Session-10

Struts

Contents

What is Struts?

- Struts is an open source web application framework developed as Apache Jakarta project.
  http://jakarta.apache.org.struts/
- Model-View-Controller (MVC) Framework
- combine JSP, Servlets, Custom Tags and message resources into a unified framework.
- Pattern Oriented
- include Custom Tag Libraries

Why Struts?

- Takes much of the complexity out of building your own MVC framework
- Encourages good design practice and modelling
- Easy to learn and use
- Feature rich
- Many supported 3rd party tools
- Flexible and extensible
- Larger user community
- Stable and mature
- Open source

Why Struts?

- Integrates well with J2EE
- Good taglib support
- Works with existing web apps
- Easy to retain from state
- Unified error handling programmatically and declaratively
- Integration with Tiles framework
- Clear delineation of responsibility makes long term maintenance easier

Installing Struts

- Prerequisite Softwares:
  - Java Development Kit(jdk)
    - Jdk 1.4.2 and above
  - Servlet Conatiner
    - E.g Apache's Tomcat
  - XML Parser
    - Bundled with jdk 1.4
  - Servlet API classes
    - Servlet.jar (included in most servlet containers)
  - Other packages
    - Comes as *.jars

Installing a Struts Binary Distribution

After installed the prerequisite softwares,

http://struts.apache.org/download.cgi

- Download Binary Distribution from the following site:
  - http://www.
- Unzip the downloaded package.
- Unpack struts-blank-1.3.10.war under struts-1.3.10/apps.
- Create a Tomcat project in Eclipse.
- Copy all files & dirs under (unpacked form of struts-blank-1.3.10.war) to the project.

Now you are on the way using Struts.

How do Struts Work?

- Struts are based on the time-proven Model-View-Controller (MVC) design pattern.

- So, the processing is broken into three distinct sections:
  - **Model Component**
    - Plays a vital role because they provide a **"model"** of the business logic or data behind a Struts program.
    - Examples: Customer, Employee, Student, etc..
    - Acts as an interface between the Struts application and real-world application.

How do Struts Work?

- **View Component**
  - Present information to users and accept input.
  - These correspond to web pages in Struts applications.
  - To display the information provided by **Model** component, **View** components are used.
  - Generally built using JSP files.

- **Controller Component**
  - Coordinates activities in the application.
  - Perform in a series of tasks:
    - Accept data from user
    - Decide which **Model** components need to be updated,
    - Then decide which **View** component needs to be called to display the result.

Building Pages with Struts

1) Implements the data entry form as JSP files.
2) Create MessageResources and Application.properties file.
3) Implements one or more **ActionForm** descendents to buffer data between JSPs and Actions.
4) Create an XML document that defines the validation rules for your application.
5) Implements one or more **Action** descendents to responds from submissions.
6) Create struts-congif.xml to associate forms with actions.
7) Create or update web.xml to reference **ActionServlet, taglibs** used by struts.
8) Do parallel tasks as follows:
   - Building
   - Unit Testing
   - Deployment

Lets see **HelloWorld** Struts Application!

Requirements

- Enabling the user to enter a name to say Hello! <name>.
- Not to allow the user to submit the entry form without entering a name.
- To understand Controller functionality
- To add more Controller functionality such as the application should not allow the user to say Hello to some people.

