# 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>  <package name="xxxx" namespace="/" extends="struts-default">  <action name="hello" class="cn.itcast.action.HelloAction" method="execute">  <result name="success">/success.jsp</result>  </action>  </package> </struts> |

## package

|  |
| --- |
| <package name="xxxx" namespace="/" extends="struts-default">  Struts2 把各种 action 分门别类地组织成不同的包. 可以把包想象为一个模块. 一个典型的 struts.xml 文件可以有一个或多个包  name: 必须，该 name 属性可被其它 package 引用  namespace: 可选, 如果它没有给出, 则以 “/” 为默认值. 若 namespace 有一个非默认值, 则要想调用这个包里的Action, 就必须把这个属性所定义的命名空间添加到有关的 URI 字符串里  package: 元素通常要对 struts-default.xml 文件里定义的 struts-default 包进行扩展. 这么做了以后, 包里的所有动作就可以使用在 struts-default.xml 文件里的结果类型和拦截器了. |

## action

|  |
| --- |
| action name="hello" class="cn.itcast.action.HelloAction" method="execute">  [http://localhost:8888/struts2/hello](http://localhost:8888/struts2/hello.action)  或  <http://localhost:8888/struts2/hello.action>  action 元素嵌套在 package 元素内部, 它表示一个 Struts请求.  name: 必须，该属性和用户请求 servletPath 之间存在着一一对应关系  class: 可选，没有配置 class 属性, Struts 将把 com.opensymphony.xwork2.ActionSupport 作为其默认值. 配置了 class 属性, 还可以使用 method 属性配置该类的一个动作方法.  method 属性的默认值为 execute |

## result

|  |
| --- |
| <result name="success" type="dispatcher">/success.jsp</result>  result 元素 <action> 的一个子元素, 它告诉 struts 在完成动作后把控制权转交到哪里.  result 元素(的name 属性)对应着 Action 方法的返回值. 因为动作方法在不同情况下可能返回不同的值, 所以同一个 action 元素可能会有多个 result 元素  result 元素的 name 属性建立 <result> 和 Action 方法返回值之间的映射关系。  name 属性的默认值为 “success”  result 元素的 type 属性负责指定结果类型.  type 属性的值必须是在包含当前包或者是当前包的父包里注册过的结果类型. type 属性的默认值为 dispatcher |

# 使用 Filter 作为控制器的 MVC

## MVC 设计模式概览

|  |
| --- |
|  |

## 使用 Filter 作为控制器的 MVC

|  |
| --- |
|  |

## 代码

|  |
| --- |
|  |

### index.jsp

|  |
| --- |
| <%@ page contentType="text/html;charset=UTF-8" language="java" pageEncoding="UTF-8"%> <html>  <head>  <title></title>  </head>  <body>  <a href="product-input.action">Product Input</a>  </body> </html> |

### web.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"  version="4.0">    <filter>  <display-name>FilterDispatcher</display-name>  <filter-name>FilterDispatcher</filter-name>  <filter-class>com.atguigu.struts2.helloworld.FilterDispatcher</filter-class>  </filter>  <filter-mapping>  <filter-name>FilterDispatcher</filter-name>  <url-pattern>\*.action</url-pattern>  </filter-mapping> </web-app> |

### input.jsp

|  |
| --- |
| <%@ page contentType="text/html;charset=UTF-8" language="java" pageEncoding="UTF-8"%> <html> <head>  <title>Title</title> </head> <body>  <form action="product-save.action" method="post">  ProductName: <input type="text" name="productName"/>  <br><br>   ProductDesc: <input type="text" name="productDesc"/>  <br><br>   ProductPrice: <input type="text" name="productPrice" />  <br><br>   <input type="submit" value="Submit"/>  <br><br>  </form> </body> </html> |

### details.jsp

|  |
| --- |
| <%@ page contentType="text/html;charset=UTF-8" language="java" pageEncoding="UTF-8"%> <html> <head>  <title>Title</title> </head> <body>  ProductId: ${requestScope.product.productId }  <br><br>   ProductName: ${requestScope.product.productName }  <br><br>   ProductDesc: ${requestScope.product.productDesc }  <br><br>   ProductPrice: ${requestScope.product.productPrice }  <br><br> </body> </html> |

