**Struts2技术速查**

目录

[1 Struts2简介 4](#_Toc419883721)

[2 搭建Struts2开发环境 4](#_Toc419883722)

[2.1 搭建Struts2开发环境--开发Struts2应用依赖的jar文件 5](#_Toc419883723)

[2.2 搭建Struts2开发环境-- Struts2应用的配置文件 5](#_Toc419883724)

[2.3 搭建Struts2开发环境--Struts2在web中的启动配置 5](#_Toc419883725)

[3 第一个Struts2应用--HelloWorld 6](#_Toc419883726)

[3.1 Struts.xml配置中的包介绍 6](#_Toc419883727)

[3.2 HelloWorld例子--Action类 7](#_Toc419883728)

[3.3 HelloWorld例子--JSP页面 7](#_Toc419883729)

[3.4 HelloWorld例子--访问HelloWorld应用 8](#_Toc419883730)

[4 Action名称的搜索顺序 8](#_Toc419883731)

[5 Action配置中的各项默认值 9](#_Toc419883732)

[6 Action中result的各种转发类型 9](#_Toc419883733)

[7 多个Action共享一个视图--全局result配置 10](#_Toc419883734)

[8 为Action的属性注入值 10](#_Toc419883735)

[9 指定需要Struts 2处理的请求后缀 10](#_Toc419883736)

[10 细说常量定义 11](#_Toc419883737)

[11 常用的常量介绍 11](#_Toc419883738)

[12 Struts2的处理流程 12](#_Toc419883739)

[13 为应用指定多个struts配置文件 13](#_Toc419883740)

[14 动态方法调用 13](#_Toc419883741)

[15 使用通配符定义action 14](#_Toc419883742)

[16 接收请求参数 14](#_Toc419883743)

[16.1 采用基本类型接收请求参数(get/post) 14](#_Toc419883744)

[16.2采用复合类型接收请求参数 15](#_Toc419883745)

[17 关于struts2.1.6接收中文请求参数乱码问题 15](#_Toc419883746)

[18 自定义类型转换器 15](#_Toc419883747)

[19 自定义全局类型转换器 16](#_Toc419883748)

[20 访问或添加request/session/application属性 17](#_Toc419883749)

[21 获取HttpServletRequest / HttpSession / ServletContext / HttpServletResponse对象 17](#_Toc419883750)

[22 文件上传 18](#_Toc419883751)

[23 多文件上传 18](#_Toc419883752)

[24 自定义拦截器 19](#_Toc419883753)

[25 输入校验 20](#_Toc419883754)

[25.1 手工编写代码实现对action中所有方法输入校验 20](#_Toc419883755)

[25.2 手工编写代码实现对action指定方法输入校验 21](#_Toc419883756)

[25.3 输入校验的流程 21](#_Toc419883757)

[25.4 基于XML配置方式实现对action的所有方法进行输入校验 22](#_Toc419883758)

[25.5 编写校验文件时，不能出现帮助信息 22](#_Toc419883759)

[25.6 struts2提供的校验器列表 23](#_Toc419883760)

[25.7 校验器的使用例子 23](#_Toc419883761)

[25.8 基于XML配置方式对指定action方法实现输入校验 24](#_Toc419883762)

[25.9 基于XML校验的一些特点 25](#_Toc419883763)

[26 国际化 25](#_Toc419883764)

[26.1 配置全局资源与输出国际化信息 26](#_Toc419883765)

[26.2 国际化—输出带占位符的国际化信息 26](#_Toc419883766)

[26.3 国际化—包范围资源文件 27](#_Toc419883767)

[26.4 国际化—Action范围资源文件 27](#_Toc419883768)

[26.5 国际化—JSP中直接访问某个资源文件 27](#_Toc419883769)

[27 OGNL表达式语言 28](#_Toc419883770)

[27.1 为何使用EL表达式能够访问valueStack中对象的属性 29](#_Toc419883771)

[27.2 采用OGNL表达式创建List/Map集合对象 30](#_Toc419883772)

[27.3 采用OGNL表达式判断对象是否存在于集合中 30](#_Toc419883773)

[27.4 OGNL表达式的投影功能 31](#_Toc419883774)

[28 Stuts2 常用标签 31](#_Toc419883775)

[28.1 property标签 31](#_Toc419883776)

[28.2 iterator标签 32](#_Toc419883777)

[28.3 if/elseif/else标签 32](#_Toc419883778)

[28.4 url标签 33](#_Toc419883779)

[28.5表单标签\_radio单选框 34](#_Toc419883780)

[28.6 表单标签\_select下拉选择框 35](#_Toc419883781)

[28.7 <s:token />标签防止重复提交 35](#_Toc419883782)

[29 Struts2+Spring2.5+Hibernate3.3整合开发 36](#_Toc419883783)

[29.1 第一步在Spring中配置数据源 37](#_Toc419883784)

[29.2 第二步集成进hibernate 38](#_Toc419883785)

[29.3 第三步使用Spring容器管理事务服务 38](#_Toc419883786)

[29.4 在web容器中使用Listener实例化spring容器和配置struts2 39](#_Toc419883787)

[29.5 配置struts2 39](#_Toc419883788)

[29.6 struts2的标签 41](#_Toc419883789)

# 1 Struts2简介

Struts2是在WebWork2基础发展而来的。和struts1一样， Struts2也属于MVC框架。不过有一点大家需要注意的是：尽管Struts2和struts1在名字上的差别不是很大，但Struts2和struts1在代码编写风格上几乎是不一样的。那么既然有了struts1，为何还要推出struts2。主要是因为struts2有以下优点：

1 > 在软件设计上Struts2没有像struts1那样跟Servlet API和struts API有着紧密的耦合，Struts2的应用可以不依赖于Servlet API和struts API。 Struts2的这种设计属于无侵入式设计，而Struts1却属于侵入式设计。

public class OrderListAction extends Action {

public ActionForward execute(ActionMapping mapping, ActionForm form,

HttpServletRequest request, HttpServletResponse response)

throws Exception {

}

}

2> Struts2提供了拦截器，利用拦截器可以进行AOP编程，实现如权限拦截等功能。

3> Strut2提供了类型转换器，我们可以把特殊的请求参数转换成需要的类型。在Struts1中，如果我们要实现同样的功能，就必须向Struts1的底层实现BeanUtil注册类型转换器才行。

4> Struts2提供支持多种表现层技术，如：JSP、freeMarker、Velocity等。

5> Struts2的输入校验可以对指定方法进行校验，解决了Struts1长久之痛。

6> 提供了全局范围、包范围和Action范围的国际化资源文件管理实现。

# 2 搭建Struts2开发环境

搭建Struts2环境时，我们一般需要做以下几个步骤的工作：

1>找到开发Struts2应用需要使用到的jar文件

2>编写Struts2的配置文件

3>在web.xml中加入Struts2框架启动配置

## 2.1 搭建Struts2开发环境--开发Struts2应用依赖的jar文件

可以到http://struts.apache.org/download.cgi#struts2014下载struts-2.x.x-all.zip，目前最新版为2.1.8。下载完后解压文件，开发struts2应用需要依赖的jar文件在解压目录的lib文件夹下。不同的应用需要的JAR包是不同的。下面给出了开发Struts 2程序最少需要的JAR。

struts2-core-2.x.x.jar :Struts 2框架的核心类库

xwork-core-2.x.x.jar :XWork类库，Struts 2在其上构建

ognl-2.6.x.jar :对象图导航语言（Object Graph Navigation Language），struts2框架通过其读写对象的属性

freemarker-2.3.x.jar :Struts 2的UI标签的模板使用FreeMarker编写

commons-logging-1.x.x.jar :ASF出品的日志包，Struts 2框架使用这个日志包来支持Log4J和JDK 1.4+的日志记录。

commons-fileupload-1.2.1.jar 文件上传组件，2.1.6版本后必须加入此文件

## 2.2 搭建Struts2开发环境-- Struts2应用的配置文件

Struts2默认的配置文件为struts.xml ，该文件需要存放在WEB-INF/classes下，该文件的配置模版如下：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

