在struts项目中jsp页面中使用了struts标签，如果不经过struts转发而是直接通过url访问页面，会出现标签异常问题。

# 1 Struts2 概述

Struts2 是目前较为普及和成熟的基于MVC设计模式的web应用程序框架，它不仅仅是Struts1 的升级版本，更是一个全新的Struts架构。最初，是以WebWork框架和Struts框架为基础，通过提供增强和改进的Struts框架，进而实现简化web技术人员开发工作的目标。不久之后，Webwork框架和Struts社区联合创造了现在流行的Struts2框架。

## 1.1 Struts2 框架的优点

了解了这几个主要的优点，会促使你考虑使用Struts2 ：

**POJO表单及POJO操作** - Struts2 去除掉了Struts框架中的Action Forms部分。在Struts2框架下，你可以用任何一POJO来接收表单输入，同样的，你可以把任一POJO视为一个Action类。

**标签支持** - Struts2 改进了标签表单，而新的标签可让开发人员减少代码编写量。

**AJAX支持** - Struts2 被认可接收进Web 2.0技术，并创建了功能非常类似于标准的Struts2 标签的AJAX标签，把AJAX支持整合进其结果中。

**易于整合** - Struts有多种整合方式可使用，现在与其他类型的框架，如Spring、Tiles、SiteMesh之类的，整合更为容易了。

**模板支持** - 支持使用模板生成视图。

**插件支持** - 有大量的插件可用于Struts2，而使用插件可以增强和扩大Struts2 核心行为。

**性能分析** - Struts2 为调试和配置应用程序提供综合的性能分析，此外，Struts也以嵌入调试工具的形式提供集成调试。

**易于修改标签** - 在Struts2 中，可使用Freemarker的模板对标签标记进行调整，而修改标签不需要JSP或是Java知识，基本的HTML、XML和CSS知识就足够了。

**促进减少配置** - Struts2 使用各种设置的默认值促进减少配置，而你不需要再配置什么除非是偏离了Struts2 设定的默认设置。

**视图技术** - Struts2 为多种视图选项（JSP、Freemarker、Velocity、XSLT等）提供支持。

以上是使Struts2 成为准企业框架的十大优点。

## 1.2 Struts2 框架的缺点

尽管Struts2 有一大列的优点，但我们还是要提到关于它的一些仍需不断改进的缺点：

**更大的学习曲线** - 使用Struts MVC，你必须要熟悉JSP、Servlet APIs标准以及一个大型、复杂的框架。

**文档缺乏** - 相比于Servlet和JSP APIs标准，Struts的在线资源较少，许多初学者会发现Apache在线文档混乱并缺乏整理。

**不够透明** - 相比于使用正常的基于Java的Web应用程序，使用Struts的应用程序有许多是进行在后台，这使得框架不易于理解。

最后说明一点，一个好的框架应该提供各种类型的应用程序都可以使用的通用行为，Struts2 是最好的Web框架之一，并频繁用于RIA（Rich Internet Applications）的发展。

# 2 Struts2 MVC架构

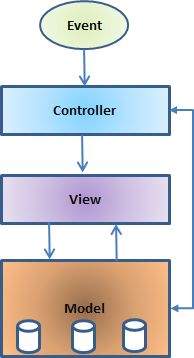
模型（Model）-视图（View）-控制器（Controller），通常简称MVC，是一种开发web应用程序的软件设计模式。该软件设计模式由以下三部分组成:

**模型**——属于软件设计模式的底层基础，主要负责数据维护。

**视图**——这部分是负责向用户呈现全部或部分数据。

**控制器**——通过软件代码控制模型和视图之间的交互。

MVC普及的原因在于它区分了应用程序的逻辑层和用户界面层，并支持开发关注点的分离。在MVC模式下，控制器接收了所有来自应用程序的请求后，调用模型去准备视图所需要的数据，然后视图使用由控制器提供的数据最终生成一个可视的响应。MVC的抽象概念可通过以下图6形进行表述：



## 2.1模型

模型主要负责管理应用程序的数据，它通过响应视图的请求和控制器的指令来更新自身的数据。

## 2.2 视图

通过控制器的指令触发所展现的一种特殊的数据格式。它们是基于像JSP、ASP、PHP之类模板系统的脚本，较易与AJAX技术进行整合。

## 2.3 控制器

控制器负责响应用户输入并执行数据模型对象的交互。控制器在接收、确认输入后执行修改数据模型状态的业务操作。  
Struts2是一个以MVC为基础的框架。在接下来的章节,让我们看看如何在Struts2中使用MVC。

# 3 Struts2 架构

## 3.1 过程分析：

用户发出请求-》调度过滤器-》选择适合的action进行分派-》拦截器拦截处理-》到达指定的action-》在action内部（数据校验、类型转换）-》获取结果和结果类型形成使视图层-》返回页面

从页面获取数据：

页面发送请求到Controller：控制层由调度过滤器 和 拦截器 组成

处理数据：请求到底模型层

数据处理：数据校验 类型转换

得到结果和结果类型形成视图

返回数据：将视图返回给用户

值栈和OGNL

异常处理

从一个高水平角度看，Struts2 是一个MVC拉动的（或MVC2）框架，Struts2 的模型-视图-控制器模式是通过以下五个核心部分进行实现的：

操作（Actions）

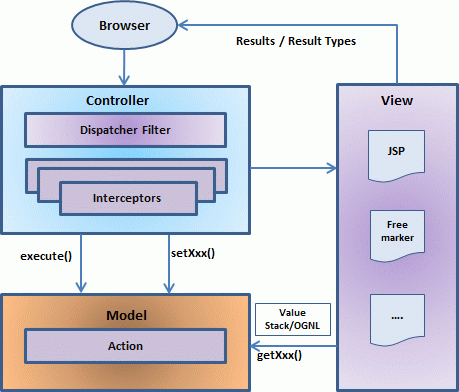
拦截器（Interceptors）

值栈（Value Stack）/OGNL

结果（Result）/结果类型

视图技术

而Struts2 与传统的MVC框架略有不同，因为它由Action扮演模型的角色，而不是控制器，虽然这样会有一些重叠。



上图描述了Struts2 高级系统架构下的模型、视图及控制器。控制器是通过Struts2 分派servlet的调度过滤器以及拦截器进行实现，模型是通过Actions进行实现，而视图则是结果类型和结果的结合。值栈和OGNL提供共同的路线、链接以及与其他组件之间的集成。

除了上述部分，还有许多组件相关的信息。web应用程序组件、Actions组件、拦截器组件、结果组件等等。

这些是Struts2 MVC模式的体系结构概述，在随后的章节中，我们将详细了解各个部分。

## 3.2 请求生命周期

通过上述图片的描述，我们可以依照下面几点解释在Struts2 中用户的请求生命周期：

1 用户发送一个资源需求的请求到服务器（例如：页面）。

2 核心控制器查看请求后确定适当的动作。

3 使用验证、文件上传等配置拦截器功能。

4 执行选择的动作来完成请求的操作。

5 另外，如果需要的话，配置的拦截器可做任何后期处理。

6 最后，由视图显示结果并返回给用户。

# 4 简单示例+配置解析

核心：

控制中心：Struts1 中DispatchFilter ； Struts2 中 StrutsPrepareAndExcutorFilter （其他还有拦截器，暂时不靠考虑）

模型 ：action类（由action类充当model）（其他还有校验器和转换器 异常处理器等，暂时不考虑）

视图 ：JSP页面

## 4.1 web.xml 配置好调度过滤器

StrutsPrepareAndExcutorFilter

url-parttern可以配置为\*.action

也可以配置为/\*

<filter>

<filter-name>strutsPrepareAndExecuteFilter</filter-name>

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

</filter>

<filter-mapping>

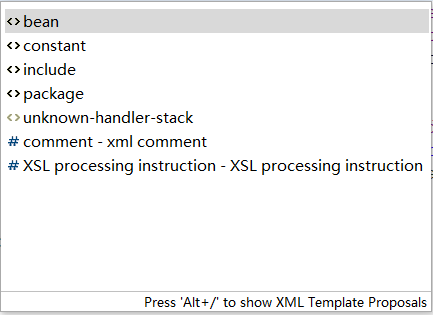
<filter-name>strutsPrepareAndExecuteFilter</filter-name>

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

</filter-mapping>

## 4.2 sturts.xml 配置类 整合模型和页面交互

### 标签简单解析



主要标签：

#### 1 bean

#### 2 constant（常量）：主要用于配置struts：比如：国际化 开发者模式 图表模型

<constant name="struts.devMode" value="true" /> 开发者模式

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

<constant name="struts.custom.i18n.resources" value="cn.hu.i18n.resource" /> 国际化文件

#### 3 include （包含）: 主要用于引入分模块的struts.xml文件

#### 4 package：是设置 action和view 映射关系的主要标签

##### 4.1 package标签属性如下图：核心属性是

·

name ：任意名称

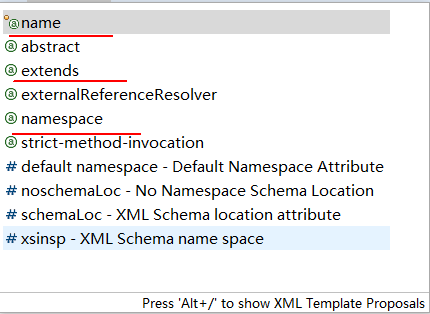
namespace ：/ 或 /data 等，可以理解为是一种路由。

如：<http://localhost:8080/struts2/data/hello.action> 这样的可以访问，如果去掉data则无法访问

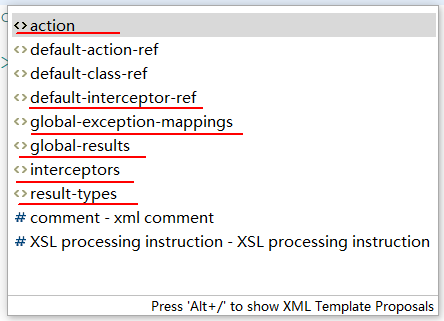
这样不同的package设置不同的namespace，就可以实现路由效果。

extends ：顾名思义，就是继承的意思。一般是继承struts-default 、 json-default

struts-default是默认的设置包，他包含了基本的拦截器设置。



##### 4.2 package的子级标签



global-results：全局结果，不需要表明action类等，只设置结果+视图，所有返回对应结果的请求都返回到该页面上

<global-results>

<result name=*"index"*>/index.jsp</result>

</global-results>

interceptors ：拦截器设置

default-interceptor-ref ：默认拦截器

##### action ：是最为核心的标签，他是设置 模型和视图 映射的管家标签

action的属性：

name:就是url的访问名称

class：就是处理数据的action

method：就是指定动作中的指定方法

<action name=*"hello"* class=*"cn.htb.MyAction"* method=*"getData"*>

<result name=*"data"*>/data.jsp</result>

</action>

action的子级标签：

result是主要设置映射的标签：name type

name是方法结果值

type是设置转发 ：redirectAction 或 异步：json 等

转发：参数是其他action的name

<result name="endBuy" type="redirectAction">taskAction\_queryTask</result>

异步：json格式的异步

<result name="inStore" type="json">

<param name="root">action</param>

<param name="includeProperties">

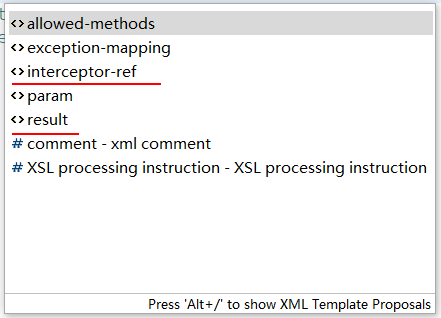
odm\.uuid,

odm\.num,

odm\.lastNum

</param>

</result>



interceptor-ref ：是拦截器只对当前action有效

### 简单的映射设置：

最基本的设置 package action result

不涉及请求路由namespace

不涉及拦截器 interceptors

不涉及全局结果 globalResult

不涉及转发和异步

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

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

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

<action name=*"hello"* class=*"cn.htb.MyAction"* method=*"getData"*>

<result name=*"data"*>/data.jsp</result>

</action>

</package>

</struts>

## 4.3 action 代码

**package** cn.htb;

**public** **class** MyAction {

**public** String getData(){

System.***out***.println("测试数据");

**return** "data";

}

}

## 4.4 测试

浏览器访问：

访问路径：没有namespace路由时：ip:port/项目名/action的name 如：<http://localhost:8080/struts2/hello.action>

有namespace路由时：ip:port/项目名/路由值/action的name 如：<http://localhost:8080/struts2/data/hello.action>

查看是否获取结果。

# 5 Struts2 配置文件

本章节将带你学习Struts2 应用程序所需的基本配置。在这里可以看到哪些将被配置到一些重要的配置文件中：web.xml、struts.xml、struts-config.xml及struts.properties。

实际上，你可以继续依赖于使用web.xml和struts.xml配置文件，并且你已经在前面的章节中了解到，我们的示例是使用这两个文件运作的，不过为了让你了解更多，我们还是再来说明一下其他的文件。

## 5.1 web.xml文件

web.xml配置文件是一种J2EE配置文件，决定servlet容器的HTTP元素需求如何进行处理。它严格来说不是一个Struts2 配置文件，但它是Struts2 运作所需要进行配置的文件。

正如前面所讨论的，这个文件为每个web应用程序提供接入点。在部署描述符（web.xml）中，Struts2 应用程序的接入点将会定义为一个过滤器。因此我们将在web.xml里定义一个FilterDispatcher类的接入点，而这个web.xml文件需要在**WebContent/WEB-INF**文件夹下创建。

如果你开始时没有模板或工具（比如Eclipse或Maven2）的辅助来生成，那这就是第一个你需要配置的文件。下面是我们在上一个例子中用到的web.xml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，我们将Struts2 过滤器映射到 **/\***，而不是**/\*.action**，这意味着所有的url都会被Struts过滤器解析。当我们学到关于注解的章节时会对这个进行讨论。

**注意：**自2.1.3版本开始，ActionContextCleanUp和FilterDispatcher都由StrutsPrepareAndExecuteFilter代替。

## 5.2 struts.xml文件

**struts.xml**文件包含有随着Actions的开发你将要修改的配置信息。它可用于覆盖应用程序的默认设置，例如：struts.devMode=false 以及其他定义为属性文件的设置。这个文件可在**WEB-INF/classes**文件夹下创建。

让我们来看一下在前一章节中阐述的Hello World示例里创建的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="cn.w3cschool.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>

首先要注意的是**DOCTYPE**（文档类型）。如我们的示例所示，所有的Struts配置文件都需要有正确的doctype。<struts>是根标记元素，在其下，我们使用<package>标签声明不同的包。 这里的<package>标签允许配置的分离和模块化。这在你进行一个大的项目并且项目分为多个不同的模块时，是非常有用的。

如果您的项目有三个域：business\_applicaiton、customer\_application和staff\_application的话，你可以创建三个包，并将相关的Actions存储到相应的包中。 <package>标签具有以下属性：

|  |  |
| --- | --- |
| **属性** | **描述** |
| name（必需） | 为package的唯一标识 |
| extends | 指定package继承另一package的所有配置。通常情况下，我们使用struts-default作为package的基础。 |
| abstract | 定义package为抽象的。如果标记为true，则package不能被最终用户使用。 |
| namespace | Actions的唯一命名空间（我理解其为路由url） |

**<constant>**标签以及name和value属性将用于覆盖**default.properties**中定义的任一属性，就像我们设置的**struts.devMode**属性一样。设置**struts.devMode**属性允许我们在日志文件中查看更多的调试消息。

我们定义**<action>**标签对应于我们想要访问的每个URL，并且使用execute()方法定义一个访问相应的URL时将要访问的类。

Results（结果）确定在执行操作后返回到浏览器的内容，而从操作返回的字符串应该是结果的名称。 Results按上述方式配置，或作为“全局”结果配置，可用于包中的每个操作。 Results有

**name**和**type**属性可选， 默认的name值是“success”。

Struts.xml文件可以随着时间的推移而增长，因此通过包打破它是使它模块化的一种方式，但struts提供了另一种模块化struts.xml文件的方法，你可以将文件拆分为多个xml文件，并用以下方式导入它们。

<?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>

我们没有覆盖的另一个配置文件是struts-default.xml。此文件包含Struts的标准配置设置，你不必再为99.99％的项目重复这些设置。 因此，我们不会深入了解这个文件的太多细节。如果你有兴趣，可以查看在struts2-core-2.2.3.jar文件中可用的**default.properties**文件。

## 5.3 struts-config.xml文件

struts-config.xml配置文件是Web Client中View和Model组件之间的链接，但在你99.99％的项目里你不必使用这些设置。 struts-config.xml配置文件包含以下主要元素：

|  |  |
| --- | --- |
| **序号** | **拦截器和说明** |
| 1 | **struts-config**  这是配置文件的根节点。 |
| 2 | **form-beans**  这是你将ActionForm子类映射到name的位置，你可以在struts-config.xml文件的其余部分，甚至在JSP页面上，将这个name用作ActionForm的别名。 |
| 3 | **global forwards**  此部分将你在webapp上的页面映射到name，你可以使用这个name来引用实际页面。这避免了对你网页上的URL进行硬编码。 |
| 4 | **action-mappings**  这是你声明表单处理程序的地方，也被称为**操作映射（action mappings）**。 |
| 5 | **controller**  这部分是配置Struts的内部，在实际情况中很少使用。 |
| 6 | **plug-in**   这部分告诉Struts在哪里找到属性文件，它包含提示和错误消息。 |