### Product

|  |
| --- |
| package com.atguigu.struts2.helloworld; public class Product {  private Integer productId;  private String productName;  private String productDesc;   private double productPrice;   public Integer getProductId() {  return productId;  }   public void setProductId(Integer productId) {  this.productId = productId;  }   public String getProductName() {  return productName;  }   public void setProductName(String productName) {  this.productName = productName;  }   public String getProductDesc() {  return productDesc;  }   public void setProductDesc(String productDesc) {  this.productDesc = productDesc;  }   public double getProductPrice() {  return productPrice;  }   public void setProductPrice(double productPrice) {  this.productPrice = productPrice;  }   public Product(Integer productId, String productName, String productDesc, double productPrice) {  this.productId = productId;  this.productName = productName;  this.productDesc = productDesc;  this.productPrice = productPrice;  }   public Product() {  }   @Override  public String toString() {  return "Product [productId=" + productId + ", productName="  + productName + ", productDesc=" + productDesc  + ", productPrice=" + productPrice + "]";  } } |

### FilterDispatcher

|  |
| --- |
| package com.atguigu.struts2.helloworld;  import java.io.IOException; import javax.servlet.Filter; import javax.servlet.FilterChain; import javax.servlet.FilterConfig; import javax.servlet.ServletException; import javax.servlet.ServletRequest; import javax.servlet.ServletResponse; import javax.servlet.http.HttpServletRequest;  public class FilterDispatcher implements Filter {   public void destroy() {}   public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {   HttpServletRequest req = (HttpServletRequest) request;   //1. 获取 servletPath  String servletPath = req.getServletPath();  System.***out***.println(servletPath);   String path = null;   //2. 判断 servletPath, 若其等于 "/product-input.action", 则转发到  ///WEB-INF/pages/input.jsp  if("/product-input.action".equals(servletPath)){  path = "/WEB-INF/pages/input.jsp";  }   //3. 若其等于 "/product-save.action", 则  if("/product-save.action".equals(servletPath)){  //1). 获取请求参数  String productName = request.getParameter("productName");  String productDesc = request.getParameter("productDesc");  String productPrice = request.getParameter("productPrice");   //2). 把请求信息封装为一个 Product 对象  Product product = new Product(null, productName, productDesc, Double.parseDouble(productPrice));   //3). 执行保存操作  System.***out***.println("Save Product: " + product);  product.setProductId(1001);   //4). 把 Product 对象保存到 request 中. ${param.productName} -> ${requestScope.product.productName}  request.setAttribute("product", product);   path = "/WEB-INF/pages/details.jsp";  }   if(path != null){  request.getRequestDispatcher(path).forward(request, response);  return;  }   chain.doFilter(request, response);  }   public void init(FilterConfig fConfig) throws ServletException {}  } |

# Hello Strtus2

## Struts2 概述

|  |
| --- |
|  |

## Struts2 VS Struts1

|  |
| --- |
|  |

## 从 Struts1 升级到 Struts2

|  |
| --- |
|  |

## 下载 Struts2

|  |
| --- |
| 打开浏览器输入 <http://struts.apache.org/>  点击超链接 “Struts 2.3.x”, 打开下载页面    点击 “struts-2.3.x-all.zip” 下载 |

## Struts2 的 Hello World

|  |
| --- |
|  |

## 添加 DTD 约束

|  |
| --- |
|  |

## Struts2 的 Hello World

|  |
| --- |
|  |

## note

|  |
| --- |
| 1. VS 自实现:  1). 搭建 Struts2 的开发环境  2). 不需要显式的定义 Filter, 而使用的是 struts2 的配置文件.  3). details.jsp 比先前变得简单了.  ${requestScope.product.productName} -> ${productName}  4). 步骤:  I. 由 product-input.action 转到 /WEB-INF/pages/input.jsp  在 struts2 中配置一个 action  <action name="product-input">  <result>/WEB-INF/pages/input.jsp</result> </action>  II. 由 input.jsp 页面的 action: product-save.action 到 Product's save, 再到 /WEB-INF/pages/details.jsp  <action name="product-save" class="com.atguigu.struts2.helloworld.Product"  method="save">  <result name="details">/WEB-INF/pages/details.jsp</result> </action>  在 Prodcut 中定义一个 save 方法, 且返回值为 detail |

