Struts2教程1：第一个Struts2程序

在本系列教程中我们将学习到Struts2的各种技术。在本教程中使用的工具和程序库的版本如下：

开发工具：MyEclipse6

Web服务器：Tomcat6

Struts版本：Struts2.0.11.1

JDK版本：JDK1.5.0_12

J2EE版本：Java EE5.0

在本系列教程中Web工程的上下文路径都是struts2，如果在Web根目录有一个index.jsp文件，则访问路径如下：

http://localhost:8080/struts2/index.jsp

由于MyEclipse6目前并不支持Struts2，所以我们需要到struts.apache.org去下载Struts2安装包。要想正常使用Struts2，至少需要如下五个包（可能会因为Struts2的版本不同，包名略有差异，但包名的前半部是一样的）。

struts2-core-2.0.11.1.jar

xwork-2.0.4.jar

commons-logging-1.0.4.jar

freemarker-2.3.8.jar

ognl-2.6.11.jar

Struts2虽然在大版本号上是第二个版本，但基本上在配置和使用上已经完全颠覆了Struts1.x的方式（当然，Struts2仍然是基于MVC模式的，也是动作驱动的，可能这是唯一没变的东西）。Struts2实际上是在Webwork基础上构建起来的MVC框架。我们从Struts2的源代码中可以看到，有很多都是直接使用的xwork(Webwork的核心技术)的包。既然从技术上来说Struts2是全新的框架，那么就让我们来学习一下这个新的框架的使用方法。

如果大家使用过Struts1.x，应该对建立基于Struts1.x的Web程序的基本步骤非常清楚。让我们先来回顾一下建立基于Struts1.x的Web程序的基本步骤。

1. 安装Struts。由于Struts的入口点是ActionServlet，所以得在web.xml中配置一下这个Servlet。

2. 编写Action类（一般从org.apache.struts.action.Action类继承）。

3. 编写ActionForm类（一般从org.apache.struts.action.ActionForm类继承），这一步不是必须的，如果要接收客户端提交的数据，需要执行这一步。

4. 在struts-config.xml文件中配置Action和ActionForm。

5. 如果要采集用户录入的数据，一般需要编写若干JSP页面，并通过这些JSP页面中的form将数据提交给Action。

下面我们就按着编写struts1.x程序的这五步和struts2.x程序的编写过程一一对应，看看它们

谁更“酷”。下面我们来编写一个基于Struts2的Web程序。这个程序的功能是让用户录入两个整数，并提交给一个Struts Action，并计算这两个数的代数和，如果代码和为非负数，则跳转到positive.jsp页面，否则跳转到negative.jsp页面。

【第1步】 安装Struts2

这一步对于Struts1.x和Struts2都是必须的，只是安装的方法不同。Struts1的入口点是一个Servlet，而Struts2的入口点是一个过滤器(Filter)。因此，Struts2要按过滤器的方式配置。下面是在web.xml中配置Struts2的代码：

```
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>
        org.apache.struts2.dispatcher.FilterDispatcher
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

【第2步】 编写Action类

这一步和Struts1.x也必须进行。只是Struts1.x中的动作类必须从Action类中继承，而Struts2.x的动作类需要从com.opensymphony.xwork2.ActionSupport类继承。下面是计算两个整数代码和的Action类，代码如下：

```
package action;

import com.opensymphony.xwork2.ActionSupport;

public class FirstAction extends ActionSupport
{
    private int operand1;
    private int operand2;

    public String execute() throws Exception
    {
        if (getSum() >= 0)    // 如果代码数和是非负整数，跳到positive.jsp页面
        {
            return "positive";
        }
        else    // 如果代码数和是负整数，跳到negative.jsp页面
        {
```

```
            return "negative";
        }
    }

    public int getOperand1()
    {
        return operand1;
    }

    public void setOperand1(int operand1)
    {
        System.out.println(operand1);
          this.operand1 = operand1;
    }

    public int getOperand2()
    {
        return operand2;
    }
    public void setOperand2(int operand2)
    {
        System.out.println(operand2);
        this.operand2 = operand2;
    }
    public int getSum()
    {
        return operand1 + operand2;    // 计算两个整数的代码数和
    }
}
```

从上面的代码可以看出，动作类的一个特征就是要覆盖execute方法，只是Struts2的execute方法没有参数了，而Struts1.x的execute方法有四个参数。而且execute方法的返回值也不同的。Struts2只返回一个String，用于表述执行结果（就是一个标志）。上面代码的其他部分将在下面讲解。

【第3步】 编写ActionForm类

在本例中当然需要使用ActionForm了。在Struts1.x中，必须要单独建立一个ActionForm类（或是定义一个动作Form），而在Struts2中ActionForm和Action已经二合一了。从第二步的代码可以看出，后面的部分就是应该写在ActionForm类中的内容。所以在第2步，本例的ActionForm类已经编写完成（就是Action类的后半部分）。

【第4步】 配置Action类

这一步 struts1.x和struts2.x都是必须的，只是在struts1.x中的配置文件一般叫 struts-config.xml（当然也可以是其他的文件名），而且一般放到WEB-INF目录中。而在struts2.x中的配置文件一般为struts.xml，放到WEB-INF"classes目录中。下面是在struts.xml中配置动作类的代码：

```
<?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="struts2" namespace="/mystruts"
          extends="struts-default">
          <action name="sum" class="action.FirstAction">
              <result name="positive">/positive.jsp</result>
              <result name="negative">/negative.jsp</result>
          </action>
      </package>
  </struts>
```

在<struts>标签中可以有多个<package>，第一个<package>可以指定一个Servlet访问路径（不包括动作名），如“/mystruts”。extends属性继承一个默认的配置文件“struts-default”，一般都继承于它，大家可以先不去管它。<action>标签中的name属性表示动作名，class表示动作类名。

<result>标签的name实际上就是execute方法返回的字符串，如果返回的是“positive”，就跳转到positive.jsp页面，如果是“negative”，就跳转到negative.jsp页面。在<struts>中可以有多个<package>，在<package>中可以有多个<action>。我们可以用如下的URL来访问这个动作：

http://localhost:8080/struts2/mystruts/sum.action

注：Struts1.x的动作一般都以.do结尾，而Struts2是以.action结尾。

【第5步】 编写用户录入接口（JSP 页面）

1. 主界面（sum.jsp）

在Web根目录建立一个sum.jsp，代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="GBK" %>
  <%@ taglib prefix="s" uri="/struts-tags"%>

  <html>
      <head>
          <title>输入操作数</title>
      </head>
```

```
    <body>
        求代数和
        <br/>
        <s:form action="mystruts/sum.action" >
            <s:textfield name="operand1" label="  操作数1"/>
            <s:textfield name="operand2"   label="  操作数2" />
            <s:submit value="代数和" />
        </s:form>
    </body>
</html>
```

在sum.jsp中使用了Struts2带的tag。在Struts2中已经将Struts1.x的好几个标签库都统一了，在Struts2中只有一个标签库/struts-tags。这里面包含了所有的Struts2标签。但使用Struts2的标签大家要注意一下。在<s:form>中最好都使用Struts2标签，尽量不要用HTML或普通文本，大家可以将sum.jsp的代码改为如下的形式，看看会出现什么效果：

```
        ... ...
        求代数和
        <br/>
        <s:form action="mystruts/sum.action" >
操作数1：<s:textfield name="operand1" /><br/>
操作数2：<s:textfield name="operand1" /><br/>
            <s:submit value="代数和" />
        </s:form>
        ... ...
```

提示一下，在<s:form>中Struts2使用<table>定位。

2. positive.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>显示代数和</title>
  </head>

  <body>
    代数和为非负整数<h1><s:property value="sum" /></h1>
  </body>
</html>
```

3. negative.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>显示代数和</title>
  </head>

  <body>
    代数和为负整数<h1><s:property value="sum" /></h1>

  </body>
</html>
```

这两个jsp页面的实现代码基本一样，只使用了一个<s:property>标签来显示Action类中的sum属性值。<s:property>标签是从request对象中获得了一个对象中得到的sum属性，如我们可以使用如下的代码来代替<s:property value=”sum”/>：

```
<%
 com.opensymphony.xwork2.util.OgnlValueStack ovs =
(com.opensymphony.xwork2.util.OgnlValueStack)request.getAttribute("struts.valueStack")
;
 out.println(ovs.findString("sum"));
%>
```