下面是struts-config.xml文件的示例：

<?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>

有关struts-config.xml文件的更多详细内容，可查看Struts Documentation。

## 5.4 struts.properties文件

这个配置文件提供了一种机制来改变框架的默认行为。实际上，**struts.properties**配置文件中包含的所有属性也可以在**web.xml**中配置使用**init-param**，以及在**struts.xml**配置文件中使用**constant**标签。 但如果你想保持事件独立以及保留更多struts细节，那么你可以在**WEB-INF/classes**文件夹下创建这个文件。

struts.properties

文件中配置的值将覆盖

**default.properties**

中配置的默认值，这些值包含在struts2-core-x.y.z.jar分布中。有一些属性，你可以考虑改为使用

**struts.properties**

文件：

### 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

这里任何以＃（hash）开头的行都将被假定为注释，并且它会被Struts 2默认忽略。

# 6 Struts2动作

动作是Struts2框架的核心，因为它们适用于任何MVC（模型视图控制器）框架。

每个URL映射到特定的动作，其提供处理来自用户的请求所需的处理逻辑。  
但动作还有另外两个重要的功能。

首先，action在将数据从请求传递到视图（无论是JSP还是其他类型的结果）方面起着重要作用。

第二，action必须协助框架确定哪个结果应该呈现在响应请求的视图中。

# 7 Struts2 结果类型

如前面所述，**<results>**标签在Struts2 MVC框架中扮演视图的角色。Action负责执行业务逻辑，下一步就是使用**<results>**标签显示视图。  
通常有一些导航规则附加的结果。例如，如果action是进行验证用户，则有三种可能的结果：（a）成功登录（b）登录失败：用户名或密码不正确（c）帐户锁定。  
在这种情况下，action将配置三个可能的结果字符串和三个不同的视图来渲染结果，这在我们前面的例子中已经看到过了。  
但是，Struts2不绑定使用JSP作为视图技术。毕竟，MVC范例的目的是保持图层分离和高度可配置。例如，对于Web2.0客户端，你可能希望返回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实现这一点。  
Struts提供了许多预定义的结果类型，我们已经看到的是默认的结果类型**dispatcher**，它用于分发到JSP页面。

Struts允许你使用其他标记语言为视图技术呈现结果，较常选用的包括**Velocity**，**Freemaker**，**XSLT**和**Tiles**。

## 7.1 dispatcher结果类型

**dispatcher**结果类型是默认的类型，如果未指定其他结果类型，则使用此类型。

它用于转发到服务器上的servlet，JSP，HTML等页面。它使用RequestDispatcher.forward()方法。  
我们在前面的示例中看到了“简写”版本，里面我们用一个JSP路径作为结果标签的主体。

<result name="data">

/data.jsp

</result>

我们还可以使用<result ...>元素中的<param name="location">标签来指定JSP文件，如下所示：

<result name="success" type="dispatcher">

<param name="location">

/data.jsp

</param >

</result>

我们还可以使用一个**parse**参数，默认情况下为true。parse参数确定是否将为OGNL表达式解析位置参数。

## 7.2 FreeMaker结果类型

在这个例子中，我们将介绍如何使用**FreeMaker**作为视图技术。Freemaker是一个流行的模板引擎，使用预定义的模板生成输出。

让我们创建一个包含以下内容的名为**hello.fm**的Freemaker模板文件：

Hello World ${name}

上面的文件是一个模板，其中**name**是一个参数，将使用定义的action从外部传递。你可以在CLASSPATH中保留此文件。接下来，让我们参考下面修改**struts.xml**以指定结果：

<?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=" cn.htb.MyAction "

method="execute">

<result name="success" type="freemarker">

<param name="location">/hello.fm</param>

</result>

</action>

</package>

</struts>

这个例子中我们使用了Freemaker，可以看到，这与JSP视图完全相同，只是我们不绑定使用JSP作为视图技术。

## 7.3 redirect结果类型

redirect结果类型调用标准的response.sendRedirect()方法，使得浏览器向给定的位置创建一个新请求。  
我们可以在<result...>元素的主体中或作为<param name="location">的元素中给定位置。redirect也支持parse参数，以下是使用XML配置的示例：

<action name="hello" class="cn.htb.MyAction" method="redirect">

<result name="success" type="redirect">

<param name="location">

/NewWorld.jsp

</param >

</result>

</action>

所以只需修改你的struts.xml文件来定义上面提到的redirect结果类型，并创建一个新的文件NewWorld.jpg，那么当你的hello action返回“success”时就会产生redirect结果

# 8 Struts2 值栈/OGNL

## **8.1 值栈是一组对象，按照提供的顺序存储以下这些对象：**

|  |  |
| --- | --- |
| **序号** | **对象和说明** |
| 1 | **Temporary对象**  实际中存在各种在页面执行期间创建的temporary对象。例如，JSP标签循环集合的当前迭代值。 |
| 2 | **Model对象**  如果在struts应用程序中使用Model对象，则当前Model对象放在值堆栈上的action之前。 |
| 3 | **Action对象**  这是指正在执行的当前action对象。 |
| 4 | **命名对象**  这些对象包括#application，#session，#request，#attr和#parameters以及所引用的相应的servlet作用域。 |

值栈可以通过为JSP，Velocity或Freemarker提供的标签进行访问。我们将在单独的章节中学习到用于获取和设置struts2 值栈的各种标签。你可以在action中获取值栈对象，如下所示：

ActionContext.getContext().getValueStack()

一旦你有一个值栈对象，你可以使用以下方法来操纵该对象：

|  |  |
| --- | --- |
| **序号** | **值栈方法和说明** |
| 1 | **Object findValue(String expr)**  通过在默认搜索顺序中对值栈评估所给定的表达式来查找值。 |
| 2 | **CompoundRoot getRoot()**  获取将对象推入值栈的CompoundRoot。 |
| 3 | **Object peek()**  获取值栈顶部的对象而不改变值栈。 |
| 4 | **Object pop()**  获取值栈顶部的对象，并将其从值栈中删除。 |
| 5 | **void push(Object o)**  将对象放在值栈的顶部。 |
| 6 | **void set(String key,Object o)**  使用给定的key在值栈上设置一个对象，使其可通过findValue（key,...）检索。 |
| 7 | **void setDefaultType(Class defaultType)**  设置在获取值时要转换的默认类型。 |
| 8 | **void setValue(String expr,Object value)**  尝试使用由默认搜索顺序给定的表达式在值栈的bean上设置属性。 |
| 9 | **int size()**  获取值栈中的对象数。 |

## 8.2 OGNL

**OGNL**（Object-Graph Navigation Language，对象图导航语言）是一种强大的表达式语言，用于引用和操作值栈上的数据，还可用于数据传输和类型转换。  
OGNL非常类似于JSP表达式语言。OGNL基于上下文中存有根对象或默认对象的理念，使用标记符号（即#号）来引用默认或根对象的属性。  
如前面所述，OGNL是基于上下文的，而Struts构建了一个ActionContext映射以供OGNL使用。 ActionContext映射包含以下内容：

**应用程序** - 应用程序作用域变量

**会话** - 会话作用域变量

**根/值栈** - 所有的action变量都存储在这里

**请求** - 请求作用域变量

**参数** - 请求参数

**属性** - 存储在页面，请求，会话和应用程序作用域中的属性

有必要了解的是，Action对象在值栈中总是可用的，因此如果你的Action对象有x和y属性，你可以随时使用。  
ActionContext中的对象使用#号引用，但是，值栈中的对象可以直接引用，例如，如果**employee**是action类的属性，则可以按如下方式引用：

<s:property value="name"/>

替代

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

如果你在会话中有一个名为“login”的属性，你可以按如下方式检索：

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

OGNL还支持处理集合 - 即Map，List和Set。例如，要显示颜色的下拉列表，你可以执行以下操作：

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

OGNL表达式很智能地将“红色”，“黄色”，“绿色”解释为颜色，并基于此构建了列表。  
在下一章我们学习各种的标签时，OGNL表达式将会广泛的用到。因此，不要用孤立的方式去了解OGNL，结合Form标签/Control标签/Data标签和Ajax标签部分中的一些示例来了解它。

# 9 Struts2 本地化/国际化（i18n）

国际化（i18n）是规划和实施产品和服务以便更好地适应特定的当地语言和文化的过程，也就是本地化过程，而国际化过程有时也被称为翻译或本地化实现。国际化缩写为**i18n**，单词以i开头，以n结尾，是因为在第一个i和最后一个n之间有18个字符。  
Struts2使用bundle资源束、拦截器和标签库提供本地化，即国际化（i18n）支持，主要用在以下几个地方：

UI标签

信息和错误。

Action类。

## 9.1 资源束

Struts2使用资源束为Web应用程序的用户提供多种语言和区域的设置选项。你不必担心需要用不同的语言编写页面，你需要做的只是为每种你想要的语言创建一个资源束。资源束将包含用户语言中的标题，消息和其他文本，是包含应用程序默认语言的一对key/value的文件。  
资源文件最简单的命名格式是：

bundlename\_language\_country.properties

这里的**bundlename（任意）**可以是ActionClass，Interface，SuperClass，Model，Package，Global资源属性。

下一部分**language\_country**表示国家区域设置，

例如西班牙语（西班牙）区域设置由es\_ES表示，英语（美国）区域设置由en\_US表示等。这里先跳过可选国家的部分。  
当你通过key引用消息元素时，Struts框架按以下顺序搜索相应的信息束：

ActionClass.properties

Interface.properties

SuperClass.properties

model.properties

package.properties

struts.properties

global.properties

要以多种语言开发应用程序，你必须维护与这些语言/区域设置对应的多个属性文件，并根据每对key/value定义所有内容。例如，如果你要开发美国英语（默认），西班牙语和法语的应用程序，必须创建三个属性文件。这里我们将使用**global.properties**文件，你也可以使用不同的属性文件来分离不同类型的信息。

**global.properties：**默认情况下使用汉语

**global\_en.properties：**这将用于英语环境。

**global\_es.properties：**这将用于西班牙语环境。

## 9.2 访问信息

有几种方法来访问信息资源，包括getText，text标签，UI标签的key属性和i18n标签。接下来让我们简要了解一下：

获取国际化消息：

### 1 getText（’key’）

显示**i18n**文本，需在property标签或其他任何标签（如UI标签）中调用**getText**，如下所示：

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

### ****2 struts表签的key属性****

**text 或** textfield标签从默认资源束（即struts.properties）中检索信息：（大多数UI标签的**key**属性可用于从资源束检索信息：）

<s:text name="key" />

<s:textfield key="key" name="textfieldName"/>

### ****3 i18n标签****

**i18n**标签会将任意资源束推送到值栈，而i18n标签内的其他标签可以显示来自该资源束的信息：

<s:i18n name="bundlename">

<s:text name=" key" />

</s:i18n>

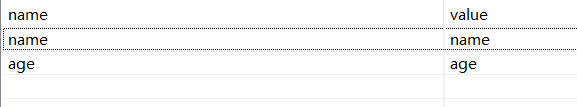
## 9.3 本地化示例

### 9.3.1 资源文件准备

#### global.properties



#### global\_en\_US.properties



### 9.3.2 action

表单跳转页面的action

**package** cn.htb;

**public** **class** I18N {

**public** String i18n(){

**return** "success";

}

}

**切换区域的action**

**package** cn.htb;

**public** **class** Locale{

**public** String execute()

{

**return** "success";

}

}

### 9.3.3 struts.xml

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

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

<!-- 开发者模式 -->

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

<!—加载国际化资源文件 -->

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

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

<action name=*"i18n"* class=*"cn.htb.I18N"* method=*"i18n"*>

<result name=*"success"*>/i18n.jsp</result>

</action>

<action name="locale" class="cn.htb.Locale" method="execute">

<result name=*"success"*>/i18n.jsp</result>

</action>

</package>

</struts>

### 9.3.4 jsp页面：

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>

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

<%

String path = request.getContextPath();

String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";

%>

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

<html>

<head>

<base href=*"*<%=basePath%>*"*>

<title>My JSP 'i18n.jsp' starting page</title>

<meta http-equiv=*"pragma"* content=*"no-cache"*>

<meta http-equiv=*"cache-control"* content=*"no-cache"*>

<meta http-equiv=*"expires"* content=*"0"*>

<meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>

<meta http-equiv=*"description"* content=*"This is my page"*>

<!--

<link rel="stylesheet" type="text/css" href="styles.css">

-->

</head>

<body>

<h1>切换语言:切换原理 见9.4</h1>

<a href="locale.action?request\_locale=zh\_CN">中文</a>

<a href="locale.action?request\_locale=en\_US">English</a>

<h2>方式一</h2>

<s:property value=*"getText('name')"* />

-----

<h2>方式二</h2>

<s:form>

<s:textfield name=*"name"* key=*"name"*></s:textfield>

<s:textfield name=*"age"* key=*"age"*></s:textfield>

</s:form>

<h2>方式三</h2>

<s:i18n name=*"gl"*>

<s:text name=*"age"* />

</s:i18n>

</body>

</html>

## 9.4 [Structs2 request\_locale国际化介绍](https://www.cnblogs.com/lechance/p/4373273.html)

众所周知，struts2中使用了大量的拦截器(即是传说中的interceptor)，其中国际化使用的拦截器为com.opensymphony.xwork2.interceptor.I18nInterceptor，那么接下来分析一下I18nInterceptor这个类。

### 9.4.1 属性

  I18nInterceptor有三个属性，分别为parameterName，requestOnlyParameterName，attributeName。

   parameterName：http request请求中的参数名字，该参数反映了应该转向的locale，并且保存在session中，所以会话过程都有效。默认为request\_locale。

   requestOnlyParameterName：http request请求中的参数名字，该参数反映了应该转向的locale，不保存在session中，所以只对当前请求有效。默认为request\_only\_locale。

   attributeName：session中保存用户选择的locale值对应的key值。默认值为WW\_TRANS\_I18N\_LOCALE

### 9.4.2 方法

  拦截器中最重要的方法当属intercept方法，其他方法暂时忽略。该方法的逻辑如下：

  1.获取parameters对象。Map<String, Object> params = invocation.getInvocationContext().getParameters()

  2.在params在取出并移除request\_locale。

Object requested\_locale = params.remove(parameterName);（parameters对象获取参数得到的是一个字符串数组）

      2.1 如果 requested\_locale不为null，是Array，长度为1，取出requested\_locale的值。否则，执行2.2

      2.2 返回requested\_locale

  3.在step2中获取到的request\_locale如果为空，则将2中的parameterName换成requestOnlyParameterName，执行step2，返回requested\_locale

  4.如果通过step2和step3获取到的requested\_locale不为null，则将requested\_locale转换成locale并保存在session中

（注意：如果获取到的requested\_locale.toString()为null或空字符串或者"\_"，此时locale=Locale.getDefault()）。

  5.如果通过step2和step3获取到的requested\_locale为null，则判断session中key=attributeName的value值是否存在，

若存在则locale=session.get(attributeName)，否则locale = invocation.getInvocationContext().getLocale()

（此处意思是没有找到重写的locale定义，则保持当前的invocation locale，即用户在浏览器中设置的语言）。

  至此，通过I18nInterceptor得到了locale，此后的处理就不言而喻了，肯定就是根据locale取值了。

可以看出：用户在画面设置的locale优先级高于浏览器设置语言得到的locale。

可以看到此处超链接处设置request\_locale=en\_US，这样struts2获取到的locale当然就是en\_US，

此时不管浏览器语言设置成什么，画面显示的都是英文的资源，并且在会话期间都有效。

若设置的是request\_only\_locale=en\_US，这样只是在当前请求下会显示英文界面，刷新页面就会显示中文了。

# 10 Struts2 主题和模板

开始本章的内容学习之前，让我们看看http://struts.apache.org给出的几个定义：

|  |  |
| --- | --- |
| **术语** | **描述** |
| **tag（标签）** | 从JSP，FreeMarker或Velocity内部执行的一小段代码。 |
| **template（模板）** | 一些代码，通常是写在FreeMarker上的，可以由某些标签（HTML标签）呈现。 |
| **theme（主题）** | 封装在一起以提供公共功能的模板集合。 |

当你在Web页面中使用Struts2 标签（如<s:submit...>，<s:textfield...>等）时，Struts2 框架会生成具有预配置样式和布局的HTML代码。Struts2 具有三个内置主题：

|  |  |
| --- | --- |
| **Theme（主题）** | **描述** |
| **simple theme** | 没有“bells and whistles”的最小主题。例如，textfield标签呈现HTML<input/>标签无标记、验证、错误报告或任何其他格式或功能。 |
| **xhtml theme** | 这是Struts 2使用的默认主题，提供了simple theme具备的所有基础，并添加了几个功能，如HTML的标准两列表布局、每个HTML的标记、验证和错误报告等。 |
| **css\_xhtml theme** | 这个主题提供了simple theme具备的所有基础，并添加了几个功能，如基于CSS的标准两列布局，对HTML Struts标签使用<div>，HTML Struts每个标签的标记，根据CSS样式表放置等。 |

如上所述，如果你不指定一个主题，那么Struts2 将默认使用xhtml theme。例如这个Struts2选择标签：

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

生成以下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action名称。

## 10.1 选择主题

你可以在每个Struts2 标签的基础上指定主题，也可以使用以下方法之一指定Struts2 应使用的主题：

特定标签上的theme属性

标签的周边表单标签的theme属性

名为“theme”的页面作用域属性

名为“theme”的请求作用域属性

名为“theme”的会话作用域属性

名为“theme”的应用程序作用域属性

struts.properties中的struts.ui.theme属性（默认为xhtml）

以下是在标签级别指定它们的语法，如果你愿意为不同的标签使用不同的主题：

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

因为在每个标签的基础上使用主题并不是很实用，所以只需使用以下标签就可以在**struts.properties**文件中指定规则：

# Standard UI theme

struts.ui.theme=xhtml

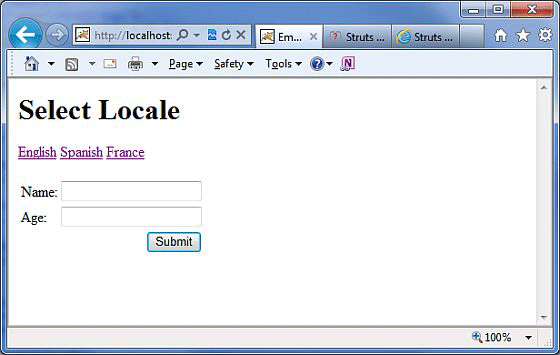
# Directory where theme template resides

struts.ui.templateDir=template

# Sets the default template type. Either ftl, vm, or jsp

struts.ui.templateSuffix=ftl

下面是我们从本地化章节中选择的结果，我们在**struts-default.properties**文件中使用了默认主题，设置为**struts.ui.theme=xhtml**，该文件默认情况下位于struts2-core.xy.z.jar文件中。