</struts>

## 2.3 搭建Struts2开发环境--Struts2在web中的启动配置

在struts1.x中， struts框架是通过Servlet启动的。在struts2中，struts框架是通过Filter启动的。他在web.xml中的配置如下：

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

<!-- 自从Struts 2.1.3以后，下面的FilterDispatcher已经标注为过时

<filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class> -->

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

在StrutsPrepareAndExecuteFilter的init()方法中将会读取类路径下默认的配置文件struts.xml完成初始化操作。

注意：struts2读取到struts.xml的内容后，以javabean形式存放在内存中，以后struts2对用户的每次请求处理将使用内存中的数据，而不是每次都读取struts.xml文件

# 3 第一个Struts2应用--HelloWorld

在默认的配置文件struts.xml 中加入如下配置：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<package name="itcast" namespace="/test" extends="struts-default">

<action name="helloworld" class="cn.itcast.action.HelloWorldAction" method="execute" >

<result name="success">/WEB-INF/page/hello.jsp</result>

</action>

</package>

</struts>

## 3.1 Struts.xml配置中的包介绍

<package name="itcast" namespace="/test" extends="struts-default">

<action name="helloworld" class="cn.itcast.action.HelloWorldAction" method="execute" >

<result name="success">/WEB-INF/page/hello.jsp</result>

</action>

</package>

在struts2框架中使用包来管理Action，包的作用和java中的类包是非常类似的，它主要用于管理一组业务功能相关的action。在实际应用中，我们应该把一组业务功能相关的Action放在同一个包下。

配置包时必须指定name属性，该name属性值可以任意取名，但必须唯一，他不对应java的类包，如果其他包要继承该包，必须通过该属性进行引用。包的namespace属性用于定义该包的命名空间，命名空间作为访问该包下Action的路径的一部分，如访问上面例子的Action，访问路径为：/test/helloworld.action。 namespace属性可以不配置，对本例而言，如果不指定该属性，默认的命名空间为“”（空字符串）。

通常每个包都应该继承struts-default包， 因为Struts2很多核心的功能都是拦截器来实现。如：从请求中把请求参数封装到action、文件上传和数据验证等等都是通过拦截器实现的。 struts-default定义了这些拦截器和Result类型。可以这么说：当包继承了struts-default才能使用struts2提供的核心功能。 struts-default包是在struts2-core-2.x.x.jar文件中的struts-default.xml中定义。 struts-default.xml也是Struts2默认配置文件。 Struts2每次都会自动加载 struts-default.xml文件。

包还可以通过abstract=“true”定义为抽象包，抽象包中不能包含action。

## 3.2 HelloWorld例子--Action类

例子中使用到的cn.itcast.action.HelloWorldAction类如下：

package cn.itcast.action;

public class HelloWorldAction{

private String message;

public String getMessage() {

return message;

}

public void setMessage(String message) {

this.message = message;

}

public String execute() {

this.message = "我的第一个struts2应用";

return "success";

}

}

## 3.3 HelloWorld例子--JSP页面

例子中使用到的/WEB-INF/page/hello.jsp如下：

<%@ page language="java" pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

<title>第一个struts2应用</title>

</head>

<body>

${message } <br>

</body>

</html>

可以使用EL表达式访问Action中的属性。

## 3.4 HelloWorld例子--访问HelloWorld应用

在struts1中，通过<action path=“/test/helloworld”>节点的path属性指定访问该action的URL路径。在struts2中，情况就不是这样了，访问struts2中action的URL路径由两部份组成：包的命名空间+action的名称，例如访问本例子HelloWorldAction的URL路径为：/test/helloworld (注意：完整路径为：http://localhost:端口/内容路径/test/helloworld)。另外我们也可以加上.action后缀访问此Action。

<package name="itcast" namespace="/test" extends="struts-default">

<action name="helloworld" class="cn.itcast.action.HelloWorldAction" method="execute" >

<result name="success">/WEB-INF/page/hello.jsp</result>

</action>

</package>

# 4 Action名称的搜索顺序

1．获得请求路径的URI，例如url是：http://server/struts2/path1/path2/path3/test.action

2．首先寻找namespace为/path1/path2/path3的package，如果不存在这个package则执行步骤3；如果存在这个package，则在这个package中寻找名字为test的action，当在该package下寻找不到action 时就会直接跑到默认namaspace的package里面去寻找action（默认的命名空间为空字符串“” ） ，如果在默认namaspace的package里面还寻找不到该action，页面提示找不到action

3．寻找namespace为/path1/path2的package，如果不存在这个package，则转至步骤4；如果存在这个package，则在这个package中寻找名字为test的action，当在该package中寻找不到action 时就会直接跑到默认namaspace的package里面去找名字为test的action ，在默认namaspace的package里面还寻找不到该action，页面提示找不到action

4．寻找namespace为/path1的package，如果不存在这个package则执行步骤5；如果存在这个package，则在这个package中寻找名字为test的action，当在该package中寻找不到action 时就会直接跑到默认namaspace的package里面去找名字为test的action ，在默认namaspace的package里面还寻找不到该action，页面提示找不到action

5．寻找namespace为/的package，如果存在这个package，则在这个package中寻找名字为test的action，当在package中寻找不到action或者不存在这个package时，都会去默认namaspace的package里面寻找action，如果还是找不到，页面提示找不到action。

# 5 Action配置中的各项默认值

<package name="itcast" namespace="/test" extends="struts-default">

<action name="helloworld" class="cn.itcast.action.HelloWorldAction" method="execute" >

<result name="success">/WEB-INF/page/hello.jsp</result>

</action>

</package>

1>如果没有为action指定class，默认是ActionSupport。

2>如果没有为action指定method，默认执行action中的execute() 方法。

3>如果没有指定result的name属性，默认值为success。

# 6 Action中result的各种转发类型

<action name="helloworld" class="cn.itcast.action.HelloWorldAction">

<result name="success">/WEB-INF/page/hello.jsp</result>

</action>

result配置类似于struts1中的forward，但struts2中提供了多种结果类型，常用的类型有： dispatcher(默认值)、 redirect 、 redirectAction 、 plainText。

在result中还可以使用${属性名}表达式访问action中的属性，表达式里的属性名对应action中的属性。如下：

<result type="redirect">/view.jsp?id=${id}</result>

下面是redirectAction 结果类型的例子，如果重定向的action中同一个包下：

<result type="redirectAction">helloworld</result>

如果重定向的action在别的命名空间下：

<result type="redirectAction">

<param name="actionName">helloworld</param>

<param name="namespace">/test</param>

</result>

plaintext:显示原始文件内容，例如：当我们需要原样显示jsp文件源代码 的时候，我们可以使用此类型。

<result name="source" type="plainText ">

<param name="location">/xxx.jsp</param>

<param name="charSet">UTF-8</param><!-- 指定读取文件的编码 -->

</result>

# 7 多个Action共享一个视图--全局result配置

当多个action中都使用到了相同视图，这时我们应该把result定义为全局视图。struts1中提供了全局forward，struts2中也提供了相似功能：

<package ....>

<global-results>

<result name="message">/message.jsp</result>

</global-results>

</package>

# 8 为Action的属性注入值

Struts2为Action中的属性提供了依赖注入功能，在struts2的配置文件中，我们可以很方便地为Action中的属性注入值。注意：属性必须提供setter方法。

public class HelloWorldAction{

private String savePath;

public String getSavePath() {

return savePath;

}

public void setSavePath(String savePath) {

this.savePath = savePath;

}

......