The **View** Component: JSP form and Form Bean

hello.jsp

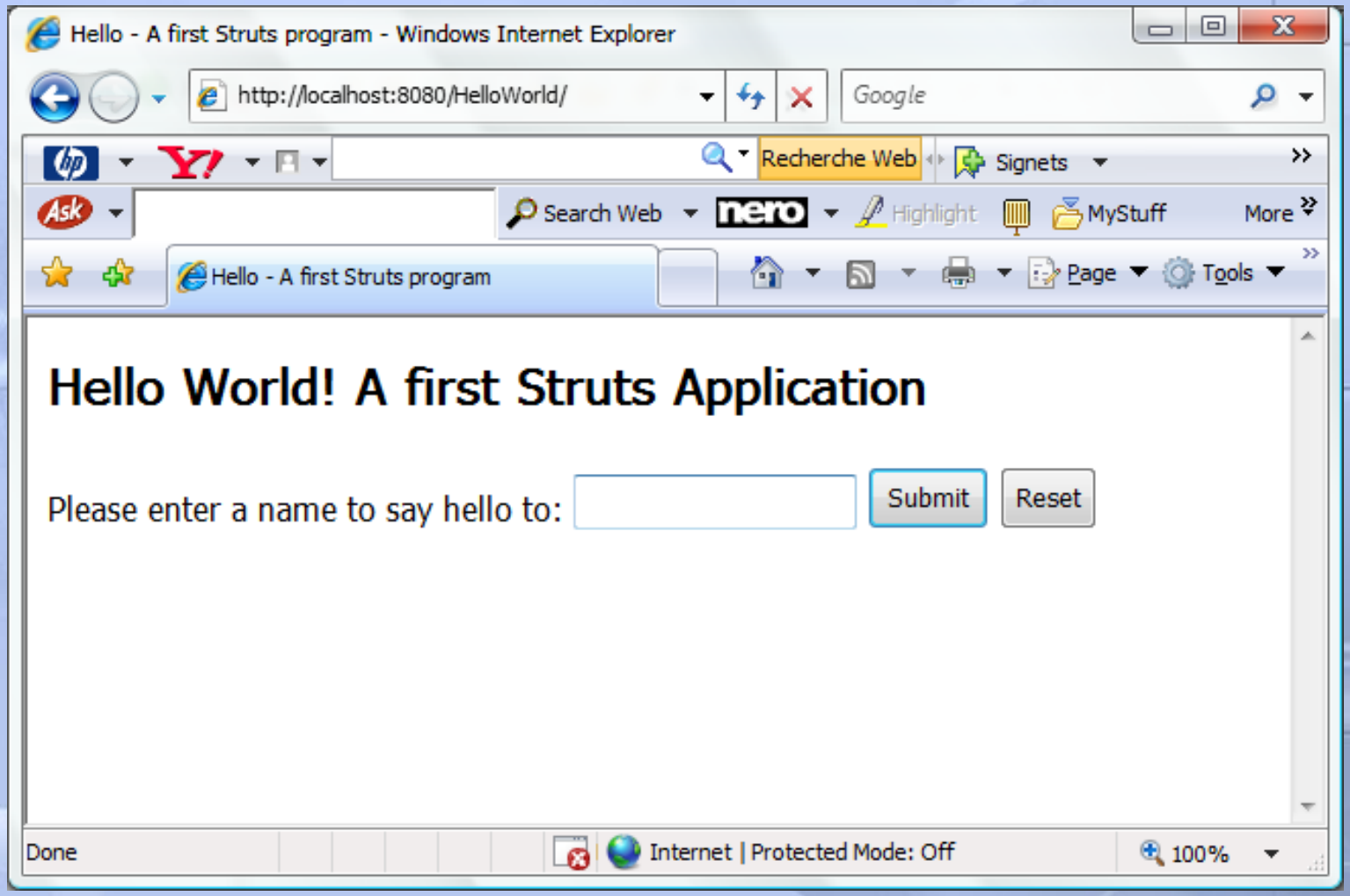
```
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>

<html:html>
    <head>
        <title><bean:message key="hello.jsp.title"/> </title>
        <html:base/>
    </head>
    <body>
        <h2><bean:message key="hello.jsp.page.heading"/> </h2>
        <font color="red">
            <p><html:errors/> </p>
        </font>
```

The **View** Component: JSP form and Form Bean

```
        <html:form action="/HelloWorld">
            <bean:message key="hello.jsp.prompt.user"/>
            <html:text property="person" size="16"></html:text>
            <html:submit property="submit" value="Submit"></html:submit>
            <html:reset></html:reset>
        </html:form>
    </body>
</html:html>
```

The **View** Component: JSP form and Form Bean



Understanding elements in **hello.jsp**

Struts tags

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

- Identify and load the Struts tag library definitions.
- This is the standard JSP syntax to load the tag libraries and make the bean tags available for use in the file.

