个 width 和 height 都是有效的，也就是你改动代码中任何一个 width 和 height 的数值，都应该在浏览器中看到变化。做到这一条不容易，需要较长时间的练习和思考。

### 3.7. 不同语种的文字之间应该有一个半角空格

不同语种的文字之间应该有一个半角空格，但避头的符号之前和避尾的符号之后除外汉字之间的标点要用全角标点，英文字母和数字周围的括号应该使用半角括号。

### 3.8. 尽量少使用全角空格

请不要在网页中连续出现多于一个的也尽量少使用全角空格（英文字符集下，全角空格会变成乱码），空白应该尽量使用 text-indent, padding, margin, hspace, vspace 以及透明的 gif 图片来实现。

## 4. JSP 规范

### 4.1. Jsp 命名规范

放在相应文件夹里，并且要按照实现的功能来命名。

### 4.2. jsp 头格式

jsp 头部一般需要遵循以下格式：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
 <%@taglib prefix="s" uri="/struts-tags" %> // struts 标签库
```

## 5. JS 规范

### 5.1. JS 命名规范

1. 变量名称开头字母 必须为 小写字母，命名使用骆驼命名规则。
2. 类的命名使用骆驼命名规则，例如：

```
Account, EventHandler
```

3. 常量 *必须* 在对象（类）或者枚举变量的前部声明。枚举变量的命名必须要有实际的意义，并且其成员 *必须* 使用骆驼命名规则或使用大写：

```
var nodeTypes = {
      Element : 1,
      DOCUMENT: 2
}
```

4. 方法的命令 *必须* 为动词或者是动词短语：

```
obj.getSomeValue()
```

5. 类的命名 *必须* 使用混合名称（mixedCase）命名。
6. 通用的变量 *必须* 使用与其名字一致的类型名称：