## 10.2 主题是如何工作的？

对于给定的主题，每个struts标签都有一个关联的模板，如**s:textfield** -> **text.ftl**和**s:password** -> **password.ftl**等。这些模板文件压缩在struts2-core.xy.z.jar文件中，它们为每个标签保留预定义的HTML布局。因此Struts2 框架使用Sturts标签和相关模板生成最终的HTML标记代码。

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

默认模板已经写在FreeMarker中，它们有扩展名**.ftl**。你可以使用velocity或JSP设计你的模板，使用**struts.ui.templateSuffix**和**struts.ui.templateDir**在struts.properties中设置配置。

## 10.3 创建新主题

创建新主题的最简单方法是复制任何现有的theme/template文件，并进行必要的修改。

那么，让我们在WebContent/WEB-INF/classes中创建一个名为template的文件夹，以及一个以新主题名称命名的子文件夹，例如WebContent/WEB-INF/classes/template/mytheme。

从这里，你可以从头开始构建模板，或者你可以从Struts2 现有版本中复制模板，并根据需要修改它们。  
出于学习的目的，我们将修改现有的默认模板xhtml。

那么现在先将内容从struts2-core-x.y.z.jar/template/xhtml复制到我们的主题目录，并只修改WebContent/WEB-INF/classes/template/mytheme/control.ftl文件。当我们打开control.ftl时它会显示以下几行：

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

<#if parameters.cssStyle??> style="${parameters.cssStyle?html}"<#rt/>

</#if>

>

让我们将以下内容修改到上面的**control.ftl**文件中：

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

如果你查看**form.ftl**，你会发现它正在使用这个**control.ftl**文件，但form.ftl是从xhtml主题中引用这个文件。那么让我们把它做如下修改：

<#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" />

我们假设你不太了解**FreeMarker**模板语言，不过你仍可以通过查看.ftl文件得到一个很好的想法。

不管怎样，让我们先保存以上更改，并返回到我们的本地化示例，创建具有以下内容的**WebContent/WEB-INF/classes/struts.properties**文件：

# Customized them

struts.ui.theme=mytheme

# Directory where theme template resides

struts.ui.templateDir=template

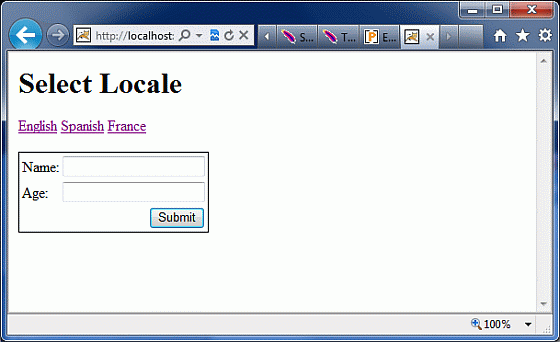
# Sets the template type to ftl.

struts.ui.templateSuffix=ftl

现在在此更改后，右键单击项目名称，然后单击**“Export”>“ WAR File”**以创建WAR文件。

然后在Tomcat的webapps目录中部署WAR文件。最后，启动Tomcat服务器并尝试访问URL <http://localhost:8080/HelloWorldStruts2>。

将显示以下界面：



你可以看到窗体组件周围的边框，这是我们从xhtml主题复制之后在out主题中所做更改的结果。

如果你稍微学习一下FreeMarker，那么就能够非常容易地创建或修改你的主题。至少现在，你一定对Sturts2 主题和模板有了基本的了解。

# 11 数据传递：

## 5.1 单文件上传

配置文件

以下是控制文件上传过程的Struts2 配置属性：

|  |  |
| --- | --- |
| **序号** | **属性和说明** |
| 1 | **struts.multipart.maxSize**  可接受的上传文件的最大值（以字节为单位），默认值为250M。 |
| 2 | **struts.multipart.parser**  用于上传多部分表单的库，默认为**jakarta**。 |
| 3 | **struts.multipart.saveDir**  存储临时文件的位置，默认是javax.servlet.context.tempdir。 |

### web.xml 配置

<**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**>

### struts.xml配置

1. 综合文件

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

<!**DOCTYPE** struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<**struts**>

<**constant** **name=***"struts.devMode"* **value=***"true"* /> **<!-- 开发者模式 -->**

<**include** **file=***"cn/htb/upload/action/upload-struts.xml"*></**include**> **<!-- 注如分出的struts.xml文件 -->**

</**struts**>

1. 散列文件

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

<!**DOCTYPE** struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<**struts**>

**<!-- 国际化 修改底层文件配置的位置 ：struts.custom.i18n.resources=testmessages,testmessages2-->**

<**constant** **name=***"struts.custom.i18n.resources"* **value=***"uploadmessage"*></**constant**> **<!-- 国际化文件注入 -->**

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

**<!-- 方式一：将拦截器放置在package包下，则对整个包内的action都起作用**

**文件上传拦截器**

**<interceptors>**

**引入拦截器默认栈**

**<interceptor-stack name="defaultStack">**

**在默认栈中注入拦截类型，注入的拦截器是文件上传拦截器**

**<interceptor-ref name="fileUpload">**

**文件后缀名**

**<param name="allowedExtensions">.doc,.jpg</param>**

**文件类型**

**<param name="allowedTypes"></param>**

**文件大小**

**<param name="maximumSize"></param>**

**</interceptor-ref>**

**再将注入完成的默认值重新注入**

**<interceptor-ref name="defaultStack"></interceptor-ref>**

**</interceptor-stack>**

**</interceptors>**

**-->**

<**action** **name=***"UploadAction\_\*"* **method=***"{1}"* **class=***"cn.htb.upload.action.UploadAction"* >

**<!-- 方式二：将拦截器放置在action下只对该action有效 -->**

**<!-- 直接在注入拦截器 -->**

<**interceptor-ref** **name=***"defaultStack"*>

**<!-- 因为没有写出注入fileUpload,所以在参数name中给出 -->**

<**param** **name=***"fileUpload.allowedExtensions"*>.doc</**param**> **<!-- 文件后缀限制 -->**

<**param** **name=***"fileUpload.maximumSize"*>2097152</**param**> **<!-- 文件大小限制 -->**

**<!-- <param name="fileUpload.AllowedTypes">JEPG/JPG</param> 文件类型限制-->**

</**interceptor-ref**>

<**result** **name=***"success"*>/index.jsp</**result**>

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

</**action**>

**<!-- 多文件上传 -->**

<**action** **name=***"UploadsAction\_\*"* **class=***"cn.htb.upload.action.UploadsAction"* **method=***"{1}"*>

<**result** **name=***"success"*>/index.jsp</**result**>

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

</**action**>

</**package**>

</**struts**>

### 3 Action编写

**/\*\***

**\***

**\* uploadFile.java是一个非常简单的类。要注意的重点是，FileUpload拦截器和Parameters拦截器为我们承担了所有的重工作量。**

**\* 默认情况下，FileUpload拦截器为你提供三个参数，它们分别按以下方式命名：**

**[文件名参数]** - **这是用户已上传的实际文件。在这个例子中它将是“image”**

**[文件名参数]ContentType** - **这是上传的文件的内容类型。在这个例子中，它将是“imageContentType” xxx**-**ContentType**

**[文件名参数]FileName** - **这是上传的文件的名称。在这个例子中，它将是“imageFileName” xxx**-**FileName**

**得益于Struts拦截器这三个参数均可供我们使用。我们要做的是在Action类中创建三个带有正确名称的参数，并使这些变量可以自动连接。**

**\***

**\*/**

@SuppressWarnings("serial")

**public** **class** UploadAction **extends** ActionSupport{

**private** File image;

**private** String imageFileName;

**private** String imageContentType;

**public** File getImage() {

**return** image;

}

**public** **void** setImage(File image) {

**this**.image = image;

}

**public** String getImageFileName() {

**return** imageFileName;

}

**public** **void** setImageFileName(String imageFileName) {

**this**.imageFileName = imageFileName;

}

**public** String getImageContentType() {

**return** imageContentType;

}

**public** **void** setImageContentType(String imageContentType) {

**this**.imageContentType = imageContentType;

}

**public** String uploadFile(){

System.***out***.println("============单文件上传测试=============");

System.***out***.println("image:"+**this**.image);

System.***out***.println("imageFileName:"+**this**.imageFileName);

System.***out***.println("imageContentType:"+**this**.imageContentType);

String path = ServletActionContext.*getServletContext*().getRealPath("/upload");

File saveFile = **new** File(path+"\\"+imageFileName);

System.***out***.println(path+"\\"+imageFileName);

**//写文件到硬盘方法一**

**// try {**

**// FileUtils.copyFile(image, saveFile);**

**// } catch (IOException e) {**

**// e.printStackTrace();**

**// }**

**//写文件到硬盘方法二**

System.***out***.println(image.getName());

image.renameTo(saveFile);

**return** "success";

}

}

### 4 jsp编写

(1)上传文件页

<form action=*"UploadAction\_uploadFile.action"* method=*"post"* enctype=*"multipart/form-data"*>

<input type=*"file"* name=*"image"*/>

<input type=*"submit"* value=*"上传"*/>

</form>

(2)错误信息显示页(两个显示方法总有一个可用的)

<s:fielderror />

<s:actionerror/>

### 5 国际化文件配置

（1）文件名：uploadmessage.properties

（2）内容：

struts.messages.error.uploading=\u6587\u4EF6\u9519\u8BEF\: **{0}**

struts.messages.error.file.too.large=\u6587\u4EF6\u592A\u5927\: **{0}** "**{1}**" **{2}**

struts.messages.error.content.type.not.allowed=\u6587\u4EF6\u7C7B\u578B\u4E0D\u652F\u6301\: **{0}** "**{1}**" "**{2}**" **{3}**

struts.messages.error.file.extension.not.allowed=\u540E\u7F00\u540D\u4E0D\u5141\u8BB8\: **{0}** "**{1}**" "**{2}**" **{3}**

即：

struts.messages.error.uploading=文件错误: {0}

struts.messages.error.file.too.large=文件太大: {0} "{1}" {2}

struts.messages.error.content.type.not.allowed=文件类型不支持: {0} "{1}" "{2}" {3}

struts.messages.error.file.extension.not.allowed=后缀名不允许: {0} "{1}" "{2}" {3}

**////////////////////////////////////////华 丽 分 割 线/////////////////////////////////////////**

## 5.2 struts 多文件上传

### 1 web.xml 配置

<**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**>

### 2 struts.xml配置

1. 综合文件

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

<!**DOCTYPE** struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<**struts**>

<**constant** **name=***"struts.devMode"* **value=***"true"* /> **<!-- 开发者模式 -->**

<**include** **file=***"cn/htb/upload/action/upload-struts.xml"*></**include**> **<!-- 注如分出的struts.xml文件 -->**

</**struts**>

1. 散列文件

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

<!**DOCTYPE** struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<**struts**>

**<!-- 国际化 修改底层文件配置的位置 ：struts.custom.i18n.resources=testmessages,testmessages2-->**

<**constant** **name=***"struts.custom.i18n.resources"* **value=***"uploadmessage"*></**constant**> **<!-- 国际化文件注入 -->**

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

**<!-- 方式一：将拦截器放置在package包下，则对整个包内的action都起作用**

**文件上传拦截器**

**<interceptors>**

**引入拦截器默认栈**

**<interceptor-stack name="defaultStack">**

**在默认栈中注入拦截类型，注入的拦截器是文件上传拦截器**

**<interceptor-ref name="fileUpload">**

**文件后缀名**

**<param name="allowedExtensions">.doc,.jpg</param>**

**文件类型**

**<param name="allowedTypes"></param>**

**文件大小**

**<param name="maximumSize"></param>**

**</interceptor-ref>**

**再将注入完成的默认值重新注入**

**<interceptor-ref name="defaultStack"></interceptor-ref>**

**</interceptor-stack>**

**</interceptors>**

**-->**

<**action** **name=***"UploadAction\_\*"* **method=***"{1}"* **class=***"cn.htb.upload.action.UploadAction"* >

**<!-- 方式二：将拦截器放置在action下只对该action有效 -->**

**<!-- 直接在注入拦截器 -->**

<**interceptor-ref** **name=***"defaultStack"*>

**<!-- 因为没有写出注入fileUpload,所以在参数name中给出 -->**

<**param** **name=***"fileUpload.allowedExtensions"*>.doc</**param**> **<!-- 文件后缀限制 -->**

<**param** **name=***"fileUpload.maximumSize"*>2097152</**param**> **<!-- 文件大小限制 -->**

**<!-- <param name="fileUpload.AllowedTypes">JEPG/JPG</param> 文件类型限制-->**

</**interceptor-ref**>

<**result** **name=***"success"*>/index.jsp</**result**>

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

</**action**>

**<!-- 多文件上传 -->**

<**action** **name=***"UploadsAction\_\*"* **class=***"cn.htb.upload.action.UploadsAction"* **method=***"{1}"*>

<**result** **name=***"success"*>/index.jsp</**result**>

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

</**action**>

</**package**>

</**struts**>

### 3 Action编写

@SuppressWarnings("serial")

**public** **class** UploadsAction **extends** ActionSupport {

**private** File[] image;

**private** String[] imageFileName;

**private** String[] imageContentType;

**public** File[] getImage() {

**return** image;

}

**public** **void** setImage(File[] image) {

**this**.image = image;

}

**public** String[] getImageFileName() {

**return** imageFileName;

}

**public** **void** setImageFileName(String[] imageFileName) {

**this**.imageFileName = imageFileName;

}

**public** String[] getImageContentType() {

**return** imageContentType;

}

**public** **void** setImageContentType(String[] imageContentType) {

**this**.imageContentType = imageContentType;

}

**public** String uploadFile(){

System.***out***.println("多文件上传方法测试");

String realPath = ServletActionContext.*getServletContext*().getRealPath("/upload");

**for**(**int** i=0;i<image.length;i++){

File saveFile = **new** File(realPath+"/"+imageFileName[i]);

image[i].renameTo(saveFile);

}

**return** "success";

}

}

### 4 jsp编写

<div style="width:*600* ; height:*20*; background:*#abcdef*" align=*"center"* >多文件上传测试</div><br />

<form action=*"UploadsAction\_uploadFile.action"* method=*"post"* enctype=*"multipart/form-data"*>

<input type=*"file"* name=*"image"*/><br />

<input type=*"file"* name=*"image"*/><br />

<input type=*"file"* name=*"image"*/><br />

<input type=*"file"* name=*"image"*/><br />

<input type=*"submit"* value=*"上传"*/><br />

</form>

### 5 国际化文件配置

（1）文件名：uploadmessage.properties

（2）内容：

struts.messages.error.uploading=\u6587\u4EF6\u9519\u8BEF\: **{0}**

struts.messages.error.file.too.large=\u6587\u4EF6\u592A\u5927\: **{0}** "**{1}**" **{2}**

struts.messages.error.content.type.not.allowed=\u6587\u4EF6\u7C7B\u578B\u4E0D\u652F\u6301\: **{0}** "**{1}**" "**{2}**" **{3}**

struts.messages.error.file.extension.not.allowed=\u540E\u7F00\u540D\u4E0D\u5141\u8BB8\: **{0}** "**{1}**" "**{2}**" **{3}**

即：

struts.messages.error.uploading=文件错误: {0}

struts.messages.error.file.too.large=文件太大: {0} "{1}" {2}

struts.messages.error.content.type.not.allowed=文件类型不支持: {0} "{1}" "{2}" {3}

struts.messages.error.file.extension.not.allowed=后缀名不允许: {0} "{1}" "{2}" {3}

## 5.3 Struts2 发送电子邮件

本章内容将教你如何使用Struts2 应用程序发送电子邮件。学习前，你需要从[JavaMail API 1.4.4](http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-eeplat-419426.html" \l "javamail-1.4.4-oth-JPR" \t "_blank)下载并安装mail.jar，并将mail.jar文件放在WEB-INF\lib文件夹中，然后继续按照以下标准步骤创建action，视图和配置文件。

### 5.3.1 创建Action

首先是创建一个Action方法来处理电子邮件发送。让我们创建一个包含以下内容的名为**Emailer.java**的新类：

package cn.w3cschool.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;

}

}

如上面的源代码所示，**Emailer.java**具有与下面给出的email.jsp页面中的表单属性相对应的属，这些属性分别是：

* **from** - 发件人的电子邮件地址。由于我们使用Google的SMTP，因此我们需要有效的gtalk ID。
* **password** - 上述帐户的密码
* **to** - 发送电子邮件给谁？
* **Subject** - 电子邮件的主题
* **body** - 实际的电子邮件内容

我们没有考虑对上述字段的任何验证，验证将在下一章添加。让我们看看execute()方法， execute()方法使用javax Mail库提供的参数发送电子邮件。如果邮件成功发送，action返回SUCCESS，否则返回ERROR。

### 5.3.2 创建主页

现行编写主页的JSP文件**index.jsp**，这将用于收集上面提到的电子邮件相关信息：

<%@ 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>

### 5.3.3 创建视图

现在创建**success.jsp**文件，这在action返回SUCCESS结果的情况下会被调用，但如果从action返回ERROR结果，我们将用另一个视图文件。

<%@ 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>Email Success</title>

</head>

<body>

Your email to <s:property value="to"/> was sent successfully.

</body>

</html>

以下将是在action返回ERROR的情况下调用的视图文件**error.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>Email Error</title>

</head>

<body>

There is a problem sending your email to <s:property value="to"/>.

