2015. java学习交流群

8918 1289

每天有免费的Java学习课堂

——学习Java就是这么简单

——为Java而燃烧——

Java 语言编码规范(Java Code Conventions)

译者　晨光（Morning）
搜集整理 ：华竹技术实验室 http://sinoprise.com
简介 ：
本文档讲述了 Java 语言的编码规范，较之陈世忠先生《c++编码规范》的浩繁详尽，此文当属短小精悍了。而其中所列之各项条款，从编码风格，到注意事项，不单只 Java，对于其他语言，也都很有借鉴意义。因为简短，所以易记，大家不妨将此作为 handbook，常备案头，逐一对验。
声明：
如需复制、传播，请附上本声明，谢谢。
原文出处：http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html，
译文出处：http://morningspace.51.net/，moyingzz@etang.com

目录

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

为了执行规范，每个软件开发人员必须一致遵守编码规范。每个人。

1.2 版权声明(Acknowledgments)

本文档反映的是 Sun MicroSystems 公司，Java 语言规范中的编码标准部分。主要贡献者包括：Peter King，Patrick Naughton，Mike DeMoney，Jonni Kanerva，Kathy Walrath 以及 Scott Hommel。

本文档现由 Scott Hommel 维护，有关评论意见请发至 shommel@eng.sun.com

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"）
- 包和引入语句（参见"包和引入语句"）
- 类和接口声明（参见"类和接口声明"）

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"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"
2	类或接口的声明	
3	类/接口实现的注释	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

	(/*……*/)如果有必要的话	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已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"

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")。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"

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"）或单行注释（参见"5.1.2"）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	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese

	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)。	
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"以及"文挡注释"。

```
/*
 * @(#)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...
    }
}
```

序

经过了两个星期不懈努力，今天终于完成了对 strtus 整体架构及核心标签库的介绍。从几乎不懂 struts 和 HTML 标签，到可以给别人解决涉及 struts 的一些小问题，这与朋友的帮助和我的努力是分不开的，但我更希望它能给那些想要学的，正在学的和已经学过的人带来不同的益处。我知道我是个新手，但我会用百倍的努力继续在这个领域进行深入性的和扩展性的学习与研究。同时，就象我说的，我们会用实际行动证明我们为你提供的帮助。

这章适合做参考资料，他不仅包含了对 struts 的整体架构及主要组件的详细和清晰的介绍，而且对于很多人都关注的 struts 标签库也进行了细致的介绍，可以说它是你手头一个可以用来进行参考的资料。当你哪里不清楚，你完全可以到这里查询。为了你查询方便，我做了一个目录。

由于开源软件不断发展，所以这不可能是最后版本，我会不断进行内容的修改和新特性的添加。比如 strus1.1 的 Nested Tag，但由于个人技术水平和时间等多方面原因，速度不会太快，这就需要大家的帮助，希望大家共同参加一些资料的整理。

最后，我真心的感谢 jag，banq，steelg,holen,七老爷，还有混血儿，要不是他，我会更早完成这个资料的，不过他又一次锻炼了我的意志。

我已经尽力避免在文字或代码中出现错误，但是人无完，疏漏总是难免的。如果你在阅读发现了其中的错误，比如文字的错误或错误的代码，我非常希望你将这些信息反馈给我，这样会帮助其他人解决遇到的问题，也会不断完善这套资料。我的邮件是：tyrones@cmmail.com QQ:36983608

希望本资料能给你带来帮助！

2002 年 08 月 26 日 午夜

胡峤整理于 2002-9-12
mailto:qiaohu@263.net

1. Struts 框架

struts 框架具有组件的模块化，灵活性和重用性的优点，同时简化了基于 MVC 的 web 应用程序的开发。

本章详细讨论 struts 架构。我们将看到 struts 是如何清晰地区分控制，事务逻辑和外观，从而简化了开发应用程序过程的。我们还将介绍 struts 提供的类如何使得开发工作更加简单，这些类包括：

1. 控制程序流程的类
2. 实现和执行程序事务逻辑的类
3. 自定义的标记库使得创建和验证 HTML 表单更加容易

1.1 Struts 压缩包内容

文件夹 jakarta-struts-1.0.2 包含两个目录，lib 和 webapps。在 lib 目录中有使用 struts 创建应用程序是所需的文件：

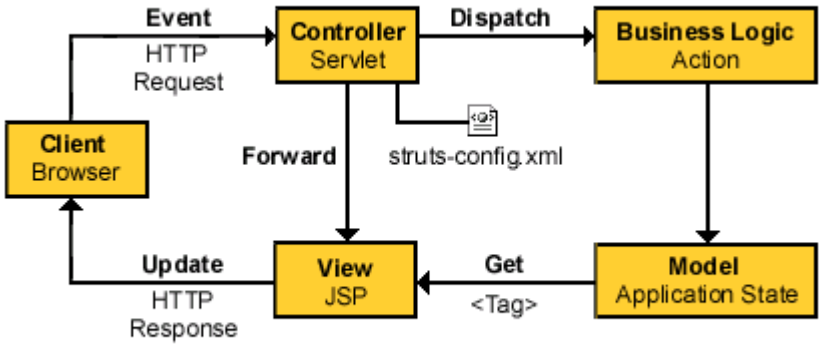
文件	描述
jdbc2_0-stdext.jar	包含 JDBC2.0 Optional Package API 类。如果我们要使用 struts 提供的数据资源，就需要将这个文件拷贝到 WEB-INF\lib 下
Struts.jar	包含 struts 中所有的 java 类。同样也需要拷贝到 WEB-INF\lib 下
*.tld	标记库描述器文件，描述了多个 struts 标记库中的自定义标记。同样要拷贝到 WEB-INF\lib 下

在 webapps 目录下有如下文件：

Web 应用程序	描述
Struts-blank.war	一个简单的 web 应用程序
Struts-documentation.war	包含 struts 站点上所有 struts 文档
Struts-example.war	Struts 很多特性的示范
Struts-exercisetaglib.war	主要用于对自定义标签库进行增加而使用的测试页，但也可以示范如何使用 struts 标记
Struts-template.war	包含 struts 模板标记的介绍和范例
Struts-upload.war	一个简单的例子，示范如何使用 struts 框架上传文件

1.2 Struts 体系结构

让我们从 MVC 角度观察 struts 框架中的组件



Struts 概览

框架中三个部分：模型，视窗和控制器。

1.2.1 模型

在 struts 框架中，模型分为两个部分：

- 系统的内部状态
- 可以改变状态的操作（事务逻辑）

内部状态通常由一组 ActinForm JavaBean 表示。根据设计或应用程序复杂度的不同，这些 Bean 可以是自包含的并具有持续的状态，或只在需要时才获得数据（从某个数据库）。

大型应用程序通常在方法内部封装事务逻辑（操作），这些方法可以被拥有状态信息的 bean 调用。比如购物车 bean，它拥有用户购买商品的信息，可能还有 checkOut()方法用来检查用户的信用卡，并向仓库发定货信息。