## 代码

|  |
| --- |
|  |

### index.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%> <html>  <head>  <title></title>  </head>  <body>  <a href="product-input.action">Product Input</a>  <br><br>  <a href="test.action">Test</a>  </body> </html> |

### web.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"  version="4.0">   <!-- 配置 Struts2 的 Filter -->  <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> </web-app> |

### input.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%> <html>  <head>  <title></title>  </head>  <body>  <form action="product-save.action" method="post">   ProductName: <input type="text" name="productName"/>  <br><br>   ProductDesc: <input type="text" name="productDesc"/>  <br><br>   ProductPrice: <input type="text" name="productPrice" />  <br><br>   <input type="submit" value="Submit"/>  <br><br>   </form>  </body> </html> |

### details.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%> <%@ taglib prefix="s" uri="/struts-tags" %> <html>  <head>  <title></title>  </head>  <body>  <%  request.setCharacterEncoding("UTF-8");  %>   ProductId: ${productId }  <br><br>   ProductName: <%= request.getAttribute("productName") %>  <br><br>   ProductDesc: ${productDesc }  <br><br>   ProductPrice: ${productPrice }  <br><br>   ProductPrice: <s:property value="productPrice"/>  <br><br>   <%= request %>  </body> </html> |

### Product

|  |
| --- |
| package com.atguigu.struts2.helloworld;  public class Product {   private Integer productId;  private String productName;  private String productDesc;   private double productPrice;   public Integer getProductId() {  return productId;  }   public void setProductId(Integer productId) {  this.productId = productId;  }   public String getProductName() {  return productName;  }   public void setProductName(String productName) {  this.productName = productName;  }   public String getProductDesc() {  return productDesc;  }   public void setProductDesc(String productDesc) {  this.productDesc = productDesc;  }   public double getProductPrice() {  return productPrice;  }   public void setProductPrice(double productPrice) {  this.productPrice = productPrice;  }   @Override  public String toString() {  return "Product [productId=" + productId + ", productName="  + productName + ", productDesc=" + productDesc  + ", productPrice=" + productPrice + "]";  }   public String save(){  System.***out***.println("save: " + this);  return "details";  }   public String test(){  System.***out***.println("test");  return "success";  }   public Product() {  System.***out***.println("Product's constructor...");  } } |

### 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>   <!--   package: 包. struts2 使用 package 来组织模块.   name 属性: 必须. 用于其它的包应用当前包.   extends: 当前包继承哪个包, 继承的, 即可以继承其中的所有的配置.  通常情况下继承 struts-default  struts-default 这个包在 struts-default.xml 文件中定义.  namespace 可选, 如果它没有给出, 则以 / 为默认值.   若 namespace 有一个非默认值, 则要想调用这个包里的Action,  就必须把这个属性所定义的命名空间添加到有关的 URI 字符串里    http://localhost:8080/contextPath/namespace/actionName.action  -->  <package name="helloWorld" extends="struts-default">    <!--   配置一个 action: 一个 struts2 的请求就是一个 action   name: 对应一个 struts2 的请求的名字(或对一个 servletPath, 但去除 / 和扩展名), 不包含扩展名  class 的默认值为: com.opensymphony.xwork2.ActionSupport  method 的默认值为: execute  result: 结果.   -->  <action name="product-input"   class="com.opensymphony.xwork2.ActionSupport"  method="execute">  <!--   result: 结果. 表示 action 方法执行后可能返回的一个结果. 所以一个 action 节点可能会有多个 result 子节点.  多个 result 子节点使用 name 来区分  name: 标识一个 result. 和 action 方法的返回值对应. 默认值为 success  type: 表示结果的类型. 默认值为 dispatcher(转发到结果.)  -->  <result name="success" type="dispatcher">/WEB-INF/pages/input.jsp</result>  </action>    <action name="product-save"  class="com.atguigu.struts2.helloworld.Product"  method="save">  <result name="details">/WEB-INF/pages/details.jsp</result>   </action>    <action name="test"  class="com.atguigu.struts2.helloworld.Product"  method="test">  <result>/index.jsp</result>  </action>    </package> </struts> |

