**J2EE****编程规范**

|  |  |  |
| --- | --- | --- |
| 文件状态：  [√] 草案  [ ] 正式发布  [ ]正在修改 | 文档类别： | 软件开发编码规范 |
| 文档编号： |  |
| 当前版本： | 1.0.0 |
| 作 者： |  |
| 完成日期： |  |

**版 本 历 史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本/状态** | **作者** | **参与者** | **起止日期** | **备注** |
| 1.0.0 | XXX | XXX |  | 创建文档 |
| 1.0.1 | 孙凡 | 于瑞智 | 20160330 | 加入内容 |
|  |  |  |  |  |

**目 录**

[1. 引言 4](#_Toc425778396)

[2. 命名规范 4](#_Toc425778397)

[3. 文件组织 5](#_Toc425778398)

[4. 缩进排版 7](#_Toc425778399)

[5. 注释 9](#_Toc425778400)

[6. 声明 11](#_Toc425778401)

[7. 语句 13](#_Toc425778402)

[8. 空白 17](#_Toc425778403)

[9. 编程惯例 18](#_Toc425778404)

### 1. 引言

本规范定义基于J2EE源码级应用程序的开发规范。它基于Java程序语言，以Eclipse开发环境 ，规定开发人员在编码过程中所要遵循的原则、标准。它可供应用开发人员及系统设计人员使用。

本规范描述开发人员在编码过程中所要遵循的编码风格、命名及Java语言的一些重要特征。这些都是通过实践经验得出的结论，而不仅仅只是理论。希望这些规范和经验能够规范大家的编程习惯，使我们的代码更加具有可读性、可维护性及标准性。

本规范适用于公司中所有使用基于J2EE框架的Java程序员进行的产品和项目的应用开发。

总的原则：

1. 简单性原则。类定义及方法定义尽量简单。任何类及定义的变量及属性一定要有确切的含义，而且尽量简单。
2. 完备性。需要写出比较完备的注释，说明代码的功能，主要特征，适用的环境，所依赖的包或子系统，前置条件，后置条件等。
3. 一致性。主要指两方面的一致性，1：代码风格的一致性。2：注释和代码的一致性。
4. 代码的健壮性。编写代码的时候必须考虑异常情况。数据校验以及错误处理等。

### 2. 命名规范

命名规范有利于提供程序的易读性，帮助理解代码。命名应当遵循简短明了的原则，在使用者可以根据上下文明确理解其含义的情况下尽可能简短

|  |  |  |
| --- | --- | --- |
| 类型 | 命名规则 | 样例 |
| 包(Package) | 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，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)。 | com.glanway.gone (项目)  com.glanway.ems (项目)  com.glanway.sz.ctl (子机构) |
| 类(Classes) | 类名使用一个名词，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  一个java 源文件必须与其中的外部类文件名保持一致 | Raster  Method |
| 接口(Interfaces) | 同类命名 |  |
| 方法(Methods) | 方法名应该使用动词命名，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  doFilterInternal |
| 变量(VAR) | 除下述常量(类常量)外，所有变量采用大小写混合方式,第一个单词的首字母小写，气候单词的首字母大写 | maxHeight |
| 常量(Constants) | 类常量命名应该全部大写,单词件使用下划线隔开 | MAX\_VALUE |
| Getter/Setter | Getter/Setter 命名必须遵从 property 规范,  以 property 首字母大写 + get/set 前缀命名  Property 与 实例变量名称相同 | private int age;  publicintgetAge() {...}  public void setAge() {...} |

### 3. 文件组织

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.

一个文件内容不应该超过 1500 行, 否则必须调整文件组织

Java 源文件应该按照如下顺序进行组织

- 开头注释

- 包和导入语句

- 类/接口文档注释和类/接口声明

1. 开头注释

所有源文件开头都应该有 Java 块注释，其中主要列出版权声明，可以根据情况添加其他信息, 样例如下:

*/\*  
 \* Copyright (c) 2005, 2014 glanway.com  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \*/*

