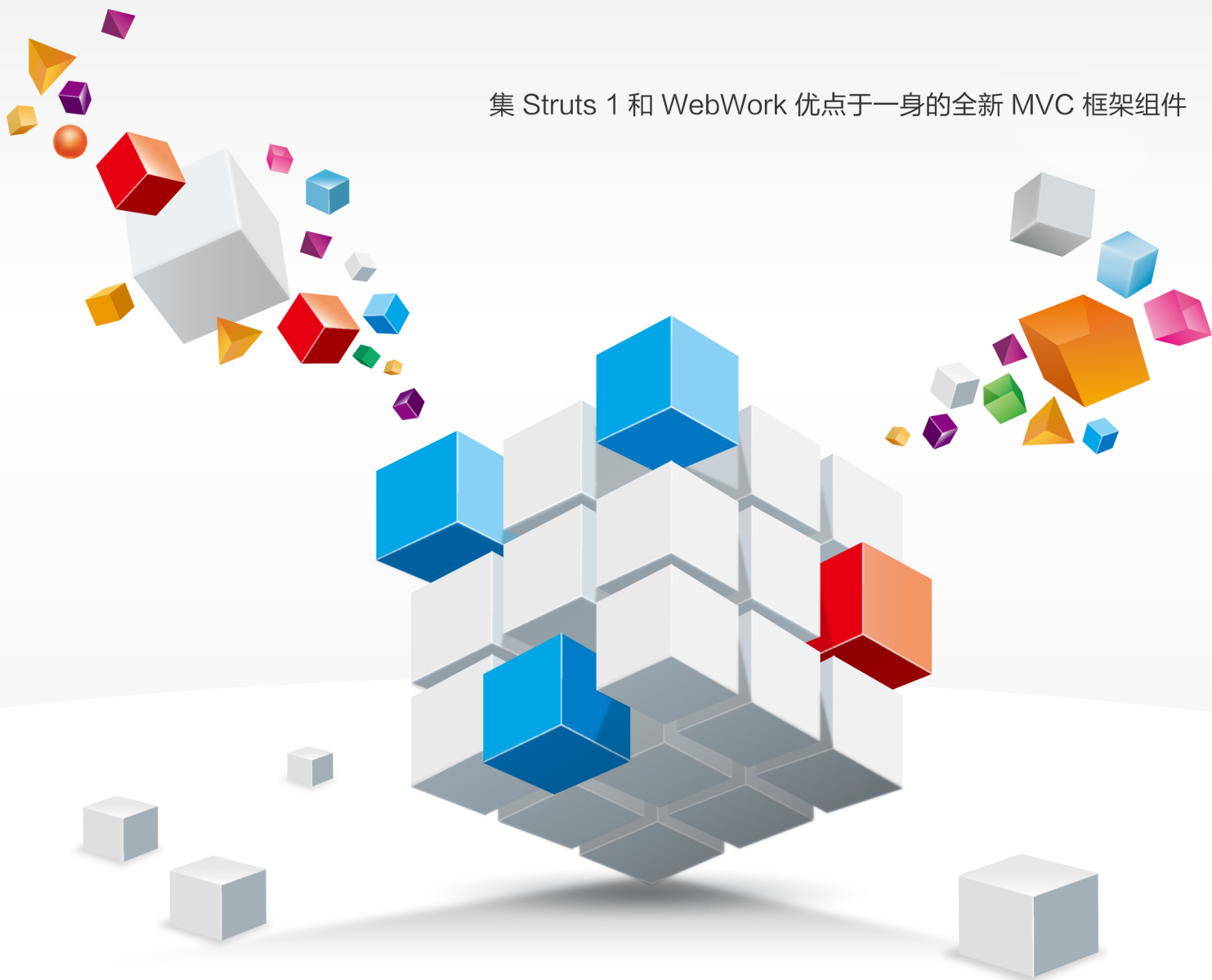


集 Struts 1 和 WebWork 优点于一身的全新 MVC 框架组件



Struts 2 教程

极客学院出版

前言

Apache Struts 2 是一个简洁的，可扩展的框架，它用来创建企业级的 Java web 应用程序。框架的目的是随着时间的积累从构建，部署，维护应用程序上简化整个开发周期。Apache Struts 2 最初被称为 WebWork 2。

本教程将教你如何用简单和容易的步骤使用 Apache Struts 来创建企业级的 Java web 应用程序。

适用人群

本教程是专门为了需要理解 Struts 2.x 框架及其应用程序的 Java 程序员设计的。本教程将使你达到专家的中等水平。

预备知识

在继续本教程之前，你应该很好的理解 Java 编程语言。对 MVC 框架和 JSP 或 Servlet 有基本的了解是很有帮助的。

更新日期	更新内容
2015-06-01	第一版发布

目录

前言	1
第 1 章 Struts 2 基础	4
基本的 MVC 架构	5
概述	7
环境配置	9
体系结构	13
实例	15
配置	24
动作	29
拦截器	36
结果类型	46
值栈/OGNL	50
文件上传	56
数据库访问	63
Struts 2 – 发送邮件	69
验证	75
本地化	82
类型转换	89
主题/模板	94
异常处理	99
注释	104
第 2 章 Struts 2 标签	117
控制标签	118
数据标签	120

	表单标签	124
	Ajax 标签	129
第 3 章	Struts 2 集成	131
	Spring 集成	132
	Tiles 集成	137
	Hibernate 集成	142
第 4 章	Struts 2 有用的资源	151
	有用的资源	152



Struts 2 基础

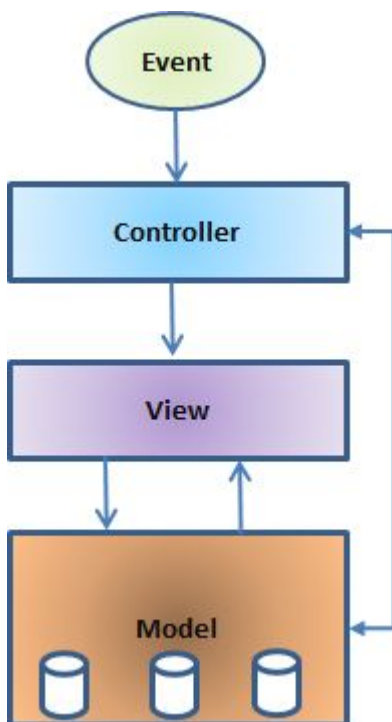


基本的 MVC 架构

模型-视图-控制器或通常被称为 MVC，是一种用于开发 web 应用程序的软件设计模式。模型-视图-控制器模式由以下三个部分组成：

- ？ Model – 模式的最低层，负责维护数据。
- ？ View – 负责显示全部或部分的数据给用户。
- ？ Controller – 控制模型和视图之间的交互的软件代码。

MVC 是受欢迎的，是因为它把应用逻辑从用户接口层中分离，而且支持关注点的分离。在这里，控制器接收应用程序的所有请求，然后与模型一起工作准备好视图需要的任何数据。然后视图使用控制器准备好的数据来生成最终正式的响应。MVC 抽象可以用图形表示，如下所示。



模型

模型负责管理应用程序的数据。它响应来自视图的请求，而且它也响应来自控制器的指令进行更新自身。

视图

在一个特定格式中数据的展示由一个控制器的决定引发来呈现该数据。它们是基于模板系统的脚本，如 JSP，ASP，PHP，而且它们很容易与 AJAX 技术进行集成。

控制器

控制器负责响应用户的输入和执行数据模型对象的交互。控制器接收输入，验证输入，然后执行修改数据模型状态的业务操作。

Struts 2 是一个基于 MVC 的框架。在接下来的章节中，我们会看到如何使用包含 Struts 2 的 MVC 方法。

概述

Struts 2 是流行且成熟的基于 MVC 设计模式的 web 应用程序框架。Struts 2 不只是 Struts 1 的版本升级，而是一个完全重写的 Struts 架构。

WebWork 框架一开始是以 Struts 框架为基础的，它的目标是提供一个建立在 Struts 上加强和改进的框架来使开发人员更容易进行 web 开发。

一段时间之后，WebWork 框架和 Struts 社区联手创建了著名的 Struts 2 框架。

Struts 2 框架的功能

这里有一些强大的功能，它可能会迫使你考虑 Struts 2：

- ？ POJO 表单和 POJO 动作 – Struts 2 已经去掉了原先是 Struts 框架的一个组成部分的动作表单，利用 Struts 2，你可以使用任何 POJO 来接收表单的输入。同样，你现在可以看到任何作为 Action 类的 POJO。
- ？ 标签支持 – Struts 2 已经改进了表单标签和新标签允许开发人员编写更少的代码。
- ？ AJAX 支持 – Struts 2 已经确认由 Web2.0 技术接管，并且通过创建 AJAX 标签把 AJAX 支持集成到产品中，AJAX 标签的功能非常类似于标准 Struts 2 标签的功能。
- ？ 易于整合 – 与其他框架如 Spring 一起集成，Tiles 和 SiteMesh 是现在更容易使用各种各样有效的使用 Struts 2 的集成。
- ？ 模板支持 – 支持使用模板生成视图。
- ？ 插件支持 – 核心 Struts 2 的特性可以使用插件得到提高和增强。大量的插件对于 Struts 2 来说是可用的。
- ？ 配置 – Struts 2 提供了集成配置来调试和配置应用程序。除此之外，Struts 还在内置调试工具的帮助下提供了集成调试。
- ？ 易于修改标签 – Struts 2 的标签标记可以使用 Freemarker 模板进行调整。这并不需要 JSP 或 Java 知识。基础的 HTML，XML 和 CSS 知识是足够修改标签的。
- ？ 提升较少的配置 – Struts 2 在使用各种设置的默认值的帮助下促进较少的配置。你不需要配置，除非它偏离 Struts 2 设定的默认设置。
- ？ 视图技术: – Struts 2 大力支持多个视图选项(JSP，Freemarker，Velocity 和 XSLT)

以上只是 Struts 2 使它成为企业级框架的前十大功能。

Struts 2 的缺点

虽然 Struts 2 有一列强大的功能，但我不会忘了提及一些关于 Struts 2 的缺点，它将需要很多改进：

- ？ 更大的学习曲线 – 要使用带有 Struts 的 MVC，你必须是熟练使用标准的 JSP，Servlet APIs 和大量精心设计的框架。
- ？ 拙劣的文档 – 与标准的 servlet 和 JSP APIs 相比，Struts 有较少的在线资源，许多第一次使用的用户发现网上 Apache 文档混乱而且缺乏组织。
- ？ 较少的透明度 – 使用 Struts 应用程序有比使用标准的基于 Java 的 Web 应用程序更多的幕后，这使得它很难理解框架。

最后一点，一个好的框架应该提供许多不同类型的应用程序可以使用的通用的特性。Struts 2 是最好的 web 框架之一，而且被广泛地用于开发 Rich 互联网应用（RIA）。

环境配置

我们的首要任务是让最小的 Struts 2 应用程序运行。本章将指导你如何准备开发环境来使用 Struts 2 开始你的工作。假设你已经在你的机器上安装了 JDK (6 +) , Tomcat 和 Eclipse。如果你还没有安装这些组件，然后按照快速通道上给出的步骤：

Step 1 – 安装 Java 开发工具包 (JDK)

你可以从 Oracle 的 Java 网站：[Java SE Downloads](#) 下载 SDK 的最新版本。你会在下载的文件中找到教你如何安装 JDK 的说明，按照给出的说明安装和配置 JDK 的设置。最后，设置 PATH 和 JAVA_HOME 环境变量，引入包含 java 和 javac 的目录，通常分别为 java _ install _ dir/bin 和 java _ install _ dir。

如果你运行的是 Windows，并在 C:\jdk1.6.0_20 上安装了 SDK，你就可以把下面这行写入 C:\autoexec.bat 文件中。

```
set PATH=C:\jdk1.6.0_20\bin;%PATH%
set JAVA_HOME=C:\jdk1.6.0_20
```

或者，在 Windows XP/7/8 中，你也可以右键单击“我的电脑”，选择“属性”，然后是“高级”，然后是“环境变量”。接下来，你将更新 PATH 值，并且按下 OK 按钮。

在 Unix(Solaris、Linux 等等)上，如果在 /usr/local/jdk1.6.0_20 上安装 SDK，并且使用 C shell 命令，你将把下面的内容添加到 .cshrc 文件中。

```
setenv PATH /usr/local/jdk1.6.0_20/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.6.0_20
```

或者，如果你使用集成开发环境 (IDE)，如 Borland JBuilder，Eclipse，IntelliJ IDEA 或者 Sun ONE Studio，编译和运行一个简单的程序，用来确认 IDE 知道你安装了 Java，否则应该根据 IDE 给定的文档做正确的设置。

Step 2 – 安装 Apache Tomcat

你可以从 <http://tomcat.apache.org/> 下载 Tomcat 最新的版本。一旦你下载完安装包，并且解压二进制的发行版本到一个方便的位置。例如在 windows 上的 C:\apache-tomcat-6.0.33 中，或在 Linux/Unix 上的 /usr/local/apache-tomcat-6.0.33 中，并且创建 CATALINA_HOME 的环境变量指向这些位置。

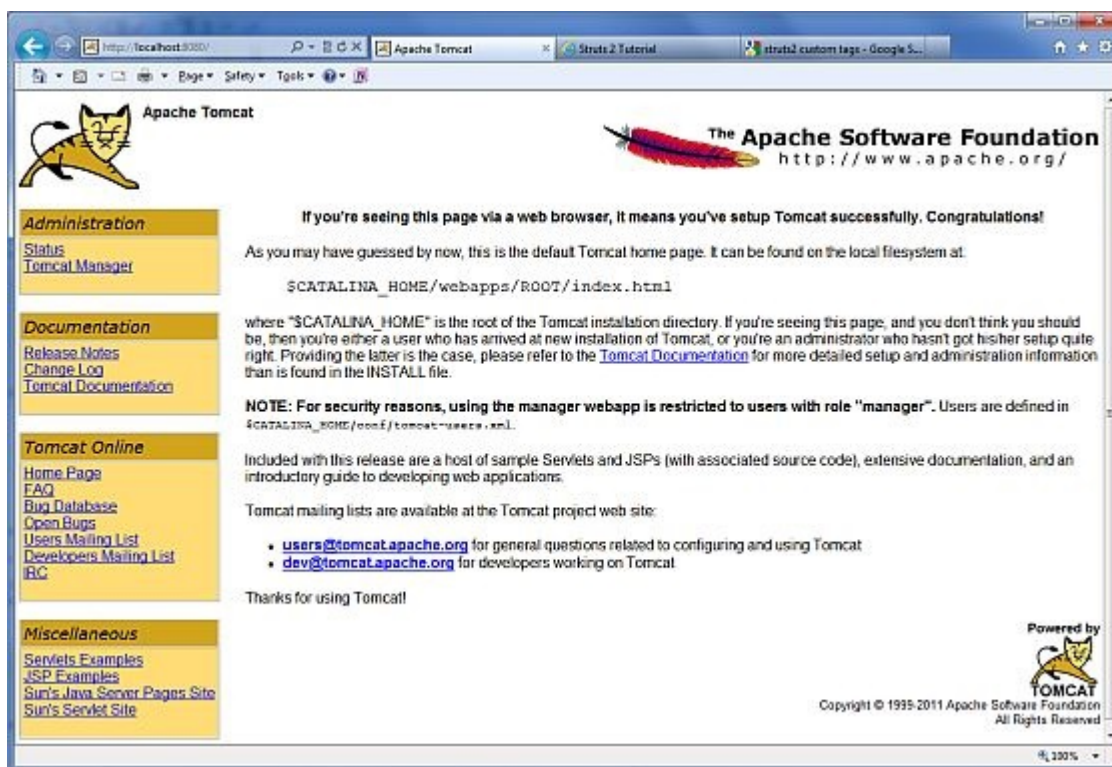
在 Windows 机器上，可以通过执行下列命令启动 Tomcat，或者你可以简单地双击 startup.bat：

```
%CATALINA_HOME%\bin\startup.bat
or
C:\apache-tomcat-6.0.33\bin\startup.bat
```

在 Unix (Solaris、Linux 等等)机器上，可以通过执行下列命令启动 Tomcat：

```
$CATALINA_HOME/bin/startup.sh
or
/usr/local/apache-tomcat-6.0.33/bin/startup.sh
```

成功启动后，可以通过访问 <http://localhost:8080/> 来使用包含 Tomcat 的默认的 web 应用程序。如果一切都正常，它应该显示下面的结果：



关于配置和运行 Tomcat 的更多信息可以在这里包含的文档中找到，也可以在 Tomcat 网站：<http://tomcat.apache.org> 中找到。

在 Windows 机器上，可以通过执行下列命令停止 Tomcat：

```
%CATALINA_HOME%\bin\shutdown
or
C:\apache-tomcat-5.5.29\bin\shutdown`
```

在 Unix (Solaris、Linux 等等)机器上，可以通过执行下列命令停止 Tomcat：

```
$CATALINA_HOME/bin/shutdown.sh  
or  
/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh
```

Step 3 – 安装 Eclipse (IDE)

本教程中的所有例子使用 Eclipse IDE 编写。所以我建议你应该在你的机器上安装 Eclipse 的最新版本。

为了安装 Eclipse，从 <http://www.eclipse.org/downloads/> 下载最新的 Eclipse 二进制文件。一旦你下载完安装包，并且解压二进制的发行版本到一个方便的位置。例如在 windows 上的 C:\eclipse 中，或在 Linux/Unix 上的 /usr/local/eclipse 中，最后恰当的设置 PATH 变量。

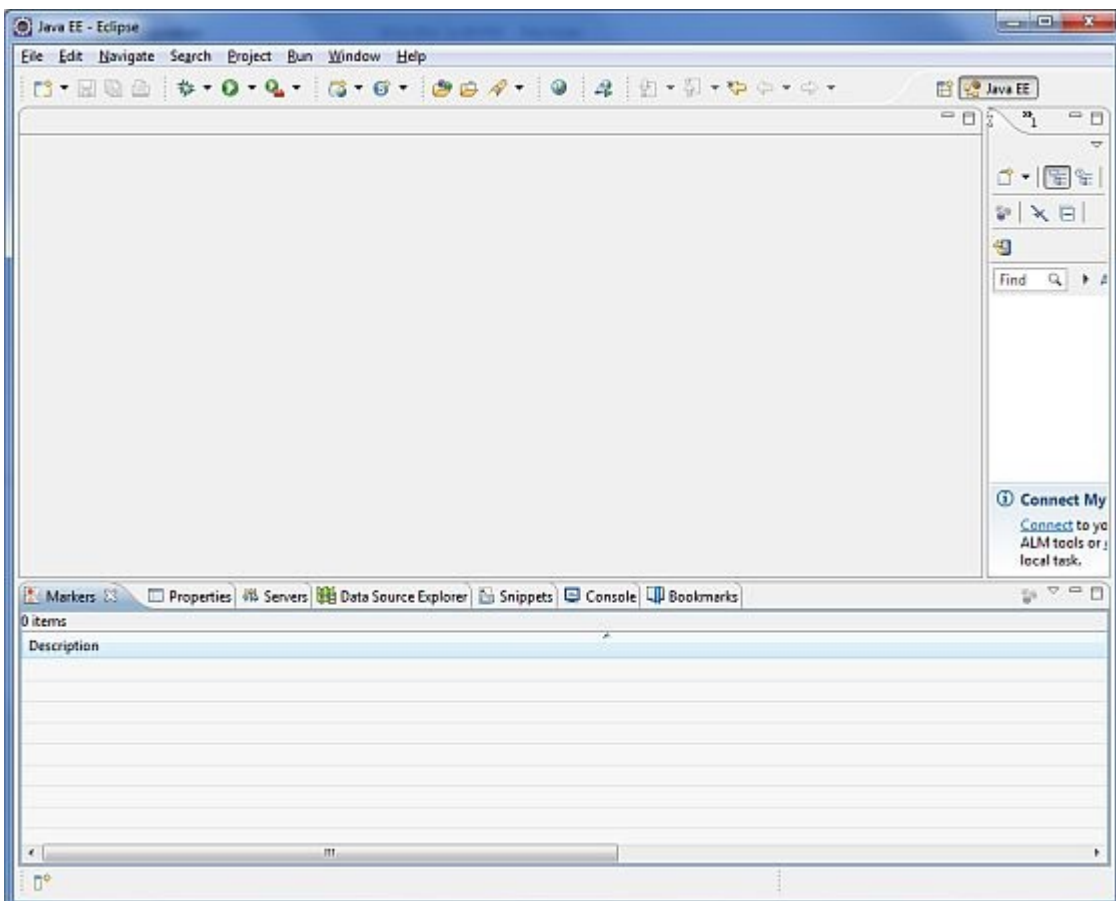
在 windows 机器上，可以通过执行以下命令启动 Eclipse，或者你可以简单地双击 eclipse.exe。

```
%C:\eclipse\eclipse.exe
```

在 Unix (Solaris 和 Linux 等) 上，可以通过执行下面的命令启动 Eclipse：

```
$/usr/local/eclipse/eclipse
```

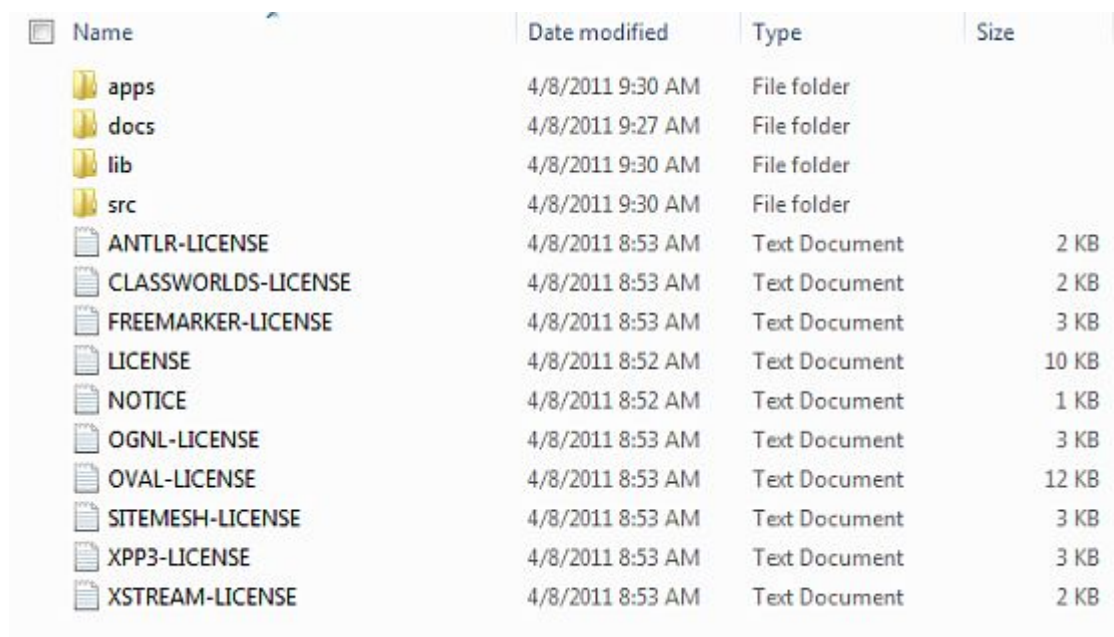
启动成功后，如果一切正常，它应该显示下面的结果：



Step 4 - 安装 Struts 2 库

现在如果一切正常，你就可以继续安装你的 Struts 2 框架。下面是在你的机器上下载并安装 Struts 2 的简单步骤。

- ？ 选择是要在 Windows 还是在 UNIX 上安装 Struts 2，然后继续进行下一个步骤，在 windows 上下载 .zip 文件,而在 Unix 上下载 .tz 文件。
- ？ 从 <http://struts.apache.org/download.cgi> 下载最新版本的 Struts 2 框架的二进制文件。
- ？ 在写本教程的时候，我下载了 struts-2.0.14-all.zip，当你解压缩下载的文件时，它内置的目录结构为 C:\struts-2.2.3，如下所示。



Name	Date modified	Type	Size
apps	4/8/2011 9:30 AM	File folder	
docs	4/8/2011 9:27 AM	File folder	
lib	4/8/2011 9:30 AM	File folder	
src	4/8/2011 9:30 AM	File folder	
ANTLR-LICENSE	4/8/2011 8:53 AM	Text Document	2 KB
CLASSWORLDS-LICENSE	4/8/2011 8:53 AM	Text Document	2 KB
FREEMARKER-LICENSE	4/8/2011 8:53 AM	Text Document	3 KB
LICENSE	4/8/2011 8:52 AM	Text Document	10 KB
NOTICE	4/8/2011 8:52 AM	Text Document	1 KB
OGNL-LICENSE	4/8/2011 8:53 AM	Text Document	3 KB
OVAL-LICENSE	4/8/2011 8:53 AM	Text Document	12 KB
SITEMESH-LICENSE	4/8/2011 8:53 AM	Text Document	3 KB
XPP3-LICENSE	4/8/2011 8:53 AM	Text Document	3 KB
XSTREAM-LICENSE	4/8/2011 8:53 AM	Text Document	2 KB

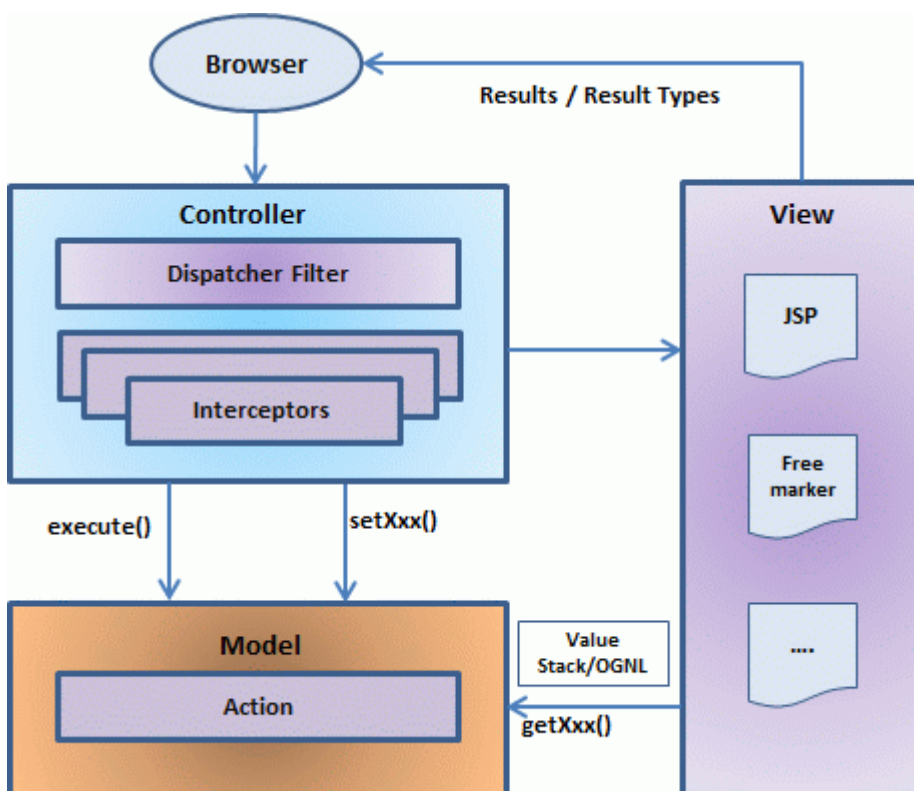
第二步是在任何位置中解压 zip 文件，我在 Windows 7 机器上的 c:\ 文件夹中下载并解压 struts-2.2.3-all.zip，因此我把所有的 jar 文件放到了 C:\struts-2.2.3\lib 中。确保你正确地设置了 CLASSPATH 变量，否则你将会在运行应用程序时遇到问题。

体系结构

从一个高水平看，Struts 2 是一个 pull-MVC（或 MVC2）框架。Struts 2 的模型-视图-控制器模式由下面的五个核心部件实现：

- ？ 动作
- ？ 拦截器
- ？ 值栈/OGNL
- ？ 结果/结果类型
- ？ 视图技术

Struts 2 与传统的 MVC 框架稍有不同，其中动作担任模型的角色，而不是控制器的角色，虽然有一些重叠。



上面的图描绘模型，视图和控制器到 Struts 2 高级架构。控制器是由 Struts 2 调度 servlet 过滤器和拦截器实现的，模型是由动作实现的，视图是由结果类型和结果结合而成的。值栈和 OGNL 提供共同主线，连接和集成其他组件。

除了上面的组件，还有很多与配置相关的信息。不仅要配置 web 应用程序，也要配置动作，拦截器，结果，等等。

这是 Struts 2 MVC 模式的体系结构的概述。我们将在后续章节中更详细的介绍每个组件。

请求生命周期

以上面的图为基础，它可以解释 Struts 2 中用户的请求的生命周期，如下所示：

- ？ 为了请求一些资源（即页面），用户发送请求到服务器。
- ？ FilterDispatcher 查看请求，然后确定适当的动作。
- ？ 配置的拦截器功能适用于如验证，文件上传等等。
- ？ 执行选定的动作来执行所请求的操作。
- ？ 再次，如果需要，配置的拦截器应用于做任何后处理。
- ？ 最后由视图准备好的结果，并且将结果返回给用户。

实例

因为你学习了 Struts 2 架构，当你在 Struts 2 web 应用程序中点击一个超链接或者提交一个 HTML 表单时，控制器会收集输入并且发送到一个称作 Actions 的 Java 类。当 Action 执行后，结果选择一个资源来显现响应。资源通常是一个 JSP，但是它也可以是一个 PDF 文件，Excel 电子表格或者 Java applet 窗口。

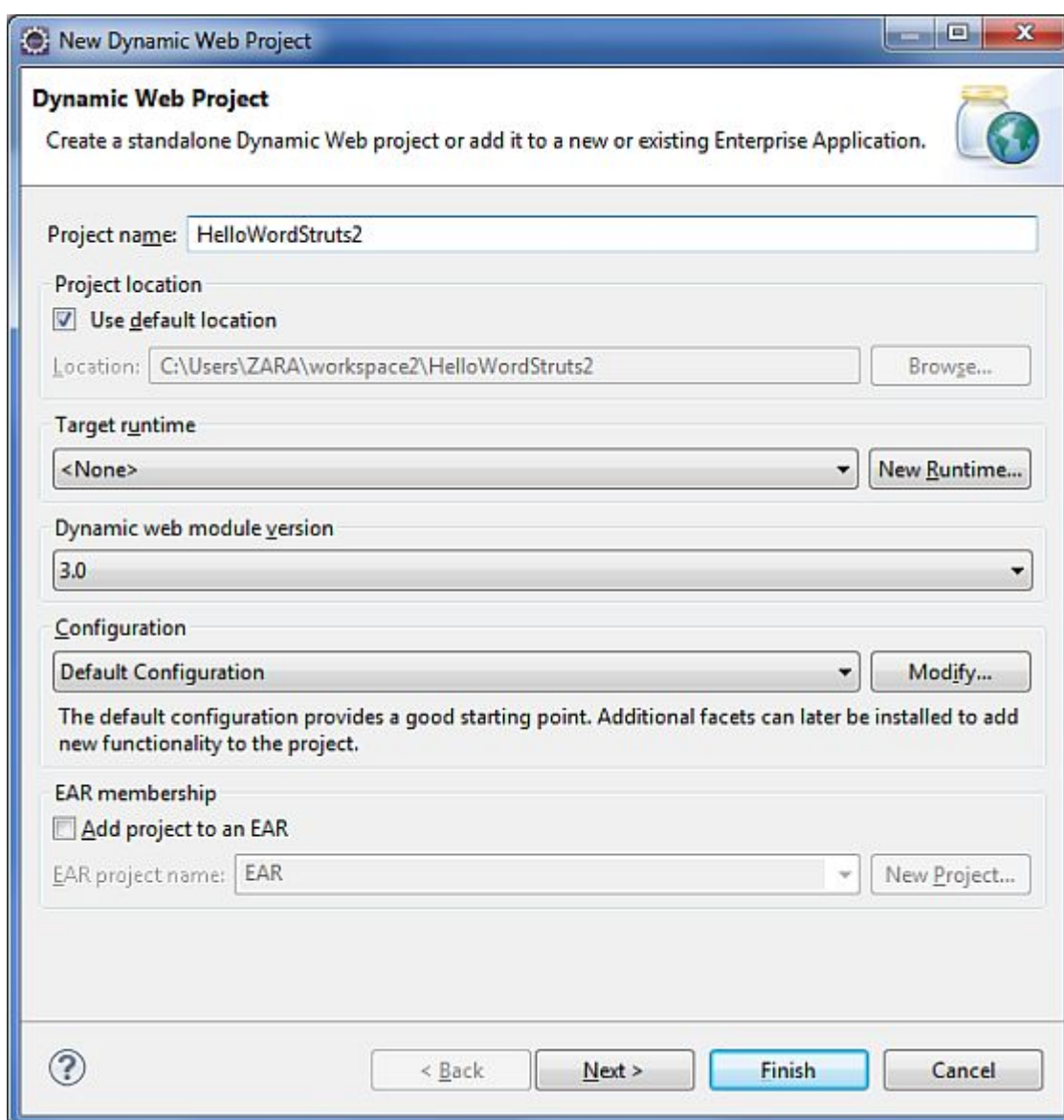
假设你已经建立了开发环境。现在让我们继续建立我们的第一个 Hello World struts2 项目。这个项目的目的是建立一个 Web 应用程序，该项目收集用户的姓名，并且在用户名后面显示 “Hello World”。为了任何的 Struts 2 项目，我们将必须创建四个组件：

序号	组件 & 描述
1	<p>动作</p> <p>创建一个包含完整的业务逻辑和控制用户，模型和视图之间的交互的动作类。</p>
2	<p>拦截器</p> <p>如果需要，则创建拦截器，或者使用已存在的拦截器。这是控制器的部分。</p>
3	<p>视图</p> <p>创建一个与用户交互的 JSPs，它接受输入并且显示最后的信息。</p>
4	<p>配置文件</p> <p>创建连接动作，视图和控制器的配置文件。这些文件是 struts.xml，web.xml，struts.properties。</p>