}

<package name="itcast" namespace="/test" extends="struts-default">

<action name="helloworld" class="cn.itcast.action.HelloWorldAction" >

<param name="savePath">/images</param>

<result name="success">/WEB-INF/page/hello.jsp</result>

</action>

</package>

上面通过<param>节点为action的savePath属性注入“/images”

# 9 指定需要Struts 2处理的请求后缀

前面我们都是默认使用.action后缀访问Action。其实默认后缀是可以通过常量”struts.action.extension“进行修改的，例如：我们可以配置Struts 2只处理以.do为后缀的请求路径：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<constant name="struts.action.extension" value="do"/>

</struts>

如果用户需要指定多个请求后缀，则多个后缀之间以英文逗号（,）隔开。如：

<constant name="struts.action.extension" value="do,go"/>

# 10 细说常量定义

常量可以在struts.xml或struts.properties中配置，建议在struts.xml中配置，两种配置方式如下：

**在struts.xml文件中配置常量**

<struts>

<constant name="struts.action.extension" value="do"/>

</struts>

**在struts.properties中配置常量**

struts.action.extension=do

(*不建议在这个文件中配置常量*)

因为常量可以在下面多个配置文件中进行定义，所以我们需要了解struts2加载常量的搜索顺序:

struts-default.xml

struts-plugin.xml

struts.xml

struts.properties

web.xml

如果在多个文件中配置了同一个常量，则后一个文件中配置的常量值会覆盖前面文件中配置的常量值.

# 11 常用的常量介绍

<!-- 指定默认编码集,作用于HttpServletRequest的setCharacterEncoding方法 和freemarker 、velocity的输出 -->

<constant name="struts.i18n.encoding" value="UTF-8"/>

<!-- 该属性指定需要Struts 2处理的请求后缀，该属性的默认值是action，即所有匹配\*.action的请求都由Struts2处理。

如果用户需要指定多个请求后缀，则多个后缀之间以英文逗号（,）隔开。 -->

<constant name="struts.action.extension" value="do"/>

<!-- 设置浏览器是否缓存静态内容,默认值为true(生产环境下使用),开发阶段最好关闭 -->

<constant name="struts.serve.static.browserCache" value="false"/>

<!-- 当struts的配置文件修改后,系统是否自动重新加载该文件,默认值为false(生产环境下使用),开发阶段最好打开 -->

<constant name="struts.configuration.xml.reload" value="true"/>

<!-- 开发模式下使用,这样可以打印出更详细的错误信息 -->

<constant name="struts.devMode" value="true" />

<!-- 默认的视图主题 -->

<constant name="struts.ui.theme" value="simple" />

<!– 与spring集成时，指定由spring负责action对象的创建 -->

<constant name="struts.objectFactory" value="spring" />

<!–该属性设置Struts 2是否支持动态方法调用，该属性的默认值是true。如果需要关闭动态方法调用，则可设置该属性为false。 -->

<constant name="struts.enable.DynamicMethodInvocation" value="false"/>

<!--上传文件的总大小限制-->

<constant name="struts.multipart.maxSize" value=“10701096"/>

# 12 Struts2的处理流程

StrutsPrepareAndExecuteFilter

Interceptor

Action

Result

Jsp/html

用户请求

Struts2内置的一些拦截器或用户自定义拦截器

用户编写的action类，类似struts1中的Action

类似struts1中的forward

响应

StrutsPrepareAndExecuteFilter是Struts 2框架的核心控制器，它负责拦截由<url-pattern>/\*</url-pattern>指定的所有用户请求，当用户请求到达时，该Filter会过滤用户的请求。默认情况下，如果用户请求的路径不带后缀或者后缀以.action结尾，这时请求将被转入Struts 2框架处理，否则Struts 2框架将略过该请求的处理。当请求转入Struts 2框架处理时会先经过一系列的拦截器，然后再到Action。**与Struts1不同，Struts2对用户的每一次请求都会创建一个Action，所以Struts2中的Action是线程安全的。**

# 13 为应用指定多个struts配置文件

在大部分应用里，随着应用规模的增加，系统中Action的数量也会大量增加，导致struts.xml配置文件变得非常臃肿。为了避免struts.xml文件过于庞大、臃肿，提高struts.xml文件的可读性，我们可以将一个struts.xml配置文件分解成多个配置文件，然后在struts.xml文件中包含其他配置文件。下面的struts.xml通过<include>元素指定多个配置文件：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<include file="struts-user.xml"/>

<include file="struts-order.xml"/>

</struts>

通过这种方式，我们就可以将Struts 2的Action按模块添加在多个配置文件中。

# 14 动态方法调用

如果Action中存在多个方法时，我们可以使用!+方法名调用指定方法。如下：

public class HelloWorldAction{

private String message;

....

public String execute() throws Exception{

this.message = "我的第一个struts2应用";

return "success";

}

public String other() throws Exception{

this.message = "第二个方法";

return "success";

}

}

假设访问上面action的URL路径为： /struts/test/helloworld.action

要访问action的other() 方法，我们可以这样调用：

/struts/test/helloworld!other.action

如果不想使用动态方法调用，我们可以通过常量struts.enable.DynamicMethodInvocation关闭动态方法调用。

<constant name="struts.enable.DynamicMethodInvocation" value="false"/>

# 15 使用通配符定义action

<package name="itcast" namespace="/test" extends="struts-default">

<action name="helloworld\_\*" class="cn.itcast.action.HelloWorldAction" method="{1}">

<result name="success">/WEB-INF/page/hello.jsp</result>

</action>

</package>

public class HelloWorldAction{

private String message;

....

public String execute() throws Exception{

this.message = "我的第一个struts2应用";

return "success";

}

public String other() throws Exception{

this.message = "第二个方法";

return "success";

}

}

匹配了以helloworld\_开头的所有方法。

要访问other()方法，可以通过这样的URL访问：/test/helloworld\_other.action

# 16 接收请求参数

## 16.1 采用基本类型接收请求参数(get/post)

在Action类中定义与请求参数同名的属性，struts2便能自动接收请求参数并赋予给同名属性。

请求路径： http://localhost:8080/test/view.action?id=78

public class ProductAction {

private Integer id;

public void setId(Integer id) {//struts2通过反射技术调用与请求参数同名的属性的setter方法来获取请求参数值

this.id = id;

}

public Integer getId() {return id;}

}

## 16.2采用复合类型接收请求参数

请求路径： http://localhost:8080/test/view.action?product.id=78

public class ProductAction {

private Product product;

public void setProduct(Product product) { this.product = product; }

public Product getProduct() {return product;}

}

Struts2首先通过反射技术调用Product的默认构造器创建product对象,然后再通过反射技术调用product中与请求参数同名的属性的setter方法来获取请求参数值。

# 17 关于struts2.1.6接收中文请求参数乱码问题

struts2.1.6版本中存在一个Bug，即接收到的中文请求参数为乱码(以post方式提交)，原因是struts2.1.6在获取并使用了请求参数后才调用HttpServletRequest的setCharacterEncoding()方法进行编码设置 ，导致应用使用的就是乱码请求参数。这个bug在struts2.1.8中已经被解决，如果你使用的是struts2.1.6，要解决这个问题，你可以这样做：新建一个Filter，把这个Filter放置在Struts2的Filter之前，然后在doFilter()方法里添加以下代码

public void doFilter(...){

HttpServletRequest req = (HttpServletRequest) request;

req.setCharacterEncoding("UTF-8");//应根据你使用的编码替换UTF-8

filterchain.doFilter(request, response);

}

# 18 自定义类型转换器

java.util.Date类型的属性可以接收格式为2009-07-20的请求参数值。但如果我们需要接收格式为20091221的请求参数，我们必须定义类型转换器，否则struts2无法自动完成类型转换。

import java.util.Date;

public class HelloWorldAction {

private Date createtime;

public Date getCreatetime() {

return createtime;

}

public void setCreatetime(Date createtime) {

this.createtime = createtime;

}

}

public class DateConverter extends DefaultTypeConverter {

@Override public Object convertValue(Map context, Object value, Class toType) {

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMdd");

try {

if(toType == Date.class){//当字符串向Date类型转换时

String[] params = (String[]) value;// Request.getParameterValues()

return dateFormat.parse(params[0]);

}else if(toType == String.class){//当Date转换成字符串时

Date date = (Date) value;

return dateFormat.format(date);

}

} catch (ParseException e) {}

return null;