1. 包和导入语句

非注释行的第一行必须是 包声明

在包声明之后, 含有一个空行, 其后紧跟 导入语句

其中

- 导入语句应该根据包类型和导入类型进行分组, 每个分组之间留一个空行

- 静态导入应该放在导入语句的最后

**package**org.talmud.common.excel.impl;  
  
 **import** org.talmud.common.excel.ExcelException;  
 **import** org.talmud.common.excel.ExcelWriter;  
  
 **import** java.lang.reflect.Array;  
 **import** java.util.Calendar;  
 **import** java.util.Date;  
  
 **import static** org.talmud.common.util.Throwables.*rethrowException*;

1. 类/接口文档注释和类/接口声明

3.1. 在类/接口声明上必须标注相关文档注释, 文档注释必须标明说明,作者,修改者,修改时间

其中

- 说明一般包含功能说明, 使用方式, 修改记录等

- 作者和修改者依次罗列

- 对于通用代码/框架添加的新功能还需要加上增加时间(使用@sine)

3.2. 类声明，一般按照常量，实例变量，构造器，方法的顺序来声明

其中

- 对于常量/实例变量必须在上一行或尾部进行注释说明

- 构造器按照参数由少到多的顺序进行排列

- 方法上必须使用文档注释来说明该方法的功能作用, 以及参数，返回值，异常说明

- 对于行注释和块注释和上一行代码之间必须存在一个空行

- 如果常量过多可以声明在底部，如果为公用常量则应该在常量类中声明

3.3. 完整实例

*/\*\*  
 \* 基础 Service 实现  
 \* 提供基本的 CRUD 操作  
 \* <p/>  
 \* update 2014.7.8 liujichao: 更新错误消息提醒  
 \* update 2014.8.1 yuruizhi: Fix 没有设置BaseDao时产生 NullPointerException  
 \*  
 \** ***@author*** *yangchanghe  
 \** ***@author*** *liujichao  
 \** ***@author*** *yuruizhi  
 \*/* **public abstract class** BaseServiceImpl<E>**extends implements** BaseService<E> {

*/\*\*  
 \* 错误消息  
 \*/***public static final** String ***RECORD\_BEEN\_USED\_MSG*** = **"Records have been used "**;  
  
@Autowired  
**public void** setBaseDao(BaseDao<E>baseDao) {  
**super**.setCrudDao(baseDao);  
 }  
  
*/\*\*  
 \* 将给定的属性名称转换为 下划线写法  
 \* eg: productName -->product\_name  
 \*  
 \** ***@paramproperty*** *需要转换的属性名称  
 \** ***@return*** *转换后的下划线写法名称  
 \*/***protected** String transformDynamicProperty(String property) {  
 String prop = StringUtils.*camelCaseToUnderscore*(property, **false**);  
**return** prop;  
 }

....

}

### 4. 缩进排版

为了使得代码清晰且在所有编辑器中显示效果一致使用4个空格来作为一个缩进排版的单位. （可以在IDE中设置 TAB 为4个空格, 另：实际一个制表符为8个空格)

1. 行长度

应该尽量避免一行的内容长度超过120 个字符（现在浏览器一般120个字符会有提醒)

过长将会造成阅读不便

注：对于文档及注释应该行长度不超过70个字符

1. 换行

当一个表达式无法容纳在一行时，依据如下规则进行换行：

* 在一个逗号后断开
* 在一个操作符前断开
* 选择较高级别表达式断开，而非较低级别表达式
* 新行应该与上一行同一级别的表达式开头对齐
* 如果以上规则会造成代码杂沓或代码都拥挤在右侧则以缩进8个空格开头

**// 和上一行同级表达式开头对齐**

**public** CustomStringBeanPropertyEditor(Class<?> clazz, String property,  
 String[] skipValues) { Assert.notNull(clazz);  
 Assert.notNull(property);  
 **this**.clazz = clazz;  
 **this**.property = property;  
 **this**.skipValues = skipValues;  
}

**// 以缩进8个空格开头**

**private static final synchronized** String findAttribute(PageContext pageContext,  
 HttpServletRequest request, HttpSession session, ServletContext context) {  
 *// 以一个缩进开头的例子* **return null**;  
}

**// if 语句使用8个空格缩进规则**

**if** (StringUtils.*hasText*(name) && StringUtils.*hasText*(title)  
 || StringUtils.*hasText*(name) && StringUtils.*hasText*(subtitle)) {  
 *//* ***TODO SOMETHING***}

**// 三目运算符的换行规则**

*// 方式 1*name = StringUtils.*hasText*(title) ? (name + title) : name;

*// 方式 2*name = StringUtils.*hasText*(title) ? (name + title)  
 : name;

*// 方式 3*name = StringUtils.*hasText*(title)  
 ? (name + title)  
 : name;

### 5. 注释

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：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。

**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。

**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。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)**

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或紧跟在声明后面的单行注释。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