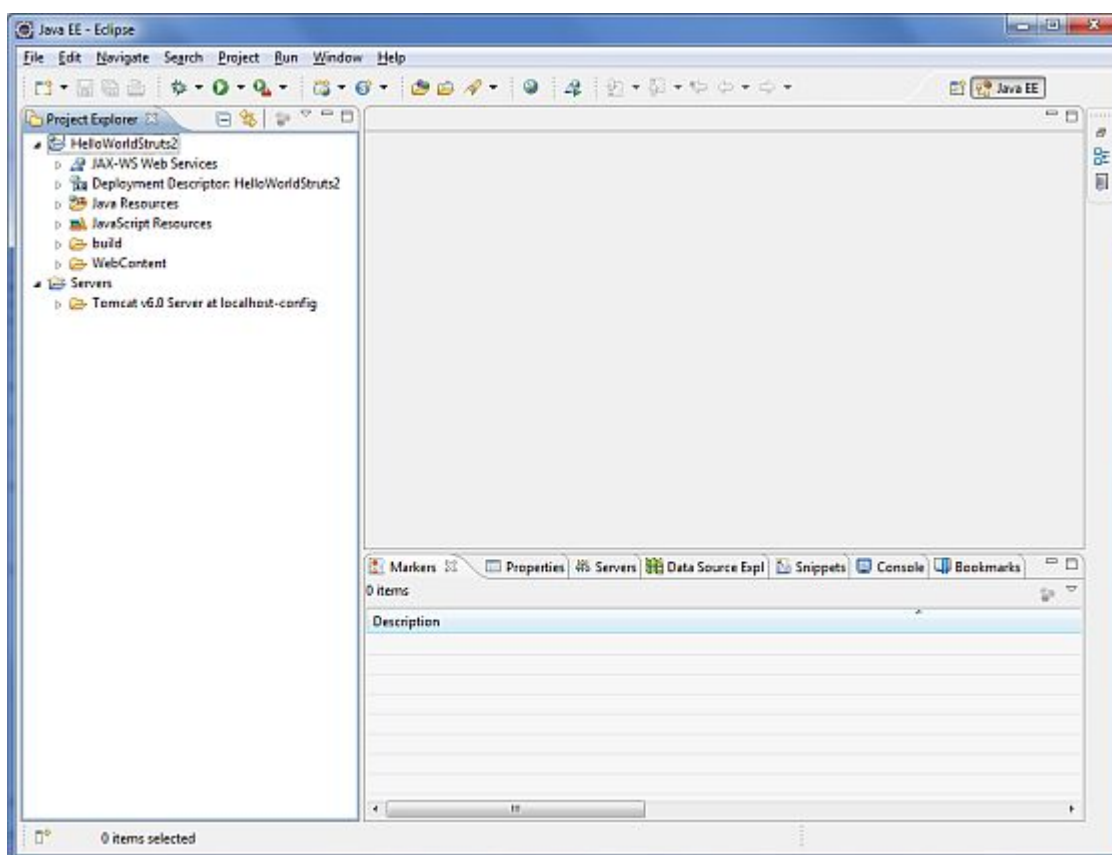
我将使用 Eclipse IDE，所以在一个动态 Web 项目中所有必需的组件都将会被创建。因此，让我们开始创建一个动态 Web 项目。

创建动态 Web 项目

启动你的 Eclipse，然后使用 File > New > Dynamic Web Project，并且输入项目名称为 HelloWorldStruts 2，根据下面画面中给出的选项设置其他的选项：



在下面的画面中选择所有的默认选项，最后检查 Generate Web.xml deployment descriptor 选项。这将在 Eclipse 中创建一个动态 web 项目。现在使用 Windows > Show View > Project Explorer，你会看到一些东西在你的项目窗口中，如下所示：



现在从 struts 2 lib 文件夹 C:\struts-2.2.3\lib 中复制下列文件到我们项目的 WEB-INF\lib 文件夹中。为了做到这个，你可以简单地把所有的下列文件拖拽到 WEB-INF\lib 文件夹中。

- ? commons-fileupload-x.y.z.jar
- ? commons-io-x.y.z.jar
- ? commons-lang-x.y.jar
- ? commons-logging-x.y.z.jar
- ? commons-logging-api-x.y.jar
- ? freemarker-x.y.z.jar
- ? javassist-.xy.z.GA
- ? ognl-x.y.z.jar
- ? struts2-core-x.y.z.jar
- ? xwork-core.x.y.z.jar

创建 Action 类

Action 类是 Struts 2 应用程序的关键，并且我们在 action 类中实现大部分的业务逻辑。所以让我们在 `Java Resources > src` 下的包名 `com.tutorialspoint.struts2` 中创建一个 java 文件 `HelloWorldAction.java`，下面给出它的内容。

当用户点击一个 URL 时，Action 类响应用户动作。执行一个或多个 Action 类中的方法并且返回一个字符串结果。基于结果的值，将呈现一个指定的 JSP 页面。

```
package com.tutorialspoint.struts2;
public class HelloWorldAction{
    private String name;
    public String execute() throws Exception {
        return "success";
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

它是一个非常简单的带有一个名为“name”属性的类。对于“name”属性，我们有标准的 getter 和 setter 方法，还有返回字符串“success”的执行方法。

Struts 2 框架将创建一个 HelloWorldAction 类的对象并且为了响应用户的动作调用执行方法。你把业务逻辑放在执行方法中，最后返回字符串常量。简单地说为了每个网址，你将必须实现一个动作类，你可以直接使用这个类名作为你的动作名，或者你也可以使用 `struts.xml` 文件映射到其他名字，如下所示。

创建视图

我们需要一个 JSP 提交最后的信息，当一个预定义的动作发生时，这个页面会被 Struts 2 框架调用，这种映射将被定义在 `struts.xml` 文件中。所以让我们在 Eclipse 项目的 WebContent 文件夹中创建下面的 jsp 文件 `HelloWorld.jsp`。为了做到这个，在项目资源管理器的 WebContent 文件夹上单击右键，选择 `New > JSP File`。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Hello World</title>
</head>
```

```
<body>
  Hello World, <s:property value="name"/>
</body>
</html>
```

标签库指令告诉 Servlet 容器这个页面将使用 Struts 2 的标签，而且这些标签将在 s 之前。s:property 标签显示动作类属性 "name" 的值，它是由 HelloWorldAction 类的 getName() 方法返回的。

创建主页

我们还需要在 WebContent 文件夹中创建 index.jsp。这个文件将作为初始动作 URL，用户可以点击它来告诉 Struts 2 框架调用 HelloWorldAction 类定义的方法，并且呈现给 HelloWorld.jsp 视图。

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Hello World</title>
</head>
<body>
  <h1>Hello World From Struts2</h1>
  <form action="hello">
    <label for="name">Please enter your name</label><br/>
    <input type="text" name="name"/>
    <input type="submit" value="Say Hello"/>
  </form>
</body>
</html>
```

使用 struts.xml 文件将把在上面的视图文件中定义的 hello 动作映射到 HelloWorldAction 类和它的执行方法中。当用户点击提交按钮时，将引起 Struts 2 框架运行在 HelloWorldAction 类中定义的执行方法，根据方法的返回值一个适当的视图将被作为响应进行选择 and 呈现。

配置文件

我们需要一个将 URL，HelloWorldAction 类（模型）和 HelloWorld.jsp（视图）连结在一起的映射。该映射告诉 Struts 2 框架哪个类将响应用户的动作（URL），这个类的哪个方法将被执行，根据方法返回的字符串结果将呈现什么视图。

因此，让我们创建一个名为 struts.xml 的文件。由于 Struts 2 需要在类文件夹下展示 struts.xml。所以在 WebContent/WEB-INF/classes 文件夹下创建 struts.xml 文件。默认的情况下，Eclipse 不会创建 “classes” 文件夹，所以需要自己创建。为了做到这个，在项目资源管理器的 WEB-INF 文件夹上点击右键，并选择 New > Folder。struts.xml 中应该像这样：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<constant name="struts.devMode" value="true" />
    <package name="helloworld" extends="struts-default">
        <action name="hello"
            class="com.tutorialspoint.struts2.HelloWorldAction"
            method="execute">
            <result name="success">/HelloWorld.jsp</result>
        </action>
    </package>
</struts>
```

关于上述配置文件有几点需要注意。在这里我们设置常量 `struts.devMode` 为 `true`，是因为我们正工作在程序开发环境中，我们需要看到一些有用的日志信息。然后，我们定义了一个名为 `helloworld` 的包。当你想要把你的动作分成一组时，创建一个包是有用的。在我们的例子中，我们命名我们的动作为 “hello”，它对应着 URL `/hello.action` 和通过 `HelloWorldAction.class` 进行备份。当 URL `/hello.action` 调用时，`HelloWorldAction.class` 的执行方法是运行的方法。如果执行方法的结果返回 “success”，然后我们把 `HelloWorld.jsp` 呈现给用户。

下一步是创建一个 `web.xml` 文件，它是一个任何对 Struts 2 请求的入口点。Struts2 应用程序的入口点将是一个在部署描述符（`web.xml`）中定义的过滤器。因此，我们将在 `web.xml` 中定义一个 `org.apache.struts2.dispatcher.FilterDispatcher` 类。`web.xml` 文件需要在 WebContent 的 WEB-INF 文件夹下创建。当你创建项目时，Eclipse 已经创建了初始的 `web.xml` 文件。所以，我们只需要修改它，如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">
    <display-name>Struts 2</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <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>
</web-app>
```

我们已经指定 index.jsp 为我们的 welcome 文件。然后我们已经配置了 Struts2 的过滤器来运行所有的 URL（即任何匹配模式 /* 的 URL）。

启用详细日志

你可以启用完整的日志记录功能，而通过在 WEB-INF/classes 文件夹下创建 logging.properties 文件使用 Struts 2 工作。在你的属性文件中保留下面两行语句：

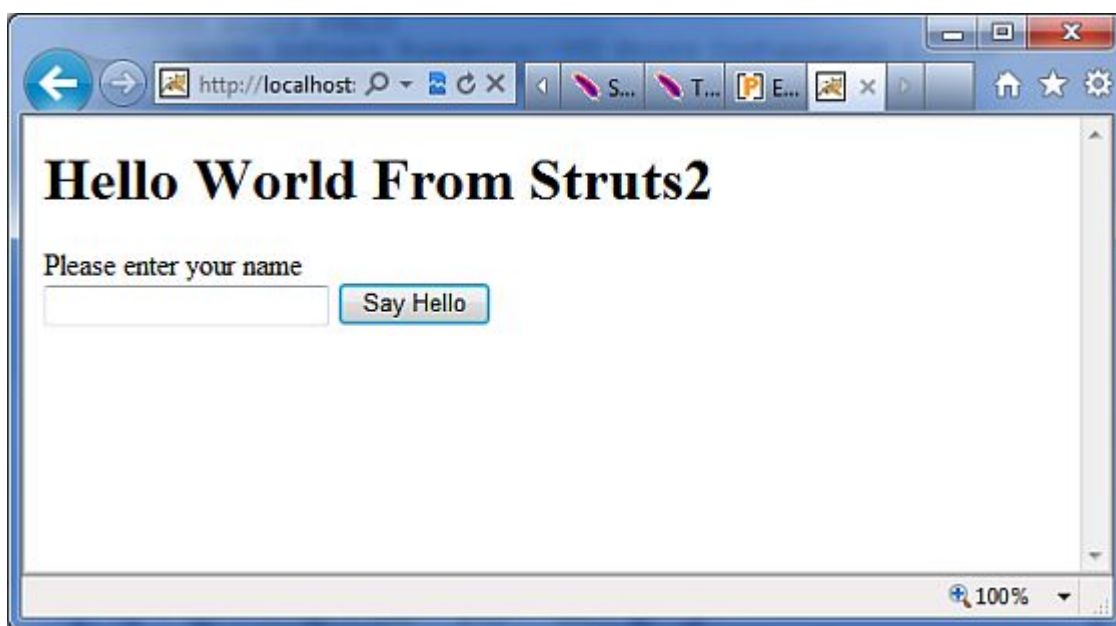
```
org.apache.catalina.core.ContainerBase.[Catalina].level = INFO
org.apache.catalina.core.ContainerBase.[Catalina].handlers = \
    java.util.logging.ConsoleHandler
```

默认的 logging.properties 指定一个 ConsoleHandler 用于把路由记录发送到 stdout，也指定一个 FileHandler。可以使用 SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST 或 ALL 设置处理程序的日志级别的阈值。

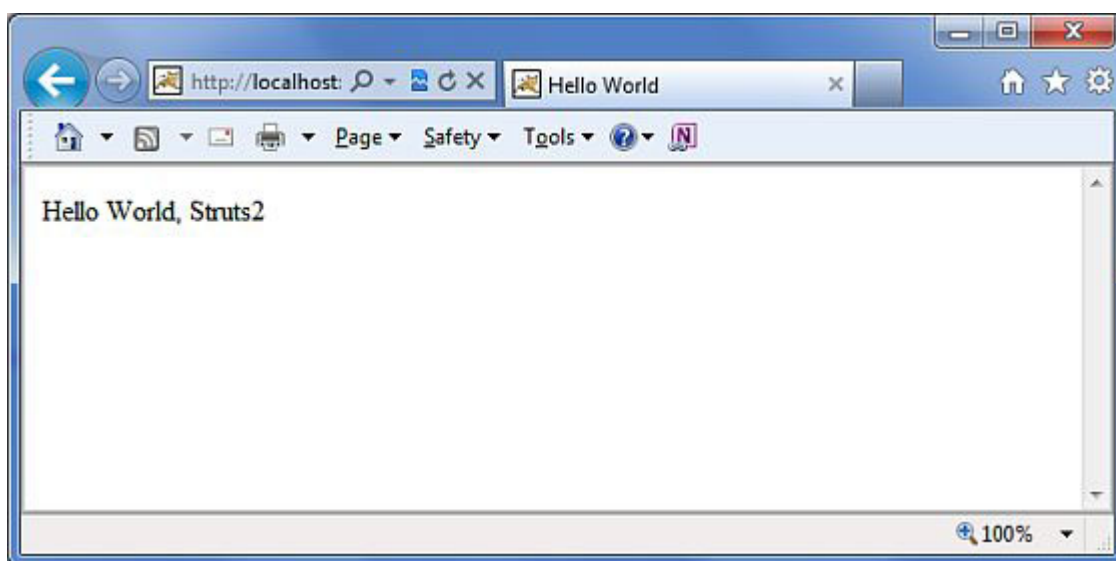
就是这样。我们已经准备好使用 Struts 2 框架来运行我们的 Hello World 应用程序。

执行应用程序

在项目名称上点击右键，并且单击 **Export > WAR File** 来创建一个 War 文件。然后在 Tomcat 的 webapps 目录下部署这个 WAR。最后，启动 Tomcat 服务器和尝试访问 URL `http://localhost:8080/HelloWorldStruts2/index.jsp`。将会给出下面的画面：



输入一个值 “Struts2”，并且提交页面。你应该看到下一个页面：



注意，你可以在 struts.xml 文件中定义索引作为一个动作。在这种情况下，你可以调用索引页面 <http://localhost:8080/HelloWorldStruts2/index.action/> 检查下面如何定义索引作为一个动作：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<constant name="struts.devMode" value="true" />
    <package name="helloworld" extends="struts-default">
        <action name="index">
            <result >/index.jsp</result>
        </action>
    </package>
</struts>
```

```
</action>  
<action name="hello"  
    class="com.tutorialspoint.struts2.HelloWorldAction"  
    method="execute">  
    <result name="success">/HelloWorld.jsp</result>  
</action>  
</package>  
</struts>
```


配置

本章将带你学习一个 Struts 2 应用程序必需的基本配置。在这里，我们将看到一些重要的配置文件：`web.xml`，`struts.xml`，`struts-config.xml` 和 `struts.properties`，它们将被配置。

老实说你可以使用 `web.xml` 和 `struts.xml` 配置文件，在前面的章节中你已经看到了我们的例子使用这两个文件进行工作，但是为了你学习知识，我也解释其他文件。

web.xml 文件

`web.xml` 配置文件是一个 J2EE 的配置文件，它决定如何用 `Servlet` 容器来处理 HTTP 请求的元素。它不是严格意义上的一个 Struts 2 的配置文件，但它是一个 Struts 2 工作时需要被配置的文件。

如前所述，这个文件为任何 web 应用程序提供了一个入口点。Struts 2 应用程序的入口点是一个在部署描述符（`web.xml`）中已定义的过滤器。因此，我们将在 `web.xml` 中定义 `FilterDispatcher` 类的入口。`web.xml` 文件需要在 `WebContent/WEB-INF` 文件夹下创建。

如果你在没有生成配置文件的模板或工具（如 Eclipse 或 Maven2）的帮助下开始，那么 `web.xml` 是你需要配置的第一个配置文件。下面是在我们最后一个例子中使用的 `web.xml` 文件的内容。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
  <display-name>Struts 2</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <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>
</web-app>
```

注意，我们映射 Struts 2 的过滤器为 `/*`，而不是 `/*.action`，这意味着所有的 urls 将被 struts 的过滤器解析。当我们进入注解这一章时，我们将讨论这个。

struts.xml 文件

`struts.xml` 文件包含配置信息，随着动作的开发，你将会修改这些配置信息。这个文件可以用来重写应用程序的默认设置，例如 `struts.devMode = false`，还有定义在属性文件中的其他设置。这个文件可以在文件夹 `** WEB-INF/classes**` 下创建。

让我们来看看在前面的章节中已经解释的 Hello World 例子中创建的 `struts.xml` 文件。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.devMode" value="true" />
    <package name="helloworld" extends="struts-default">
        <action name="hello"
            class="com.tutorialspoint.struts2.HelloWorldAction"
            method="execute">
            <result name="success">/HelloWorld.jsp</result>
        </action>
        <!-- more actions can be listed here -->
    </package>
    <!-- more packages can be listed here -->
</struts>
```

要注意的第一件事是 DOCTYPE。所有的 struts 配置文件需要有正确的 doctype，正如我们的小例子所示。是根标签的元素，在它的下面我们使用 标签声明不同的包。在这里，允许配置的分离和模块化。当你有一个大项目，并且该项目被划分成不同的模块时，它是非常有用的。

也就是说，如果你的项目有三个域 - `business_applicaiton`，`customer_application` 和 `staff_applicatio` n，你可以创建三个包，并且在适当的包中存储相关的动作。包标签具有以下属性：

属性	描述
name (required)	包的唯一标识符。
extends	这个包是由哪个包扩展的?默认情况下,我们使用 struts-default 作为基础包。
abstract	如果标记为 true，对于终端用户消费来说，这个包是不可用的。

namespace	动作的唯一命名空间。
-----------	------------

带着 name 和 value 属性的常量标签将被用于重写任何 default.properties 中定义下面的属性，就如我们刚刚设置 struts.devMode 属性的过程。设置 struts.devMode 属性允许我们在日志文件中看到更多的调试信息。

我们定义对应于每一个我们要访问的 URL 的动作标签，我们定义了一个带有 execute() 方法的类，每当我们访问相应的 URL 时，它将被访问。

在动作被执行后，结果决定把得到的哪些返回给浏览器。从动作中返回的字符串应该是一个结果的名称。同上，每次执行动作，结果都被配置，或者都作为一个“global”的结果，包中的每个动作都是可用的。结果有可选的 name 和 type 属性。默认名称的值为“success”。

随着时间的推移，Struts.xml 文件可以扩展，用包阻止它是一种使它模块化的方式，但是 struts 提供了另一种使 struts.xml 文件模块化的方式。你可以将文件分割成多个 xml 文件，并且用下列的方式导入它们。

```
<?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="my-struts1.xml"/>
    <include file="my-struts2.xml"/>
</struts>
```

我们还没有重写的其他的配置文件是 struts-default.xml。这个文件包含了 Struts 的标准配置设置，你就不必要为了你的项目的 99.99% 修改这些设置。为此，我们打算详细地介绍这个文件。如果你有兴趣，可以在 struts 2-core-2.2.3.jar 文件中有效的 default.properties 文件里看到它。

struts-config.xml 文件

struts-config.xml 配置文件是在 Web 客户端中视图和模型组件之间的链接，但是你不必要为了你的项目的 99.99% 而修改这些设置。基本配置文件包含下面的主要内容：

序号 拦截器 & 描述	
1	struts-config 它是配置文件的根节点。
2	form-beans

	它是你把 ActionForm 子类映射到名称上的位置。你使用这个名字作为 ActionForm 的别名，贯穿 struts-config.xml 的其余部分，甚至在 JSP 页面中。
3	global forwards 这个部分把 web 应用的页面映射到名称上。你可以使用该名称来引用实际的页面。这个避免了在 web 页面上硬编码 URLs。
4	action-mappings 它是你声明表单处理程序的位置，他们也被称为动作映射。
5	controller 这个部分配置 Struts 内部，而且很少在实际情况中使用。
6	plug-in 这个部分告诉 Struts 在哪里找到包含提示和错误信息的属性文件。

下面是示例 struts-config.xml 文件：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">

<struts-config>

  <!-- ===== Form Bean Definitions ===== -->
  <form-beans>
    <form-bean name="login" type="test.struts.LoginForm" />
  </form-beans>

  <!-- ===== Global Forward Definitions ===== -->
  <global-forwards>
  </global-forwards>

  <!-- ===== Action Mapping Definitions ===== -->
  <action-mappings>
    <action
      path="/login"
      type="test.struts.LoginAction" >

      <forward name="valid" path="/jsp/MainMenu.jsp" />
      <forward name="invalid" path="/jsp/LoginView.jsp" />
    </action>
  </action-mappings>

  <!-- ===== Controller Definitions ===== -->
  <controller
    contentType="text/html;charset=UTF-8"
    debug="3"
```

```
maxFileSize="1.618M"  
locale="true"  
nocache="true"/>  
</struts-config>
```

关于 struts-config.xml 文件更多的详细信息，请查看你的 struts 文档。

struts.properties 文件

这个配置文件提供了一种改变框架的默认行为的机制。实际上，包含在 struts.properties 配置文件内的所有属性也可以在 web.xml 中使用 init-param 被配置，同样也可以在 struts.xml 配置文件中使用 constant 标签。但是如果你喜欢保持事情分离和有更多特定的 struts，你就可以在文件夹 WEB-INF/classes 下创建这个文件。

在这个文件中配置的值将重写在 default.properties 中配置的默认值，default.properties 包含在 struts2-core-x.y.z.jar 分布中。这里有几个属性，你可能会考虑使用 struts.properties 文件改变它们：

```
#### When set to true, Struts will act much more friendly for developers  
struts.devMode = true  
  
#### Enables reloading of internationalization files  
struts.i18n.reload = true  
  
#### Enables reloading of XML configuration files  
struts.configuration.xml.reload = true  
  
#### Sets the port that the server is run on  
struts.url.http.port = 8080
```

在这里，任何以井号（#）开始的行会被假定为注释，它将被 Struts 2 忽略。

动作

动作是 Struts 2 框架的核心，因为它们是服务于任何 MVC（模型-视图-控制器）的框架。每个 URL 被映射到一个指定的动作中，它提供了必要的处理逻辑来服务用户的请求。

但是动作也在其他两个重要的能力上起作用。首先，动作在从请求到视图传输数据中起着重要的作用，无论它是一个 JSP 还是其它的结果类型。第二，动作必须协助框架来确定哪些结果应该呈现给视图，该视图在响应中返回给请求。

创建动作

对 Struts 2 中动作的唯一要求是必须有一个无参数方法，该方法返回 String 或结果对象，而且必须是一个 POJO。如果无参数方法没有被指定，那么默认的动作是使用 execute() 方法。

你可以选择扩展 **ActionSupport** 类，它实现了包括动作接口的 6 个接口。动作接口如下所示：

```
public interface Action {
    public static final String SUCCESS = "success";
    public static final String NONE = "none";
    public static final String ERROR = "error";
    public static final String INPUT = "input";
    public static final String LOGIN = "login";
    public String execute() throws Exception;
}
```

让我们来看看 Hello World 例子中的动作方法：

```
package com.tutorialspoint.Struts 2;
public class HelloWorldAction{
    private String name;
    public String execute() throws Exception {
        return "success";
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

为了举例说明动作方法控制着视图，让我们对 `execute` 方法做下面的修改，并且扩展 `ActionSupport` 类，如下所示：

```
package com.tutorialspoint.Struts 2;
import com.opensymphony.xwork2.ActionSupport;
public class HelloWorldAction extends ActionSupport{
    private String name;
    public String execute() throws Exception {
        if ("SECRET".equals(name))
        {
            return SUCCESS;
        }else{
            return ERROR;
        }
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

在这个例子中，我们在 `execute` 方法中有一些逻辑，这些逻辑着眼于 `name` 属性。如果属性等于字符串 “SECRET”，我们就返回 `SUCCESS` 作为结果，否则我们就返回 `ERROR` 作为结果。因为我们已经扩展了 `ActionSupport`，所以我们可以使用字符串常量 `SUCCESS` 和 `ERROR`。现在，让我们修改我们的 `struts.xml` 文件，如下所示：

```
<?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>
    <constant name="struts.devMode" value="true" />
    <package name="helloworld" extends="struts-default">
        <action name="hello"
            class="com.tutorialspoint.Struts 2.HelloWorldAction"
            method="execute">
            <result name="success">/HelloWorld.jsp</result>
            <result name="error">/AccessDenied.jsp</result>
        </action>
    </package>
</struts>
```

创建视图

让我们在 eclipse 项目的 WebContent 文件夹下创建下面的 jsp 文件 HelloWorld.jsp。为了做到这个，在项目资源管理器的 WebContent 文件夹上点击右键，并选择 **New > JSP File**。假如返回的结果是 SUCCESS，则这个文件将被调用，SUCCESS 是定义在动作接口中的一个字符串常量 “success”：

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Hello World</title>
</head>
<body>
  Hello World, <s:property value="name"/>
</body>
</html>
```

假如返回的结果是 ERROR，则下面的文件将被框架调用，ERROR 等于一个字符串常量 “error”。下面是 AccessDenied.jsp 的内容。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Access Denied</title>
</head>
<body>
  You are not authorized to view this page.
</body>
</html>
```

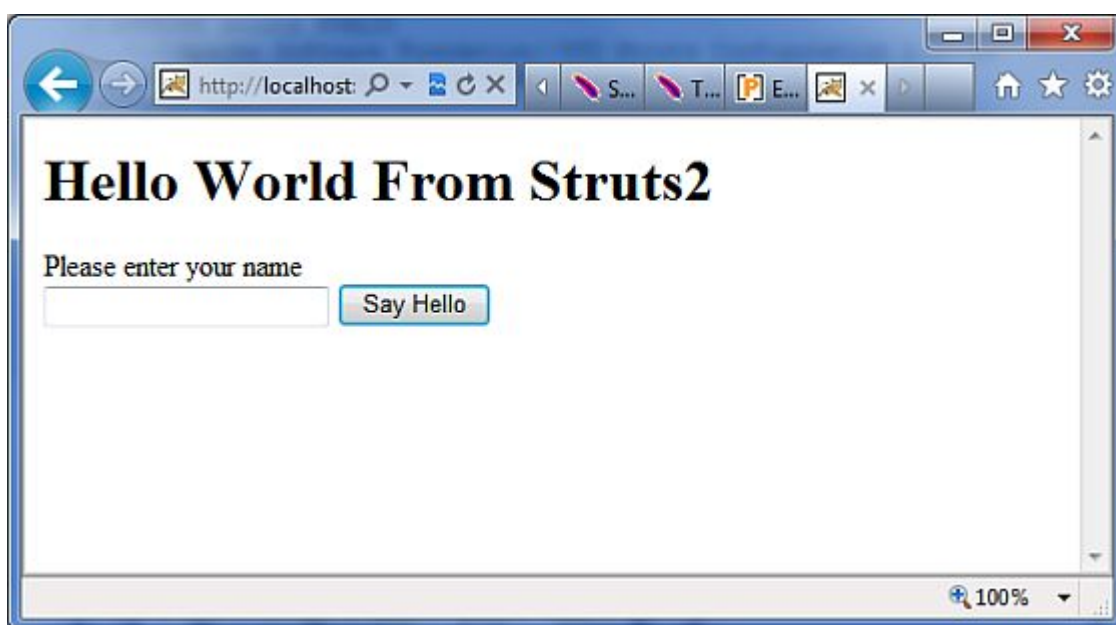
我们还需要在 WebContent 文件夹中创建 index.jsp。这个文件将作为初始动作 URL，用户可以点击它来告诉 Struts 2 框架调用 HelloWorldAction 类的 execute 方法，并且呈现 HelloWorld.jsp 视图。

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Hello World</title>
</head>
<body>
  <h1>Hello World From Struts 2</h1>
  <form action="hello">
    <label for="name">Please enter your name</label><br/>
    <input type="text" name="name"/>
    <input type="submit" value="Say Hello"/>
  </form>
</body>
</html>
```

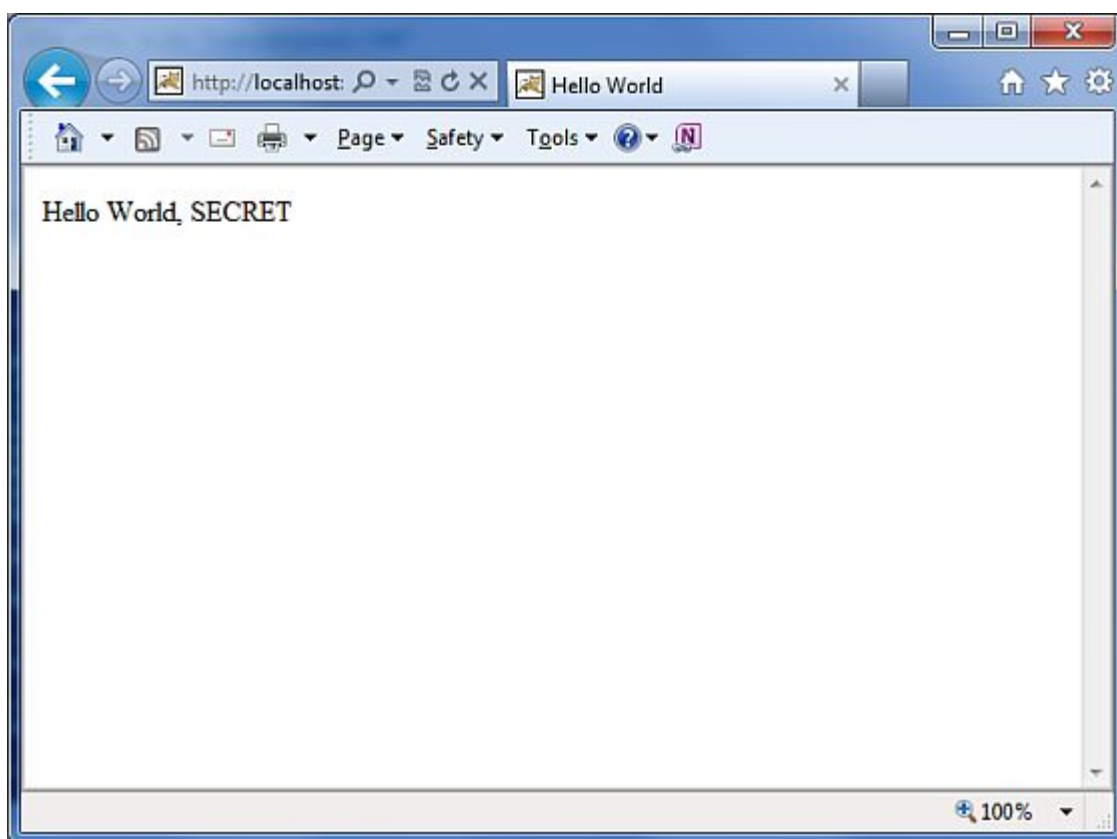

就是这样，不需要改变 web.xml 文件，所以我们使用 Examples 章节中已经创建的同一个 web.xml。现在，我们已经准备好使用 Struts 2 框架来运行我们的 Hello World 应用程序。

执行应用程序

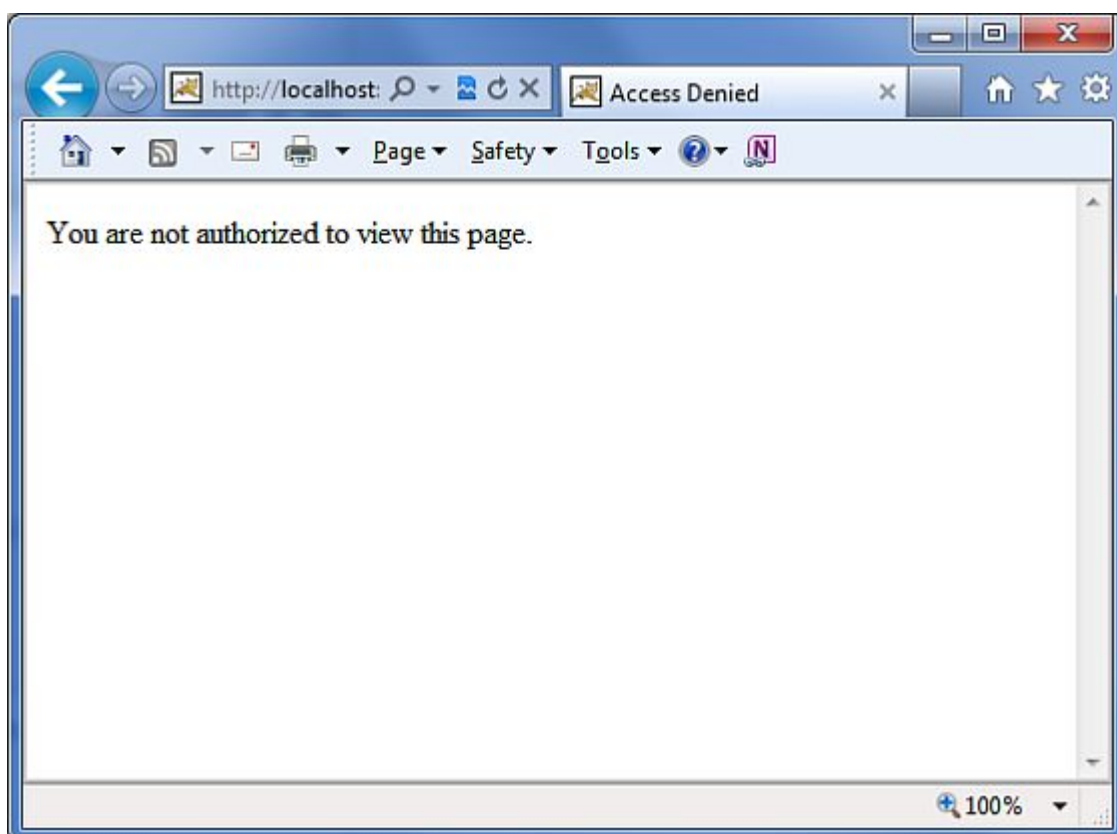
在项目名称上点击右键，并且单击 Export > WAR File 来创建一个 War 文件。然后在 Tomcat 的 webapps 目录中部署这个 WAR。最后，启动 Tomcat 服务器，并且尝试访问 URL `http://localhost:8080/HelloWorldStruts2/index.jsp`。它将会给出下面的画面：



让我们输入一个单词 “SECRET”，你应该可以看到下面的页面：



现在输入除了“SECRET”以外的任何单词，你应该可以看到下面的页面：



创建多个动作

你将经常定义一个以上的动作来处理不同的请求，并且给用户提供不同的 URL，因此你可以定义不同的类，定义如下所示：

```
package com.tutorialspoint.Struts 2;
import com.opensymphony.xwork2.ActionSupport;
class MyAction extends ActionSupport{
    public static String GOOD = SUCCESS;
    public static String BAD = ERROR;
}
public class HelloWorld extends ActionSupport{
    ...
    public String execute()
    {
        if ("SECRET".equals(name)) return MyAction.GOOD;
        return MyAction.BAD;
    }
    ...
}
public class SomeOtherClass extends ActionSupport{
    ...
    public String execute()
    {
        return MyAction.GOOD;
    }
    ...
}
```

你将在 struts.xml 文件中配置这些动作，如下所示：

```
<?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>
<constant name="struts.devMode" value="true" />
<package name="helloworld" extends="struts-default">
    <action name="hello"
        class="com.tutorialspoint.Struts 2.HelloWorld"
        method="execute">
        <result name="success">/HelloWorld.jsp</result>
        <result name="error">/AccessDenied.jsp</result>
```

```
</action>
<action name="something"
  class="com.tutorialspoint.Struts 2.SomeOtherClass"
  method="execute">
  <result name="success">/Something.jsp</result>
  <result name="error">/AccessDenied.jsp</result>
</action>
</package>
</struts>
```