Understanding elements in **hello.jsp**

Some elements under **html** tag library

- **<html:errors>** - This tag is used to access and to present the results of Struts' data validation. Errors detected by other portions of the framework are accessible to the **View** component via this tag.

- **<html:form>** - This tag is used for all HTML form processing in Struts. It ties the form fields to properties in Struts derived from beans. Each form field will be tied to a corresponding property in the form bean.

- **<html:text>** - This tag is used inside an <html:form> tag. It ties a text field in the form to a property in the form bean.

Understanding elements in **hello.jsp**

Some elements under **bean** tag library

- **<bean:message>** - This tag is used to output the locale-specific text from a MessageResources bundle.

- **<bean:write>** - This is a general purpose tag, which is been used to output property values form a bean.

E.g.,

<bean:message key="hello.jsp.prompt.user"/>

Understanding elements in **hello.jsp**

Some elements under **logic** tag library

- **<logic:present>** - This tag is used to render output if a bean is present and available to the JSP page.

- **<logic:notPresent>** - opposite to <logic:notPresent>

MessageResources.properties file

```
hello.jsp.title =Hello - A first Struts program
hello.jsp.page.heading =Hello World! A first Struts Application
hello.jsp.prompt.user=Please enter a name to say hello to:
hello.user.prompt.error =Please enter a <i>USER</i> to say hello!
hello.user.prompt.badPerson =You are not welcomed!
```

The Struts Form Bean: **HelloForm.java**

- When the user clicks submit button available on an HTML form, the data from that form is populated into a Java bean.
- This is called a **form bean**.

- A form bean is a simple java bean.
- A form bean has properties.
- These properties can match up with all the fields on the form.

- Form bean provides support for **automatic data validation** and **resetting** of the bean property values.

The Struts Form Bean: **HelloForm.java**

```
package hello.app;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class HelloForm extends ActionForm
{
    private String person;

    public HelloForm() {
        super();
    }

    public String getPerson() {
        return person;
    }
```

The Struts Form Bean: **HelloForm.java**

```
    public void setPerson(String person) {
        this.person = person;
    }
    public void reset(ActionMapping mapping, HttpServletRequest request){
        this.person=null;
    }

    public ActionErrors validate(ActionMapping mapping, HttpServletRequest
    request) {
        ActionErrors e=new ActionErrors();
        if(this.person==null || this.person.trim().length()==0)
            e.add("personError",new ActionMessage("hello.user.prompt.error"));
        return e;
    }
}
```

Data Validation and ActionErrors

- Struts provides a powerful way of handling data validation as follows:
  1) Providing an easy to use method of capturing error information as it is discovered.
  2) Making that information available to the **View** component to access and display the information as needed.

- Struts provides two classes for this tasks:
  1) **ActionError /ActionMessage**: This class is used to represent a single validation error.
  2) **ActionErrors** : This class provides a place to store all the individual ActionMessage objects you create.

Two types of errors defined in HelloWorld

- **Form Validation** : In the data entry form, you have to make sure that the user doesn't submit the form with the user field empty.

- **Business Logic** : You must enforce a rule that the user can't say hello to a user to whom he isn't allowed to talk to.

Form Validation in HelloForm.java

```
public ActionErrors validate(ActionMapping mapping, HttpServletRequest request){
    ActionErrors errors = new ActionErrors();
    if(person==null || person.trim().length()==0)
        errors.add("personError", new ActionMessage("……..."));
}
```

- If the validate() method returns the ActionErrors object empty , Struts will assume that there are no errors and therefore processing moves to **Action** class.
- Otherwise, the user is redirected to the appropriate page to correct the errors.
  - The ActionErrors object carried the individual ActionMessage elements back to the **View** for display.
  - The **View** component access it either directly or through <html:errors> tag.
- Validation can be enabled/disabled on a page-by-page basis.