**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

**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

**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://morningspace.51.net/resource/javacodeconv.html" \l "511)"）或单行注释（参见"[5.1.2](http://morningspace.51.net/resource/javacodeconv.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. 编程惯例

9.1 控制对实例变量及类变量的访问

一般不要把实例变量和类变量声明为 public, 而是提供 getter/setter 进行访问

9.2 不要通过类的实例来访问静态变量（类变量）和静态方法（类方法）

obj.staticMethod() // 不要这样

9.3 常量

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。

9.4 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;

9.5 返回值

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);

9.6 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;

9.7 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。

用 TODO 来标示需要处理的工作和内容

9.8 与常量比较

当一个变量与常量进行比较时，将常量前置

var.equals("str") 应替换为 "str".equals(var)

var == 1 应替换为 1 == var

### 10. 项目规范

#### 10.1 项目结构

项目统一使用 maven 管理按照以下结构组织

project

|-- pom.xml (maven 项目配置文件)

|-- src (项目代码文件)

|-- main (项目主代码文件)

|-- java (主代码 -- java 源码)

|-- resources (主代码 - 配置等资源文件)

|-- webapp (web应用相关文件)

|-- test (单元测试文件夹)

|-- java (单元测试 -- java 源码)

|-- resources (单元测试资源)

#### 10.2 包组织

对于 main/java 和 test/java 中 包命名规则如下:

com.glanway.gone

|-- controller/action (spring mvc, struts 控制器)

|-- service (服务层)

|-- system (服务层, system模块接口定义)

|-- impl (服务层, system模块接口实现)

|-- dao

|-- system (Dao 层 system 模块接口定义)

|-- impl (Dao 层 system 模块接口实现)

|-- util (工具类, 辅助类)

对于包命名规则为： 基本包.分类/分层.模块

基准包 (基准包命名参考基本规范中《命名规范》）

#### 10.3 资源组织

main/resources和test/resources 为资源文件

资源文件结构如下

resources

|-- com.glanway.gone.dao.system (相关类对应资源或Mybatis Mapper)

|-- sql (数据库脚本)

|-- support (其他支持配置文件)

|-- 配置文件 (applicationContext.xml,log4j.properties等)

#### 10.4 Web应用文件组织

webapp

|-- js (JS 资源)

|-- css (CSS 资源)

|-- images (图片资源)

|-- favicon.ico

|-- robots.txt

|-- humans.txt

|-- crossdomain.xml

|-- 其他按照约定需要放在 根目录的文件

|-- WEB-INF

|-- web.xml

|-- tags (jsp tag 文件)

|-- tld (jsp 标签定义文件)

|-- urlrewrite.xml

|-- view (视图文件, jsp, freekmarker, .....)

#### 10.5 驼峰式命名补充说明

对于驼峰命名方式（包括首字母大写的大驼峰和首字母小写的驼峰）中，对于缩略词认为是普通单词，使用驼峰写法

eg:

HTTPRequest写作 HttpRequest

ID 写作 Id

DAO 写作 Dao

#### 10.6 接口定义说明

接口中所有定义的常量均省略 public static final 关键字

接口中所有定义的方法均省略 public abstract 关键字

**public interface** CrudDao<E, ID **extends** Serializable> {  
 String ***OFFSET\_PROP*** = **"\_offset"**;  
 String ***MAX\_RESULTS\_PROP*** = **"\_maxResults"**;  
 String ***FILTERS\_PROP*** = **"\_filters"**;  
 String ***SORT\_PROP*** = **"\_sort"**;  
  
 */\*\*  
 \* 删除给定对象  
 \* 删除前建议先判断是否存在, 给出合理提示(是否存在), 而不应该由数据库来决定删除  
 \* 因此这里使用被删除对象作为参数来简单限制, 但是删除时只会使用 id  
 \*  
 \** ***@param e*** *\*/* **void** delete(E e);

}

#### 10.7 方法名前缀后缀要有意义

       方法名前缀：

- init或则initialize   初始化，暗示会做些诸如获取资源等特殊动作   initializeObjectPool()

- destroy     销毁，暗示会做些诸如释放资源的特殊动作    destroyObjectPoolc()

- open        打开                     openConnection()

- close       关闭                     closeConnection()

方法名后缀：

- Listener    响应某种事件的类    PaymentSuccessListener()

- Event     这个类代表了某种事件     PaymentSuccessEvent()

- Servlet 一个Servlet                  PaymentCallbackServlet()