正如你在上述假设的例子中看到的，动作的结果 SUCCESS 和 ERROR 是重复的。为了解决这个问题，建议你创建一个包含输出结果的类。

拦截器

拦截器在概念上和servlet过滤器或JDKs代理类一样。拦截器允许横切功能在动作和框架中单独实现。你可以使用拦截器实现下面的内容：

- ？ 在动作被调用之前提供预处理逻辑。
- ？ 在动作被调用之后提供预处理逻辑。
- ？ 捕获异常，以便可以执行交替处理。

Struts 2 框架提供的许多功能都是使用拦截实现的；例如包括异常处理，文件上传，生命周期回调和验证等。事实上，由于 Struts 2 是许多拦截器功能的基础，所以每次动作不是不可能有 7 个或 8 个拦截器被分配。

Struts 2 框架的拦截器

Struts 2 框架提供了一列开箱即用的拦截器来预先设定和准备使用。下面列出了几个重要的拦截器：

序号	拦截器及描述
1	alias 允许参数有不同的跨请求的别名。
2	checkbox 通过为没有被检查的复选框添加一个参数值 false 来协助管理复选框。
3	conversionError 把从字符串转化为参数类型的错误信息放置到动作的字段错误中。
4	createSession 如果不存在 HTTP 会话，则自动创建一个 HTTP 会话。
5	debugging 为开发人员提供几种不同的调试屏幕。
6	execAndWait

	当动作在后台执行的时候，把用户定向到一个中间的等待页面。
7	exception 映射动作抛出的异常到一个结果中，通过重定向允许自动异常处理。
8	fileUpload 有利于简单的文件上传。
9	i18n 在用户的会话期间，跟踪选定的语言环境。
10	logger 通过输出被执行的动作的名称提供简单的日志。
11	params 设置动作的请求参数。
12	prepare 它通常是用来做预处理工作，如设置数据库连接。
13	profile
14	scope 在会话或应用程序的范围中存储和检索动作的状态。
15	ServletConfig 为行动提供了各种基于 servlet 信息的访问。
16	timer 以动作需要多长时间执行的形式提供了简单的配置信息。
17	token 为有效的标记检查动作用来防止重复地表单提交。
18	validation

为动作提供了验证支持。

关于上面提到的拦截器的完整信息，请查看 Struts 2 的文档。但是我会告诉你通常如何在你的 Struts 应用程序中使用一个拦截器。

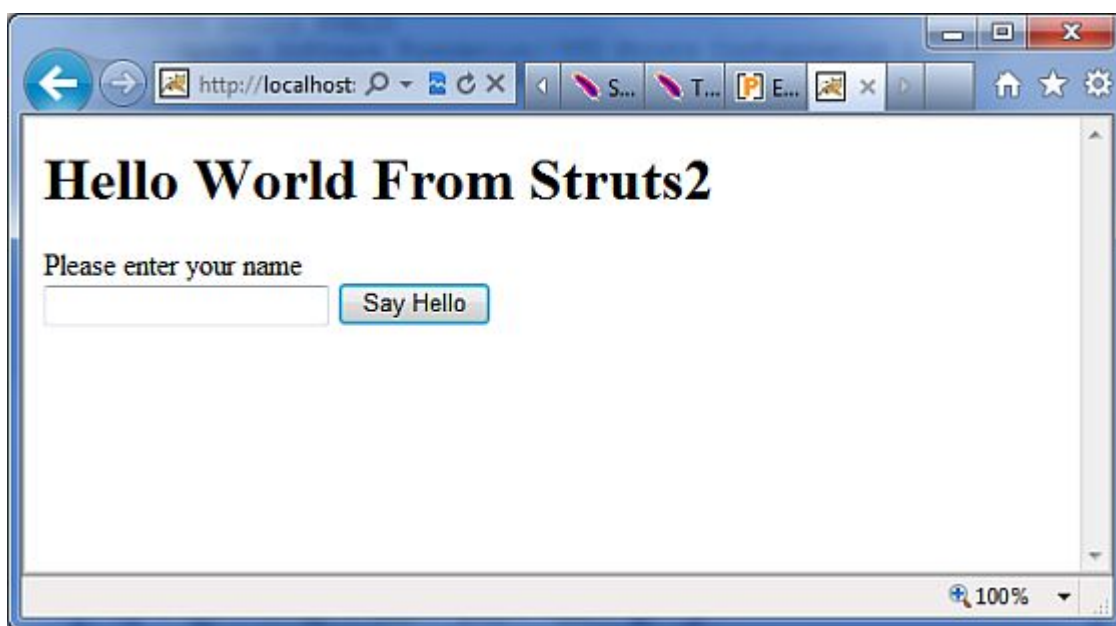
如何使用拦截器？

让我们来看看如何在我们的 “Hello World” 程序中使用已存在的拦截器。我们将使用 **timer** 拦截器，它的目的是测量它多长时间执行一个动作的方法。同时我使用 **params** 拦截器，它的目的是给动作发送请求参数。你可以尝试在你的例子中不使用这个拦截器，你将会发现 **name** 属性没有被设置，因为参数是无法达到动作中的。

我们将保留 HelloWorldAction.java，web.xml，HelloWorld.jsp 和 index.jsp 文件，因为他们已经在 Examples 章节被创建了，但是让我们修改 struts.xml 文件，添加一个拦截器，如下所示：

```
<?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>
  <constant name="struts.devMode" value="true" />
  <package name="helloworld" extends="struts-default">
    <action name="hello"
      class="com.tutorialspoint.Struts 2.HelloWorldAction"
      method="execute">
      <interceptor-ref name="params"/>
      <interceptor-ref name="timer" />
      <result name="success">/HelloWorld.jsp</result>
    </action>
  </package>
</struts>
```

右键单击项目名称，并且单击 **Export > WAR File** 来创建一个 War 文件。然后在 Tomcat 的 webapps 目录下部署这个 WAR。最后，启动 Tomcat 服务器和尝试访问 URL `http://localhost:8080/HelloWorldStruts 2/index.jsp`。将会给出下面的画面：



现在，在给定的文本框中输入任何单词，并且单击 Say Hello 按钮执行已定义的动作。现在，如果你查看生成的日志，就会发现下面的文字：

```
INFO: Server startup in 3539 ms
27/08/2011 8:40:53 PM
com.opensymphony.xwork2.util.logging.commons.CommonsLogger info
INFO: Executed action [/hello!execute] took 109 ms.
```

在这里最后一行是因为 timer 拦截器生成成长的，它告诉动作被执行的时间的 109ms。

创建自定义的拦截器

在你的应用程序中使用自定义的拦截器是一种提供横切的应用功能的简洁的方式。创建一个自定义的拦截器是很容易的，需要扩展的接口是下面的 `Interceptor` 接口：

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

正如名称所显示的，`init()` 方法提供了一种初始化拦截器的方法，而 `destroy()` 方法提供了一种清理拦截器的工具。与动作不同的是，拦截器在请求之间被重用，而且需要是线程安全的，尤其是 `intercept()` 方法。

`ActionInvocation` 对象提供运行时环境的访问。它允许访问动作本身和方法来调用该动作，并判定动作是否已经被调用。

如果你不需要初始化或清理代码，可以扩展 `AbstractInterceptor` 类。它提供了一个对 `init()` 和 `destroy()` 方法的默认的空操作实现。

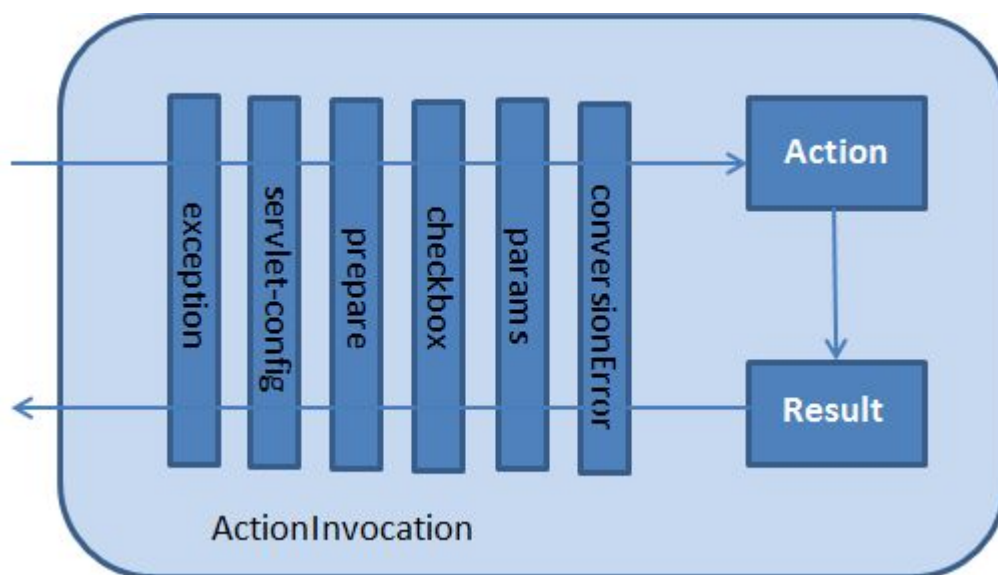
创建拦截器类

让我们在 `Java Resources > src` 文件夹中创建下面的 `MyInterceptor.java`：

```
package com.tutorialspoint.Struts 2;
import java.util.*;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;
public class MyInterceptor extends AbstractInterceptor {
    public String intercept(ActionInvocation invocation)throws Exception{
        /* let us do some pre-processing */
        String output = "Pre-Processing";
        System.out.println(output);
        /* let us call action or next interceptor */
        String result = invocation.invoke();
        /* let us do some post-processing */
        output = "Post-Processing";
        System.out.println(output);
        return result;
    }
}
```

如你注意到的，实际的动作将通过使用拦截器调用 `invocation.invoke()` 来执行。所以，你可以根据你的需求做一些预处理和一些后处理。

这个框架本身通过第一次调用 `ActionInvocation` 对象的 `invoke()` 来启动过程。每次 `invoke()` 被调用，`ActionInvocation` 查询它的状态，并且执行接下来的拦截器。当所有已配置的拦截器已经被配置时，`invoke()` 方法将引发这个动作本身被执行。下面的图通过请求流显示了相同的概念：



创建动作类

让我们在 Java Resources > src 中名为 com.tutorialspoint.Struts 2 的包下创建一个 java 文件 HelloWorld Action.java，它的内容在下面给出。

```
package com.tutorialspoint.Struts 2;
import com.opensymphony.xwork2.ActionSupport;
public class HelloWorldAction extends ActionSupport{
    private String name;
    public String execute() throws Exception {
        System.out.println("Inside action....");
        return "success";
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

我们已经在前面的例子中看到这个相同的类。我们对于 “name” 属性有标准的 getters 和 setters 方法，还有返回字符串 “success” 的 execute 方法。

创建视图

让我们在 eclipse 项目的 WebContent 文件夹中创建下面的 jsp 文件 `helloWorld.jsp`。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Hello World</title>
</head>
<body>
  Hello World, <s:property value="name"/>
</body>
</html>
```

创建主页面

我们还需要在 WebContent 文件夹中创建 `index.jsp`。这个文件将作为初始动作 URL，用户可以点击它告诉 Struts 2 框架调用 `HelloWorldAction` 类定义的方法，并且呈现 `HelloWorld.jsp` 视图。

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Hello World</title>
</head>
<body>
  <h1>Hello World From Struts 2</h1>
  <form action="hello">
    <label for="name">Please enter your name</label><br/>
    <input type="text" name="name"/>
    <input type="submit" value="Say Hello"/>
  </form>
</body>
</html>
```

上面的视图文件中定义的 `hello` 动作将使用 `struts.xml` 文件映射到 `HelloWorldAction` 类和它的 `execute` 方法中。

配置文件

现在，我们需要注册我们的拦截器，然后调用它，就如我们已经在前面的例子中调用了默认的拦截器一样。为了注册一个新定义的拦截器，可以把 `...` 标签直接放置在 `struts.xml` 文件 标签内。对于默认的拦截器，你可以跳过这一步，就像我们在前面的例子中一样。但在这里让我们注册和使用它，如下所示：

```

<?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>
    <constant name="struts.devMode" value="true" />
    <package name="helloworld" extends="struts-default">

        <interceptors>
            <interceptor name="myinterceptor"
                class="com.tutorialspoint.Struts 2.MyInterceptor" />
        </interceptors>

        <action name="hello"
            class="com.tutorialspoint.Struts 2.HelloWorldAction"
            method="execute">
            <interceptor-ref name="params"/>
            <interceptor-ref name="myinterceptor" />
            <result name="success">/HelloWorld.jsp</result>
        </action>

    </package>
</struts>

```

应该注意的是，你可以在 `<interceptors>` 标签内注册多个拦截器，并且同时你可以在 `<interceptor-ref>` 标签内调用多个拦截器。你可以用不同的动作调用相同的拦截器。

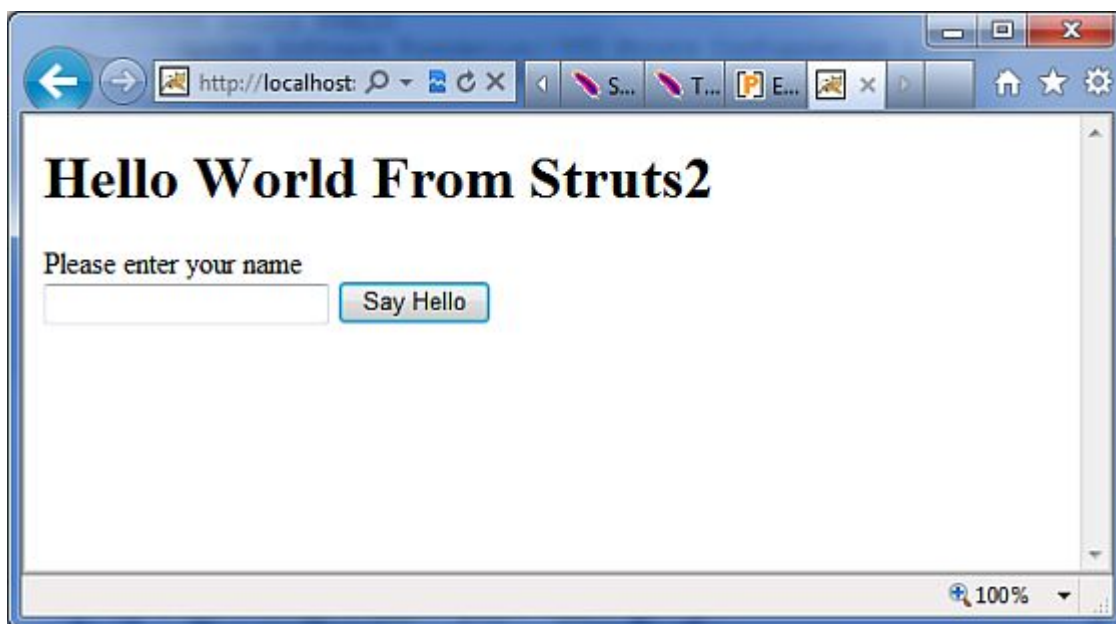
web.xml 文件需要在 WebContent 的 WEB-INF 文件夹下创建，如下所示：

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">
    <display-name>Struts 2</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <filter>
        <filter-name>Struts 2</filter-name>
        <filter-class>
            org.apache.Struts 2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>Struts 2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>

```

右键单击项目名称，并且单击 **Export > WAR File** 来创建一个 War 文件。然后在 Tomcat 的 webapps 目录下部署这个 WAR。最后，启动 Tomcat 服务器和尝试访问 URL `http://localhost:8080/HelloWorldStruts2/index.jsp/` 将会给出下面的画面：



现在，在给定的文本框中输入任何单词，并且单击 Say Hello 按钮执行已经定义的动作。现在，如果你查看生成的日志，就会在底部发现下面的文字：

```
Pre-Processing
Inside action....
Post-Processing
```

堆叠多个拦截器

可想而知，为每个动作配置多个拦截器很快就会变得非常难以管理。为此，拦截器用拦截器栈进行管理。这儿有一个例子，直接来自 struts-default.xml 文件：

```
<interceptor-stack name="basicStack">
  <interceptor-ref name="exception"/>
  <interceptor-ref name="servlet-config"/>
  <interceptor-ref name="prepare"/>
  <interceptor-ref name="checkbox"/>
  <interceptor-ref name="params"/>
  <interceptor-ref name="conversionError"/>
</interceptor-stack>
```

上面的栈被称为 **basicStack**，它可以用在你的配置中，如下所示。这个配置节点被放置在 `<package.../>` 节点下。每个 `<interceptor-ref.../>` 标签引用一个拦截器或在当前的拦截器栈之前已配置的拦截器栈。因此，当配置初始的拦截器和拦截器栈时，确保在所有拦截器和拦截器栈配置中这个名称是唯一的，这是非常重要的。

我们已经看到了如何应用拦截器到动作中，应用拦截器栈是没有什么不同的。实际上，我们完全使用相同的标签：

```
<action name="hello" class="com.tutorialspoint.Struts 2.MyAction">
  <interceptor-ref name="basicStack"/>
  <result>view.jsp</result>
</action>
```

上述注册的“basicStack”将完整地注册所有带有 hello 动作的 6 个拦截器。应该指出的是，拦截器按照已配置的顺序执行。例如，在上述情况下，将首先执行异常，第二个执行的是 servlet 配置等等。

结果类型

正如前面提到的，标签在 Struts 2 的 MVC 框架中担当 视图的角色。动作是负责执行业务逻辑。在执行业务逻辑之后，下一步是使用 标签显示视图。

经常有一些附带结果的导航规则。例如，如果一个动作方法是对用户进行验证，那么有三种可能的结果。

- (a) 成功的登录；
- (b) 不成功的登录-用户名或密码错误；
- (c) 帐户被锁定。

在这种情况下，这个动作方法将使用三种可能的结果字符串被配置，并且有三个不同的视图呈现的结果。我们已经在前面的例子中看到了这个。

但是，Struts 2 没有阻碍你使用 JSP 作为视图技术。毕竟 MVC 范例的整个目的是保持层是独立和高度可配置的。例如，对于一个 Web2.0 的客户端，你可能想要返回 XML 或 JSON 作为输出。在这种情况下，你可以为 XML 或 JSON 创建一个新的结果类型，并且实现它。

Struts 自带一些预定义的结果类型，无论什么我们已经看到的都是默认的结果类型 `dispatcher`，它是用来调度 JSP 页面的。Struts 允许你为了视图技术使用其它标记语言来呈现结果和流行的选择，包括 Velocity，Freemarker，XSLT 和 Tiles。

调度结果类型

调度 结果类型是默认的类型，如果没有其他的结果类型被指定，则使用它。它被用来在服务器上转发到一个 servlet，JSP，HTML 页面，等等。它使用 `RequestDispatcher.forward()` 方法。

在我们前面的例子中，我们看到了 “shorthand” 版本，其中我们提供了 JSP 路径作为结果标签的主体。

```
<result name="success">
  /HelloWorld.jsp
</result>
```

我们也可以在 `<result...>` 元素内使用一个 标签来指定 JSP 文件，如下所示：

```
<result name="success" type="dispatcher">
  <param name="location">
    /HelloWorld.jsp
  </param>
</result>
```

我们也可以提供一个 `parse` 参数，默认为 `true`。解析参数决定位置参数是否将被解析为 OGNL 表达式。

FreeMaker 结果类型

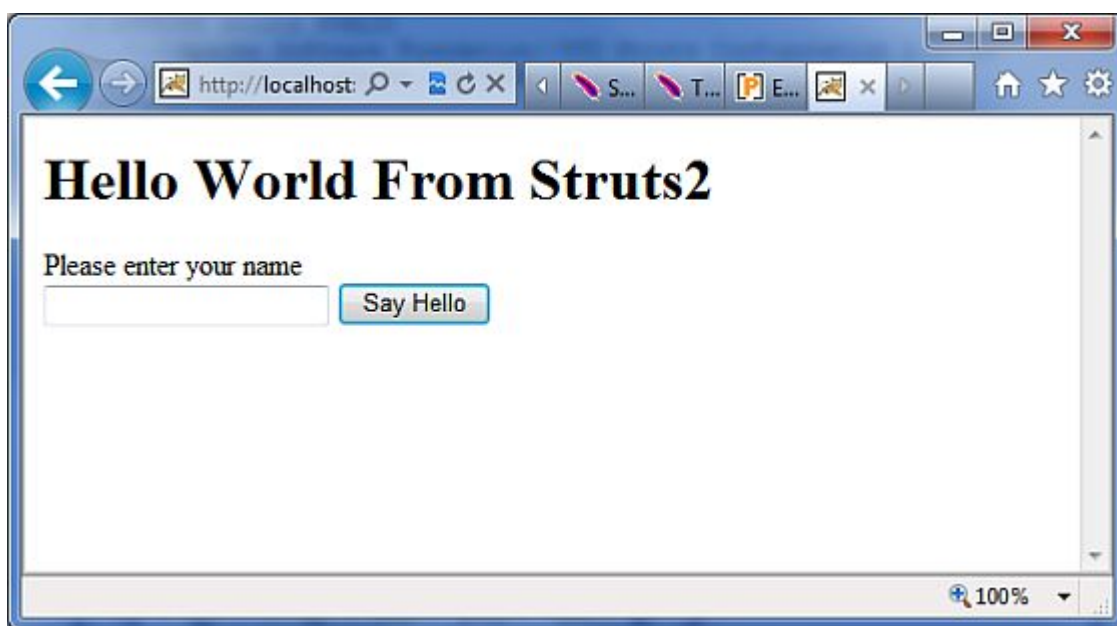
在这个例子中，我们将看到如何使用 FreeMaker 作为视图技术。Freemaker 是一种流行的模板引擎，用于使用预定义的模板来生成输出。让我们创建一个称为 `hello.fm` 的 Freemake 模板一个文件，它的内容如下所示：

```
Hello World ${name}
```

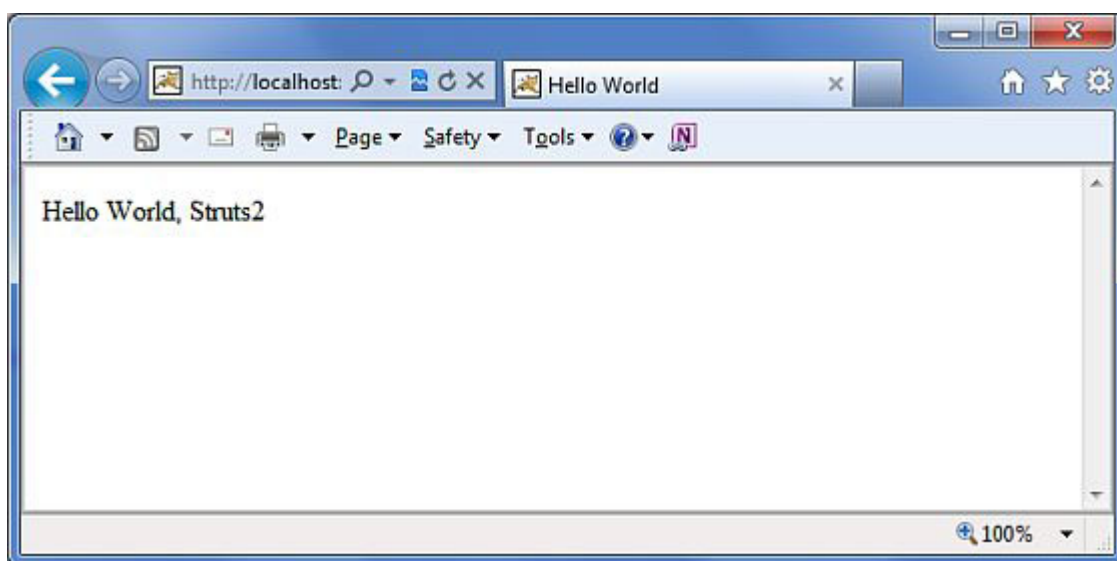
在这里，上述文件是一个模板，其中 `name` 是使用已定义的动作从外部传递的一个参数。你将会在 CLASSPATH 中保留这个文件。接下来，让我们修改 `struts.xml` 来指定结果，如下所示：

```
<?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>
  <constant name="struts.devMode" value="true" />
  <package name="helloworld" extends="struts-default">
    <action name="hello"      class="com.tutorialspoint.Struts 2.HelloWorldAction"
      method="execute">
      <result name="success" type="freemarker">
        <param name="location">/hello.fm</param>
      </result>
    </action>
  </package>
</struts>
```

让我们保留 `HelloWorldAction.java`，`HelloWorldAction.jsp` 和 `index.jsp` 文件，因为我们在 `examples` 章节中已经创建了他们。现在，在项目名称上点击右键，并且单击 **Export > WAR File** 来创建一个 War 文件。然后在 Tomcat 的 `webapps` 目录下部署这个 WAR。最后，启动 Tomcat 服务器和尝试访问 URL `http://localhost:8080/HelloWorldStruts 2/index.jsp`。将会给出下面的画面：



输入一个值 “Struts 2”，并提交该页面，你应该看到下一个页面：



正如你所看到的，这是与 JSP 视图完全一样的，除了我们不依赖于使用 JSP 作为视图技术。在这个例子中，我们已经使用 Freemaker。

重定向结果类型

重定向的结果类型调用标准的 `response.sendRedirect()` 方法，使浏览器给指定的位置创建一个新的请求。

我们可以在 `<result...>` 元素的元素体中或作为一个元素来提供位置。重定向也支持 `parse` 参数。这儿是一个使用 XML 配置的例子：

```
<action name="hello"
  class="com.tutorialspoint.Struts 2.HelloWorldAction"
  method="execute">
  <result name="success" type="redirect">
    <param name="location">
      /NewWorld.jsp
    </param >
  </result>
</action>
```

因此，仅仅修改你的 struts.xml 文件来定义如上所提及的重定向类型，并且创建一个新的文件 NewWorld.jsp，无论何时 hello 动作返回 success，你将会重定向到那里。

值栈/OGNL

值栈

值栈是一个几个对象的集合，根据提供的顺序保持下列的对象：

序号	对象及描述
1	Temporary Objects 在页面执行期间，有各种各样的临时对象被创建。例如，在一个 JSP 标签中集合中循环的当前迭代值。
2	The Model Object 如果你在struts应用程序中使用模型对象，在动作之前当前模型对象被放置到值栈中。
3	The Action Object 它是被执行的当前动作对象。
4	Named Objects 这些对象包括 #application，#session，#request，#attr 和 #parameters，并且适用于相应的 servlet 范围。

值栈可以通过 JSP，Velocity 或者 Freemarker 提供的标签来访问。我们在单独章节学到的各种各样的标签被用来获取和设置 Struts 2.0 的值栈。你可以在你的动作中得到值栈对象，如下所示：

```
ApplicationContext.getContext().getValueStack()
```

一旦你有了值栈对象，就可以用下面的方法来操作这个对象：

序号	值栈方法及描述
1	Object findValue(String expr) 通过用默认搜索顺序对堆栈计算给定的表达式找到一个值。
2	CompoundRoot getRoot()

	获取 CompoundRoot，它保存压入堆栈中的对象。
3	Object peek() 在不改变栈的情况下，获取栈顶的对象。
4	Object pop() 获取栈顶的对象，并且把它从栈顶移除。
5	void push(Object o) 把这个对象放进栈顶中。
6	void set(String key, Object o) 如果一个对象被 findValue(key,...) 检索，则在堆栈上使用给定的键设置它。
7	void setDefaultType(Class defaultType) 当获得值时，如果没有提供任何类型，则设置默认的类型转换。
8	void setValue(String expr, Object value) 试图根据默认的搜索顺序在堆栈上使用给定的表达式设置一个 bean 的属性。
9	int size() 获取堆栈中对象的数量。

OGNL

对象图形导航语言（OGNL）是一个强大的表达式语言，它是用来在值栈上参考和操作数据。OGNL 也有助于数据传输和类型转换。

OGNL 与 JSP 表达式语言非常类似。OGNL 是以在上下文中有跟对象或默认对象的概念为基础的。默认对象或根对象的属性可以使用的标记符号英镑来引用。

正如前面所提到的，OGNL 是基于上下文的，而 Struts 使用 OGNL 来构建 ActionContext 映射。ActionContext 映射由以下组成：

? application – 应用范围的变量

- ? session – 会话范围的变量
- ? root / value stack – 所有的动作变量都保存在这里
- ? request – 请求范围的变量
- ? parameters – 请求参数
- ? attributes – 在页面，请求，会话和应用范围内存储属性

理解这一点很重要，即动作对象在值栈中始终是可用的。所以，因此如果你的动作对象有 x 和 y 属性，那么它们是随时可供你使用的。

ActionContext 中的对象使用英镑符号来引用，然而，值栈中的对象可以被直接引用，例如如果 employee 是一个动作类的属性，那么它就可以被引用，如下所示：

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

代替

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

如果在会话中有一个名为 “login” 的属性，你就可以检索它，如下所示：

```
<s:property value="#session.login"/>
```

OGNL 还支持处理集合,即 Map, List 和 Set。例如，为了显示颜色的下拉列表，你可以这样做：

```
<s:select name="color" list="{ 'red','yellow','green' }" />
```

OGNL 表达式巧妙地解释 “red”，“yellow”，“green” 为颜色，并且在它的基础上建立一个列表。

当我们学习不同的标签时，OGNL 表达式将被广泛使用在接下来的章节中。因此，让我们在 Form 标签 / 控制标签 / 数据标签和 Ajax 标签中使用一些例子来看它，而不是孤立地看他们。

值栈/OGNL例子

创建动作

让我们考虑我们访问值栈的下面的动作类，然后设置在视图即 JSP 页面上使用 OGNL 访问的几个键。

```
package com.tutorialspoint.struts2;
import java.util.*;
import com.opensymphony.xwork2.util.ValueStack;
```

```

import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class HelloWorldAction extends ActionSupport{
    private String name;
    public String execute() throws Exception {
        ValueStack stack = ActionContext.getContext().getValueStack();
        Map<String, Object> context = new HashMap<String, Object>();
        context.put("key1", new String("This is key1"));
        context.put("key2", new String("This is key2"));
        stack.push(context);
        System.out.println("Size of the valueStack: " + stack.size());
        return "success";
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}

```

实际上，当动作执行时，Struts 2 将它添加到值栈的顶部。所以，把东西放在值栈中通常的方法是为你的动作类的值添加 getter/setter 方法，然后使用 标签来访问值。但是，我将给你展示在 struts 中 ActionContext 和 ValueStack 如何正确地工作。

创建视图

让我们在 eclipse 项目的 WebContent 文件夹下创建下面的 jsp 文件 HelloWorld.jsp。在动作返回 success 的情况下，这个视图将被显示：

```

<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Hello World</title>
</head>
<body>
    Entered value : <s:property value="name"/><br/>
    Value of key 1 : <s:property value="key1" /><br/>
    Value of key 2 : <s:property value="key2" /> <br/>
</body>
</html>

```

我们还需要在 WebContent 文件夹下创建的 index.jsp，其内容如下所示：

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>

```

```

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Hello World</title>
</head>
<body>
<h1>Hello World From Struts2</h1>
<form action="hello">
  <label for="name">Please enter your name</label><br/>
  <input type="text" name="name"/>
  <input type="submit" value="Say Hello"/>
</form>
</body>
</html>

```

配置文件

下面是 `struts.xml` 文件的内容:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name="struts.devMode" value="true" />
  <package name="helloworld" extends="struts-default">
    <action name="hello"      class="com.tutorialspoint.struts2.HelloWorldAction"
      method="execute">
      <result name="success">/HelloWorld.jsp</result>
    </action>
  </package>
</struts>

```

下面是 `web.xml` 文件的内容:

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
  <display-name>Struts 2</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <filter>
    <filter-name>struts2</filter-name>

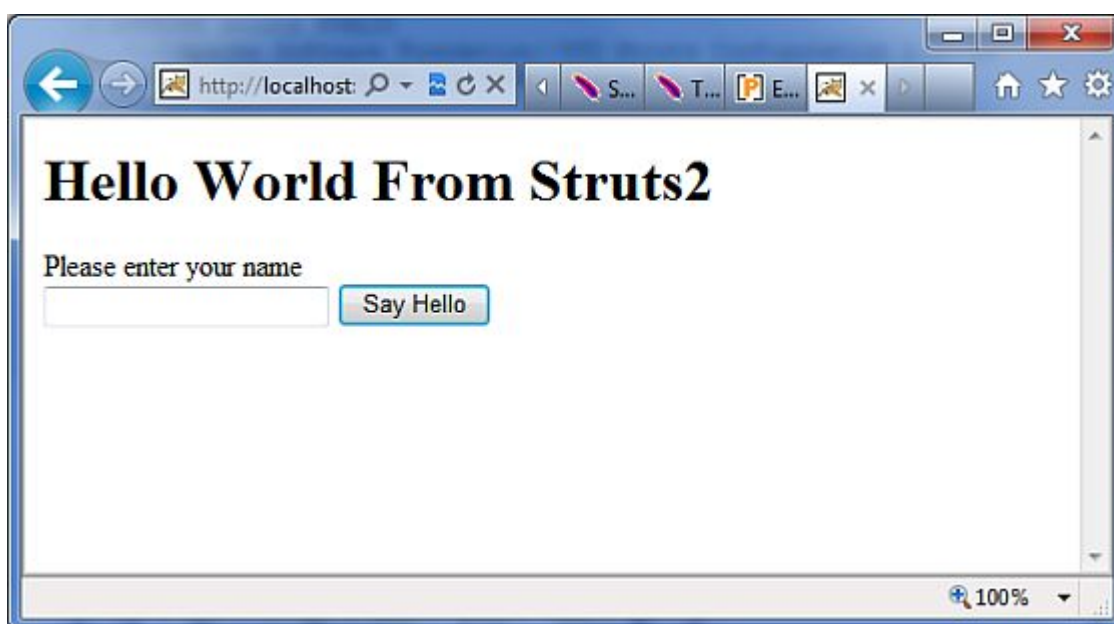
```

```

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

```

在项目名称上点击右键，并且单击 **Export > WAR File** 来创建一个 War 文件。然后在 Tomcat 的 webapps 目录下部署这个 WAR。最后，启动 Tomcat 服务器和尝试访问 URL `http://localhost:8080/HelloWorldStruts2/index.jsp`。将会给出下面的画面：



现在，在给定的文本框中输入任何单词，并且单击“Say Hello”按钮执行已经定义的动作。现在，如果你查看生成的日志，就会在底部发现下面的文字：

```
Size of the valueStack: 3
```

无论你输入什么值，它都将显示下面的画面，我们已经把 key1 和 key2 的值放入了值栈中。

文件上传

Struts 2 框架为处理文件上传提供了内置支持，它使用“在 HTML 中基于表单的文件上传”。当上传一个文件时，它通常会被存储在一个临时目录中，而且它们应该由 Action 类进行处理或移动到一个永久的目录，用来确保数据不丢失。

注意服务器在恰当的位置可能有一个安全策略，它会禁止你写到除了临时目录以外的目录，而且这个目录属于你的 web 应用应用程序。

通过预定义的名为文件上传的拦截器，Struts 的文件上传是可能的，这个拦截器在 `org.apache.struts2.interceptor.FileUploadInterceptor` 类是可用的，而且是 `defaultStack` 的一部分。你仍然可以使用在 `struts.xml` 中设置各种参数，我们将在下面看到。

创建视图文件

让我们开始创建需要浏览和上传选定的文件的视图。因此，让我们创建一个带有简单的 HTML 上传表单的 `index.jsp`，它允许用户上传文件：

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>File Upload</title>
</head>
<body>
  <form action="upload" method="post" enctype="multipart/form-data">
    <label for="myFile">Upload your file</label>
    <input type="file" name="myFile" />
    <input type="submit" value="Upload"/>
  </form>
</body>
</html>
```