The Controller Component: **HelloAction.java**

```
package hello.app;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class HelloAction extends Action {
    public ActionForward execute (ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response)
    {
        ActionErrors errors=new ActionErrors();
        if(this.isCancelled(request))
        {
            return (mapping.findForward("mainPage"));
        }
        HelloForm hf=(HelloForm)form;
        String p=hf.getPerson();
```

The Controller Component: **HelloAction.java**

```
        String badPerson="virus";
        if(p.equalsIgnoreCase(badPerson))
        {
            errors.add("badPerson",new ActionMessage("hello.user.prompt.badPerson"));
            saveErrors(request, errors);
            return new ActionForward(mapping.getInput());
        }
        HelloModel hm=new HelloModel(p);
        hm.savePerson();
        request.setAttribute("name", p);
        return (mapping.findForward("sayHello"));
        }
}
```

How the Action class works

- An Action class is a class that extends the base class:

  org.apache.struts.action.Action

- There is a method that must be written in an Action class:

```
public ActionForward  execute (ActionMapping mapping, ActionForm form,
    HttpServletRequest  request, HttpServletResponse  response) throws
    Exception
{


}
```

- A framework will call this method after the form bean is populated and validated correctly.

How the Action class works

- Four parameters in execute(…) methods:

1. **ActionMapping** mapping: provides access to information stored in configuration file (struts-config.xml).

2. **ActionForm** form : It is form bean. By this time, the form bean has been populated and validate() method has been called and returned with no errors.

3. **HttpServletRequest** request : It is standard JSP / Servlet request object.

4. **HttpServletResponse** response : It is standard JSP / Servlet response object.

Tasks in **HelloAction.java**

- Accessing the locale specific text in MessageResources.properties
- Business logic level validation
- Interacting with model components
- Passing data to View component
- Forwarding to the Appropriate View Component

Tasks in **HelloAction.java**

- Accessing the locale specific text in MessageResources.properties

```
MessageResources messages = getResources(request);
```

Tasks in **HelloAction.java**

- Business logic level validation

```
String badPerson="virus";

if(p.equalsIgnoreCase(badPerson))
{
        errors.add("badPerson",new  ActionMessage("hello.user.prompt.badPerson"));
        saveErrors(request, errors);
        return new ActionForward(mapping.getInput());
}
```

Tasks in **HelloAction.java**

- Interacting with model components

```
HelloModel hm=new HelloModel(p);
hm.savePerson();
```

Tasks in **HelloAction.java**

- Passing data to View component

```
request.setAttribute("name", p);
```

Tasks in **HelloAction.java**

- Forwarding to the Appropriate View Component

```
return (mapping.findForward("sayHello"));
```

The Model Component

```
package hello.app;

public class HelloModel {
    private String person;

    public HelloModel() { }
    public HelloModel(String person) { this.person = person; }
    public String getPerson() { return person; }
    public void setPerson(String person) {  this.person = person; }
    public void savePerson()
    {       //.....
            System.out.println("Save user = " + person);
    }
}
```