- Factory 生成某种对象工厂的类              PaymentOrderFactory()

- Job 某种按时间运行的任务     PaymentOrderCancelJob()

- Wrapper 这是一个包装类，为了给某个类提供没有的能力   SelectableOrderListWrapper()

- Bean 这是一个POJO                  MenuStateBean()

#### 10.7 Controller 命名

Controller 命名以功能名词 + Controller/Action 结尾命名

Struts: com.glanway.gone.action.system.ConfigAction

Spring MVC: com.glanway.gone.controller.system.ConfigController

#### 10.8 Service 命名

对于Service接口命名以功能/模块名词 + Service 结尾命名

对于 Service 实现一般以 Service + Impl 结尾命名

com.glanway.gone.service.system.ConfigService

com.glanway.gone.service.system.impl.ConfigServiceImpl

#### 10.9 Dao 命名

同 Serivce 命名，但是接口是以"Dao" 结尾

#### 10.10 工具类

工具类定义为： 与具体业务逻辑无关的 java 代码

其中工具类命名为 功能/模块名词 + "Utils" 命名

StringUtils, SubjectUtils

#### 10.11 辅助类

附注类定义为：与具体逻辑相关，用于处理统一逻辑的 java 代码

附注类命名为 功能/模块名词 + "Helper" 命名

FisherHelper, UserFormHelper

#### 10.12常量

对于同一个用途的字面量如果出现多次，或需要使用多次必须定义为常量

#### 10.13 覆写要求

对于需要继承父类,实现接口，复写的方法必须标注 @Overide

#### 10.14泛型使用

对于泛型，必须指定类型参数，除非无法编译通过

List<String> all = Lists.newArrayList();

#### 10.15 异常处理

对于程序中的异常，只允许以下两种处理方式：

1. 直接上抛

2. 转换为 RuntimeException 及其子类上抛

3. catch 后处理

不允许 catch 后 打印完毕，除非不影响正常逻辑

try {

....

}catch(IOException ex) {

LOG.warn("write data fail");

throw convertToRuntimeException(ex);

}

#### 10.16 Controller 与 Service 交互

Controller 与 Service 交互时禁止 Service 返回原本应该控制器处理的结果

**public** Map<String, Object> signin(String username, String password) {  
 Map<String, Object> ret = **userService**.signin(username, password);  
 **if** (Boolean.***TRUE***.equals(ret.get(**"success"**))) {  
 *//* ***TODO SOMETHING*** } **else** {  
 *//* ***TODO SOMETHING*** }  
 **return** ret;  
 }

对于 Service 处理如果出现错误，应该通过排除异常的形式与Controller交互

**public** String signin(String username, String password) {  
 **try** {  
 userService.sign(username, password)  
 } **catch**(LockedAccountException ex) {  
 *//* ***TODO*** } **catch** (DisabledAccountException ex) {  
 *//* ***TODO*** } **catch** (AccountException ex) {  
 *//* ***TODO*** }  
 }

#### 10.17方法删除与过时 @Deprecated

对于项目中的类或方法定义后一旦提交存在引用后，不允许随意删除， 如果不希望再使用必须标注 @Deprecated，并说明方法过时原因

另外，项目中不允许使用 @Deprecated 标注的方法或类!

#### 10.17不要随意空行和缩进

空行、随意锁紧不美观，而且不易读

#### 10.17删除未使用的依赖和变量

当前系统中未使用的依赖请删除掉，定义后未使用的变量也应该删除

#### 10.17参数校验

1：在方法内对入参进行校验。（用户是不可信的，web环境中即使前端校验，后端仍需校验。）

2：可能被别人共享的方法，也请对入参进行校验，如果不满足条件请尽可能提前做处理，不要等到运行时抛异常

#### 10.17 不要临时定义的字符串（尽可能）

系统中多次被重用的字符串，应该抽出为常量，不要临时定义。

#### 10.17 不要出现System.out.println()

System.out.println() 仅仅出现在开发时，提交文件时请删除，如需要记录日志一律采用log系统。

#### 10.17 import不要使用通配符

即，不要出现类似这样的的import语句：import java.util.\*;