在上面的例子中有几点值得注意。首先，表单的编码类型设置为 `multipart/form-data`。为了使用文件上传拦截器来成功地处理文件上传，它应该被设置。下一个注意点是表单的动作方法 `upload` 和文件上传字段的名称是 `myFile`。我们需要这些信息来创建动作方法和 `struts` 配置。

接下来让我们创建一个简单的 `jsp` 文件 `success.jsp` 来显示我们的文件上传的结果，假使它成功地执行。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
```

```
<head>
<title>File Upload Success</title>
</head>
<body>
You have successfully uploaded <s:property value="myFileFileName"/>
</body>
</html>
```

下面是结果文件 error.jsp，假使在上传文件过程中有一些错误：

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>File Upload Error</title>
</head>
<body>
There has been an error in uploading the file.
</body>
</html>
```

创建 action 类

接下来让我们创建一个称为 `uploadFile.java` 的 Java 类，它负责上传文件，并且把这个文件存储在一个安全的位置：

```
package com.tutorialspoint.struts2;
import java.io.File;
import org.apache.commons.io.FileUtils;
import java.io.IOException;
import com.opensymphony.xwork2.ActionSupport;
public class uploadFile extends ActionSupport{
    private File myFile;
    private String myFileContentType;
    private String myFileFileName;
    private String destPath;
    public String execute()
    {
        /* Copy file to a safe location */
        destPath = "C:/apache-tomcat-6.0.33/work/";
        try{
            System.out.println("Src File name: " + myFile);
            System.out.println("Dst File name: " + myFileFileName);
            File destFile = new File(destPath, myFileFileName);
            FileUtils.copyFile(myFile, destFile);
        }catch(IOException e){
            e.printStackTrace();
            return ERROR;
        }
    }
}
```

```

    return SUCCESS;
}
public File getMyFile() {
    return myFile;
}
public void setMyFile(File myFile) {
    this.myFile = myFile;
}
public String getMyFileContentType() {
    return myFileContentType;
}
public void setMyFileContentType(String myFileContentType) {
    this.myFileContentType = myFileContentType;
}
public String getMyFileFileName() {
    return myFileFileName;
}
public void setMyFileFileName(String myFileFileName) {
    this.myFileFileName = myFileFileName;
}
}

```

uploadFile.java 是一个非常简单的类。值得注意的一件重要的事情是文件上传拦截器和参数拦截器为我们处理所有繁重的工作。文件上传拦截器在默认情况下有三个可用的参数。它们被命名为下列模式：

- ？ [your file name parameter] – 这是实际的用户已经上传的文件。在这个例子中它是 “myFile”。
- ？ [your file name parameter]ContentType – 这是被上传的文件的内容类型。在这个例子中它是 “myFileContentType”。
- ？ [your file name parameter]FileName – 这是被上传的文件的名称。在这个例子中它是 “myFileFileName”。

对我们来说，这三个参数是可用的，这要归功于 Struts 的拦截器。我们要做的是在我们的动作类中用正确的名称创建三个参数，而且这些变量是为我们自动连线的。所以，在上面的例子中，我们三个参数和一个动作方法，如果一切正常，则简单地返回 “success”，否则返回 “error”。

配置文件

下面是可以控制文件上传过程的 Struts2 的配置属性：

序号	属性 & 描述
----	---------

1	<code>struts.multipart.maxSize</code> 当一个文件上传时，可以接受的最大的文件大小(以字节为单位)。默认设置是 250 M。
2	<code>struts.multipart.parser</code> 用于上传多个表单的库。默认情况下是 jakarta。
3	<code>struts.multipart.saveDir</code> 存储临时文件的位置。默认情况下是 <code>javax.servlet.context.tempdir</code> 。

为了改变这些设置，你可以使用在你的应用程序 `struts.xml` 文件中 `constant` 标签，就像我改变上传文件的最大大小的操作。让我们有如下所示的 `struts.xml`：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name="struts.devMode" value="true" />
  <constant name="struts.multipart.maxSize" value="1000000" />
  <package name="helloworld" extends="struts-default">
    <action name="upload" class="com.tutorialspoint.struts2.uploadFile">
      <result name="success">/success.jsp</result>
      <result name="error">/error.jsp</result>
    </action>
  </package>
</struts>
```

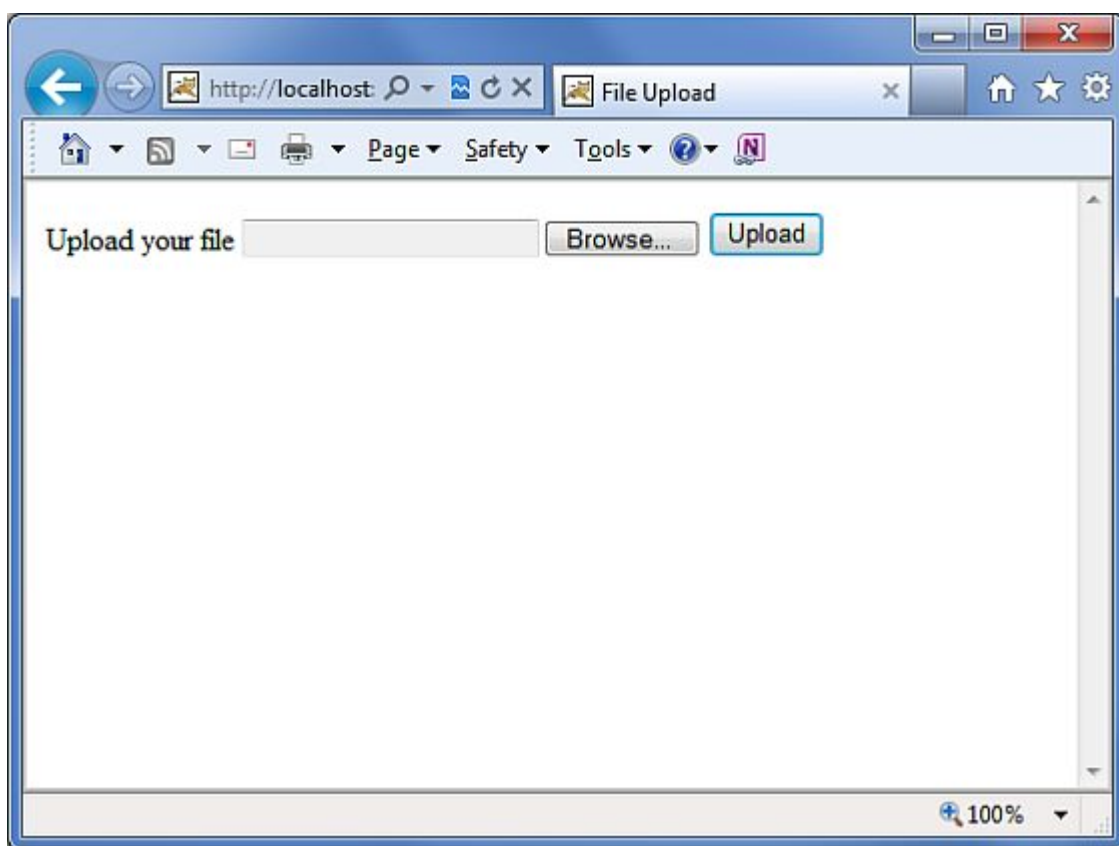
由于拦截器是拦截器 `defaultStack` 的一部分，我们并不需要明确地配置它。但是你可以在 里面添加 标签。文件上传拦截器需要两个参数：（a）`maximumSize`（b）`allowedTypes`。`maximumSize` 参数设置允许的最大文件大小（默认为约 2 MB）。`allowedTypes` 参数是一个逗号分隔的公认的内容（MIME）类型列表，如下所示：

```
<action name="upload" class="com.tutorialspoint.struts2.uploadFile">
  <interceptor-ref name="basicStack">
    <interceptor-ref name="fileUpload">
      <param name="allowedTypes">image/jpeg,image/gif</param>
    </interceptor-ref>
  </interceptor-ref>
  <result name="success">/success.jsp</result>
  <result name="error">/error.jsp</result>
</action>
```

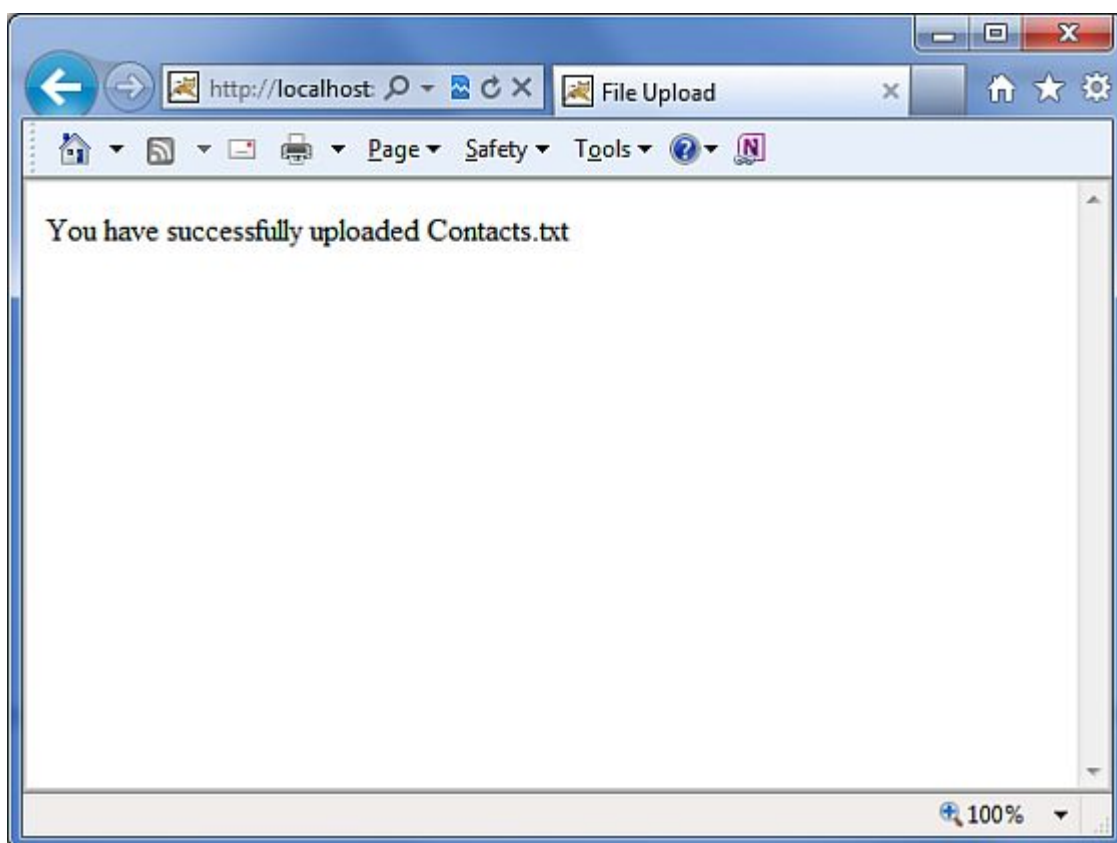
下面是 web.xml 文件的内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
  <display-name>Struts 2</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <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>
</web-app>
```

现在，在项目名称上点击右键，并且单击 **Export > WAR File** 来创建一个 War 文件。然后在 Tomcat 的 webapps 目录下部署这个 WAR。最后，启动 Tomcat 服务器和尝试访问 URL `http://localhost:8080/HelloWorldStruts2/upload.jsp`。将会给出下面的画面：



现在使用浏览按钮选择一个文件 “Contacts.txt”，而且单击上传按钮，它将在你的服务器上上传文件，然后你应该看到页面。你可以查看保存在 C:\apache-tomcat-6.0.33\work 中上传的文件。



注意文件上传拦截器自动删除上传的文件，所以你需要通过编程在某个位置上保存上传的文件，在它被删除之前。

错误消息

文件上传拦截器使用几个默认的错误消息键：

序号	错误消息键 & 描述
1	<code>struts.messages.error.uploading</code> 一个发生在文件无法上传时的通常的错误。
2	<code>struts.messages.error.file.too.large</code> 发生在上传文件比指定的 <code>maximumSize</code> 大时。
3	<code>struts.messages.error.content.type.not.allowed</code> 发生在上传文件不匹配指定的预期内容类型。

你可以重写在 `WebContent/WEB-INF/classes/messages.properties` 源文件中消息的内容。

数据库访问

本章将用简单的步骤教你如何使用 Struts 2 来访问数据库。Struts 是一个 MVC 框架，而不是一个数据库框架，但它为 JPA/Hibernate 集成提供了很好的支持。我们将在后面的章节中看到 Hibernate 集成，但时在本章中我们将使用普通的 JDBC 来访问数据库。

本章中的第一步是设置和准备我们的数据库。在这个例子中，我使用 MySQL 作为我的数据库。我已经在我的机器上安装了 MySQL，并且创建了一个新的数据库，称为 “struts_tutorial”。我创建了一个表，称为 **login**，并且用一些值填充它。下面是用来创建和填充表的脚本。

MYSQL 数据库默认的用户名是 “root”，密码为 “root123”。

```
CREATE TABLE `struts_tutorial`.`login` (  
  `user` VARCHAR( 10 ) NOT NULL ,  
  `password` VARCHAR( 10 ) NOT NULL ,  
  `name` VARCHAR( 20 ) NOT NULL ,  
  PRIMARY KEY ( `user` )  
) ENGINE = InnoDB;  
INSERT INTO `struts_tutorial`.`login` (`user`, `password`, `name`)  
VALUES ('scott', 'navy', 'Scott Burgemott');
```

下一步是下载 [MySQL Connector jar](#) 文件，并把这个文件放在你的项目的 WEB-INF\lib 文件夹下。在我们已经做到了这个之后，现在就可以准备创建动作类。

创建动作

动作类有对应于数据库表中的列的属性。我们把 **user**，**password** 和 **name** 作为字符串属性。在动作方法中，我们使用 **user** 和 **password** 参数来检查用户是否存在，如果存在，我们在下一个画面中显示用户名。如果用户输入了错误的信息，我们再次把他们发送到登录画面。下面是 **LoginAction.java** 文件的内容：

```
package com.tutorialspoint.struts2;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import com.opensymphony.xwork2.ActionSupport;  
public class LoginAction extends ActionSupport {  
  private String user;  
  private String password;  
  private String name;
```



```

public String execute() {
    String ret = ERROR;
    Connection conn = null;
    try {
        String URL = "jdbc:mysql://localhost/struts_tutorial";
        Class.forName("com.mysql.jdbc.Driver");
        conn = DriverManager.getConnection(URL, "root", "root123");
        String sql = "SELECT name FROM login WHERE";
        sql+=" user = ? AND password = ?";
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, user);
        ps.setString(2, password);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            name = rs.getString(1);
            ret = SUCCESS;
        }
    } catch (Exception e) {
        ret = ERROR;
    } finally {
        if (conn != null) {
            try {
                conn.close();
            } catch (Exception e) {
            }
        }
    }
    return ret;
}

public String getUser() {
    return user;
}

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

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getName() {
    return name;
}

public void setName(String name) {

```

```

    this.name = name;
}
}

```

创建主页面

现在，让我们创建一个 JSP 文件 `index.jsp`，用来收集用户名和密码。将对数据库进行检查这个用户名和密码。

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Login</title>
</head>
<body>
<form action="loginaction" method="post">
    User:<br/><input type="text" name="user"/><br/>
    Password:<br/><input type="password" name="password"/><br/>
    <input type="submit" value="Login"/>
</form>
</body>
</html>

```

创建视图

现在，让我们创建 `success.jsp` 文件，假如动作返回 SUCCESS，该文件将被调用，但是假如动作返回 ERROR，我们将有另一个视图。

```

<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Successful Login</title>
</head>
<body>
    Hello World, <s:property value="name"/>
</body>
</html>

```

下面将是一个视图文件 `error.jsp`，假如动作返回 ERROR。

```

<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Invalid User Name or Password</title>
</head>
<body>
    Wrong user name or password provided.

```

```
</body>
</html>
```

配置文件

最后，让我们使用 struts.xml 配置文件把一切都综合起来，如下所示：

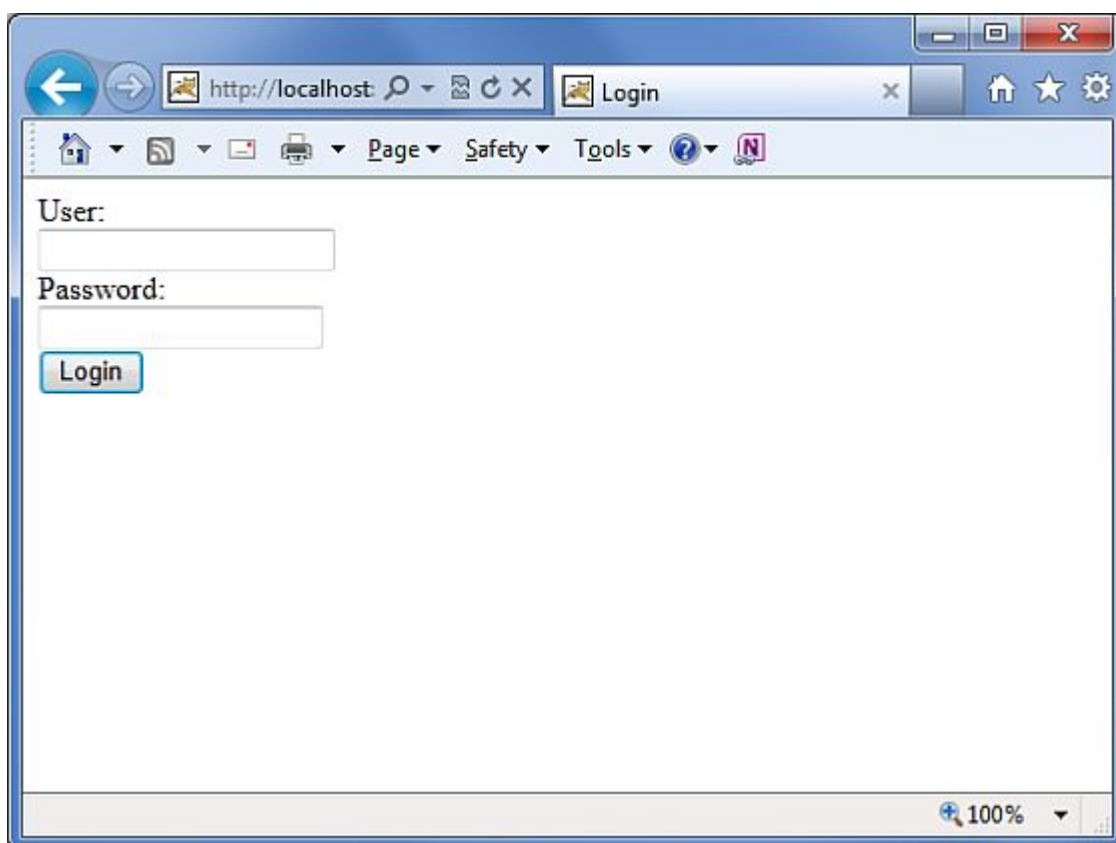
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name="struts.devMode" value="true" />
  <package name="helloworld" extends="struts-default">
    <action name="loginaction"
      class="com.tutorialspoint.struts2.LoginAction"
      method="execute">
      <result name="success">/success.jsp</result>
      <result name="error">/error.jsp</result>
    </action>
  </package>
</struts>
```

下面是 web.xml 文件的内容：

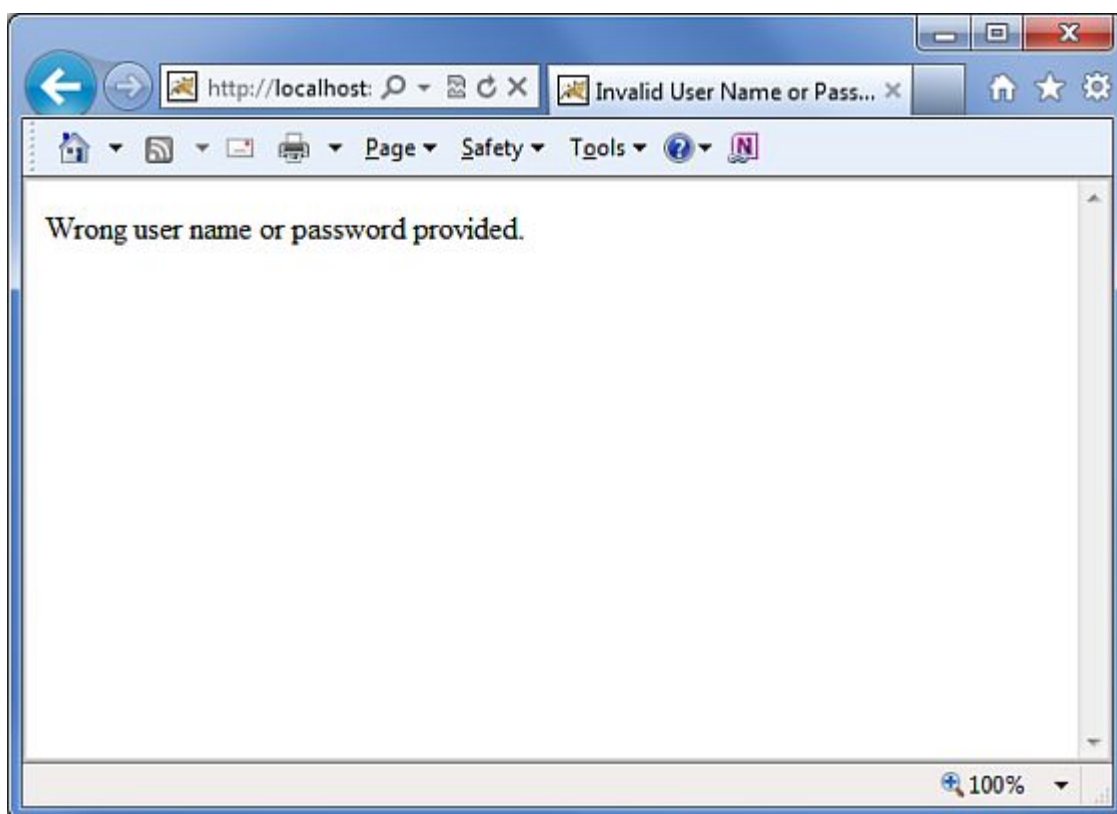
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
  <display-name>Struts 2</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <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>  
</web-app>
```

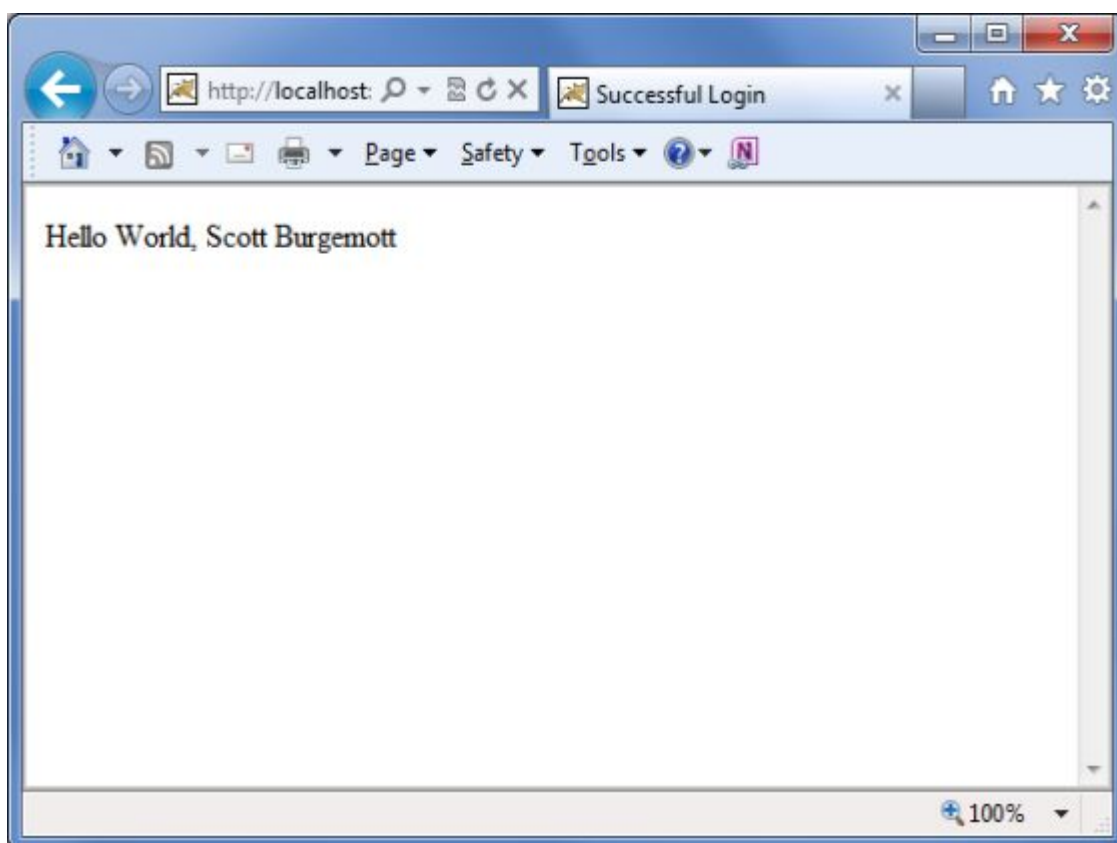
现在，在项目名称上点击右键，并且单击 **Export > WAR File** 来创建一个 War 文件。然后在 Tomcat 的 `webapps` 目录下部署这个 WAR。最后，启动 Tomcat 服务器并尝试访问 URL `http://localhost:8080/HelloWorldStruts2/index.jsp`，将会给出下面的画面：



输入了错误的用户名和密码。你应该看到下面的页面：



现在输入用户名为 scott 和密码为 navy。你应该看到下面的页面：



Struts 2 – 发送邮件

本章将教你如何使用 Struts 2 的应用程序发送电子邮件。为了这个练习，你需要从 [JavaMail API 1.4.4](#) 下载并安装 mail.jar，并将 mail.jar 文件放置在你的 WEB-INF\lib 文件夹下，然后继续按照创建动作，视图和配置文件的标准步骤进行。

创建动作

下一步是创建一个发送电子邮件的动作方法。让我们创建一个新类，称为 `Emailer.java`，它的内容如下。

```
package com.tutorialspoint.struts2;
import java.util.Properties;
import javax.mail.Message;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import com.opensymphony.xwork2.ActionSupport;
public class Emailer extends ActionSupport {
    private String from;
    private String password;
    private String to;
    private String subject;
    private String body;
    static Properties properties = new Properties();
    static
    {
        properties.put("mail.smtp.host", "smtp.gmail.com");
        properties.put("mail.smtp.socketFactory.port", "465");
        properties.put("mail.smtp.socketFactory.class",
            "javax.net.ssl.SSLSocketFactory");
        properties.put("mail.smtp.auth", "true");
        properties.put("mail.smtp.port", "465");
    }
    public String execute()
    {
        String ret = SUCCESS;
        try
        {
            Session session = Session.getDefaultInstance(properties,
```

```

        new javax.mail.Authenticator() {
            protected PasswordAuthentication
            getPasswordAuthentication() {
                return new
                PasswordAuthentication(from, password);
            }
        });
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress(from));
        message.setRecipients(Message.RecipientType.TO,
            InternetAddress.parse(to));
        message.setSubject(subject);
        message.setText(body);
        Transport.send(message);
    }
    catch(Exception e)
    {
        ret = ERROR;
        e.printStackTrace();
    }
    return ret;
}

public String getFrom() {
    return from;
}

public void setFrom(String from) {
    this.from = from;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getTo() {
    return to;
}

public void setTo(String to) {
    this.to = to;
}

public String getSubject() {
    return subject;
}

public void setSubject(String subject) {
    this.subject = subject;
}
}

```

```

public String getBody() {
    return body;
}
public void setBody(String body) {
    this.body = body;
}
public static Properties getProperties() {
    return properties;
}
public static void setProperties(Properties properties) {
    Emailer.properties = properties;
}
}

```

正如上面的源代码中看到的，Emailer.java 有对应于下面给出的email.jsp页面中的表单属性的属性。这些属性是：

- ? from – 发件人的电子邮件地址。由于我们使用的是谷歌的 SMTP，因此我们需要一个有效的 gtalk id。
- ? password – 上述帐户的密码
- ? to – 给谁发送电子邮件?
- ? Subject – 电子邮件的主题
- ? body – 实际的电子邮件消息

我们有没有考虑过上述属性的任何验证，验证将会在下一章中添加。现在让我们看看 execute() 方法。execute() 方法通过使用提供的参数用 javax 邮件库发送一封电子邮件。如果邮件发送成功，动作返回 SUCCESS，否则它返回 ERROR。

创建主页面

让我们编写主页 JSP 文件 index.jsp，它将被用来收集电子邮件中上面提到的相关信息：

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Email Form</title>
</head>
<body>
    <em>The form below uses Google's SMTP server.
    So you need to enter a gmail username and password
    </em>
    <form action="emailer" method="post">

```



```

<label for="from">From</label><br/>
<input type="text" name="from"/><br/>
<label for="password">Password</label><br/>
<input type="password" name="password"/><br/>
<label for="to">To</label><br/>
<input type="text" name="to"/><br/>
<label for="subject">Subject</label><br/>
<input type="text" name="subject"/><br/>
<label for="body">Body</label><br/>
<input type="text" name="body"/><br/>
<input type="submit" value="Send Email"/>
</form>
</body>
</html>

```

创建视图

我们将使用 JSP 文件 `success.jsp`，假如动作返回 SUCCESS，它将被调用，但假如动作返回 ERROR，我们将有另一个视图。

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Email Success</title>
</head>
<body>
    Your email to <s:property value="to"/> was sent successfully.
</body>
</html>

```

下面将是一个视图文件 `error.jsp`，假如动作返回 ERROR。

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Email Error</title>
</head>
<body>
    There is a problem sending your email to <s:property value="to"/>.
</body>
</html>

```

配置文件

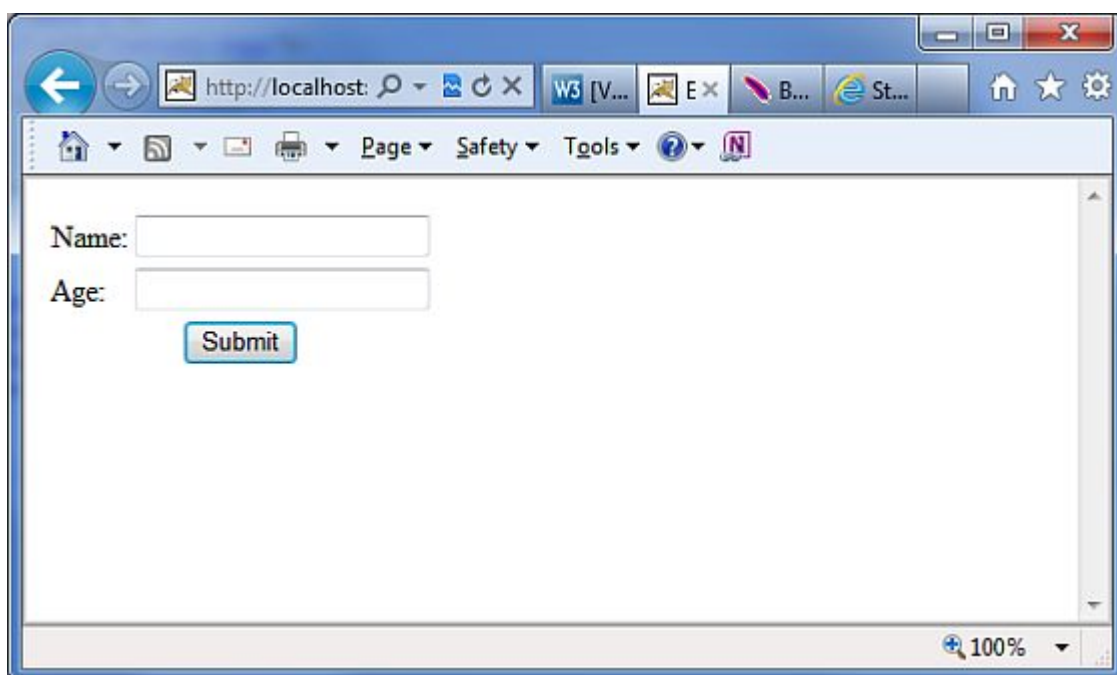
最后，让我们使用 `struts.xml` 配置文件把一切都综合起来，如下所示：

```
<?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>
    <constant name="struts.devMode" value="true" />
    <package name="helloworld" extends="struts-default">
        <action name="emailer"
            class="com.tutorialspoint.struts2.Emailer"
            method="execute">
            <result name="success">/success.jsp</result>
            <result name="error">/error.jsp</result>
        </action>
    </package>
</struts>
```

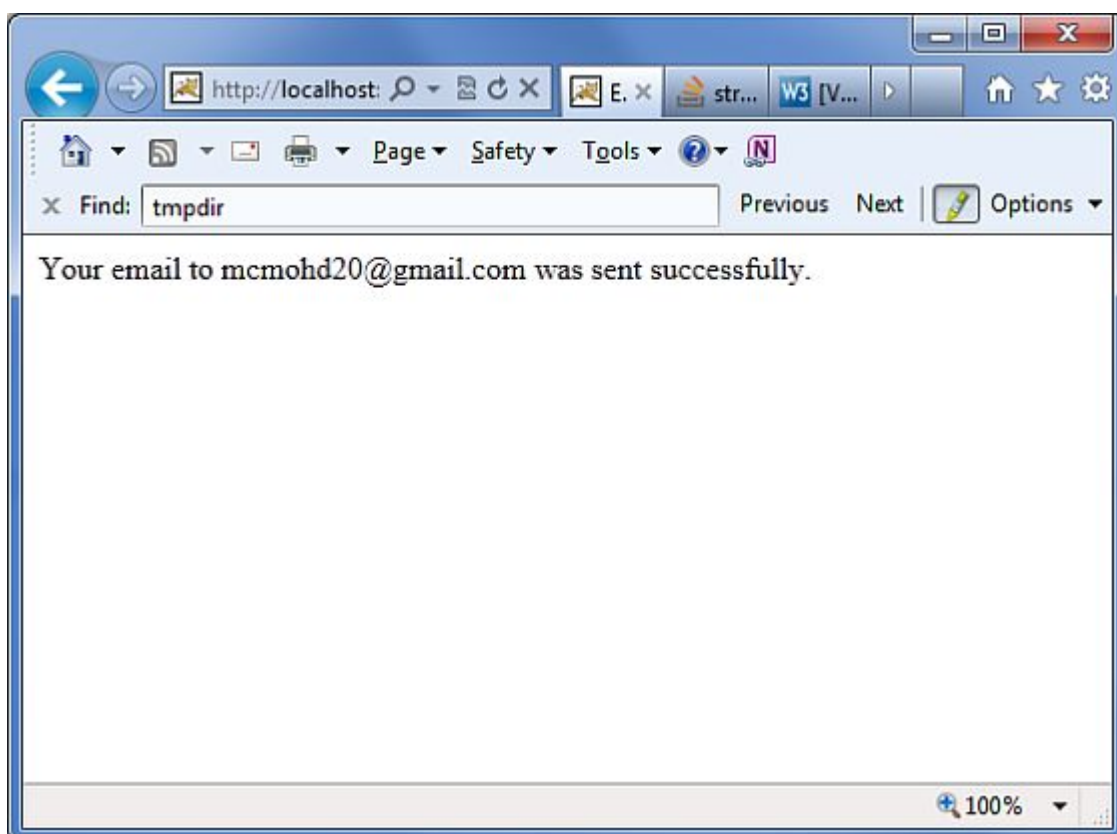
下面是 web.xml 文件的内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">
    <display-name>Struts 2</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <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>
</web-app>
```

现在，在项目名称上点击右键，并且单击 **Export > WAR File** 来创建一个 War 文件。然后在 Tomcat 的 webapps 目录下部署这个 WAR。最后，启动 Tomcat 服务器和尝试访问 URL `http://localhost:8080/HelloWorldStruts2/index.jsp`。将会给出下面的画面：



输入所需的信息，并单击 Send Email 按钮。如果一切正常，那么你应该看到下面的页面：



验证

现在我们将观察 Struts 验证框架如何。在 Struts 的核心中，我们有验证框架，它能在动作方法执行之前协助应用程序运行规则来执行验证。

客户端验证通常是使用 Javascript 来实现的。但是不应该单独依赖于客户端验证。最佳的实践建议验证应该引入到应用程序框架的各个层中。现在，让我们来看看两种在我们的 Struts 项目中添加验证的方式。

这里我们将举一个 Employee 的例子，使用一个简单的页面来捕获它们的姓名和年龄，我们提出两个验证用来确保总是输入一个名字而且年龄应该是在 28 和 65 之间。所以，让我们从例子的主 JSP 页面开始。

创建主页面

让我们编写主页面 JSP 文件 index.jsp，它将被用来收集 Employee 上面提到的相关信息。

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Employee Form</title>
</head>

<body>
<s:form action="empinfo" method="post">
  <s:textfield name="name" label="Name" size="20" />
  <s:textfield name="age" label="Age" size="20" />
  <s:submit name="submit" label="Submit" align="center" />
</s:form>
</body>
</html>
```

index.jsp 使用我们还没有涉及的 Struts 的标签，但是我们将在标签相关的章节学习它们。此时此刻，假设 s:textfield 标签打印一个输入字段，s:submit 打印一个提交按钮。我们已经为每个标签使用 label 属性，而且每个标签都建立了 label。