}

}

将上面的类型转换器注册为局部类型转换器：

在Action类所在的包下放置ActionClassName-conversion.properties文件，ActionClassName是Action的类名，后面的-conversion.properties是固定写法，对于本例而言，文件的名称应为HelloWorldAction-conversion.properties 。在properties文件中的内容为：

属性名称=类型转换器的全类名

对于本例而言， HelloWorldAction-conversion.properties文件中的内容为：

createtime= cn.itcast.conversion.DateConverter

# 19 自定义全局类型转换器

将上面的类型转换器注册为全局类型转换器：

在WEB-INF/classes下放置xwork-conversion.properties文件 。在properties文件中的内容为：

待转换的类型=类型转换器的全类名

对于本例而言， xwork-conversion.properties文件中的内容为：

java.util.Date= cn.itcast.conversion.DateConverter

# 20 访问或添加request/session/application属性

public String scope() throws Exception{

ActionContext ctx = ActionContext.getContext();

ctx.getApplication().put("app", "应用范围");//往ServletContext里放入app

ctx.getSession().put("ses", "session范围");//往session里放入ses

ctx.put("req", "request范围");//往request里放入req

return "scope";

}(**重要！**)

JSP:

<body>

${applicationScope.app} <br>

${sessionScope.ses}<br>

${requestScope.req}<br>

</body>

# 21 获取HttpServletRequest / HttpSession / ServletContext / HttpServletResponse对象

方法一，通过ServletActionContext.类直接获取：

public String rsa() throws Exception{

HttpServletRequest request = **ServletActionContext**.getRequest();

ServletContext servletContext = ServletActionContext.getServletContext();

request.getSession();

HttpServletResponse response = ServletActionContext.getResponse();

return "scope";

}

方法二，实现指定接口，由struts框架运行时注入：

public class HelloWorldAction implements ServletRequestAware, ServletResponseAware, ServletContextAware{

private HttpServletRequest request;

private ServletContext servletContext;

private HttpServletResponse response;

public void setServletRequest(HttpServletRequest req) {

this.request=req;

}

public void setServletResponse(HttpServletResponse res) {

this.response=res;

}

public void setServletContext(ServletContext ser) {

this.servletContext=ser;

}

}

# 22 文件上传

**第一步：在WEB-INF/lib下加入commons-fileupload-1.2.1.jar、commons-io-1.3.2.jar。这两个文件可以从http://commons.apache.org/下载。**

**第二步：把form表的enctype设置为：“multipart/form-data“，如下：**

<form enctype="multipart/form-data" action="${pageContext.request.contextPath}/xxx.action" method="post">

<input type="file" name="**uploadImage**">

</form>

**第三步：在Action类中添加以下属性，属性红色部分对应于表单中文件字段的名称：**

public class HelloWorldAction{

private File **uploadImage**;//得到上传的文件

private String **uploadImage**ContentType;//得到文件的类型

private String **uploadImage**FileName;//得到文件的名称

//这里略省了属性的getter/setter方法

public String upload() throws Exception{

String realpath = ServletActionContext.getServletContext().getRealPath("/images");

File file = new File(realpath);

if(!file.exists()) file.mkdirs();//mkdirs是创建多级目录，mkdir是创建单级目录

FileUtils.copyFile(uploadImage, new File(file, uploadImageFileName));

return "success";

}

}

# 23 多文件上传

**第一步：在WEB-INF/lib下加入commons-fileupload-1.2.1.jar、commons-io-1.3.2.jar。这两个文件可以从http://commons.apache.org/下载。**

**第二步：把form表的enctype设置为：“multipart/form-data“，如下：**

<form enctype="multipart/form-data" action="${pageContext.request.contextPath}/xxx.action" method="post">

<input type="file" name="uploadImages">

<input type="file" name="uploadImages">

</form>

**第三步：在Action类中添加以下属性，属性红色部分对应于表单中文件字段的名称：**

public class HelloWorldAction{

private File[] uploadImages;//得到上传的文件

private String[] uploadImagesContentType;//得到文件的类型

private String[] uploadImagesFileName;//得到文件的名称

//这里略省了属性的getter/setter方法

public String upload() throws Exception{

String realpath = ServletActionContext.getServletContext().getRealPath("/images");

File file = new File(realpath);

if(!file.exists()) file.mkdirs();

for(int i=0 ;i<uploadImages.length; i++){ File uploadImage = uploadImages[i];

FileUtils.copyFile(uploadImage, new File(file, uploadImagesFileName[i]));

}

return "success";

}}

# 24 自定义拦截器

要自定义拦截器需要实现com.opensymphony.xwork2.interceptor.Interceptor接口：

public class PermissionInterceptor implements Interceptor {

private static final long serialVersionUID = -5178310397732210602L;

public void destroy() {

}

public void init() {

}

public String intercept(ActionInvocation invocation) throws Exception {

System.out.println("进入拦截器");

if(session里存在用户){

String result = invocation.invoke();

}else{

return “logon”;

}

//System.out.println("返回值:"+ result);

//return result;

}

}

<package name="itcast" namespace="/test" extends="struts-default">

<interceptors>

<interceptor name=“permission" class="cn.itcast.aop.PermissionInterceptor" />

<interceptor-stack name="permissionStack">

<interceptor-ref name="defaultStack" />

<interceptor-ref name=" permission " />

</interceptor-stack>

</interceptors>

<action name="helloworld\_\*" class="cn.itcast.action.HelloWorldAction" method="{1}">

<result name="success">/WEB-INF/page/hello.jsp</result>

<interceptor-ref name="permissionStack"/>

</action>

</package>

因为struts2中如文件上传，数据验证，封装请求参数到action等功能都是由系统默认的defaultStack中的拦截器实现的，所以我们定义的拦截器需要引用系统默认的defaultStack，这样应用才可以使用struts2框架提供的众多功能。

如果希望包下的所有action都使用自定义的拦截器，可以通过<default-interceptor-ref name=“permissionStack”/>把拦截器定义为默认拦截器。注意：每个包只能指定一个默认拦截器。另外，一旦我们为该包中的某个action显式指定了某个拦截器，则默认拦截器不会起作用。

# 25 输入校验

在struts2中，我们可以实现对action的所有方法进行校验或者对action的指定方法进行校验。

对于输入校验struts2提供了两种实现方法:

1. 基于手工编写代码实现。

2. 基于XML配置方式实现。

## 25.1 手工编写代码实现对action中所有方法输入校验

通过在action中提供validate() 方法实现输入校验， validate()方法会校验action中所有与execute方法签名相同的方法。当某个数据校验失败时，我们应该调用addFieldError()方法往系统的fieldErrors添加校验失败信息（为了使用addFieldError()方法，action可以继承ActionSupport ），如果系统的fieldErrors包含失败信息，struts2会将请求转发到名为input的result。在input视图中可以通过<s:fielderror/>显示失败信息。

**validate()使用例子：**

public void **validate**() {

if(this.mobile==null || "".equals(this.mobile.trim())){ this.addFieldError(“mobile", "手机号不能为空");

}else{ if(!Pattern.compile("^1[358]\\d{9}").matcher(this.mobile.trim()).matches()){

this.addFieldError(“mobile", "手机号的格式不正确"); }

}

}

**验证失败后，请求转发至input视图：**

<result name="input">/WEB-INF/page/addUser.jsp</result>

**在addUser.jsp页面中使用<s:fielderror/>显示失败信息。**

## 25.2 手工编写代码实现对action指定方法输入校验

通过validateXxx()方法实现， validateXxx()只会校验action中方法名为Xxx的方法。其中Xxx的第一个字母要大写。当某个数据校验失败时，我们应该调用addFieldError()方法往系统的fieldErrors添加校验失败信息（为了使用addFieldError()方法，action可以继承ActionSupport ），如果系统的fieldErrors包含失败信息，struts2会将请求转发到名为input的result。在input视图中可以通过<s:fielderror/>显示失败信息。

