第1章 领略清晰的MVC流程（Struts讲解）

Struts 是学习轻量级的 J2EE 框架必须了解的一个框架，它的各种优点使得它成为了目前最流行的三个框架之一，它实现了 MVC 模式，本章主要对 Struts 所实现的 MVC 流程，Struts 的配置，Struts 标签进行介绍。

1.1 Struts 的流程

本节将着重介绍 Struts 的基本流程，并对个流程进行简单的分析。

Struts 的业务流程比较复杂，但仔细体会也不难理解，为了便于读者理解这里使用一个流程图对 Struts 的运行机制进行描述，如图 1-1 所示：

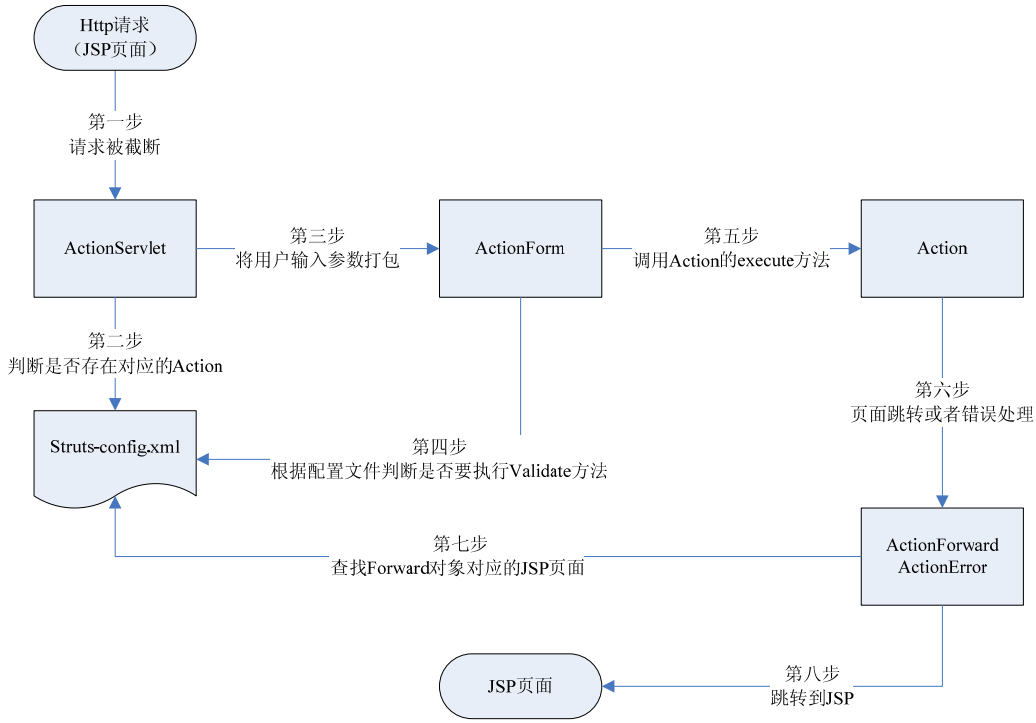


图 1-1 Struts 业务流程图

下面针对上面的流程，做一些必要的讲评。

（1）第一步，一般情况下，用户所有的以 do 结尾的请求都会被一个叫 ActionServlet 截获。ActionServlet 是 Struts 中一个特殊的 Servlet，它本质上还是一个 HttpServlet，只是它在 Struts 中充当“门卫”的作用。 当有以 do 结尾的 Http 请求时，它会认为这

是在向它要服务。接下来，它要做的事情就是查找这个“服务”是否存在。也就是 Action。ActionServlet 的存在体现了一种在门槛集中控制的思想。以使得后面的工作有序的进行。

（2）第二步，这一步牵涉到一个配置文件，它叫 struts-config.xml。这个文件中包含了 ActionForm,Action，Forward 等信息，它的存在主要有两个意义，第一，它告诉人们这个系统“有什么”。第二，它告诉系统其它组件请求的服务“在哪里”。来自第一步的 Http 请求就会在这个文件中查找，判断其是否存在对应的 Action，如果有则继续下去，否则进行错误处理，这就体现了“有什么”的意义。 下面步骤中会提到“在哪里”的意义。总之，struts-config.xml 起到一种站点地图的作用。

（3）第三步，在 Struts 观点中，一个 JSP 的后面应该跟着一个 ActionForm。尤其是当这个 JSP 包含表单的时候。于是当表单提交的时候，这个 ActionForm，就会调用它的 Setter 方法将字段进行填充。然后,一般是在 Action 部分调用 getter 方法将字段内容进行提取，起到这种作用的 Java 组件，非 JavaBean 莫属。所以 ActionForm 本质上就是一个 JavaBean，在视图和控制器中充当数据通信兵。

（4）第四步，这时候又需要查看 struts-config.xml。看来它还负责告诉系统组件可以“做什么”。ActionForm 把数据填充完毕后，正打算将这些数据打包传送给 Action，但它不能这样自作主张，先要查看下 struts-config.xml 确定下是否还有其它的事要做，最典型的就是是否需要调用 validate 对数据进检验。

（5）第五步，ActionForm 当前面的工作都顺利通过后，就可以顺理成章的将数据打包给 Action 了。Action 会调用它的 execute 方法，打算将封装在 ActionForm 中的数据移交到别的地方，从 struts 的设计思想来看，Action 最好不要一厢情愿的去对 ActionForm 中的数据进行复杂的处理，因为那不是一个控制器应该做的。最好还是留点事情给模型层做。

（6）第六步， Action 把 ActionForm 中的数据提交给模型层如此这般（业务逻辑）后，还需要负责将处理的结果在视图层呈现出来，一个可行的做法是使用最原始的办法在 Action 中进行页面跳转，这时候就像编写古老的 servlet 一样。但 struts 认为所有的页面跳转都可以统一起来管理，于是就有了 ActionForward 这个对象。通过这个对象，Action 可以查找 struts-config.xml 文件来确定它所请求的页面的位置（struts-config.xml “在哪里”的意义得以体现）。在实际开发中，我们也的确会体验到这种集中管理的好处。

1.2 Struts 的配置

1.2.1 struts-config.xml 的配置

在讲述配置 struts-config.xml 之前，需要讲解一下如何使得一个 web 应用具有 struts 能力。

在 MyEclicpse J2EE Development 透视图中选择 MyEclipse|add struts capablitity 命令，打开“Struts support for MyEclipse Web Project”对话框，如图 1-2 所示：

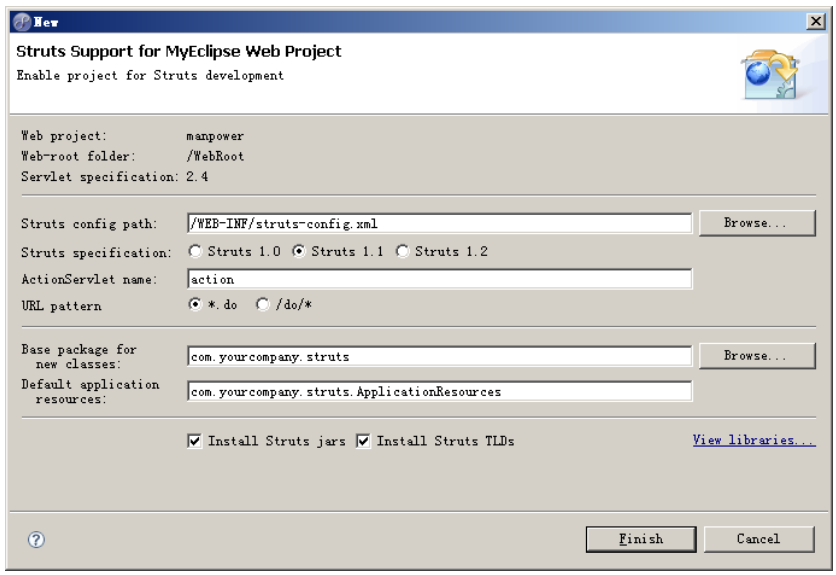


图 1-2 添加 Struts 能力

- Struts config path 表示 struts-config.xml 文件存放的路径，当然 struts-confg.xml 本身文件名也是可以修改的。
- Struts-specification 表示使用 struts 的版本
- ActionServlet name 表示的是在 web.xml 中 servlet 的名字
- URL pattern 表示 Struts 截获的地址类型，默认情况下截获所有以 do 结尾的地址。
- Base package for new classes 表示资源文件存放的包路径和以后新建文件的一个包路径，可以修改。
- Default appliation resources 表示默认资源文件的路径。