The struts-config.xml

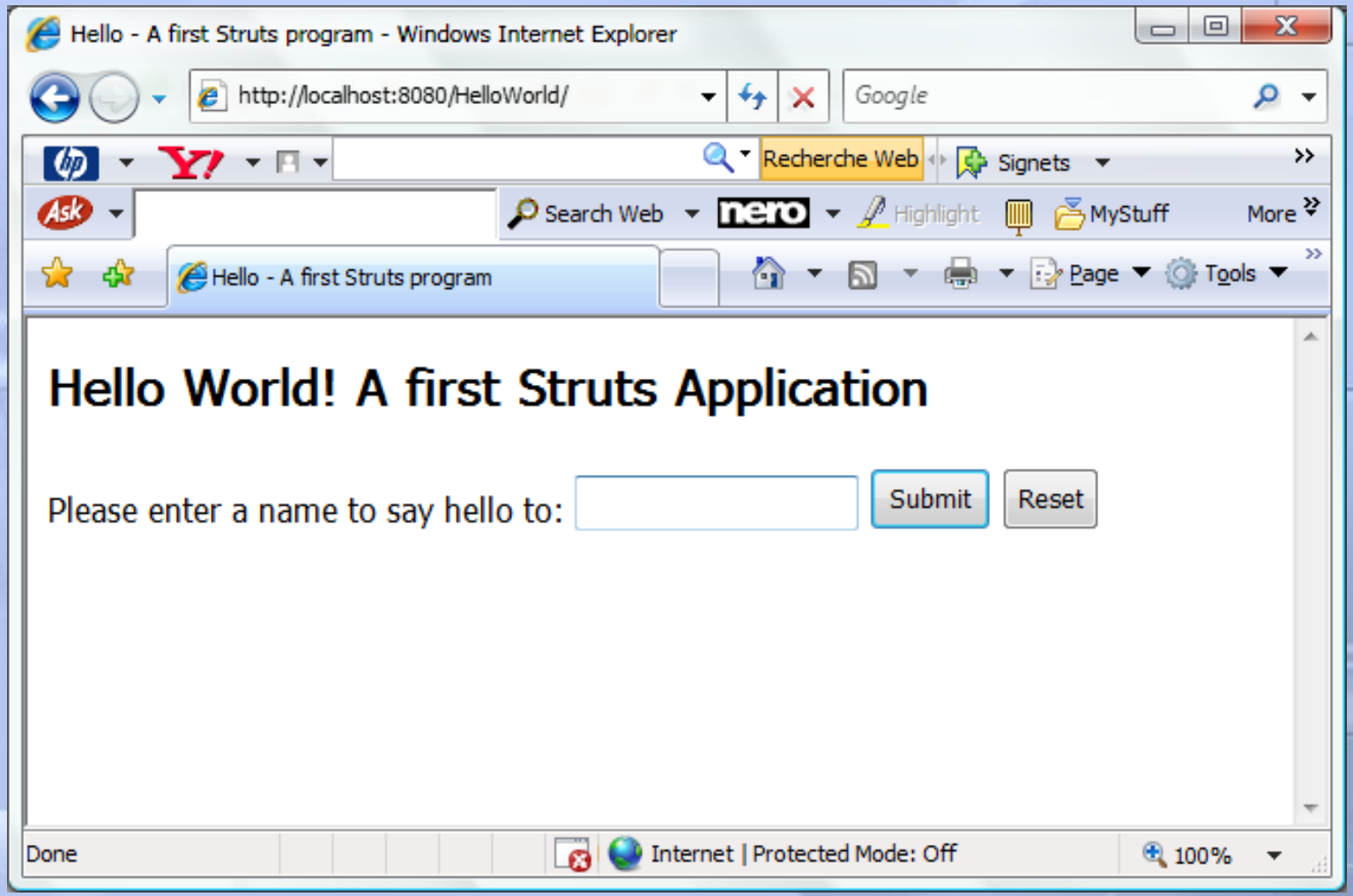
```
<struts-config>
<!-- ================ Form Bean Definitions ================ -->
    <form-beans>
        <form-bean name="helloForm"      type="hello.app.HelloForm">
        </form-bean>
    </form-beans>

<!-- ========== Global Exception Definitions ==================-->
    <global-forwards>
        <forward name="mainPage" path="/hello.jsp"/>
    </global-forwards>
```

