

**计算机科学与工程学院题库管理系统**

Java软件编程规范说明书

团队：upower团队

**目 录**

[1 引言 1](#_Toc32581)

[2 编码规范 1](#_Toc17549)

[2.1 Java 编码规范 1](#_Toc17849)

[2.1.1 命名规范 1](#_Toc2970)

[2.1.2 变量定义规范 3](#_Toc26750)

[2.1.3 代码编码风格 4](#_Toc26156)

[2.1.4编码约定 5](#_Toc4161)

[2.1.5函数、过程 8](#_Toc18447)

[2.1.6注释规范 9](#_Toc24623)

[2.1.7排版规范 10](#_Toc29729)

[2.1.8 Java 文件格式 10](#_Toc4741)

[2.1.9 可读性 11](#_Toc6941)

[2.1.10 性能 11](#_Toc16061)

[2.1.11 代码测试、维护 12](#_Toc8596)

[2.1.12 质量保证 12](#_Toc15901)

[2.1.13 代码编译 13](#_Toc20167)

[2.2 JSP 编码规范 13](#_Toc2475)

[2.3 Struts 编码规范 14](#_Toc29509)

[2.3.1 Action，From，Bean命名规则 14](#_Toc1238)

[2.3.2 struts-config.xml 的定义规则 15](#_Toc19828)

# **1 引言**

代码规范提高软件代码的可读性，使得开发人员快速和彻底的理解新代码。好的代码风格不仅会提高可读性，而且会使代码更健壮，更为重要的是在修改时不容易出错。在现代软件开发中，维护工作会占用 80% 的时间，而且开发者和维护者通常不是同一个程序员。这意味着你经常要阅读和修改别人开发的程序，别人也同样可能需要阅读和修改你开发的程序。既然如此，为什么不把这利人利己的事情作好呢？一些习惯自由程序人员可能对这些规则很不适应，但是在多个开发人员共同写作的情况下，这些规则是必需的。

# **2 编码规范**

## 2.1 Java 编码规范

## **2.1.1 命名规范**

定义这个规范的目的是让项目中所有的文档都看起来像一个人写的，增加可读性，减少项目组中因为换人而带来的损失。（这些规范并不是一定要绝对遵守，但是一定要让程序有良好的可读性）

（1）代码中的命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束。

反例： \_name/\_\_name/$Object/name\_/name$/Object$

（2） 代码中的命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。

（3）类名使用UpperCamelCase风格，必须遵从驼峰形式，但以下情形例外：（领

域模型的相关命名）EO / BO / DTO / VO等。

正例：MarcoPolo / UserEO / XmlService / TcpUdpDeal / TaPromotion

反例：macroPolo / UseEo / XMLService / TCPUDPDeal / TAPromotion

（4）方法名、参数名、成员变量、局部变量都统一使用lowerCamelCase风格，必须遵从驼峰形式。

正例： localValue / getHttpMessage() / inputUserId

（5）常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。

正例： MAX\_STOCK\_COUNT

反例： MAX\_COUNT

（6）抽象类命名使用Abstract或Base开头；异常类命名使用Exception结尾；测试类命名以它要测试的类的名称开始，以Test结尾。

（7）中括号是数组类型的一部分，数组定义如下：String[] args;

反例：使用String args[]的方式来定义。

（8）POJO类中布尔类型的变量，都不要加is，否则部分框架解析会引起序列化错误。

反例：定义为基本数据类型Boolean isSuccess；的属性，它的方法也是isSuccess()，RPC框架在反向解析的时候，“以为”对应的属性名称是success，导致属性获取不到，进而抛出异常。