小型程序中，操作可能会被内嵌在 Action 类，它是 struts 框架中控制器角色的一部分。当逻辑简单时这个方法很适合。

建议用户将事务逻辑（要做什么）与 Action 类所扮演的角色（决定做什么）分开。

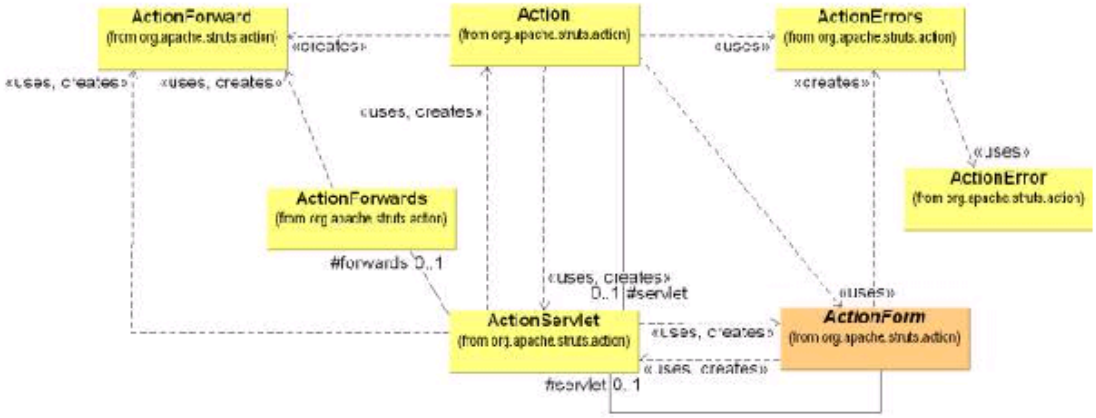
1.2.2 视窗

由 JSP 建立，struts 包含扩展自定义标签库，可以简化创建完全国际化用户界面的过程。

1.2.3 控制器

struts 中，基本的控制器组件是 ActionServlet 类中的实例 servelt，实际使用的 servlet 在配置文件中由一组映射（由 ActionMapping 类进行描述）进行定义。

1.3 Struts 框架中的组件



Servlet 与 Model 关系 UML 图

(由于 ROSE 工具还未能下载，只能找来这幅图，它说明了一定问题，特别是 ActionErrors，但它并没有将 ActionMapping，JSP 和 Tag Library 包含进来，有时间作完替换)

框架中所使用的组件：

ActionServlet	**控制器**
ActionClass	**包含事务逻辑**
ActionForm	**显示模块数据**
ActionMapping	**帮助控制器将请求映射到操作**
ActionForward	**用来指示操作转移的对象**
ActionError	**用来存储和回收错误**
Struts 标记库	**可以减轻开发显示层次的工作**

下面我们看看各自在框架中所扮演的角色和责任。

1.3.1 Struts 配置文件

这是将 struts 组件结合在一起的东东：struts-config.xml。默认值 \WEB-INF\struts-config.xml。配置文件可以定义：

- 全局转发
- ActionMapping 类
- ActionForm bean
- JDBC 数据源

配置全局转发

全局转发用来在 JSP 页之间创建逻辑名称映射。转发都可以通过对调用操作映射的实例来获得，例如：

```
actionMappingInstace.findForward(“ logicalName ”);
```

全局转发的例子：(所有的例子我没有进行解释，一是结合表可以理解，二是例子大部分来自系列四的示例，你应该在作完实验后，再来看一便)

```
<global-forwards>
```

```
    <forward   name="bookCreated"  path="/BookView.jsp"/>
  </global-forwards>
```

属性	描述
Name	全局转发的名字
Path	与目标 URL 的相对路径

配置 ActionMapping

ActionMapping 对象帮助进行框架内部的流程控制，它们可将请求 URI 映射到 Action 类,并且将 Action 类与 ActionForm bean 相关联。ActionServlet 在内部使用这些映射，并将控制转移到特定 Action 类的实例。所有 Action 类使用 perform()方法实现特定应用程序代码，返回一个 ActionForward 对象，其中包括响应转发的目标资源名称。例如：

```
<action-mappings>
    <action  path="/createBook"
             type="BookAction"
             name="bookForm"
             scope="request"
             input="/CreateBook.jsp">
    </action>
    <forward name=”failure” path=”/CreateBook.jsp”/>
    <forward name=”cancel” path=”/index.jsp”/>
</action-mappings>
```

属性	描述
Path	Action类的相对路径
Name	与本操作关联的Action bean的名称
Type	连接到本映射的Action类的全称（可有包名）
Scope	ActionForm bean的作用域（请求或会话）
Prefix	用来匹配请求参数与bean属性的前缀
Suffix	用来匹配请求参数与bean属性的后缀
attribute	作用域名称。
className	ActionMapping对象的类的完全限定名默认的类是org.apache.struts.action.ActionMapping
input	输入表单的路径，指向bean发生输入错误必须返回的控制
unknown	设为true，操作将被作为所有没有定义的ActionMapping的URI的默认操作
validate	设置为true，则在调用Action对象上的perform()方法前，ActionServlet将调用ActionForm bean的validate()方法来进行输入检查

通过<forward>元素，可以定义资源的逻辑名称，该资源是Action类的响应要转发的目标。

属性	描述
Id	ID
ClassName	ActionForward 类的完全限定名，默认是org.apache.struts.action.ActionForward
Name	操作类访问 ActionForward 时所用的逻辑名
Path	响应转发的目标资源的路径

redirect	若设置为 true，则 ActionServlet 使用 sendRedirect()方法来转发资源

配置 ActionForm Bean

ActionServlet 使用 ActionForm 来保存请求的参数，这些 bean 的属性名称与 HTTP 请求参数中的名称相对应，控制器将请求参数传递到 ActionForm bean 的实例，然后将这个实例传送到 Action 类。例子：

```
<form-beans>
<form-bean name="bookForm" type="BookForm"/>
</form-beans>
```

属性	描述
Id	ID
className	ActionForm bean的完全限定名，默认值是 org.apache.struts.action.ActionFormBean
Name	表单bean在相关作用域的名称，这个属性用来将bean与ActionMapping进行关联
Type	类的完全限定名

配置JDBC数据源

用<data-sources>元素可以定义多个数据源。

属性	描述
Id	ID
Key	Action 类使用这个名称来寻找连接
Type	实现 JDBC 接口的类的名称

下面属性需要<set-property>元素定义，在框架 1.1 版本中已不在使用，但你可用<data-source>元素。例子：

```
<data-sources>
    <data-source id=”DS1”
        key=”conPool”
        type=”org.apache.struts.util.GenericDataSource”
    <set-property id=”SP1”
        autoCommit="true"
        description="Example Data Source Configuration"
        driverClass="org.test.mm.mysql.Driver"
        maxCount="4"
        minCount="2"
        url="jdbc:mysql://localhost/test"
        user="struts"
        password="wrox" />
    <data-source/>
</data-sources>
```

属性	描述
desciption	数据源的描述
autoCommit	数据源创建的连接所使用的默认自动更新数

	据库模式
driverClass	数据源所使用的类，用来显示 JDBC 驱动程序接口
loginTimeout	数据库登陆时间的限制，以秒为单位
maxCount	最多能建立的连接数目
minCount	要创建的最少连接数目
password	数据库访问的密码
readOnly	创建只读的连接
User	访问数据库的用户名
url	JDBC 的 URL

通过指定关键字名称，Action 类可以访问数据源，比如：

```
javax.sql.DataSource ds = servlet.findDataSource(“conPool”);
javax.sql.Connection con = ds.getConnection();
```

1.4 ActionServlet 类

框架中的控制器组件是有 org.apache.struts.action.ActionServlet 类实现的,这个类是 javax.servlet.http.HttpServlet 类的扩展。

Struts controller 基本功能是：

1．截获用户的 Http 请求

2．把这个请求映射到相应的 Action 类，如果这是此类收到的第一个请求，将初始化实例并

缓寸。

3．创建或发现一个 ActionForm bean 实例（看配置文件是否定义），然后将请求过程移植到

bean.