启动Tomcat后，在IE中输入如下的URL来测试这个例子：

http://localhost:8080/struts2/sum.jsp

下一篇：Struts2教程2：处理一个form多个submit

Struts2教程2：处理一个form多个submit

在很多Web应用中，为了完成不同的工作，一个HTML form标签中可能有两个或多个submit按钮，如下面的代码所示：

```
<!--[if !supportLineBreakNewLine]-->
<html action="…"  method="post">
… …
<input type="submit" value="保存" />
```

```
<input type="submit" value="打印" />
</html>
```

由于在<form>中的多个提交按钮都向一个action提交，使用Struts2 Action的execute方法就无法判断用户点击了哪一个提交按钮。如果大家使用过Struts1.x就会知道在Struts1.2.9之前的版本需要使用一个LookupDispatchAction动作来处理含有多个submit的form。但使用LookupDispatchAction动作需要访问属性文件，还需要映射，比较麻烦。从Struts1.2.9开始，加入了一个EventDispatchAction动作。这个类可以通过java反射来调用通过request参数指定的动作（实际上只是判断某个请求参数是不存在，如果存在，就调用在action类中和这个参数同名的方法）。使用EventDispatchAction必须将submit的name属性指定不同的值以区分每个submit。而在Struts2中将更容易实现这个功能。

当然，我们也可以模拟EventDispatchAction的方法通过request获得和处理参数信息。但这样比较麻烦。在Struts2中提供了另外一种方法，使得无需要配置可以在同一个action类中执行不同的方法（默认执行的是execute方法）。使用这种方式也需要通过请求参来来指定要执行的动作。请求参数名的格式为

action!method.action

注：由于Struts2只需要参数名，因此，参数值是什么都可以。

下面我就给出一个实例程序来演示如何处理有多个submit的form：

【第1步】实现主页面(more_submit.jsp)

```
<%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>My JSP 'hello.jsp' starting page</title>
  </head>

  <body>
    <s:form action="submit.action" >
        <s:textfield name="msg" label="输入内容"/>
        <s:submit name="save" value="保存" align="left" method="save"/>
        <s:submit name="print" value="打印" align="left" method="print" />
    </s:form>
  </body>
</html>
```

在more_submit.jsp中有两个submit：保存和打印。其中分别通过method属性指定了要调用的方法：save和print。因此，在Action类中必须要有save和print方法。

【第2步】实现Action类（MoreSubmitAction）

```
package action;
```

```
import javax.servlet.http.*;

import com.opensymphony.xwork2.ActionSupport;
import org.apache.struts2.interceptor.*;

public class MoreSubmitAction extends ActionSupport implements ServletRequestAware
{
    private String msg;
    private javax.servlet.http.HttpServletRequest request;
    // 获得HttpServletRequest对象
    public void setServletRequest(HttpServletRequest request)
    {
        this.request = request;
    }
    // 处理save submit按钮的动作
    public String save() throws Exception
    {
        request.setAttribute("result", "成功保存[" + msg + "]");
        return "save";
    }

    // 处理print submit按钮的动作
    public String print() throws Exception
    {
        request.setAttribute("result", "成功打印[" + msg + "]");
        return "print";
    }
    public String getMsg()
    {
        return msg;
    }

    public void setMsg(String msg)
    {
        this.msg = msg;
    }
}
```

上面的代码需要注意如下两点：

save和print方法必须存在，否则会抛出java.lang.NoSuchMethodException异常。

Struts2 Action动作中的方法和Struts1.x Action的execute不同，只使用Struts2 Action动作的execute方法无法访问request对象，因此，Struts2 Action类需要实现一个Struts2自带的拦截器来获得request对象，拦截器如下：

org.apache.struts2.interceptor. ServletRequestAware

【第3步】配置Struts2 Action

struts.xml的代码如下：

```
<?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>
    <package name="demo" extends="struts-default" >
        <action name="submit"   class="action.MoreSubmitAction">
            <result name="save" >
                /result.jsp
            </result>
            <result name="print">
                /result.jsp
            </result>
        </action>
    </package>
</struts>
```

【第4步】编写结果页（result.jsp）

```
<%@ page pageEncoding="GBK"%>
<html>
  <head>
    <title>提交结果</title>
  </head>
  <body>
    <h1>${result}</h1>
  </body>
</html>
```

在result.jsp中将在save和print方法中写到request属性中的执行结果信息取出来，并输出到客户端。

启动Tomcat后，在IE中执行如下的URL来测试程序：

http://localhost:8080/moresubmit/more_submit.jsp

大家也可以直接使用如下的URL来调用save和print方法：

调用save方法：http://localhost:8080/moresubmit/submit!save.action

调用print方法：http://localhost:8080/moresubmit/submit!print.action

源代码：http://www.itpub.net/attachment.php?aid=520773

Struts2教程3：struts.xml常用配置解析

在本文中将详细讲述struts.xml文件的常用配置及注意事项。

1. 使用<include>标签重用配置文件

在Struts2中提供了一个默认的struts.xml文件，但如果package、action、interceptors等配置比较多时，都放到一个struts.xml文件不太容易维护。因此，就需要将struts.xml文件分成多个配置文件，然后在struts.xml文件中使用<include>标签引用这些配置文件。这样做的优点如下：

结构更清晰，更容易维护配置信息。

配置文件可以复用。如果在多个Web程序中都使用类似或相同的配置文件，那么可以使用<include>标签来引用这些配置文件，这样可以减少工作量。

假设有一个配置文件，文件名为newstruts.xml，代码如下：

```
<?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>
    <package name="demo" extends="struts-default" >
        <action name="submit"   class="action.MoreSubmitAction">
            <result name="save" >
                /result.jsp
            </result>
            <result name="print">
                /result.jsp
            </result>
        </action>
    </package>
</struts>
```

则struts.xml引用newstruts.xml文件的代码如下：

```
<?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>
    <include file="newstruts.xml"/>
```

```
    <package name="test" extends="struts-default">
    … …
    </package>
</struts>
```

大家要注意一下，用<include>引用的xml文件也必须是完成的struts2的配置。实际上<include>在引用时是单独解析的xml文件，而不是将被引用的文件插入到struts.xml文件中。

2. action的别名

在默认情况下，Struts2会调用动作类的execute方法。但有些时候，我们需要在一个动作类中处理不同的动作。也就是用户请求不同的动作时，执行动作类中的不同的方法。为了达到这个目的，可以在<action>标签中通过method方法指定要指行的动作类的方法名，并且需要为不同的动作起不同的名子（也称为别名）。如下面代码所示：

```
<?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="demo" extends="struts-default" >
    <action name="test"    class="action.MyAction">
        … …
    </action>
    <action name="my"    class="action. MyAction" method="my">
        … …
    </action>
</package>
</struts>
```

上面代码的两个动作的class属性都指向同一个类，name为这个类起了两个动作别名：test和my。在动作my中，使用了method属性指定要要运行的方法名为my。

在MyAction类中必须要有my方法，代码如下：

```
package action;

import com.opensymphony.xwork2.ActionSupport;

public class MyAction extends ActionSupport
{
    … …
    public String execute() throws Exception
    {
        // 处理test动作的代码
```

```
    }
    public String my() throws Exception
    {
        // 处理my动作的代码
    }
    … …
}
```

除了在struts.xml中配置别名，还可以通过请求参数来描述指定动作（并不需要在struts.xml中配置）。请求参数的格式如下：

http://localhost:8080/contextPath/actionName!method.action

关于通过请求指定动作的详细内容，请参阅笔者写的《Struts2教程2：处理一个form多个submit》。

3. 为action指定参数

在struts2中还可以为action指定一个或多个参数。大家还记着struts1.x是如何设置的action参数不？ 在struts1.x中可以使用<action>标签的parameter属性为其指定一个action参数，如果要指定多个，就只能通过逗号（,）或其他的分隔符将不同的参数隔开。而在struts2中可以通过<param>标签指定任意多个参数。代码如下：

```
<action name="submit"  class="action.MyAction">
<param name="param1">value1</param>
<param name="param2">value2</param>
    <result name="save" >
        /result.jsp
    </result>
    … …
</action>
```

当然，在action中读这些参数也非常简单，只需要象获取请求参数一样在action类中定义相应的setter方法即可（一般不用定义getter方法）。如下面的代码将读取param1和param2参数的值：

```
package action;

import com.opensymphony.xwork2.ActionSupport;

public class MyAction extends ActionSupport
{
    private String param1;
    private String param2;
```

```
    public String execute() throws Exception
    {
        System.out.println(param1 + param2);
    }
    public void setParam1(String param1)
    {
        this.param1 = param1;
    }
    public void setParam2(String param2)
    {
        this.param2 = param2;
    }
    … …
}
```

当struts2在调用execute之前，param1和param2的值就已经是相应参数的值了，因此，在execute方法中可以直接使用param1和param2。