（9）包名统一使用小写，点分隔符之间有且仅有一个自然语义的英语单词。包名统一使用单数形式，但是类名如果有复数含义，类名可以使用复数形式。

正例： 应用工具类包名为com.adc.open.util、类名为MessageUtils（此规则参考spring的框架结构）

（10）杜绝完全不规范的缩写，避免望文不知义。

反例： AbstractClass“缩写”命名成AbsClass；condition“缩写”命名成 condi，此类随意缩写严重降低了代码的可阅读性。

（11）接口和实现类的命名有两套规则：

①对于Service和DAO类，基于SOA的理念，暴露出来的服务一定是接口，内部的实现类用Impl的后缀与接口区别。

正例：CacheServiceImpl实现CacheService接口。

（12）【参考】各层命名规约：

A）Service/DAO层方法命名规约

1） 获取单个对象的方法用get做前缀。

2） 获取多个对象的方法用list做前缀。

3） 获取统计值的方法用count做前缀。

4） 插入的方法用save（推荐）或insert做前缀。

5） 删除的方法用remove（推荐）或delete做前缀。

6） 修改的方法用update做前缀。

B)领域模型命名规约

1） 数据对象：xxxEO，xxx即为数据表名。

2） 数据传输对象：xxxDTO，xxx为业务领域相关的名称。

3） 展示对象：xxxVO，xxx一般为网页名称。

4） POJO是EO/DTO/BO/VO的统称，禁止命名成xxxPOJO。

## **2.1.2 变量定义规范**

1. 去掉没必要的公共变量。

2. 构造仅有一个模块或函数可以修改、创建，而其余有关模块或函数只访问的公共变量，防止多个不同模块或函数都可以修改、创建同一公共变量的现象。

3. 仔细定义并明确公共变量的含义、作用、取值范围及公共变量间的关系。

4. 明确公共变量与操作此公共变量的函数或过程的关系，如访问、修改及创建等。

5. 当向公共变量传递数据时，要十分小心，防止赋与不合理的值或越界等现象发生。

6. 防止局部变量与公共变量同名。

7. 仔细设计结构中元素的布局与排列顺序，使结构容易理解、节省占用空间，并减少引起误用现象。

8.结构的设计要尽量考虑向前兼容和以后的版本升级，并为某些未来可能的应用保留余地（如预留一些空间等）。

9.留心具体语言及编译器处理不同数据类型的原则及有关细节。

10.严禁使用未经初始化的变量。声明变量的同时对变量进行初始化。

11.编程时，要注意数据类型的强制转换。

## **2.1.3 代码编码风格**

使用 Eclipse 的程序员建议使用 format 功能优化源代码的格式。

1. 大括号的使用约定。如果是大括号内为空，则简洁地写成{}即可，不需要换

行；如果是非空代码块则：

（1） 左大括号前不换行。

（2） 左大括号后换行。

（3） 右大括号前换行。

（4） 右大括号后还有 else 等代码则不换行；表示终止的右大括号后必须换行。

2. 左小括号和右边相邻字符之间不出现空格；同样，右小括号和左边相邻字符

之间也不出现空格。详见第 5 条下方正例提示。

3. if/for/while/switch/do 等保留字与小括号之间都必须加空格。

4. 任何运算符左右必须加一个空格。 说明：运算符包括赋值运算符=、逻辑运

算符&&、加减乘除符号、三目运算符等。

5. 缩进采用 4 个空格，禁止使用 tab 字符。

说明：如果使用 tab 缩进，必须设置缩进，必须设置 1 个 tab 为 4 个空格。IDEA设置 tab 为 4 个空格时，请勿勾选 Use tab character；而在 eclipse 中，必须勾选insert spaces for tabs。

6. 单行字符数限不超过 120 个，超出需要换行时遵循如下原则：

1） 第二行相对一缩进 4 个空格，从第三行开始不再继续缩进参考示例。

2） 运算符与下文一起换行。

3） 方法调用的点符号与下文一起换行。

4） 在多个参数超长，逗号后进行换行。

5） 在括号前不要换行，见反例。

正例：

StringBuffer sb = new StringBuffer();

//超过 120 个字符的情况下，换行缩进 4 个空格，并且方法前的点符号一起换行

sb.append("zi").append("xin")...

append("huang")...

.append("huang")...

.append("huang");

反例：

StringBuffer sb = new StringBuffer();

//超过 120 个字符的情况下，不要在括号前换行

sb.append("zi").append("xin")...append ("huang");

//参数很多的方法调用可能超过 120 个字符，不要在逗号前换行

method(args1, args2, args3, ... , argsX);

7. 方法参数在定义和传入时，多个参数逗号后边必须加空格。 正例：下例中实

参的"a",后边必须要有一个空格。

method("a", "b", "c");

8. IDE 的 text file encoding 设置为 UTF-8; IDE 中文件的换行符使用 Unix 格式，不要使用 windows 格式。

## **2.1.4编码约定**

**1. exit()**

exit 除了在 main 中可以被调用外，其他的地方不应该调用。因为这样做不给任何代码代码机会来截获退出。一个类似后台服务地程序不应该因为某一个库模块决定了要退出就退出。