单击 Finish 按钮后，一个空的 web project 就会多出很多的导入包和一些标签文件，如图 1-3 所示。

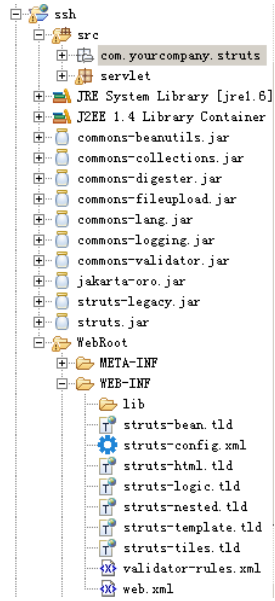


图 1-3 具有 Struts 能力的 web project

这时候，一个 web Project 就具备了 Struts 的能力 ,同时我们也看到了它自动生成的 struts-config.xml 文件，下面的工作就是针对这个文件而言的。

文件配置前的内容如下

```
<struts-config>
<!—配置数据源-->

  <data-sources />
<!—配置 ActionForm-->

  <form-beans />
<!—配置异常-->

  <global-exceptions />
<!—配置全局 Forward，相对应的可以在 ActionMapping 中配置局部 Forward-->

  <global-forwards />
<!—配置 Action 信息-->

  <action-mappings />
<!—消息资源文件路径-->

  <message-resources parameter="com.yourcompany.struts.ApplicationResources" />
</struts-config>
```

下面的配置都逐一讲解各个元素的具体含义。

下面的配置将围绕一个实例展开，这个例子的背景是大名鼎鼎的登录。用户通过 JSP 页面尝试登录系统，经过 ActionForm 的数据提交，Action 的逻辑判断，最终用户成功登陆到一个成功页面或者是转到一个友好的错误页面。

1.2.1.1 ActionForm 的配置

先通过 myeclipse 来创建一个 ActionForm 。双击 struts-config.xml 文件打开图形化编辑器

在空白处右击，在弹出的快捷菜单中选择 new|Form 命令，将出现如图 1-4 的界面。

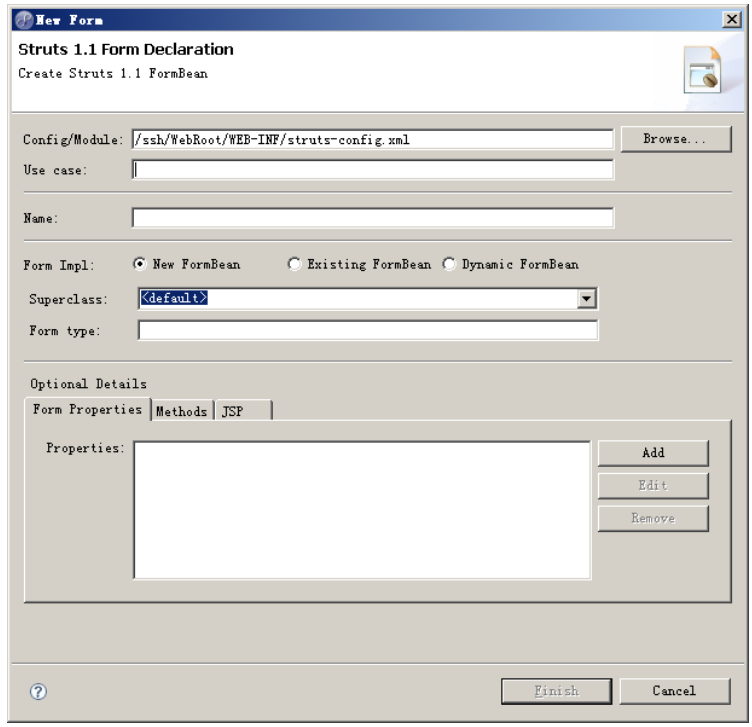


图 1-4 新建 ActionForm 向导

填写 usecase 并添加两个属性 usename 和 password 其它的保持默认或者会默认生成，如图 1-5 所示。

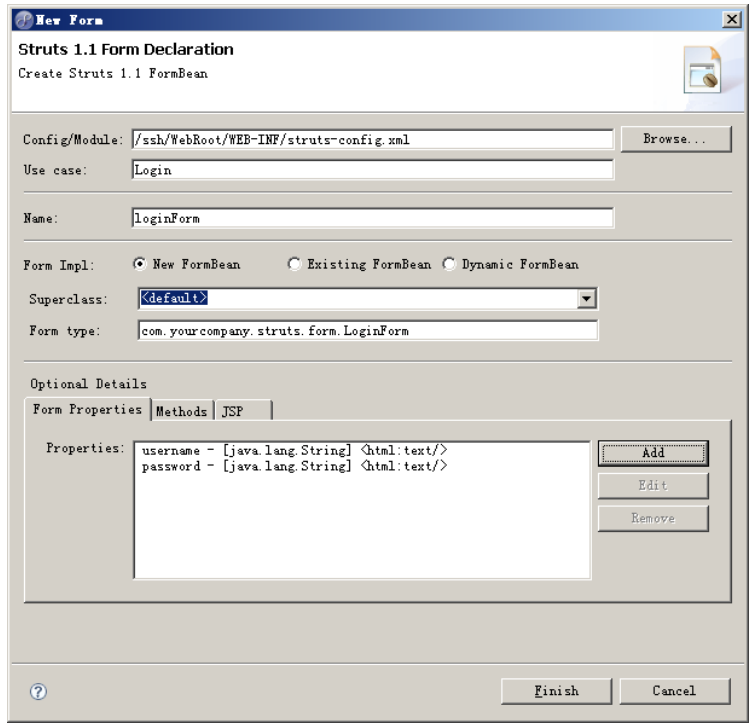


图 1-5 新建 ActionForm 完成

单击 Finish 按钮后，web 应用程序就会发生两个变化，一个是添加了一个 Java Class，代码如下：

该 Form 类的对象存储了从页面提交的用户名和密码信息，并可进行验证。

```
/*
 * Generated by MyEclipse Struts
 * Template path: templates/java/JavaClass.vtl
 */
package com.yourcompany.struts.form;

import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

/**
 * MyEclipse Struts
 * Creation date: 09-10-2007
 *
 * XDoclet definition:
 * @struts.form name="loginForm"
 */
//默认继承 ActionForm
public class LoginForm extends ActionForm {
```

```
/*
 * Generated fields
 */
   //密码
/** password property */
private String password;
 //用户名
/** username property */
private String username;

/*
 * Generated Methods
 */

/**
 * Method validate
 * @param mapping
 * @param request
 * @return ActionErrors
 */
//验证方法，它是否执行取决于 对应 action 中的 validate 属性 ，如果为 true 则执行，
//否则不执行。
public ActionErrors validate(ActionMapping mapping,
            HttpServletRequest request) {
    // TODO Auto-generated method stub
//可以在这里添加验证代码
    return null;
}

/**
 * Method reset
 * @param mapping
 * @param request
 */
//重置方法，它在 validate 之前运行，一般用于数据类型转化
public void reset(ActionMapping mapping, HttpServletRequest request) {
    // TODO Auto-generated method stub
}

/**
 * Returns the password.
 * @return String
 */
public String getPassword() {
    return password;
```

```
    }

    /**
     * Set the password.
     * @param password The password to set
     */
    public void setPassword(String password) {
        this.password = password;
    }

    /**
     * Returns the username.
     * @return String
     */
    public String getUsername() {
        return username;
    }

    /**
     * Set the username.
     * @param username The username to set
     */
    public void setUsername(String username) {
        this.username = username;
    }
}
```

另一个变化就是修改了 struts-config.xml 文件，修改后的文件内容如下：

```
<struts-config>
  <data-sources />
  <form-beans >
<!—所有的 ActionForm 都需要在这里配置-->
    <form-bean name="loginForm" type="com.yourcompany.struts.form.LoginForm" />
  </form-beans>

  <global-exceptions />
  <global-forwards />
  <action-mappings />
  <message-resources parameter="com.yourcompany.struts.ApplicationResources" />
</struts-config>
```

通过对比发现新的 struts-config.xml 文件中 <form-beans>元素的内容不再是空了，而是多了一项

<form-bean name="loginForm" type="com.yourcompany.struts.form.LoginForm" />

事实上，每个 ActionForm 都对应一个 from-bean 元素，这个元素描述了这个 Form

的名称和类路径信息。以便在其它地方引用的时候可以找到这个 ActionForm 的具体位置，同时体现了一种简单隐藏复杂的思想：给一个复杂的事务命上一个简单的名称。

1.2.1.2 Action 的配置

当用户登录后 ActionForm 将用户名和密码信息包装后提交一个 Action。在 myeclipse 中这样配置 Action，先用新建 ActionForm 的办法新建一个 Action，如图 1-6 所示。