4. 选择result类型

在默认时，<result>标签的type属性值是“dispatcher”（实际上就是转发，forward）。开发人员可以根据自己的需要指定不同的类型，如redirect、stream等。如下面代码所示：

```
<result name="save" type="redirect">
        /result.jsp
</result>
```

这此result-type可以在struts2-core-2.0.11.1.jar包或struts2源代码中的struts-default.xml文件中找到，在这个文件中找到<result-types>标签，所有的result-type都在里面定义了。代码如下：

```
<result-types>
        <result-type name="chain" class="com.opensymphony.xwork2.ActionChainResult"/>
        <result-type name="dispatcher" class="org.apache.struts2.dispatcher.ServletDispatcherResult" default="true"/>
        <result-type name="freemarker" class="org.apache.struts2.views.freemarker.FreemarkerResult"/>
        <result-type name="httpheader" class="org.apache.struts2.dispatcher.HttpHeaderResult"/>
        <result-type name="redirect" class="org.apache.struts2.dispatcher.ServletRedirectResult"/>
        <result-type name="redirectAction"
```

```
class="org.apache.struts2.dispatcher.ServletActionRedirectResult"/>
        <result-type name="stream" class="org.apache.struts2.dispatcher.StreamResult"/>
        <result-type name="velocity" class="org.apache.struts2.dispatcher.VelocityResult"/>
        <result-type name="xslt" class="org.apache.struts2.views.xslt.XSLTResult"/>
        <result-type name="plainText" class="org.apache.struts2.dispatcher.PlainTextResult" />
        <!-- Deprecated name form scheduled for removal in Struts 2.1.0. The camelCase versions
are preferred. See ww-1707 -->
        <result-type                                                    name="redirect-action"
class="org.apache.struts2.dispatcher.ServletActionRedirectResult"/>
        <result-type name="plaintext" class="org.apache.struts2.dispatcher.PlainTextResult" />
</result-types>
```

5. 全局result

有很多时候一个<result>初很多<action>使用，这时可以使用<global-results>标签来定义全局的<result>，代码如下：

```
<struts>
    <package name="demo" extends="struts-default">
        <global-results>
            <result name="print">/result.jsp</result>
        </global-results>
        <action name="submit" class="action.MoreSubmitAction">
        … …
        </action>
        <action name="my" class="action.MoreSubmitAction" method="my">
        … …
        </action>
    </package>
</struts>
```

如果<action>中没有相应的<result>，Struts2就会使用全局的<result>。

Struts2教程4：使用validate方法验证数据

在Struts2中最简单的验证数据的方法是使用validate。我们从ActionSupport类的源代码中可以看到，ActionSupport类实现了一个Validateable接口。这个接口只有一个validate方法。如果Action类实现了这个接口，Struts2在调用execute方法之前首先会调用这个方法，我们可以在validate方法中验证，如果发生错误，可以根据错误的level选择字段级错误，还是动作级错误。并且可使用addFieldError或addActionError加入相应的错误信息，如果存在Action或Field错误，Struts2会返回“input”（这个并不用开发人员写，由Struts2自动返回），如果返回了“input”，

Struts2就不会再调用execute方法了。如果不存在错误信息，Struts2在最后会调用execute方法。

这两个add方法和ActionErrors类中的add方法类似，只是add方法的错误信息需要一个ActionMessage对象，比较麻烦。除了加入错误信息外，还可以使用addActionMessage方法加入成功提交后的信息。当提交成功后，可以显示这些信息。

以上三个add方法都在ValidationAware接口中定义，并且在ActionSupport类中有一个默认的实现。其实，在ActionSupport类中的实现实际上是调用了ValidationAwareSupport中的相应的方法，也就是这三个add方法是在ValidationAwareSupport类中实现的，代码如下：

```
private final ValidationAwareSupport validationAware = new ValidationAwareSupport();

public void addActionError(String anErrorMessage)
{       validationAware.addActionError(anErrorMessage);
}
public void addActionMessage(String aMessage)
{
    validationAware.addActionMessage(aMessage);
}
public void addFieldError(String fieldName, String errorMessage)
{
    validationAware.addFieldError(fieldName, errorMessage);
}
```

下面我们来实现一个简单的验证程序，来体验一个validate方法的使用。

先来在Web根目录建立一个主页面（validate.jsp），代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>验证数据</title>
  </head>

  <body>
    <s:actionerror/>
    <s:actionmessage/>
    <s:form action="validate.action"   theme="simple">
        输入内容：<s:textfield name="msg"/>
         <s:fielderror key="msg.hello" />
         <br/>
        <s:submit/>
    </s:form>
  </body>
```

</html>

在上面的代码中，使用了 Struts2 的 tag：<s:actionerror>、<s:fielderror> 和 <s:actionmessage>，分别用来显示动作错误信息，字段错误信息，和动作信息。如果信息为空，则不显示。

现在我们来实现一个动作类，代码如下：

```
package action;

import javax.servlet.http.*;

import com.opensymphony.xwork2.ActionSupport;
import org.apache.struts2.interceptor.*;

public class ValidateAction extends ActionSupport
{
    private String msg;
    public String execute()
    {
        System.out.println(SUCCESS);
        return SUCCESS;
    }
    public void validate()
    {
        if(!msg.equalsIgnoreCase("hello"))
        {
            System.out.println(INPUT);
            this.addFieldError("msg.hello", "必须输入hello!");
            this.addActionError("处理动作失败!");
        }
        else
        {
            this.addActionMessage("提交成功");
        }
    }
    public String getMsg()
    {
        return msg;
    }
    public void setMsg(String msg)
    {
        this.msg = msg;
    }
}
```

大家从上面的代码可以看出，Field错误需要一个key（一般用来表示是哪一个属性出的错误），而Action错误和Action消息只要提供一个信息字符串就可以了。

最后来配置一下这个Action，代码如下：

```
<package name="demo" extends="struts-default">
    <action name="validate" class="action.ValidateAction">
        <result name="success">/error/validate.jsp</result>
        <result name="input">/error/validate.jsp</result>
    </action>
</package>
```

假设应用程序的上下文路径为demo，则可通过如下的URL来测试程序：

http://localhost:8080/demo/validate.jsp

我们还可以使用ValidationAware接口的其他方法（由ValidationAwareSupport类实现）获得或设置字段错误信息、动作错误信息以及动作消息。如hasActionErrors方法判断是否存在动作层的错误，getFieldErrors获得字段错误信息（一个Map对象）。下面是ValidationAware接口提供的所有的方法：

```
package com.opensymphony.xwork2;

import java.util.Collection;
import java.util.Map;

public interface ValidationAware
{
    void setActionErrors(Collection errorMessages);
    Collection getActionErrors();

    void setActionMessages(Collection messages);
    Collection getActionMessages();
    void setFieldErrors(Map errorMap);
    Map getFieldErrors();
    void addActionError(String anErrorMessage);
    void addActionMessage(String aMessage);
    void addFieldError(String fieldName, String errorMessage);
    boolean hasActionErrors();
    boolean hasActionMessages();
    boolean hasErrors();
    boolean hasFieldErrors();
}
```

Struts2教程5：使用Validation框架验证数据

在《Struts2教程4：使用validate方法验证数据》中曾讲到使用validate方法来验证客户端提交的数据，但如果使用validate方法就会将验证代码和正常的逻辑代码混在一起，但这样做并不利于代码维护，而且也很难将过些代码用于其他程序的验证。在Struts2中为我们提供了一个Validation框架，这个框架和Struts1.x提供的Validation框架类似，也是通过XML 文件进行配置。

一、服务端验证

下面将给出一个例子来演示如何使用Struts2的validation框架来进行服务端验证。我们可以按着如下四步来编写这个程序：

【第1步】建立Action类（NewValidateAction.java）

```
package action;

import com.opensymphony.xwork2.ActionSupport;

public class NewValidateAction extends ActionSupport
{
    private String msg;   // 必须输入
    private int age;     // 在13和20之间
    public String getMsg()
    {
        return msg;
    }
    public void setMsg(String msg)
    {
        this.msg = msg;
    }
    public int getAge()
    {
        return age;
    }
    public void setAge(int age)
    {
        this.age = age;
    }
}
```

下面我们来验证msg和age属性。

【第2步】配置Action类，struts.xml 的代码如下：