**2.对类成员的访问**

一般成员变量应该定义为 private。子类也应该通过方法来访问。

**3.方法的复杂性**

应避免单一方法过于复杂。如果方法实现主要由简单的赋值语句和方法调用构成，长度不应超过 50 行。如果由分支和循环，建议不超过 30 行。

**4.常量定义**

（1）尽量不要出现任何魔法值（即未经定义的常量）直接出现在代码中。若需直接使用常量则必须加上注释说明含义。

反例： String key = "Id#adc\_"+tradeId； cache.put(key, value);

（2）long或者Long初始赋值时，必须使用大写的L，不能是小写的l，小写容易跟数字1混淆，造成误解。

说明：Long a = 2l; 写的是数字的 21，还是 Long 型的 2?

**5.OOP 规约**

（1）避免通过一个类的对象引用访问此类的静态变量或静态方法，无谓增加编译器解析成本，直接用类名来访问即可。

（2）所有的覆写方法，必须加@Override注解。

反例：getObject()与get0bject()的问题。一个是字母的O，一个是数字的0，加@Override可以准确判断是否覆盖成功。另外，如果在抽象类中对方法签名进行修改，其实现类会马上编译报错。

（3）相同参数类型，相同业务含义，才可以使用Java的可变参数，避免使用Object。

说明：可变参数必须放置在参数列表的最后。（提倡同学们尽量不用可变参数编程） 正例：public User getUsers(String type, Integer... ids) {...}

（4）外部正在调用或者二方库依赖的接口，不允许修改方法签名，避免对接口调 用方产生影响。接口过时必须加@Deprecated注解，并清晰地说明采用的新接口或者新服务是什么。

（5）不能使用过时的类或方法。

说明：java.net.URLDecoder 中的方法decode(String encodeStr) 这个方法已经过 时，应该使用双参数decode(String source, String encode)。接口提供方既然明确是过时接口，那么有义务同时提供新的接口；作为调用方来说，有义务去考证过时方法的新实现是什么。

（6）Object的equals方法容易抛空指针异常，应使用常量或确定有值的对象来调用equals。

正例： "test".equals(object);

反例： object.equals("test");

说明：推荐使用java.util.Objects#equals （JDK7引入的工具类）

1. 所有的相同类型的包装类对象之间值的比较，全部使用equals方法比较。

说明：对于Integer var = ?在-128至127之间的赋值，Integer对象是在IntegerCache.cache产生，会复用已有对象，这个区间内的Integer值可以直接使用==进行判断，但是这个区间之外的所有数据，都会在堆上产生，并不会复用已有对象，这是一个大坑，推荐使用equals方法进行判断。

（8）关于基本数据类型与包装数据类型的使用标准如下：

1） 所有的POJO类属性必须使用包装数据类型。

2） RPC方法的返回值和参数必须使用包装数据类型。

（9）定义EO/DTO/VO等POJO类时，不要设定任何属性默认值。

反例：POJO类的gmtCreate默认值为new Date();但是这个属性在数据提取时并没有置入具体值，在更新其它字段时又附带更新了此字段，导致创建时间被修改成当前时间。

（10）序列化类新增属性时，请不要修改serialVersionUID字段，避免反序列失败；如果完全不兼容升级，避免反序列化混乱，那么请修改serialVersionUID值。

说明：注意serialVersionUID不一致会抛出序列化运行时异常。

（11）构造方法里面禁止加入任何业务逻辑，如果有初始化逻辑，请放在init方法中。

（12）POJO类必须写toString方法。

使用IDE的中工具：source> generate toString时，如果继承了另一个POJO类，注意在前面加一下super.toString。

说明：在方法执行抛出异常时，可以直接调用POJO的toString()方法打印其属性值，便于排查问题。

**6.其它**

（1）在使用正则表达式时，利用好其预编译功能，可以有效加快正则匹配速度。

说明：不要在方法体内定义：Pattern pattern = Pattern.compile(规则);

（2）velocity调用POJO类的属性时，建议直接使用属性名取值即可，模板引擎会自动按规范调用POJO的getXxx()，如果是boolean基本数据类型变量（boolean命名不需要加is前缀），会自动调用isXxx()方法。 说明：注意如果是Boolean包装类对象，优先调用getXxx()的方法。

（3）后台输送给页面的变量必须加$!{var}——中间的感叹号。 说明：如果var=null

或者不存在，那么${var}会直接显示在页面上。

（4）注意 Math.random() 这个方法返回是double类型，注意取值的范围 0≤x<1（能够取到零值，注意除零异常），如果想获取整数类型的随机数，不要将x放大1-10的若干倍然后取整，直接使用Random对象的nextInt或者nextLong方法。