创建视图

我们将使用 JSP 文件 success.jsp，假如动作返回 SUCCESS，该文件将被调用。

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Success</title>
</head>
<body>
  Employee Information is captured successfully.
</body>
</html>
```

创建动作

因此让我们定义一个小的动作类 `Employee`，然后如下所示在 `Employee.java` 文件中添加一个方法，称为 `validate()`。确保你的动作类扩展了 `ActionSupport` 类，否则你的 `validate` 方法将不会被执行。

```
package com.tutorialspoint.struts2;
import com.opensymphony.xwork2.ActionSupport;
public class Employee extends ActionSupport{
    private String name;
    private int age;
    public String execute()
    {
        return SUCCESS;
    }
    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;
    }
    public void validate()
    {
        if (name == null || name.trim().equals(""))
        {
            addFieldError("name","The name is required");
        }
        if (age < 28 || age > 65)
        {
            addFieldError("age","Age must be in between 28 and 65");
        }
    }
}
```

```
}
}
```

如上面的例子所示，验证方法检查 “Name” 字段是否有一个值。如果没有值被提供，我们为 “Name” 字段添加一个带有自定义错误信息的字段错误。其次，我们检查 “Age” 字段的输入值是否在 28 和 65 之间，如果这个条件不符合，我们添加一个上述验证字段错误。

配置文件

最后，让我们使用 `struts.xml` 配置文件把一切都综合起来，如下所示：

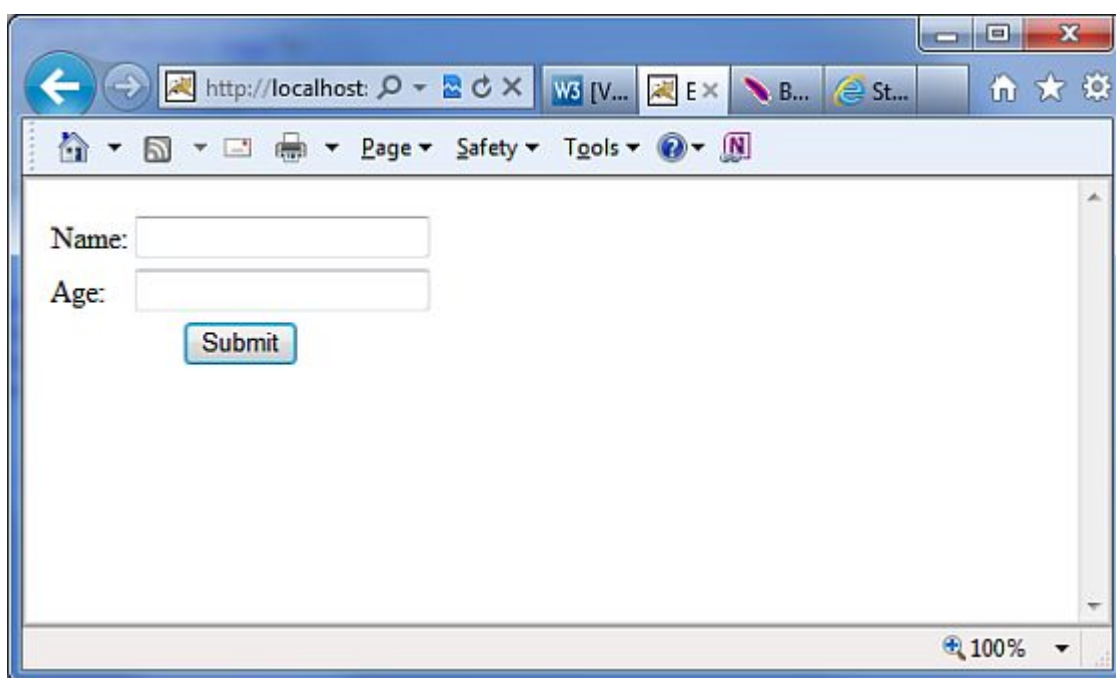
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.devMode" value="true" />
    <package name="helloworld" extends="struts-default">
        <action name="empinfo"
            class="com.tutorialspoint.struts2.Employee"
            method="execute">
            <result name="input">/index.jsp</result>
            <result name="success">/success.jsp</result>
        </action>
    </package>
</struts>
```

下面是 `web.xml` 文件的内容：

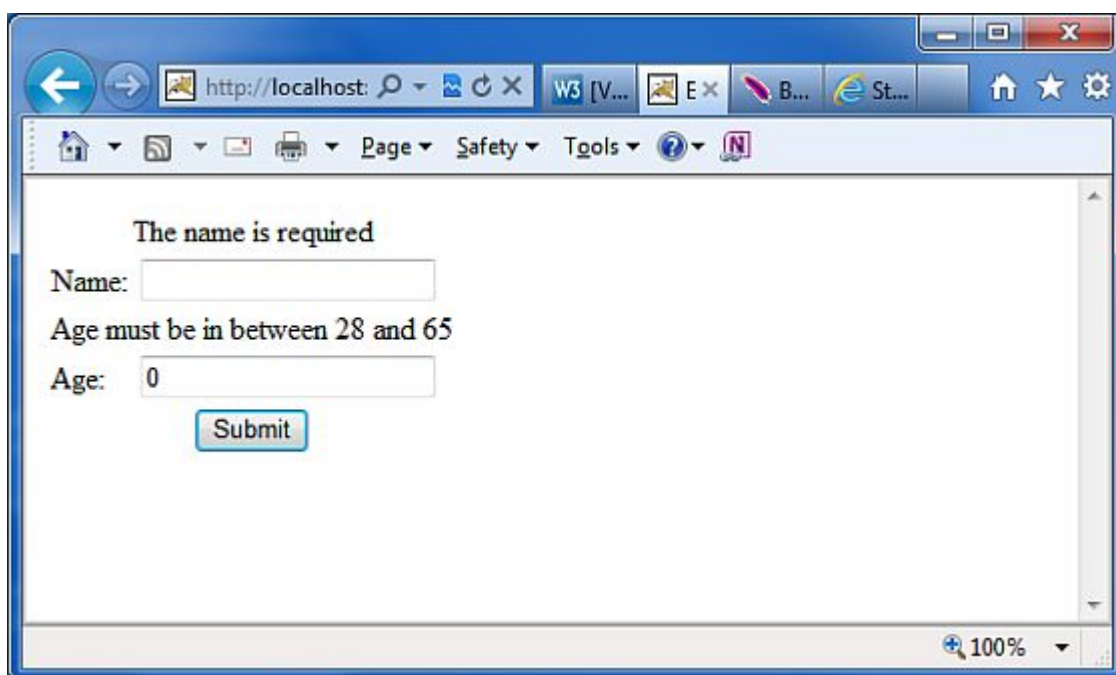
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">
    <display-name>Struts 2</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
```

```
</filter-class>
</filter>
<filter-mapping>
  <filter-name>struts2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
</web-app>
```

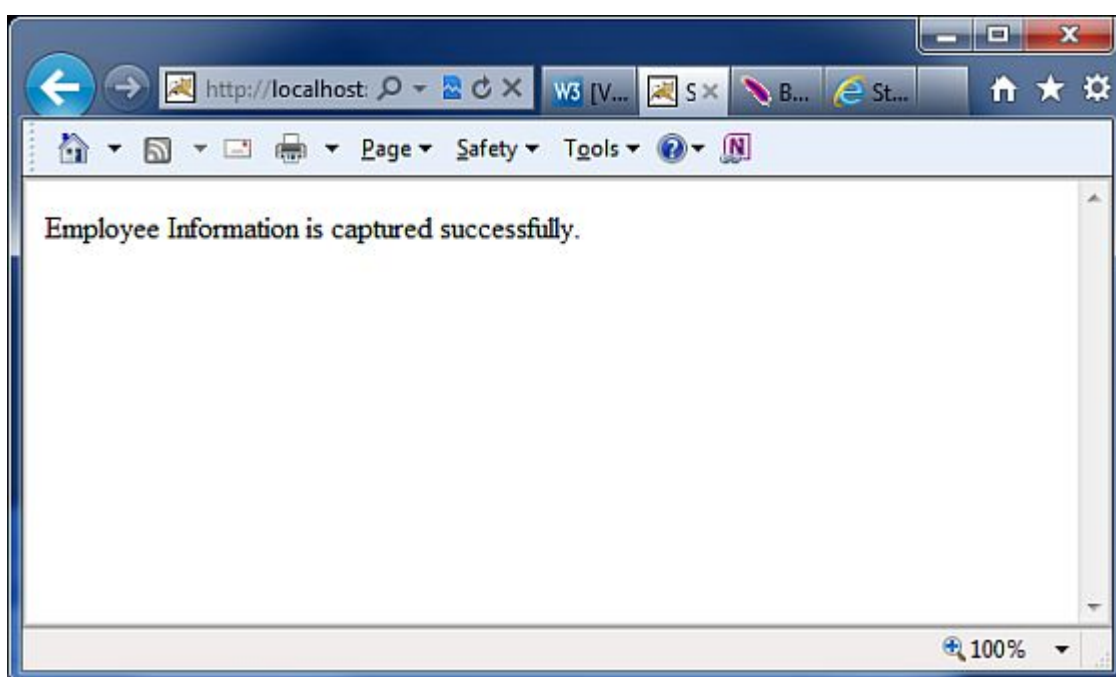
现在，右键点击项目名称，并单击 **Export > WAR File** 创建一个 WAR 文件。然后在 Tomcat 的 webapps 目录下部署此 WAR。最后，启动 Tomcat 服务器和尝试访问 URL `http://localhost:8080/HelloWorldStruts2/index.jsp`。这会给出以下画面：



现在不输入任何必需的信息，只需要单击 **Submit** 按钮。你将看到下面的结果：



输入必需的信息，但是输入了错误的 From 字段，假如说 name 为 “test” 和年龄为 30，最后单击 Submit 按钮。你将看到下面的结果：



验证是如何工作的？

当用户按下提交按钮时，Struts2 会自动执行验证方法，如果任何列出的 if 语句内部的方法都是 true，那么 Struts 2 调用它的 addFieldError 方法。如果已添加任何错误，那么 Struts 2 将不会继续调用 execute 方法。而 Struts 2 框架将返回输入作为调用这个行动的结果。

所以当验证失败和 Struts2 返回输入时，Struts 2 框架将重新显示 index.jsp 文件。由于我们使用了 Struts 2 的表单标签，Struts2 将会自动添加只是上述表单字段的错误信息。

这些错误信息是我们在 addFieldError 方法调用中指定的。addFieldError 方法有两个参数。第一个参数是错误应用的表单字段名，第二个参数是显示上述表单字段的错误信息。

```
addFieldError("name","The name is required");
```

为了处理输入的返回值，我们需要在 struts.xml 添加下面的结果到我们的动作节点。

```
<result name="input">/index.jsp</result>
```

基于XML的验证

验证的第二个方法是通过把一个 xml 文件紧挨着动作类放置。Struts2 基于 XML 验证提供了更多的验证选择，如电子邮件验证，整数范围验证，表单验证字段，表达式验证，正则表达式验证，请求验证，请求字符串验证，字符串长度的验证，等等。

XML 文件需要被命名为 '[action-class]-validation.xml'。所以，在我们的例子中，我们创建一个文件，名为 Employee-validation.xml，它的内容如下所示：

```
<!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="required">
      <message>
        The name is required.
      </message>
    </field-validator>
  </field>
  <field name="age">
    <field-validator type="int">
      <param name="min">29</param>
      <param name="max">64</param>
      <message>
        Age must be in between 28 and 65
      </message>
    </field-validator>
  </field>
</validators>
```

理想的情况下，上面的 XML 文件会和类文件一起保存在 CLASSPATH 中。让我们有不包含 validate() 方法的 Employee 动作类，如下所示：

```
package com.tutorialspoint.struts2;
import com.opensymphony.xwork2.ActionSupport;
public class Employee extends ActionSupport{
    private String name;
    private int age;
    public String execute()
    {
        return SUCCESS;
    }
    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;
    }
}
```

其余的设置将保持，因为它是前面的例子，现在如果你运行这个应用程序，它将会产生相同的结果，我们在前面的例子获得的。

用 xml 文件来存储配置的优点是允许验证从应用程序代码中分离。你可以让开发人员编写代码和业务分析来创建验证 xml 文件。需要注意的另一件事情是默认验证类型是可用的。默认情况下，有大量使用 Struts 的验证器。常见的验证器包括日期验证器，正则表达式验证器和字符串长度验证器。

本地化

国际化 (i18n) 是规划和实施产品和服务，以便他们可以很容易地适应特定的本地语言和文化的过程，这个过程被称为本地化。国际化过程有时被称为翻译或本地化启用。国际化缩写为 i18n，因为这个单词以 i 开始，以 n 结束，而且第一个 i 和最后的 n 之间有 18 个字符。

Struts2 提供本地化，即国际化 (i18n) 支持，通过使用下面的资源包，拦截器和标签库：

- ? UI 标签
- ? 消息和错误
- ? 动作类内

资源包

Struts 2 使用资源包来给 Web 应用程序的用户提供多种语言和本地化选项。你不必担心用不同的语言编写网页。你所要做的是为每个你想要的语言创建一个资源包。这个资源包将包含你的用户语言的标题，消息和其他文本。资源包是包含使用你的应用程序的默认语言的键/值对的文件。

对资源文件最简单的命名格式是：

```
bundlename_language_country.properties
```

这里，**bundlename** 可以是动作类，接口，超类，模型，包，全局资源属性。接下来的部分语言****_****国家代表国家地区，例如 Spanish (Spain) 地区用 es_ES 代表，English (United States) 地区用 en_US 代表等等。这里你可以跳过可选的国家部分。

当你用它的键引用消息元素时，Struts 框架按照下列顺序进行检索相应的消息：

- ? ActionClass.properties
- ? Interface.properties
- ? SuperClass.properties
- ? model.properties
- ? package.properties
- ? struts.properties
- ? global.properties

为了用多种语言开发你的应用程序，你将不得不保留多个对应于这些语言/地区的属性文件相，并且根据键/值对定义所有的内容。例如，如果你要为美国英语（默认），西班牙语，和法语开发应用程序，你就必须创建三个属性文件。这里，我将只使用 `global.properties` 文件，你可以利用不同的属性文件来隔离不同类型的消息。

？ `global.properties`：默认情况下，英语（美国）将被应用

？ `global_fr.properties`：这将用于法语环境。

？ `global_es.properties`：这将被用于西班牙语语言环境。

访问信息

有几种方法可以访问信息资源，包括 `getText`，文本标签，UI 标签的键属性和国际化标签。让我们来看看他们，如下简单地说：

为了显示 `i18n` 的文本，在属性标签或其他任何标签中调用 `getText`，例如 UI 标签，如下所示：

```
<s:property value="getText('some.key')"/>

```

文本标签检索默认的资源包的信息，即 `struts.properties`

```
<s:text name="some.key"/>

```

`i18n` 标签把任意资源包放进值栈中。`i18n` 标签范围内的其他标签可以显示该资源包的信息：

```
<s:i18n name="some.package.bundle">
  <s:text name="some.key"/>
</s:i18n>

```

大多数 UI 标签的键属性，可以用来检索一个资源包的消息：

```
<s:textfield key="some.key" name="textfieldName"/>

```

本地化例子

让我们把前一章中用多种语言创建的 `index.jsp` 作为目标。相同的文件将被编写，如下所示：

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Employee Form with Multilingual Support</title>

```

```

</head>

<body>
  <h1><s:text name="global.heading"/></h1>

  <s:url id="indexEN" namespace="/" action="locale" >
    <s:param name="request_locale" >en</s:param>
  </s:url>
  <s:url id="indexES" namespace="/" action="locale" >
    <s:param name="request_locale" >es</s:param>
  </s:url>
  <s:url id="indexFR" namespace="/" action="locale" >
    <s:param name="request_locale" >fr</s:param>
  </s:url>

  <s:a href="%{indexEN}" >English</s:a>
  <s:a href="%{indexES}" >Spanish</s:a>
  <s:a href="%{indexFR}" >France</s:a>

  <s:form action="empinfo" method="post" namespace="/">
    <s:textfield name="name" key="global.name" size="20" />
    <s:textfield name="age" key="global.age" size="20" />
    <s:submit name="submit" key="global.submit" />
  </s:form>

</body>
</html>

```

我们将创建 `success.jsp` 文件，假如动作返回 SUCCESS，该文件将被调用。

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Success</title>
</head>
<body>
  <s:property value="getText('global.success')"/>
</body>
</html>

```

这里，我们需要创建下面的两个动作。（a）第一个动作监督区域，显示相同的用不同的语言编写的 `index.jsp` 文件（b）另一个行动是为了监督提交表单本身。动作都将返回 SUCCESS，但根据返回值，我们会采取不同的动作，因为对两个动作来说我们的目的是不同的：

监督区域的动作

```

package com.tutorialspoint.struts2;
import com.opensymphony.xwork2.ActionSupport;
public class Locale extends ActionSupport{
  public String execute()
  {

```

```

    return SUCCESS;
}
}

```

提交表单的动作

```

package com.tutorialspoint.struts2;
import com.opensymphony.xwork2.ActionSupport;
public class Employee extends ActionSupport{
    private String name;
    private int age;
    public String execute()
    {
        return SUCCESS;
    }
    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;
    }
}

```

现在，让我们创建下面三个 `global.properties` 文件，并且把它们放在 CLASSPATH 中：

global.properties

```

global.name = Name
global.age = Age
global.submit = Submit
global.heading = Select Locale
global.success = Successfully authenticated

```

global_fr.properties

```
global.name = Nom d'utilisateur
global.age = l'?ge
global.submit = Soumettre des
global.heading = S é lectionnez Local
global.success = Authentifi é avec succ è s
```

global_es.properties

```
global.name = Nombre de usuario
global.age = Edad
global.submit = Presentar
global.heading = seleccionar la configuracion regional
global.success = Autenticado correctamente
```

我们将创建带有两个动作的 **struts.xml**，如下所示：

```
<?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>
  <constant name="struts.devMode" value="true" />
  <constant name="struts.custom.i18n.resources" value="global" />
  <package name="helloworld" extends="struts-default" namespace="/">
    <action name="empinfo"
      class="com.tutorialspoint.struts2.Employee"
      method="execute">
      <result name="input">/index.jsp</result>
      <result name="success">/success.jsp</result>
    </action>
    <action name="locale"
      class="com.tutorialspoint.struts2.Locale"
      method="execute">
      <result name="success">/index.jsp</result>
    </action>
  </package>
</struts>
```

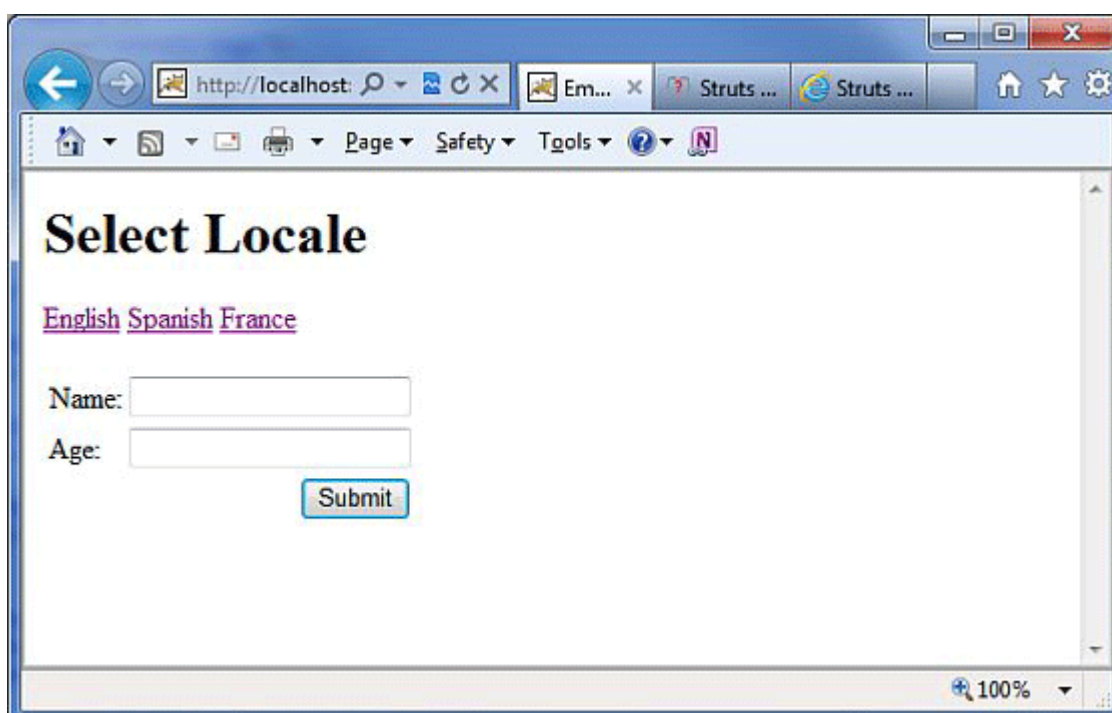
下面是 **web.xml** 文件中的内容：

```

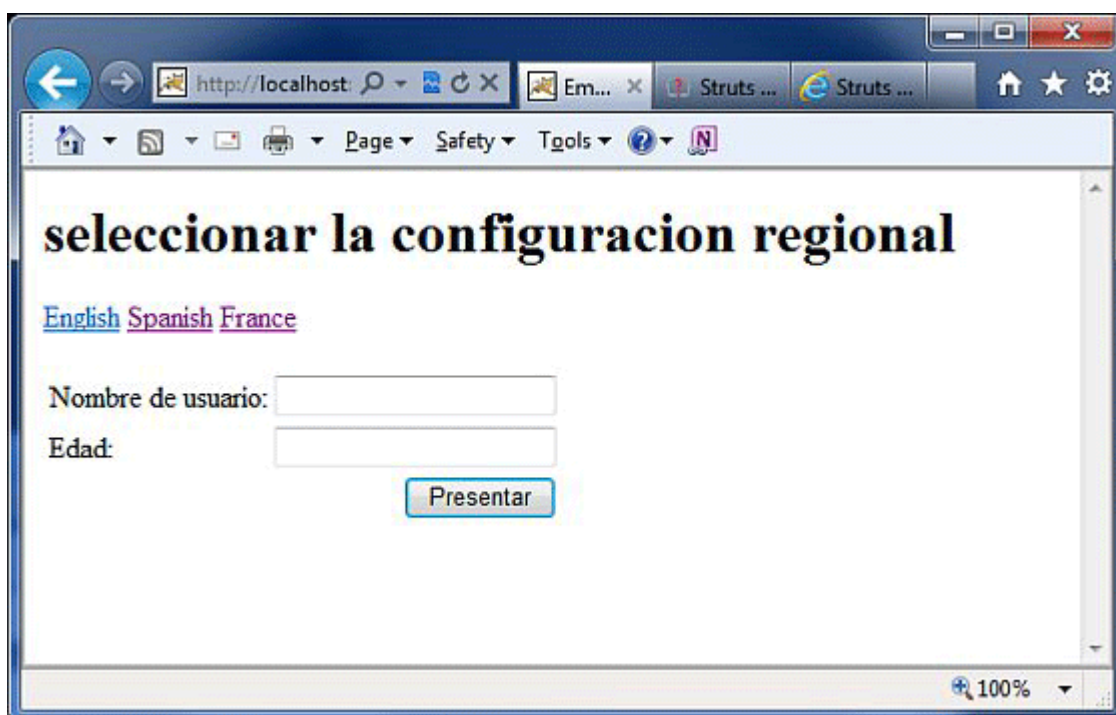
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
  <display-name>Struts 2</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <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>
</web-app>

```

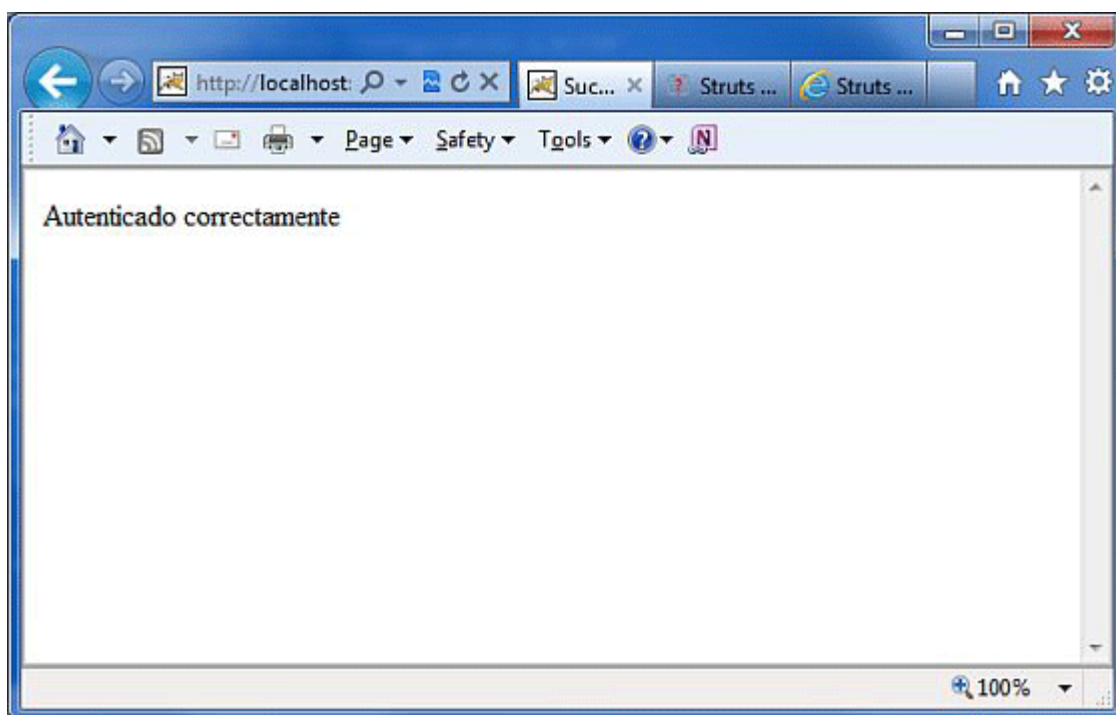
现在，右键点击项目名称，并单击 **Export > WAR File** 创建一个 WAR 文件。然后部署此 WAR 在 Tomcat 的 webapps 目录下。最后，启动 Tomcat 服务器和尝试访问 URL `http://localhost:8080/HelloWorldStruts2/index.jsp`。这会给出以下画面：



现在选择任何一种语言，假如说我们选择西班牙语，它将显示下面的结果：



同样，你可以尝试用法语。最后，当我们使用西班牙语时，让我们尝试单击 Submit 按钮，它会显示下面的画面：



恭喜你，现在你有一个多语言的网页，你可以在全局范围内启动你的网站。

类型转换

所有 HTTP 请求都被视为一个 `String` 的协议。它包括数字，布尔值，整数，日期，小数和其他的一切。根据 HTTP，将所有类型都看成一个字符串。然而，在 Struts 类中，你会有任何数据类型的属性。Struts 是如何自动装配的属性？

Struts 在幕后使用了多种类型转换器用来做繁重的工作。例如，如果在动作类中你有一个整数的属性，那么你不需做任何事情，Struts 就会自动转换请求参数到整数属性。默认情况下，Struts 附带了一些类型转换器。下面列出了它们中的一些，如果你正在使用它们中的一个，那么你就没有什么可担心的：

？ Integer, Float, Double, Decimal

？ Date 和 Datetime

？ Arrays 和 Collections

？ Enumerations

？ Boolean

？ BigDecimal

有时候，当你使用自己的数据类型时，它需要添加自己的转换来使 Struts 知道如何在显示之前转换这些值。考虑下面的 POJO 类 `Environment.java`。

```
package com.tutorialspoint.struts2;

public class Environment {
    private String name;
    public Environment(String name)
    {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

这是一个非常简单的类，它有一个名为 `name` 的属性，所以这个类并没有什么特别之处。让我们创建另一个类，它包含关于系统 – `SystemDetails.java` 的信息。为了这个练习，我有硬编码环境 “Development” 和操

作系统 “Windows XP SP3”。在实际项目中，你会从系统配置中得到这个信息。所以，让我们有下面的动作类：

```
package com.tutorialspoint.struts2;
import com.opensymphony.xwork2.ActionSupport;
public class SystemDetails extends ActionSupport {
    private Environment environment = new Environment("Development");
    private String operatingSystem = "Windows XP SP3";
    public String execute()
    {
        return SUCCESS;
    }
    public Environment getEnvironment() {
        return environment;
    }
    public void setEnvironment(Environment environment) {
        this.environment = environment;
    }
    public String getOperatingSystem() {
        return operatingSystem;
    }
    public void setOperatingSystem(String operatingSystem) {
        this.operatingSystem = operatingSystem;
    }
}
```

接下来让我们创建一个简单的 JSP 文件 **System.jsp** 来显示环境和操作系统信息。

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>System Details</title>
</head>
<body>
    Environment: <s:property value="environment"/><br/>
    Operating System:<s:property value="operatingSystem"/>
</body>
</html>
```

让我们用 **struts.xml** 来同时编写 **system.jsp** 和 **SystemDetails.java** 类。**SystemDetails** 类有一个简单的 **execute()** 方法，它返回字符串 “SUCCESS”。

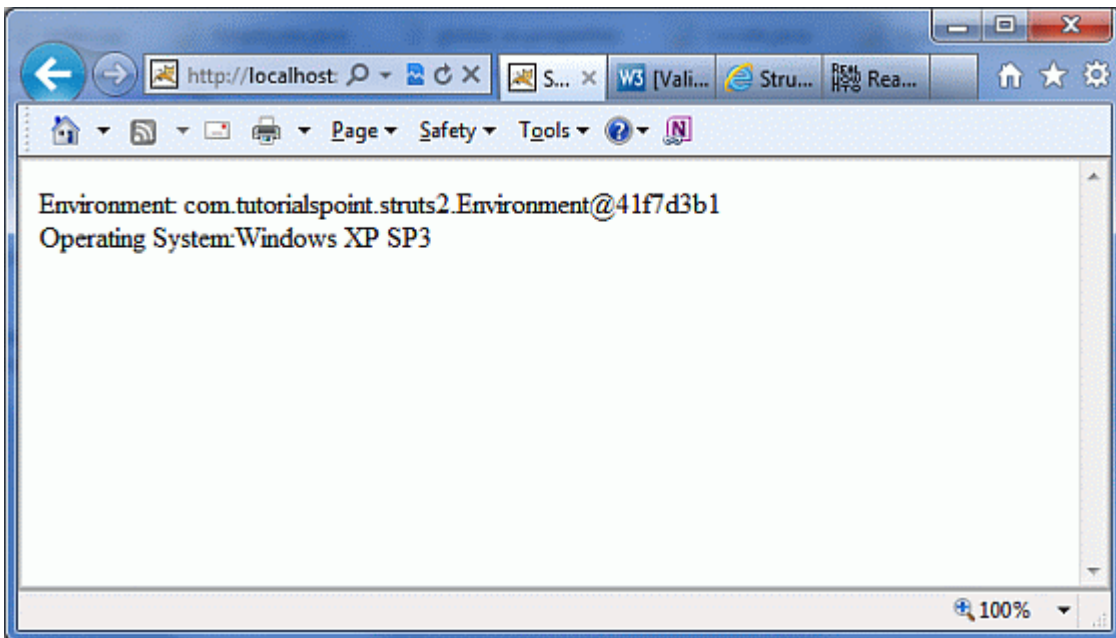
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
```

```

"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name="struts.devMode" value="true" />
  <package name="helloworld" extends="struts-default">
    <action name="system"      class="com.tutorialspoint.struts2.SystemDetails"
      method="execute">
      <result name="success">/System.jsp</result>
    </action>
  </package>
</struts>

```

在项目名称上点击右键，并且单击 **Export > WAR File** 来创建一个 War 文件。然后在 Tomcat 的 webapps 目录下部署这个 WAR。最后，启动 Tomcat 服务器和尝试访问 URL `http://localhost:8080/HelloWorldStruts2/system.action`。将会给出下面的画面：



上面的输出有什么错？Struts 知道如何显示和转换字符串 “Windows XP SP3” 和其他内置数据类型，但是它不知道如何处理环境类型的属性。所以，它仅仅被称为类中的 `toString()` 方法。为了解决这个问题，现在，让我们为环境类创建并注册一个简单的 `TypeConverter`。创建一个类，名为 `EnvironmentConverter.java`，内容如下所示。

```

package com.tutorialspoint.struts2;
import java.util.Map;
import org.apache.struts2.util.StrutsTypeConverter;

public class EnvironmentConverter extends StrutsTypeConverter {
  @Override
  public Object convertFromString(Map context, String[] values,
    Class clazz) {
    Environment env = new Environment(values[0]);

```

```

    return env;
}
@Override
public String convertToString(Map context, Object value) {
    Environment env = (Environment) value;
    return env == null ? null : env.getName();
}
}