4．调用 Action 实例的 perform()方法并将 ActioForm bean,Action Mapping 对象,request 和 response 对象传给它。

如：public ActionForword perform(ActionMapping mapping,
ActionForm form,
HttpServletRequest request,
HttpServletResponse response)

5.perform 返回一个 ActionForword 对象，此对象连接到相应的 jsp 页面.

1.4.1 ActionServlet 配置

我们需要在 web.xml 中声明 ActionServlet，并且将它配置成启动时进行加载。以下可以配置的初始化参数：

参数	默认值	描述
application	null	应用程序的资源集合的类
bufferSize	4096	文件上传的缓冲区大小

config	/WEB-INF/struts-config.xml	配置文件的位置和名称
content	Text/html	默认的内容类型
debug	0	程序调试的级别
detail	0	程序调试细节的级别
factory	null	消息资源工厂，用于国际化中解释消息资源
formBean	org.apache.struts.action.ActionFormBean	封装 ActionForm bean 信息的类的名称
forward	org.apache.struts.action.ActionForward	封装 ActionForward 对象信息的类的名称
locale	true	为 true,将在用户会话中存储一个本地对象
mapping	org.apache.struts.action.ActionForward	封装 ActionMapping 信息的类的名称
maxFileSize	250M	上传文件的最大尺寸
multipartClass	org.apache.struts.action.ActionForward	处理多部分请求的类的名称
noCache	False	HTTP 标头是否要设置为禁止缓寸
Null	True	设置为 true，对于无效的信息关键字将返回 null
tempDir	作为一个 servlet 参数提供给程序的工作目录	处理下载文件是使用的临时工作目录
validate	True	是否使用新格式的配置文件
vallidating	True	是否对配置文件进行有效性分析

大多数情况下，标准的 servlet 就能够满足用户需要。

第一次收到特定请求的 URI 时，ActionServlet 将适当的 Action 类进行实例化，然后 ActionServlet 在 Action 类实例中以 servlet 为变量名存储一个引用。当被实例化后，Action 类会被暂存以备再用。

ActionServlet 也提供一些方法,由 Action 类用来访问数据源和转发目标之类的资源。

1.4.2 ActionServlet 方法

ActinServlet 提供了一组能够被 Action 对象使用的方法。

Struts API 的全部信息在 struts-documentation.war 中可以找到。动态的添加或删除，这些方法只影响应用程序当前的实例：

```
public void addFormBean(ActionFormBean formBean)
public void removeFormBean(ActionFormBean formBean)
public void addForward(ActionForward actionForward)
public void removeForward(ActionForward actionForward)
public void addMapping(ActionMapping actionMapping)
```

```
public void removeMapping(ActionMapping actionMapping)
```
根据名称查找对象：
```
public ActionFormBean findFormBean(String name)
public ActionForward findForward(String name)
public ActionMapping findMapping(String name)
```
用来处理数据源：
```
public void addDataSource(String key , DataSource ds)
public DataSource findDataSource(String key)
```
我们还可以：

- 使用 destroy()方法结束 ActionServlet
- 使用 reload()方法从 struts 配置文件将信息重新加载。

1.5 ActionMapping 类

将特定请求映射到特定 Action 的相关信息存储在 ActionMapping 中，ActionServelt 将 ActionMapping 传送到 Action 类的 perform()方法，Action 将使用 ActionMapping 的 findForward()方法，此方法返回一个指定名称的 ActionForward，这样 Action 就完成了本地转发。若没有找到具体的 ActionForward，就返回一个 null.
```
public ActionForward findForward(String name)
```
可在映射中动态添加 ActionForward:
```
public void addForward(ActionForward forward)
```
可返回与映射关联的表单 bean:
```
public String getName()
```
可返回映射的属性域（会话或请求）
```
public String getScope()
```

1.6 Action 类

Action 类真正实现应用程序的事务逻辑，它们负责处理请求。在收到请求后，ActionServlet 会：

- 为这个请求选择适当的 Action
- 如果需要，创建 Action 的一个实例
- 调用 Action 的 perform()方法

如果 ActionServlet 不能找到有效的映射，它会调用默认的 Action 类(在配置文件中定义)。如果找到了 ActionServlet 将适当的 ActionMapping 类转发给 Action,这个 Action 使用 ActionMapping 找到本地转发，然后获得并设置 ActionMapping 属性。根据 servlet 的环境和被覆盖的 perform()方法的签名，ActionServlet 也会传送 ServletRequest 对象或 HttpServletRequest 对象。

所有 Action 类都扩展 org.apache.struts.action.Action 类,并且覆盖类中定义的某一个 perform()方法。有两个 perform()方法：

处理非 HTTP（一般的）请求：

```
public ActionForward perform(ActionMapping action,
                             AcionForm form,
                             ServletRequest request,
                             ServletResponse response)
throws IOException,ServletException
```

处理 HTTP 请求：

```
public ActionForward perform(ActionMapping action,
                             AcionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
throws IOException,ServletException
```

Action 类必须以“线程安全”的方式进行编程，因为控制器会令多个同时发生的请求共享同一个实例，相应的，在设计 Action 类时就需要注意以下几点：

- 不能使用实例或静态变量存储特定请求的状态信息，它们会在同一个操作中共享跨越请求的全局资源
- 如果要访问的资源（如 JavaBean 和会话变量）在并行访问时需要进行保护，那么访问就要进行同步

1.6.1 Action 类的方法

除了 perform()方法外，还有以下方法：

可以获得或设置与请求相关联的区域：

```
public Locale getLocale(HttpServletRequest request)
public void setLocale(HttpServletRequest request,Locale locale)
```

为应用程序获得消息资源：

```
public MessageResources getResources()
```

检查用户是否点击表单上的“取消”键，如果是，将返回 true:

```
public Boolean isCancelled(HttpServletRequest request)
```

当应用程序发生错误时，Action 类能够使用下面方法存储错误信息：

```
public void saveErrors(HttpServletRequest request,ActionErrors errors)
```

ActionError 实例被用来存储错误信息，这个方法在错误关键字下的请求属性列表中存储 ActionError 对象。通过使用在 struts 标记库中定义的自定义标记，JSP 页能够显示这些错误信息，稍后我们就介绍。

1.7 ActionForm 类

框架假设用户在应用程序中为每个表单都创建了一个 ActionForm bean，对于每个在 struts-config.xml 文件中定义的 bean，框架在调用 Action 类的 perform()方法之前会进行以下操作：

- 在相关联的关键字下，它检查用于适当类的 bean 实例的用户会话，如果在会话中没有可用的 bean，它就会自动创建一个新的 bean 并添加到用户的会话中。
- 对于请求中每个与 bean 属性名称对应的参数，Action 调用相应的设置方法。