```
<?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>
    <package name="demo" extends="struts-default" namespace="/test">
        <action name="new_validate" class="action.NewValidateAction">
            <result name="input">/validate_form.jsp</result>
            <result name="success">/validate_form.jsp</result>
        </action>
    </package>
</struts>
```

【第3步】编写验证规则配置文件

这是一个基于 XML 的配置文件，和 struts1.x中的validator框架的验证规则配置文件类似。但一般放到和要验证的.class文件在同一目录下，而且配置文件名要使用如下两个规则中的一个来命名：

<ActionClassName>-validation.xml

<ActionClassName>-<ActionAliasName>-validation.xml

其中<ActionAliasName> 就是struts.xml 中<ation>的name属性值。在本例中我们使用第一种命名规则，所以文件名是NewValidateAction-validation.xml 。文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
    <field name="msg">
        <field-validator type="requiredstring">
            <message>请输入信息</message>
        </field-validator>
    </field>
    <field name="age">
        <field-validator type="int">
            <param name="min">13</param>
            <param name="max">20</param>
            <message>
                必须在 13至20之间
            </message>
        </field-validator>
    </field>
</validators>
```

这个文件使用了两个规则：requiredstring（必须输入）和int（确定整型范围）。关于其他更详细的验证规则，请读者访问http://struts.apache.org/2.0.11.1/docs/validation.html来查看。

【第4步】编写数据录入JSP页。

在Web根目录中建立一个validate_form.jsp文件，代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<link rel="stylesheet" type="text/css" href="<s:url value="/styles/styles.css"/>">
<html>
  <head>
    <title>验证数据</title>
  </head>
  <body>
    <s:form action="new_validate" namespace="/test" >
        <s:textfield name="msg" label="姓名"  />
        <s:textfield name="age" label="年龄"/>
        <s:submit/>
    </s:form>
  </body>
</html>
```

大家要注意一下，如果在struts.xml的<package>标签中指定namespace属性，需要在<s:form>中也将namespace和action分开写，如上面代码所示。不能将其连在一起，Struts2需要分开的action和namespace。如下面的代码是错误的：

```
<s:form action="/test/new_validate" >
  ... ...
</s:form>
```

在上面的程序中还使用了一个styles.css来定制错误信息的风格。代码如下：

```
.label {font-style:italic; }
.errorLabel {font-style:italic; color:red; }
.errorMessage {font-weight:bold; color:red; }
```

需要在Web根目录中建立一个styles目录，并将styles.css

假设Web工程的上下文路径是validation，可以使用如下的URL来测试这个程序：

http://localhost:8080/validation/validate_form.jsp

显示结果如图1所示。

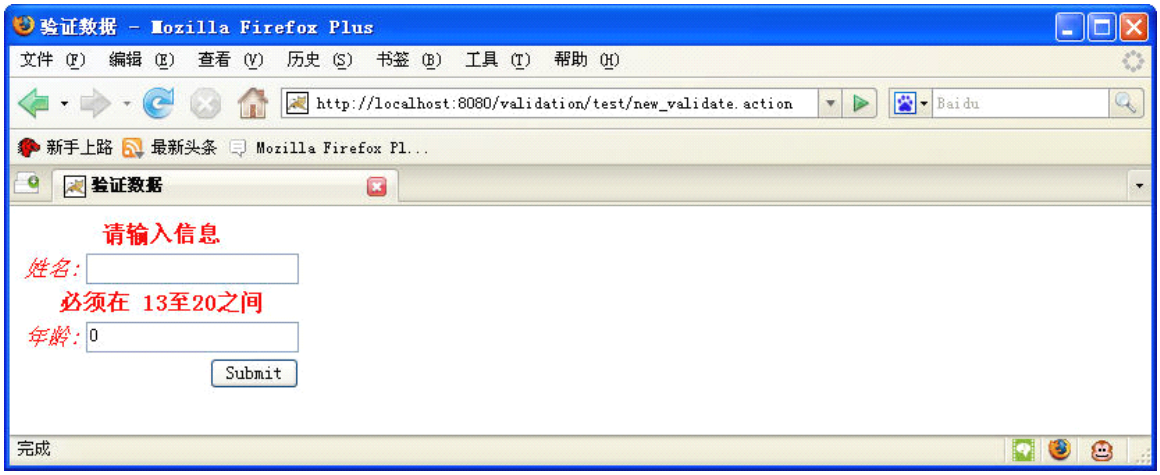

图1

二、客户端验证

在Struts2中实现客户端验证非常简单，只需要在<s:form>中加入一个validate属性，值为true。如<s:form validate="true" ... > ... </form>即可。

三、验证嵌套属性

有一类特殊的属性，即这个属性的类型是另外一个JavaBean，如有一个User类，代码如下：

```
package data;

public class User
{
    private String name;
    private int age;
    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
    public int getAge()
    {
        return age;
    }
    public void setAge(int age)
```

```
    {
        this.age = age;
    }
}
```

在NewValidateAction类中加一个user属性，代码如下：

```
package action;

import com.opensymphony.xwork2.ActionSupport;
import data.User;

public class NewValidateAction extends ActionSupport
{
    private String msg;
    private int age;
    private User user;
    public String getMsg()
    {
        return msg;
    }

    public void setMsg(String msg)
    {
        this.msg = msg;
    }
    public int getAge()
    {
        return age;
    }
    public void setAge(int age)
    {
        this.age = age;
    }
    public User getUser()
    {
        return user;
    }

    public void setUser(User user)
    {
        this.user = user;
    }
}
```

如果要验证NewValidateAction中的user属性，可以使用visitor验证器。操作过程如下：

首先在NewValidateAction-validation.xml中加入一个<field>标签，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
    … …
    <field name="user">
    <field-validator type="visitor">
            <param name="context">abc</param>
            <param name="appendPrefix">true</param>
            <message>User:</message>
        </field-validator>
    </field>
</validators>
```

其中context参数将作为验证User类属性的文件名的一部分，如user属性返回一个User对象，那么用于验证User对象属性的文件名为User-abc-validation.xml。这个文件要和User.class文件在同一个目录中。appendPrefix表示是否在字段里加user，如果为true，Struts2就会使用user.name在form提交的数据中查找要验证的数据。这个属性的默认值是true。如果出错，Struts2会将<message>标签中的信息加到User-abc-validation.xml文件中的相应错误信息前面。

User-abc-validation.xml文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
 "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
    <field name="name">
        <field-validator type="requiredstring">
            <message>请输入name</message>
        </field-validator>
    </field>
    <field name="age">
        <field-validator type="int">
            <param name="min">5</param>
            <param name="max">20</param>
            <message>
                必须在 5至20之间
            </message>
        </field-validator>
```

```
    </field>
</validators>
```

下面修改validate_form.jsp，代码如下：

```
<s:form validate="true" action="new_validate" namespace="/test" >
        <s:textfield name="msg" label="姓名"   />
        <s:textfield name="age" label="年龄"/>
        <s:textfield name="user.name" label="姓名1"   />
        <s:textfield name="user.age" label="年龄1"/>
      <s:submit/>
</s:form>
```

大家可以看到，最后两个<s:textfield>的name属性是user.name和user.age，正好是加了前缀的。

现在重新访问 http://localhost:8080/validation/validate_form.jsp，验证界面如图2所示。

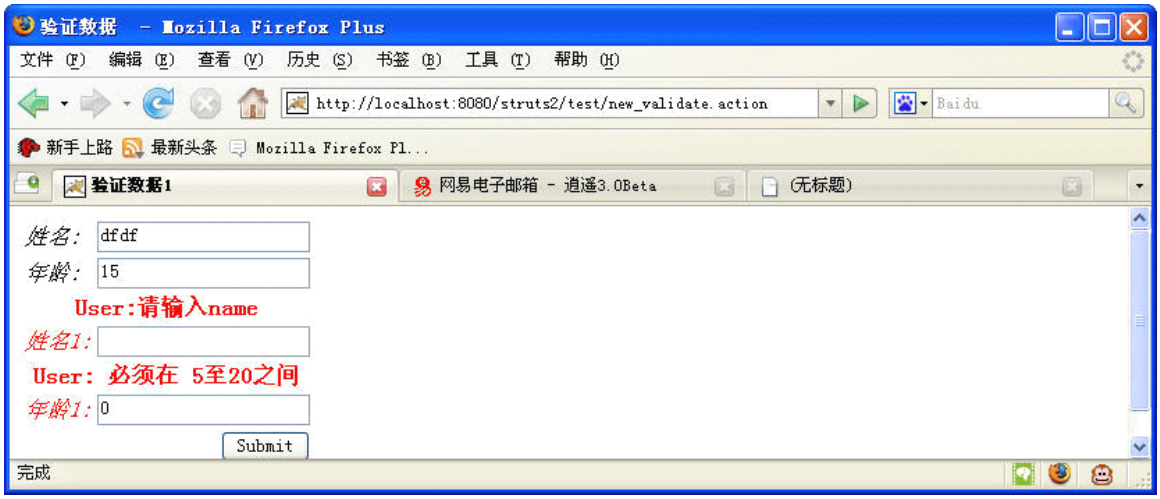

图2

经笔者测试，使用visitor无法以客户端验证的方式来验证user属性，但NewValidateAction中其他的属性可以使用客户端测试。

Struts2教程6：在Action类中获得HttpServletResponse对象

的四种方法

在struts1.x Action类的execute方法中，有四个参数，其中两个就是response和request。而在Struts2中，并没有任何参数，因此，就不能简单地从execute方法获得HttpServletResponse或HttpServletRequest对象了。

但在Struts2 Action类中仍然有很多方法可以获得这些对象。下面就列出四种获得这些对象的方法。

【方法1】使用Struts2 Aware拦截器

这种方法需要Action类实现相应的拦截器接口。如我们要获得HttpServletResponse对象，需要实现org.apache.struts2.interceptor.ServletResponseAware接口，代码如下：