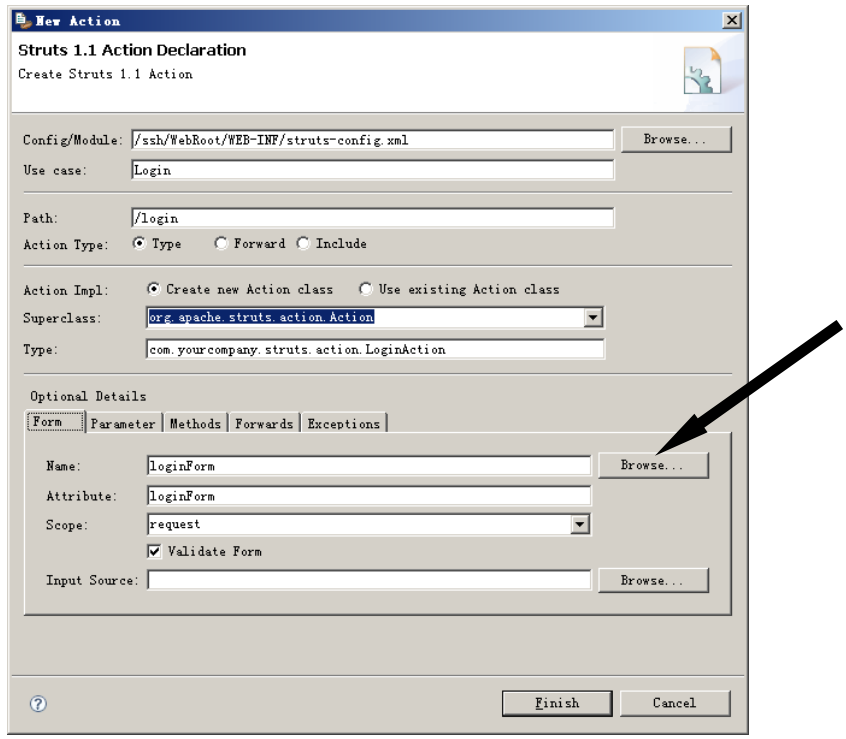


图 1-6 新建 Action

将刚才新建的 ActionForm 在 Form 标签页中引用进来，这样这个将要建立的 Login Action 与 loginForm 就扯上了关系。单击 Finish 按钮后整个应用也会发生两个变化，一个是新添加了一个 Java Class 内容如下：

```
/*
 * Generated by MyEclipse Struts
 * Template path: templates/java/JavaClass.vtl
 */
package com.yourcompany.struts.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
```

```
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.yourcompany.struts.form.LoginForm;

/**
 * MyEclipse Struts
 * Creation date: 09-10-2007
 *
 * XDoclet definition:
 * @struts.action path="/login" name="loginForm" scope="request" validate="true"
 */
//默认继承 Action
public class LoginAction extends Action {
 /*
  * Generated Methods
  */

 /**
  * Method execute
  * @param mapping
  * @param form
  * @param request
  * @param response
  * @return ActionForward
  */
//执行控制的方法。
 public ActionForward execute(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response) {
        LoginForm loginForm = (LoginForm) form;
        return null;
 }
}
```

可以通过代码看到 execute 方法中添加了 LoginForm 的相关代码，也就是说这个 Action 获得了 LoginForm 的一个实例，通过这个 JavaBean 的 getter 方法就可以将用户名和密码取出来，后面的事情就是进行验证等逻辑操作了。

另一个变化就是在 struts-config.xml 中填加了如下内容：

```
   <action-mappings >
<action
<!—属性值-->
        attribute="loginForm"
<!— ActionForm 的名称-->
        name="loginForm"
<!—//通过 login.do 可以访问到此 Action-->
        path="/login"
```

```
    <!—ActionForm 的生命周期，而不是 Action -->
          scope="request"
    <!—Action 的类路径-->
           type="com.yourcompany.struts.action.LoginAction" />
      </action-mappings>
```

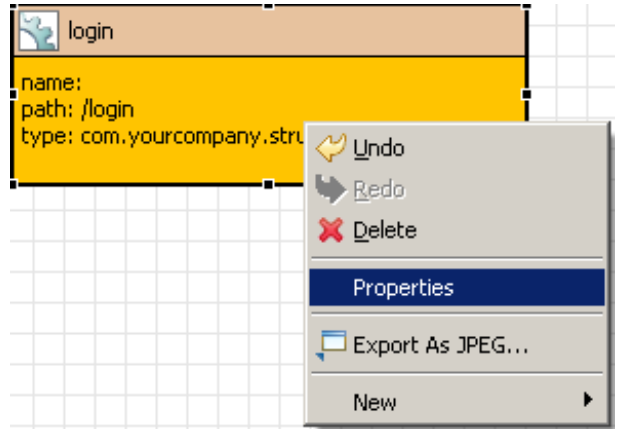
类似于 ActionForm，每个 Action 都会对应一个 action 元素，各属性的含义是：

- attribute：与此 Action 关联的 ActionForm 在 request/session 内 key.值 通过它们的 getAttribute()方法可以得到它。这个名称不一定要是前面出现过的 ActionForm 的名称，虽然默认如此。
- name：它不是 Action 的名称，而是它所关联的 ActionForm 的名称，且这个名称必须实在<form-bean>中声明过了的。
- path：ActionServlet 如何找到这个 Action 的呢？就是通过这个 path。它代表这个 Action 访问路径
- scope：此 Action 所关联的 ActionForm 的存在范围。它是 request 或者 session.
- type：代表当前 Action 的类路径。

1.2.1.3 Forward 的配置

Forward 有两种，局部的和全局的，它都是相对于 Action 而言。配置局部的 Forward 只要修改下某个具体的 Action 的配置。仍以上面的 LoginAction 为例，这里我们为它添加一个 Forward。

做法是进入 struts-config.xml 的图形编辑界面，右击刚才建立的 LoginAction，在快捷菜单里选择 property 命令，进入到开始创建时候的界面，在下面选择 forward 标签页，单击“Add”按钮，为它添加一个 Forwad 如图 1-7 所示。



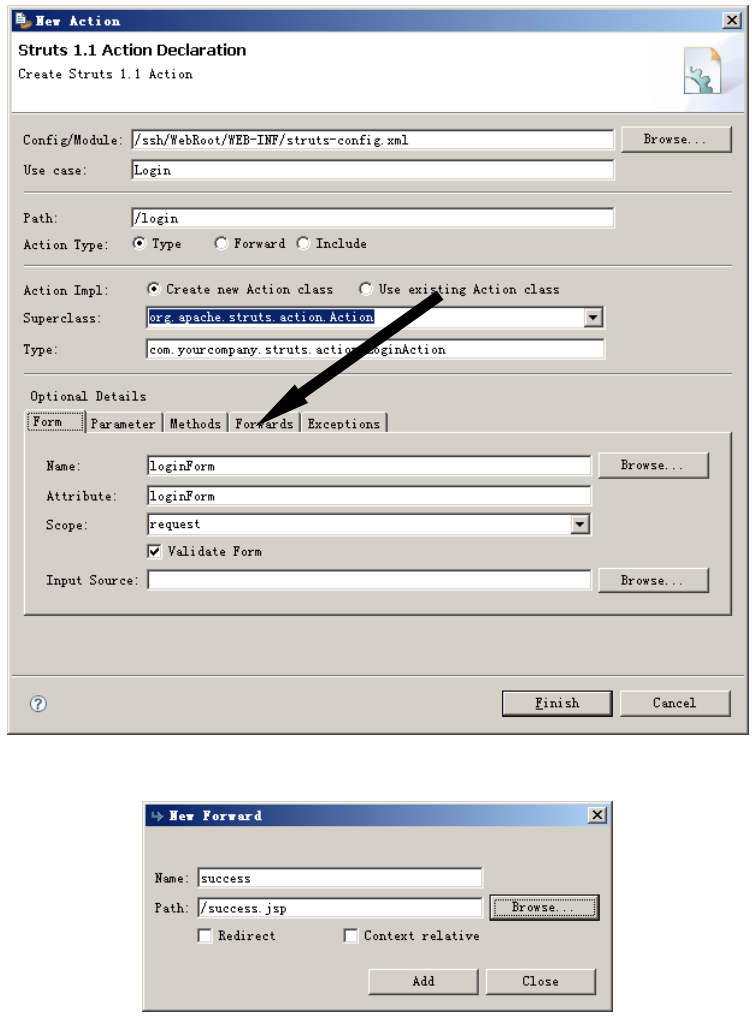


图 1-7 新建 Forward

确认添加后，发现 struts-config.xml 文件 loginAction 的相关部分发生了如下的变化：

```
<action
        attribute="loginForm"
        name="loginForm"
        path="/login"
        scope="request"
        type="com.yourcompany.struts.action.LoginAction">
        <forward name="success" path="/success.jsp" />
    </action>
```

可以看出，内容多了一行<forward name="success" path="/success.jsp" /> 其含义是自明的。

这个 forward 只能被 loginAction 使用，那么如何配置一个全局的 Forwad 呢。通过新建 Action 或者 ActionForm 的办法，新建一个 Forward，如图 1-8 所示。