```

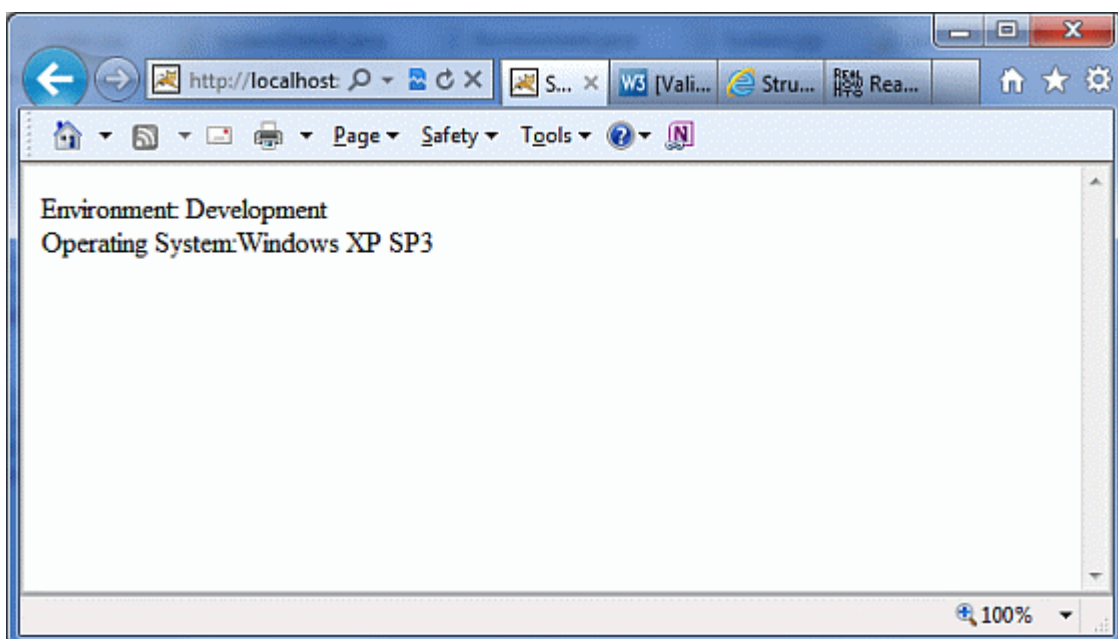
EnvironmentConverter 扩展了 StrutsTypeConverter 类，告诉 Struts 如何通过重写两个方法 convertFrom String() 和 convertToString() 把环境转换为一个字符串，反之亦然。现在，让我们在应用程序中使用这个转换器之前注册它。有两种注册转换器的方法。如果转换器将只用于一个特定的动作中，那么你将需要创建一个属性文件，它需要被命名为 [action-class]-conversion.properties，所以，在我们的例子中，我们创建一个带有下面的注册入口的文件，名为 SystemDetails-conversion.properties：

```
environment=com.tutorialspoint.struts2.EnvironmentConverter
```

在上面的例子中，“environment”是 SystemDetails.java 类中一个属性的名字，我们告诉 Struts 为了来回转换这个属性要使用 EnvironmentConverter。然而，我们不会这样做，相反，我们会在全局范围内注册这个转换器，以便它可以在整个应用程序中使用。为了做到这个，在 WEB-INF/classes 文件夹下创建一个带有下面句子的属性文件，名为 xwork-conversion.properties：

```
com.tutorialspoint.struts2.Environment = \ com.tutorialspoint.struts2.EnvironmentConverter
```

它简单地在全局范围内注册转换器，为此每次 Struts 遇到一个环境类型的对象，就可以自动做转换。现在，如果你重新编译并重新运行该程序，你将会得到更好的输出，如下所示：



显然，现在的结果更好，这意味着我们的 Struts 转换器工作正常。这就是你可以按你的需求创建多个转换器，并且注册它们后使用。

主题/模板

在开始本章教程之前，让我们看几个 <http://struts.apache.org> 给出的定义：

项	描述
标签	从JSP，FreeMarker 或 Velocity 内部执行的代码小片段
模板	一小段代码，通常在 FreeMarker 中编写，并且通过某种标签能够被呈现出来（HTML 标签）
主题	打包到一起的模板的集合，用来提供自定义功能的。

我还建议浏览 [Struts 2 本地化](#) 章节，因为我们会再次采用相同的例子来完成我们的实践。

当你在你的 web 页面上使用一个 Struts 2 标签时，如 `<s:submit...>`，`<s:textfield...>` 等，Struts 2 框架会生成 HTML 代码和预配置的样式及布局。Struts 2 有三种内置的主题：

主题	描述
simple theme	一个最小的主题，不带有 "bells and whistles"。例如，文本框标签呈现不带有标记，验证，错误报告，或其他任何格式化或功能的 HTML <code><input/></code> 标签。
xhtml theme	这是 Struts 2 使用的默认的主题，提供了所有简单主题提供的基础，并添加了几个功能，如为 HTML 添加的标准的两列表布局，为每个 HTML 添加的标记，验证和错误报告等等。
css_xhtml theme	这个主题提供了所有简单主题提供的基础，并添加了几个功能，如标准的两列基于 CSS 的布局，为 HTML Struts 标签使用 <code><div></code> ，为每个 HTML Struts 标签添加布局，根据 CSS 样式表替换。

正如上面提到的，如果你没有指定一个主题，那么 Struts 2 会使用默认的 xhtml 主题。例如这个 Struts 2 选择标签：

```
<s:textfield name="name" label="Name" />
```

产生如下所示的 HTML 标记：

```
<tr>
<td class="tdLabel">
  <label for="empinfo_name" class="label">Name:</label>
</td><td>
  <input type="text" name="name" value="" id="empinfo_name"/>
</td>
</tr>
```

这里 `empinfo` 是在 `struts.xml` 文件中定义的操作名。

选择主题

你可以在每个 Struts 2 标签基础上选择主题或者你可以使用下述方法中的一个来指定 Struts 2 要用的主题：

- ？ 特定标签上的主题属性
- ？ 在标签的环绕表单标签上的主题属性
- ？ 命名为“主题”的限定页面范围的属性
- ？ 命名为“主题”的限定请求范围的属性
- ？ 命名为“主题”的限定会话范围的属性
- ？ 命名为“主题”的限定应用范围的属性
- ？ 在 `struts.properties` 中的 `struts.ui.theme` 属性 (xhtml 的默认值)

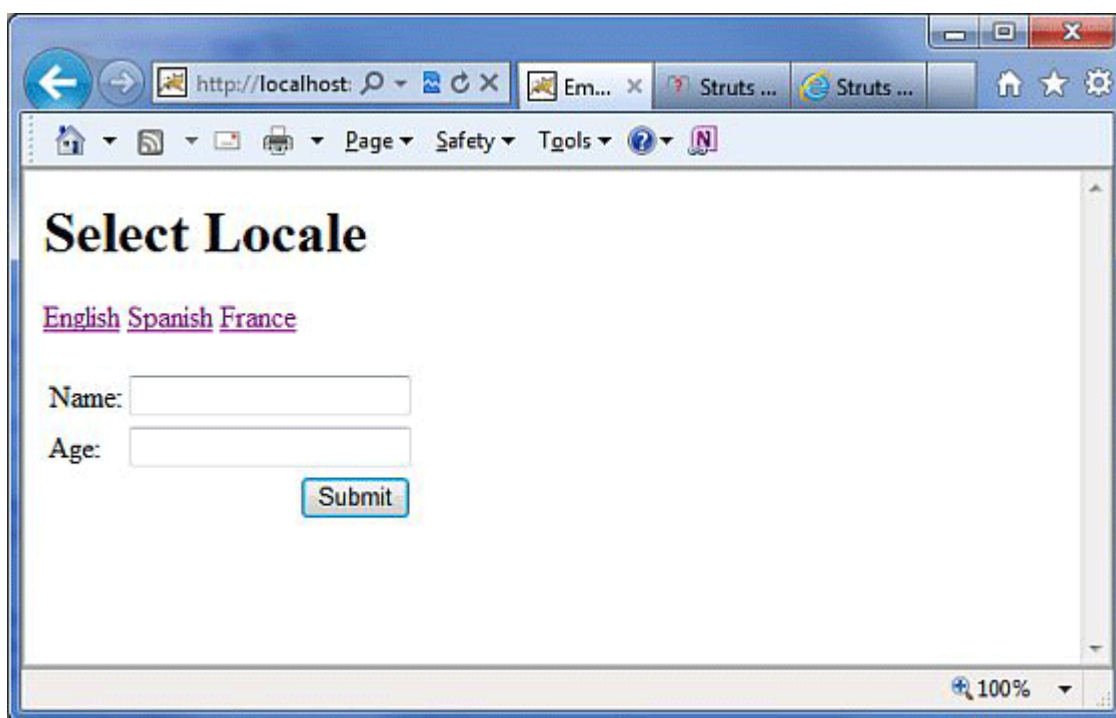
如果你不想为不同的标签使用不同的主题，以下是在标签级别指定一个主题的语法：

```
<s:textfield name="name" label="Name" theme="xhtml"/>
```

由于在每个标签上都使用主题是不太实际的，所以简单来说，我们可以使用下列标签在 `struts.properties` 文件中指定规则：

```
## Standard UI theme  
  
struts.ui.theme=xhtml  
## Directory where theme template resides  
  
struts.ui.templateDir=template  
## Sets the default template type. Either ftl, vm, or jsp  
  
struts.ui.templateSuffix=ftl
```

以下是我们从本地化章节中选取的结果，我们使用了默认的主题并设置了 `struts-default.properties` 文件中的 `struts.ui.theme=xhtml`，该文件是 `struts2-core.x.y.z.jar` 文件中默认的。



主题怎样工作？

对于一个给定的主题，每一个 struts 标签都有一个相关的模板，如 `s:textfield` -> `text.ftl` 和 `s:password` -> `password.ftl` 等。这些模板文件压缩到 `struts2-core.x.y.z.jar` 文件中。这些模板文件为每个标签保存一个预定义的 HTML 布局。所以 Struts 2 框架用 Struts 标签和相关的模板生成最终的 HTML 标记代码。

Struts 2 tags + Associated template file = Final HTML markup code.

默认模板已经被写入 FreeMarker 中并且它们有扩展版本 `.ftl`。你可以使用 velocity 或 JSP 来设计自己的模板，因此可以使用 `struts.ui.templateSuffix` 和 `struts.ui.templateDir` 在 `struts.properties` 中设置配置。

创建新的主题

创建新主题的最简单的方式是复制任何现存的主题/模板文件并做一些必需的修改。所以让我们以在 `WebContent/WEB-INF/classes` 中创建一个命名为 `template` 的文件夹开始，并创建一个我们的新的主题命名的子文件夹，如 `WebContent/WEB-INF/classes/template/mytheme`。从这里开始，你可以构建模板，或者你可以从 Struts2 发行版中复制模板并根据需要修改它们。

处于学习的目的，我们要修改一些现存的默认模板 `xhtml`。所以现在让我们复制 `struts2-core-x.y.z.jar/template/xhtml` 的内容到我们的主题目录中并只修改 `WebContent/WEB-INF/classes/template/mytheme/control.ftl` 文件。当我们打开 `control.ftl` 时，它会有如下所示的行：

```
<table class="{parameters.cssClass?default('wwFormTable')}?html}"<#rt/>
<#if parameters.cssStyle??> style="{parameters.cssStyle?html}"<#rt/>
</#if>
>
```

让我们修改上述文件 `control.ftl`，使得它包含以下内容：

```
<table style="border:1px solid black;">
```

如果你检查 `form.ftl`，你会发现 `control.ftl` 正在该文件中使用，而 `form.ftl` 文件正在从 `xhtml` 主题引用该文件。所以让我们做出如下所示的修改：

```
<#include "/${parameters.templateDir}/xhtml/form-validate.ftl" />
<#include "/${parameters.templateDir}/simple/form-common.ftl" />
<#if (parameters.validate?default(false))>
  onreset="{parameters.onreset?default('clearErrorMessages(this);\
  clearErrorLabels(this);')}"
<#else>
  <#if parameters.onreset??>
    onreset="{parameters.onreset?html}"
  </#if>
</#if>
>
<#include "/${parameters.templateDir}/mytheme/control.ftl" />
```

我假设你不太理解 FreeMarker 模板语言，但是你仍然能知道通过浏览 `.ftl` 文件需要做什么。但是，让我们保存上述改变，然后回到我们的本地化实例中，并创建 `WebContent/WEB-INF/classes/struts.properties` 文件，其内容如下所示：

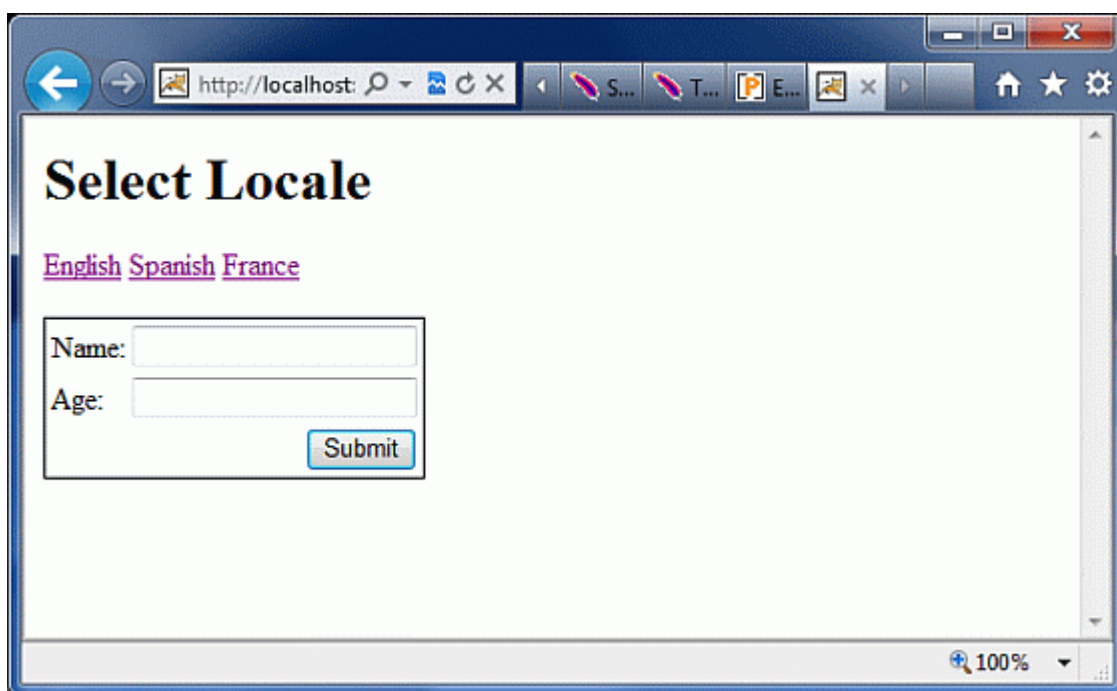
```
## Customized them

struts.ui.theme=mytheme
## Directory where theme template resides

struts.ui.templateDir=template
## Sets the template type to ftl.

struts.ui.templateSuffix=ftl
```

现在，做出改变之后，鼠标右键单击项目名，点击 `Export > WAR File` 来创建一个 War 文件。然后将这个 WAR 文件部署到 Tomcat 的 web 应用程序目录中。最后，启动 Tomcat 服务器并尝试访问 URL `http://localhost:8080/HelloWorldStruts2`。这将显示如下所示的画面：



你可以在表单组件的周围看到一个边界，这是从 xhtml 主题中复制主题并做出改变的结果。如果你没有学习 FreeMarker，那么你可以很容易的创建或修改你的主题。至少现在你对 Struts 2 主题和模板有一个基本的了解，不是吗？

异常处理

Struts 提供了一种简单的方式来处理未捕获的异常，并将用户重定向到一个专门的错误页面。你可以简单的配置 Struts，使得不同的异常有不同的错误页面。

Struts 通过使用“异常”拦截器来使异常处理变得简单。“异常”拦截器是默认的栈的一部分，所以配置它你不需要做额外的工作。它为你的使用已经做好了准备，开箱即用。让我们看一个简单的 Hello World 实例，在 `HelloWorldAction.java` 文件中带有一些修改。在这里我们在 `HelloWorldAction` 操作代码中有意的引入一个 `NullPointerException` 异常。

```
package com.tutorialspoint.struts2;
import com.opensymphony.xwork2.ActionSupport;
public class HelloWorldAction extends ActionSupport{
    private String name;
    public String execute(){
        String x = null;
        x = x.substring(0);
        return SUCCESS;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

让我们将下入内容保存到 `HelloWorld.jsp` 中：

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Hello World</title>
</head>
<body>
    Hello World, <s:property value="name"/>
</body>
</html>
```

以下是 `index.jsp` 中的内容：

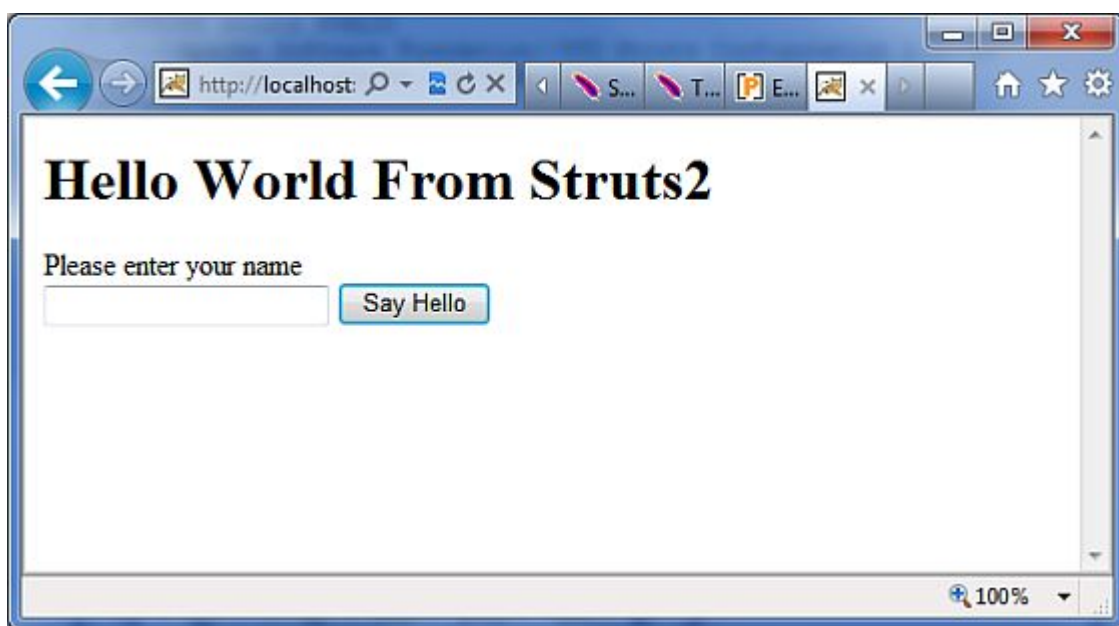
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Hello World</title>
</head>
<body>
<h1>Hello World From Struts2</h1>
<form action="hello">
  <label for="name">Please enter your name</label><br/>
  <input type="text" name="name"/>
  <input type="submit" value="Say Hello"/>
</form>
</body>
</html>
```

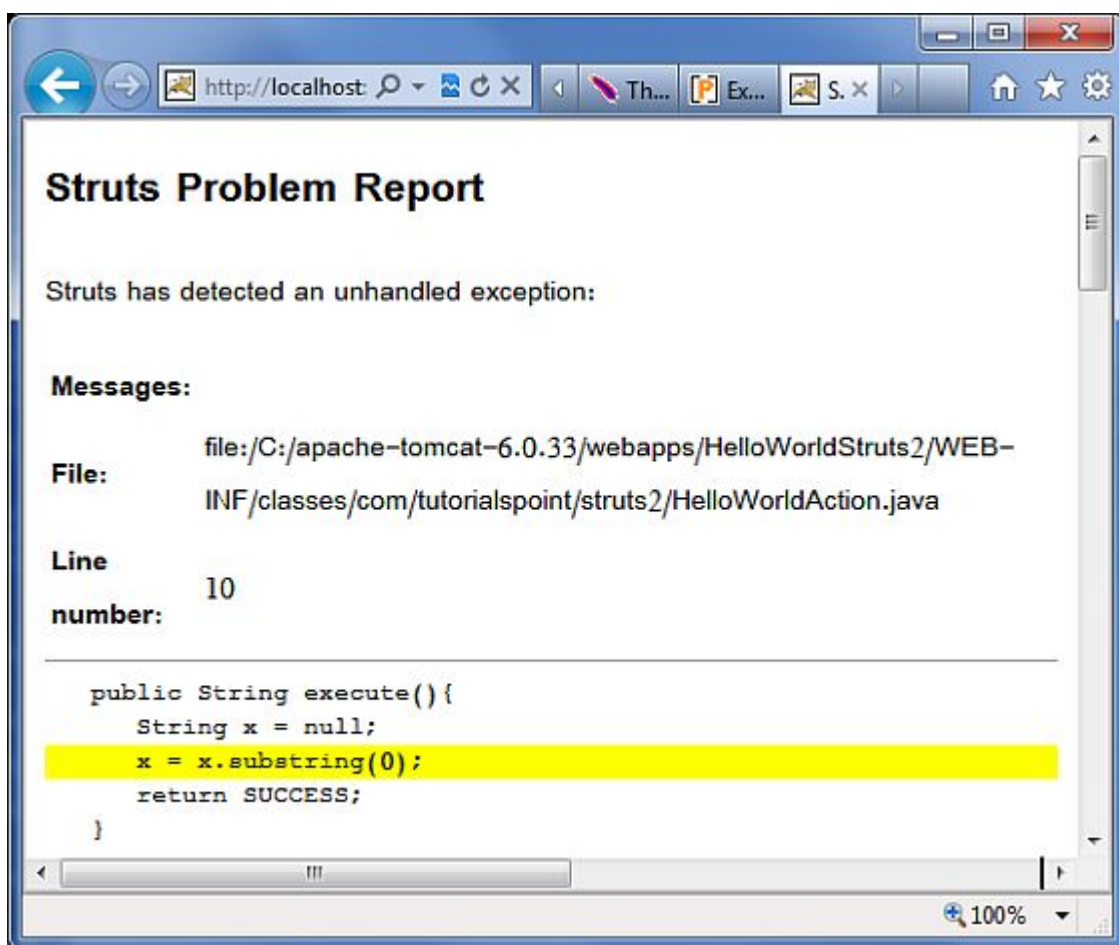
你的 **struts.xml** 文件看起来应该如下所示：

```
<?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>
<constant name="struts.devMode" value="true" />
  <package name="helloworld" extends="struts-default">
    <action name="hello"
      class="com.tutorialspoint.struts2.HelloWorldAction"
      method="execute">
      <result name="success">/HelloWorld.jsp</result>
    </action>
  </package>
</struts>
```

现在鼠标右键单击项目名并点击 **Export > WAR File** 来创建一个 War 文件。然后把这个 WAR 部署到 Tomcat 的 web 应用程序目录中。最后，启动 Tomcat 服务器并尝试访问 URL `http://localhost:8080/HelloWorldStruts2/index.jsp`。这将呈现如下所示的画面：



输入值 "Struts2" 并提交页面。你应该能看到如下所示的页面：



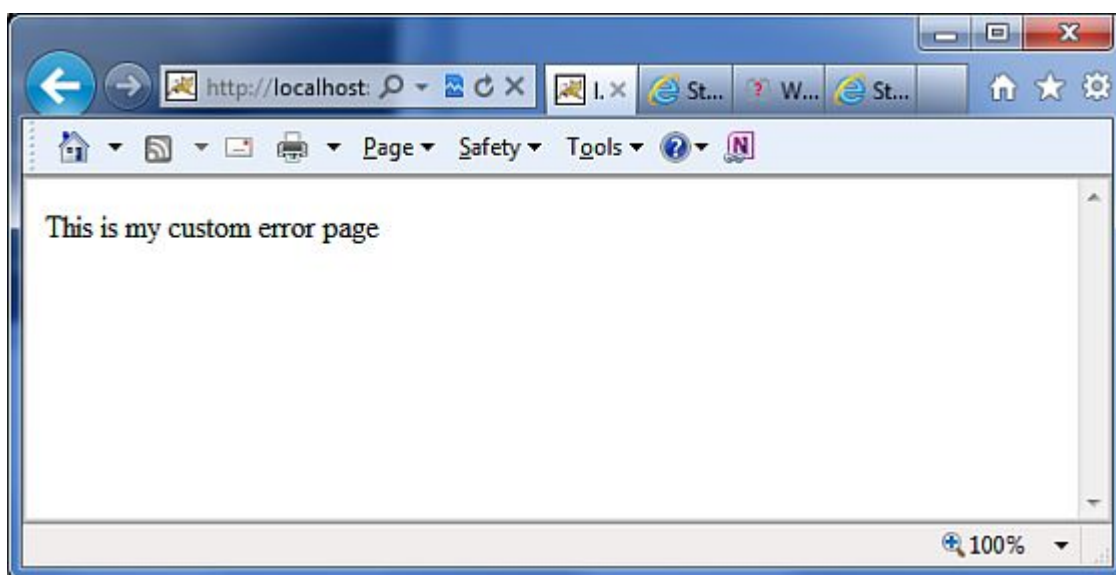
正如上述例子显示的一样，默认异常拦截器做了许多处理异常的工作。现在让我们为我们的异常创建一个专门的错误页面。创建一个名为 `Error.jsp` 的文件，其内容如下所示：

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title></title>
</head>
<body>
    This is my custom error page
</body>
</html>
```

现在让我们配置 Struts 来在异常的情况下使用这个错误页面。让我们修改 `struts.xml` 文件，如下所示：

```
<?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>
<constant name="struts.devMode" value="true" />
    <package name="helloworld" extends="struts-default">
        <action name="hello"
            class="com.tutorialspoint.struts2.HelloWorldAction"
            method="execute">
            <exception-mapping exception="java.lang.NullPointerException"
                result="error" />
            <result name="success">/HelloWorld.jsp</result>
            <result name="error">/Error.jsp</result>
        </action>
    </package>
</struts>
```

正如上述例子显示的一样，现在我们已经配置了 Struts 来为 `NullPointerException` 使用专门的 `Error.jsp`。如果你现在重新运行这个程序，你应该能看到如下所示的输出：



此外，Struts2 框架会生成一个 "logging" 拦截器来记录异常。通过使记录器记录未捕获的异常，我们可以很容易的查看栈跟踪并找到哪里出现了错误。

全局异常映射

我们已经了解了如何处理操作指定的异常。我们可以设置一个全局异常，能够应用带所有的操作中。例如，为了捕获相同的 `NullPointerException` 异常，我们可以在 `<package...>` 标签内部添加 `<global-exception-mappings...>` 标签且它的 `<result...>` 标签应该被添加到 `struts.xml` 文件的 `<action...>` 标签内，如下所示：

```
<?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>
<constant name="struts.devMode" value="true" />
<package name="helloworld" extends="struts-default">
    <global-exception-mappings>
        <exception-mapping exception="java.lang.NullPointerException"
            result="error" />
    </global-exception-mappings>
    <action name="hello"
        class="com.tutorialspoint.struts2.HelloWorldAction"
        method="execute">
        <result name="success">/HelloWorld.jsp</result>
        <result name="error">/Error.jsp</result>
    </action>
</package>
</struts>
```


注释

正如前面所提到的，Struts 提供了两种形式的配置。传统的方式是为所有的配置使用 `struts.xml` 文件。目前为止，在本教程中我们已经见过了太多这样的例子。另一种配置 Struts 的方式是使用 Java 5 注释功能。使用 Struts 注释，我们可以实现 **零配置**。

在你的项目中开始使用注释之前，请确保在你的 `WebContent/WEB-INF/lib` 文件夹中包含下述 jar 文件：

- ? `struts2-convention-plugin-x.y.z.jar`
- ? `asm-x.y.jar`
- ? `antlr-x.y.z.jar`
- ? `commons-fileupload-x.y.z.jar`
- ? `commons-io-x.y.z.jar`
- ? `commons-lang-x.y.jar`
- ? `commons-logging-x.y.z.jar`
- ? `commons-logging-api-x.y.jar`
- ? `freemarker-x.y.z.jar`
- ? `javassist-.xy.z.GA`
- ? `ognl-x.y.z.jar`
- ? `struts2-core-x.y.z.jar`
- ? `xwork-core.x.y.z.jar`

现在让我们看看不用 `struts.xml` 文件中的可用的配置而是用注释代替，应该怎么做。

为了解释 Struts2 中只是的概念，我们应该重新考虑在 [Struts 2 – 验证](#) 章节中解释的验证示例。

这里我们采用 `Employee` 的例子，它使用一个简单的页面来获取姓名和年龄，并且我们会设置两个验证来确保用户会输入姓名以及年龄在 28-65 之间。所以让我们从例子中的主 JSP 页面开始。

创建主页面

让我们编写主页面 JSP 文件 `index.jsp`，用于与上述信息相关的 `Employee`。

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Employee Form</title>
</head>

<body>

    <s:form action="empinfo" method="post">
        <s:textfield name="name" label="Name" size="20" />
        <s:textfield name="age" label="Age" size="20" />
        <s:submit name="submit" label="Submit" align="center" />
    </s:form>

</body>
</html>
```

`index.jsp` 使用 Struts 标签，Struts 标签我们还没有涉及到，但是在标签相关的章节中我们会学习到。但是现在，只能假设 `s:textfield` 标签产生一个输入字段，`s:submit` 标签产生一个提交按钮。我们已经为每个标签使用了标记属性，会为每个标签创建标记。

创建视图

我们使用 JSP 文件 `success.jsp`，当定义的操作返回 `SUCCESS` 时，该文件会被调用。

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Success</title>
</head>
<body>
    Employee Information is captured successfully.
</body>
</html>
```

创建操作

这是注释会被用到的地方。让我们用注释重新定义操作类 `Employee`，然后将如下所示的 `validate()` 方法添加到 `Employee.java` 文件中。确保你的操作类扩展了 `ActionSupport` 类，否则你的验证方法不会被执行。

```
package com.tutorialspoint.struts2;
import com.opensymphony.xwork2.ActionSupport;
import org.apache.struts2.convention.annotation.Action;
import org.apache.struts2.convention.annotation.Result;
import org.apache.struts2.convention.annotation.Results;
import com.opensymphony.xwork2.validator.annotations.*;

@Results({
    @Result(name="success", location="/success.jsp"),
    @Result(name="input", location="/index.jsp")
})
public class Employee extends ActionSupport{
    private String name;
    private int age;
    @Action(value="/empinfo")
    public String execute()
    {
        return SUCCESS;
    }
    @RequiredFieldValidator( message = "The name is required" )
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    @IntRangeFieldValidator(message = "Age must be in between 28 and 65",
                           min = "29", max = "65")
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

在这个例子中我们已经使用了几个注释。让我们一个一个的仔细浏览一下：

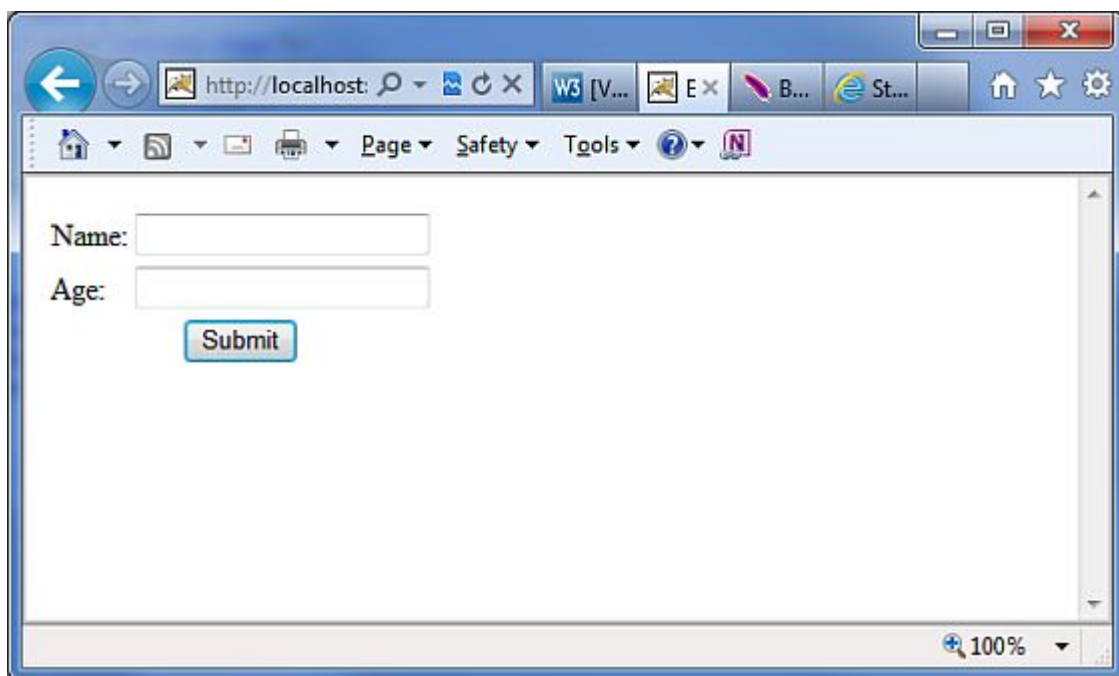
- ？ 首先，我们包含了 **Results** 注释。Results 注释时结果的集合。在结果注释下，我们有两个结果注释。结果注释有名字，对应于该执行方法的输出。它们还包括视图被服务的位置，对应于从 `execute()` 返回的值。
- ？ 下一个注释是 **Action** 注释。这是用于装饰 `execute()` 方法的。操作方法带有一个值，该值是这个操作被调用的 URL。
- ？ 最后，我使用了两个 **validation** 注释。我已经在 **name** 字段配置了必需的字段验证器，并且在 **age** 字段配置了整数范围验证器。我还为验证指定了一个自定义的消息。

配置字段

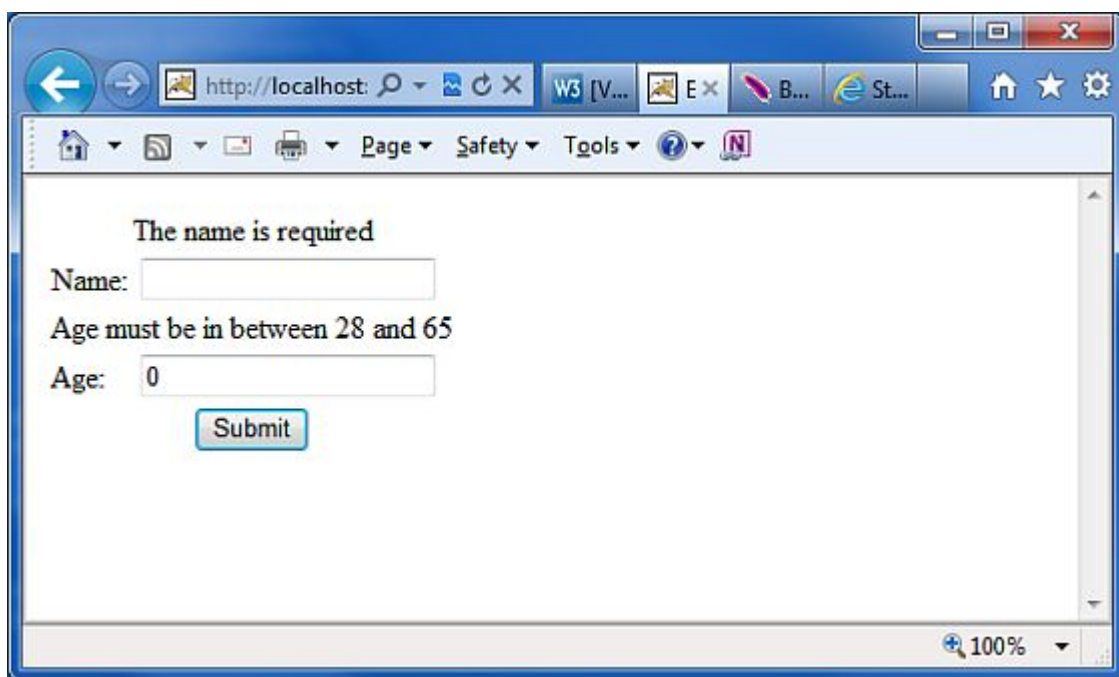
我们实在不需要 `struts.xml` 配置文件，所以让我们删除这个文件并查看 `web.xml` 文件中的内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
  <display-name>Struts 2</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>
      org.apache.struts2.dispatcher.FilterDispatcher
    </filter-class>
    <init-param>
      <param-name>struts.devMode</param-name>
      <param-value>true</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

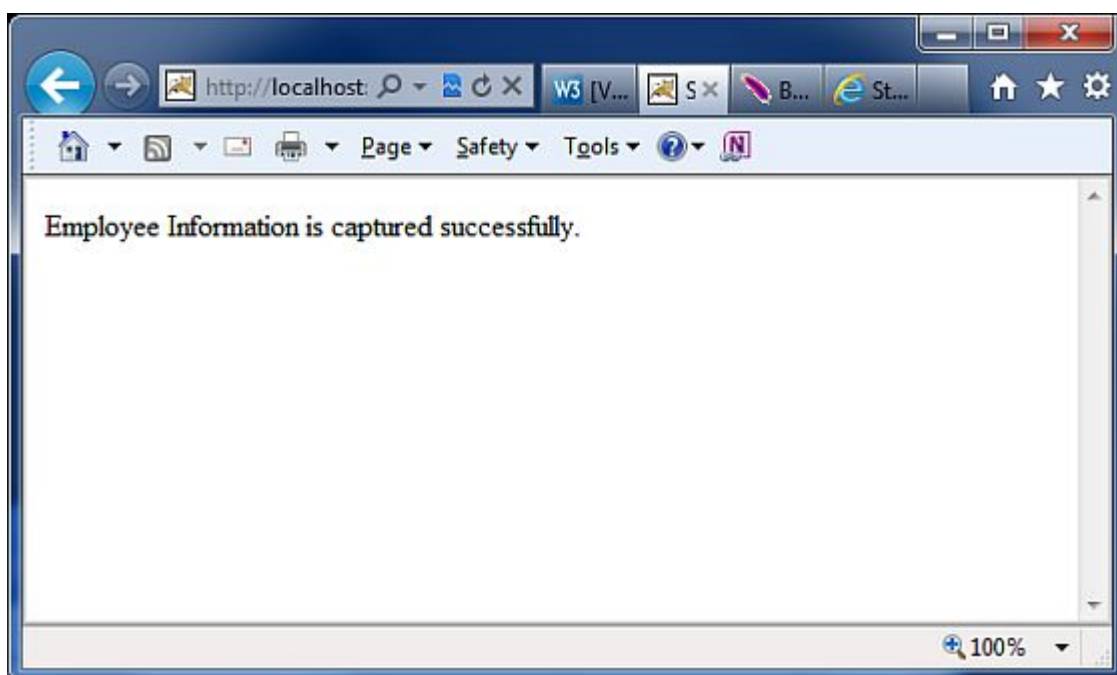
现在，鼠标右键单击项目名，点击 **Export > WAR File** 来创建一个 War 文件。然后将这个 WAR 文件部署到 Tomcat 的 web 应用程序的目录中。最后，启动 Tomcat 服务器并尝试访问 URL `http://localhost:8080/HelloWorldStruts2/index.jsp`。这将为你呈现如下所示的画面：



现在不要键入任何必需的信息，只是点击提交按钮。你会看到如下所示的结果：



键入必需的信息，但是输入一个错误的表单字段，如姓名输入为 "test"，年龄输入为 30，然后点击 提交按钮。你会看到如下所示的结果：



Struts 2 注释类型

Struts 2 应用程序可以使用 Java 5 注释作为 XML 和 Java 属性配置的可选项。你可以查看与不同类别相关的最重要的注释列表：

[Struts 2 注释类型](#)

4j: struts_annotations_types 这页要翻译，直接加在下面，不用单独一页。

Struts 2 注解类型

Struts 2 应用程序可以使用 Java 5 注解作为 XML 和 Java 属性配置的一个替代。这里是和不同类别相关的最重要的注解列表：

Namespace 注解（动作注解）

@Namespace 注解允许在 Action 类中允许定义一个动作的 namespace，而不是基于 Zero 配置的约定。