- 当 Action perform()被调用时，最新的 ActionForm bean 传送给它，参数值就可以立即使用了。

ActionForm 类扩展 org.apache.struts.action.ActionForm 类，程序开发人员创建的 bean 能够包含额外的属性，而且 ActionServlet 可能使用反射（允许从已加载的对象中回收信息）访问它。

ActionForm 类提供了另一种处理错误的手段，提供两个方法：

```
Public ActionErrors validate(ActionMappin mapping,
                             ServletRequest request)
Public ActionErrors validate(ActionMappin mapping,
                             HttpServletRequest request)
```

你应该在自己的 bean 里覆盖 validate()方法，并在配置文件里设置<action>元素的 validate 为 true。在 ActionServlet 调用 Action 类前，它会调用 validate()，如果返回的 ActionErrors 不是 null，则 ActinForm 会根据错误关键字将 ActionErrors 存储在请求属性列表中。

如果返回的不是 null，而且长度大于 0，则根据错误关键字将实例存储在请求的属性列表中，然后 ActionServlet 将响应转发到配置文件<action>元素的 input 属性所指向的目标。

如果需要执行特定的数据有效性检查，最好在 Action 类中进行这个操作，而不是在 ActionForm 类中进行。

方法 reset()可将 bean 的属性恢复到默认值：

```
public void reset(ActionMapping mapping,HttpServletRequest request)
public void reset(ActionMapping mapping,ServletRequest request)
```

典型的 ActionFrom bean 只有属性的设置与读取方法（getXXX）,而没有实现事务逻辑的方法。只有简单的输入检查逻辑，使用的目的是为了存储用户在相关表单中输入的最新数据，以便可以将同一网页进行再生，同时提供一组错误信息，这样就可以让用户修改不正确的输入数据。而真正对数据有效性进行检查的是 Action 类或适当的事务逻辑 bean。

1.8 ActionForward 类

目的是控制器将 Action 类的处理结果转发至目的地。

Action 类获得 ActionForward 实例的句柄,然后可用三种方法返回到 ActionServlet，所以我们可以这样使用 findForward():

- ActionServlet 根据名称获取一个全局转发
- ActionMappin 实例被传送到 perform()方法，并根据名称找到一个本地转发

另一种是调用下面的一个构造器来创建它们自己的一个实例：

```
public ActionForward()
public ActionForward(String path)
public ActionForward(String path,Boolean redirect)
```

1.9 错误处理

struts 提供了两个类来处理错误：ActionErrors 和 ActionError，它们都扩展 org.apache.struts.action。ActionErrors 保存着 ActionError 对象的集合，其中每一个代表了独立的错误信息。每个 ActionError 都包含了关键字，能够映射到资源文件中存储的错误信息，而这个资源文件是在 ActionServlet 初始化参数中指定的。

1.9.1 ActionError 类

ActionError 类定义了一组重载的构造器来创建错误信息，第一个构造器方法使用一个字符串作为参数，例如：

```
ActionError error = new ActionError(“error.Invalid”);
```

实例 error 映射到应用程序资源文件中的一个错误消息：

```
error.invalid=<b>Invalid Number</b>
```

如果在 JSP 页使用<html:error>，用户就会看见加粗的 Invalid Number。

另一种使用了 java.text.MessageFormat 类，可在消息中指定替换字符串，例如：

```
error.invalid=<b>Invalid Number{0}</b>
```

创建一个错误消息：

```
ActionError error = new ActionError(‘error.invalid’,new Double(-1));
```

JSP 页显示：Invalild Number －1

还有获得特定消息的错误关键字：

```
public String getKey()
```

还有获得替换字符串数组：

```
public String[] getValues()
```

1.9.2 ActionError 类

ActionError 类从不独立进行错误处理，它们总是被存储在 ActionErrors 对象中。ActionErrors 对象保存 ActionError 类的集合以及它们特定的属性值，我们可以使用自己定义的属性值，或是使用 ActionErrors.GLOBAL_ERROR.

下面是典型 Action 类的 perform()中错误处理情况：

```
MyForm form = (MyForm) form;
if (number == -1) {
    ActionErrors errors = new ActionErrors();
    ActionError error = new ActionError(“error.Invalid”,new Double(-1));
    errors.add(ActionErrors.GLOBAL_ERROR,error);
    saveErrors(req,errors);
    String input = mapping.getInput();
    Return new ActionForward(input);
}
```

ActionErrors 有如下有用方法：

方法	描述
clear()	清除所有错误信息
empty()	如果 ActionErrors 对象是空的，它返回 true
get()	返回错误信息。若无参数，所有信息将作为一个 Iterator 对象返回
properties()	返回包含属性名称的 Iterator，这些属性至少有一个错误
size()	返回错误的数目（整型数）

2 Struts 标记库

Struts 标记库

JSP 视窗组件所使用的 struts 标记库由四类标记组成：

2. Bean 标记：用来在 JSP 页中管理 bean
3. 逻辑标记：用来在 JSP 页中控制流程
4. HTML 标记：用来生成 HTML 标记，在表单中显示数据，使用会话 ID 对 URL 进行编程
5. 模板标记：使用动态模板构造普通格式的页

2.1 Bean 标记

这个标记库中包含用于定义新 bean、访问 bean 及其属性的标记。Struts 框架提供了多种自定义标记用来在 JSP 页中处理 JavaBean。这些标记被封装在一个普通的标记库中，在文件 struts-bean.tld 中定义了它的标记库描述器。Bean 标记库将标记定义在四个子类别中：

- 创建和复制 bean 的标记
- 脚本变量定义标记
- bean 翻译标记
- 消息国际化标记

2.1.1 Bean 复制标记

可定义新 bean，可复制现有 bean，还可从现有 bean 复制属性。

<bean:define>标记用来：

- 定义新字符串常数
- 将现有的 bean 复制到新定义的 bean 对象
- 复制现有 bean 的属性来创建新的 bean

<bean:define>标记属性：

属性	描述
Id	新定义的 bean 脚本变量名称，必须设置
Type	定义引入脚本变量的类
Value	为 id 属性定义的脚本变量分配一个新的对象
Name	目标 bean 的名称。若 value 属性没有设置，这个属性就必须设置
property	Name 属性定义的 bean 的属性名称，用来定义新的 bean

Scope	源 bean 的作用域。若没有设置，搜索范围是从页作用域到应用程序作用域
toScope	目标 bean 的作用域。若没有设置，默认值是页作用域

例如：定义一个 bean:

```
<bean:define id=”test” value=”this is a test”/>
```

源 bean 在页作用域中被拷贝大哦请求作用域中的另一个 bean:

```
<bean:define id=”targetBean” name=”sourceBean”
             scope=”page” toScope=”request”/>
```

2.1.2 定义脚本变量的标记

从多种资源中定义和生成脚本变量，这些资源包括 cookie,请求参数，HTTP 标头等等。属性如下：

属性	**描述**
Id	脚本变量和要定义的页作用域属性的名称
Name	cookie/标头/参数的名称
multiple	如果这个属性设置了任意一个数值，所有匹配的 cookie 都会被积累并存储到一个 Cookie[](一个数组)类型的 bean 里。若无设置，指定 cookie 的第一个值将作为 Cookie 类型的值
Value	如果没有匹配的 cookie 或数值，就返回这个属性指定的默认值

例如：

```
<bean:cookie id=”myCookie” name=”userName”/>
```

脚本变量名称是 myCookie，用来创建这个属性的 cookie 的名称是 userName。

```
<bean:header id=”myHeader” name=”Accept-Language”/>
```

脚本变量名称是 myHeader,请求标头的名称是 Accept-Language.

```
<bean:parameter id=”myParameter” name=”myParameter”>
```

脚本变量名称是 myPatameter,它保存的请求参数的名称也是 myParameter.