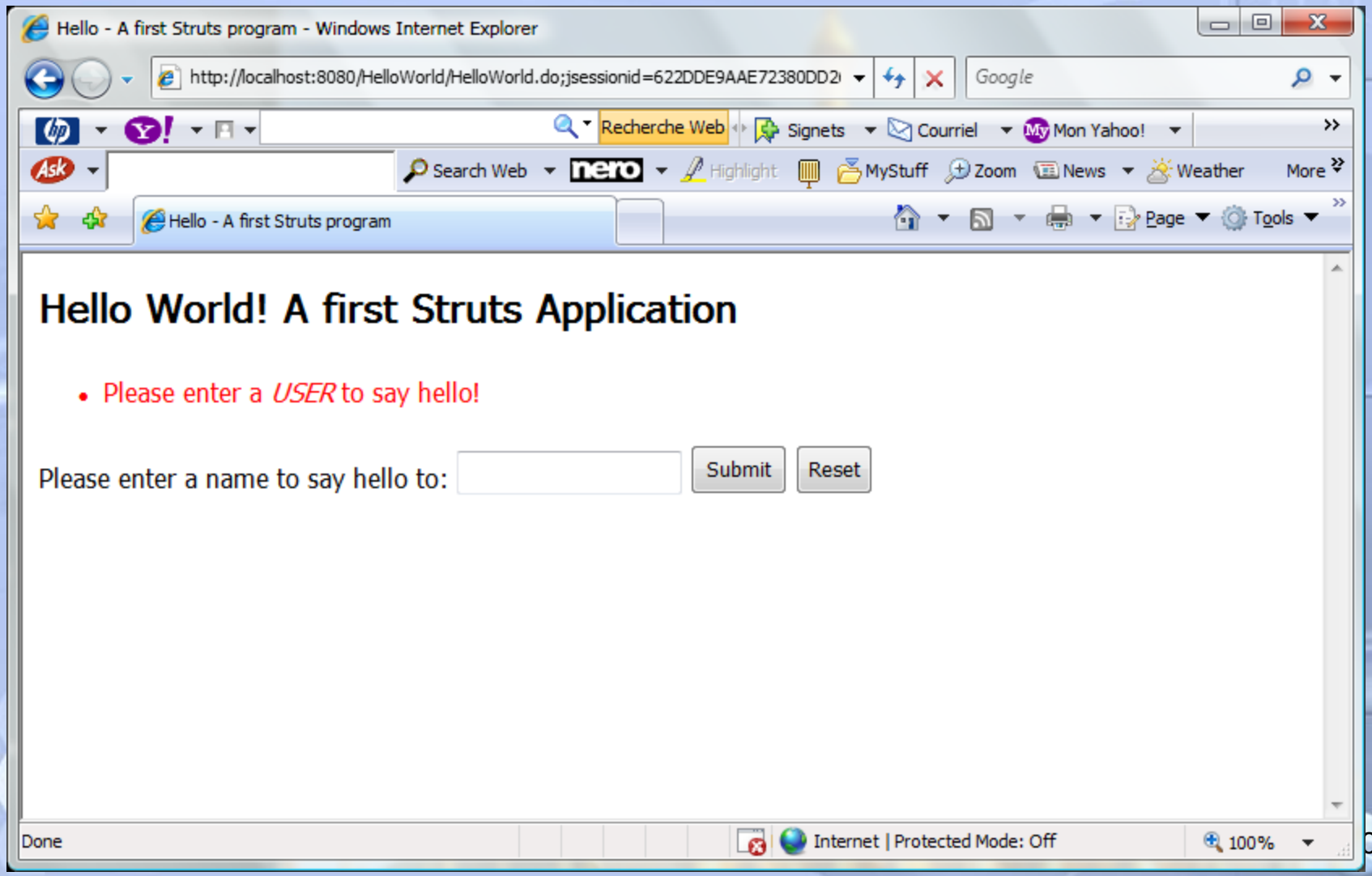
The struts-config.xml

```
<!-- ============= Action Mapping Definitions ==============-->
   <action-mappings>
         <action    path="/HelloWorld"
                    type="hello.app.HelloAction"
                    name="helloForm"
                    scope="request"
                    validate="true"
                    cancellable="true"
                    input="/hello.jsp" >
           <forward name="sayHello" path="/pages/Welcome.jsp"/>
         </action>
   </action-mappings>
<!-- ============== Message Resources Definitions ============-->
   <message-resources parameter="/java/MessageResources" />
</struts-config>
```