**validateXxx()方法使用例子：**

public String **add**() throws Exception{ return "success";}

public void **validateAdd**(){

if(username==null && "".equals(username.trim())) **this.addFieldError("username", "用户名不能为空");**

}

**验证失败后，请求转发至input视图：**

<result name="input">/WEB-INF/page/addUser.jsp</result>

**在addUser.jsp页面中使用<s:fielderror/>显示失败信息。**

## 25.3 输入校验的流程

1.类型转换器对请求参数执行类型转换，并把转换后的值赋给action中的属性。

2.如果在执行类型转换的过程中出现异常，系统会将异常信息保存到ActionContext，conversionError拦截器将异常信息添加到fieldErrors里。不管类型转换是否出现异常，都会进入第3步。

3.系统通过反射技术先调用action中的validateXxx()方法，Xxx为方法名。

4.再调用action中的validate()方法。

5.经过上面4步，如果系统中的fieldErrors存在错误信息（即存放错误信息的集合的size大于0)，系统自动将请求转发至名称为input的视图。如果系统中的fieldErrors没有任何错误信息，系统将执行action中的处理方法。

## 25.4 基于XML配置方式实现对action的所有方法进行输入校验

使用基于XML配置方式实现输入校验时，Action也需要继承ActionSupport，并且提供校验文件，校验文件和action类放在同一个包下，文件的取名格式为：ActionClassName-validation.xml，其中ActionClassName为action的简单类名，-validation为固定写法。如果Action类为cn.itcast.UserAction，那么该文件的取名应为：UserAction-validation.xml。下面是校验文件的模版：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.3//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.3.dtd">

<validators>

<field name="username">

<field-validator type="requiredstring">

<param name="trim">true</param>

<message>用户名不能为空!</message>

</field-validator>

</field>

</validators>

<field>指定action中要校验的属性，<field-validator>指定校验器，上面指定的校验器requiredstring是由系统提供的，系统提供了能满足大部分验证需求的校验器，这些校验器的定义可以在xwork-core-2.x.jar中com.opensymphony.xwork2.validator.validators下的default.xml中找到。

<message>为校验失败后的提示信息,如果需要国际化，可以为message指定key属性，key的值为资源文件中的key。

在这个校验文件中，对action中字符串类型的username属性进行验证，首先要求调用trim()方法去掉空格，然后判断用户名是否为空。

## 25.5 编写校验文件时，不能出现帮助信息

在编写ActionClassName-validation.xml校验文件时,如果出现不了帮助信息,可以按下面方式解决：

windwos->preferences->myeclipse->files and editors->xml->xmlcatalog

点“add”,在出现的窗口中的location中选“File system”,然后在xwork-2.1.2解压目录的src\java目录中选择xwork-validator-1.0.3.dtd,回到设置窗口的时候不要急着关闭窗口,应把窗口中的Key Type改为URI 。Key改为http://www.opensymphony.com/xwork/xwork-validator-1.0.3.dtd

## 25.6 struts2提供的校验器列表

系统提供的校验器如下：

required (必填校验器,要求被校验的属性值不能为null)

requiredstring (必填字符串校验器,要求被校验的属性值不能为null，并且长度大于0,默认情况下会对字符串去前后空格)

stringlength(字符串长度校验器，要求被校验的属性值必须在指定的范围内，否则校验失败,minLength参数指定最小长度，maxLength参数指定最大长度，trim参数指定校验field之前是否去除字符串前后的空格)

regex(正则表达式校验器，检查被校验的属性值是否匹配一个正则表达式，expression参数指定正则表达式，caseSensitive参数指定进行正则表达式匹配时，是否区分大小写,默认值为true)

int(整数校验器，要求field的整数值必须在指定范围内，min指定最小值，max指定最大值)

double(双精度浮点数校验器,要求field的双精度浮点数必须在指定范围内,min指定最小值,max指定最大值)

fieldexpression(字段OGNL表达式校验器,要求field满足一个ognl表达式，expression参数指定ognl表达式,该逻辑表达式基于ValueStack进行求值,返回true时校验通过，否则不通过)

email(邮件地址校验器，要求如果被校验的属性值非空，则必须是合法的邮件地址)

url(网址校验器,要求如果被校验的属性值非空,则必须是合法的url地址)

date(日期校验器,要求field的日期值必须在指定范围内,min指定最小值,max指定最大值)

conversion(转换校验器，指定在类型转换失败时，提示的错误信息)

visitor(用于校验action中复合类型的属性，它指定一个校验文件用于校验复合类型属性中的属性)

expression(OGNL表达式校验器，它是一个非字段校验器， expression参数指定ognl表达式,该逻辑表达式基于ValueStack进行求值,返回true时校验通过，否则不通过，该校验器不可用在字段校验器风格的配置中)

## 25.7 校验器的使用例子

required 必填校验器

<field-validator type="required">

<message>性别不能为空!</message>

</field-validator>

requiredstring 必填字符串校验器

<field-validator type="requiredstring">

<param name="trim">true</param>

<message>用户名不能为空!</message>

</field-validator>

stringlength：字符串长度校验器

<field-validator type="stringlength">

<param name="maxLength">10</param>

<param name="minLength">2</param>

<param name="trim">true</param>

<message>产品名称应在2-10个字符之间</message>

</field-validator>

email：邮件地址校验器

<field-validator type="email">

<message>电子邮件地址无效</message>

</field-validator>

regex：正则表达式校验器

<field-validator type="regex">

<param name="expression"><![CDATA[^1[358]\d{9}$]]></param>

<message>手机号格式不正确!</message>

</field-validator>

int：整数校验器

<field-validator type="int">

<param name="min">1</param>

<param name="max">150</param>

<message>年龄必须在1-150之间</message>

</field-validator>

字段OGNL表达式校验器

<field name="imagefile">

<field-validator type="fieldexpression">

<param name="expression"><![CDATA[imagefile.length() <= 0]]></param>

<message>文件不能为空</message>

</field-validator>

</field>

## 25.8 基于XML配置方式对指定action方法实现输入校验

当校验文件的取名为ActionClassName-validation.xml时，会对 action中的所有处理方法实施输入验证。如果你只需要对action中的某个action方法实施校验，那么，校验文件的取名应为:ActionClassName-ActionName-validation.xml，其中ActionName为访问该action的名称。例如：在实际应用中，常有以下配置：

<action name="user\_\*" class="cn.itcast.action.UserAction" method="{1}“ >

<result name="success">/WEB-INF/page/message.jsp</result>

<result name="input">/WEB-INF/page/addUser.jsp</result>

</action>

UserAction中有以下两个处理方法：

public String add() throws Exception{

....

}

public String update() throws Exception{

....

}

要对add()方法实施验证，校验文件的取名为： UserAction-user\_add-validation.xml

要对update()方法实施验证，校验文件的取名为： UserAction-user\_update-validation.xml

## 25.9 基于XML校验的一些特点

当应用为某个action提供了两种规则的校验文件ActionClassName-validation.xml和ActionClassName-ActionName-validation.xml，系统按下面顺序寻找校验文件：

1。AconClassName-validation.xml

2。ActionClassName-ActionName-validation.xml

系统寻找到第一个校验文件时还会继续搜索后面的校验文件，当搜索到所有校验文件时，会把校验文件里的所有校验规则汇总，然后全部应用于action方法的校验。如果两个校验文件中指定的校验规则冲突，则只使用后面文件中的校验规则。

当action继承了另一个action，父类action的校验文件会先被搜索到。

假设UserAction继承BaseAction：

<action name="user" class="cn.itcast.action.UserAction" method="{1}">

</action>

访问上面action，系统先搜索父类的校验文件：BaseAction-validation.xml， BaseAction-user-validation.xml，接着搜索子类的校验文件： UserAction-validation.xml， UserAction-user-validation.xml。应用于上面action的校验规则为这四个文件的总和。