<bean:include>标记将对一个资源的响应进行检索，并引入一个脚本变量和字符串类型的页作用域属性。这个资源可以是一个页，一个 ActionForward 或一个外部 URL。与<jsp:include>的不同是资源的响应被存储到一个页作用域的 bean 中，而不是写入到输出流。属性如下：

属性	**描述**
Id	脚本变量和要定义的页作用域属性的名称
Page	一个内部资源
forward	一个 ActionForward
Href	要包含的资源的完整 URL

例如：

```
<bean:include id=”myInclude” page=”MyJsp?x=1”/>
```

脚本变量的名称是 myInclude，要检索的响应来自资源 MyJsp?x=1。

<bean:resource>标记将检索 web 应用中的资源，并引入一个脚本变量和 InputStream 或字符串类型的页作用域属性。如果在检索资源时发生问题，就会产生一个请求时间异常。属性如下：

属性	**描述**

Id	脚本变量和要定义的页作用域属性的名称
Name	资源的相对路径
Input	如果这个属性不存在，资源的类型就是字符串

例如：

```
<bean:resource id=”myResource” name=”/WEB-INF/images/myResource.xml”/>
```

脚本变量的名称是 myResource，要检索的资源的名称是 myResource.xml。

2.1.3 显示 Bean 属性

标记库中定义了<bean:write>标记，用来将 bean 的属性输送到封装的 JSP 页写入器。这个标记与<jsp:getProperty>类似，属性如下：

属性	**描述**
Name	要进行属性显示的 bean 的名称
property	要显示的属性的名称。如果这个属性类有 java.beans.PropertyEditor,getAsText()或 toString 方法会被调用
Scope	Bean 的作用域，若没有设置，搜索范围是从页到应用程序作用域
Filter	如果设置 true,属性中的所有特殊 HTML 字符都将被转化为相应的实体引用
Ignore	如果设置 false，当发现属性时会产生一个请求时间异常，否则返回 null

例如：

```
<bean:write name=”myBean” property=”myProperty” scope=”request”
                                                    filter=”true”/>
```

myBean 的属性 myProperty 将会被显示，作用域为请求，如果发现任何 HTML 特殊字符都将被转化为相应的实体引用。

2.1.4 消息标记和国际化

strtus 框架支持国际化和本地化。用户在他们的计算机中定义自己所在的区域，当 web 应用程序需要输出一条消息时，它将引用一个资源文件，在这个文件中所有的消息都使用了适当的语言。一个应用程序可能提供了很多资源文件，每个文件提供了用不同语言编写的消息。如果没有找到所选语言的资源文件，就将使用默认的资源文件。

struts 框架对国际化的支持是使用<bean:message>标记，以及使用 java.util 数据包中定义的 Locale 和 ResourceBundle 类来实现 Java2 平台对这些任务的支持。Java.text.MessageFormat 类定义的技术可以支持消息的格式。利用此功能，开发人员不需了解这些类的细节就可进行国际化和设置消息的格式。

用 strtus 实现国际化和本地化：

第一步要定义资源文件的名称，这个文件会包含用默认语言编写的在程序中会出现的所有消息。这些消息以“关键字-值”的形式存储，如下：

```
error.validation.location = The entered location is invalid
```

这个文件需要存储在类的路径下，而且它的路径要作为初始化参数传送给 ActionServlet 作为参数进行传递时，路径的格式要符合完整 Java 类的标准命名规范。比如，如果资源文件存储在 WEB-INF\classes 目录中，文件名是

ApplicationResources.properties，那么需要传递的参数值是 ApplicationResources。如果文件在 WEB-INF\classes\com\test 中，那么参数值就应该是 com.test.ApplicationResources.

为了实现国际化，所有的资源文件必须都存储在基本资源文件所在的目录中。基本资源文件包含的是用默认地区语言-本地语言编写的消息。如果基本资源文件的名称是 ApplicationResources.properties,那么用其他特定语言编写的资源文件的名称就应该是 ApplicationResources_xx.properties(xx 为 ISO 编码，如英语是 en)。因此这些文件应包含相同的关键字，但关键字的值是用特定语言编写的。

ActionServlet 的区域初始化参数必须与一个 true 值一起传送，这样 ActionServlet 就会在用户会话中的 Action.LOCALE_KEY 关键字下存储一个特定用户计算机的区域对象。现在可以运行一个国际化的 web 站点，它可以根据用户计算机上的设置的区域自动以相应的语言显示。

我们还可以使用特定的字符串来替换部分消息，就象用 java.text.MessageFormat 的方法一样：

```
error.invalid.number = The number {0} is valid
```

我们可以把字符串{0}替换成任何我们需要的数字。<bean:message>标签属性如下：

属性	描述
Key	资源文件中定义消息关键字
Locale	用户会话中存储的区域对象的属性名称。若没有设置，默认值是 Action.LOCALE_KEY
Bundle	在应用程序上下文中，存储资源对象的属性的名称。如果没有设置这个属性，默认值是 Action.MESSAGE_KEY
arg0	第一个替换参数值
arg1	第二个替换参数值
arg2	第三个替换参数值
arg3	第四个替换参数值

例如：资源文件中定义了一个消息：

```
info.myKey = The numbers entered are {0},{1},{2},{3}
```

我们可使用下面的消息标记：

```
<bean:message key=”info.myKey” arg0=”5” arg1=”6” arg2=”7” arg3=”8”/>
```

这个信息标记输出到 JSP 页会显示为：The numbers entered are 5,6,7,8

2.2 逻辑标记

逻辑库的标记能够用来处理外观逻辑而不需要使用 scriptlet。Struts 逻辑标签库包含的标记能够有条件地产生输出文本，在对象集合中循环从而重复地产生输出文本，以及应用程序流程控制。它也提供了一组在 JSP 页中处理流程控制的标记。这些标记封装在文件名为 struts-logic.tld 的标记包中。逻辑标记库定义的标记能够执行下列三个功能：

- 条件逻辑
- 重复
- 转发/重定向响应

2.2.1 条件逻辑

struts 有三类条件逻辑。第一类可以比较下列实体与一个常数的大小：

- cookie
- 请求参数
- bean 或 bean 的参数
- 请求标头

以下列出了这一类标记：

标记	功能
<equal>	如果常数与被定义的实体相等，返回 true
<notEqual>	如果常数与被定义的实体不相等，返回 true
<greaterEqual>	如果常数大于等于被定义的实体，返回 true
<lessEqual>	如果常数小于等于被定义的实体，返回 true
<lessThan>	如果常数小于被定义的实体，返回 true
<greaterThan>	如果常数大于被定义的实体，返回 true