```
package action;

import com.opensymphony.xwork2.ActionSupport;
import javax.servlet.http.*;
import org.apache.struts2.interceptor.*;

public class MyAction extends ActionSupport implements ServletResponseAware
{
    private javax.servlet.http.HttpServletResponse response;
    // 获得HttpServletResponse对象
    public void setServletResponse(HttpServletResponse response)
    {
        this.response = response;
    }
    public String execute() throws Exception
    {
        response.getWriter().write("实现ServletResponseAware接口");
    }
}
```

在上面的代码中，MyAction实现了一个ServletResponseAware接口，并且实现了setServletResponse方法。如果一个动作类实现了ServletResponseAware接口，Struts2在调用execute方法之前，就会先调用setServletResponse方法，并将response参数传入这个方法。如果想获得HttpServletRequest、HttpSession和Cookie等对象，动作类可以分别实现ServletRequestAware、SessionAware和CookiesAware等接口。这些接口都在org.apache.struts2.interceptor包中。

如果要获得请求参数，动作类可以实现org.apache.struts2.interceptor. ParameterAware接口，但如果只想判断某个参数是否存在，也可以实现com.opensymphony.xwork2.interceptor.

ParameterNameAware接口。这个接口有一个acceptableParameterName 方法，当Struts2获得一个请求参数时，就会调用一次。读者可以在这个方法中将所有的请求参数记录下来，以便以后使用。这个方法的定义如下：

boolean acceptableParameterName(String parameterName);

【方法2】使用RequestAware拦截器

这种方法和第 1 种方法类似。动作类需要实现一个org.apache.struts2.interceptor.RequestAware接口。所不同的是RequestAware将获得一个com.opensymphony.xwork2.util.OgnlValueStack对象，这个对象可以获得 response、request及其他的一些信息。代码如下所示：

```
package action;

import java.util.Map;
import org.apache.struts2.*;
import com.opensymphony.xwork2.ActionSupport;
import javax.servlet.http.*;
import com.opensymphony.xwork2.util.*;
import org.apache.struts2.interceptor.*;

public class FirstAction extends ActionSupport implements RequestAware
{
    private Map request;
private HttpServletResponse response;

    public void setRequest(Map request)
    {
        this.request = request;
    }
    public String execute() throws Exception
    {
        java.util.Set<String> keys = request.keySet();
        // 枚举所有的key值。实际上只有一个key：struts.valueStack
        for(String key: keys)
            System.out.println(key);
        // 获得OgnlValueStack 对象
        OgnlValueStack stack = (OgnlValueStack)myRequest.get("struts.valueStack");
        // 获得HttpServletResponse对象
        response =
(HttpServletResponse)stack.getContext().get(StrutsStatics.HTTP_RESPONSE);
        response.getWriter().write("实现RequestAware 接口");
    }
}
```

我们也可以使用StrutsStatics.HTTP_REQUEST、StrutsStatics.PAGE_CONTEXT来获得HttpServletRequest和PageContext对象。这种方法有些麻烦，一般很少用，读者可以作为一个参考。

【方法3】使用ActionContext类

这种方法比较简单，我们可以通过org.apache.struts2.ActionContext类的get方法获得相应的对象。代码如下：

```
HttpServletResponse response(HttpServletResponse) =
ActionContext.getContext().get(org.apache.struts2.StrutsStatics.HTTP_RESPONSE);
HttpServletRequest request(HttpServletRequest) =
ActionContext.getContext().get(org.apache.struts2.StrutsStatics.HTTP_REQUEST);
```

【方法4】使用ServletActionContext类

Struts2为我们提供了一种最简单的方法获得HttpServletResponse及其他对象。这就是org.apache.struts2.ServletActionContext类。我们可以直接使用ServletActionContext类的getRequest、getResponse方法来获得HttpServletRequest、HttpServletResponse对象。代码如下：

```
HttpServletResponse response = ServletActionContext.getResponse()
response.getWriter().write("hello world");
```

从这四种方法来看，最后一种是最简单的，读者可以根据自己的需要和要求来选择使用哪一种方法来获得这些对象。

Struts2教程7：上传任意多个文件

一、上传单个文件

上传文件是很多Web程序都具有的功能。在Struts1.x中已经提供了用于上传文件的组件。而在Struts2中提供了一个更为容易操作的上传文件组件。所不同的是，Struts1.x的上传组件需要一个ActionForm来传递文件，而Struts2的上传组件是一个拦截器（这个拦截器不用配置，是自动装载的）。在本文中先介绍一下如何用struts2上传单个文件，最后介绍一下用struts2上传任意多个文件。

要用Struts2实现上传单个文件的功能非常容易实现，只要使用普通的Action即可。但为了获得一些上传文件的信息，如上传文件名、上传文件类型以及上传文件的Stream对象，就需要按着一定规则来为Action类增加一些getter和setter方法。

在Struts2中，用于获得和设置java.io.File对象（Struts2将文件上传到临时路径，并使用java.io.File打开这个临时文件）的方法是getUpload和setUpload。获得和设置文件名的方法是getUploadFileName和setUploadFileName，获得和设置上传文件内容类型的方法是getUploadContentType和setUploadContentType。下面是用于上传的动作类的完整代码：

```
package action;

import java.io.*;
import com.opensymphony.xwork2.ActionSupport;

public class UploadAction extends ActionSupport
{
    private File upload;
    private String fileName;
    private String uploadContentType;

    public String getUploadFileName()
    {
        return fileName;
    }

    public void setUploadFileName(String fileName)
    {
        this.fileName = fileName;
    }

    public File getUpload()
    {
        return upload;
    }

    public void setUpload(File upload)
    {
        this.upload = upload;
    }
    public void setUploadContentType(String contentType)
    {
        this.uploadContentType=contentType;

    }

    public String getUploadContentType()
    {
        return this.uploadContentType;
    }
    public String execute() throws Exception
    {
        java.io.InputStream is = new java.io.FileInputStream(upload);
```

```
            java.io.OutputStream os = new java.io.FileOutputStream("d:\\upload\\" + fileName);
            byte buffer[] = new byte[8192];
            int count = 0;
            while((count = is.read(buffer)) > 0)
            {
                os.write(buffer, 0, count);
            }
            os.close();
            is.close();
            return SUCCESS;
        }
}
```

在execute方法中的实现代码就很简单了，只是从临时文件复制到指定的路径（在这里是d:\upload）中。上传文件的临时目录的默认值是javax.servlet.context.tempdir的值，但可以通过struts.properties（和struts.xml在同一个目录下）的struts.multipart.saveDir属性设置。Struts2上传文件的默认大小限制是2M（2097152字节），也可以通过struts.properties文件中的struts.multipart.maxSize修改，如struts.multipart.maxSize=2048 表示一次上传文件的总大小不能超过2K字节。

下面的代码是上传文件的JSP页面代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
<%@ taglib prefix="s" uri="/struts-tags"%>

<html>
    <head>
        <title>上传单个文件</title>
    </head>

    <body>
        <s:form action="upload" namespace="/test"
            enctype="multipart/form-data">
            <s:file name="upload" label="输入要上传的文件名" />
            <s:submit value="上传" />
        </s:form>

    </body>
</html>
```

也可以在success.jsp页中通过<s:property>获得文件的属性（文件名和文件内容类型），代码如下：

```
    <s:property value="uploadFileName"/>
```