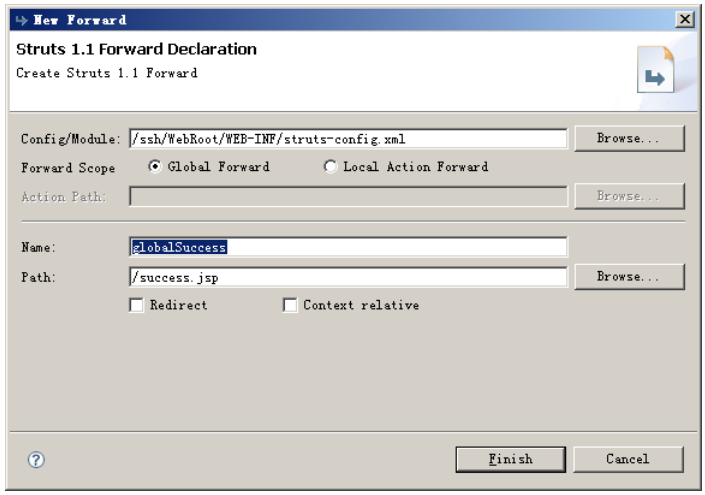


图 1-8 新建一个全局 Forward

单击 Finish 按钮后查看 struts-config.xml 文件发现新添加了：

```
<global-forwards >
<!—配置一个全局 forward →
<forward name="globalSuccess" path="/success.jsp" />

</global-forwards>
```

这样配置的 Forward 不属于任何一个 Action，可以被所有的 Action 使用。所以是全局的。

综上可见，Forward 的配置是很简单的。

1.2.1.4 struts-config.xml 的其它配置

上面几小节介绍的都是 struts 配置中的常用元素，除此之外，还有一些非常用但在某些场合下又非常有用的元素：

添加过渡引言

- <data-sourse>：配置数据源
- <global-execptions>：全局异常
- <controller>：定义控制配置类
- <message-resources>：消息资源配置
- <plug-in>：插件配置

1.3 Struts 的标签库

在了解完 struts 的开发流程后，本节将开始学习 struts 的标签技术，有人说“struts 就是一些标签”，这种说法虽然欠严谨，但也可以看出标签在 struts 中的地位。本节不再详述自定义标签技术细节，那是 JSP 所关注的内容。而是学习 Struts 中标签的具体

用法。Struts 的主要三个标签库：HTML 标签库，logic 标签库，bean 标签库。

1.1.1 html 标签库

这个标签库主要用于显示，它基本上只是对 HTML 标签进行简单封装，使得用户使用更加方便，同时可以和 Struts 其它组件很好的结合在一起，比如<html:form>和 ActionForm 的关系。

1.1.1.1 <html:html>标签

这是最简单的标签，它的作用仅仅是在使用标签处产生 一个<html>元素，可能还包含一个 locale 或者 lang 属性。这个属性取决于 struts 的版本，struts1.1 为 locale 属性，struts1.2 为 lang。

用法分别为：

```
<html:html locale=”true”>
<html:html lang=”true”>
```

使用 myeclipse 创建一个 struts1.1 模板的 JSP 后默认代码如下：

```
<%@ page language="java" pageEncoding="GB18030"%>

<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-tiles" prefix="tiles" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-template" prefix="template" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-nested" prefix="nested" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html locale="true">
  <head>
    <html:base />

    <title>tag.jsp</title>

 <meta http-equiv="pragma" content="no-cache">
 <meta http-equiv="cache-control" content="no-cache">
 <meta http-equiv="expires" content="0">
 <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
 <meta http-equiv="description" content="This is my page">
 <!--
 <link rel="stylesheet" type="text/css" href="styles.css">
 -->

  </head>

  <body>
```

```
        这是一个 Struts 页面  This a struts page. <br>
      </body>
    </html:html>
```

黑色加粗部分就是自动生成的<html:html>标签。同时我们也可以了解到 struts1.1 支持的标签库列表。

1.1.1.2 <html:base>标签

在上面的例子中还看到了一个<html:base>标签。它的用法比较简单。就像使用 html 的 base 标签一样。访问上面的 jsp。查看客户端代码：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="zh">
  <head>
    <base href="http://localhost:9090/ssh/tag.jsp">

    <title>tag.jsp</title>

 <meta http-equiv="pragma" content="no-cache">
 <meta http-equiv="cache-control" content="no-cache">
 <meta http-equiv="expires" content="0">
 <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
 <meta http-equiv="description" content="This is my page">
 <!--
 <link rel="stylesheet" type="text/css" href="styles.css">
 -->

  </head>

  <body>
    这是一个 Struts 页面  <br>
  </body>
</html>
```

可见 <html:base/>标签的用法是在其出现的位置上添加一句 base 语句，并给其 href 属性赋值。

1.1.1.3 <html:link>标签

它本质上是在生成一个<a>标签，下面通过不同的属性介绍它的作用。

连接一个外部的 URL 地址，用法非常简单。如下：

```
<html:link href=”http://www.oyxf.cn”>
本书作者博客
</html:link>
```

从这个意义上而言，它完全等价于

```
<a href=”http://www.oyxf.cn”>本书作者博客</a>
```

如果是连接一个内部的文件，可以使用 page 属性，用法如下：

```
<html:link page=”/welcome.jsp”>首页</html:link>
```

如果需要添加参数可以直接在后面以?开始进行添加和<a>的用法一致。如

```
<html:link page=”/welcome.jsp?index=1”>首页</html:link>
```

如果说<html:link>仅仅是具有上面一些鸡毛蒜皮的能力，那么我们就应该对他的存在表示质疑了。还好，他没有让我们失望的是它的功能不仅仅如此。如果在 struts-config.xml 文件中定义了一个全局的 Forward。比如：

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

那么可以通过 <html:link> 使用到这个 forward 地址：

```
<html:link forward=”index” >首页</html:link>
```

效果等价于超级链接到这个页面。它的方便在于不用关心这个页面的可能长长的绝对路径。

再来看<html:link>的另一个功能。

假如在第一个 jsp 文件中有如下内容：

```
<%
String test="this is a test string";
request.setAttribute("teststr",test);
%>
```

那么在第二个 jsp 文件如果要将第一个 jsp 文件中的 test 值做为参数放在<a>标签的后面一般的做法是（前提是第一个 jsp 对第二个 jsp 文件有 http 请求操作比如 forward）

```
<%
String teststr=(String)request.getAttribute(“teststr”);
%>
<a href=”somepage.jsp?param=<%=teststr%>”></a>
```

为了减少代码量。<html:link>可以直引用，用法如下：

```
<html:link forward="index" paramId="param" paramName="teststr">首页</html:link>
```

查看客户端代码：

```
<a href="/ssh/index.jsp?param=this+is+a+test+string">首页</a>
```

1.1.1.4<html:img>标签

这个标签用来显示图片，用法简单，如下：

```
<html:img page=”/button.jpg”/>
```

1.1.1.5<html:form>标签

这个在前面已经接触过，现在进行详细讲解，并采用前面的代码 form.jsp：

```
<%@ page language="java" pageEncoding="GB18030"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"%>

<html>
 <head>
```

```
        <title>JSP for FutureForm form</title>
    </head>
    <body>
        <html:form action="/future">
    <!—姓名-- >
     Name:<html:text property="name"/><html:errors property="name"/><br/>
    <!— 年龄 -- >
    age : <html:text property="age"/><html:errors property="age"/><br/>
            <html:submit/><html:cancel/>
        </html:form>
    </body>
    </html>
```

这段页面代码中使用到了<html>标签库中的<html:form>标签做为表单元素，元素包含姓名和年龄两个字段，它对应相应的 ActionForm 中的两个属性。<html:form>标记可以生成表单元素，但和普通的<form>标签有所不同。<form>的含义比较单纯，它的 action 属性只是一个 servlet 或者 jsp。但<html:form>的含义会比较复杂一些。在它的属性约束上可以得到体现：

- action 属性的书写形式往往以.do 结尾
- action 必须为 struts-config.xml 中已经配置好了的 action 名称
- 这个表单都对应一个 ActionForm。所以表单的填充域标签属性都不能随意。

用法：

```
    <html:form action="/future">
```

Action 属性往往对应一个 Action 类的 path 路径，它在 Struts-config.xml 中配置。

1.1.1.6<html:text>标签

它用来生成一个文本框，往往出现在<html:form>中，如果不在<html:form>中会报错。错误类似于如下代码：

```
    javax.servlet.ServletException: Cannot find bean org.apache.struts.taglib.html.BEAN in any
scope
     org.apache.jasper.runtime.PageContextImpl.doHandlePageException(PageContextImpl.java:82
5)
     org.apache.jasper.runtime.PageContextImpl.handlePageException(PageContextImpl.java:758)
     org.apache.jsp.MyJsp_jsp._jspService(MyJsp_jsp.java:76)
     org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:94)
     javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
     org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:324)
     org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:292)
     org.apache.jasper.servlet.JspServlet.service(JspServlet.java:236)
     javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
```

其属性值与<html:form>对应的 ActionForm 属性值一致。

其用法如下：