# Action 类

|  |
| --- |
| action: 应用程序可以完成的每一个操作. 例如: 显示一个登陆表单; 把产品信息保存起来  Action类: 普通的 Java 类, 可以有属性和方法, 同时必须遵守下面这些规则:  1.属性的名字必须遵守与 JavaBeans 属性名相同的命名规则. 属性的类型可以是任意类型. 从字符串到非字符串(基本数据库类型)之间的数据转换可以自动发生  2.必须有一个不带参的构造器  3.至少有一个供 struts 在执行这个 action 时调用的方法  4.同一个 Action 类可以包含多个 action 方法.  5.Struts2 会为每一个 HTTP 请求创建一个新的 Action 实例 |

## 访问 web 资源

|  |
| --- |
|  |

## 与Servlet API解耦的访问方式

|  |
| --- |
|  |

## 通过 ActionContext 访问 Web 资源

|  |
| --- |
|  |

### 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>   <!-- 配置 Struts 可以受理的请求的扩展名 -->  <constant name="struts.action.extension" value="action,do,"></constant>    <!-- 打开允许动态方法调用的开关, 默认是 false -->  <constant name="struts.enable.DynamicMethodInvocation" value="true"></constant>    <package name="default" namespace="/" extends="struts-default">   <action name="TestActionContext"   class="com.atguigu.struts2.action.TestActionContextAction">  <result name="success">/test-actionContext.jsp</result>   </action>  </package> </struts> |

### index.jsp

|  |
| --- |
| <a href="TestActionContext.action?name=atguigu">Test ActionContext</a> |

### test-actionContext.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%> <html> <head> <title></title> </head> <body>    <h4>Test ActionContext Page</h4>    application : ${applicationScope.applicationKey }  <br><br>    session: ${sessionScope.sessionKey }  <br><br>    request: ${requestScope.requestKey }  <br><br>    age: ${parameters.age }   </body> </html> |

### TestActionContextAction

|  |
| --- |
| package com.atguigu.struts2.action;  import java.util.Map;  import org.apache.struts2.dispatcher.SessionMap;  import com.opensymphony.xwork2.ActionContext;  public class TestActionContextAction {   public String execute(){    //0. 获取 ActionContext 对象  //ActionContext 是 Action 的上下文对象. 可以从中获取到当前 Action 需要的一切信息  ActionContext actionContext = ActionContext.getContext();    //1. 获取 application 对应的 Map, 并向其中添加一个属性  //通过调用 ActionContext 对象的 getApplication() 方法来获取 application 对象的 Map 对象  Map<String, Object> applicationMap = actionContext.getApplication();  //设置属性  applicationMap.put("applicationKey", "applicationValue");    //获取属性  Object date = applicationMap.get("date");  System.***out***.println("date: " + date);    //2. session  Map<String, Object> sessionMap = actionContext.getSession();  sessionMap.put("sessionKey", "sessionValue");    System.***out***.println(sessionMap.getClass());     if(sessionMap instanceof SessionMap){  SessionMap sm = (SessionMap) sessionMap;  sm.invalidate();  System.***out***.println("session 失效了. ");  }    //3. request\*   //ActionContext 中并没有提供 getRequest 方法来获取 request 对应的 Map  //需要手工调用 get() 方法, 传入 request 字符串来获取.   Map<String, Object> requestMap = (Map<String, Object>) actionContext.get("request");  requestMap.put("requestKey", "requestValue");    //4. 获取请求参数对应的 Map, 并获取指定的参数值.   //键: 请求参数的名字, 值: 请求参数的值对应的字符串数组  //注意: 1. getParameters 的返回值为在 Map<String, Object>, 而不是 Map<String, String[]>  // 2. parameters 这个 Map 只能读, 不能写入数据, 如果写入, 但不出错, 但也不起作用!  Map<String, Object> parameters = actionContext.getParameters();  System.***out***.println(((String[])parameters.get("name"))[0]);    parameters.put("age", 100);    return "success";  } } |

## 通过实现 Aware 接口访问 Web 资源

|  |
| --- |
| Action 类通过可以实现某些特定的接口, 让 Struts2 框架在运行时向 Action 实例注入 parameters, request, session 和 application 对应的 Map 对象: |