```
@Namespace("/content")
public class Employee extends ActionSupport{
    ...
}
```

Result 注解 – (动作注解)

@Result 注解允许在 Action 类中定义动作结果，而不是在一个 XML 文件中。

```
@Result(name="success", value="/success.jsp")
public class Employee extends ActionSupport{
    ...
}
```

Results 注解 – (动作注解)

@Results 注解为一个 Action 定义了一组结果。

```
@Results({
    @Result(name="success", value="/success.jsp"),
    @Result(name="error", value="/error.jsp")
})
public class Employee extends ActionSupport{
    ...
}
```

After 注解 – (拦截注解)

@After 注解标记一个在主动作方法之后需要调用的动作方法，并且结果被执行。返回值将被忽略。

```
public class Employee extends ActionSupport{
    @After
    public void isValid() throws ValidationException {
        // validate model object, throw exception if failed
    }
    public String execute() {
        // perform secure action
        return SUCCESS;
    }
}
```

Before 注解 – (拦截注解)

@Before 注解标记一个在主动作方法之前需要调用的动作方法，并且结果被执行。返回值将被忽略。

```
public class Employee extends ActionSupport{
    @Before
    public void isAuthorized() throws AuthenticationException {
        // authorize request, throw exception if failed
    }
    public String execute() {
        // perform secure action
        return SUCCESS;
    }
}
```

BeforeResult 注解 –（拦截注解）

@BeforeResult 注解标记一个在结果之前需要执行的动作方法。返回值将被忽略。

```
public class Employee extends ActionSupport{
    @BeforeResult
    public void isValid() throws ValidationException {
        // validate model object, throw exception if failed
    }
    public String execute() {
        // perform action
        return SUCCESS;
    }
}
```

ConversionErrorFieldValidator 注解 –（验证注解）

这个验证注解检查一个字段是否有任何转换错误，如果转换错误存在，则应用它们。

```
public class Employee extends ActionSupport{
    @ConversionErrorFieldValidator(message = "Default message",
        key = "i18n.key", shortCircuit = true)
    public String getName() {
        return name;
    }
}
```

DateRangeFieldValidator 注解 –（验证注解）

这个验证注解检查一个日期字段有一个值在指定的范围内。


```
public class Employee extends ActionSupport{
    @DateRangeFieldValidator(message = "Default message",
        key = "i18n.key", shortCircuit = true,
        min = "2005/01/01", max = "2005/12/31")
    public String getDOB() {
        return dob;
    }
}
```

DoubleRangeFieldValidator 注解 – (验证注解)

这个验证注解检查一个 double 字段有一个值在指定的范围内。如果设置的既不是最小的也不是最大的，那么什么都不要做。

```
public class Employee extends ActionSupport{
    @DoubleRangeFieldValidator(message = "Default message",
        key = "i18n.key", shortCircuit = true,
        minInclusive = "0.123", maxInclusive = "99.987")
    public String getIncome() {
        return income;
    }
}
```

EmailValidator 注解 – (验证注解)

这个验证注解检查一个字段是一个有效的 e-mail 地址，如果它包含一个非空的字符串。

```
public class Employee extends ActionSupport{
    @EmailValidator(message = "Default message",
        key = "i18n.key", shortCircuit = true)
    public String getEmail() {
        return email;
    }
}
```

ExpressionValidator 注解 – (验证注解)

这个非字段级验证器验证一个提供的正则表达式。

```
@ExpressionValidator(message = "Default message", key = "i18n.key",
    shortCircuit = true, expression = "an OGNL expression" )
```

IntRangeFieldValidator 注解 – (验证注解)

这个验证注解检查一个数字字段有一个值在指定的范围内。如果设置的既不是最小的也不是最大的，那么什么都不要做。

```

public class Employee extends ActionSupport{
    @IntRangeFieldValidator(message = "Default message",
        key = "i18n.key", shortCircuit = true,
        min = "0", max = "42")
    public String getAge() {
        return age;
    }
}

```

RegexFieldValidator 注解 – (验证注解)

这个注解使用一个正则表达式来验证一个字符串字段。

```

@RegexFieldValidator( key = "regex.field", expression = "yourregexp")

```

RequiredFieldValidator 注解 – 验证注解)

这个验证注解检查一个字段是非空的。这个注解必须应用在方法层。

```

public class Employee extends ActionSupport{
    @RequiredFieldValidator(message = "Default message",
        key = "i18n.key", shortCircuit = true)
    public String getAge() {
        return age;
    }
}

```

RequiredStringValidator 注解 – (验证注解)

这个验证注解检查一个字符串字段不是空的（即非空，长度 > 0）。

```

public class Employee extends ActionSupport{
    @RequiredStringValidator(message = "Default message",
        key = "i18n.key", shortCircuit = true, trim = true)
    public String getName() {
        return name;
    }
}

```

```
}
}
```

StringLengthFieldValidator 注解 – (验证注解)

这个验证器检查一个字符串字段是正确的长度。假定该字段是一个字符串。如果设置的既不是最小长度也不是最大长度，那么什么都不要做。

```
public class Employee extends ActionSupport{
    @StringLengthValidator(message = "Default message",
        key = "i18n.key", shortCircuit = true,
        trim = true, minLength = "5", maxLength = "12")
    public String getName() {
        return name;
    }
}
```

UrlValidator 注解 – (验证注解)

这个验证器检查一个字段是一个有效的 URL。

```
public class Employee extends ActionSupport{
    @UrlValidator(message = "Default message",
        key = "i18n.key", shortCircuit = true)
    public String getURL() {
        return url;
    }
}
```

Validations 注解 – (验证注解)

如果你想使用多个相同类型的注解，这些注解必须在 @Validations() 注解中嵌套。

```
public class Employee extends ActionSupport{
    @Validations(
        requiredFields =
            {@RequiredFieldValidator(type = ValidatorType.SIMPLE,
                fieldName = "customfield",
                message = "You must enter a value for field."}},
        requiredStrings =
            {@RequiredStringValidator(type = ValidatorType.SIMPLE,
                fieldName = "stringisrequired",
```

```

        message = "You must enter a value for string.")]
    )
    public String getName() {
        return name;
    }
}

```

CustomValidator 注解 – (验证注解)

这个注解可以用于自定义验证器。使用 ValidationParameter 注解来提供额外的参数。

```
@CustomValidator(type ="customValidatorName", fieldName = "myField")
```

Conversion 注解 – (类型转换注解)

在类型层中，这是类型转换的一个标记注解。Conversion 注解必须应用在类型层。

```

@Conversion()
public class ConversionAction implements Action {
}

```

CreateIfNull 注解 – 类型转换注解)

这个注解为类型转换设置为 CreateIfNull。CreateIfNull 注解必须应用在字段或方法层。

```

@CreateIfNull( value = true )
private List<User> users;

```

Element 注解 – (类型转换注解)

这个注解为类型转换设置为 Element。Element 注解必须应用在字段或方法层。

```

@Element( value = com.acme.User )
private List<User> userList;

```

Key 注解 – (类型转换注解)

这个注解为类型转换设置为 Key。Key 注解必须应用在字段或方法层。

```

@Key( value = java.lang.Long.class )
private Map<Long, User> userMap;

```

KeyProperty 注解 – (类型转换注解)

这个注解为类型转换设置为 KeyProperty。KeyProperty 注解必须应用在字段或方法层。

```
@KeyProperty( value = "userName" )  
protected List<User> users = null;
```

TypeConversion 注解 – (类型转换注解)

这个注解的注解用于类和应用程序范围的转换规则。TypeConversion 注解必须应用在属性或方法层。

```
@TypeConversion(rule = ConversionRule.COLLECTION,  
converter = "java.util.String")  
public void setUsers( List users ) {  
    this.users = users;  
}
```



Struts 2 标签



控制标签

Struts 2 有一组标签使得控制页面执行流非常容易。以下是重要的 Struts 2 控制标签的列表：

if 和 else 标签

该标签实现基本的能够在每种语言中找到的条件流。'If' 标签能够单独使用或与 'Else If' 标签一起使用，单个/多个 'Else' 标签如下所示：

```
<s:if test="%{false}">
  <div>Will Not Be Executed</div>
</s:if>
<s:elseif test="%{true}">
  <div>Will Be Executed</div>
</s:elseif>
<s:else>
  <div>Will Not Be Executed</div>
</s:else>
```

迭代标签

迭代标签会迭代一个值。一个迭代的值可以是任何 `java.util.Collection` 或 `java.util.Iterator`。当迭代一个值时，你可以使用 `Sort` 标签来把结果分类或者使用 `SubSet` 标签得到列表或数组的子集。

以下例子检索了值栈中当前对象的 `getDays()` 方法的值，并用它迭代。标签输出了当前迭代的值。

```
<s:iterator value="days">
  <p>day is: <s:property/></p>
</s:iterator>
```

融合标签

融合标签将两个或多个列表作为参数并把它们融合在一起，如下所示：

```
<s:merge var="myMergedIterator">
  <s:param value="%{myList1}" />
  <s:param value="%{myList2}" />
  <s:param value="%{myList3}" />
</s:merge>
```

```
<s:iterator value="%{#myMergedIterator}">
  <s:property />
</s:iterator>
```

附加标签

附加标签将两个或多个列表作为参数并把它们附加在一起，如下所示：

```
<s:append var="myAppendIterator">
  <s:param value="%{myList1}" />
  <s:param value="%{myList2}" />
  <s:param value="%{myList3}" />
</s:append>
<s:iterator value="%{#myAppendIterator}">
  <s:property />
</s:iterator>
```

生成器标签

生成器标签生成基于提供的 val 属性的迭代器。下面的例子中，生成器标签生成了一个迭代器并用迭代器标签把它输出出来。

```
<s:generator val="%{'aaa,bbb,ccc,ddd,eee'}">
  <s:iterator>
    <s:property /><br/>
  </s:iterator>
</s:generator>
```


数据标签

Struts 2 的 **数据标签**主要用于操作显示在页面中的数据。以下列出的是重要的数据标签：

操作标签

该标签允许开发人员通过指定操作名称和可选的名称空间来从 JSP 页面直接调用操作。该标签的主题内容用于呈现来自操作的结果。在 struts.xml 文件中任何为该操作定义的结果处理器都将被忽略，除非指定了 `executeResult` 参数。

```
<div>Tag to execute the action</div>
<br />
<s:action name="actionTagAction" executeResult="true" />
<br />
<div>To invokes special method in action class</div>
<br />
<s:action name="actionTagAction!specialMethod" executeResult="true" />
```

包含标签

这些包含标签用于在一个 JSP 页面中包含另一个 JSP 文件。

```
<-- First Syntax -->
<s:include value="myJsp.jsp" />
<-- Second Syntax -->
<s:include value="myJsp.jsp">
  <s:param name="param1" value="value2" />
  <s:param name="param2" value="value2" />
</s:include>
<-- Third Syntax -->
<s:include value="myJsp.jsp">
  <s:param name="param1">value1</s:param>
  <s:param name="param2">value2</s:param>
</s:include>
```

bean 标签

这些 bean 标签实例化符合 JavaBeans 规范的类。该标签的主体可以包含一组参数元素来在那个类中设置任何设值方法。如果 `var` 属性设置在 `BeanTag` 中，那么它会把初始化的 bean 放到栈的上下文中。

```
<s:bean name="org.apache.struts2.util.Counter" var="counter">
  <s:param name="first" value="20"/>
  <s:param name="last" value="25" />
</s:bean>
```

日期标签

这些日期标签允许你以快速简单的方法设置日期格式。你可以指定一个自定义的格式（如 "dd/MM/yyyy hh:mm"），你可以生成简单的可读符号（如 "in 2 hours, 14 minutes"），或者你也可以用你的属性文件中的键 'struts.date.format' 来后到预定义的格式。

```
<s:date name="person.birthday" format="dd/MM/yyyy" />
<s:date name="person.birthday" format="%{getText('some.i18n.key')}" />
<s:date name="person.birthday" nice="true" />
<s:date name="person.birthday" />
```

参数标签

参数标签用于参数化其他标签。这个标签有以下两个参数。

? name（字符串）——参数的名称

? value（对象）——参数的值

```
<pre>
<ui:component>
  <ui:param name="key" value="[0]"/>
  <ui:param name="value" value="[1]"/>
  <ui:param name="context" value="[2]"/>
</ui:component>
</pre>
```

属性标签

属性标签用于获取值的属性，如果不存在指定的属性，那么就会默认为栈顶的属性。

```
<s:push value="myBean">
  <!-- Example 1: -->
  <s:property value="myBeanProperty" />
  <!-- Example 2: -->TextUtils
  <s:property value="myBeanProperty" default="a default value" />
</s:push>
```

push 标签

push 标签用于栈中 push 值的简单操作。

```
<s:push value="user">
  <s:property value="firstName" />
  <s:property value="lastName" />
</s:push>
```

set 标签

set 标签将一个值赋给指定范围中的变量。当你想要为一个复杂的表达式分配一个变量，每次只需要简单的引用这个变量而不需要引用这个复杂的表达式时，这个标签是非常有用的。可用的范围是 应用程序，会话，请求，页面 和 操作。

```
<s:set name="myenv" value="environment.name"/>
<s:property value="myenv"/>
```

文本标签

文本标签用于呈现 i18n 文本消息。

```
<!-- First Example -->
<s:i18n name="struts.action.test.i18n.Shop">
  <s:text name="main.title"/>
</s:i18n>

<!-- Second Example -->
<s:text name="main.title" />

<!-- Third Example -->
<s:text name="i18n.label.greetings">
  <s:param >Mr Smith</s:param>
</s:text>
```

url 标签

url 标签用于创建 URL。

```
<!-- Example 1 -->
<s:url value="editGadget.action">
  <s:param name="id" value="%{selected}" />
```

```
</s:url>
<-- Example 2 -->
<s:url action="editGadget">
  <s:param name="id" value="%{selected}" />
</s:url>
<-- Example 3-->
<s:url includeParams="get">
  <s:param name="id" value="%{'22'}" />
</s:url>
```

表单标签

form 标签的列表是 Struts UI 标签的子集。这些标签帮助呈现 Struts web 应用程序必需的用户接口，并且能够被划分为三种类别。本章将会带你浏览 UI 标签的这三种类别。

简单的 UI 标签

我们已经在我们的示例中使用过这些标签了，在本章中我们将一带而过。让我们看看带有几个简单的 UI 标签的简单的视图页面 `email.jsp`：

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<s:head/>
<title>Hello World</title>
</head>
<body>
  <s:div>Email Form</s:div>
  <s:text name="Please fill in the form below:" />
  <s:form action="hello" method="post" enctype="multipart/form-data">
    <s:hidden name="secret" value="abracadabra"/>
    <s:textfield key="email.from" name="from" />
    <s:password key="email.password" name="password" />
    <s:textfield key="email.to" name="to" />
    <s:textfield key="email.subject" name="subject" />
    <s:textarea key="email.body" name="email.body" />
    <s:label for="attachment" value="Attachment"/>
    <s:file name="attachment" accept="text/html,text/plain" />
    <s:token />
    <s:submit key="submit" />
  </s:form>
</body>
</html>
```

如果你了解 HTML，那么所有的标签都是非常普通的 HTML 标签，只不过在每个标签和不同的属性中带有一个额外的前缀 `s:`。当我们执行上述程序时，我们会得到如下所示的用户接口，前提是你已经为所有用到的键设置了合适的映射。

The screenshot shows a web browser window with the address bar set to `http://localhost`. The page content is titled "Email Form" and includes the instruction "Please fill in the form below:". The form consists of several input fields: "From:", "Password:", "To:", "Subject:", and "Body:". Below these is an "Attachment" section with a text input field and a "Browse..." button. At the bottom right of the form is a "[Submit]" button. The browser's status bar at the bottom right shows a magnifying glass icon and "100%".

正如显示的一样，`s:head` 生成了 Struts2 应用程序必需的 javascript 和 stylesheet。

接下来，我们有 `s:div` 和 `s:text` 元素。`s:div` 元素用于呈现 HTML Div 元素。这对不想把 HTML 和 Struts 标签混合到一起的用户来说是非常有用的。对于这些人，他们必须选择使用 `s:div` 元素来呈现 div。

`s:text` 元素用于在屏幕上呈现一个文本。

接下来，我们有类似的 `s:form` 标签。`s:form` 标签有一个操作属性，决定了表单提交的位置。因为在表单中有一个文件上传元素，我们必须把 `enctype` 设置为 `multipart`。否则，我们可以离开这个空白。

在表单标签的结尾，我们有 `s:submit` 标签。这是用于提交表单的。当表单被提交时，所有的表单的值都会提交给 `s:form` 标签中指定的操作。

在 `s:form` 标签内部，有一个隐藏的属性称为 `secret`。这会呈现在 HTML 中隐藏的元素。在我们的例子中，`"secret"` 元素有 `"abracadabra"` 值。这个元素对终端用户是不可见的且它用于将状态从一个视图传递给另一个视图。

接下来，我们有 `s:label`，`s:textfield`，`s:password` 和 `s:textarea` 标签。这是分别用于呈现标记、输入字段、密码和文本域的。我们已经在 "Struts – Sending Email" 例子中的操作中了解了这些标签的使用方法。在这里需要注意的很重要的一点是 `"key"` 属性的使用。`"key"` 属性是用于从属性文件中为这些控制提取标记的。在 Struts2 本地化，国际化章节中，我们已经介绍了这个特征。

然后，我们有 `s:file` 标签，用于呈现输入文件上传组件。这个组件允许用户上传文件。在这个例子中，我们使用的是 `s:file` 标签的 `"accept"` 属性来指定哪个文件类型是允许上传的。

最后我们有 s:token 标签。token 标签生成了一个唯一的 token，用于确认一个表单是否被提交了两次。

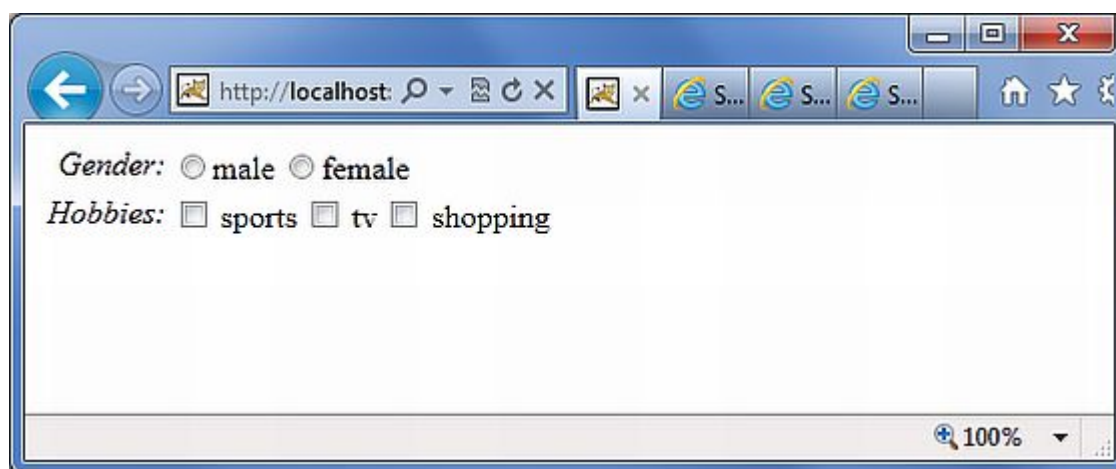
当呈现表单时，一个隐藏的变量就会被替换成 token 值。比如，token 是 "ABC"。当表单被提交时，Struts 过滤器检查与会话中存储的 token 不一致的 token。如果相匹配的话，它会把这个 token 从会话中移除出去。现在，如果表单被意外的重复提交了（或通过刷新或点击了浏览器的后退按钮），表单就会被重提交，其中 token 为 "ABC"。在这种情况下，过滤器会再次检查会话中与这个 token 不一致的 token。但是由于 token "ABC" 已经从会话中被删除了，它不会再次匹配，Struts 过滤器就会拒绝这个请求。

组 UI 标签

组 UI 标签是用于创建单选按钮和复选框的。让我们看一个带有复选框和单选按钮标签的简单的视图页面 HelloWorld.jsp:

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
<title>Hello World</title>
<s:head />
</head>
<body>
<s:form action="hello.action">
<s:radio label="Gender" name="gender" list="{ 'male','female' }" />
<s:checkboxlist label="Hobbies" name="hobbies"
list="{ 'sports','tv','shopping' }" />
</s:form>
</body>
</html>
```

当我们执行上述程序时，会得到类似如下所示的输出：



现在让我们看看这个例子。在第一个例子中，我们创建的是一个简单的带有标记 "Gender" 的单选按钮。单选按钮的名字属性是强制性的，所以我们指定了名字为 "Gender"。然后我们给这个 gender 提供了一个列表。列表用值 "male" 和 "female" 填充。因此，在输出中，我们得到了带有两个值的单选按钮。

在第二个例子中，我们创建的是复选框列表。这是用来收集用户爱好的。用户可能有多个爱好，因此我们使用的是复选框而不是单选按钮。复选框用列表 "sports", "Tv" 和 "Shopping" 填充。这把爱好以复选框列表的形式展现出来。

选择 UI 标签

让我们展示 Struts 提供的选择标签的不同的变量。请看如下所示的带有选择标签的简单的视图页面 HelloWorld.jsp:

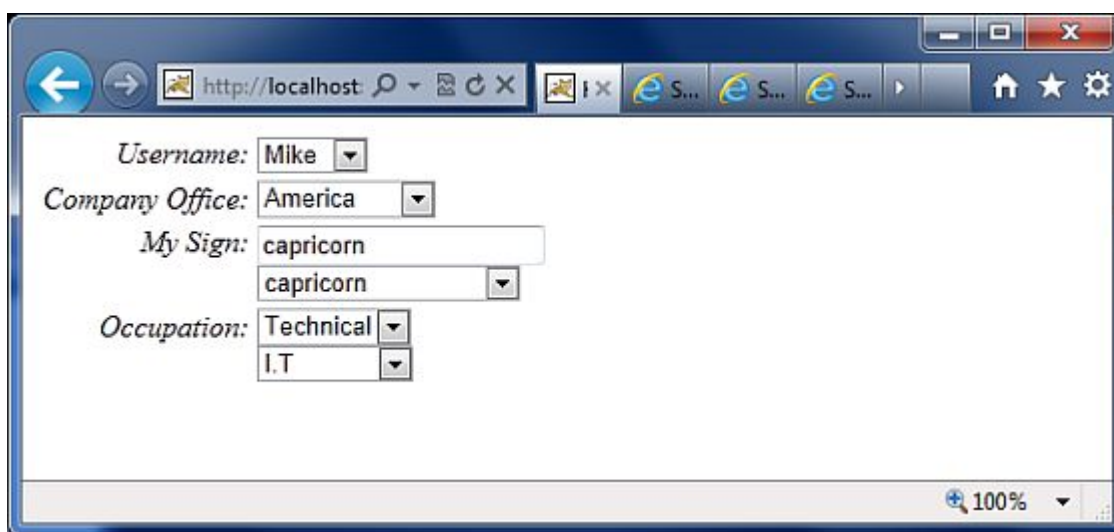
```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
<title>Hello World</title>
<s:head />
</head>
<body>
  <s:form action="login.action">
    <s:select name="username" label="Username"
      list="{ 'Mike', 'John', 'Smith' }" />

    <s:select label="Company Office" name="mySelection"
      value="%{ 'America' }"
      list="%{ #{'America': 'America' } }">
      <s:optgroup label="Asia"
        list="%{ #{'India': 'India', 'China': 'China' } }" />
      <s:optgroup label="Europe"
        list="%{ #{'UK': 'UK', 'Sweden': 'Sweden', 'Italy': 'Italy' } }" />
    </s:select>

    <s:combobox label="My Sign" name="mySign"
      list="#{ 'aries': 'aries', 'capricorn': 'capricorn' }"
      headerKey="-1"
      headerValue="--- Please Select ---" emptyOption="true"
      value="capricorn" />
    <s:doubleselect label="Occupation" name="occupation"
      list="{ 'Technical', 'Other' }" doubleName="occupations2"
      doubleList="top == 'Technical' ?
        { 'I.T', 'Hardware' } : { 'Accounting', 'H.R' }" />

  </s:form>
</body>
</html>
```

当我们执行上述程序时，我们会得到类似于如下所示的输出：



现在让我们仔细浏览上述情况

- ？ 首先，选择标签用于呈现 HTML 选择框。在第一个例子中，我们创建的是一个简单的选择框，带有名字 "username" 和标记 "username"。该选择框会由包含 Mike, John 和 Smith 名字的列表填充。
- ？ 在第二个例子中，我们的公司总部在美国。它也在亚洲和欧洲拥有全球办公室。我们想在选择框中显示办公室，但是我们想把全球办公室按照洲名分组。这就是 optgroup 的方便之处。我们使用 s:optgroup 标签来创建一个新组。给这个组一个标记和一个单独的列表。
- ？ 在第三个例子中，用到了组合框。组合框是输入字段和选择框的组合。用户可以从选择框中选择一个值，然后输入字段就会用用户选取的值自动填充。或者用户也可以直接输入一个值，那么选择框中的值就不会被选中。
- ？ 在我们的例子中有一个列出了星座的组合框。组合框只列出了四个星座，如果用户的星座不在组合框中，用户可以自己键入他自己的星座。我们还在选择框中添加了一个标题入口。标题入口显示在选择框的顶部。在我们的例子中我们想显示“请选择”。如果用户没有做出任何选择，那么我们假定值为 -1。在一些情况下，我们不希望用户选取空值。在这种情况下，设置 "emptyOption" 属性为 false。最后，在我们的例子中，我们提供了 "capricorn" 作为组合框的默认值。
- ？ 在第四个例子中，我们有一个双选。双选用于你想显示两个选择框的情况。在第一个选择框中选择的值决定了第二个选择框中出现的内容。在我们的例子中，第一个选择框显示 "Technical" 和 "Other"。如果用户选择 Technical，我们会在第二个选择框中显示 IT 和 Hardware。否则我们会显示 Accounting 和 HR。使用例子中展示的 "list" 和 "doubleList" 属性也有可能实现这一功能。

在上述例子中，我们做了一个比较来看顶部选择框是否就是 Technical。如果是的话，那么会显示 IT 和 Hardware。我们也需要给顶部框("name='Occupations'")和底部框(doubleName='occupations2')设置名称。

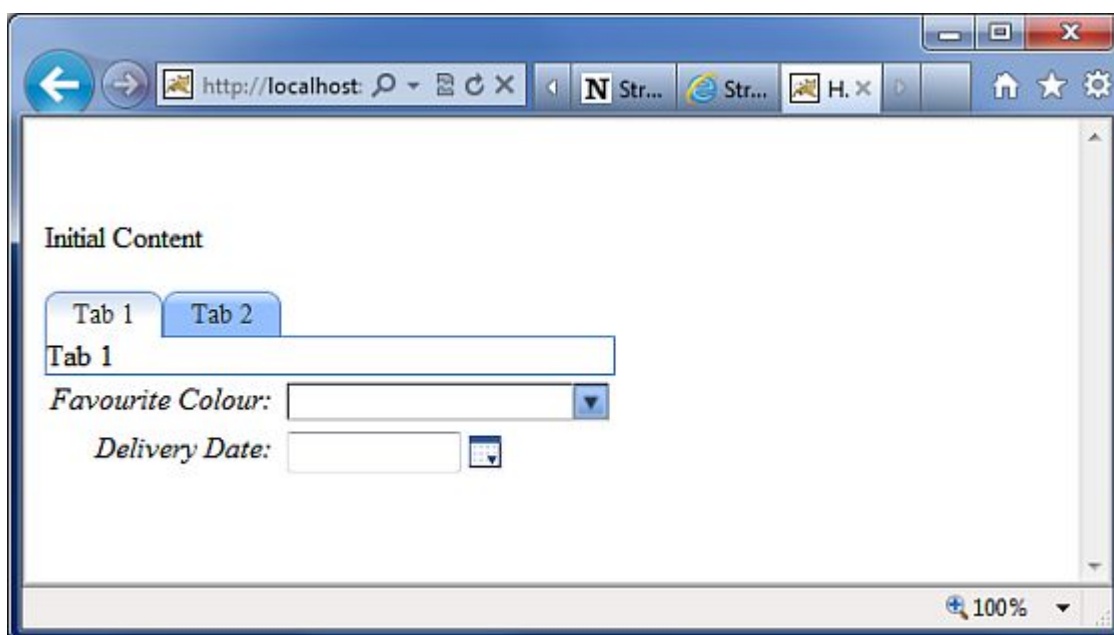
Ajax 标签

Struts 使用 DOJO 框架实现 AJAX 标签。首先，在介绍例子之前，你需要将 struts2-doj-plugin-2.2.3.jar 添加到你的类路径中。你可以从下载的 struts2 的 lib 文件夹中获取这个文件(C:\struts-2.2.3-all\struts-2.2.3\lib\struts2-doj-plugin-2.2.3.jar)。

在这次实践中，让我们修改 HelloWorld.jsp 文件，如下所示：

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<%@ taglib prefix="sx" uri="/struts-doj-tags"%>
<html>
<head>
<title>Hello World</title>
<s:head />
<sx:head />
</head>
<body>
<s:form>
<sx:autocompleter label="Favourite Colour"
  list="{ 'red','green','blue' }" />
<br />
<sx:datetimepicker name="deliverydate" label="Delivery Date"
  displayFormat="dd/MM/yyyy" />
<br />
<s:url id="url" value="/hello.action" />
<sx:div href="%{#url}" delay="2000">
  Initial Content
</sx:div>
<br />
<sx:tabbedpanel id="tabContainer">
  <sx:div label="Tab 1">Tab 1</sx:div>
  <sx:div label="Tab 2">Tab 2</sx:div>
</sx:tabbedpanel>
</s:form>
</body>
</html>
```

当我们运行上述例子时，会得到如下所示的输出：



现在让我们一步步的看这个例子。

首先需要注意的是用 prefix `sx` 添加的新的标签库。这(struts-dojotags) 是专门为 ajax 集成创建的标签库。

然后在 HTML 头信息内部，我们调用 `sx:head`。它会初始化 dojo 框架并为页面内的所有的 AJAX 调用做好准备。这一步是很重要的——没有 `sx:head` 初始化，你的 ajax 调用不会起作用。

首先我们得到 `autocomplete` 标签。`autocomplete` 标签看起来与下拉选框非常相似。它有三个常用的值 `red`，`green` 和 `blue`。但是它和下拉选框之间的区别是它是自动实现的。也就是说，如果你键入 `gr`，那么它就会用 `"green"` 填充。除此之外，该标签与我们之前提到的 `s:select` 标签就是非常相似的了。

接下来，我们有一个日期时间选择器。该标签创建了一个输入字段和一个紧挨输入字段的按钮。当按下按钮，就会弹出一个日期时间选择器。当用户选择日期后，该日期就会填充到该标签属性指定的表单的输入文本中。在我们的例子中，我们为日期指定了 `dd/MM/yyyy` 格式。