</body>

</html>

### 5.3.4 配置文件

最后，让我们使用struts.xml配置文件将所有内容放在一起，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="cn.w3cschool.struts2.Emailer"

method="execute">

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

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

</action>

</package>

</struts>

# 12 四大器：

创建行动：唯一要求，方法必须无参数 返回值是字符串或结果对象，并且必须是POJO

Struts2的中行动的唯一要求是必须有一个无参数方法返回字符串或结果对象，并且必须是POJO。

如果没有指定不带参数的方法，则默认是使用的execute（）方法。

## 6.1 拦截器：两大类

### 问题：

1 如果单独使用全局拦截器，而不在action中引入任何拦截器时，全局拦截器正常生效。

2 如果在使用全局拦截器时，又在action中注入了单独的拦截器，将导致全局拦截器失效。是因为加载了指定的拦截器后，不去加载默认拦截器了。

解决：在action加入了单独的拦截器后，再注入默认拦截器栈。

拦截器在概念上与servlet过滤器或JDK代理类相同。拦截器允许横切功能，把action以及框架分开实现。你可以使用拦截器实现以下操作：

在调用action之前提供预处理逻辑。

在调用action后提供后处理逻辑。

捕获异常，以便可以执行备用处理。

Struts2框架中提供的许多功能都是使用拦截器实现的，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Struts2将其大部分功能基于拦截器，因此不太可能为每个action分配7个或8个拦截器。

### 6.1.1 Struts2框架内置拦截器

Struts 2框架提供了一个良好的开箱即用的拦截器列表，这些拦截器预先配置好并可以使用。 下面列出了几个重要的拦截器：

|  |  |
| --- | --- |
| **序号** | **拦截器和说明** |
| 1 | **alias**  允许参数在请求之间使用不同的别名。 |
| 2 | **checkbox**  通过为未检查的复选框添加参数值false，以辅助管理复选框。 |
| 3 | **conversionError**  将字符串转换为参数类型的错误信息放置到action的错误字段中。 |
| 4 | **createSession**  自动创建HTTP会话（如果尚不存在）。 |
| 5 | **debugging**  为开发人员提供一些不同的调试屏幕。 |
| 6 | **execAndWait**  当action在后台执行时，将用户发送到中间的等待页面。 |
| 7 | **exception**  映射从action到结果抛出的异常，允许通过重定向自动处理异常。 |
| 8 | **fileUpload**  便于文件上传。 |
| 9 | **i18n**  在用户会话期间跟踪选定的区域。 |
| 10 | **logger**  通过输出正在执行的action的名称提供简单的日志记录。 |
| 11 | **params**  设置action上的请求参数。 |
| 12 | **prepare**  这通常用于执行预处理工作，例如设置数据库连接。 |
| 13 | **profile**  允许记录action的简单分析信息。 |
| 14 | **scope**  在会话或应用程序范围内存储和检索action的状态。 |
| 15 | **ServletConfig**  提供可访问各种基于servlet信息的action。 |
| 16 | **timer**  以action执行时间的形式提供简单的分析信息。 |
| 17 | **token**  检查action的有效性，以防止重复提交表单。 |
| 18 | **validation**  提供action的验证支持。 |

你可以阅读Struts 2文档，了解上述拦截器的完整信息。接下来我们会告诉你如何在Struts应用程序中使用拦截器。

### 6.1.2 如何使用内置拦截器？

我们将首先使用**timer**拦截器，目的是测量执行action方法所需的时间。

（同时我们使用**params**拦截器，目的是将请求参数发送给action。你可以尝试不在你的例子中使用这个拦截器，然后你会发现**name**属性没有被设置，因为参数无法发送给action。）

6.1.1 Struts自带拦截器使用

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

<!-- action映射设置 -->

<action name=*"hello"* class=*"cn.htb.MyAction"* method=*"getData"*>

<interceptor-ref name=*"timer" /*> <-- 此处直接引入拦截器，仅在这个action中生效 -->

<result name=*"data"*>/data.jsp</result>

</action>

</package>

目的是计算调用getData方法所用时间：结果如下

10:34:03.791 [http-nio-8080-exec-2] INFO com.opensymphony.xwork2.interceptor.TimerInterceptor - Executed action [//hello!getData] took 477 ms.

这一行是由**timer**拦截器生成的，是表示ation总共需要109ms来执行。

如果打印不出来则另外需要添加日志配置：

#### a 依赖问题：

依赖包的版本问题，会导致一些异常和数据无法数据问题。

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>2.10.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.10.0</version>

</dependency>

</dependencies>

#### b 配置文件：log4j2.xml

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

<Configuration>

<Appenders>

<Console name=*"Console"* target=*"SYSTEM\_OUT"*>

<PatternLayout pattern=*"%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"*/>

</Console>

</Appenders>

<Loggers>

<Root level=*"info"*>

<AppenderRef ref=*"Console"*/>

</Root>

</Loggers>

</Configuration>

### 6.1.3 创建自定义拦截器

在你的应用程序中使用自定义拦截器是提供跨切割应用程序功能的简洁方式。创建自定义拦截器很容易，需要扩展的是以下**Interceptor**接口：

public interface Interceptor extends Serializable{

void destroy();

void init();

String intercept(ActionInvocation invocation) throws Exception;

}

正如name所指出的，init()方法提供了一种初始化拦截器的方法，而destroy()方法为拦截器清理提供了一个工具。与action不同，拦截器在请求之间重复使用，需要线程安全，特别是intercept()方法。  
ActionInvocation对象可访问运行时的环境。它允许访问action本身以及方法来调用action，并确定action是否已被调用。  
如果你不需要初始化或清理代码，可以扩展AbstractInterceptor类，以实现init()和destroy()的默认的无操作指令。

##### 6.1.3.1 创建拦截器类

**public** **class** MyInterceptor **extends** AbstractInterceptor{

@Override

**public** String intercept(ActionInvocation invocation) **throws** Exception {

System.***out***.println("前置处理。。。");

String result = invocation.invoke();

System.***out***.println(result);

System.***out***.println("后置处理。。。");

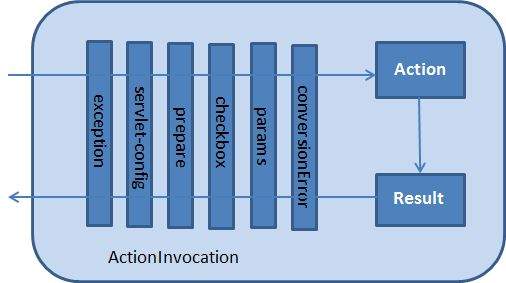
**return** result;

}

}

你可以发现，实际中action将通过拦截器使用invocation.invoke()调用执行，所以你可以根据你的需求做一些预处理和一些后处理。

框架本身通过第一次调用ActionInvocation对象的invoke()来启动进程。每次调用invoke()时，ActionInvocation都会查询其状态，并执行下一个拦截器。当所有配置的拦截器都被调用时，invoke()将使得action本身被执行。以下图表通过请求流显示了所说的概念：



##### 6.1.3.2 创建Action类

**package** cn.htb;

**public** **class** MyAction {

**public** String getData(){

System.***out***.println("测试数据");

**return** "data";

}

}

这是我们在前面的例子中看到的同一个类。

##### 6.1.3.3 配置文件

目的：注册拦截器

要注册一个新的拦截器，把<interceptors> ... </ interceptors>标签直接放置在<package>标签下的struts.xml文件中即可。

###### a 配置全局拦截器

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

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

<!-- 开发者模式 -->

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

<!-- 映射设置总包 -->

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

<!-- 自定义拦截器配置 -->

<interceptors>

<!-- 自定义拦截器 -->

<interceptor name=*"myInterceptor"* class=*"cn.htb.MyInterceptor"*/>

<!-- 默认拦截器栈 -->

<interceptor-stack name=*"defaultStack"*>

<!-- 按照注入顺序执行，所以最后要注入默认栈 -->

<!-- 注入自定义拦截器 -->

<interceptor-ref name=*"myInterceptor"*></interceptor-ref>

<!-- 注入内置拦截器 -->

<interceptor-ref name=*"timer"*></interceptor-ref>

<!-- 最终注入默认拦截器栈 -->

<interceptor-ref name=*"defaultStack"*></interceptor-ref>

</interceptor-stack>

</interceptors>

<!-- action映射设置 -->

<action name=*"hello"* class=*"cn.htb.MyAction"* method=*"getData"*>

<result name=*"data"*>/data.jsp</result>

</action>

</package>

</struts>

反例：此处我在action内部，引入了timer拦截器，导致了自定义拦截器不生效。

是因为没有将默认栈注入到action中

问题：

如果单独使用全局拦截器，而不在action中引入任何拦截器时，全局拦截器正常。

如果在使用全局拦截器时，由在action中注入了单独的拦截器，将导致全局拦截器失效。是因为加载了指定的拦截器后，不去加载默认拦截器了。

解决：在action加入了单独的拦截器后，再注入默认拦截器栈。

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

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

<!-- 开发者模式 -->

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

<!-- 映射设置总包 -->

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

<!-- 自定义拦截器配置 -->

<interceptors>

<!-- 自定义拦截器 -->

<interceptor name=*"myInterceptor"* class=*"cn.htb.MyInterceptor"*/>

<!-- 默认拦截器栈 -->

<interceptor-stack name=*"defaultStack"*>

<!-- 按照注入顺序执行，所以最后要注入默认栈 -->

<!-- 注入自定义拦截器 -->

<interceptor-ref name=*"myInterceptor"*></interceptor-ref>

<!-- 最终注入默认拦截器栈 -->

<interceptor-ref name=*"defaultStack"*></interceptor-ref>

</interceptor-stack>

</interceptors>

<!-- action映射设置 -->

<action name=*"hello"* class=*"cn.htb.MyAction"* method=*"getData"*>

<interceptor-ref name=*"timer"*></interceptor-ref>

<result name=*"data"*>/data.jsp</result>

</action>

</package>

</struts>

解决方案：在action中再注入默认拦截器栈

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

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

<!-- 开发者模式 -->

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

<!-- 映射设置总包 -->

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

<!-- 自定义拦截器配置 -->

<interceptors>

<!-- 自定义拦截器 -->

<interceptor name=*"myInterceptor"* class=*"cn.htb.MyInterceptor"*/>

<!-- 默认拦截器栈 -->

<interceptor-stack name=*"defaultStack"*>

<!-- 按照注入顺序执行，所以最后要注入默认栈 -->

<!-- 注入自定义拦截器 -->

<interceptor-ref name=*"myInterceptor"*></interceptor-ref>

<!-- 最终注入默认拦截器栈 -->

<interceptor-ref name=*"defaultStack"*></interceptor-ref>

</interceptor-stack>

</interceptors>

<!-- action映射设置 -->

<action name=*"hello"* class=*"cn.htb.MyAction"* method=*"getData"*>

<!-- 引入内置拦截器 -->

<interceptor-ref name=*"timer"*></interceptor-ref>

<!-- 注入默认栈中的拦截器 -->

<interceptor-ref name=*"defaultStack"*></interceptor-ref>

<result name=*"data"*>/data.jsp</result>

</action>

</package>

</struts>

###### b 配置局部拦截器

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

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

<!-- 开发者模式 -->

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

<!-- 映射设置总包 -->

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

<!-- 自定义拦截器配置 -->

<interceptors>

<interceptor name=*"myInterceptor"* class=*"cn.htb.MyInterceptor"* />

</interceptors>

<!-- action映射设置 -->

<action name=*"hello"* class=*"cn.htb.MyAction"* method=*"getData"*>

<interceptor-ref name=*"timer"*></interceptor-ref>

<!-- 自定义拦截器引入 -->

<interceptor-ref name=*"myInterceptor"*></interceptor-ref>

<result name=*"data"*>/data.jsp</result>

</action>

</package>

</struts>

需要注意的是，你可以在<package>标签内注册多个拦截器，同时可以在<action>标签内调用多个拦截器，也可以用不同的action调用同一个拦截器。

##### 6.1.3.4 拦截器堆栈

你可以想象，为每个action配置的多个拦截器将很快变得极其难以管理。

为此，拦截器使用拦截器堆栈进行管理。

这里是直接从struts-default.xml文件展示的一个例子：

<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>

上面的堆栈称为basicStack，可以如下所述在你的配置中使用，

此配置节点放置在<package ... />节点下。

<interceptor-ref ... />标签引用的是在当前拦截器堆栈之前配置的拦截器或拦截器堆栈。

因此非常重要的是在配置初始拦截器和拦截器堆栈时，确保name在所有拦截器和拦截器堆栈配置中是唯一的。

我们已经学习了如何将拦截器应用到action中，而拦截器堆栈的应用也是类似的。事实上，使用的标签也是一样的：

<action name="hello" class="com.tutorialspoint.struts2.MyAction">

<interceptor-ref name="basicStack"/>

<result>view.jsp</result>

</action

上述的“basicStack”注册将完整注册hello action的所使用的六个拦截器。

要注意的是，拦截器按照它们被配置的顺序执行。

例如，在上面的例子中，exception将首先执行，第二个将是servlet-config等。

## 6.2 校验器：两大方法

* 手动校验(代码校验) ：在服务器端通过编写java代码，完成数据校验
* 自动校验(配置校验) ：XML配置校验（主流） 和 注解配置校验

### 内置校验器：

Struts2的内置校验器规则在路径:xwork-core-x.x.x.jar中com.opensymphony.xwork2.validator.validators包的default.xml文件：

<validators>

<!--validator:声明验证规则

name:验证规则的名称

class:验证规则对应的类的完整路径

-->

<!--required:必填校验器,要求被校验的属性值不能为null-->

<validator name="required" class="com.opensymphony.xwork2.validator.validators.RequiredFieldValidator"/>

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

参数trim:该参数为可选参数,用于指定是否在校验之前对字符串进行整理。

-->

<validator name="requiredstring" class="com.opensymphony.xwork2.validator.validators.RequiredStringValidator"/>

<!--int:整数校验器,要求field的整数值必须在指定范围内

参数min:指定字段值的最小值,该参数为可选

参数max:指定字段值的最大值,该参数为可选

-->

<validator name="int" class="com.opensymphony.xwork2.validator.validators.IntRangeFieldValidator"/>

<!--long:整数校验器,要求field的整数值必须在指定范围内

参数min:指定字段值的最小值,该参数为可选

参数max:指定字段值的最大值,该参数为可选

-->

<validator name="long" class="com.opensymphony.xwork2.validator.validators.LongRangeFieldValidator"/>

<!--short:整数校验器,要求field的整数值必须在指定范围内

参数min:指定字段值的最小值,该参数为可选

参数max:指定字段值的最大值,该参数为可选

-->

<validator name="short" class="com.opensymphony.xwork2.validator.validators.ShortRangeFieldValidator"/>

<!--double:双精度浮点数校验器,要求field的双精度浮点数必须在指定范围内

参数minInclusive:指定字段数值所接受的最小值,该参数为可选

参数maxInclusive:指定字段数值所接受的最大值,该参数为可选

参数minExclusive:指定字段数值被排除的最小值,该参数为可选

参数maxExclusive:指定字段数值被排除的最大值,该参数为可选

-->

<validator name="double" class="com.opensymphony.xwork2.validator.validators.DoubleRangeFieldValidator"/>

<!--date:日期校验器,要求field的日期值必须在指定范围内

参数min:指定字段日期值的最小值,该参数为可选

参数max:指定字段日期值的最大值,该参数为可选

-->

<validator name="date" class="com.opensymphony.xwork2.validator.validators.DateRangeFieldValidator"/>

<!--expression:表达式校验器,它是一个非字段校验器,当参数expression计算的值为true时,校验通过,否则返回提示.

参数expression:该参数为一个逻辑表达式,该参数使用OGNL表达式,并基于值栈计算,返回一个Boolean类型值。

-->

<validator name="expression" class="com.opensymphony.xwork2.validator.validators.ExpressionValidator"/>

<!--fieldexpression:字段表达式校验器,当参数expression计算的值为true时,校验通过,否则返回提示.

参数expression:该参数为一个逻辑表达式,该参数使用OGNL表达式,并基于值栈计算,返回一个Boolean类型值.

-->

<validator name="fieldexpression" class="com.opensymphony.xwork2.validator.validators.FieldExpressionValidator"/>

<!--email:邮件地址校验器,要求如果被校验的属性值非空,则必须是合法的邮件地址-->

<validator name="email" class="com.opensymphony.xwork2.validator.validators.EmailValidator"/>

<!--url:网址校验器,要求如果被校验的属性值非空,则必须是合法的url地址-->

<validator name="url" class="com.opensymphony.xwork2.validator.validators.URLValidator"/>

<!--visitor复合类型校验器,该校验器用来校验Action中定义的复合类型属性,支持简单的复合类型或数组类型或集合类型.

参数context:校验器引用的上下文,可以随意取,可选参数

参数appendPrefix:是否为字段校验信息加前缀(message中设置的值),可选参数

-->

<validator name="visitor" class="com.opensymphony.xwork2.validator.validators.VisitorFieldValidator"/>

<!--conversion:类型转换校验器,该校验器检查是否存在转换异常

参数repopulateField:指定当类型转换失败后返回input页面时,类型转换失败的表单是否保留原来的错误输入,true为保留,false为不保留.

-->

<validator name="conversion" class="com.opensymphony.xwork2.validator.validators.ConversionErrorFieldValidator"/>

<!--stringlength:字符串长度校验器,用于检验字段中字符串长度是否在指定的范围

参数trim:该参数为可选参数,用于指定是否在校验之前对字符串进行整理。

参数maxLength:用于指定最大字符串长度,该参数为可选

参数minLength:用于指定最小字符串长度,该参数为可选

-->

<validator name="stringlength" class="com.opensymphony.xwork2.validator.validators.StringLengthFieldValidator"/>

<!--regex:正则表达式校验器,检查被校验的属性值是否匹配一个正则表达式.

参数expression:指定正则表达式

参数caseSensitive:指定进行正则表达式匹配时,是否区分大小写,默认值为true,为可选参数.

-->

<validator name="regex" class="com.opensymphony.xwork2.validator.validators.RegexFieldValidator"/>

<validator name="conditionalvisitor" class="com.opensymphony.xwork2.validator.validators.ConditionalVisitorFieldValidator"/>

</validators>

### 6.2.1手动校验（代码校验）

关键点：让校验的action继承ActionSupport，并重写validate方法。

#### 6.2.1.1 Struts2的手动校验步骤：

  首先要从页面中获取对应的标签name属性的值，在动作类action中声明同名的属性，提供get和set方法;

  要继承ActionSupport类或者实现Validateable接口；

重写Validateable接口的validate()方法，需要通过addFieldError(fieldName, errorMessage)方法来添加校验提示，fieldName为错误提示字段，errorMessage为错误提示信息，在jsp页面需要通过Struts2的<s:fielderror/>标签来提示添加的信息，该标签有个name属性，值为addFieldError方法中的fieldName，当添加了校验信息，Struts2框架会返回到name值为"input"的<result/>设置的视图中；

#### 6.2.1.2 创建主页

我们接下来编写用于收集employee的相关信息的JSP文件**index.jsp**。

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>

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

<%

String path = request.getContextPath();

String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";

%>

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

<html>

<head>

<base href=*"*<%=basePath%>*"*>

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

<meta http-equiv=*"pragma"* content=*"no-cache"*>

<meta http-equiv=*"cache-control"* content=*"no-cache"*>

<meta http-equiv=*"expires"* content=*"0"*>

<meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>

<meta http-equiv=*"description"* content=*"This is my page"*>

<!--

<link rel="stylesheet" type="text/css" href="styles.css">

-->

</head>

<body>

<form action=*"registry.action"* method=*"post"* enctype=*"application/x-www-form-urlencoded"*>

<input name=*"name"* type=*"text"* />

<input name=*"age"* type=*"text"* />

<input type=*"submit"* value=*"提交"* />

</form>

<s:fielderror/>

</body>

</html>

#### 6.2.1.3 创建视图

我们将使用在action返回SUCCESS的情况下调用的JSP文件**success.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>Success</title>

</head>

<body>

Employee Information is captured successfully.