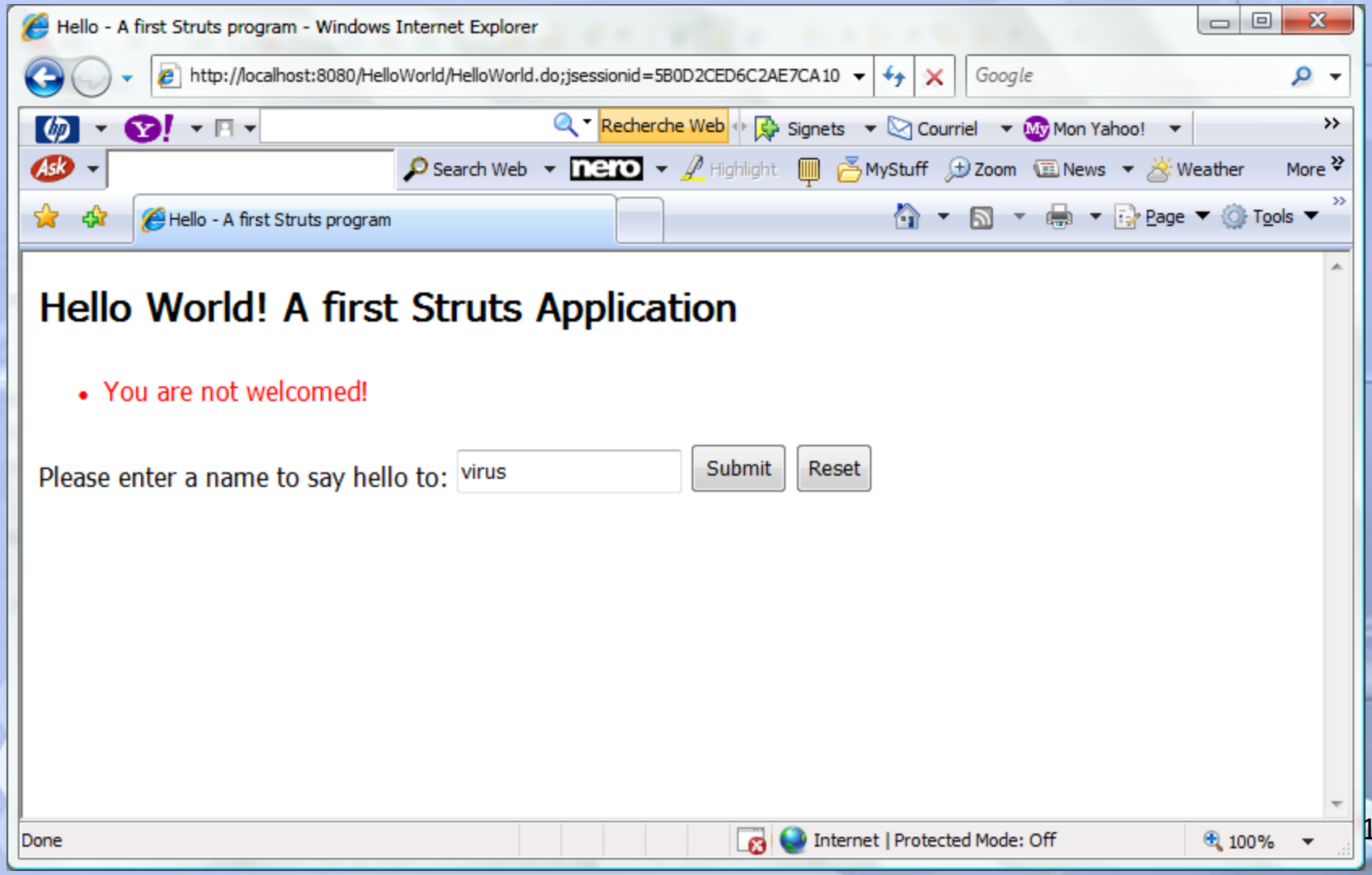
Output of View Component



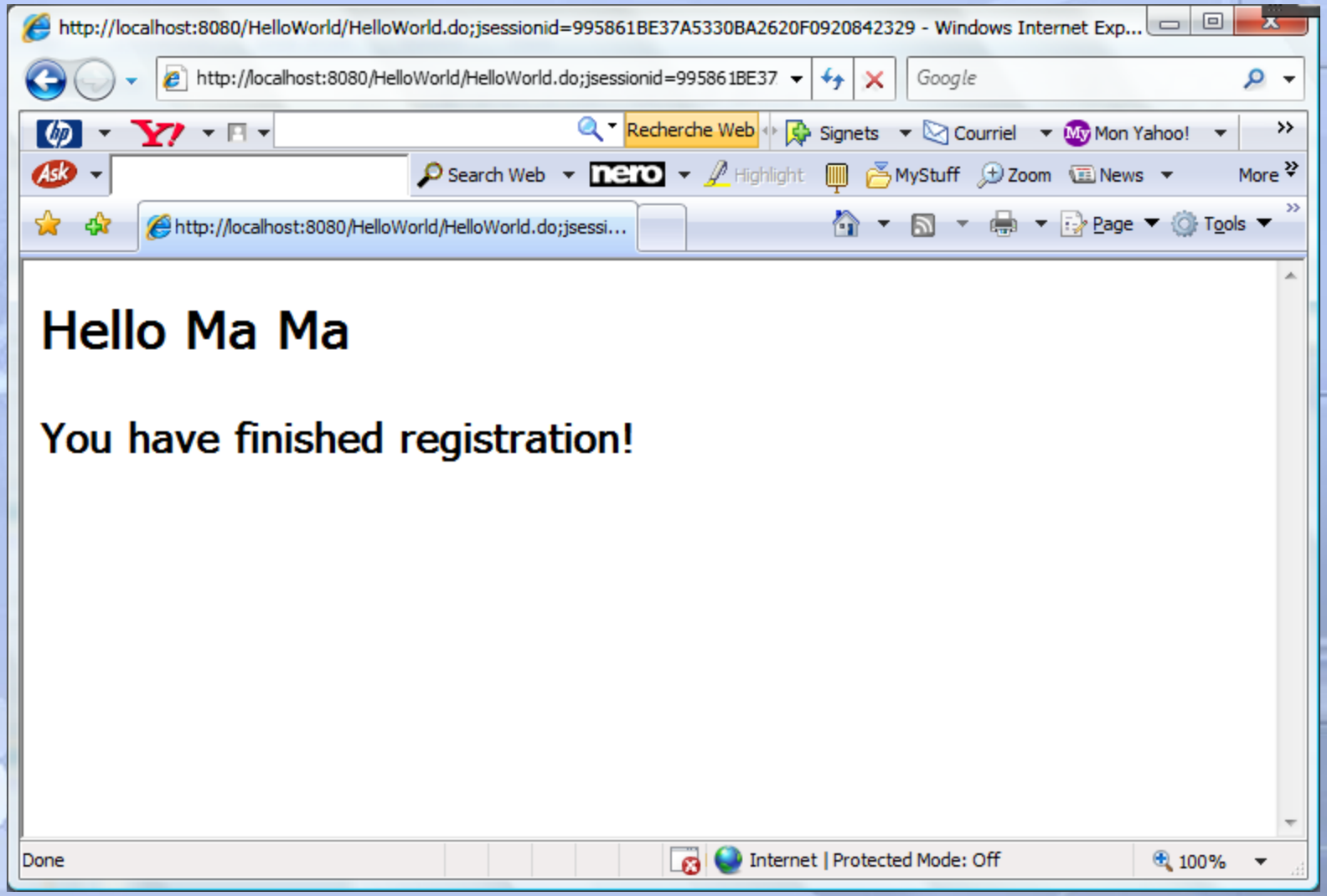
Output of View Component



Output of View Component



Output of View Component



Thank you!