二、上传任意多个文件

在Struts2中，上传任意多个文件也非常容易实现。首先，要想上传任意多个文件，需要在客户端使用DOM技术生成任意多个<input type=”file” />标签。name属性值都相同。代码如下：

```
<html>
    <head>
        <script language="javascript">

function addComponent()
{
        var uploadHTML = document.createElement( "<input type='file'  name='upload'/>");
        document.getElementById("files").appendChild(uploadHTML);
        uploadHTML = document.createElement( "<p/>");
        document.getElementById("files").appendChild(uploadHTML);
}
</script>
    </head>
    <body>
        <input type="button" onclick="addComponent();" value="添加文件" />
        <br />
        <form onsubmit="return true;" action="/struts2/test/upload.action"
            method="post" enctype="multipart/form-data">
            <span id="files"> <input type='file' name='upload' />
                <p />
            </span>
            <input type="submit" value="上传" />
        </form>
    </body>

</html>
```

上面的javascript代码可以生成任意多个<input type=’file’>标签，name的值都为file（要注意的是，上面的javascript代码只适合于IE浏览器，firefox等其他浏览器需要使用他的代码）。至于Action类，和上传单个文件的Action类基本一至，只需要将三个属性的类型改为List即可。代码如下：

```
package action;

import java.io.*;
import com.opensymphony.xwork2.ActionSupport;

public class UploadMoreAction extends ActionSupport
{
```

```
private java.util.List<File> uploads;
private java.util.List<String> fileNames;
private java.util.List<String> uploadContentTypes;

public java.util.List<String> getUploadFileName()
{
	return fileNames;
}
public void setUploadFileName(java.util.List<String> fileNames)
{
	this.fileNames = fileNames;
}
public java.util.List<File> getUpload()
{
	return uploads;
}

public void setUpload(java.util.List<File> uploads)
{
	this.uploads = uploads;
}

public void setUploadContentType(java.util.List<String> contentTypes)
{
	this.uploadContentTypes = contentTypes;

}

public java.util.List<String> getUploadContentType()
{
	return this.uploadContentTypes;
}

public String execute() throws Exception
{
	if (uploads != null)
	{
		int i = 0;
		for (; i < uploads.size(); i++)
		{
			java.io.InputStream is = new java.io.FileInputStream(uploads.get(i));
			java.io.OutputStream os = new java.io.FileOutputStream(
					"d:\\upload\\" + fileNames.get(i));
			byte buffer[] = new byte[8192];
```

```
                int count = 0;
                while ((count = is.read(buffer)) > 0)
                {
                    os.write(buffer, 0, count);
                }
                os.close();
                is.close();
            }
        }
        return SUCCESS;
    }
}
```

在execute方法中，只是对List对象进行枚举，在循环中的代码和上传单个文件时的代码基本相同。如果读者使用过struts1.x的上传组件，是不是感觉Struts2的上传功能更容易实现呢？在Struts1.x中上传多个文件时，可是需要建立带索引的属性的。而在Struts2中，就是这么简单就搞定了。图1是上传任意多个文件的界面。

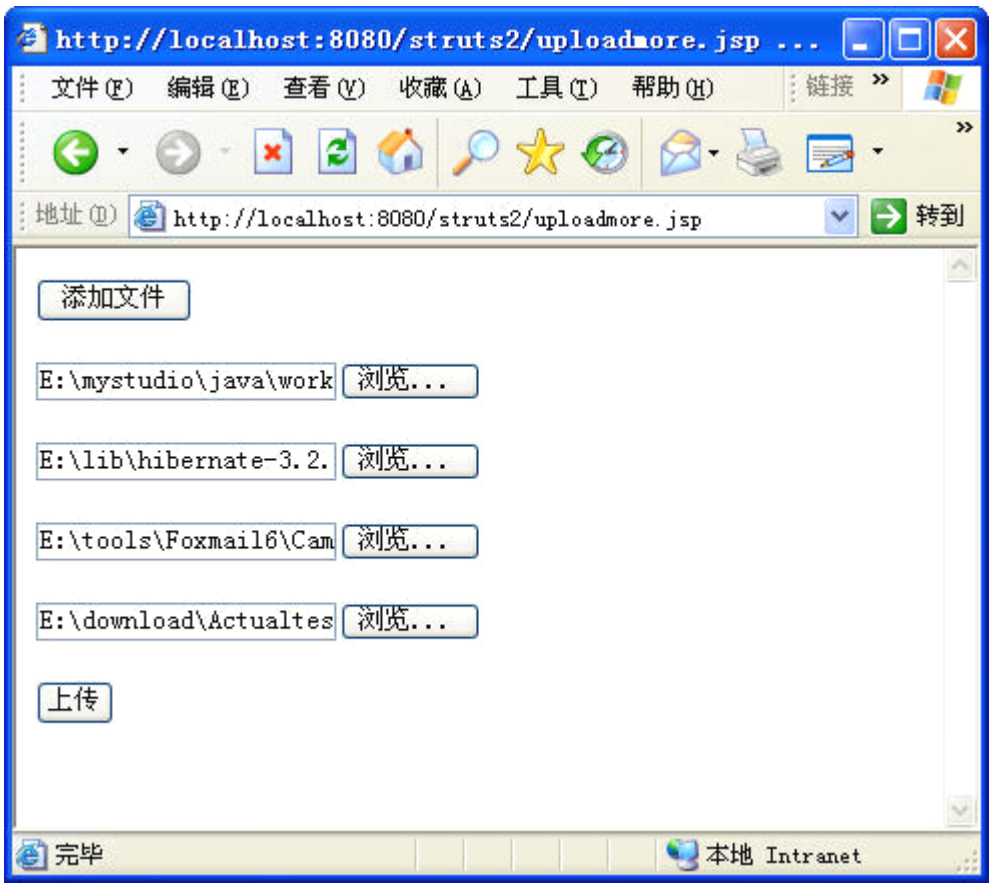

图1

在本文中给出了用Struts2上传任意多个文件的一个方法，如果哪位读者有更好的方法，请跟贴！

Struts2教程8：拦截器概述

Struts2的拦截器和Servlet过滤器类似。在执行Action的execute方法之前，Struts2会首先执行在struts.xml中引用的拦截器，在执行完所有引用的拦截器的intercept方法后，会执行Action的execute方法。

Struts2拦截器类必须从com.opensymphony.xwork2.interceptor.Interceptor接口继承，在Intercepter接口中有如下三个方法需要实现：

```
void destroy();
```

```
void init();
String intercept(ActionInvocation invocation) throws Exception;
```

其中intercept方法是拦截器的核心方法，所有安装的拦截器都会调用之个方法。在Struts2中已经在struts-default.xml中预定义了一些自带的拦截器，如timer、params等。如果在<package>标签中继承struts-default，则当前package就会自动拥有struts-default.xml中的所有配置。代码如下：

```
<package name="demo" extends="struts-default" > ... </package>
```

在struts-default.xml中有一个默认的引用，在默认情况下（也就是<action>中未引用拦截器时）会自动引用一些拦截器。这个默认的拦截器引用如下：

```
<default-interceptor-ref name="defaultStack"/>
```

```
<interceptor-stack name="defaultStack">
    <interceptor-ref name="exception"/>
    <interceptor-ref name="alias"/>
    <interceptor-ref name="servletConfig"/>
    <interceptor-ref name="prepare"/>
    <interceptor-ref name="i18n"/>
    <interceptor-ref name="chain"/>
    <interceptor-ref name="debugging"/>
    <interceptor-ref name="profiling"/>
    <interceptor-ref name="scopedModelDriven"/>
    <interceptor-ref name="modelDriven"/>
    <interceptor-ref name="fileUpload"/>
    <interceptor-ref name="checkbox"/>
    <interceptor-ref name="staticParams"/>
    <interceptor-ref name="params">
```

```
            <param name="excludeParams">dojo\..*</param>
        </interceptor-ref>
        <interceptor-ref name="conversionError"/>
        <interceptor-ref name="validation">
                <param name="excludeMethods">input,back,cancel,browse</param>
         </interceptor-ref>
         <interceptor-ref name="workflow">
                 <param name="excludeMethods">input,back,cancel,browse</param>
         </interceptor-ref>
</interceptor-stack>
```

上面在defaultStack中引用的拦截器都可以在<action>中不经过引用就可以使用（如果在<action>中引用了任何拦截器后，要使用在defaultStack中定义的拦截器，也需要在<action>中重新引用，在后面将详细讲解）。

下面我们来看几个简单的拦截器的使用方法。

一、记录拦截器和execute方法的执行时间(timer)

timer 是 Struts2 中 最 简 单 的 拦 截 器 ， 这 个 拦 截 器 对 应 的 类 是 com.opensymphony.xwork2.interceptor.TimerInterceptor。它的功能是记录execute方法和其他拦截器（在timer后面定义的拦截器）的intercept方法执行的时间总和。如下面的配置代码所示：

```
<action name="first" class="action.FirstAction">
    <interceptor-ref name="logger"/>
    <interceptor-ref name="timer" />
</action>
```

由于在timer后面没有其他的拦截器定义，因此，timer只能记录execute方法的执行时间，在访问first动作时，会在控制台输出类似下面的一条信息：

信息: Executed action [/test/first!execute] took 16 ms.