</body>

</html>

#### 6.2.1.4 创建Action

那么，让我们定义一个小的action类：Validation，然后添加一个名为validate()的方法，如下Validation.java文件所示。

确保action类扩展了ActionSupport类，否则将不会执行validate方法。

**package** cn.htb;

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

**public** **class** Validation **extends** ActionSupport{

**private** Integer age;

**private** String name;

**public** Integer getAge() {

**return** age;

}

**public** **void** setAge(Integer age) {

**this**.age = age;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String registry(){

System.***out***.println(name);

System.***out***.println(age);

**return** "success";

}

**public** **void** validate() {

**if** (name == **null**|| name.trim().equals("")){

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

}

**if** (age == **null** ||age < 28 || age > 65){

addFieldError("age","Age must be in between 28 and 65");

}

}

}

如上例所示，验证方法首先检查“Name”字段是否具有值。如果没有任何值，则会为“Name”字段添加一个字段“错误”，并显示自定义的错误消息。其次，检查“年龄”字段的输入值是否在28和65之间，如果不符合这个条件，我们在验证字段之上添加一个错误。

#### 6.2.1.5 配置文件

最后，让我们使用**struts.xml**配置文件将所有内容放在一起，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="registry" class="cn.htb.Validation" method="registry">

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

<result name="input">/login.jsp</result>

</action>

</package>

</struts>

#### 这个验证是如何进行的？

当用户按下提交按钮时，Struts2 将自动执行validate方法，如果方法中列出的任何if语句为真，Struts2 将调用addFieldError方法。

如果添加了任何错误信息，Struts2 将不会调用execute方法。然后，Struts2 框架将返回**input**作为调用操作的结果。  
因此，当验证失败并且Struts2 返回**input**时，Struts2 框架将重新显示index.jsp文件。由于我们使用Struts2 的form标签，Struts2 会自动在form字段上方添加错误信息。  
这些错误信息是我们在addFieldError方法调用中指定的信息。addFieldError方法接受两个参数，第一个是出错时应用的**form**字段名称，第二个是在form字段上方显示的错误信息。

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

要处理**input**的返回值，我们需要将以下结果添加到**struts.xml**中的action节点。

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

### 6.2.2 XML验证

#### XML配置文件方式数据校验步骤：

首先要从页面中获取对应的标签name属性的值，在动作类action中声明同名的属性，提供get和set方法；

创建一个XML格式验证文件，文件的命名方式：ActionClassName-validation.xml，ActionClassName指的是Action类的名称，该文件需要和Action类放在同一目录下；

Struts2 基于XML的验证提供了更多的验证方式，

如email验证、integer range验证、form验证、expression验证、regex验证、required验证、requiredstring验证、stringlength验证等。  
xml文件需要命名为**'[action-class]'-validation.xml**。因此，在我们的示例中，创建了一个名为Validation**-validation.xml**的文件，其含以下内容：

<!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>

上面的XML文件将保存在CLASSPATH中，理想情况下是与类文件一起保存。以下是没有**validate()**方法的Employee action类：

**package** cn.htb;

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

**public** **class** Validation **extends** ActionSupport{

**private** Integer age;

**private** String name;

**public** Integer getAge() {

**return** age;

}

**public** **void** setAge(Integer age) {

**this**.age = age;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String registry(){

System.***out***.println(name);

System.***out***.println(age);

**return** "success";

}

}

其余的设置将保持，如同前面的例子一样。现在如果你运行应用程序，它将产生与我们前面例子相同的结果。

1 使用xml文件来存储配置的优点是允许验证与应用程序代码分离。

你可以让开发人员编写代码，让测试人员来创建xml验证文件。

2 另一个需要注意的是默认情况下可用的验证器。

Struts有许多默认的验证器，常见的包括：Date验证器，Regex验证器和String Length验证器。

#### 6.2.2.1 其他xml验证器

以下是Struts2 中可用的各种字段级别和非字段级别类型的验证列表：

##### 1 int验证器

<field name="age">

<field-validator type="int">

<param name="min">5</param>

<param name="max">50</param>

<message>

Age needs to be between ${min} and ${max}

</message>

</field-validator>

</field>

##### 2 required验证器

<field name="username">

<field-validator type="required">

<message>

username must not be null

</message>

</field-validator>

</field>

##### 3 requiredstring验证器

<field name="username">

<field-validator type="requiredstring">

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

<message>

username must not be null

</message>

</field-validator>

</field>

##### 4 stringlength验证器

<field name="username">

<field-validator type="stringlength">

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

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

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

<message>

Your username needs to be 10 characters long

</message>

</field-validator>

</field>

##### 5 date验证器

<field name="birthday">

<field-validator type="date">

<param name="min">01/01/1990</param>

<param name="max">01/01/2000</param>

<message>

Birthday must be within ${min} and ${max}

</message>

</field-validator>

</field>

##### 6 email验证器

<field name="myEmail">

<field-validator type="email">

<message>

Must provide a valid email

</message>

</field-validator>

</field>

##### 7 url验证器

<field-validator type="url">

<param name="fieldName">myHomePageURL</param>

<message>

Invalid homepage url

</message>

</field-validator>

##### 8 double验证器

<field name="percentage">

<field-validator type="double">

<param name="minExclusive">0.123</param>

<param name="maxExclusive">99.98</param>

<message>

Percentage needs to be between ${minExclusive}

${maxExclusive} (exclusive)

</message>

</field-validator>

</field>

##### 9 expression验证器

基于正则表达式提供验证的非字段级验证器。

<field-validator type="expression">

<param name="expression">....</param>

<message>

Failed to meet Ognl Expression ....

</message>

</field-validator>

##### 10 regex验证器

<field name="myStrangePostcode">

<field-validator type="regex">

<param name="fieldName">myStrangePostcode</param>

<param name="expression">

<![CDATA[([aAbBcCdD][123][eEfFgG][456])]]>

</param>

</field-validator>

</field>

#### 6.2.2.2 xml校验书写规则

字段校验风格：先指定校验的属性：我来校验谁，由谁来校验！

非字段校验风格：先指定校验器：由谁来校验，来校验谁！

字段风格：

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

<!DOCTYPE validators PUBLIC

"-//Apache Struts//XWork Validator 1.0.2//EN"

"http://struts.apache.org/dtds/xwork-validator-1.0.2.dtd">

<validators>

<!-- name值为需要被校验的字段，这里对email字段进行校验 -->

<field name="email">

<!-- type值为校验器名，requiredstring：非空字符串校验器 -->

<field-validator type="requiredstring">

<message>邮箱不能为空</message>

</field-validator>

<!-- 一个字段可以配置多个校验器，email：校验邮箱格式 -->

<field-validator type="email">

<message>邮箱格式不正确</message>

</field-validator>

</field>

<field name="password">

<field-validator type="requiredstring">

<message>密码不能为空</message>

</field-validator>

<!-- stringlength：字符串长度校验 -->

<field-validator type="stringlength">

<!-- 为校验器指定参数实现不同的校验规则，minlength：最少长度 -->

<param name="minLength">6</param>

<message>密码长度不能少于6位</message>

</field-validator>

</field>

<!-- 校验两次密码是否一致，使用 fieldexpression 校验器。expression参数指定判断用的表达式-->

<field name="repassword">

<field-validator type="fieldexpression">

<!-- 表达式要写在 <![CDATA[]]> 内 -->

<param name="expression"><![CDATA[repassword==password]]></param>

<message>两次密码不一至</message>

</field-validator>

</field>

<!-- 校验电话格式，使用regex正则表达示校验，regexExpression参数指定正则表达式 -->

<field name="phone">

<field-validator type="regex">

<!-- 注意！2.3.15前的版本使用 expression 参数 -->

<param name="regexExpression"><![CDATA[\d{11}]]></param>

<message>电话号码格式不正确</message>

</field-validator>

</field>

</validators>

非字段风格：

<validator type="校验器名">

<param name="fieldName">需要被校验的字段</param>

<!--此处需要为不同校验器指定数量不等的校验规则 -->

<param name="参数名">参数值</param>

<!--校验失败后的提示信息 -->

<message>校验失败后的提示信息</message>

</validator>

<validator type="expression">  
    <param name="expression"><![CDATA[password==repassword]]></param>  
    <message>Password is not equals to repassword</message>  
 </validator>

#### 6.2.2.3 visitor验证规则示例

xml文件的头要使用呢1.0.3，或高版本，不然会出现问题。（1.0.2会出问题）

vsitor校验器主要用于校验Action里的复合类型、数组类型或者集合类型，如一个Action里面包含了Person类型的属性，需要验证Person类型的name属性不能为空。

  1)新建Person类：

**package** cn.htb.validation.domain;

**public** **class** Person {

**private** String name;

**private** Integer age;

**public** Person() {

}

**public** Person(String name, Integer age) {

**super**();

**this**.name = name;

**this**.age = age;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Integer getAge() {

**return** age;

}

**public** **void** setAge(Integer age) {

**this**.age = age;

}

}

  2)新建Action类：

**package** cn.htb.validation.action;

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

**import** cn.htb.validation.domain.Person;

**public** **class** MyValidation **extends** ActionSupport{

**private** Person person ;

**public** **void** setPerson(Person person) {

**this**.person = person;

}

**public** Person getPerson() {

**return** person;

}

**public** String person(){

System.***out***.println(person.getName());

**return** "success";

}

**public** String person2(){

**return** "login";

}

}

3)新建visitor验证规则文件，该文件需要和Action类在同一目录下，命名规则和上面讲的一样ActionClassName-validation.xml，这里为ValidateAction-validation.xml：

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

<!DOCTYPE validators PUBLIC

"-//Apache Struts//XWork Validator 1.0.3//EN"

"http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">

<validators>

<field name=*"person"*>

<field-validator type=*"visitor"*>

<param name=*"context"*>test</param>

<param name=*"appendPrefix"*>true</param>

<!--指定校验失败的提示信息前缀-->

<message>错误：</message>

</field-validator>

</field>

</validators>

4)新建Person类的name属性验证规则文件，该文件需要和Person类在同一目录下，命名规则为ClassName-context值-validation.xml，ClassName为要验证的类，context值为<param name="context">test</param>中的内容，-validation为固定内容，这里的文件名为Person-test-validation.xml：

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

<!DOCTYPE validators PUBLIC

"-//Apache Struts//XWork Validator 1.0.3//EN"

"http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">

<validators>

<field name=*"name"*>

<field-validator type=*"requiredstring"*>

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

<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>

5)新建jsp页面：

<body>

<s:fielderror name="name"/>

<form action="${pageContext.servletContext.contextPath}/validateAction.action">

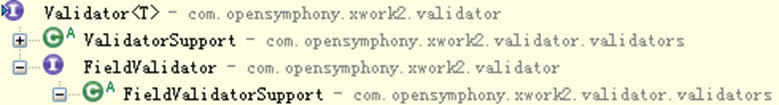
姓名：<input type="text" name="person.name"><br>

<input type="submit" value="提交">

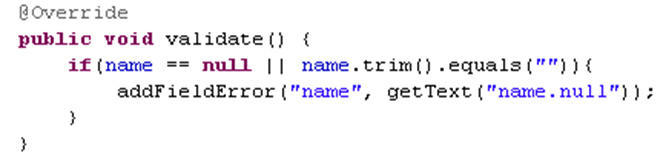
</form>

</body>

### 6.2.3 自定义校验器

自定义验证器  
  
  
自定义验证器必须实现 Validator 接口.  
ValidatorSupport 和 FieldValidatorSupport 实现了 Validator 接口  
  
  
1 若需要普通的验证程序, 可以继承 ValidatorSupport 类  
2 若需要字段验证程序, 可以继承 FieldValidatorSupport 类  
3 若验证程序需要接受一个输入参数, 需要为这个参数增加一个相应的属性  
4 注册验证程序:  
 自定义验证器需要在类路径里的某个 validators.xml 文件里注册:  
 验证框架首先在根目录下找validators.xml文件,没找到validators.xml文件,  
 验证框架将调用默认的验证设置,即default.xml里面的配置信息.  
示例代码：  
  
  
自定义一个 18 位身份证验证器  
编写验证器类  
public class IDCardValidation extends FieldValidatorSupport {  
    @Override  
    public void validate(Object object) throws ValidationException {  
        //1、获取字段名称  
        String fieldName = getFieldName();  
        //2、获取值  
        Object value = this.getFieldValue(fieldName, object);  
        //3、验证  
       IDCard idCard = new IDCard();  
       boolean result =  idCard.Verify(value.toString());  
       //4、验证失败加入错误消息  
        if (!result) {  
            addFieldError(fieldName, object);  
        }  
    }  
}  
在 validators.xml 文件中进行注册  
参考 /com/opensymphony/xwork2/validator/validators/default.xml  
<validators>  
       <validator name="idCard"  
       class="org.pan.validation.app.MyValidation.IDCardValidation">  
       </validator>  
</validators>  
在验证配置文件中使用  
 <!-- 测试自定验证器 -->  
<field name="idCard">  
        <field-validator type="idCard">  
                <message>This is Not IDCard</message>  
        </field-validator>  
</field>  
编程验证  
Struts2 提供了一个 Validateable 接口, 可以使 Action 类实现这个接口以提供编程验证功能.

ActionSupport 类已经实现了 Validateable 接口



## 6.3 转换器

HTTP请求上的所有内容都按协议处理为字符串，包括数字，布尔值，整数，日期，小数和其他。对HTTP来说，每个事件都是一个字符串。但是，在Struts类中，你可以具有任意数据类型的属性。

我们如何让Struts自动匹配属性呢？  
Struts使用各种类型的转换器在幕后做了许多繁重的工作。例如，如果你的Action类中有一个integer属性，你不需要执行任何操作，Struts会自动将请求参数转换为integer属性。默认情况下，Struts提供了多个类型的转换器。其中一些列出如下，如果你使用其中一个，那你就不用担心什么了：

Integer，Float，Double，Decimal

Date，Datetime

Arrays，Collections

Enumerations

Boolean

BigDecimal

有时，当你使用自己的数据类型时，有必要添加自己的转换器，以使Struts知道如何在显示之前转换这些值。可以考虑使用下面的POJO类Person**.java**。

### 6.3.1 POJO

**package** cn.htb;

**public** **class** Person {

**private** String name;

**public** Person() {

}

**public** Person(String name) {

**super**();

**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**name**属性，除此之外没有什么特别的。让我们接下来创建另一个包含系统信息的类：ConverterAction**.java**。

**package** cn.htb;

**public** **class** ConverterAction {

**private** Person person = **new** Person("张三");

**private** String address = "北京";

**public** Person getPerson() {

**return** person;

}

**public** **void** setPerson(Person person) {

**this**.person = person;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

**public** String getData(){

**return** "success";

}

}

接着让我们创建一个简单的JSP文件来显示环境和操作系统信息。

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>

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

<%

String path = request.getContextPath();

String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";

%>

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

<html>

<head>

<base href=*"*<%=basePath%>*"*>

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

<meta http-equiv=*"pragma"* content=*"no-cache"*>

<meta http-equiv=*"cache-control"* content=*"no-cache"*>

<meta http-equiv=*"expires"* content=*"0"*>

<meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>

<meta http-equiv=*"description"* content=*"This is my page"*>

<!--

<link rel="stylesheet" type="text/css" href="styles.css">

-->

</head>

<body>

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

address:<s:property value=*"address"*/>

</body>

</html>

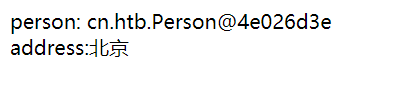
struts.xml

<action name="converter" class="cn.htb.ConverterAction" method="getData">

<result name="success">/converter.jsp</result>

</action>

### 6.3.2 不使用转换器的结果：



上面的输出有什么问题？

Struts知道如何显示和转换“北京”字符串和其他内置数据类型，但它不知道如何处理Person类型的属性。

所以，它只是简单地在类上调用了**toString()**方法。

为了解决这个问题，让我们现在为Person类创建和注册一个简单的**TypeConverter**。创建一个名为**PersonConverter.java**的类，包含有以下内容：

### 6.3.3 自定义转换器：

继承SturtsTypeConverter 或者 DefaultTypeConverter

转换器一：对象向string转换

**package** cn.htb;

**import** java.util.Map;

**import** org.apache.struts2.util.StrutsTypeConverter;

**public** **class** PersonConverter **extends** StrutsTypeConverter{

@Override

**public** Object convertFromString(Map context, String[] values, Class toClass) {

Person person = **new** Person(values[0]);

**return** person;

}

@Override

**public** String convertToString(Map context, Object o) {

Person person = (Person)o;

**return** person == **null** ? **null** : person.getName();

}

}

PersonConverter扩展了StrutsTypeConverter类，并告诉Struts如何通过覆盖convertFromString()和convertToString()两个方法将Person转换为String，反之亦然。

转换器二：向对象转换

import java.util.Map;

import java.util.StringTokenizer;

import ognl.DefaultTypeConverter;

import simpleStruts2.bean.User;

public class UserConverter extends DefaultTypeConverter {

@Override

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

//类型转换方向有页面字符串转换为自定义类型对象，或自定义类型对象转换为字符串

//toType参数代表你需要转换的目标类型

//value参数是具体的需要转换的对象,如果value是页面往自定义类型转，则value应为String[]类型

if(User.class == toType){//从页面向后台对象转换

String[] str = (String[])value;

String firstValue = str[0];//取得页面提交的值

//等价于String[] st = firstValue.split(";");

StringTokenizer st = new StringTokenizer(firstValue,";");