### 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>   <!-- 配置 Struts 可以受理的请求的扩展名 -->  <constant name="struts.action.extension" value="action,do,"></constant>    <!-- 打开允许动态方法调用的开关, 默认是 false -->  <constant name="struts.enable.DynamicMethodInvocation" value="true"></constant>    <package name="default" namespace="/" extends="struts-default">  <action name="TestAware"   class="com.atguigu.struts2.action.TestAwareAction">  <result>/test-aware.jsp</result>   </action>  </package> </struts> |

### index.jsp

|  |
| --- |
| <a href="TestAware.do?name=atguigu">Test Aware</a> |

### test-aware.jsp

|  |
| --- |
| <h4>Test Aware Page</h4> application: ${applicationScope.applicationKey2 } |

### TestAwareAction

|  |
| --- |
| package com.atguigu.struts2.action;  import java.util.Map;  import org.apache.struts2.interceptor.ApplicationAware; import org.apache.struts2.interceptor.ParameterAware; import org.apache.struts2.interceptor.RequestAware; import org.apache.struts2.interceptor.SessionAware;  public class TestAwareAction implements ApplicationAware, SessionAware, RequestAware,  ParameterAware{    public String execute(){    //1. 向 application 中加入一个属性: applicationKey2 - applicationValue2  application.put("applicationKey2", "applicationValue2");    //2. 从 application 中读取一个属性 date, 并打印.   System.***out***.println(application.get("date"));    return "success";  }    public String save(){  return null;  }   private Map<String, Object> application;    @Override  public void setApplication(Map<String, Object> application) {  this.application = application;  }   @Override  public void setParameters(Map<String, String[]> parameters) {    }   @Override  public void setRequest(Map<String, Object> request) {   }   @Override  public void setSession(Map<String, Object> session) {   } } |

## 与 Servlet 耦合的访问方式

|  |
| --- |
| 直接访问 Servlet API 将使 Action 与 Servlet 环境耦合在一起, 测试时需要有 Servlet 容器, 不便于对 Action 的单元测试.  直接获取 HttpServletRequest 对象: ServletActionContext.getRequest()  直接获取 HttpSession 对象：ServletActionContext.getRequest().getSession()  直接获取 ServletContext 对象：ServletActionContext.getServletContext()  通过实现 ServletRequestAware, ServletContextAware 等接口的方式 |
|  |

## ActionSupport

|  |
| --- |
| **com.opensymphony.xwork2.ActionSupport 类是默认的 Action 类.**  **在编写 Action 类时, 通常会对这个类进行扩展** |

## 练习

|  |
| --- |
|  |

## note

|  |
| --- |
|  |

# result

|  |
| --- |
| 每个 action 方法都将返回一个 String 类型的值, Struts 将根据这个值来决定响应什么结果.  每个 action 声明都必须包含有数量足够多的 result 元素, 每个 result 元素分别对应着 action 方法的一个返回值.  result 元素可以有下面两个属性  name: 结果的名字, 必须与 Action 方法的返回值相匹配, 默认值为 success  type: 响应结果的类型. 默认值为 dispatcher |

## 结果类型

|  |
| --- |
|  |

## 结果类型: dispatcher

|  |
| --- |
| dispatcher 结果类型是最常用的结果类型, 也是 struts 框架默认的结果类型  该结果类型有一个 location 参数, 它是一个默认参数    dispatcher 结果类型将把控制权转发给应用程序里的指定资源.  dispatcher 结果类型不能把控制权转发给一个外部资源. 若需要把控制权重定向到一个外部资源, 应该使用 redirect 结果类型 |

## 结果类型: redirect

|  |
| --- |
| redirect 结果类型将把响应重定向到另一个资源, 而不是转发给该资源.  redirect 结果类型接受下面这些参数:  -location: 用来给出重定向的目的地.它是默认属性  -parse: 用来表明是否把 location 参数的值视为一个 OGNL 表达式来解释.  默认值为 true  redirect 结果类型可以把响应重定向到一个外部资源  实例代码: |