# 26 国际化

准备资源文件，资源文件的命名格式如下：

baseName\_language\_country.properties

baseName\_language.properties

baseName.properties

其中baseName是资源文件的基本名，我们可以自定义，但language和country必须是java支持的语言和国家。如：

中国大陆： baseName\_zh\_CN.properties

美国： baseName\_en\_US.properties

现在为应用添加两个资源文件：

第一个存放中文：itcast\_zh\_CN.properties

内容为：welcome=欢迎来到传智播客

第二个存放英语（美国）： itcast\_en\_US.properties

内容为： welcome=welcome to itcast

对于中文的属性文件，我们编写好后，应该使用jdk提供的native2ascii命令把文件转换为unicode编码的文件。命令的使用方式如下：

native2ascii 源文件.properties 目标文件.properties

## 26.1 配置全局资源与输出国际化信息

当准备好资源文件之后，我们可以在struts.xml中通过struts.custom.i18n.resources常量把资源文件定义为全局资源文件，如下：

<constant name="struts.custom.i18n.resources" value="itcast" />

itcast为资源文件的基本名。

**后面我们就可以在页面或在action中访问国际化信息：**

在JSP页面中使用<s:text name=“”/>标签输出国际化信息：等价于<lable></lable>

<s:text name=“user”/>，name为资源文件中的key

在Action类中，可以继承ActionSupport，使用getText()方法得到国际化信息，该方法的第一个参数用于指定资源文件中的key。

在表单标签中，通过key属性指定资源文件中的key，如：

<s:textfield name="realname" key="user"/>

## 26.2 国际化—输出带占位符的国际化信息

资源文件中的内容如下：

welcome= {0}，欢迎来到传智播客{1}

在jsp页面中输出带占位符的国际化信息

<s:text name="welcome">

<s:param>liming<s:param>

<s:param>study</s:param>

</s:text>

在Action类中获取带占位符的国际化信息，可以使用getText(String key, String[] args)或getText(String aTextName, List args)方法。

## 26.3 国际化—包范围资源文件

在一个大型应用中，整个应用有大量的内容需要实现国际化，如果我们把国际化的内容都放置在全局资源属性文件中，显然会导致资源文件变的过于庞大、臃肿，不便于维护，这个时候我们可以针对不同模块，使用包范围来组织国际化文件。

**方法如下：**

在java的包下放置package\_language\_country.properties资源文件，package为固定写法，处于该包及子包下的action都可以访问该资源。当查找指定key的消息时，系统会先从package资源文件查找，当找不到对应的key时，才会从常量struts.custom.i18n.resources指定的资源文件中寻找。

## 26.4 国际化—Action范围资源文件

当一些显示数据只会被某个action使用，这时候我们可以把这些显示数据定义在只能被该action访问的Action范围资源文件中。

为某个action指定资源文件，方法如下：

在Action类所在的路径，放置ActionClassName\_language\_country.properties资源文件，ActionClassName为action类的简单名称。

当查找指定key的消息时，系统会先从ActionClassName\_language\_country.properties资源文件查找，如果没有找到对应的key，然后沿着当前包往上查找基本名为package 的资源文件，一直找到最顶层包。如果还没有找到对应的key，最后会从常量struts.custom.i18n.resources指定的资源文件中寻找。

## 26.5 国际化—JSP中直接访问某个资源文件

struts2为我们提供了<s:i18n>标签，使用<s:i18n>标签我们可以直接从类路径下的某个资源文件中获取国际化数据，而无需任何配置：

<s:i18n name="itcast">

<s:text name=“welcome”/>

</s:i18n>

Itcast为类路径下资源文件的基本名。

如果要访问的资源文件在类路径的某个包下，可以这样访问：

<s:i18n name=“cn/itcast/action/package">

<s:text name="welcome">

<s:param>小张</s:param>

</s:text>

</s:i18n>

上面访问cn.itcast.action包下基本名为package的资源文件。

# 27 OGNL表达式语言

OGNL是Object Graphic Navigation Language（对象图导航语言）的缩写，它是一个开源项目。 Struts 2框架使用OGNL作为默认的表达式语言。

相对EL表达式，它提供了平时我们需要的一些功能，如：

支持对象方法调用，如xxx.sayHello()；

支持类静态方法调用和常量值访问，表达式的格式为@[类全名（包括包路径）]@[方法名 | 值名]，例如：@java.lang.String@format('foo %s', 'bar')或@cn.itcast.Constant@APP\_NAME；

操作集合对象。

Ognl 有一个上下文（Context）概念，说白了上下文就是一个MAP结构，它实现了java.utils.Map接口，在Struts2中上下文（Context）的实现为ActionContext，下面是上下文（Context）的结构示意图

Struts 2中的OGNL Context实现者为ActionContext，它结构示意图如下:



当Struts2接受一个请求时，会迅速创建ActionContext，ValueStack，action 。然后把action存放进ValueStack，所以action的实例变量可以被OGNL访问.值栈保存的是action对象

访问上下文（Context）中的对象需要使用#符号标注命名空间，如#application、#session

另外OGNL会设定一个根对象（root对象），在Struts2中根对象就是ValueStack（值栈） 。如果要访问根对象（即ValueStack）中对象的属性，则可以省略#命名空间，直接访问该对象的属性即可。

在struts2中，根对象ValueStack的实现类为OgnlValueStack，该对象不是我们想像的只存放单个值，而是存放一组对象。在OgnlValueStack类里有一个List类型的root变量，就是使用他存放一组对象

|--request

|--application

context ------|--OgnlValueStack root变量[action, OgnlUtil, ... ]

|--session

|--attr

|--parameters

在root变量中处于第一位的对象叫栈顶对象。通常我们在OGNL表达式里直接写上属性的名称即可访问root变量里存放的对象的属性，搜索顺序是从栈顶对象开始寻找，如果栈顶对象不存在该属性，就会从第二个对象寻找，如果没有找到就从第三个对象寻找，依次往下访问，直到找到为止。

大家注意： Struts2中，OGNL表达式需要配合Struts标签才可以使用。如：<s:property value="name"/>

由于ValueStack(值栈)是Struts 2中OGNL的根对象，如果用户需要访问值栈中的对象，在JSP页面可以直接通过下面的EL表达式访问ValueStack(值栈)中对象的属性：

${foo} //获得值栈中某个对象的foo属性

如果访问其他Context中的对象，由于他们不是根对象，所以在访问时，需要添加#前缀。

**application对象**：用于访问ServletContext，例如#application.userName或者#application['userName']，相当于调用ServletContext的getAttribute("username")。

**session对象**：用来访问HttpSession，例如#session.userName或者#session['userName']，相当于调用session.getAttribute("userName")。

**request对象**：用来访问HttpServletRequest属性（attribute）的Map，例如#request.userName或者#request['userName']，相当于调用request.getAttribute("userName")。

**parameters对象**：用于访问HTTP的请求参数，例如#parameters.userName或者#parameters['userName']，相当于调用request.getParameter("username")。

**attr对象**：用于按page->request->session->application顺序访问其属性。

## 27.1 为何使用EL表达式能够访问valueStack中对象的属性

原因是Struts2对HttpServletRequest作了进一步的封装。简略代码如下：

public class StrutsRequestWrapper extends HttpServletRequestWrapper {

public StrutsRequestWrapper(HttpServletRequest req) {

super(req);

}

public Object getAttribute(String s) {

......

ActionContext ctx = ActionContext.getContext();

Object attribute = super.getAttribute(s);//先从request范围获取属性值

if (ctx != null) {

if (attribute == null) {//如果从request范围没有找到属性值,即从ValueStack中查找对象的属性值

......

ValueStack stack = ctx.getValueStack();

attribute = stack.findValue(s);

......