在使用timer拦截器时，需要commons-logging.jar 的支持。将logger引用放到timer的后面，就可以记录logger拦截器的intercept方法和Action的execute方法的执行时间总和，代码如下：

```
<action name="first" class="action.FirstAction">
    <interceptor-ref name="timer" />
    <interceptor-ref name="logger"/>
</action>
```

大家可以使用如下的Action类来测试一下timer拦截器：

```
package action;
```

```
import com.opensymphony.xwork2.ActionSupport;

public class FirstAction extends ActionSupport

{
        public String execute() throws Exception

        {
           Thread.sleep(1000); //  延迟1秒
              return null;
        }

}
```

如果只记录execute方法的执行时间，一般会输出如下的信息：

信息: Executed action [/test/first!execute] took 1000 ms.

二、通过请求调用Action的setter方法(params)

当客户端的一个form向服务端提交请求时，如有一个textfield，代码如下：

```
<s:form action="first" namespace="/test">
    <s:textfield name="name"/>
    <s:submit/>
</s:form>
```

在提交后，Struts2将会自动调用first动作类中的setName方法，并将name文本框中的值通过setName方法的参数传入。实际上，这个操作是由params拦截器完成的，params对应的类是com.opensymphony.xwork2.interceptor.ParametersInterceptor。由于params已经在defaultStack中定义，因此，在未引用拦截器的<action>中是会自动引用params的，如下面的配置代码，在访问first动作时，Struts2是会自动执行相应的setter方法的。

```
<action name="first" class="action.FirstAction">
    ... ...
</action>
```

但如果在<action>中引用了其他的拦截器，就必须再次引用params拦截器，Struts2才能调用相应的setter方法。如下面的配置代码所示：

```
<action name="first" class="action.FirstAction">
     <interceptor-ref name="timer" />
     <interceptor-ref name="params"/>
</action>
```

三、通过配置参数调用Action的setter方法(static-params)

static-params拦截器可以通过配置<params>标签来调用Action类的相应的setter方法，static-params拦截器对应的类是com.opensymphony.xwork2.interceptor.StaticParametersInterceptor。

下面配置代码演示了如何使用static-params拦截器：

```
<action name="first" class="action.FirstAction">
    <interceptor-ref name="timer" />
    <param name="who">比尔</param>
    <interceptor-ref name="params"/>
    <interceptor-ref name="static-params"/>
</action>
```

如果first动作使用上面的配置，在访问first动作时，Struts2会自动调用setWho方法将“比尔”作为参数值传入setWho方法。

四、使用拦截器栈

为了能在多个动作中方便地引用同一个或几个拦截器，可以使用拦截器栈将这些拦截器作为一个整体来引用。拦截器栈要在<package>标签中使用<interceptors>和子标签<interceptor-stack>来定义。代码如下：

```
<package name="demo" extends="struts-default" >
    <interceptors>
        <interceptor-stack name="mystack">
            <interceptor-ref name="timer" />
            <interceptor-ref name="logger" />
            <interceptor-ref name="params" />
            <interceptor-ref name="static-params" />
        </interceptor-stack>
    </interceptors>

    <action name="first" class="action.FirstAction">
        <param name="who">比尔</param>
        <interceptor-ref name="mystack"/>
    </action>
</package>
```

可以象使用拦截器一样使用拦截器栈，如上面代码所示。

Struts2教程9：实现自己的拦截器

在上一篇中介绍了Struts2拦截器的原理，在这一篇中我们将学习一下如何编写自己的拦截

器。

一、拦截器的实现

实现一个拦截器非常简单。实际上，一个拦截器就是一个普通的类，只是这个类必须实现com.opensymphony.xwork2.interceptor.Interceptor接口。Interceptor接口有如下三个方法：

```
public interface Interceptor extends Serializable
{
    void destroy();
    void init();
    String intercept(ActionInvocation invocation) throws Exception;
}
```

其中init和destroy方法只在拦截器加载和释放（都由Struts2自身处理）时执行一次。而intercept方法在每次访问动作时都会被调用。Struts2在调用拦截器时，每个拦截器类只有一个对象实例，而所有引用这个拦截器的动作都共享这一个拦截器类的对象实例，因此，在实现Interceptor接口的类中如果使用类变量，要注意同步问题。

下面我们来实现一个简单的拦截器，这个拦截器通过请求参数action指定一个拦截器类中的方法，并调用这个方法（我们可以使用这个拦截器对某一特定的动作进行预处理）。如果方法不存在，或是action参数不存在，则继续执行下面的代码。如下面的URL：

http://localhost:8080/struts2/test/interceptor.action?action=test

访问上面的url后，拦截器会就会调用拦截器中的test方法，如果这个方法不存在，则调用invocation.invoke方法， invoke方法和Servlet过滤器中调用FilterChain.doFilter方法类似，如果在当前拦截器后面还有其他的拦截器，则invoke方法就是调用后面拦截器的intercept方法，否则，invoke会调用Action类的execute方法（或其他的执行方法）。

下面我们先来实现一个拦截器的父类ActionInterceptor。这个类主要实现了根据action参数值来调用方法的功能，代码如下：

```
package interceptor;

import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.Interceptor;
import javax.servlet.http.*;
import org.apache.struts2.*;

public class ActionInterceptor implements Interceptor
{
    protected final String INVOKE = "##invoke";
```

```
    public void destroy()
    {
        System.out.println("destroy");
    }

    public void init()
    {
        System.out.println("init");
    }

    public String intercept(ActionInvocation invocation) throws Exception
    {
        HttpServletRequest request = ServletActionContext.getRequest();
        String action = request.getParameter("action");
        System.out.println(this.hashCode());
        if (action != null)
        {
            try
            {
                java.lang.reflect.Method method = this.getClass().getMethod(action);
                String result = (String)method.invoke(this);
                if(result != null)
                {
                    if(!result.equals(INVOKE))
                        return result;
                }
                else
                    return null;
            }
            catch (Exception e)
            {
            }
        }
        return invocation.invoke();
    }
}
```

从上面代码中的intercept方法可以看出，在调用action所指定的方法后，来判断返回值。可能发生的情况有三种：

1. 返回值为null，执行return null。
2. 返回值为INVOKE，执行return invockation.invoke()。
3. 其他情况，执行return result。 result表示指定方法的返回值，如上面代码所示。

在实现完上面的拦截器父类后，任何继承于ActionInterceptor类的拦截器都可以自动根

据action的参数值调用自身的相应方法。下面我们来实现一个拥有两个动作方法test和print的拦截器类。代码如下：

```
package interceptor;

import javax.servlet.http.HttpServletResponse;
import org.apache.struts2.ServletActionContext;

public class MultiMethodInterceptor extends ActionInterceptor
{
    public String test() throws Exception
    {
        HttpServletResponse response = ServletActionContext.getResponse();
        response.getWriter().println("invoke test");
        return this.INVOKE;
    }

    public String print() throws Exception
    {
        HttpServletResponse response = ServletActionContext.getResponse();
        response.getWriter().println("invoke print");

        return null;
    }
}
```

test方法返回了INVOKE，因此，在执行完这个方法后，Struts2会接着调用其他拦截器的intercept方法或Action类的execute方法。而print方法在执行完后，只是返回了null，而不再调用其他的方法了，也就是访问如下的url时，动作的execute方法将不会执行：

http://localhost:8080/struts2/test/ddd.action?action=print

下面我们来实现一个Action类，代码如下：

```
package action;

import org.apache.struts2.*;
import com.opensymphony.xwork2.ActionSupport;

public class InterceptorAction extends ActionSupport
{
    public String abcd() throws Exception
    {
```

```
            ServletActionContext.getResponse().getWriter()
                    .println("invoke abcd");
            return null;
        }
}
```

在这个Action类中，只有一个abcd方法，实际上，这个方法相当于execute方法，在下面会设置动作的method属性为abcd。下面我们来在struts.xml中定义拦截器类和动作，代码如下：

```
<?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>
    <package name="demo" extends="struts-default" namespace="/test">
        <interceptors>
            <interceptor name="method" class="interceptor.MultiMethodInterceptor" />
                <interceptor-stack name="methodStack">
                    <interceptor-ref name="method" />
                    <interceptor-ref name="defaultStack" />
                </interceptor-stack>
        </interceptors>

        <action name="interceptor" class="action.InterceptorAction" method="abcd">
            <interceptor-ref name="methodStack" />
        </action>
    </package>
</struts>
```