（5）获取当前毫秒数System.currentTimeMillis(); 而不是new Date().getTime(); 说明：如果想获取更加精确的纳秒级时间值，使用System.nanoTime()的方式。在JDK8中，针对统计时间等场景，推荐使用Instant类。

## **2.1.5函数、过程**

1. 函数的规模尽量限制在 200 行以内。

2.一个函数最好仅完成一件功能。

3. 为简单功能编写函数。

4.函数的功能应该是可以预测的，也就是只要输入数据相同就应产生同样的输出。

5. 尽量不要编写依赖于其他函数内部实现的函数。

6. 避免设计多参数函数，不使用的参数从接口中去掉。

7. 用注释详细说明每个参数的作用、取值范围及参数间的关系。

8.检查函数所有参数输入的有效性。

9. 检查函数所有非参数输入的有效性，如数据文件、公共变量等。

10. 函数名应准确描述函数的功能。

11. 避免使用无意义或含义不清的动词为函数命名。

12. 函数的返回值要清楚、明了，让使用者不容易忽视错误情况。

13. 明确函数功能，精确（而不是近似）地实现函数设计。

14. 减少函数本身或函数间的递归调用。

15. 编写可重入函数时，若使用全局变量，则应通过关中断、信号量（即 P、 V 操作）等手段对其加以保护。

## **2.1.6注释规范**

定义这个规范的目的是让项目中所有的文档都看起来像一个人写的，增加可读性，减少项目组中因为换人而带来的损失。 （这些规范并不是一定要绝对遵守，但是一定要让程序有良好的可读性）。

Java的语法与 C++ 及为相似，那么，你知道 Java 的注释有几种吗？是两种？

1. 注释一行
2. 注释若干行
3. 不完全对，除了以上两种之外，还有第三种，文档注释：

注释若干行，并写入 javadoc 文档，注释要简单明了。

String userName = null; // 用户

a.边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。

b.在必要的地方注释，注释量要适中。注释的内容要清楚、明了，含义准确，防止注释二义性。保持注释与其描述的代码相邻，即注释的就近原则。

c.对代码的注释应放在其上方相邻位置,不可放在下面。

d.对数据结构的注释应放在其上方相邻位置，不可放在下面； 对结构中的每个域的注释应放在此域的右方；同一结构中不同域的注释要对齐。

e.变量、常量的注释应放在其上方相邻位置或右方。全局变量要有较详细的注释，包括对其功能、取值范围、 哪些函数或过程存取它以及存取时注意事项等的说明。在每个源文件的头部要有必要的注释信息，包括：文件名；版本号；作者；生成日期；模块功能描述（如功能、主要算法、内部各部分之间的关系、该文件与其它文件关系等） ；主要函数或过程清单及本文件历史修改记录等。

f.在每个函数或过程的前面要有必要的注释信息，包括： 函数或过程名称； 功能描述； 输入、输出及返回值说明；调用关系及被调用关系说明等

/\*\*  
\* Description :checkout 提款  
\* @param Hashtable cart info  
\* @param OrderBean order info  
\* @return String  
\*/  
public String checkout(Hashtable htCart,  
OrderBean orderBean)  
throws Exception{  
}  
g. javadoc注释标签语法

@author 对类的说明 标明开发该类模块的作者

@version 对类的说明 标明该类模块的版本

@see 对类、属性、方法的说明 参考转向，也就是相关主题

@param 对方法的说明 对方法中某参数的说明

@return 对方法的说明 对方法返回值的说明

@exception 对方法的说明 对方法可能抛出的异常进行说明

h.注释率要求

代码注释率 = 注释行数/(有效代码行数+注释行数)

有效代码行数是指不包含注释行和空白行的行数

## **2.1.7排版规范**

1.关键词和操作符之间加适当的空格。

2.相对独立的程序块与块之间加空行

3.较长的语句、表达式等要分成多行书写。

4.划分出的新行要进行适应的缩进，使排版整齐，语句可读。

5.长表达式要在低优先级操作符处划分新行，操作符放在新行之首。

6.循环、判断等语句中若有较长的表达式或语句，则要进行适应的划分。

7.若函数或过程中的参数较长，则要进行适当的划分。

8.不允许把多个短语句写在一行中，即一行只写一条语句。

9.函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格。独占一行并且位于同一列，同时与引用它们的语句左对齐。在函数体的开始、类的定义、结构的定义、枚举的定义以及 if 、 for、 do、 while、 switch、 case语句中的程序都要采用如上的缩进方式。