这一类的所有标记有相同的属性

属性	描述
Value	要进行比较的常数值
Cookie	要进行比较的 HTTP cookie 的名称
Header	要进行比较的 HTTP 请求标头的名称
parameter	要进行比较的 HTTP 请求参数的名称
Name	如果要进行比较的是 bean 或 bean 的属性，则这个属性代表 bean 的名称
property	要进行比较的 bean 属性的名称
Scope	Bean 的作用域，如果没有指定作用域，则它的搜索范围是从页到应用程序

例如：

```
<logic:equal parameter=”name” value=”SomeName”>
  The entered name is SomeName
</logic:equal>
```

判断名为 ”name” 的请求参数的值是否是 ”SomeName” 。

```
<logic:greaterThan name=”bean” property=”prop” scope=”page”
                                                value=”7”>
  The value of bean.Prop is greater than 7
</logic:greaterThan>
```

判断在页的作用域中是否有一个名为 ”bean” 的 bean，它有一个 prop 属性，这个属性的值是否大于 7。如果这个属性能够转化为数值，就进行数值比较，否则就进行字符串比较。

第二类条件标记定义了两个标记：

- <logic:present>
- <logic:notPresent>

它们的功能是在计算标记体之前判断特定的项目是否存在。标记的属性和属性值决定了要进行检查的项目。

属性	描述

Cookie	由这个属性指定的 cookie 将被检查是否存在
Header	由这个属性指定的请求标头将被检查是否存在
parameter	由这个属性指定的请求参数将被检查是否存在
Name	如果没有设置 property 属性，那么有这个属性指定的 bean 将被检查是否存在。如果设置了，那么 bean 和 bean 属性都将被检查是否存在。
property	检查有 name 属性指定的 bean 中是否存在指定的属性
Scope	如果指定了 bean 的名称，这就是 bean 的作用域。如果没有指定作用域，搜索的范围从页到应用程序作用域。
Role	检查当前已经确认的用户是否属于特殊的角色
User	检查当前已经确认的用户是否有特定的名称

例如：

```
<logic:notPresent name=”bean” property=”prop” scope=”page”>
  The bean property bean.prop is present
</logic:notPresent>
```

标记判断在页作用域中是否存在一个名为 ”bean” 的 bean，这个 bean 有一个 prop 属性。

第三类条件标记比较复杂，这些标记根据模板匹配的结果检查标记体的内容。换句话说，这些标记判断一个指定项目的值是否是一个特定常数的子字符串：

- <logic:match>
- <logic:notMatch>

这些标记允许 JSP 引擎在发现了匹配或是没有发现时计算标记主体。属性如下：

属性	**描述**
Cookie	要进行比较的 HTTP cookie 的名称
Header	要进行比较的的 HTTP 标头 的名称
parameter	要进行比较的的 HTTP 请求参数的名称
Name	若要对 bean 或 bean 的属性进行比较，这个属性是用户指定 bean 的名称
location	如果设置了这个属性的值，将会在这个指定的位置(索引值)进行匹配
scope	如果对 bean 进行比较，这个属性指定了 bean 的作用域。如果没有设置这个参数，搜索范围是从页到应用程序作用域
property	要进行比较的 bean 的属性名称
value	要进行比较的常数值

例如：

```
<logic:match parameter=”name” value=”xyz” location=”1”>
  The parameter name is a sub-string of the string xyz from index 1
</logic:match>
```

标记检查名为 ”name” 的请求参数是否是 ”xyz” 的子字符串，但是子字符串必须从 ”xyz” 的索引位置 1 开始（也就是说子字符串必须是 ”y” 或 ”yz”）。

2.2.2 重复标记

在逻辑标记库中定义了<logic:iterate>标记,它能够根据特定集合中元素的数目对标记体的内容进行重复的检查。集合的类型可以是 java.util.Iterator,java.util.Collection

,java.util.Map 或是一个数组。有三种方法可以定义这个集合：

- 使用运行时间表达式来返回一个属性集合的集合
- 将集合定义为 bean，并且使用 name 属性指定存储属性的名称。
- 使用 name 属性定义一个 bean，并且使用 property 属性定义一个返回集合的 bean 属性。

当前元素的集合会被定义为一个页作用域的 bean。属性如下，所有这些属性都能使用运行时表达式。

属性	描述
collection	如果没有设置 name 属性，它就指定了要进行重复的集合
Id	页作用域 bean 和脚本变量的名称，它保存着集合中当前元素的句柄
indexed	页作用域 JSP bean 的名称，它包含着每次重复完成后集合的当前索引
Length	重复的最大次数
Name	作为集合的 bean 的名称，或是一个 bean 名称，它由 property 属性定义的属性，是个集合
Offset	重复开始位置的索引
property	作为集合的 Bean 属性的名称
Scope	如果指定了 bean 名称，这个属性设置 bean 的作用域。若没有设置，搜索范围从页到应用程序作用域
Type	为当前定义的页作用域 bean 的类型

例如：

```
<logic:iterate id=”currentInt”
                collection=”<% =myList %>”
                type=”java.lang.Integer”
                offset=”1”
                length=”2”>
  <% =currentint %>
</logic:iterate>
```

代码将从列表中的第一个元素开始重复两个元素并且能够让当前元素作为页作用域和 java.lang.Integer 类型的脚本变量来使用。也就是说，如果 myList 包含元素 1，2，3，4 等，代码将会打印 1 和 2。

2.2.3 转发和重定向标记

转发标记

<logic:forward>标记能够将响应转发给重定向到特定的全局 ActionForward 上。ActionForward 的类型决定了是使用 PageContext 转发响应，还是使用 sendRedirect 将响应进行重定向。此标记只有一个 ”name ” 属性，用来指定全局 ActionForward 的名称，例如：

```
<logic:forward name=”myGlobalForward”/>
```

重定向标记

<logic:redirect>标记是一个能够执行 HTTP 重定向的强大工具。根据指定的不同属性，它能够通过不同的方式实现重定向。它还允许开发人员指定重定向 URL 的查询参数。属性如下：

属性	描述
Forward	映射了资源相对路径的 ActionForward
Href	资源的完整 URL
Page	资源的相对路径
Name	Map 类型的页名称，请求，会话或程序属性的名称，其中包含要附加大哦重定向 URL（如果没有设置 property 属性）上的“名称-值”参数。或是具有 Map 类型属性的 bean 名称，其中包含相同的信息（没有设置 property 属性）
Property	Map 类型的 bean 属性的名称。Bean 的名称由 name 属性指定。
Scope	如果指定了 bean 的名称，这个属性指定搜索 bean 的范围。如果没有设置，搜索范围从页到应用程序作用域
ParamID	定义特定查询参数的名称
ParamName	字符串类型的 bean 的名称，其中包含查询参数的值(如果没有设置 paramProperty 属性)；或是一个 bean 的名称，它的属性(在 paramProperty 属性中指定)包含了查询参数值
paramProperty	字符串 bean 属性的名称，其中包含着查询参数的值
ParamScope	ParamName 定义的 bean 的搜索范围

使用这个标记时至少要指定 forward,href 或 page 中的一个属性，以便标明将响应重定向到哪个资源。

2.3 HTML 标记

Struts HTML 标记可以大致地分为以下几个功能：

- 显示表单元素和输入控件
- 显示错误信息
- 显示其他 HTML 元素

2.3.1 显示表单元素和输入控件

struts 将 HTML 表单与为表单操作而定义的 ActionForm bean 紧密联系在一起。表单输入字段的名称与 ActionForm bean 里定义的属性名称是对应的。当第一次显示表单时，表单的输入字段是从 ActionForm bean 中移植过来的，当表单被提交时，请求参数将移植到 ActionForm bean 实例。