```
    <html:form action="future.do">
    <html:text property="name"/> <!—姓名-- >
```

```
    </html:password>   <!— 密码-- >
    <html:form/>
```

<html:text>它必须出现在<html:form>标签体里面。

1.1.1.7 <html:password>标签

它用来在页面生成一个密码输入框，在这个框中输入的字符都以*显示。这个标签必须与<html:form>配合使用，也就是说它必须出现在<html:form>标签体里面。否则会出错。它的属性值必须是对应的 ActionForm 中存在的属性名称。

用法如下：

```
    <html:form action="future.do">
    <!—这里的 age 必须是 对应的 ActionForm 中存在的属性 -- >
     <html:password property="age">
     </html:password>
```

<html:password>标签和<html:text>的区别是在<html:password>输入的字符不会被显示，一般以黑点呈现。

1.1.1.8 <html:textarea>标签

它的用法和<textarea>没多大的区别，在页面上显示一个多行的文本输入框。约束和<html:text>等一样。

用法如下：

```
    <html:form action="future.do">
     <html:textarea property="name" rows="2"></html:textarea>
     </html:password>
```

<html:textarea>标签的 rows 属性表示这个文本框的高度，它以行为单位，如果要设置宽度，可以用 cols 属性。

1.1.1.9 <html:hidden>标签

它是隐藏填字段的 struts 写法，但有所扩展。这个标签可以在<html:form>标签体外存在。

用法如下：

```
    <html:hidden property="test" value="I am hidden"/>
```

等价于：

```
    <input type=hidden name=”test” value=”I am hidden”>
```

不过，<html:hidden>具有一个 write 属性，当这个属性的值为 true 的时候，这个隐藏的值就会被显示出来，如：

```
    <html:hidden property="test" value="I am hidden" write=”true”/>
```

在客户端页面上将输出 I am hidden，效果如图 1-9 所示：

图 1-9 输出 I am hidden

1.1.1.10 <html:submit>标签

这个标签没有什么扩展的内容。

用法如下：

```
<html:submit/>
```

等价于：

```
<input type=submit />
```

1.1.1.11<html:reset>标签

它用于复位表单中的各输入内容。

用法如下：

```
<html:reset/>
```

它必须出现<html:form>表单标签体内部才有意义，在页面以按钮形式呈现，当点击这个这个按钮时，表单中所有已经输入的内容都会被置空，其效果等价于<input type=reset />

1.1.1.12 <html:cancel>标签

这是一个取消按钮标签，它的功能非常类似于浏览器工具栏中的取消按钮，它用在<html:form>中，当表单提交后，用户突然想取消提交，那么就可以使用这个标签。

这个标签的本质一个 submit 类型的标签，提交地址为表单 action 属性地址。提交后运行了一段 JavaScript 脚本，将一个称为 bCancel 的变量赋值为 true，使得 isCanceled() 方法为 true.

要使得这个 cancel 有效，必须在 Action 中捕获。代码类似于：

```
/**
  * Method execute
  * @param mapping
  * @param form
  * @param request
  * @param response
  * @return ActionForward
  */
 public ActionForward execute(ActionMapping mapping, ActionForm form,
```

```
                HttpServletRequest request, HttpServletResponse response) {
        //如果被取消， isCancelled 方法是继承父类的。
        if(isCancelled(request)){
            return mapping.findForward("index");
          //返回主页
        }
          //否则可以在这里实现一些控制逻辑代码
      //do something here

    }
```

isCancelled 方法继承于 Action，它用来判断用户时候取消了 Http 请求。

1.1.1.13 html 标签库中的其它标签

Struts 的 html 标签库本质上是对标准 HTML 标签库的一个再度封装，所以 Struts 的 html 标签库基本囊括了 HTML 标签库中的常用标签，如下面的：

添加过渡引言

- <html:checkbox>：生成选择框
- <html:multibox>：生成复选框
- <html:radio>：生成单选框
- <html:select> <html:option> <html:options> <html:options>：生成选择列表
- <html:file>：用于文件上传
- <html:errors>：这个标签用来显示错误信息。其实现机制较复杂，它和 ActionErrors，消息资源文件结合一起生成错误信息。
- <html:messages>标签：用于显示消息类似于<html:errors>

1.1.2 logic 标签库

这个标签库中的标签主要是用来实现一些控制逻辑，避免在 JSP 页面上书写过多的 java 代码，让非开发人员也能容易的做开发人员做的事情。

1.1.2.1 <logic:equal>标签

这个标签用来判断一个变量和指定的常量是否相等。

用法如下：

```
    <% pageContext.setAttribute("str","123"); %>
  <logic:equal name="str" value="123">
  相等
  </logic:equal>
```

它判断 str 是否和字符串“123”相等它的效果等价于：

```
    <% pageContext.setAttribute("str","123");
      String str=(String)pageContext.getAttribute("str");
      if(str.equals("123")){//更好的写法是: “123”.equals(str);
      out.print("相等");
```

```
        }
    %>
```

这样避免了在 JSP 中书写具有逻辑意义的 Java 代码。<logic:equal>的 name 属性不仅接受 pageContext 同时也接受 request 对象的 Attribute 等。<logic:equal>还有几个基本属性，header 属性的含义和用法非常简单，下面通过一个例子进行描述：

```
<logic:equal header="host" value="localhost:9090">服务器地址为 localhost:9090</logic:equal>
```

可见它就是判断主机 ip:port 地址是否和指定的地址相等。<logic:equal>的 parameter 属性是一个有用的属性，它可以获取 http 请求参数。用法如下所示：

```
<a href="tag.jsp?param=haha">click me</a>
<logic:equal parameter="param" value="haha">haha</logic:equal>
```

这端代码用来判断 param 是否与"haha"相等，而这个过程如果采用传统方式的话，需要显示的使用 request 对象调用它的 getter 方法。这段代码效果等价于：

```
<% String param=request.getParameter("param");
  out.print(param);
  %>
```

除此，cookie 属性可以判断 cookie 值是否与一个指定的值相等。

既然有相等，那一定还有不相等，大于，大于等于，小于，小于等于逻辑标签。它们分别是<logic:notEqual> <logic:greaterThan> <logic:greaterEqual> <logic:lessThan> <logic:lessEqual> 它们具有同样的属性，类似的用法。

1.1.2.2 <logic:iterate>标签

这是一个旨在取代 for 或者 while 循环语句的标签。它能进行 Array，Collection，Map 的迭代。用法如下：

```
<% String array[]={"a","b","c","d","e"};
  pageContext.setAttribute("testArray",array);
  %>
<logic:iterate id="testArray" name="testArray">
  <bean:write name="testArray"/>
  </logic:iterate>
```

代码逻辑是依次输出 array 数组的内容，输出结果为 a b c d e 如图 1-26 所示。

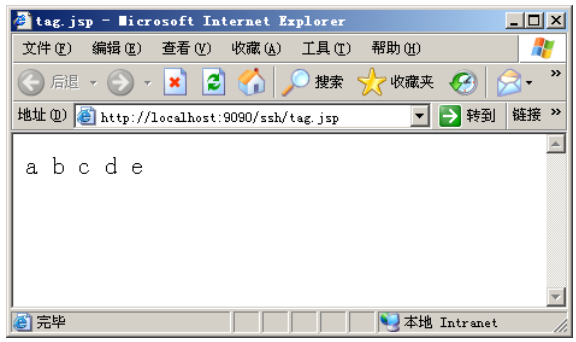


图 1-26 输出 a b c d e

效果等价于：

```
<% String array[]={"a","b","c","d","e"};
```

```
<!—pageContext 是一个内置的 jsp 对象，在自定义标签技术中使用的比较多 -->
pageContext.setAttribute("testArray",array);
  String testArray[]=(String[])pageContext.getAttribute("testArray");
  for(int i=0;i<testArray.length;i++){
  out.print(testArray[i]+" ");
  }
  %>
```

这段代码也是依次将 array 数组的内容输出，只是使用简单的 for 方式。当然<logic:iterate>标签的内在实现比这要复杂些。这里使用到了<bean>标签库，这将在下节中详述。

1.1.2.3 <logic:match>标签

这个标签的作用是用来判断字符串是否匹配给定的字符串。用法：

```
<%
String test="test";
  pageContext.setAttribute("test",test);
  %>
<logic:match name="test" value="test">相等</logic:match>
```

代码判断 test 对象的值是否等于”test”. 既然如此，那和<logic:equal>又有什么区别呢，区别是有的，因为<logic:match>还具有另外一个属性就是 location，它的值为 end 或者 start。是用来判断字符串是否以给定的字符串开头或者结尾。

<logic:notmatch>的用法与<logic:match>同出一辙。

1.1.2.4 <logic:empty>标签

这个标签的作用是用来判断指定的字符串是否为空（内容为空或者 null）用法非常简单，代码如下：