## **2.1.8 Java 文件格式**

所有的 Java(\*.java)文件都必须遵守如下的样式规则：

1. 类、类属性、类方法的注释必须使用Javadoc规范，使用/\*\*内容\*/格式，不得

使用//xxx方式。

说明：在IDE编辑窗口中，Javadoc方式会提示相关注释，生成Javadoc可以正确输出相应注释；在IDE中，工程调用方法时，不进入方法即可悬浮提示方法、参数、返回值的意义，提高阅读效率。

2. 所有的抽象方法（包括接口中的方法）必须要用Javadoc注释、除了返回值、

参数、异常说明外，还必须指出该方法做什么事情，实现什么功能。

说明：对子类的实现要求，或者调用注意事项，请一并说明。

3. 所有的类都必须添加创建者和创建日期。

配合 SVN，Eclipse 的使用，所有 Java 文件的头部固定为：

/\*

\*Lastmodifiedby$Author$on$Date$(UTC)

\*Revision:$Revision$

\*/

Author，Date 处写明作者和日期。

4. 方法内部单行注释，在被注释语句上方另起一行，使用//注释。方法内部多行

注释使用/\* \*/注释，注意与代码对齐。

5. 所有的枚举类型字段必须要有注释，说明每个数据项的用途。

6. Imports

避免import整个package而造成的浪费，而必须import具体的类。如：

import java.util.\*;//犯规！

import java.util.Observable;//可以用。

7. ClassFields

所有的静态变量和成员变量出现在类定义之后。一般按照常量、静态变量、

成员变量的顺序定义。

5main方法：如果有main(String[])方法,那么它应该写在类的底部。

8. 与其“半吊子”英文来注释，不如用中文注释把问题说清楚。专有名词与关键

字保持英文原文即可。

反例：“TCP 连接超时”解释成“传输控制协议连接超时”，理解反而费脑筋。

## **2.1.9 可读性**

1.避免使用不易理解的数字，用有意义的标识来替代。

2.不要使用难懂的技巧性很高的语句。

3.源程序中关系较为紧密的代码应尽可能相邻。

## **2.1.10 性能**

在写代码的时候， 从头至尾都应该考虑性能问题。 这不是说时间都应该浪费在优化代码上，而是我们时刻应该提醒自己要注意代码的效率。比如：如果没有时间来实现一个高效的算法，那么我们应该在文档中记录下来，以便在以后有空的时候再来实现它。不是所有的人都同意在写代码的时候应该优化性能这个观点的， 他们认为性能优化的问题应该在项目的后期再去考虑，也就是在程序的轮廓已经实现了以后。

1.不必要的对象构造

不要在循环中构造和释放对象

2.使用 **StringBuffer** 对象

在处理 String 的时候要尽量使用 StringBuffer 类， StringBuffer 类是构成 String 类的基础。 String 类将 StringBuffer 类封装了起来，（以花费更多时间为代价）为开发人员提供了一个安全的接口。当我们在构造字符串的时候，我们应该用 StringBuffer 来实现大部分的工作，当工作完成后将 StringBuffer 对象再转换为需要的 String 对象。比如：如果有一个字符串必须不断地在其后添加许多字符来完成构造，那么我们应该使用 StringBuffer 对象和它的append() 方法。如果我们用 String 对象代替 StringBuffer 对象的话，会花费许多不必要的创  
建和释放对象的 CPU 时间。