所有可以在<form>标记中使用的用来显示 HTML 输入控件的内嵌标记都使用下列属性来定义 JavaScript 事件处理器。

属性	描述
Onblur	字段失去了焦点
Onchange	字段失去了焦点并且数值被更改了
Onclick	字段被鼠标点击
Ondblclick	字段被鼠标双击
Onfocus	字段接收到输入焦点
Onkeydown	字段拥有焦点并且有键按下

onkeypress	字段拥有焦点并且有键按下并释放
Onkeyup	字段拥有焦点并且有键被释放
onmousedown	鼠标指针指向字段并且点击
onmousemove	鼠标指针指向字段并且在字段内移动
onmouseout	鼠标指针指向控件，但是指针在元素外围移动
onmouseover	鼠标指针没有指向字段，但是指针在元素内部移动
Onmouseup	鼠标指针指向字段，并且释放了鼠标按键

<form>元素中能够被定义的其他一般属性有：

属性	**描述**
Accesskey	定义访问输入字段的快捷键
Style	定义输入字段的样式
styleClass	定义输入字段的样式表类
Tabindex	输入字段的 tab 顺序

a) 表单标记

<html:form>标记用来显示 HTML 标记，可以指定 AcitonForm bean 的名称和它的类名。如果没有设置这些属性，就需要有配置文件来指定 ActionMapping 以表明当前输入的是哪个 JSP 页，以及从映射中检索的 bean 名和类。如果在 ActionMapping 指定的作用域中没有找到指定的名称，就会创建并存储一个新的 bean，否则将使用找到的 bean。

<form>标记能够包含与各种 HTML 输入字段相对应的子标记。

<html:form>标记属性如下：

属性	**描述**
Action	与表单相关的操作。在配置中，这个操作也用来标识与表单相关的 ActionForm bean
Enctype	表单 HTTP 方法的编码类型
Focus	表单中需要初始化焦点的字段
Method	表单使用的 HTTP 方法
Name	与表单相关的 ActionForm bean 的名称。如果没有设置这个属性，bean 的名称将会从配置信息中获得
Onreset	表单复位时的 JavaScript 事件句柄
Onsubmit	表单提交时的 JavaScript 事件句柄
Scope	搜索 ActionForm bean 的范围。如果没有设置，将从配置文件中获取
Style	使用的格式
styleClass	这个元素的格式表类
Type	ActionForm bean 的完整名称。如果没有设置，将从配置文件获得

例如：

```
<html:form action=”validateEmploee.do” method=”post”>
</html:form>
```

与表单相关的操作路径是 validateEmployee,而表单数据是通过 POST 传递的。对于这个表单来说，ActionForm bean 的其他信息，如 bean 名称类型，作用域，都是从表单指定操作的 ActionMapping 中检索得到的：

```
<form-beans>
  <form-bean name=”empForm” type=”com.example.EmployeeForm”/>
</form-beans>
<action-mappings>
  <action path=”/validateEmployee”
          type=”com.example.ValidateExampleAction”
          name=”empForm”
          scope=”request”
          input=”/employeeInput.jsp”>
      <forward name=”success” path=”/employeeOutput.jsp”>
   </action>
</action-mapping>
```

如果配置文件中包含上述信息，并且请求 URI 的*.do 被映射到 ActionServlet，与表单相关的 ActionForm bean 的名称，类型和作用域分别是 empForm,com.example.EmployeeForm 和 request.这些属性也可以使用<html:form>标记属性进行显示的定义。

以下标记必须嵌套在<html:form>标记里

b) 按钮和取消标记

<html:button>标记显示一个按钮控件；<html:cancel>标记显示一个取消按钮。属性如下：

属性	描述
Property	定义在表单被提交时返回到服务器的请求参数的名称
Value	按钮上的标记

c) 复位和提交标记

<html:reset>和<html:submit>标记分别能够显示 HTML 复位按钮和提交按钮。

d) 文本和文本区标记

<html:text>和<html:textarea>标记分别 HTML 文本框和文本区，属性如下：

属性	描述
Property	定义当表单被提交时送回到服务器的请求参数的名称，或用来确定文本元素当前值的 bean 的属性名称
Name	属性被查询的 bean 的名称，它决定了文本框和文本区的值。如果没有设置，将使用与这个内嵌表单相关的 ActionForm 的名称

<html:text>标记还有以下属性：

属性	描述
Maxlength	能够输入的最大字符数

Size	文本框的大小（字符数）

<html:textarea>标记特有的属性如下：

属性	描述
Rows	文本区的行数
Cols	文本区的列数

e) 检查框和复选框标记

<html:checkbox>标记能够显示检查框控件。<html:multibox>标记能够显示 HTML 复选框控件，请求对象在传递检查框名称时使用的 getParameterValues()调用将返回一个字符串数组。属性如下：

属性	描述
Name	Bean 的名称，其属性会被用来确定检查是否以选中的状态显示。如果没有设置，将使用与这个内嵌表单相关的 ActionFrom bean 的名称。
Property	检查框的名称，也是决定检查框是否以选中的状态显示的 bean 属性名称。在复选框的情况下，这个属性必须是一个数组。
Value	当检查框被选中时返回到服务器的请求参数的值

例如：

<html:checkbox property=”married” value=”Y”/>

一个名为 married 的检查框，在表单提交时会返回一个”Y”.

f) 文件标记

<html:file>标记可以显示 HTML 文件控件。属性如下：

属性	描述
Name	Bean 的名称，它的属性将确定文件控件中显示的内容。如果没设置，将使用与内嵌表单相关的 ActionForm bean 的名称
property	这个属性定义了当表单被提交时送回到服务器的请求参数的名称，以及用来确定文件控件中显示内容的 bean 属性名称
Accept	服务器能够处理的内容类型集。它也将对客户浏览器对话框中的可选文件类型进行过滤
Value	按钮上的标记，这个按钮能够在本地文件系统中浏览文件

g) 单选钮标记

<html:radio>标记用来显示 HTML 单选钮控件，属性如下：

属性	描述
Name	Bean 的名称，其属性会被用来确定单选钮是否以选中的状态显示。如果没有设置，将使用与这个内嵌表单相关的 ActionFrom bean 的名称。
property	当表单被提交时送回到服务器的请求参数的名称，以及用来确定单选钮是否以被选中状态进行显示的 bean 属性的名称