```
<%
String test="";
//String test=null;
  pageContext.setAttribute("test",test);
  %>
<logic:empty name="test">test is empty</logic:empty>
```

代码段判断 test 值时候为 null。

1.1.2.5 <logic:present>标签

上面的标签都是用来比较两个对象的值。这个标签用来判断对象是否存在。用法如下：

```
<%
String test="test";
  pageContext.setAttribute("test",test);
  %>
```

```
<logic:present name="test"> test exist!</logic:present>
```

上面代码用来判断 test 对象时候存在。<logic:present>具有的一些其它的属性，包括 header，parameter，cookie,property，这里不一一介绍。

1.1.2.6 <logic:forward>标签

这个标签是用来转发页面。类似于 jsp 标签<jsp:forward>，但它的优点是书写更简洁。

用法如下：

```
<logic:forward name="index"/>
```

Name 的值为一个在 struts-config.xml 中定义过的全局 forward。

此外还有一个<logic:redirect>标签，它和<logic:forward>的效果基本一样。

1.1.3 bean 标签库

顾名思义，bean 标签库的作用是用来访问存在 web 应用中各种 JavaBean。

1.1.1.1 <bean:header>标签

在 logic 里面已经接触到了 header 这个关键词，事实上，只要 header，struts 一般都用它来处理 HTTP 请求头。HTTP 请求头是由键值对构成，它和 JavaBean 有着天然的联系，对它封装后，就可以进行 getter 操作。假如要得到客户端的浏览器信息，那么可以使用下面的办法：

```
<bean:header id="explorer" name="user-agent"/>
<!— 浏览器信息 它存在 http 消息头中 -->
explorer you used is: <bean:write name="explorer" />
```

代码说明：User-agent 是 Http 协议头中的代表客户端信息的一个属性。上面这段代码的实现过程并不复杂，也不难理解:<bean:header>的作用相当于

```
…
pageContext.setAttribute(“explorer”,user-agent);
String explorer=user-agent.
```

而<bean:write>的作用就相当于：

```
pageContext.getAttribute(“explorer”);
```

运行后的结果如图 1-26 所示：

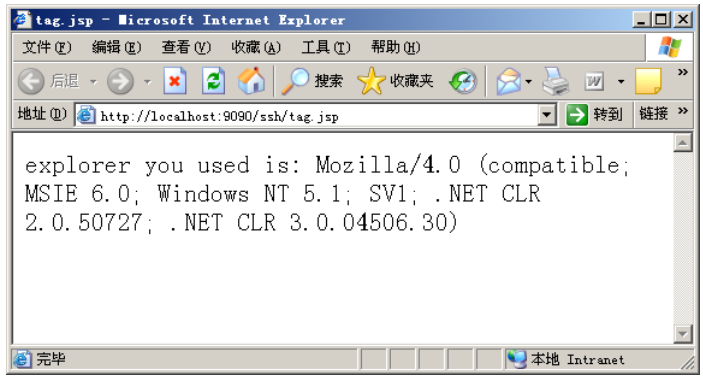


图 1-26 运行结果

1.1.1.2<bean:parameter>与<bean:cookie>标签

学了<logic:parameter>标签后，读者应该可以举一反三的猜测到这个标签与 HTTP 请求中的参数有关。其用法如下：

```
<html:link page="/tag.jsp?param1=xixi&param2=haha" >click me</html:link>
<bean:parameter id="p1" name="param1" value=""/>
参数 1： is :<bean:write name="p1"/>
<bean:parameter id="p2" name="param2" value=""/>
参数 2： is :<bean:write name="p2"/>
```

点击 cick me 后，客户端将对本页进行一次 HTTP 请求，且带有两个参数

使用<bean:paramter>标签将第一个参数内容添加到 pageContext 的 Attribute 中。命名为 p1，同时 value 的值必须显示的给出，如果 param1 并不是 HTTP 请求的参数，那么就会 value 做为默认参数，否则会出现异常。然后使用<bean:write>调用 pageContext 的 getAttribute 方法将 p1 取出。

可见，<bean>标签库都利用了 pageContext 这个 JavaBean 做为载体进行数据 setter 和 getter.

1.1.1.3<bean:cookie>标签

有了上面的介绍和基础，读者应该不难推测到这个标签的含义，甚至用法。Cookie 存储的是键值对，所以也可以将其内容使用 JavaBean 进行 setter 和 getter 操作。用法与<logic:parameter>类似：

它也具有 id,name,value 三个属性，且含义和用法没什么不同。

1.1.1.4<bean:page>标签

JSP 具有 9 个内置对象：request，response，pageContext，session，application，out，config，page 和 exception。这个标签的作用的命名可能会让人产生误会，认为这是一个针对 page 对象的标签，事实上，这太低估了他的作用。上面 9 个内置对象都在它的操作范围之类。用法简单，下面代码无需讲解就可明白其简单的含义，因为我们已经经历了前几个标签的学习：

```
<!—通过 Bean 取得 session 值 -->
<bean:page id="sess" property="session"/>
<%
out.print("session:id:"+sess.getId()+"<br>");
 %>
```

<bean:page>中的 Id 和前面介绍的标签 id 含义一样，property 是内置对象的名称。

1.1.1.5<bean:include>标签

这个标签和<jsp:include>标签的作用非常类似，都是将一个外部的资源（一般是一个 JSP 文件）包含到本页面。但区别在于。<bean:include>只是把这个外部资源放置在 JavaBean 中。如果要让它显示还需要借助<bean:write>标签。而<jsp:include>仅仅是将外部资源直接显示在标签位置。

<bean:include>的用法如下：

```
<bean:include id="index" forward="index" />
<% out.print(index); %>
<bean:write name="index"/>
this is tag.jsp
```

Index.jsp 页面内容仅含有一句 this is index page 语句，如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="GB18030"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+
"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>My JSP 'index.jsp' starting page</title>

 <meta http-equiv="pragma" content="no-cache">
 <meta http-equiv="cache-control" content="no-cache">
 <meta http-equiv="expires" content="0">
 <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
 <meta http-equiv="description" content="This is my page">
 <!--
 <link rel="stylesheet" type="text/css" href="styles.css">
 -->

  </head>

  <body>
```

```
    <!— 首页 -->
        This is index page. <br>
    </body>
</html>
```

运行 tag.jsp 后，结果如图 1-27 所示：

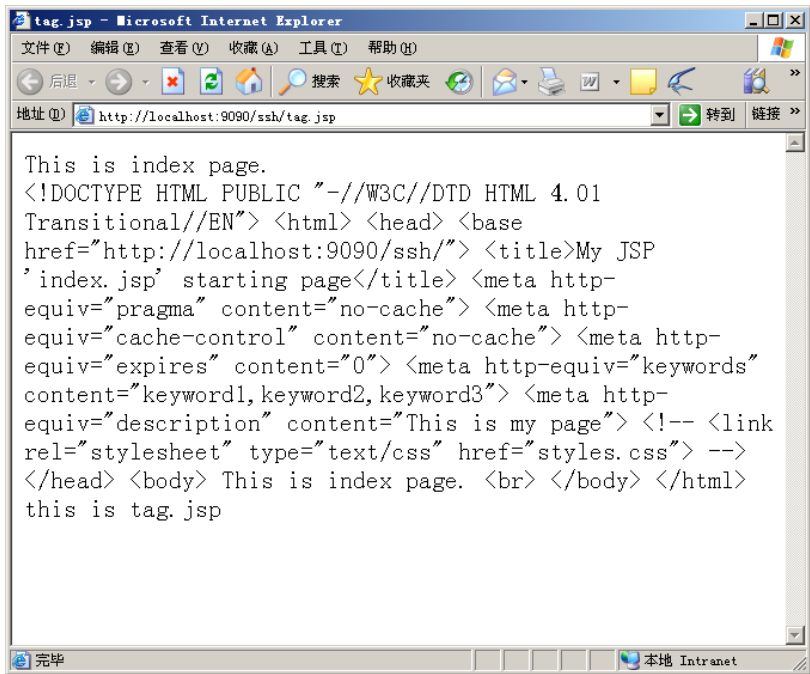


图 1-27　运行 tag.jsp

从这个结果，可以发现<bean:write>仅仅是将请求 index.jsp 后访问的 html 文本信息显示出来。但并不解析它们。这我们提供了一种显示<html>标签元素本身的办法。如果要解析这些标签，需要使用 out.print()方法。<bean:include>标签具有四个属性

- id：对象引用名称，Map 中的 key. 与上面的标签 id 含义一样。
- page：本地的一个资源。如/index.jsp
- forward：存在 struts-config.xml 中的一个全局 forward。
- href：一个绝对的地址，一般用于引用一个外部的资源如：http://www.google.com

1.1.1.6<bean:message>标签

当用户需要将资源文件中的内容显示出来的时候，就可以使用到这个标签。最简单的用法是：

```
    <bean:message key="nullnameerror"/>
```

key 的值 nullnameerror 需要在资源文件中定义，资源文件内容：

```
    # Resources for parameter 'com.yourcompany.struts.ApplicationResources'
    # Project ssh
```

```
# 属性文件的注释以 '# '开始
nullnameerror=姓名不能为空
nullageerror=年龄不能为空
illegalname=输入的姓名不合法
illegalage= 输入的年龄不合法
```