String username = st.nextToken();

String password = st.nextToken();

User user = new User();

user.setUsername(username);

user.setPassword(password);

return user;

}

return null;

}

}

### 6.3.4 注册转换器

现在，让我们在使用前先在应用程序中注册这个转换器，有两种方式注册转换器。

#### 6.3.4.1 局部转换器

如果转换器将只在特定的操作中使用，那么你必须创建一个命名为'[action-class]'-conversion.properties的属性文件，

所以，在我们的例子中，创建了一个包含以下注册表的名为ConverterAction-conversion.properties文件：

person=cn.htb.PersonConverter

在上面的例子中，“person”是**ConverterAction.java**类中属性的名称，用以告诉Struts使用**PersonConverter**来转换这个属性。

#### 6.3.4.2 全局转换器

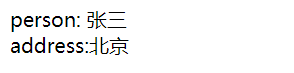
全局注册转换器，以便它可以在整个应用程序中使用。

需要在classpath路径下创建一个名为**xwork-conversion.properties**的属性文件，使用以下内容：

cn.htb.Person = cn.htb.PersonConverter

这简单地将转换器进行了全局注册，使得Struts可以在每次遇到类型为Person的对象时自动执行转换。

现在，如果你重新编译并重新运行程序，你会得到一个更好的输出结果，如下所示：



显然，现在的结果是更好的，这意味着我们的Struts转换器工作正常。这就是如何根据你的要求创建多个转换器和注册使用它们。

## 6.4 异常处理器

Struts提供了一种更简单的方法来处理未捕获的异常，并将用户重定向到专门的错误页面。你可以很轻松地配置Struts为不同的异常显示不同的错误页面。  
Struts通过使用“exception”拦截器来使异常处理变得简单。

“exception”拦截器被作为默认堆栈的一部分包含其中，所以你不必对它做任何额外的配置。它提供开箱即用的功能可供使用。

### 6.4.1 局部异常映射

你的**struts.xml**应该如下：

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

<action name=*"test"* class=*"cn.htb.exception.ExceptionAction"* method=*"test"*>

<exception-mapping name=*"error"* result=*"error"* exception=*"java.lang.NullPointerException"*></exception-mapping>

<result name=*"success"*>/success.jsp</result>

<result name=*"error"*>/error.jsp</result>

</action>

</package>

error.jsp页面：

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"* isELIgnored=*"false"*%>

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

<%

String path = request.getContextPath();

String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";

%>

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

<html>

<head>

<base href=*"*<%=basePath%>*"*>

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

<meta http-equiv=*"pragma"* content=*"no-cache"*>

<meta http-equiv=*"cache-control"* content=*"no-cache"*>

<meta http-equiv=*"expires"* content=*"0"*>

<meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>

<meta http-equiv=*"description"* content=*"This is my page"*>

<!--

<link rel="stylesheet" type="text/css" href="styles.css">

-->

</head>

<body>

EL获取异常信息：${exception.message} <br>

property获取：<s:property value=*"exception.message"*/>

<hr>

异常堆栈信息：<s:property value=*"exceptionStack"*/>

</body>

</html>

除此之外，Struts2框架还带有一个“logging”拦截器来记录异常。通过启用记录器记录未捕获的异常，我们可以轻松地查看堆栈跟踪记录，并找出哪里出错了。

### 6.4.2 全局异常映射

现在，我们可以设置一个全局异常，这将应用于所有的action。例如，为了捕获相同的NullPointerException异常，

我们可以在<package...>标签中添加**<global-exception-mappings...>**标签，并在struts.xml文件的<action...>标签中添加<result...>标签，

如下：

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

<global-exception-mappings>

<exception-mapping result=*"error"* exception=*"java.lang.NullPointerException"*></exception-mapping>

</global-exception-mappings>

<action name=*"test2"* class=*"cn.htb.exception.ExceptionAction"* method=*"test"*>

<result name=*"success"*>/success.jsp</result>

<result name=*"error"*>/error.jsp</result>

</action>

</package>

### 6.4.3 页面接收异常处理提示信息

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

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

<h2>用户名不正确时的异常提示</h2>

使用EL表达式输出： ${exception.message}<br/><

使用<s:property>输出： <s:property value="exception.message"/><br/>

<s:property value="exceptionStack"/>

===================================================================

可以采用两种方式在页面中输出异常提示信息

【第一种】${exception.message}

【第二种】<s:property value="exception.message"/>

exception对应产生的Exception对象

exception.message指的是对象的属性

这里的message就是在UsernameException中定义的message属性

===================================================================

如果将result设为为type="redirect"的话，即客户端跳转

那么在页面中将无法接收<s:property value="exception"/>信息

所以如果想在页面中打印出异常信息的话，就应该将type设置为跳转

所谓的跳转就是将type设置为dispatcher，或者是不设置，按照默认

===================================================================

【补充】<s:property/>标签还可以打印出堆栈信息

即<s:property value="exceptionStack"/>

其中exceptionStack是固定不变的，不是随便写的

===================================================================

在实际开发中，有的人会这么写

<--

<s:property value="exception"/>

<HR>

<s:property value="exceptionStack"/>

-->

也就是说在给用户显示异常信息的页面中打印输出人性化的提示信息

并且在该页使用<!-- -->注释掉原本需要打印的异常信息

这样的好处就是在页面中，用户看到的仍是我们设定给他的提示信息

但是我们在这个页面中右键查看源代码时

就可以在源代码中查看本应在控制输出的异常信息和异常堆栈信息了

也就是说将异常信息和堆栈信息隐藏在异常页面的源代码中，而不显示给用户

比较适用于项目上线之后，当出现错误时，方便查看错误信息，快速修正BUG

# 13 Struts2 注释

如前章节中所讲到的，Struts提供了两种形式的配置。传统的方式是对所有配置使用struts.xml文件。到目前为止，我们在教程里已经看到了好些这样的例子。配置Struts的另一种方法是使用Java5 的注释功能。使用struts注释，我们可以实现零配置。  
要在项目中开始使用注释，请确保WebContent/WEB-INF/lib文件夹中包含以下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

pom.xml额外注解依赖：

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-convention-plugin</artifactId>

<version>2.5.18</version>

</dependency>

这里我们将举一个Person的例子，person的名字和年龄使用一个简单的页面捕获，

我们将进行两次验证，以确保用户始终输入一个名字，并且年龄应在28和65之间。

## 13.1 创建person.jsp

首先，先开始写用来收集上面提到的Person相关信息的主页JSP文件person.jsp。

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>

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

<%

String path = request.getContextPath();

String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";

%>

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

<html>

<head>

<base href=*"*<%=basePath%>*"*>

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

<meta http-equiv=*"pragma"* content=*"no-cache"*>

<meta http-equiv=*"cache-control"* content=*"no-cache"*>

<meta http-equiv=*"expires"* content=*"0"*>

<meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>

<meta http-equiv=*"description"* content=*"This is my page"*>

<!--

<link rel="stylesheet" type="text/css" href="styles.css">

-->

</head>

<body>

<s:form action=*"personAction"* 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>

## 13.2 创建视图

我们将使用JSP文件的**success.jsp**，在定义的action返回SUCCESS的情况下调用它。

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>

<%

String path = request.getContextPath();

String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";

%>

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

<html>

<head>

<base href=*"*<%=basePath%>*"*>

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

<meta http-equiv=*"pragma"* content=*"no-cache"*>

<meta http-equiv=*"cache-control"* content=*"no-cache"*>

<meta http-equiv=*"expires"* content=*"0"*>

<meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>

<meta http-equiv=*"description"* content=*"This is my page"*>

<!--

<link rel="stylesheet" type="text/css" href="styles.css">

-->

</head>

<body>

This is success page. <br>

</body>

</html>

## 13.3 创建Action

Action是使用注释的地方。让我们重新定义具有注释的action类**PersonAction**，然后如下所示在**PersonAction.java**文件中添加一个名为**validate()**的方法。需要确保action类扩展了**ActionSupport**类，否则将不会执行validate方法。

package cn.htb;

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 PersonAction 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;

}

}

我们在这个例子中使用了一些注释，让我们逐个了解一下：

首先是例子中包含的**Results**注释，Results注释是结果的集合。在这个注释集合下，有两个结果注释。它们包含对应于execute方法结果的名称，还包含对应于execute()的返回值所提的视图位置。

下一个注释是**Action**注释，这可用于装饰execute()方法。Action方法还接收一个值，该值是调用action的URL。

最后，我们使用了两个**validation**注释。我们已经配置了必填字段验证器上的**name**字段和整数范围验证器上的**age**字段，此外还为验证指定了自定义的信息。

写法二：

**package** cn.htb;

**import** org.apache.struts2.convention.annotation.Action;

**import** org.apache.struts2.convention.annotation.Namespace;

**import** org.apache.struts2.convention.annotation.ParentPackage;

**import** org.apache.struts2.convention.annotation.Result;

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

**import** com.opensymphony.xwork2.validator.annotations.IntRangeFieldValidator;

**import** com.opensymphony.xwork2.validator.annotations.RequiredFieldValidator;

// 结果集配置

@ParentPackage("struts-default")

@Namespace("/")

**public** **class** PersonAction **extends** ActionSupport{

**private** String name;

**private** **int** age;

//action动作配置

@Action(value="/login2",results={

@Result(name="login2", location="/person.jsp"),

})

**public** String login2(){

**return** "login2";

}

//action动作配置

@Action(value="/personAction",results={

@Result(name="input", location="/person.jsp"),

@Result(name="success", location="/success.jsp")

})

**public** String execute(){

**return** ***SUCCESS***;

}

//校验注解

@RequiredFieldValidator( message = "The name is required" )

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

//校验注解

@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;

}

}

## 13.4 配置文件

实际上我们真的不需要**struts.xml**配置文件，可以删除这个文件，直接来查看**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.filter.StrutsPrepareAndExecuteFilter

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

</filter-mapping>

</web-app>

## 13.5 Struts2 注释类型

Struts2 应用程序可以使用Java5注释来替代XML和Java属性的配置。你可以查看与不同类别相关的最重要注释的列表：[Struts2 注释类型](https://www.w3cschool.cn/struts_2/struts_annotations_types.html" \t "_blank)。

Struts2 应用程序可以使用Java5注释来替代XML和Java属性的配置。以下是与不同类别相关的最重要注释的列表：

### 13.5.1 Action注解

#### 1 Namespace注释（Action注释）

@Namespace注释允许在**Action**类中定义Action的命名空间，而不是基于零配置的约定。

@Namespace("/content")

public class Employee extends ActionSupport{

...

}

#### 2 Result注释（Action注释）

@Result注释允许在Action类中定义Action的结果，而不是XML文件。

@Result(name="success", value="/success.jsp")

public class Employee extends ActionSupport{

...

}

#### 3 Results注释（Action注释）

@Results注释定义了一个Action的一组结果。

@Results({

@Result(name="success", value="/success.jsp"),

@Result(name="error", value="/error.jsp")

})

public class Employee extends ActionSupport{

...

}

### 13.5.2 拦截器注解

#### 1 After注释（拦截器注释）

@After注释标记需要在执行主action方法和结果后调用的action方法（忽略返回值）。

public class PersonAction extends ActionSupport{

@After

public void isValid() throws ValidationException {

// validate model object, throw exception if failed

}

public String execute() {

// perform secure action

return SUCCESS;

}

}

#### 2 Before注释（拦截器注释）

@Before注释标记需要在执行主action方法和结果之前调用的action方法（忽略返回值）。

public class PersonAction extends ActionSupport{

@Before

public void isAuthorized() throws AuthenticationException {

// authorize request, throw exception if failed

}

public String execute() {

// perform secure action

return SUCCESS;

}

}

#### 3 BeforeResult注释（拦截器注释）

@BeforeResult注释标记需要在结果之前执行的action方法（忽略返回值）。

public class PersonAction extends ActionSupport{

@BeforeResult

public void isValid() throws ValidationException {

// validate model object, throw exception if failed

}

public String execute() {

// perform action

return SUCCESS;

}

}

### 13.5.3 校验注解

#### 1 ConversionErrorFieldValidator注释（验证注释）

此验证注释检查字段是否存在任何转换错误，并在存在时应用。

public class PersonAction extends ActionSupport{

@ConversionErrorFieldValidator(message = "Default message",

key = "i18n.key", shortCircuit = true)

public String getName() {

return name;

}

}

#### 2 DateRangeFieldValidator注释（验证注释）

此验证注释检查日期字段的值是否在指定范围内。

public class PersonAction extends ActionSupport{

@DateRangeFieldValidator(message = "Default message",

key = "i18n.key", shortCircuit = true,

min = "2005/01/01", max = "2005/12/31")

public String getDOB() {

return dob;

}

}

#### 3 DoubleRangeFieldValidator注释（验证注释）

此验证注释检查具有指定范围内值的双字段。如果既不设置min也不设置max，则不会执行任何操作。

public class PersonAction extends ActionSupport{

@DoubleRangeFieldValidator(message = "Default message",

key = "i18n.key", shortCircuit = true,

minInclusive = "0.123", maxInclusive = "99.987")

public String getIncome() {

return income;

}

}

#### 4 EmailValidator注释（验证注释）

如果该字段包含非空字符串，则该验证注释将检查该字段是否为有效的电子邮件地址。

public class PersonAction extends ActionSupport{

@EmailValidator(message = "Default message",

key = "i18n.key", shortCircuit = true)

public String getEmail() {

return email;

}

}

#### 5 ExpressionValidator注释（验证注释）

此非字段级验证器验证所提供的正则表达式。

@ExpressionValidator(message = "Default message", key = "i18n.key",

shortCircuit = true, expression = "an OGNL expression" )

#### 6 IntRangeFieldValidator注释（验证注释）

此验证注释检查数字字段是否具有指定范围内的值。如果既不设置min也不设置max，则不会执行任何操作。

public class PersonAction extends ActionSupport{

@IntRangeFieldValidator(message = "Default message",

key = "i18n.key", shortCircuit = true,

min = "0", max = "42")

public String getAge() {

return age;

}

}

#### 7 RegexFieldValidator注释（验证注释）

此注释使用正则表达式验证字符串字段。

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

#### 8 RequiredFieldValidator注释（验证注释）

此验证注释检查字段是否为非空。注释必须在方法级别应用。

public class PersonAction extends ActionSupport{

@RequiredFieldValidator(message = "Default message",

key = "i18n.key", shortCircuit = true)

public String getAge() {

return age;

}

}

#### 9 RequiredStringValidator注释（验证注释）

此验证注释检查String字段不为空（即非null，长度大于0）。

public class PersonAction extends ActionSupport{

@RequiredStringValidator(message = "Default message",

key = "i18n.key", shortCircuit = true, trim = true)

public String getName() {

return name;

}

}

#### 10 StringLengthFieldValidator注释（验证注释）

此验证器检查字符串字段是否具有正确的长度，它假定该字段是一个字符串。如果既没有设置minLength也没有设置maxLength，则不会执行任何操作。

public class PersonAction extends ActionSupport{

@StringLengthFieldValidator(message = "Default message",

key = "i18n.key", shortCircuit = true,

trim = true, minLength = "5", maxLength = "12")

public String getName() {

return name;

}

}

#### 11 UrlValidator注释（验证注释）

此验证器检查字段是否是有效的URL。

public class PersonAction extends ActionSupport{

@UrlValidator(message = "Default message",

key = "i18n.key", shortCircuit = true)

public String getURL() {

return url;

}

}

#### 12 Validations注释（验证注释）

如果要使用多个相同类型的注释，这些注释必须嵌套在@Validations()注释中。

public class PersonAction 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;

}

}

#### 13 CustomValidator注释（验证注释）

此注释可用于自定义验证器。使用ValidationParameter注释提供其他参数。

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

### 13.5.4 类型转换注解

#### 1 Conversion注释（类型转换注释）

这是类型级别的类型转换的标记注释。转换注释必须在类型级别应用。

@Conversion()

public class ConversionAction implements Action {

}

#### 2 CreateIfNull注释（类型转换注释）

此注释为类型转换设置CreateIfNull。CreateIfNull注释必须在字段或方法级别应用。

@CreateIfNull( value = true )

private List<User> users;

#### 3 Element注释（类型转换注释）

此注释设置类型转换的元素。Element注释必须在字段或方法级别应用。

@Element( value = com.acme.User )

private List<User> userList;

#### 4 Key注释（类型转换注释）

此注释设置类型转换的key。Key注释必须在字段或方法级别应用。

@Key( value = java.lang.Long.class )

private Map<Long, User> userMap;

#### 5 KeyProperty注释（类型转换注释）

此注释设置类型转换的KeyProperty。KeyProperty注释必须在字段或方法级别应用。

@KeyProperty( value = "userName" )

protected List<User> users = null;

#### 6 TypeConversion注释（类型转换注释）

此注释用于类和应用程序范围的转换规则。TypeConversion注释可以在属性和方法级别应用。

@TypeConversion(rule = ConversionRule.COLLECTION,

converter = "java.util.String")

public void setUsers( List users ) {

this.users = users;

}

# 14 标签

## 14.1 Struts2 控制标签

Struts2 标签有一组标签，可以方便地控制页面执行的流程。以下是主要的Struts2 控制标签：

### 14.1.1 if和else标签

这些标签执行在每种语言中找到的基本条件流。“If”标签可以单独使用，也可以与“Else If”标签，和单个或是多个“Else”标签一起使用，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>

示例：

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

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

<html>

<head>

<title>Example of If and Else</title>

</head>

<body>

<b>Example of If and Else</b><br/>

<s:if test="name=='Mike'">

You have selected 'Mike'.

</s:if>

<s:elseif test="name=='Jason'">

You have selected 'Jason'

</s:elseif>

<s:else>

You have not selected 'Mike' or 'Jason'.