```
setTopic(topic) // 变量 topic 为 Topic 类型的变量
```

7. 所有的变量名 *必须* 使用英文名称。

## 5.2. JS 文件

放在相应文件夹里，并且要按照实现的功能来命名。

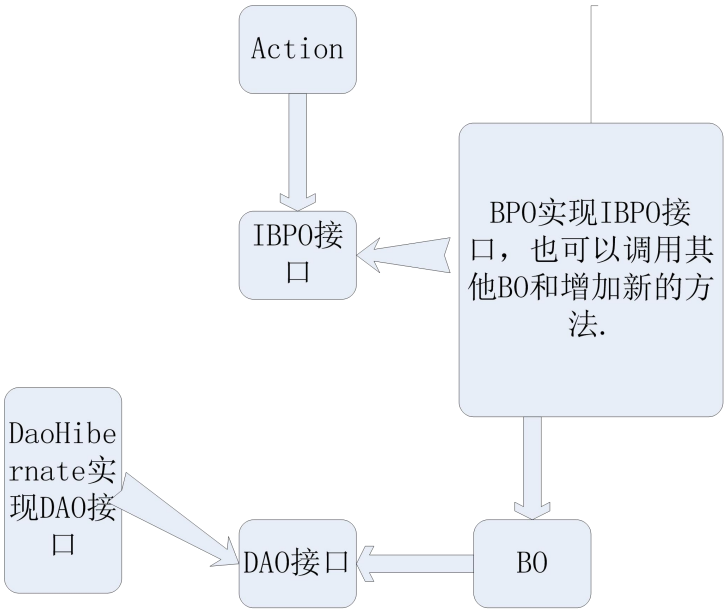
# 6. 其他规范

## 6.1. 限制 session 的使用

在代码中使用 session 需要听取项目经理的意见，项目经理需要在设计文档中登记项目中所有使用到的 session 的名字和作用。

## 6.2. 项目层次调用结构图

描述:Action 调用 IBPO 接口, IBPO 由 BPO 实现，BPO 调用 BO，BO 调用 DAO 接口，DAO 接口由 DAOHIBERNATE 实现.




其中 IBPO 接口类命名规范为 XxxxIBPO.java

BPO 实现类名规范为 XxxxBPO.java

业务逻辑处理类名规范为 XxxxBO.java

Dao 接口类命名规范为 XxxxDaO.java

DaoHibernate 实现类命名规范为 XxxxDaoHibernate.java

第一个是封装的查询枚举字段类名规范为XxxxKey.java
第二个是封装的查询条件类名规范为 XxxxQuery.java

注：Xxxx 表示自己定义的类名

## 6.3. JSP 文件

在尽可能合理的情况下，把业务逻辑从 JSP 中移走。具体于 HTTP 的逻辑（如，对 Cookie 的处理）属于 bean 或支持类中，而不是 JSP 中。也就是说在 jsp 文件中不能出现<%.....%>的代码。而是用标签代替。

## 6.4. JS 文件

js 代码不能以<script type="text/javascript">….</script>的形式写在的 jsp 文件中，而是单独写一个 js 文件，在 jsp 文件中引用。例如：<script type="text/javascript" src="some.js"></script>

## 6.5. Struts Action

Struts Action 类受 Spring 的管理,该类只负责收集页面中的资料和调用 IBPO 接口，不能有处理业务逻辑的代码。

## 6.6. Jdbc

项目中尽量使用 Hibernate 操作 db，jdbc 必须在特定情况下使用，在代码中使用 jdbc 需要听取项目经理的意见。

# 7. Eclipse 中编码规范配置

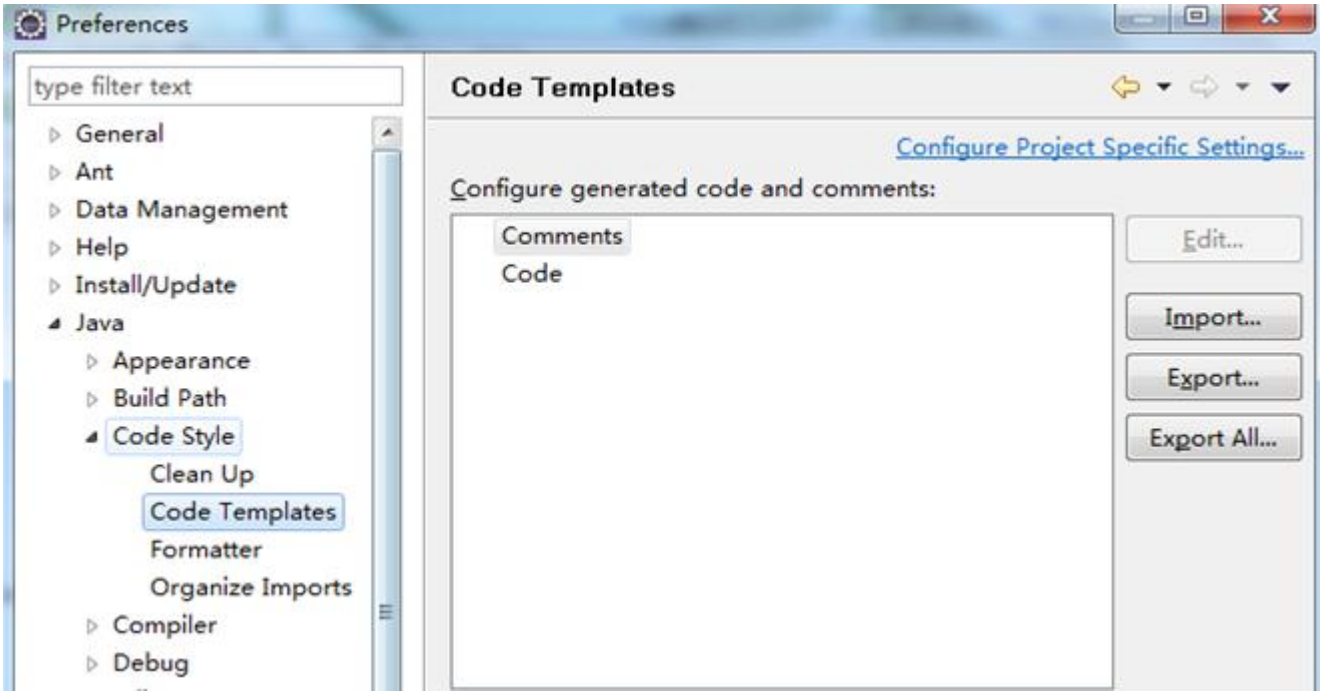
使用 eclipse IDE 轻松规范代码格式，只需按以下步骤对 eclipse 进行配置。

1、将以下两个 XML 文件另存到你的桌面.


cdsf.xml


codetemplates.xml

2、打开 eclipse，选择上面的菜单依次: Window(窗口)->Prefrences(首选项)，弹出如下界面，按图中所示选择左边菜单 Code Templates:




3、点击右边的 import.选择桌面上的 codetemplates.xml 进行导入配置。点击右下角 Apply 应用一下配置。

4、选择左边菜单 Formatter，点击右边的 import.选择桌面上的 cdsf.xml 进行导入配置。点击右下角 Apply 应用一下配置。点击 OK 就完成了整个配置过程。

5、新建一个 java 文件测试一下.

Test.java

```
1  /*
2   * @(#) test Test.java  Nov 11, 2010
3   *
4   * Copyright 2010 CDSF Corporation, Inc. All rights reserved.
5   */
6  package com.cdsf.test;
7  /**
8   *
9   * @author wujianjun
10  * @createtime Nov 11, 2010 11:00:52 AM
11  * @version 1.0
12  * @see JDK1.6
13  **/
14 public class Test {
15
16 }
17
```

6、如何对代码进行格式化？

在 Java 源文件上右键->Source->Format，也可以用快捷键 Ctrl + Shift + F。