在配置上面的methodStack拦截器时要注意，最好在后面引用defaultStack，否则很多通过拦截器提供的功能将失去。

OK，现在访问如下的URL：

http://localhost:8080/struts2/test/ddd.action?action=test

在浏览器中将会出现如下的字符串：

invoke test

invoke abcd

而如果访问http://localhost:8080/struts2/test/ddd.action?action=print，将会只出现如下的字符串：

invoke print

大家可以看出，访问这个url时并没有调用abcd方法。如果随便指定的action值的话，则只调用abcd方法，如访问http://localhost:8080/struts2/test/ddd.action?action=aaa，就只会输出invoke abcd。

二、拦截器的参数

我们在使用很多Struts2内置的拦截器时会发现有很多拦截器都带参数，当然。我们自己做的拦截器也可以加上同样的参数。有两个参数比较常用，这两个参数是includeMethods和excludeMethods，其中includeMethods指定了拦截器要调用的Action类的执行方法（默认是execute），也就是说，只有在includeMethods中指定的方法才会被Struts2调用，而excludeMethods恰恰相反，在这个参数中指定的执行方法不会被Struts2调用。如果有多个方法，中间用逗号(,)分隔。在Struts2中提供了一个抽象类来处理这两个参数。这个类如下：

com.opensymphony.xwork2.interceptor.MethodFilterInterceptor

如有继承于这个类的拦截器类都会自动处理includeMethods和excludeMethods参数，如下面的拦截器类所示：

```
package interceptor;

import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.*;

public class MyFilterInterceptor extends MethodFilterInterceptor
{
    private String name;
    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
    @Override
    protected String doIntercept(ActionInvocation invocation) throws Exception
    {
        System.out.println("doIntercept");
        System.out.println(name);
        return invocation.invoke();
    }
```

}

MethodFilterInterceptor的子类需要实现doIntercept方法（相当于Interceptor的intercept方法），如上面代码所示。在上面的代码中还有一个name属性，是为了读取拦截器的name属性而设置的，如下面的配置代码所示：

```
<?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="demo" extends="struts-default" namespace="/test">
        <interceptors>
            <interceptor name="method" class="interceptor.MultiMethodInterceptor" />
                <interceptor name="filter"
                    class="interceptor.MyFilterInterceptor">
                    <param name="includeMethods">abcd</param>
                    <param name="name">中国</param>
                </interceptor>
                <interceptor-stack name="methodStack">
                    <interceptor-ref name="method" />
                    <interceptor-ref name="filter" />
                    <interceptor-ref name="defaultStack" />
                </interceptor-stack>
        </interceptors>

        <action name="interceptor" class="action.InterceptorAction" method="abcd">
            <interceptor-ref name="methodStack" />
        </action>
    </package>
</struts>
```

再次访问 http://localhost:8080/struts2/test/ddd.action?action=test, Struts2就会调用MyFilterInterceptor的doIntercept方法来输出name属性值。如果将上面的includeMethods参数值中的abcd去掉，则Action类的abcd方法不会被执行。

Struts2教程10：国际化

国际化的作用就是根据不同国家的用户在访问Web或其他类型的程序时，将各种信息以本地的常用形式显示出来，如界面信息在中国，就会显示中文信息，在以英文为主的国家里，就会显示英文信息。还有就是一些信息的格式，如日期格式等。

从属性文件中获得字符串信息是国际化的基本应用。在Struts2中使用的属性文件就是Java

属性文件，扩展名为properties。在Struts2中的属性文件可以有很多默认的位置，Struts2可按如下的顺序（或步骤）来定位属性文件：

1. ActionClass.properties：属性文件名和动作类同名。Struts2会首先查询与当前访问的动作类同名，并且和ActionClass.class在同一个目录下的属性文件。

2. BaseClass.properties： BaseClass表示动作类的基类。所有动作类都会查找Object.properties文件（因为Object是所有Java类的基类），但要注意的是Object.properties文件可不能放到当前动作类的目录中，由于Object在java.lang包中，因此，Object.properties要放到jdk包的java"lang目录中。而对于ActionSupport.properties文件，当然也不能放到动作类的当前目录中，由于ActionSupport类中com.opensymphony.xwork2名中，因此，需要将ActionSupport.properties文件放到xwork2.jar包中的com\opensymphony\xwork2目录中，由于放到jar文件中不太方便，因此，可以使一个和当前动作类在一个目录的类先继承ActionSupport，然后所有的动作类都继承于这个类。代码如下：

```
public class MyActionSupport extends ActionSupport
{
    ... ...
}
public class ActionClass extends MyActionSupport
{
    ... ...
}
```

这样的话，只要存在一个MyActionSupport.properties，在当前目录下的所有动作类都会读取这个文件。

3. Interface.properties：这类文件和BaseClass.properties类似，Interface表示动作类实现的接口。

4. 如果动作类实现了ModelDriven，那么重复第1步。

5. package.properties：大家要注意。这个文件就叫 package.properties。不象Interface和BaseClass都是泛指。这个文件可以放到当前动作类的包的任何一层目录下。如当前动作类在action.test包中。那么package.properties可以放到action目录中，也可以放到action"test目录中。Struts2会从离动作类最近的位置开始查找package.properties文件。

6. 搜索il8n资源信息

7. 查找全局资源属性文件

如下面是一个动作类

```
package action.test;

import org.apache.struts2.*;
import com.opensymphony.xwork2.ActionSupport;
```

```
public class Internationalizing  extends ActionSupport
{
    public String execute() throws Exception
    {
        return "forward";
    }
}
```

在action\test目录下有一个Internationalizing.properties文件，内容如下：

```
delete =  删除
save =  保存
```

我们可以在jsp文件中使用如下几种方法取出资源信息：

```
<s:property value="getText('delete')"/>
<s:text name="save" />
```

3. 使用<s:il8n>标签。这个标签可以直接定位属性文件，如abc.properties在WEB-INF\classes\test目录下，内容和Internationalizing.properties一样，则可以使用如下的代码读取abc.properties的内容：

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<s:i18n name="test.abc">
    <s:text name="save" />
    <s:text name="delete" />
</s:i18n>
```

当然，我们也可以使用全局的属性文件，在WEB-INF"classes目录下建立一个struts.properties文件，内容如下：

```
struts.custom.i18n.resources=my
```

在WEB-INF\classes目录下建立一个my.properties文件，当Struts2按着上述的顺序没有找到相应的属性文件时，最后就会考虑寻找全局的属性文件，因此，就会找到my.properties。

还可以通过属性文件名来让Struts2按着客户端浏览器的语言环境来找符合某种语言的属性文件。如有三个属性文件language.properties、language_en.properties、language_zh.properties。如果客户端的语言是中文，Struts2就会读language_zh.properties，如果是英文，就会读language_en.properties。如果这两个文件的某个不存在，就会读language.properties。读者可通过IE的[工具]->[Internet]->[语言]来测试客户端浏览器的语言，如图1所示：

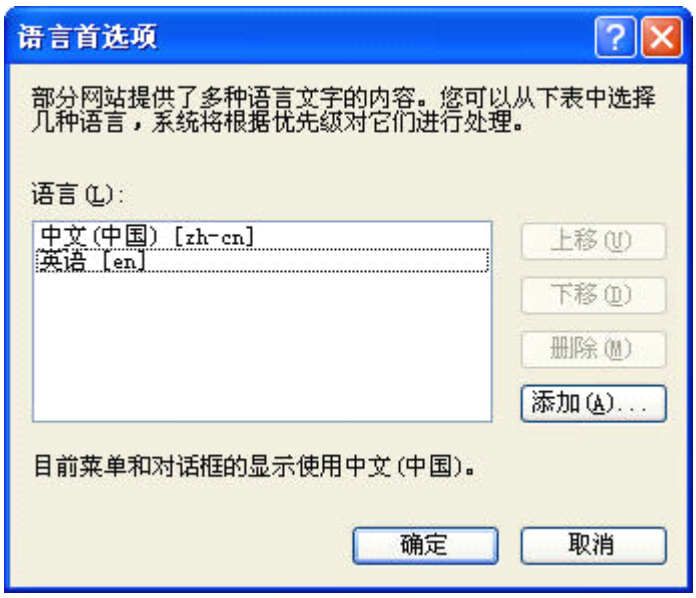

图1

2008年7月5日