}

}

return attribute;

}

}

## 27.2 采用OGNL表达式创建List/Map集合对象

如果需要一个集合元素的时候（例如List对象或者Map对象），可以使用OGNL中同集合相关的表达式。

使用如下代码直接生成一个List对象：

<s:set name="list" value="{'zhangming','xiaoi','liming'}" />

<s:iterator value="#list" >

<s:property/><br>

</s:iterator>

Set标签用于将某个值放入指定范围。

scope：指定变量被放置的范围，该属性可以接受application、session、request、 page。如果没有设置该属性，则默认放置在OGNL Context中。

value：赋给变量的值.如果没有设置该属性,则将ValueStack栈顶的值赋给变量。

生成一个Map对象：

<s:set name="foobar" value="#{'foo1':'bar1', 'foo2':'bar2'}" />

<s:iterator value="#foobar" >

<s:property value="key"/>=<s:property value="value"/><br>

</s:iterator>

## 27.3 采用OGNL表达式判断对象是否存在于集合中

对于集合类型，OGNL表达式可以使用in和not in两个元素符号。其中，in表达式用来判断某个元素是否在指定的集合对象中；not in判断某个元素是否不在指定的集合对象中，如下所示。

**in表达式：**

<s:if test="'foo' in {'foo','bar'}">

在

</s:if>

<s:else>

不在

</s:else>

**not in表达式：**

<s:if test="'foo' not in {'foo','bar'}">

不在

</s:if>

<s:else>

在

</s:else>

## 27.4 OGNL表达式的投影功能

除了in和not in之外，OGNL还允许使用某个规则获得集合对象的子集，常用的有以下3个相关操作符。

?：获得所有符合逻辑的元素。

^：获得符合逻辑的第一个元素。

$：获得符合逻辑的最后一个元素。

例如代码：

<s:iterator value="books.{?#this.price > 35}">

<s:property value="title" /> - $<s:property value="price" /><br>

</s:iterator>

在上面代码中，直接在集合后紧跟.{}运算符表明用于取出该集合的子集，{}内的表达式用于获取符合条件的元素，this指的是为了从大集合books筛选数据到小集合，需要对大集合books进行迭代，this代表当前迭代的元素。本例的表达式用于获取集合中价格大于35的书集合。

public class BookAction extends ActionSupport {

private List<Book> books;

....

@Override

public String execute() {

books = new LinkedList<Book>();

books.add(new Book("A735619678", "spring", 67));

books.add(new Book("B435555322", "ejb3.0",15));

}

}

# 28 Stuts2 常用标签

## 28.1 property标签

property标签用于输出指定值：

<s:set name="name" value="'kk'" />

<s:property value="#name"/>

default：可选属性，如果需要输出的属性值为null，则显示该属性指定的值

escape：可选属性，指定是否格式化HTML代码。

value：可选属性，指定需要输出的属性值，如果没有指定该属性，则默认输出ValueStack栈顶的值。

id：可选属性，指定该元素的标识

## 28.2 iterator标签

iterator标签用于对集合进行迭代，这里的集合包含List、Set和数组。

<s:set name="list" value="{'zhangming','xiaoi','liming'}" />

<s:iterator value="#list" status="st">

<font color=<s:if test="#st.odd">red</s:if><s:else>blue</s:else>>

<s:property /></font><br>

</s:iterator>

value：可选属性，指定被迭代的集合，如果没有设置该属性，则使用ValueStack栈顶的集合。

id：可选属性，指定集合里元素的id（已被标注为过时）。

status：可选属性，该属性指定迭代时的IteratorStatus实例。该实例包含如下几个方法：

int getCount()，返回当前迭代了几个元素。

int getIndex()，返回当前迭代元素的索引。

boolean isEven()，返回当前被迭代元素的索引是否是偶数

boolean isOdd()，返回当前被迭代元素的索引是否是奇数

boolean isFirst()，返回当前被迭代元素是否是第一个元素。

boolean isLast()，返回当前被迭代元素是否是最后一个元素。

## 28.3 if/elseif/else标签

<s:set name="age" value="21" />

<s:if test="#age==23">

23

</s:if>

<s:elseif test="#age==21">

21

</s:elseif>

<s:else>

都不等

</s:else>

## 28.4 url标签

<s:url action="helloworld\_add" namespace="/test"><s:param name="personid" value="23"/></s:url>

生成类似如下路径：

/struts/test/helloworld\_add.action?personid=23

红色部分为内容路径。

当标签的属性值作为字符串类型处理时， “%”符号的用途是计算OGNL表达式的值。

<s:set name="myurl" value="'http://www.foshanshop.net'"/>

<s:url value="#myurl" /><br>

<s:url value="%{#myurl}" />

输出结果：

#myurl

http://www.foshanshop.net

28.5 表单标签\_checkboxlist复选框

**如果集合为list**

**<s:checkboxlist name="list" list="{'Java','.Net','RoR','PHP'}" value="{'Java','.Net'}"/>**

生成如下html代码：

<input type="checkbox" name="list" value="Java" checked="checked"/><label>Java</label>

<input type="checkbox" name="list" value=".Net" checked="checked"/><label>.Net</label>

<input type="checkbox" name="list" value="RoR"/><label>RoR</label>

<input type="checkbox" name="list" value="PHP"/><label>PHP</label>

**如果集合为MAP**

**<s:checkboxlist name="map" list="#{1:'瑜珈用品',2:'户外用品',3:'球类',4:'自行车'}" listKey="key" listValue="value" value="{1,2,3}"/>**

生成如下html代码：

<input type="checkbox" name="map" value="1" checked="checked"/><label>瑜珈用品</label>

<input type="checkbox" name="map" value="2" checked="checked"/><label>户外用品</label>

<input type="checkbox" name="map" value="3" checked="checked"/><label>球类</label>

<input type="checkbox" name="map" value="4"/><label>自行车</label>

**如果集合里存放的是javabean**

<%

Person person1 = new Person(1,"第一个");

Person person2 = new Person(2,"第二个");

List<Person> list = new ArrayList<Person>();

list.add(person1);

list.add(person2);

request.setAttribute("persons",list);

%>

<s:checkboxlist name="beans" list="#request.persons" listKey="personid" listValue="name"/>

**Personid和name为Person的属性**

生成如下html代码：

<input type="checkbox" name=“beans" value="1"/><label>第一个</label>

<input type="checkbox" name=“beans" value="2"/><label>第二个</label>

## 28.5表单标签\_radio单选框

该标签的使用和checkboxlist复选框相同。

如果集合里存放的是javabean(personid和name为Person的属性)

< s:radio name="beans" list="#request.persons" listKey="personid" listValue="name"/>

生成如下html代码：

<input type="radio" name="beans" id="beans1" value="1"/><label>第一个</label>

<input type="radio" name="beans" id="beans2" value="2"/><label>第二个</label>

如果集合为MAP

<s:radio name="map" list="#{1:'瑜珈用品',2:'户外用品',3:'球类',4:'自行车'}" listKey="key" listValue="value“ value="1"/>

生成如下html代码：

<input type="radio" name="map" id="map1" value="1"/><label for="map1">瑜珈用品</label>

<input type="radio" name="map" id="map2" value="2"/><label for="map2">户外用品</label>

<input type="radio" name="map" id="map3" value="3"/><label for="map3">球类</label>

<input type="radio" name="map" id="map4" value="4"/><label for="map4">自行车</label>

如果集合为list

<s:radio name="list" list="{'Java','.Net'}" value="'Java'"/>

生成如下html代码：

<input type="radio" name="list" checked="checked" value="Java"/><label>Java</label>

<input type="radio" name="list" value=".Net"/><label>.Net</label>

## 28.6 表单标签\_select下拉选择框

<s:select name="list" list="{'Java','.Net'}" value="'Java'"/>

<select name="list" id="list">

<option value="Java" selected="selected">Java</option>

<option value=".Net">.Net</option>

</select>

<s:select name="beans" list="#request.persons" listKey="personid" listValue="name"/>

<select name="beans" id="beans">

<option value="1">第一个</option>

<option value="2">第二个</option>

</select>

<s:select name="map" list="#{1:'瑜珈用品',2:'户外用品',3:'球类',4:'自行车'}" listKey="key" listValue="value" value="1"/>