运行标签实例所在的 tag.jsp 后，在标签位置显示如图 1-28 所示。

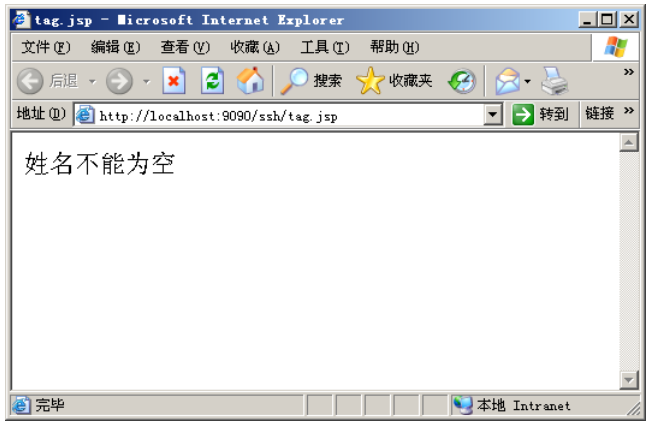


图 1-28 姓名不能为空

当然，<bean:message>还有一些其它的功能。如显示一个 JavaBean 的内容。

1.1.1.7<bean:resource>标签

这个标签用来定义一些资源，而这些资源会在后面使用到。它有三个属性：

- Id：和其它标签一样
- Name：指定一个资源路径
- Input：指定类型。缺省情况下为 java.lang.String 类型。如果值为 yes,那么就是 java.io.InputStream 类型。

以下是一段示例代码：

```
<bean:resource id="index" name="/index.jsp"/>
<bean:write name="index"/>
```

代码目的在于将 index.jsp 的内容显示出来，结果显示如图 1-29 所示。

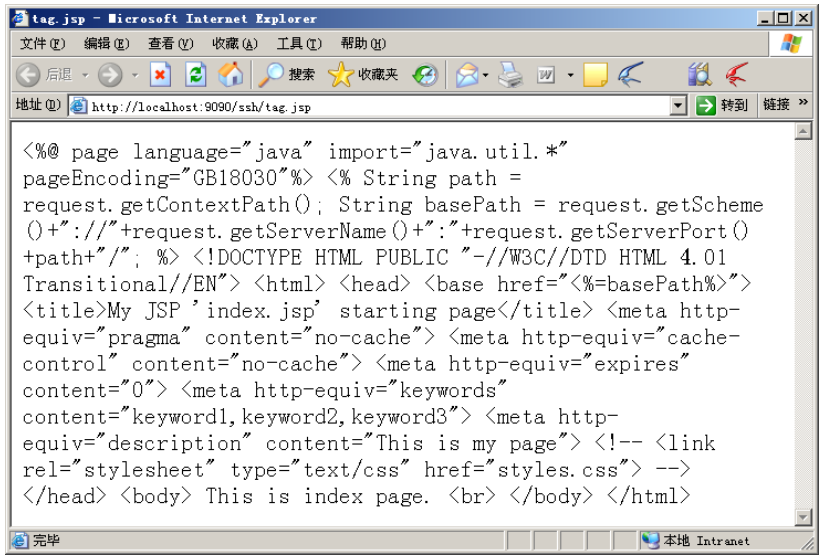


图 1-29 客户端显示信息

其效果类似于前面介绍的<bean:include>。

1.1.1.8<bean:struts>标签

前面的标签基本上从其名字上可以猜测其用途，而这个似乎没什么信息量，事实上，它是用来获取 Struts 这个框架中的一些对象值，如 ActionForm.ActionForward，ActionErrors.等等。<bean:struts>包含四个属性

- id ：和前面提到的一样，它就好像赋值语句中的左值存在着。即使在这里也没有多大的创新。
- formBean：它对应 ActionFormBean 和 struts-config.xml 中的<form-bean>元素。
- forward：对应 ActionForward 对象和 struts-config.xml 中<global-forward>元素中的<forward>也就是说，它只针对全局 forward。
- mapping：对应 ActionMapping 对象和 struts-config.xml 中的<action>元素。

用法示例：

```
<bean:struts id="future" formBean="futureForm"/>
    <bean:write name="future"/>
```

Struts-config.xml 中关于 futureForm 的描述如下：

```
<form-beans >
  <form-bean name="futureForm" type="com.yourcompany.struts.form.FutureForm" />
   </form-beans>
```

运行结果如图 1-30。

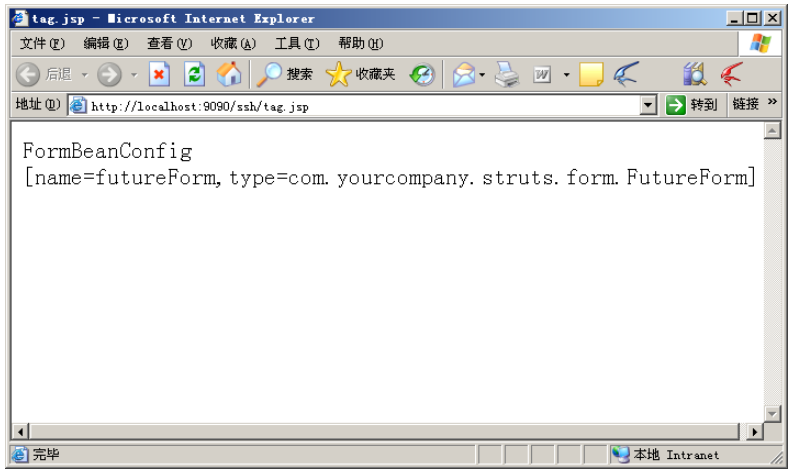


图 1-30 输出 FuturForm 信息

1.1.1.9<bean:define>标签

这个标签用来定义一个变量。最简单的用法如下：

```
<bean:define id="teststr" value="hello,world"></bean:define>
<bean:write name="teststr"/>
```

运行后客户端结果如图 1-31 所示。

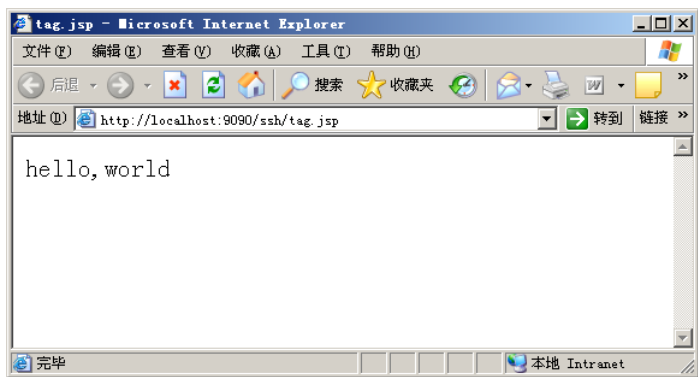


图 1-31 输出 hello,world

上面的标签作用等价于：

```
<% String teststr=”hello,world”;
    out.print(teststr);
%>
```

1.1.1.10<bean:size>标签

它用来获取 collection 对象，Map 对象或者 Array 对象的元素个数。用法简单如：

```
<%
List<Object> datalist=new ArrayList<Object>();
//为其添加三个对象
datalist.add(new Object());
datalist.add(new Object());
datalist.add(new Object());
```

```
pageContext.setAttribute("datalist",datalist);
 %>
<bean:size id="dataListSize" name="datalist"/>
dataListSize:<bean:write name="dataListSize"/>
```