</s:else>

</body>

</html>

这里，如果在“test”属性中指定的条件返回true，if标签返回true。在我们的例子中，我们将其与“Mike”进行比较。如果名称是Mike，标签返回true，则印出字符串；否则“elseif”块被执行，如果不满足，则else块被执行。这与在Java语言中传统的if，else if和else没有区别。

### 14.1.2 iterator标签

这些iterator将迭代一个值。可迭代值可以是以下任一值：java.util.Collection，java.util.Iterator。

在迭代一个iterator时，可以使用**Sort**标签对结果进行排序，或者使用**SubSet**标签来获取列表或数组的子集。  
以下示例是检索值栈上当前对象的getDays()方法的值，并使用它迭代。<s:property/>标签印出迭代器的当前值。

<s:iterator value="days">

<p>day is: <s:property/></p>

</s:iterator>

示例：

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

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

<html>

<head>

<title>Employees</title>

</head>

<body>

<b>Example of Iterator Tag</b><br/>

<s:iterator value="employees">

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

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

</s:iterator>

<br/><br/>

<b>Employees sorted by Department</b><br/>

<s:bean name="cn.w3cschool.struts2.DepartmentComparator" var="deptComparator" />

<s:sort comparator="deptComparator" source="employees">

<s:iterator>

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

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

</s:iterator>

</s:sort>

<br/><br/>

<b>SubSet Tag - Employees working in Recruitment department </b><br/>

<s:subset decider="recruitmentDecider" source="employees">

<s:iterator>

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

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

</s:iterator>

</s:subset>

<br/><br/>

<b>SubSet Tag - Employees 2 and 3 </b><br/>

<s:subset start="1" count="2" source="employees">

<s:iterator>

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

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

</s:iterator>

</s:subset>

</body>

</html>

iterator标签

我们使用**iterator**标签来迭代employee列表，提供“employees”属性作为iterator标签的源。

在iterator标签中，我们可以访问employee列表中的Employee对象，打印员工的名字，以及他们的部门。

Sort标签

首先，我们将**DepartmentComparator**声明为bean，给这个bean命名为**deptComparator**。

然后我们使用**sort**标签，指定“employees”列表作为源和指定“deptComparator”作为比较器使用。

最后，按照前面的例子，迭代列表并打印员工名字。从输出中可以看到，将会打印按部门排序的员工列表。

Subset标签

**subset**标签用于获取列表或数组的子集。

有两种类型的subset标签：

在第一个示例中，我们使用**recreutationDecider**获取在招聘部门工作的员工列表（请参阅Employee.java中的getRecruitmentDecider()方法:如下）。  
在第二个例子中，我们不使用任何决策者，而是在列表中的元素2和3之后。

subset标签接受两个参数“count”和“start”。“start”确定子集的起始点，“count”确定子集的长度。

package cn.w3cschool.struts2;

import java.util.ArrayList;

import java.util.List;

import org.apache.struts2.util.SubsetIteratorFilter.Decider;

public class Employee {

private String name;

private String department;

public Employee(){}

public Employee(String name,String department)

{

this.name = name;

this.department = department;

}

private List employees;

private List contractors;

public String execute() {

employees = new ArrayList();

employees.add(new Employee("George","Recruitment"));

employees.add(new Employee("Danielle","Accounts"));

employees.add(new Employee("Melissa","Recruitment"));

employees.add(new Employee("Rose","Accounts"));

contractors = new ArrayList();

contractors.add(new Employee("Mindy","Database"));

contractors.add(new Employee("Vanessa","Network"));

return "success";

}

public Decider getRecruitmentDecider() {

return new Decider() {

public boolean decide(Object element) throws Exception {

Employee employee = (Employee)element;

return employee.getDepartment().equals("Recruitment");

}

};

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDepartment() {

return department;

}

public void setDepartment(String department) {

this.department = department;

}

public List getEmployees() {

return employees;

}

public void setEmployees(List employees) {

this.employees = employees;

}

public List getContractors() {

return contractors;

}

public void setContractors(List contractors) {

this.contractors = contractors;

}

}

### 14.1.3 merge标签

这些**merge**标签采用两个或多个列表作为参数，并将它们合并在一起，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>

示例：

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

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

<html>

<head>

<title>Employees</title>

</head>

<body>

<b>Employees and Contractors Merged together</b>

<br />

<s:merge id="allemployees">

<s:param value="employees" />

<s:param value="contractors" />

</s:merge>

<s:iterator value="allemployees">

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

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

</s:iterator>

</body>

</html>

**merge**标签需要两个或更多列表作为参数。我们需要给merge标签一个**id**，以便以后可以重新使用它。

在此示例中，我们提供employees和contractors作为merge标签的参数。然后，使用“allemployees”id迭代合并列表并打印员工详细信息。

### 14.1.4 append标签

这些**append**标签采用两个或多个列表作为参数，并将它们全部附加在一起，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>

示例：

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

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

<html>

<head>

<title>Employees</title>

</head>

<body>

<b>Employees and Contractors Merged together</b>

<br />

<s:append id="allemployees">

<s:param value="employees" />

<s:param value="contractors" />

</s:append >

<s:iterator value="allemployees">

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

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

</s:iterator>

</body>

</html>

**append**标签需要两个或多个列表作为参数。我们需要给append标签一个**id**，以便以后可以重新使用它。在此示例中，我们提供employees和contractors作为merge标签的参数。然后，使用“allemployees”id迭代附加的列表并打印员工详细信息。

### 14.1.5 generator标签

这些**generator**标签基于提供的val属性生成迭代器。下面的generator标签生成一个迭代器并使用iterator标签打印出来。

<s:generator val="%{'aaa,bbb,ccc,ddd,eee'}">

<s:iterator>

<s:property /><br/>

</s:iterator>

</s:generator>

示例：

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

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

<html>

<head>

<title>Hello World</title>

</head>

<body>

<h2>Example of Generator Tag</h2>

<h3>The colours of rainbow:</h3>

<s:generator val="%{'Violet,Indigo,Blue,

Green,Yellow,Orange,Red '}" count="7"

separator=",">

<s:iterator>

<s:property /><br/>

</s:iterator>

</s:generator>

</body>

</html>

这里我们创建一个**generator**标签，要求它解析包含形成彩虹的颜色列表，用逗号分隔字符串。我们告诉生成器标签，分隔符是“，”，并需要列表中的所有七个值。如果我们只对前三个值感兴趣，那么我们将计数设置为3。在generator标签中，我们使用迭代器来迭代generator标签创建的值，然后打印属性的值 。

## 14.2 Struts2 数据标签

Struts2 的**数据标签**主要用于操作页面上显示的数据。下面列出了主要的数据标签：

### 1 action标签

此标签允许开发人员通过指定action名称和可选的命名空间直接从JSP页面调用action。

标签的正文内容用于呈现action的结果。

在struts.xml中为此action定义的任何结果处理程序将会被忽略，除非指定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" />

查看详细示例

action标签允许开发者从视图页面执行action。

他们可以通过指定action名称来实现此目的，将“executeResult”参数设置为“true”，以便直接在视图中呈现结果。

或者，他们可以将此参数设置为“false”，但使用action方法展现的request属性。

<s:action name="myAction\_generator" executeResult="true"></s:action>

### 2 include标签

这些include标签将用于在另一个JSP页面中包含一个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>

查看详细示例

Struts的include标签非常类似于jsp的include标签，它很少被使用。

我们已经看到了如何使用<s:action>标签将struts action的输出包含到jsp中。

而<s:include>标签略有不同，它允许你将jsp，servlet或任何其他资源（除了struts action之外的其他资源）的输出包含到jsp中。

在幕后，它完全类似于<jsp:include>，但它允许你传递参数到包含的文件中，并且它也是Struts框架的一部分。

### 3 bean标签

这些**bean**标签实例化一个符合JavaBeans规范的类。这个标签有一个主体，可以包含一些Param元素来设置任何mutator方法。如果在BeanTag上设置了var属性，它将把实例化的bean放入值栈的Context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>

查看详细示例

bean标签是**set**和**push**标签的组合，它允许你创建一个对象的新实例并设置变量的值。然后它使该bean在值栈中可用，以便它可以在JSP页面中使用。  
Bean标签需要一个java bean来处理。因此需要遵循标准的java bean规则，也就是说，bean应该有一个没有参数的构造函数。

要公开和使用的所有属性都应该具有getter和setter方法。

<head>

<title>Hello World</title>

</head>

<body>

<s:bean name="org.apache.struts2.util.Counter" var="counter">

<s:param name="first" value="20"/>

<s:param name="last" value="25" />

</s:bean>

<ul>

<s:iterator value="#counter">

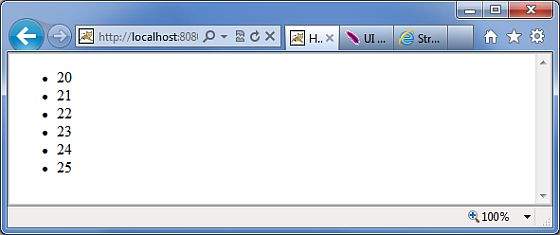
<li><s:property /></li>

</s:iterator>

</ul>

</body>

</html>



### 4 date标签

**data**标签允许以快速简单的方式格式化日期。

用户可以指定自定义日期格式（例如“dd/MM/yyyy hh:mm”），可以生成易读的符号（例如“在2小时14分钟内”），

或者可以使用属性文件中的key:“struts.date.format”来回退到预定义的格式。

<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" />

查看详细示例

### 5 param标签

这些**param**标签可用于参数化其他标签。此标签具有以下两个参数。

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>

查看详细示例

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

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

<html>

<head>

<title>Hello World</title>

</head>

<body>

<s:bean name="org.apache.struts2.util.Counter" var="counter">

<s:param name="first" value="20"/>

<s:param name="last" value="25" />

</s:bean>

<ul>

<s:iterator value="#counter">

<li><s:property /></li>

</s:iterator>

</ul>

</body>

</html>

### 6 property标签

这些**property**标签用于获取一个值的属性，如果没有指定，它将默认为在值栈的顶部。

<s:push value="myBean">

<!-- Example 1: -->

<s:property value="myBeanProperty" />

<!-- Example 2: -->TextUtils

<s:property value="myBeanProperty" default="a default value" />

</s:push>

查看详细示例

**property**标签用于获取值的属性，如果没有指定，它将默认为在值栈的顶部。此示例显示了三个简单的数据标签的用法，即**set**，**push**和**property**。

<%@ 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>

<p>The environment name property can be accessed in three ways:</p>

(Method 1) Environment Name:

<s:property value="environment.name"/><br/>

(Method 2) Environment Name:

<s:push value="environment">

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

</s:push>

(Method 3) Environment Name:

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

<s:property value="myenv"/>

</body>

</html>

现在，我们逐个过一下三个标签选项：

1 在第一种方法中，我们使用property标签获取环境名称的值。因为环境变量在action类中，所以它在值栈中自动可用，我们可以使用**environment.name**属性直接引用它。当一个类中的属性数量有限时，方法一工作正常。

想象一下，如果你在Environment类中有20个属性，每次你需要引用这些变量，你都需要添加“environment.”作为前缀，而这就是需要push标签来处理的地方了。

2 在第二种方法中，我们将“environment”属性推送到值栈。因而在push标签的中，environment属性在值栈的根目录下可用。所以你可以很容易地引用属性，如示例所示。

3 在最后的方法中，我们使用set标签创建一个名为myenv的新变量。此变量的值设置为environment.name。那么，无论我们在哪引用environment的名称，我们都可以使用这个变量。

### 7 push标签：

这些**push**标签用于推送堆栈中的值，以简化使用。

<s:push value="user">

<s:propery value="firstName" />

<s:propery value="lastName" />

</s:push>

### 8 set标签

这些**set**标签为指定范围内的变量赋值。当你希望将变量分配给复杂表达式，然后仅仅引用该变量而不是复杂表达式时，它是很有用的。可应用的范围是应用程序，会话，请求，页面和action。

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

<s:property value="myenv"/>

### 9 text标签

这些**text**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 Examlpe -->

<s:text name="i18n.label.greetings">

<s:param >Mr Smith</s:param>

</s:text>

查看详细示例

**text**标签是用于呈现I18n文本信息的通用标签。按照以下三个步骤之一：

该信息必须在与其关联的action的有相同名称的资源束中。实际上，这意味着你应该在与Java类相同的包中创建与你的类具有相同名称但带有“.properties”扩展名的属性文件。

如果找不到指定的信息，那么标签的主体将用作默认信息。

如果没有使用主体，则将使用信息的名称。

让我们查看下面的例子来理解text标签的用法：

创建action类

**package** cn.htb;

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

**public** **class** MyAction **extends** ActionSupport{

**public** String text(){

**return** "text";

}

}

创建视图

让我们创建包含以下内容**HelloWorld.jsp**：

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>

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

<%

String path = request.getContextPath();

String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";

%>

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

<html>

<head>

<base href=*"*<%=basePath%>*"*>

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

<meta http-equiv=*"pragma"* content=*"no-cache"*>

<meta http-equiv=*"cache-control"* content=*"no-cache"*>

<meta http-equiv=*"expires"* content=*"0"*>

<meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>

<meta http-equiv=*"description"* content=*"This is my page"*>

<!--

<link rel="stylesheet" type="text/css" href="styles.css">

-->

</head>

<body>

<s:i18n name=*"MyAction"*>

<s:text name=*"name.success"*></s:text><br/>

<s:text name=*"name.xyz"*></s:text><br/>

<s:text name=*"name.msg.param"*>

<s:param>你好</s:param>

</s:text>

</s:i18n>

</body>

</html>

配置文件

创建一个与你的action类包名称相同的属性文件。然后创建**MyAction.properties**文件并保存在类的路径中：

name.success = This is success message

name.msg.param = The param example - param : {0}

你的**struts.xml**应该如下所示：

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

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.5.dtd">

<struts>

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

<package name=*"i18n"* namespace=*"/"* extends=*"struts-default"*>

<action name=*"myAction\_\*"* class=*"cn.htb.MyAction"* method=*"{1}"*>

<result name=*"text"*>/page/text.jsp</result>

</action>

</package>

</struts>

### 10 url标签

这些**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>

查看详细示例

**url**标签负责生成URL字符串。这样做的优点是你可以向标签提供参数。让我们通过一个例子来展示url标签的用法。

创建包含以下内容的HelloWorld.jsp：

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

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

<html>

<head>

<title>Hello World</title>

</head>

<body>

<s:url id="login" action="login" var="myurl">

<s:param name="user">Zara</s:param>

</s:url>

<a href='<s:property value="#myurl"/>'>

<s:property value="#myurl"/></a>

</body>

</html>

这里我们生成一个url链接到“login.action”，我们给这个URL命名为“myurl”。这是为了使我们可以在jsp文件中的多个地方重复使用这个url链接。然后，我们为url提供一个名为user的参数。参数值实际上附加到查询字符串中，你可以从上面的输出中看到。  
当你想要基于bean的属性值创建动态超链接时，URL标签起到主要作用。

## 14.3 Struts2 表单标签

表单标签列表是Struts UI标签的子集。这些标签有助于渲染Struts Web应用程序所需的用户界面，主要分为三类，本章将介绍这三种类型的UI标签：

使用struts2自带ui时：

注意事项：

使用该标签时，要保证web.xml的路由地址是拦截所有 /\*

### 14.3.1 简单UI标签

我们其实已经在前面的示例中使用了这些标签，接下来将在本章中重新回顾一下。以下是一个简单的视图页面**email.jsp**与几个简单的UI标签：

<%@ 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" />

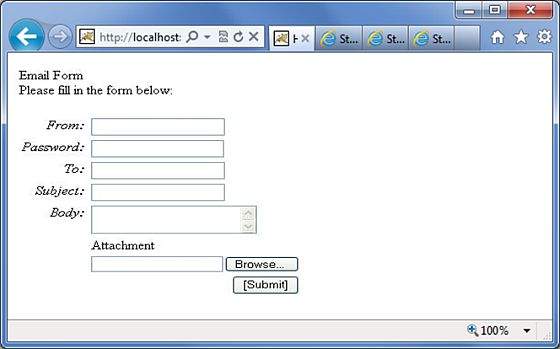
</s:form>

</body>

</html>

如果你了解HTML，那么就知道所有使用的标签都是非常常见的HTML标签，每个标签带有一个额外的前缀“**s:**”以及不同的属性。

当我们执行上面的程序时，将得出以下用户界面，只要你已经为所有的key设置了正确的映射。



1 如图所示，s:head生成Struts2 应用程序所需的javascript和stylesheet元素。  
2 接下来，s:div和s:text元素。

s:div用于呈现HTML Div元素。这对于不喜欢将HTML和Struts标签混合在一起的人很有用，他们可选择使用s:div来渲染div。

如图所示，s:text用于在屏幕上呈现文本。  
3 接下来是相类似的s:form标签。

s:form标签具有确定在何处提交表单的action属性。

因为在表单中有一个文件上传元素，我们必须将enctype设置为multipart。否则，就留空。

在表单标签的末尾，有s:submit标签，这用于提交表单。提交表单时，所有表单值都将提交到s:form标签中指定的action。  
在s:form标签中，我们有一个称为secret的隐藏属性，这将在HTML中呈现一个隐藏元素。在我们的例子中，“secret”元素的值为“abracadabra”。此元素对最终用户不可见，并用于将状态从一个视图传递到另一个视图。

4 接下来是s:label，s:textfield，s:password和s:textarea标签。

这些分别用于渲染标签，输入字段，密码和文本区域。

我们已经在“Struts2 发送电子邮件”章节示例中看到了这些。

这里要注意的重要事情是使用“key”属性。“key”属性用于从属性文件中提取这些控件的标签。我们已经在Struts2本地化/国际化（i18n）一章中讨论了这个特性。  
5 然后是s:file标签，它呈现输入文件上传的组件，此组件允许用户上传文件。

在这个例子中，我们使用了s:file标签的“accept”参数来指定允许上传哪些文件类型。  
6 最后，s:token标签。

token标签生成唯一的token，用于查明表单是否已被两次提交。  
呈现表单时，会将一个隐藏变量放置为token（令牌）值。

例如令牌是“ABC”，提交此表单时，Struts Fitler将根据存储在会话中的令牌进行检查。

如果匹配，则从会话中删除令牌。现在，如果表单意外被重新提交（通过刷新或通过点击浏览器后退按钮），表单将重新提交，用“ABC”作为令牌。

在这种情况下，过滤器将对照存储在会话中的令牌再次进行检查。但是因为令牌“ABC”已经从会话中删除，它将不匹配，Struts过滤器将拒绝请求。

### 14.3.2 群组UI标签

群组UI标签用于创建单选按钮和复选框。让我们看一个简单的带有复选框和单选按钮标签的视图页面**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'}" />