<select name="map" id="map">

<option value="1" selected="selected">瑜珈用品</option>

<option value="2">户外用品</option>

<option value="3">球类</option>

<option value="4">自行车</option>

</select>

## 28.7 <s:token />标签防止重复提交

<s:token />标签防止重复提交，用法如下：

第一步：在表单中加入<s:token />

<s:form action="helloworld\_other" method="post" namespace="/test">

<s:textfield name="person.name"/><s:token/><s:submit/>

</s:form>

第二步：

<action name="helloworld\_\*" class="cn.itcast.action.HelloWorldAction" method="{1}">

<interceptor-ref name="defaultStack" />

<interceptor-ref name="token" />

<result name="invalid.token">/WEB-INF/page/message.jsp</result>

<result>/WEB-INF/page/result.jsp</result>

</action>

以上配置加入了“token”拦截器和“invalid.token”结果，因为“token”拦截器在会话的token与请求的token不一致时，将会直接返回“invalid.token”结果。

在debug状态,控制台出现下面信息,是因为Action中并没有struts.token和struts.token.name属性,我们不用关心这个错误：

严重: ParametersInterceptor - [setParameters]: Unexpected Exception caught setting 'struts.token' on 'class xxx: Error setting expression 'struts.token' with value '[Ljava.lang.String;@39f16f'

严重: ParametersInterceptor - [setParameters]: Unexpected Exception caught setting 'struts.token.name'

# 29 Struts2+Spring2.5+Hibernate3.3整合开发

下面给出整合开发时Struts 2、 Hibernate、Spring需要的JAR。

struts2-core-2.x.x.jar :Struts 2框架的核心类库

Xwork-core-2.x.x.jar :XWork类库，Struts 2在其上构建

ognl-2.6.x.jar :对象图导航语言（Object Graph Navigation Language），struts2框架通过其读写对象的属性

freemarker-2.3.x.jar :Struts 2的UI标签的模板使用FreeMarker编写

commons-fileupload-1.2.x.jar 文件上传组件，2.1.6版本后需要加入此文件

struts2-spring-plugin-2.x.x.jar ：用于struts2集成Spring的插件

**hibernate核心安装包下的**(下载路径：http://www.hibernate.org/，点击“Hibernate Core”右边的“Downloads”):

hibernate3.jar

lib\bytecode\cglib\hibernate-cglib-repack-2.1\_3.jar

lib\required\\*.jar

**hibernate 注解安装包下的**(下载路径：www.hibernate.org，点击“Hibernate Annota

tions”右边的“Downloads”):

hibernate-annotations.jar

lib\ejb3-persistence.jar、hibernate-commons-annotations.jar

**Hibernate针对JPA的实现包**(下载路径：www.hibernate.org，点击“Hibernate Entitymanager”右边的“Downloads”):

hibernate-entitymanager.jar

lib\test\log4j.jar、slf4j-log4j12.jar

**Spring安装包下的**

dist\spring.jar

lib\c3p0\c3p0-0.9.1.2.jar lib\aspectj\aspectjweaver.jar、aspectjrt.jar lib\cglib\cglib-nodep-2.1\_3.jar

lib\j2ee\common-annotations.jar lib\log4j\log4j-1.2.15.jar lib\jakarta-commons\commons-logging.jar

MYSQL数据库驱动jar

Spring的配置模版：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

</beans>

## 29.1 第一步在Spring中配置数据源

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">

<property name="driverClass" value="org.gjt.mm.mysql.Driver"/>

<property name="jdbcUrl" value="jdbc:mysql://localhost:3306/itcast?useUnicode=true&amp;characterEncoding=UTF-8"/>

<property name="user" value="root"/>

<property name="password" value="123456"/>

<!--初始化时获取的连接数，取值应在minPoolSize与maxPoolSize之间。Default: 3 -->

<property name="initialPoolSize" value="1"/>

<!--连接池中保留的最小连接数。-->

<property name="minPoolSize" value="1"/>

<!--连接池中保留的最大连接数。Default: 15 -->

<property name="maxPoolSize" value="300"/>

<!--最大空闲时间,60秒内未使用则连接被丢弃。若为0则永不丢弃。Default: 0 -->

<property name="maxIdleTime" value="60"/>

<!--当连接池中的连接耗尽的时候c3p0一次同时获取的连接数。Default: 3 -->

<property name="acquireIncrement" value="5"/>

<!--每60秒检查所有连接池中的空闲连接。Default: 0 -->

<property name="idleConnectionTestPeriod" value="60"/>

</bean>

采用utf8字符集编码创建数据库,名为:itcast。DDL语句如下：

CREATE DATABASE `itcast` CHARACTER SET 'utf8' COLLATE 'utf8\_general\_ci';

## 29.2 第二步集成进hibernate

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource"/>

<property name="mappingResources">

<list>

<value>cn/itcast/bean/buyer.hbm.xml</value>

</list>

</property>

<property name="hibernateProperties">

<value>

hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

hibernate.hbm2ddl.auto=update

hibernate.show\_sql=false

hibernate.format\_sql=false

</value>

</property>

</bean>

## 29.3 第三步使用Spring容器管理事务服务

<bean id="txManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory"/>

</bean>

<!--使用基于注解方式配置事务 -->

<tx:annotation-driven transaction-manager="txManager"/>

## 29.4 配置hibernate实体映射文件buyer.hbm.xml

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="cn.itcast.bean">

<class name=“Buyer" table=“buyer">

<id name="username" length="20"/>

<property name="password" length="20" not-null="true"/>

<property name="gender" not-null="true" length="5">

<type name="org.hibernate.type.EnumType">

<param name="enumClass">cn.itcast.bean.Gender</param>

<!-- 12为java.sql.Types.VARCHAR常量值，即保存枚举的字面值到数据库。如果不指定type参数，保存枚举的索引值(从0开始)到数据库-->

<param name="type">12</param>

</type>

</property>

</class>

</hibernate-mapping>

## 29.5 在web容器中使用Listener实例化spring容器和配置struts2

<!-- 指定spring的配置文件，默认从web根目录寻找配置文件，我们可以通过spring提供的classpath:前缀指定从类路径下寻找 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:beans.xml</param-value>

</context-param>

<!-- 对Spring容器进行实例化 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

## 29.6 配置struts2

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

struts2的配置文件模版struts.xml如下。常量struts.objectFactory=spring明确指出将由Spring负责创建Action实例。

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<!-- 默认的视图主题 -->

<constant name="struts.ui.theme" value="simple" />

<constant name="struts.objectFactory" value="spring" />

<package name="person" namespace="/person" extends="struts-default">

<global-results>

<result name="message">/WEB-INF/page/message.jsp</result>

</global-results>

<action name="action\_\*" class="personList" method="{1}">

<result name="list">/WEB-INF/page/persons.jsp</result> </action>

</package>

</struts>

为了能从spring容器中寻找到Action bean，要求action配置的class属性值和spring中bean的名称相同。

使用spring解决struts2.1.6版本中文参数乱码问题，注意2.1.8版本不存在乱码问题。

<filter>

<filter-name>encoding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encoding</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

使用spring解决hibernate因session关闭导致的延迟加载例外问题。

<filter>

<filter-name>OpenSessionInViewFilter</filter-name>

<filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>

<init-param>

<!-- 指定org.springframework.orm.hibernate3.LocalSessionFactoryBean在spring配置文件中的名称,默认值为sessionFactory.如果LocalSessionFactoryBean在spring中的名称不是sessionFactory,该参数一定要指定,否则会出现找不到sessionFactory的例外 -->

<param-name>sessionFactoryBeanName</param-name>

<param-value>sessionFactory</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>OpenSessionInViewFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

## 29.7 struts2的标签

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<%@taglib uri="/struts-tags" prefix="s" %>

<s:form action="action\_edit" method="post" namespace="/person">

<s:hidden name="person.id"/>

姓名:<s:textfield name="person.name"/><br>

<input type="submit" value="发送"/>

</s:form>

<s:iterator value="persons" >

<s:property value="id"/>, <s:property value="name"/>

<a href='<s:url action="action\_editUI" namespace="/person"><s:param name="person.id" value="id"/></s:url>'>修改</a>

</s:iterator>