3.避免太多的使用 **synchronized** 关键字

避免不必要的使用关键字 synchronized，应该在必要的时候再使用她，这是一个避免死锁的好方法。

## **2.1.11 代码测试、维护**

1.单元测试要求至少达到语句覆盖。

2.单元测试开始要跟踪每一条语句，并观察数据流及变量的变化。

3.清理、整理或优化后的代码要经过审查及测试。

4.代码版本升级要经过严格测试。

## **2.1.12 质量保证**

1.在软件设计过程中构筑软件质量。

代码质量保证优先原则  
（ 1）正确性，指程序要实现设计要求的功能。

（ 2）稳定性、安全性，指程序稳定、可靠、安全。

（ 3）可测试性，指程序要具有良好的可测试性。

（ 4）规范 /可读性，指程序书写风格、命名规则等要符合规范。

（ 5）全局效率，指软件系统的整体效率。

（ 6）局部效率，指某个模块 /子模块 /函数的本身效率。

（ 7）个人表达方式 /个人方便性，指个人编程习惯。

2.只引用属于自己的存贮空间。

3.防止引用已经释放的内存空间。

4.过程 /函数中分配的内存，在过程 /函数退出之前要释放。

5.过程 /函数中申请的（为打开文件而使用的）文件句柄，在过程 /函数退出前要关闭。

6.防止内存操作越界。

7.时刻注意表达式是否会上溢、下溢。

8.认真处理程序所能遇到的各种出错情况。

9.系统运行之初，要初始化有关变量及运行环境，防止未经初始化的变量被引用。

10.系统运行之初，要对加载到系统中的数据进行一致性检查。

11.严禁随意更改其它模块或系统的有关设置和配置。

12.不能随意改变与其它模块的接口。

13.充分了解系统的接口之后，再使用系统提供的功能。

14.要时刻注意易混淆的操作符。当编完程序后，应从头至尾检查一遍这些操作符。

15.不使用与硬件或操作系统关系很大的语句，而使用建议的标准语句。

16.建议：使用第三方提供的软件开发工具包或控件时，要注意以下几点：

（ 1）充分了解应用接口、使用环境及使用时注意事项。

（ 2）不能过分相信其正确性。

（ 3）除非必要，不要使用不熟悉的第三方工具包与控件。

## **2.1.13 代码编译**

1.编写代码时要注意随时保存，并定期备份， 防止由于断电、硬盘损坏等原因造成代码丢失。

2.同一项目组内，最好使用相同的编辑器，并使用相同的设置选项。

3.合理地设计软件系统目录，方便开发人员使用。

4.打开编译器的所有告警开关对程序进行编译。

5.在同一项目组或产品组中，要统一编译开关选项。

## 2.2 JSP 编码规范

1.整个 jsp/jsp bean表示层应当尽可能的瘦和简单化。

2.牢记大多数的 JSP都应当是只读的视图，而由页面 bean来提供模型。

3.应当一起设计 JSP和 JSP bean

4.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。

5.尽量把条件逻辑放在控制器中而不是放在视图中。

6.为 JSP、包含的文件、 JSP Bean和实现扩展标记的类使用遵循标准的命名惯例如：jsp 控制器 xxxxController.jsp

被包含的 : jsp \_descriptiveNameOfFragment.jsp

jsp 页面模型 bean: <pagename>Bean如 loginBena.java

jsp 会话 bena: xxxxSessionBean

标记类 : xxxxTag,xxxxTagExtraInfo

7.应当在 JSP中避免使用页面引入指令。 import 指令会促使类的实例化而不是 jsp bean的实例化：

不用： <%@ page import = "com.java.util.\*" %>

而用： <% java。util。List l = new java.util.LinkedList(); %>

8. jsp 不应该直接去访问请求参数。 bean应当执行这样的处理过程并且输出所处理的模型数据。

9.jsp 不应当访问属性文件，或者使用 JNDI。 bean可以访问属性。如果 jsp bean具有的所有的属性不能够从页面请求中被映射到，就要尽力在 <jsp:useBean>

标记中设置属性。

10.应当避免设计既显示表单又处理结果的页面。

11.在 jsp 中避免代码重复。把要重复的功能放在一个包含的 jsp、 bean或标记扩展中，使得它能够被重用。

12. jsp bean应当永远不要去产生 HTML。

## 2.3 Struts 编码规范

## **2.3.1 Action，From，Bean命名规则**

Action: 命名以Action结尾

例：

添加文本的 Actoin,命名为 AddTextsAction

修改文本的 Action,命名为 UpTextsAction

删除文本的 Action,命名为 DelTextsAction

查看文本的 Action,命名为 ViewTextsAction

From：命名以From结尾

例：

操作文本的 Form，命名为 TextsForm

Bean：命名以Bean结尾

例：

操作文本的 Bean，命名为 TextsBean

## **2.3.2 struts-config.xml 的定义规则**

From属性定义规则：

<form-bean name="testimgForm" type="cn.TestimgForm" />

Action属性定义规则：

<action  
attribute="testimgForm"  
name="viewForm"  
path="/admin/example/viewTableAction"  
scope="request"  
type="cn.struts.action.example.ViewTableAction" />