Value	当单选钮被选中时返回到服务器的值

h) 隐藏标记

<html:hidden>标记能够显示 HTML 隐藏输入元素，属性如下：

属性	描述
Name	Bean 的名称，其属性会被用来确定隐藏元素的当前值。如果没有设置，将使用与这个内嵌表单相关的 ActionFrom bean 的名称。
property	定义了当表单被提交时送回到服务器的请求参数的名称，以及用来确定隐藏元素当前值的 bean 属性的名称
Value	用来初始化隐藏输入元素的值

i) 密码标记

<html:password>标记能够显示 HTML 密码控件，属性如下：

属性	描述
maxlength	能够输入的最大字符数
Name	Bean 的名称，它的属性将用来确定密码元素的当前值。如果没有设置，将使用与这个内嵌表单相关的 ActionFrom bean 的名称。
property	定义了当表单被提交时送回到服务器的请求参数的名称，以及用来确定密码元素当前值的 bean 属性的名称
redisplay	在显示这个字段时，如果相应的 bean 属性已经被设置了数据，这个属性决定了是否显示密码的内容
Size	字段的大小

j) 选择标记

<html:select>标记能够显示 HTML 选择控件，属性如下：

属性	描述
multiple	表明这个选择控件是否允许进行多选
Name	Bean 的名称，它的属性确定了哪个。如果没有设置，将使用与这个内嵌表单相关的 ActionFrom bean 的名称。
property	定义了当表单被提交时送回到服务器的请求参数的名称，以及用来确定哪个选项需要被选中的 bean 属性的名称
Size	能够同时显示的选项数目
Value	用来表明需要被选中的选项

k) 选项标记 (这个元素需要嵌套在<html:select>标记里)

<html:option>标记用来显示 HTML 选项元素集合，属性如下：

属性	描述
collection	Bean 集合的名称，这个集合存储在某个作用域的属性中。选项的数目与集合中元素的数目相同。Property 属性能够定义选项值所使用的 bean 属性，而 labelProperty 属性定义选项标记所使用的 bean 的属性
labelName	用来指定存储于某个作用域的 bean，这个 bean 是一个字符串的集合，能够定义<html:option>元素的标记(如果标志与值不相同)
labelProperty	与 collection 属性共同使用时，用来定义了存储于某个作用域的 bean，这个 bean 将返回一个字符串集合，能够用来写入<html:option>元素的 value 属性
Name	如果这是唯一被指定的属性，它就定义了存储于某个作用域的 bean，这个 bean 将返回一个字符串集合，能够用来写入<html:option>元素的 value 属性
property	这个属性在与 collection 属性共同使用时,定义了每个要显示选项值的独立 bean 的 name 属性。如果不是与 collection 属性共同使用，这个属性定义了由 name 属性指定的 bean 的属性名称(如果有 name 属性),或是定义了一个 ActionForm bean，这个 bean 将返回一个集合来写入选项的值

我们看一下这个标记的一些例子：

```
<html:option collection=”optionCollection” property=”optionValue”
                                                labelProperty=”optionLabel”/
>
```

标记假设在某个作用域中有一个名为 optionCollection 的集合，它包含了一些具有 optionValue 属性的独立的 bean，每个属性将作为一个选项的值。每个选项的标志由 bean 的 optionLabel 属性属性进行定义。

```
<html:option name=”optionValues” labelName=”optionLabels”/>
```

标记中 optionValues 代表一个存储在某个作用域中的 bean，它是一个字符串集合，能够用来写入选项的值，而 optionLabels 代表一个存储在某个作用域中的 bean，它也是一个字符串集合，能够用来写入选项的标志。

2.3.2.显示错误信息的标记

<html:errors>标记能够与 ActionErrors 结合在一起来显示错误信息。这个标记首先要从当前区域的资源文件中读取消息关键字 errors.header，然后显示消息的文本。接下去它会在 ActionErrors 对象(通常作为请求参数而存储在 Action.ERROR_KEY 关键字下)中循环，读取单个 ActionError 对象的消息关键字，从当前区域的资源文件中读取并格式化相应的消息，并且显示它们。然后它读取与 errors.footer 关键字相对应的消息并且显示出来。

通过定义 property 属性能够过滤要显示的消息，这个属性的值应该与 ActionErrors 对象中存储 ActionError 对象的关键字对应。属性如下：

属性	描述
Bundle	表示应用程序作用域属性的名称，它包含着消息资源，其默认值 Acion.MESSAGE_KEY
Locale	表示会话作用域属性的名称，它存储着用户当前登录的区域信息。其默认值是 Action.ERROR_KEY

Name	表示请求属性的名称，它存储着 ActionErrors 对象。其默认值是 Action.ERROR_KEY
property	这个属性指定了ActionErrors对象中存储每个独立ActionError对象的关键字，它可以过滤消息

例子：

```
<html:errors/>
```

显示集合中所有的错误。

```
<html:errors property=”missing.name”/>
```

显示存储在 missing.name 关键字的错误。

2.3.3.其他 HTML 标记

struts HTML 标记还定义了下列标记来显示其他 HTML 元素：

- <html:html> ： 显示 HTML 元素
- <html:img> ： 显示图象标记
- <html:link> ： 显示 HTML 链接或锚点
- <html:rewrite> ： 创建没有锚点标记的 URI

这些标记的详细内容请参照 struts 文档。

2.4. 模板标记

动态模板是模块化 WEB 页布局设计的强大手段。Struts 模板标记库定义了自定义标记来实现动态模板。

2.4.1.插入标记

<template:insert>标记能够在应用程序的 JSP 页中插入动态模板。这个标记只有一个 template 属性，用来定义模板 JSP 页。要插入到模板的页是有多个<template:put>标记来指定的，而这些标记被定义为<template:insert>标记的主体内容。

2.4.2.放置标记

<template:put>标记是<template:insert>标记内部使用的,用来指定插入到模板的资源。属性如下：

属性	**描述**
content	定义要插入的内容，比如一个 JSP 文件或一个 HTML 文件
direct	如果这个设置为 true，由 content 属性指定的内容将直接显示在 JSP 上而不是作为包含文件
Name	要插入的内容的名称
Role	如果设置了这个属性，只有在当前合法用户具有特定角色时才能进行内容的插入。

2.4.3.获得标记

在模板 JSP 页中使用<template:get>标记能够检索由<template:put>标记插入到 JSP 页的资源。属性如下：

属性	描述
Name	由<template:put>标记插入的内容的名称
Role	如果设置了这个属性，只有在当前合法用户具有特定角色时才能进行内容的检索

2.4.4.使用模板标记

首先编写一个模板 JSP 页，它将被所有的 web 页使用：

```
<html>
  <%@ taglib uri=”/template” prefix=”template” %>
  <head>
    <title></title>
  </head>
  <body>
    <table width=”100%” height=”100%” >
      <tr height=”10%”>
        <td>
          <template:get name=”header”/>
        </td>
      </tr>
      <tr height=”80%”>
        <td>
          <template:get name=”content”/>
        </td>
      </tr>
      <tr height=”10%”>
        <td>
          <template:get name=”footer”/>
        </td>
      </tr>
    </table>
  </body>
</html>
```

我们将这个文件命名为 template.jsp。这个文件使用<template:get>标记来获得由 JSP 页使用<template:put>标记提供的内容，并且将内容在一个 HTML 表格中显示出来。这三个内容是标题，内容和页脚。典型的内容 JSP 会是这样：

```
<%@ taglib uri=”/template” prefix=”/template” %>
<template:insert template=”template.jsp”>
  <template:put name=”header” content=”header.html”/>
```

```
  <template:put name=”content” content=”employeeList.jsp”/>
  <template:put name=”footer” content=”footer.html”/>
</template:insert>
```

这个应用程序 JSP 页使用<template:insert 标记来定义模板，然后使用<template:put>标记将特定内容名称指定的资源放到模板 JSP 页中。如果我们有上百个布局相同的页，但突然想改变这个模板，我们只需要改变 template.jsp 文件。

待续