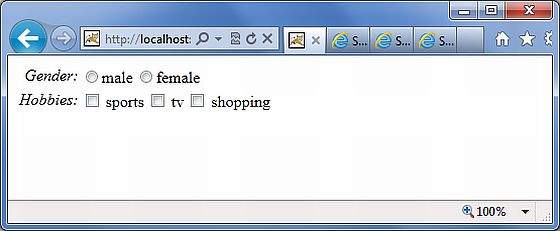
<s:checkboxlist label="Hobbies" name="hobbies" list="{'sports','tv','shopping'}" />

</s:form>

</body>

</html>

当我们执行上面的程序时，我们的输出将类似于以下内容：



现在让我们看看这些例子。

1 在第一个例子中，我们创建一个简单的radiobutton，标签为“Gender”。

name属性对于radiobutton标签是必需的，那么我们指定一个名为“gender”的name。

然后我们提供一个性别列表，该列表用值“male”和“female”填充。

因此，在输出中我们得到一个带有两个值的radiobutton。

2 在第二个例子中，我们创建一个复选框列表。

这是为了收集用户的爱好。用户可以有多个爱好，因此我们使用复选框而不是单选按钮。

复选框用“sports”，“Tv”和“Shopping”填充列表，将这些爱好作为复选框列表。

### 14.3.3 选择UI标签

让我们来探讨Struts提供的Select标签的不同变化。让我们看一个简单的带select标签的视图页面**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"

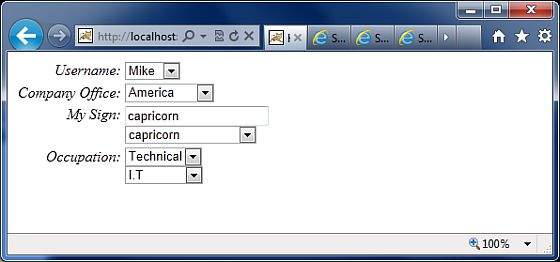
doubleList="top == 'Technical' ? {'I.T', 'Hardware'} : {'Accounting', 'H.R'}" />

</s:form>

</body>

</html>

当我们执行上面的程序时，输出的结果将类似于以下内容：



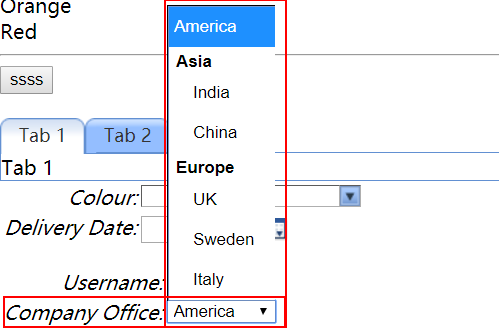
现在，让我们逐一查看每个案例。

首先，select标签呈现HTML选择框。

1 在第一个例子中，我们创建一个名为“username”和标记“username”的简单选择框。选择框将填充包含姓名Mike，John和Smith的列表。

2 在第二个例子中，公司在美国设有总部。它还在亚洲和欧洲设有全球办事处。我们想在一个选择框中显示办公点，但要按全球大陆的名称对全球办事处进行分组。

这是optgroup的用武之地。我们使用s:optgroup标签创建一个新组，给组一个标记和一个单独的列表。



3 在第三个示例中，使用组合框。

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，在这种情况下，输入字段将自动填入用户选择的值。如果用户直接输入值，则将不选择来自选择框的值。

在我们的示例中，我们有组合框列出了星座。选择框只列出四个条目，允许用户输入他的星座，如果它不在列表中。

我们还向选择框中添加一个标题条目。headerentry是显示在选择框顶部的。在示例中，我们要显示“Please Select”。如果用户没有选择任何东西，那么我们假设-1作为值。在某些情况下，我们不希望用户选择一个空值。那么，可以将“emptyOption”属性设置为false。最后，在我们的示例中，我们提供“capricorn”作为组合框的默认值。

4 在第四个例子中，我们有一个双选框。

当要显示两个选择框时，使用double select。

在第一个选择框中选择的值确定在第二个选择框中显示的值。在示例中，第一个选择框显示“Technical”和“Other”。

如果用户选择Technical，我们将在第二个选择框中显示IT和Hardware。

否则，将显示Accounting和HR。这可以使用“list”和“doubleList”属性，如示例所示。

在上面的例子中，我们做一个比较，看看顶部的选择框是否等于Techical。

如果是，那么我们显示IT和Hardware。我们还需要给顶部框（“name ='Occupations'）和底部框（doubleName ='occupations2'）命名。

## 14.4 Struts2 Ajax标签

注意事项：

使用该标签时，要保证web.xml的路由地址是拦截所有 /\*

Struts使用DOJO框架来实现AJAX标签。

首先，执行示例前，你需要将struts2-dojo-plugin-2.2.3.jar添加到类路径。

你可以从struts2 下载的lib文件夹中获取这个文件（C:\struts-2.2.3-all\struts-2.2.3\lib\struts2-dojo-plugin-2.2.3.jar）。

对于这个示例，让我们参照以下内容修改**HelloWorld.jsp**：

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

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

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

<html>

<head>

<title>Hello World</title>

<s:head />

<sx:head />

</head>

<body>

<s:form>

<sx:autocompleter id="Colour" label="Favourite Colour" list="{'red','green','blue'}" /> <!—这个标签的id是必须的，不然会报错 -->

<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>

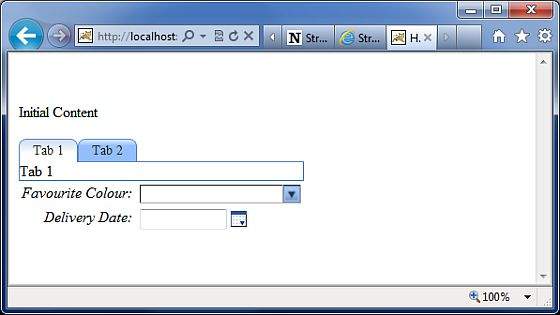
</sx:tabbedpanel>

</s:form>

</body>

</html>

当我们运行上面的例子，将得到以下的输出：



现在，让我们一步一步地完成这个例子。

1 首先要注意的是添加一个带有前缀sx的新标签库。这个（struts-dojo-tags）是为ajax集成专门创建的标签库。

2 然后在HTML头部内，我们称之为sx:head。

这将初始化dojo框架，并使其准备好在页面中被所有AJAX调用。这个步骤很重要，如果没有初始化sx:head，你的ajax调用将无法工作。

3 首先我们有autocompleter标签。

autocompleter标签看起来非常像一个选择框。它使用红色，绿色和蓝色值填充。

但选择框和autocompleter标签之间的不同是autocompleter标签自动完成。

也就是说，如果你开始在gr中输入，它将填充“绿色”。

除此之外，这个标签非常类似于我们前面介绍的s:select标签。

4 接下来，我们有一个日期时间选择器。

此标标签创建一个旁边带有按钮的输入字段。当按下按钮时，显示弹出日期时间选择器。

当用户选择日期时，日期以在tag属性中指定的格式填充到输入文本中。在我们的示例中，我们指定了dd/MM/yyyy作为日期的格式。

5 接下来，是我们在之前的示例中为system.action文件创建的一个url标签。它不必是system.action，它可以是之前创建的任何action文件。

然后我们有一个div，超链接设置为url，延迟设置为2秒。

当你运行时会发生什么，“Initial Content”将显示2秒，然后div的内容将替换为hello.action执行的内容。

6 最后，我们有一个简单的标签面板，有两个标签。选项卡将其标签标记为Tab1和Tab2。

值得注意的是，Struts中的AJAX标签集成仍然是一项进展，并且此集成的成熟度在每个版本中都在缓慢增长。

# 15 Struts2 Spring集成

Spring是当下流行的Web框架，可提供带有许多常见Web任务的简易的集成。那么问题是，当我们有Struts2时，为什么我们需要Spring？因为Spring不仅仅是一个MVC框架，它提供了许多其他在Struts中不可用的好东西。例如：依赖注入可以对任何框架有用。在本章中，我们将通过一个简单的例子来了解如何将Spring和Struts2集成在一起。  
首先，你需要将以下文件添加到Spring的项目的构建路径。你可以从[http://www.springsource.org/download](https://spring.io/projects" \t "_blank)下载并安装最新版本的Spring框架。

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

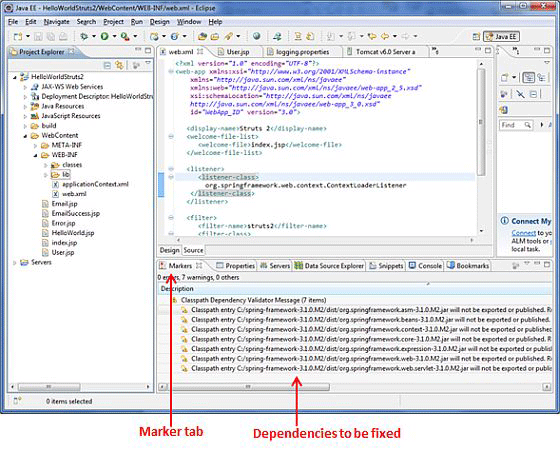
org.springframework.core-x.y.z.M(a).jar

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

最后从你的struts lib目录中添加**struts2-spring-plugin-x.y.z.jar**到你的**WEB-INF/lib**中。如果你使用Eclipse，那么你可能会遇到一个异常java.lang.ClassNotFoundException:org.springframework.web.context.ContextLoaderListener。要解决这个问题，你应该进到**Marker**选项卡中右键逐个点击类依赖关系，然后快速修复以发布/导出所有的依赖关系。最后，确保在Marker选项卡下没有可用的依赖冲突。



现在让我们为Struts-Spring集成设置**web.xml**，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.filter.StrutsPrepareAndExecuteFilter

</filter-class>

</filter>

<filter-mapping>

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

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

</filter-mapping>

</web-app>

这里需要注意的重点是我们配置的监听器。

加载spring上下文文件需要**ContextLoaderListener**。

Spring的配置文件称为**applicationContext.xml**文件，它必须放置在与**web.xml**文件相同的级别。  
让我们创建一个简单的action类**User.java**，它有两个属性 - firstName和lastName。

package cn.w3cschool.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，创建spring配置文件**applicationContext.xml**并实例化**User.java**类。如前所述，此文件应在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="cn.w3cschool.struts2.User">

<property name="firstName" value="Michael" />

<property name="lastName" value="Jackson" />

</bean>

</beans>

如上所示，我们已经配置了用户bean，并且我们已经将值**Michael**和**Jackson**注入到bean中。我们还给这个bean一个名称“userClass”，以便我们可以在别处重新使用它。接下来，让我们在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**文件很简单。它仅用于一个目的：显示用户对象的firstname和lastname的值。

最后，让我们使用**struts.xml**文件将所有实体放在一起。

<?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>

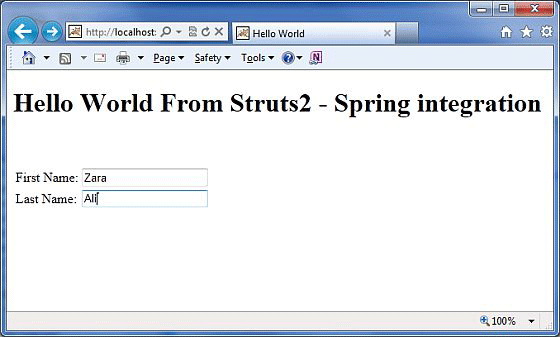
</action>

</package>

</struts>

重点需要注意的是，我们使用id **userClass**来引用类，这意味着我们使用spring为User类做依赖注入。

现在，右键单击项目名称，然后单击**“Export”> “WAR File”**以创建WAR文件。然后在Tomcat的webapps目录中部署WAR文件。最后，启动Tomcat服务器并尝试访问URL http://localhost:8080/HelloWorldStruts2/User.jsp，将显示以下界面：



我们现在已经看到如何将两个伟大的框架结合在一起，这就结束了Struts-Spring集成章节。

# 16 Struts2 Hibernate集成

Hibernate是一个高性能的对象/关系留存和查询服务，它是根据开源GNU宽通用公共许可证（LGPL）许可的，可以免费下载。在这一章。 我们将学习如何实现Struts2 与Hibernate的集成。如果你不熟悉Hibernate，可以查看我们的[Hibernate教程](https://www.w3cschool.cn/hibernate/" \t "_blank)。

## 16.1 数据库设置

对于本教程，我们将使用“struts2\_tutorial”MySQL数据库，使用用户名“root”连接到机器上的这个数据库，不设密码。首先，你需要运行以下脚本。此脚本创建一个名为**student**的新表，并在此表中创建少量记录：

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);

## 16.2 Hibernate配置

接下来让我们创建hibernate.cfg.xml，它是hibernate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">com.mysql.jdbc.Driver</property>

<property name="hibernate.connection.url"> jdbc:mysql://www.w3cschool.cn/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="cn.w3cschool.hibernate.Student" />

</session-factory>

</hibernate-configuration>

让我们过一下hibernate配置文件。

1 首先，我们声明使用MySQL驱动程序。

2 然后我们声明了jdbc url用于连接到数据库。

3 接着我们声明连接的用户名，密码和池大小。

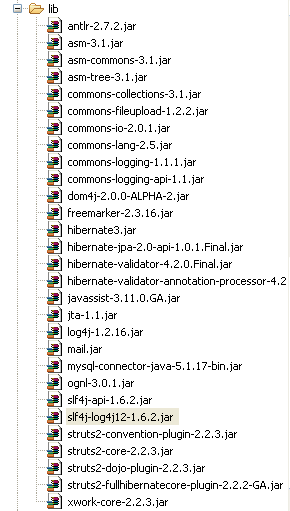
4 我们还指出希望通过将“show\_sql”打开为true来查看日志文件中的SQL。

请通过[Hibernate教程](https://www.w3cschool.cn/hibernate/)了解这些属性的含义。

5 最后，我们将映射类设置为在本章中创建的cn.w3cschool.hibernate.Student。

## 16.3 环境搭建

这个项目需要很多jar文件，以下的是所需的JAR文件的完整列表的屏幕截图：



大多数JAR文件可以作为struts分发的一部分获取。如果你有安装一个应用程序服务器，如glassfish，websphere或jboss，那么你可以从appserver的lib文件夹中获取大多数剩余的jar文件。如果没有，你可以单独下载文件：

Hibernate的jar文件- [Hibernate.org](http://www.hibernate.org/)

Struts的Hibernate插件- [Struts hibernate plugin](https://code.google.com/p/full-hibernate-plugin-for-struts2/downloads/list)

JTA文件- [JTA file](http://www.oracle.com/technetwork/java/javaee/jta/index.html)

Dom4j文件- [Dom4j](http://dom4j.github.io/)

SLF4J文件- [SLF4J](http://www.slf4j.org/)

log4j文件- [log4j](http://logging.apache.org/log4j/1.2/)

其余的文件，你应该能够从struts2 中分配。

## 16.4 Hibernate类

现在让我们为hibernate集成创建所需的java类。按照**Student.java**的内容：

package cn.w3cschool.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，根据Hibernate规范表示的**student**表。它具有对应于student表列名的属性id，firstName和lastName。

接下来让我们创建**StudentDAO.java**文件，如下所示：

package cn.w3cschool.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类的数据访问层。它有方法来列出所有学生，然后保存一个新的学生记录。

## 16.5 Action类

下面的文件**AddStudentAction.java**定义了我们的action类。

我们在这里有两个action方法：execute()和listStudents()。execute()方法用于添加新的学生记录。

我们使用dao的save()方法来实现这一点。另一种listStudents()方法用于列出学生。

我们使用dao的列表方法来获取所有学生的列表。

package cn.w3cschool.struts2;

import java.util.ArrayList;

import java.util.List;

import com.opensymphony.xwork2.ActionSupport;

import com.opensymphony.xwork2.ModelDriven;

import cn.w3cschool.hibernate.Student;

import cn.w3cschool.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。

这是在你的action类处理一个具体的模型类（如Student）而不是个别的属性（如firstName，lastName）时使用。

ModelAware接口需要你实现一个方法来返回模型。在我们的例子中，我们返回“student”对象。

## 16.6 创建视图文件

现在创建包含以下内容的**student.jsp**视图文件：

<%@ 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。在上面部分，我们有一个提交到“addStudent.action”的表单。它接受firstName，lastName和marks。因为addStudent action绑定到ModelAware的“AddSudentAction”，所以将自动创建一个学生bean，其值自动填充为firstName，lastName和marks。  
在下面部分，我们浏览学生列表（参见AddStudentAction.java），并在表中显示firstname，lastname和marks的值。

## 16.7 Struts配置

让我们使用**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" />

<package name="myhibernate" extends="hibernate-default">

<action name="addStudent" method="execute" class="cn.w3cschool.struts2.AddStudentAction">

<result name="success" type="redirect">

listStudents

</result>

</action>

<action name="listStudents" method="listStudents" class="cn.w3cschool.struts2.AddStudentAction">

<result name="success">/students.jsp</result>

</action>

</package>

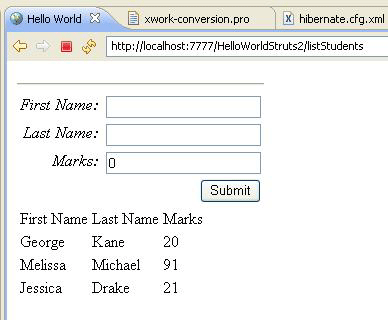
</struts>

要注意的要点是，我们的包“myhibernate”扩展了名为“hibernate-default”的struts2默认包。

然后我们声明两个action：addStudent和listStudents。addStudent调用AddStudentAction类的execute()，然后在成功时调用listStudents方法。

listStudent方法调用AddStudentAction类上的listStudents()，并使用student.jsp作为视图。

现在，右键单击项目名称，然后单击**“Export”> “WAR File”**以创建WAR文件。然后在Tomcat的webapps目录中部署WAR文件。最后，启动Tomcat服务器并尝试访问URL http://localhost:8080/HelloWorldStruts2/student.jsp，将显示以下界面：



在上面部分，我们得到一个表单，输入新学生记录的值，下面部分列出数据库中的学生。继续添加一个新的学生记录，然后按提交。屏幕将刷新，并在每次单击提交时向你显示更新了的列表。