接下来我们为 `system.action` 文件创建一个 `url` 标签，该文件在之前的实践中已经创建过了。不一定非得是 `system.action` 文件——它可以是你之前创建的任何操作文件。然后我们得到一个 `div`，带有对 `url` 的超连接设置以及延迟设置为 2 秒。当你运行它的时候，`"Initial Content"` 会延迟 2 秒，然后 `div` 的内容会被 `hello.action` 执行中的内容取代。

最后我们得到一个带有两个标签的标签面板。标签是带有 `Tab 1` 和 `Tab 2` 标记的 `div`。

值得注意的是，在 Struts 中的 AJAX 标签集成仍然是一项有待改善的工作，该集成也随着每次发布慢慢的成熟起来。



3

Struts 2 集成



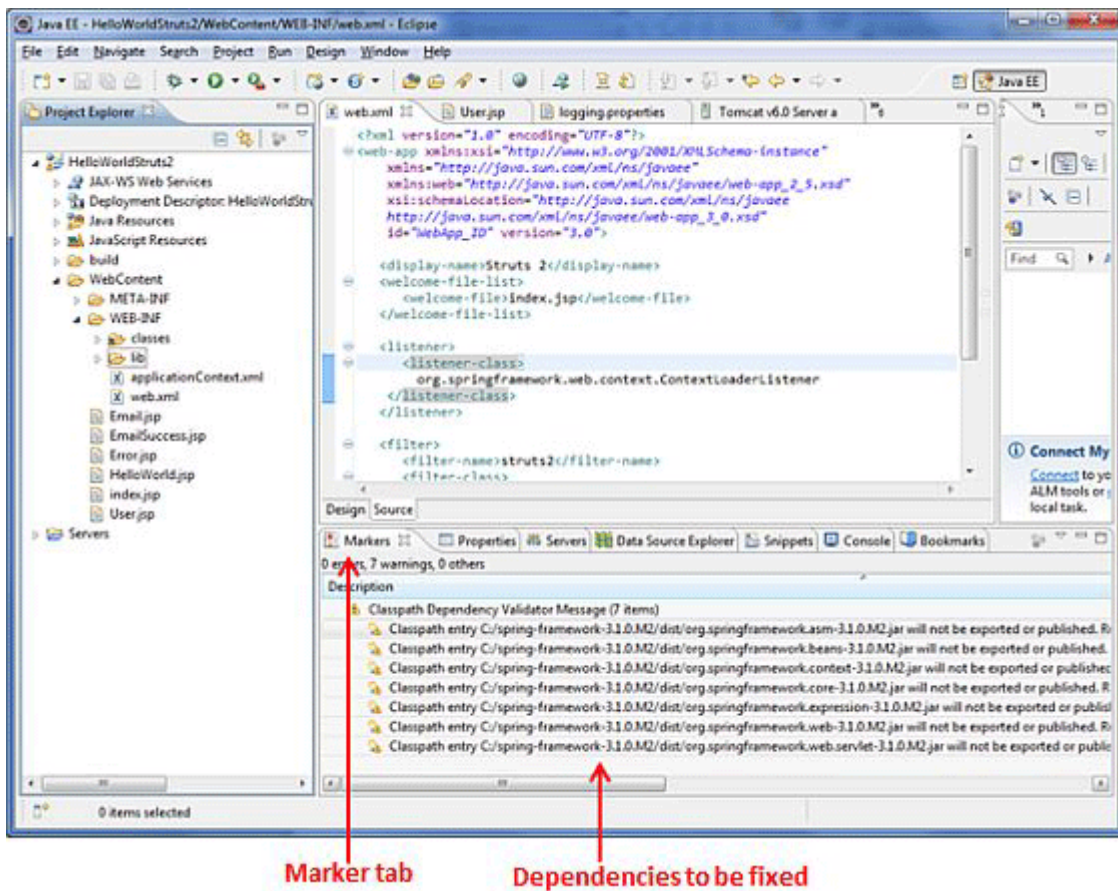
Spring 集成

Spring 是一个流行的 web 框架，提供了与大量普通 web 任务的简单集成。所以问题是，当我们已经有了 Struts 2 时，为什么还需要 Spring？好吧，Spring 不仅仅是一个 MVC 框架——它还提供了许多在 Struts 中不可用的东西。例如：对任何框架都有用的依赖注入。在本章中，我们将通过一个简单的示例来展示如何集成 Spring 和 Struts 2。

首先，你需要将如下所示的文件添加到 Spring 安装的项目的构建路径中。你可以从 <http://www.springframework.org/download> 下载并安装最新的 Spring 框架版本。

- ? org.springframework.asm-x.y.z.M(a).jar
- ? org.springframework.beans-x.y.z.M(a).jar
- ? org.springframework.context-x.y.z.M(a).jar
- ? org.springframework.core-x.y.z.M(a).jar
- ? org.springframework.expression-x.y.z.M(a).jar
- ? org.springframework.web-x.y.z.M(a).jar
- ? org.springframework.web.servlet-x.y.z.M(a).jar

最后，将 `struts2-spring-plugin-x.y.z.jar` 添加到你的 struts lib 目录 `WEB-INF/lib` 中。如果你使用的是 Eclipse，那么你可能会面临一个异常 `java.lang.ClassNotFoundException: org.springframework.web.context.ContextLoaderListener`。为了修复这个问题，进入 **Marker** 标签，然后一个接一个的右键单击类依赖，并且进行快速修复来发布/导出所有的依赖。最后，确保在 marker 标签下没有可用的依赖冲突。



现在让我们为 Struts-Spring 集成配置 web.xml 文件，如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
  <display-name>Struts 2</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
  <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>
</web-app>

```

在这里需要注意的一个重要的点是我们已经配置的监听器。加载 spring 上下文文件需要 `ContextLoaderListener`。Spring 的配置文件称为 `applicationContext.xml`，且它必须要放置在和 `web.xml` 文件同级的位置。

让我们创建一个简单的操作类，命名为 `User.java`，带有两个属性——`firstName` 和 `lastName`。

```

package com.tutorialspoint.struts2;
public class User {
    private String firstName;
    private String lastName;
    public String execute()
    {
        return "success";
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}

```

现在让我们创建 `applicationContext.xml` spring 配置文件并实例化 `User.java` 类。正如先前提到的一样，这个文件应该在 `WEB-INF` 文件夹下：

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <bean id="userClass" class="com.tutorialspoint.struts2.User">
        <property name="firstName" value="Michael" />
        <property name="lastName" value="Jackson" />
    </bean>
</beans>

```

```
</bean>
</beans>
```

正如我们在上面看见的一样，我们已经配置了 user bean 并且已经将值 Michael 和 Jackson 注入到 bean 中。我们还将这个 bean 命名为 "userClass"，这样我们就可以在任何地方重用它。接下来，我们在 WebContent 文件夹中创建 User.jsp：

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<title>Hello World</title>
</head>
<body>

    <h1>Hello World From Struts2 – Spring integration</h1>

    <s:form>
        <s:textfield name="firstName" label="First Name"/><br/>
        <s:textfield name="lastName" label="Last Name"/><br/>
    </s:form>

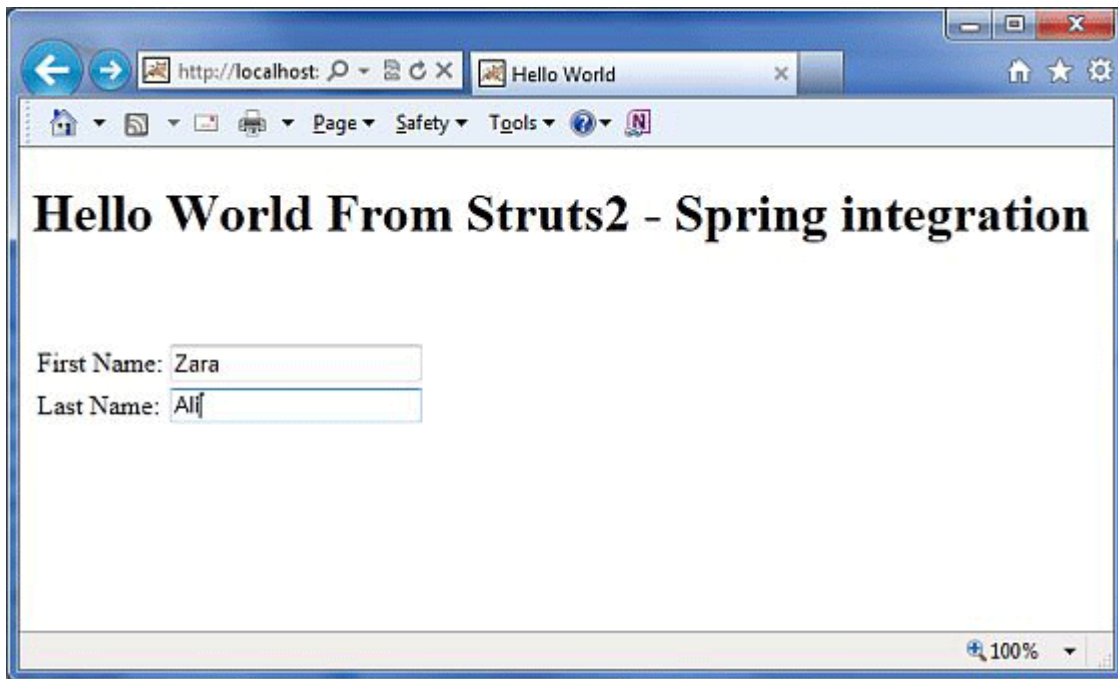
</body>
</html>
```

User.jsp 文件非常简单。它只用于一个目的——显示用户对象的 firstname 和 lastname 的值。最后，让我们将在 struts.xml 文件中用到的条目放到一起。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.devMode" value="true" />
    <package name="helloworld" extends="struts-default">
        <action name="user" class="userClass"
            method="execute">
            <result name="success">/User.jsp</result>
        </action>
    </package>
</struts>
```

需要注意的重要的一点是我们用 id userClass 来代表类。这意味着我们使用的是 spring 来为 User 类进行依赖注入。

现在鼠标右键单击项目名，然后单击 **Export > WAR File** 来创建一个 War 文件。然后将这个 WAR 文件部署到 Tomcat 的 web 应用程序目录中。最后，启动 Tomcat 服务器并尝试访问 URL `http://localhost:8080/HelloWorldStruts2/User.jsp`。这将呈现如下所示的画面：



现在我们已经了解了如何将这两个好的框架集成到一起。这包括 Struts – Spring 集成这一章。

Tiles 集成

在本章中，让我们浏览用 Struts 2 集成 Tiles 框架的步骤。Apache Tiles 是模板框架，用于简化 web 应用程序用户接口开发的。

首先，我们需要从 [Apache Tiles](#) 网站下载 tiles jar 文件。你需要将下列 jar 文件添加到项目的类路径中。

- ? tiles-api-x.y.z.jar
- ? tiles-compatible-x.y.z.jar
- ? tiles-core-x.y.z.jar
- ? tiles-jsp-x.y.z.jar
- ? tiles-servlet-x.y.z.jar

除了上述文件，我么还需要将下列来自 struts 2 的 jar 文件复制到你的 WEB-INF/lib 中。

- ? commons-beanutils-x.y.z.jar
- ? commons-digester-x.y.jar
- ? struts2-tiles-plugin-x.y.z.jar

现在让我们为 Struts-Tiles 集成配置 web.xml 文件，如下所示。这里有两个重要的点需要注意。首先，我们需要告诉 tiles 在哪里可以找到 tiles 配合文件 tiles.xml。在我们的例子中，它位于 /WEB-INF 文件夹下。接下来，我们需要初始化来自 Struts2 下载的 Tiles 监听器。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  id="WebApp_ID" version="2.5">
  <display-name>Struts2Example15</display-name>
  <context-param>
    <param-name>
      org.apache.tiles.impl.BasicTilesContainer.DEFINITIONS_CONFIG
    </param-name>
    <param-value>
      /WEB-INF/tiles.xml
    </param-value>
  </context-param>
</web-app>
```

```

</param-value>
</context-param>
<listener>
<listener-class>
    org.apache.struts2.tiles.StrutsTilesListener
</listener-class>
</listener>
<filter>
<filter-name>struts2</filter-name>
<filter-class>
org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
</filter-class>
</filter>
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>

```

接下来，让我们在 /WEB-INF 文件夹下创建 tiles.xml，其内容如下所示：

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC
    "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
    "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
<tiles-definitions>
    <definition name="baseLayout" template="/baseLayout.jsp">
        <put-attribute name="title" value="Template"/>
        <put-attribute name="banner" value="/banner.jsp"/>
        <put-attribute name="menu" value="/menu.jsp"/>
        <put-attribute name="body" value="/body.jsp"/>
        <put-attribute name="footer" value="/footer.jsp"/>
    </definition>
    <definition name="tiger" extends="baseLayout">
        <put-attribute name="title" value="Tiger"/>
        <put-attribute name="body" value="/tiger.jsp"/>
    </definition>
    <definition name="lion" extends="baseLayout">
        <put-attribute name="title" value="Lion"/>
        <put-attribute name="body" value="/lion.jsp"/>
    </definition>
</tiles-definitions>

```

接下来，我们在 `baseLayout.jsp` 文件中定义一个基本框架布局。它有五个可重用的/可覆盖的域。即 `title`, `banner`, `menu`, `body` 和 `footer`。我们为此基本布局提供了默认值，然后创建了从默认值中扩展的两个自定义值。`tiger` 布局与基本布局类似，除了它使用 `tiger.jsp` 作为它的主体，文本 "Tiger" 作为标题。类似的，`lion` 布局也与基本布局相似，除了它使用 `lion.jsp` 作为它的主体，文本 "Lion" 作为标题。

让我们看看单独的 `jsp` 文件。以下是 `baseLayout.jsp` 文件的内容：

```
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title><tiles:insertAttribute name="title" ignore="true" />
</title>
</head>

<body>
<tiles:insertAttribute name="banner" /><br/>
<hr/>
<tiles:insertAttribute name="menu" /><br/>
<hr/>
<tiles:insertAttribute name="body" /><br/>
<hr/>
<tiles:insertAttribute name="footer" /><br/>
</body>
</html>
```

这里我们将有 `tiles` 属性的 HTML 页面放到一起。我们将 `tiles` 属性放到我们需要的位置。接下来，让我们创建 `banner.jsp` 文件，其内容如下所示：

```

```

`menu.jsp` 文件有如下所示的行，能够链接到 `TigerMenu.action` 和 `LionMenu.action` struts 操作。

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

<a href="<s:url action="tigerMenu"/>">Tiger</a><br>
<a href="<s:url action="lionMenu"/>">Lion</a><br>
```

`lion.jsp` 文件有如下所示的内容：

```

The lion
```

`tiger.jsp` 文件有如下所示的内容：

```

The tiger
```

接下来，让我们创建操作类文件 `MenuAction.java`，其内容如下所示：

```
package com.tutorialspoint.struts2;
import com.opensymphony.xwork2.ActionSupport;
public class MenuAction extends ActionSupport {
    public String tiger() { return "tiger"; }
    public String lion() { return "lion"; }
}
```

这是一个非常简单的类。我们声明两种方法 `tiger()` 和 `lion()` 来分别返回输出 `tiger` 和 `lion`。让我们把它们都放到 `struts.xml` 文件中：

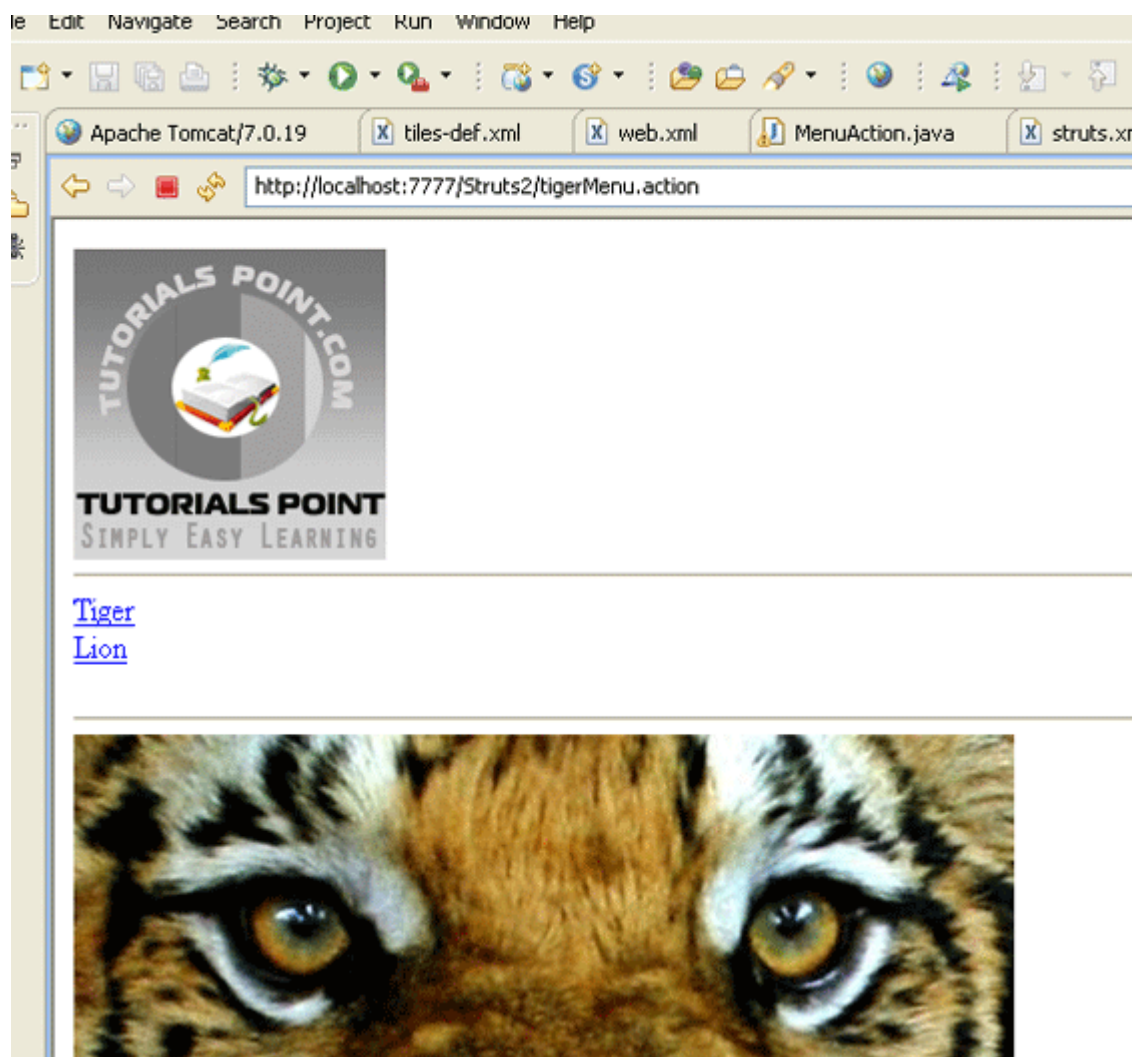
```
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="default" extends="struts-default">
        <result-types>
            <result-type name="tiles"
                class="org.apache.struts2.views.tiles.TilesResult" />
        </result-types>
        <action name="*Menu" method="{1}"
            class="com.tutorialspoint.struts2.MenuAction">
            <result name="tiger" type="tiles">tiger</result>
            <result name="lion" type="tiles">lion</result>
        </action>
    </package>
</struts>
```

检查在上述文件中做的工作。首先，我们为视图技术声明了一个命名为 "tiles" 的新的结果类型，而不是简单的 `jsp`，作为我们正在使用的 `tiles`。Struts2 有 `Tiles` 视图结果类型支持，所以我们创建的结果类型 "tiles" 会成为 "org.apache.struts2.view.tiles.TilesResult" 类。

接下来，如果说请求是为 `/tigerMenu.action` 的，那么将用户带到 `tiger tiles` 页面，如果请求是为 `/lionMenu.action` 的，那么将用户带到 `lion tiles` 页面。

我们通过使用一些常规表达式来实现这点。在我们的操作定义中，我们说匹配模式 "`*Menu`" 将由这个操作处理。匹配方法包括在 `MenuAction` 类中。也就是，`tigerMenu.action` 会包括 `tiger()` 且 `lionMenu.action` 会包括 `lion()`。然后我们需要将产生的结果映射到适当的 `tiles` 页面中。

现在鼠标右键单击项目名，然后单击 `Export > WAR 文件` 来创建一个 `War` 文件。然后在 `Tomcat` 的应用程序目录中部署该 `WAR` 文件。最后，启动 `Tomcat` 服务器并尝试访问 URL `http://localhost:8080/HelloWorldStruts2/tigerMenu.jsp`。这会呈现如下所示的画面：



同样的，如果你进入 lionMenu.action 页面，那么你将会看到 lion 页面，它使用的相同的 tiles 布局。

Hibernate 集成

Hibernate 是一个高性能的对象/关系持久性，能够查询在开源 GNU Lesser General Public License(LGPL) 下的服务较小，并且是免费下载的。在这一章，我们要学习如何用 Hibernate 集成实现 Struts 2。

数据库设置

在本教程中，我将使用 “struts2_tutorial” MySQL 数据库。我使用用户名 “root”、无密码来连接到我计算机中的数据库。首先，你需要运行以下脚本。这个脚本创建一个新表，名为 **student**，并在表中创建几条记录：

```
CREATE TABLE IF NOT EXISTS `student` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `first_name` varchar(40) NOT NULL,  
  `last_name` varchar(40) NOT NULL,  
  `marks` int(11) NOT NULL,  
  PRIMARY KEY (`id`)  
);  
  
--  
## -- Dumping data for table `student`  
INSERT INTO `student` (`id`, `first_name`, `last_name`, `marks`)  
VALUES(1, 'George', 'Kane', 20);  
INSERT INTO `student` (`id`, `first_name`, `last_name`, `marks`)  
VALUES(2, 'Melissa', 'Michael', 91);  
INSERT INTO `student` (`id`, `first_name`, `last_name`, `marks`)  
VALUES(3, 'Jessica', 'Drake', 21);
```

Hibernate 配置

接下来让我们创建 hibernate.cfg.xml 文件，它是 hibernate 的配置文件。

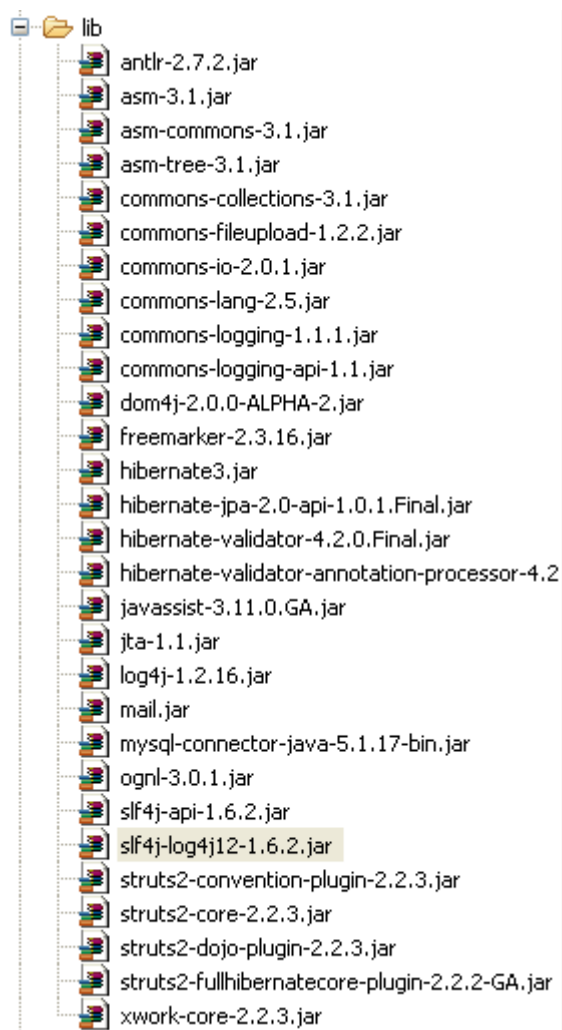
```
<?xml version='1.0' encoding='utf-8'?>  
<!DOCTYPE hibernate-configuration PUBLIC  
"-//Hibernate/Hibernate Configuration DTD//EN"  
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  
<hibernate-configuration>  
  <session-factory>  
    <property name="hibernate.connection.driver_class">c  
      om.mysql.jdbc.Driver  
    </property>  
    <property name="hibernate.connection.url">  
      jdbc:mysql://www.tutorialspoint.com/struts_tutorial  
    </property>  
    <property name="hibernate.connection.username">root</property>
```

```
<property name="hibernate.connection.password"></property>
<property name="hibernate.connection.pool_size">10</property>
<property name="show_sql">true</property>
<property name="dialect">
    org.hibernate.dialect.MySQLDialect
</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<mapping class="com.tutorialspoint.hibernate.Student" />
</session-factory>
</hibernate-configuration>
```

让我们仔细检查这个 hibernate 配置文件。首先，我们声明我们使用的是 MySQL 驱动。然后我们为连接到数据库声明 jdbc url。接下来我们声明连接的用户名，密码和池的大小。同时我们也表明通过将 "show_sql" 设置为 true，我们会在日志文件中看到这个 SQL。请仔细浏览 hibernate 教程来理解这些属性的含义。最后，我们将 mapping 类设置为 com.tutorialspoint.hibernate.Student，也就是我们在本章中将要创建的。

环境设置

接下来你需要为这个项目创建许多 jar 文件。附件是所需的 JAR 文件的完整列表截图：



大多数 JAR 文件可以作为你的 struts 发行版的一部分获取。如果你有一个已安装的应用程序服务器，如 glassfish, websphere 或 jboss，那么你就可以从应用程序服务器的 lib 文件夹中得到大部分剩余的 jar 文件。如果没有的话，那么你可以分别下载这些文件：

- ? Hibernate jar files – [Hibernate.org](http://hibernate.org)
- ? Struts hibernate plugin – [Struts hibernate plugin](#)
- ? JTA files – [JTA files](#)
- ? Dom4j files – [Dom4j](#)
- ? SLF4J files – [SLF4J](#)
- ? log4j files – [log4j](#)

剩余的文件，你应该能够从你的 struts2 发行版中得到。

Hibernate 类

现在让我们为 hibernate 集成创建必需的 java 类。下面是 Student.java 的内容：

```
package com.tutorialspoint.hibernate;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="student")
public class Student {
    @Id
    @GeneratedValue
    private int id;
    @Column(name="last_name")
    private String lastName;
    @Column(name="first_name")
    private String firstName;
    private int marks;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public int getMarks() {
        return marks;
    }
    public void setMarks(int marks) {
```

```

    this.marks = marks;
}
}

```

这是一个 POJO 类，代表了每个 Hibernate 指定的 student 表。它有 id, firstname 和 lastname 属性，与 student 表中的列名相对应。下面让我们创建 StudentDAO.java 文件，如下所示：

```

package com.tutorialspoint.hibernate;
import java.util.ArrayList;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.googlecode.s2hibernate.struts2.plugin.\
    annotations.SessionTarget;
import com.googlecode.s2hibernate.struts2.plugin.\
    annotations.TransactionTarget;
public class StudentDAO {
    @SessionTarget
    Session session;
    @TransactionTarget
    Transaction transaction;
    @SuppressWarnings("unchecked")
    public List<Student> getStudents()
    {
        List<Student> students = new ArrayList<Student>();
        try
        {
            students = session.createQuery("from Student").list();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        return students;
    }
    public void addStudent(Student student)
    {
        session.save(student);
    }
}

```

StudentDAO 类是 Student 类的数据库访问层。它有列出所有 students 和保存一个新的 student 记录的方法。

Action 类

下述文件 `AddStudentAction.java` 定义了 action 类。在这里我们同样有两个 action 方法 – `execute()` 和 `listStudents()`。`execute()` 方法用来添加新的 student 记录。我们使用 dao 的 `save()` 方法来获取。另一个方法，`listStudents()` 用来列出 students。我们使用 dao 的 `list` 方法来得到所有 student 的列表。

```
package com.tutorialspoint.struts2;
import java.util.ArrayList;
import java.util.List;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;
import com.tutorialspoint.hibernate.Student;
import com.tutorialspoint.hibernate.StudentDAO;
public class AddStudentAction extends ActionSupport
    implements ModelDriven<Student>{
    Student student = new Student();
    List<Student> students = new ArrayList<Student>();
    StudentDAO dao = new StudentDAO();
    @Override
    public Student getModel() {
        return student;
    }
    public String execute()
    {
        dao.addStudent(student);
        return "success";
    }
    public String listStudents()
    {
        students = dao.getStudents();
        return "success";
    }
    public Student getStudent() {
        return student;
    }
    public void setStudent(Student student) {
        this.student = student;
    }
    public List<Student> getStudents() {
        return students;
    }
    public void setStudents(List<Student> students) {
        this.students = students;
    }
}
```

```
}
}
```

你会注意到我们正在实现 `ModelDriven` 接口。当你的 `action` 类正在处理一个具体的模型类（如 `Student`）而不是单个的属性（如 `firstName`，`lastName`）时，该接口就会被用到。`ModelAware` 接口需要你实现一个方法来返回这个模型。在我们的例子中，我们返回的是 “student” 对象。

创建视图文件

现在让我们创建 `student.jsp` 视图文件，其内容如下所示：

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
<title>Hello World</title>
<s:head />
</head>
<body>
<s:form action="addStudent">
<s:textfield name="firstName" label="First Name"/>
<s:textfield name="lastName" label="Last Name"/>
<s:textfield name="marks" label="Marks"/>
<s:submit/>
<hr/>
<table>
<tr>
<td>First Name</td>
<td>Last Name</td>
<td>Marks</td>
</tr>
<s:iterator value="students">
<tr>
<td><s:property value="firstName"/></td>
<td><s:property value="lastName"/></td>
<td><s:property value="marks"/></td>
</tr>
</s:iterator>
</table>
</s:form>
</body>
</html>
```

`student.jsp` 非常简单。在最上面一节中，我们有一个要提交到 “`addStudent.action`” 的表单。它带有 `firstName`，`lastName` 和标记。由于 `addStudent` 操作与 `ModelAware` “`AddStudentAction`” 绑定了，那么就会自动创建一个带有 `firstName`，`lastName` 和自动填充标记的 `student` bean。

在最下面一节中，我们查看了 `student` 列表（请看 `AddStudentAction.java`）。我们遍历该列表，并显示表中的 `first name`，`last name` 和标记的值。

Struts 配置

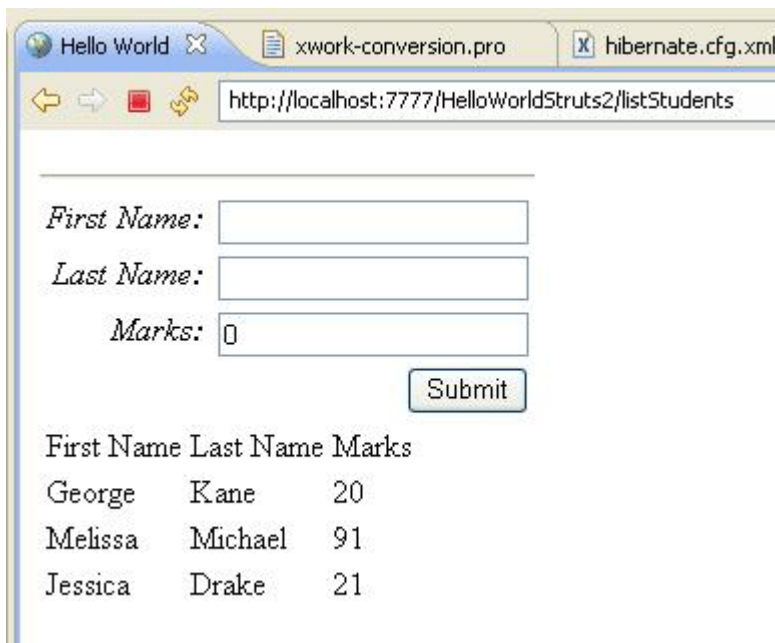
让我们使用 `struts.xml` 将全部的东西放到一起：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.devMode" value="true" />
    <package name="myhibernate" extends="hibernate-default">
        <action name="addStudent" method="execute"
            class="com.tutorialspoint.struts2.AddStudentAction">
            <result name="success" type="redirect">
                listStudents
            </result>
        </action>
        <action name="listStudents" method="listStudents"
            class="com.tutorialspoint.struts2.AddStudentAction">
            <result name="success">/students.jsp</result>
        </action>
    </package>
</struts>
```

在这里需要注意的最重要的事情是我们的 "myhibernate" 包扩展了命名为 "hibernate-default" 的 struts2 默认包。然后我们声明两个操作——addStudent 和 listStudents。addStudent 在 AddStudentAction 类中调用 execute()，调用成功之后，会调用 listStudents 操作方法。

listStudent 操作方法在 AddStudentAction 类中调用 listStudents()，并把 student.jsp 作为视图来使用。

现在鼠标右键单击项目名，点击 **Export > WAR 文件** 来创建一个 War 文件。然后在 Tomcat 的 web 应用程序目录中部署这个 WAR 文件。最后，启动 Tomcat 服务器并尝试访问 URL `http://localhost:8080/HelloWorldStruts2/student.jsp`。这将显示如下所示的画面：



The screenshot shows a web browser window with three tabs: 'Hello World', 'xwork-conversion.pro', and 'hibernate.cfg.xml'. The address bar shows the URL 'http://localhost:7777/HelloWorldStruts2/listStudents'. The page contains a form with three input fields: 'First Name:', 'Last Name:', and 'Marks:'. The 'Marks' field has the value '0'. Below the form is a 'Submit' button. Below the button is a table with three columns: 'First Name', 'Last Name', and 'Marks'. The table contains three rows of data: George Kane 20, Melissa Michael 91, and Jessica Drake 21.

First Name	Last Name	Marks
George	Kane	20
Melissa	Michael	91
Jessica	Drake	21

在最上面一节中，我们得到了一个表单，能够为新的 student 记录输入值，在最下面一节中列出了数据库中的 students。请继续添加新的 student 记录并按下提交。每次你按下提交按钮后，屏幕会刷新并显示更新的列表。



4

Struts 2 有用的资源



有用的资源

以下资源包含关于 Struts 2 的附加说明。请使用它们来获得更多的对这一主题的深入了解。

Struts 2 中可用的链接

- ? [Apache Struts 项目](#) – 提供关于 Struts 材料的 Apache 的官方网站。
- ? [关于 JSP 的 Sun 网站](#) – 提供关于 JSP 材料的 Sun 的官方网站。
- ? [关于 Servlets 的 Sun 网站](#) – 提供关于 Servlets 材料的 Sun 的官方网站。
- ? [JSP 引擎 – Tomcat](#) – Apache Tomcat 是一个开放源码软件，可以实现 Java Servlet 和 JavaServer Pages 技术。
- ? [MySQL Connector/J](#) – MySQL Connector/J 是 MySQL 的官方 JDBC 服务器。
- ? [Java™ 教程](#) – Java 教程对于想要使用 Java 编程语言创建应用程序的程序员来说是实用指南。
- ? [免费下载 Java](#) – 现在为你的计算机现在 Java 吧！
- ? [Sun 开发人员社区](#) – Sun 微系统的官方网站，列出了所有的 API 文档、最新的 Java 技术、书籍和其他资源。

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台



更多信息请访问 

<http://wiki.jikexueyuan.com/project/struts-2/>