代码中的 dataList 包含了三个对象，最后使用<bean:write>标签将 dataList 的对象个数显示了出来，显示结果如图 1-12 所示：

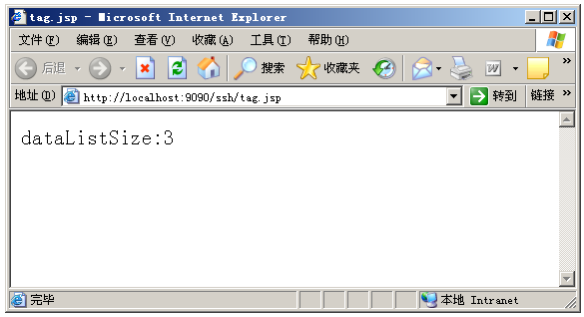


图 1-12 输出 dataListSize

1.4 Struts 的扩展方法

Struts 不仅具有定义良好的固定组件，而且具有较高的扩展性，用户通过自定义这些组件可以满足一些更复杂的需求。

1.4.1 自定义 ActionServlet

ActionServlet 在整个 Struts 应用中起到门神的作用。但这并不意味着他就神圣不可侵犯，如果它不能满足你的要求，那就扩展他。在扩展之前，先来看下 ActionServlet 的真实面貌：

```
//ActionServlet 本质上是一个 httpServlet
public class ActionServlet extends HttpServlet {


    // ----------------------------------------------------- Instance Variables
…
    /**
     * <p>The <code>RequestProcessor</code> instance we will use to process
     * all incoming requests.</p>
     *
     * @since Struts 1.1
     */
    protected RequestProcessor processor = null;
```

```
// ---------------------------------------------------- HttpServlet Methods


/**
 * <p>Process an HTTP "GET" request.</p>
 *
 * @param request The servlet request we are processing
 * @param response The servlet response we are creating
 *
 * @exception IOException if an input/output error occurs
 * @exception ServletException if a servlet exception occurs
 */
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws IOException, ServletException {

    process(request, response);

}


/**
 * <p>Process an HTTP "POST" request.</p>
 *
 * @param request The servlet request we are processing
 * @param response The servlet response we are creating
 *
 * @exception IOException if an input/output error occurs
 * @exception ServletException if a servlet exception occurs
 */
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
    throws IOException, ServletException {

    process(request, response);

}



/**
```

```
        /**
         * <p>Perform the standard request processing for this request, and create
         * the corresponding response.</p>
         *
         * @param request The servlet request we are processing
         * @param response The servlet response we are creating
         *
         * @exception IOException if an input/output error occurs
         * @exception ServletException if a servlet exception is thrown
         */
    //本质上，ActionServlet 是通过 process 这个方法工作的，在这个方法中 调用了
//RequestProcessor 对象进行核心控制，所以一般要扩展 ActionServlet 的一个有效的办法//就是扩展 RequestProcessor
        protected void process(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {

            ModuleUtils.getInstance().selectModule(request, getServletContext());
            ModuleConfig config = getModuleConfig(request);

            RequestProcessor processor = getProcessorForModule(config);
            if (processor == null) {
                processor = getRequestProcessor(config);
            }
            processor.process(request, response);

        }
    …
    }
```

这是一个普通的 servlet，它包含 doGet 和 doPost 方法，不过最终都是调用一个 process 方法，所以这个 process 方法才是整个 ActionServlet 的核心所在。通过这段代码可见 ActionServlet 实质上是通过 RequestProcessor 进行工作的，所以扩展 ActionServlet 的最直接和最必要的办法就是扩展 RequestProcessor。仍然以 ssh 这个项目为例，我们来通过自定义 RequestProcessor 来扩展 ActionServlet.在此之前，需要对 struts-config.xml 文件进行相关配置，配置方法很简单，在配置文件中添加一个 controller 元素，processorClass 属性是自定义的 Processor 类路径。如下所示：

```
    <!—控制器的配置-->
    <controller    processorClass="com.yourcompany.struts.MyRequestProcessor"></controller>” />
```

再来分析下 RequsetProcessor，可以看到它包含的几个重要方法是

```
    /**
         * <p>Return an <code>Action</code> instance that will be used to process
         * the current request, creating a new one if necessary.</p>
         *
```

```
     * @param request The servlet request we are processing
     * @param response The servlet response we are creating
     * @param mapping The mapping we are using
     *
     * @exception IOException if an input/output error occurs
     */
    //创建 Action
protected Action processActionCreate(HttpServletRequest request,
                                                HttpServletResponse response,
                                                ActionMapping mapping)
        throws IOException


    /**
     * <p>Retrieve and return the <code>ActionForm</code> associated with
     * this mapping, creating and retaining one if necessary. If there is no
     * <code>ActionForm</code> associated with this mapping, return
     * <code>null</code>.</p>
     *
     * @param request The servlet request we are processing
     * @param response The servlet response we are creating
     * @param mapping The mapping we are using
     */
//处理 ActionForm
    protected ActionForm processActionForm(HttpServletRequest request,
                                                  HttpServletResponse response,
                                                  ActionMapping mapping)
    /**
     * <p>If this request was not cancelled, and the request's
     * {@link ActionMapping} has not disabled validation, call the
     * <code>validate</code> method of the specified {@link ActionForm},
     * and forward to the input path if there were any errors.
     * Return <code>true</code> if we should continue processing,
     * or <code>false</code> if we have already forwarded control back
     * to the input form.</p>
     *
     * @param request The servlet request we are processing
     * @param response The servlet response we are creating
     * @param form The ActionForm instance we are populating
     * @param mapping The ActionMapping we are using
     *
     * @exception IOException if an input/output error occurs
     * @exception ServletException if a servlet exception occurs
     * @exception InvalidCancelException if a cancellation is attempted
     *                       without the proper action configuration
```

```
            */
        //处理验证
            protected boolean processValidate(HttpServletRequest request,
                                              HttpServletResponse response,
                                              ActionForm form,
                                              ActionMapping mapping)
```

对于代码中的这几个方法，如果在这个类的子类中覆盖它们就可以在不同的时机插入自定义的逻辑，从而实现了扩展。各个方法的运行时机分别是：

- processActionCreate 调用 Action 时执行
- processActionForm 在填充 ActionForm 时执行
- processValidate 进行校验时执行。

根据这一点，下面开发 MyRequestProcessor.代码如下：

```
package com.yourcompany.struts;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.RequestProcessor;

public class MyRequestProcessor extends RequestProcessor {

 @Override
//通过覆盖的办法进行扩展
 protected Action processActionCreate(HttpServletRequest arg0, HttpServletResponse arg1,
ActionMapping arg2) throws IOException {
        // TODO Auto-generated method stub
        System.out.println(" MyRequestProcessor:rocessActionCreate:do some thing here");
        return super.processActionCreate(arg0, arg1, arg2);
 }

 @Override
//覆盖 processActionForm 方法
 protected ActionForm processActionForm(HttpServletRequest arg0, HttpServletResponse arg1,
ActionMapping arg2) {
        // TODO Auto-generated method stub
        System.out.println(" MyRequestProcessor:processActionForm:do some thing here");
        return super.processActionForm(arg0, arg1, arg2);
 }
```

```
@Override
//覆盖 processValidate 方法
protected boolean processValidate(HttpServletRequest arg0, HttpServletResponse arg1,
ActionForm arg2, ActionMapping arg3) throws IOException, ServletException {
 // TODO Auto-generated method stub
 System.out.println("MyRequestProcessor:processValidate:do some thing here");
 return super.processValidate(arg0, arg1, arg2, arg3);
}
}
```

这段代码没有做特别的事情，而是在各个方法中输出一条测试语句用来测试所在方法是否被调用，何时被调用，在实际开发中可以使用有意义的逻辑代码替换之。这里需要特别注意的地方是每个方法都需要调用 super 方法。否则你的行为不是扩展而是改写和覆盖。这样会造成 struts 不工作。

1.4.2 实现 plugIn 接口

Struts 应用在开启时候会进行一些初始化工作，在应用销毁时会进行一些资源回收之类的工作。Struts 允许用户在这两个时间点执行一些自己的业务逻辑办法是实现 org.apache.struts.action.PlugIn 接口。例子如下：

```
package com.yourcompany.struts;

import javax.servlet.ServletException;

import org.apache.struts.action.ActionServlet;
import org.apache.struts.action.PlugIn;
import org.apache.struts.config.ModuleConfig;
//必须实现 PlugIn 接口
public class MyPlugIn implements PlugIn {
//在应用销毁时执行的方法
//如关闭 web 服务器时
 public void destroy() {
        System.out.println("MyplugIn:destroy: to do something here");

 }
//在应用开启时执行的方法
//如开启 web 服务器时
 public void init(ActionServlet arg0, ModuleConfig arg1)
                throws ServletException {
        System.out.println("MyPlugIn:init:to do something here");

 }
```

```
    }
```

这段代码的核心方法是：

- void init(ActionServlet arg0, ModuleConfig arg1)：在 struts 应用开启时调用
- void destroy()：在 struts 应用销毁时调用

尽管如此，要让这个类工作，还必须在 struts-config.xml 文件中进行配置。

（随便一提，本章提到的 struts-config.xml 这一文件命名是可以修改的）。

配置方法是：

```
    <!—需要配置，就想需要注册一样，它才能工作-->
    <plug-in className="com.yourcompany.struts.MyPlugIn"></plug-in>
```

只要在配置文件中添加一个 plug-in 元素，className 属性值指向自定义插件类的路径，需要注意的是<plug-in>元素在配置文件中不是任意地方都可以出现的。

扩展 Struts 的另一个办法是.使用自定义的 ActionForm 扩展 ActionForm。用法如下：

```
package com.yourcompany.struts;

import org.apache.struts.action.ActionForm;

public class MyActionForm extends ActionForm {
//在这里实现你的方法
    public void yourFunction(){
        // to do something here
    }

}
```

这个 MyActionForm 只是直接简单的继承了 ActionForm。并在里面添加一些自己的方法，供它的子类所复用，一般 yourFunction 里面编写一些公共的验证代码。要达到扩展的效果，必须继承 MyActionForm 这个类。