#### 10.17 重载：永不分离

当一个类有多个构造函数，或是多个同名方法，这些函数/方法应该按顺序出现在一起，中间不要放进其它函数/方法。

#### 10.18 未完成代码与存在问题代码

- 对于项目中未完成代码必须使用 TODO 来标识，标识格式为: TODO 作者 内容

*//* ***TODO yangchanghe: 排序算法优化***

- 对于项目中需要修复的内容必须使用FIXME 来标识， 标识格式为 FIXME 作者 内容

*//* ***FIXME yangchanghe ID 为空时，存在 NullPointerException***

#### 10.19 注释说明

请先阅读 基本规范《文件组织》

1. 每个类中必须存在 包含版权信息的 开头注释(package之上)

*/\*  
 \* Copyright (c) 2005, 2014 glanway.com  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \*/*

2. 每个类/及方法上必须包含该类的文档注释

- 介绍类的用途以及使用方法，以及更新历史

- 必须使用 @author 按照顺序依次标明作者，更新者

格式如下：

*/\*\*  
 \* 基础 Service 实现  
 \* 提供基本的 CRUD 操作  
 \* <p/>  
 \* update 2014.7.8 liujichao: 更新错误消息提醒  
 \* update 2014.8.1 yuruizhi: Fix 没有设置BaseDao时产生 NullPointerException  
 \*  
 \** ***@author*** *yangchanghe  
 \** ***@author*** *liujichao  
 \** ***@author*** *yuruizhi  
 \*/*

3. 每个类的属性上根据具体情况选择性使用文档注释说明字段含义

- 对于有歧义或第一次看到不易理解的必须加上注释

- 对于实体中的业务相关属性必须加上注释

4. 接口方法及常量必须添加文档注释, 方法的文档注释必须明确说明其功能以及参数，

返回值，异常

5. 对于子类,实现类中的方法 可以使用以下注解来继承文档注释

*/\*\*  
 \* {****@inheritDoc****}  
 \** ***@author*** *yangchanghe  
 \*/*

对于非接口的属性和方法,如果名称和其含义相同且无相关业务的字段不需要添加注释

#### 10.20 数据库表/字段命名规则

对于数据库表和字段名必须保证如下规则

1. 表名/字段名全部大写 下划线写法 eg: TB\_PRODUCT\_IMAGE

2. 表名和实体名关系为: 表名统一前缀 + 实体名的全大写下划线写法 --> 表名

eg: 实体ProductImage 对应表为 TB\_PRODUCT\_IMAGE

3. 数据库字段名与实体属性名保持一致， 实体属性名转换为 全大写下划线写法即为字段名

eg: 属性 storagePath 对应字段为 STORAGE\_PATH

#### 10.21 SQL 书写命名规则

- SQL 中所有的关键字以及表名，字段名均使用大写

- SQL 应该合理的缩进以及换行， 一行长度不允许超过 100 个字符

eg:

SELECT ID, TITLE FROM TB\_PRODUCT;

SELECT

ID,

TITLE

FROM

TB\_PRODUCT P

LEFT JOIN TB\_PRODUCT\_IMG IMG ON P.ID = IMG.PRODUCT\_ID;

MyBatis 说明：

- MyBatis Mapper 中SQL要遵循以上规则

- MyBatis Mapper 中 插入语句不允许与查询语句公用一个SQL片段

以下严重不允许，经常会因为修改 baseCol 造成大量错误产生

<sql id="baseCol">  
 ID,  
 NAME  
 </sql>  
 <insert id="save" >  
 INSERT INTO TB\_PRODUCT(<include refid="baseCol" />)  
 VALUES (#{id}, #{name})  
 </insert>  
 <select id="findAll">  
 SELECT <include refid="baseCol" /> FROM TB\_PRODUCT  
 </select>

- 理解业务，明白功能的意图，寻找简洁的方法达到目的！

- 查询时不要select \* ，用什么字段就查什么字段，即便为了复用，也要尽可能减少全字段检索的使用。

- 避免全表扫描，select时尽可能利用索引。

Where后避免使用 is null (字段尽可能not null ,num字段加默认值)

少用or，可用union all代替

少用in 和 not in

Where 后少用表达式

Where后减少对字段的函数操作

复合索引，第一个优先使用，使用多个时注意保持顺序

#### 10.22 开发环境说明

Eclipse IDE for Java EE Developers Luna (4.4+)

Apache Tomcat 7.35 + (测试，发布使用)

jetty-maven-plugin-7.6.15.v20140411 (maven 插件，仅用于开发)

配置如下

<**groupId**>org.mortbay.jetty</**groupId**>  
 <**artifactId**>jetty-maven-plugin</**artifactId**>  
 *<!-- v7 的嵌入式做的最好 -->* <**version**>7.6.15.v20140411</**version**>