## 结果类型: redirectAction

|  |
| --- |
| redirectAction 结果类型把响应重定向到另一个 Action  redirectAction 结果类型接受下面这些参数:  -actionName: 指定 “目的地” action 的名字. 它是默认属性  -namespace: 用来指定 “目的地” action 的命名空间. 如果没有配置该参数,  Struts 会把当前 Action 所在的命名空间作为 “目的地” 的命名空间  示例代码: |

## 结果类型: chain

|  |
| --- |
| chain 结果类型的基本用途是构成一个 action 链: 前一个 action 把控制权转发给后一个 action, 而前一个 action 的状态在后一个 action 中依然保持  chain 结果类型接受下面这些参数:  -actionName: 指定目标 action 的名字. 它是默认属性  -namespace: 用来指定 “目的地” action 的命名空间. 如果没有配置该参数, Struts 会把当前 action 所在的命名空间作为 “目的地” 的命名空间  -method: 指定目标 action 方法. 默认值为 execute |

## 通配符映射

|  |
| --- |
| 一个 Web 应用可能有成百上千个 action 声明. 可以利用 struts 提供的通配符映射机制把多个彼此相似的映射关系简化为一个映射关系  通配符映射规则  - 若找到多个匹配, 没有通配符的那个将胜出  - 若指定的动作不存在, Struts 将会尝试把这个 URI 与任何一个包含着通配符 \* 的动作名及进行匹配  - 被通配符匹配到的 URI 字符串的子串可以用 {1}, {2} 来引用. {1} 匹配第一个子串, {2} 匹配第二个子串…  - { 0} 匹配整个 URI  - 若 Struts 找到的带有通配符的匹配不止一个, 则按先后顺序进行匹配  - \* 可以匹配零个或多个字符, 但不包括 / 字符. 如果想把 / 字符包括在内,  需要使用 \*\*. 如果需要对某个字符进行转义, 需要使用 \.  当匹配的Action有两个以上时，则会按匹配精确度高的那个Action，当有个相同的匹配精确度时，则按先后顺序进行。 |

## 通配符映射示例(1)

|  |
| --- |
| 包声明:    上面的包声明可以由正确的命名空间和\_add 组成的 URI 来调用, 包括: |

## 通配符映射示例(2)

|  |
| --- |
| 包声明:    上面的包可改写为: |

## 通配符映射示例(3)

|  |
| --- |
| 包声明:    上面的包可改写为: |

## 动态方法调用

|  |
| --- |
| 动态方法调用: 通过 url 动态调用 Action 中的方法  action 声明:    URI:  - /struts-app2/Product.action: Struts 调用 Product 类的 execute  - /struts-app2/Product!save.action: Struts 调用 Product 类的 save() 方法  默认情况下, Struts 的动态方法调用处于禁用状态  struts.xml  <!-- 打开允许动态方法调用的开关, 默认是 false --> <constant name="struts.enable.DynamicMethodInvocation" value="true"></constant>  <action name="testDynamicMethodInvocation"  class="com.atguigu.struts2.action.TestDynamicMethodInvocationActoin"  method="save">  <result>/success.jsp</result>  </action>  TestDynamicMethodInvocationActoin  package com.atguigu.struts2.action;  public class TestDynamicMethodInvocationActoin {   public String save(){  System.***out***.println("save...");  return "success";  }    public String update(){  System.***out***.println("update...");  return "success";  } } |

# OGNL

## 从页面显示说起

|  |
| --- |
|  |

## 值 栈

|  |
| --- |
|  |

## OGNL

|  |
| --- |
|  |

## property 标签

|  |
| --- |
| Struts 的 property 标签用来输出值栈中的一个属性值. |

## 读取 ObjectStack 里的对象的属性

|  |
| --- |
|  |

## 读取 Context Map 里的对象的属性

|  |
| --- |
|  |

## 调用字段和方法

|  |
| --- |
|  |

## 访问数组类型的属性

|  |
| --- |
|  |

## 访问 List 类型的属性

|  |
| --- |
|  |

## 访问 Map 类型的属性

|  |
| --- |
|  |

## 使用 EL 访问值栈中对象的属性

|  |
| --- |
| <s:property value=“fieldName”> 也可以通过 JSP EL 来达到目的: ${fieldName}  原理: Struts2 将包装 HttpServletRequest 对象后的  org.apache.struts2.dispatcher.StrutsRequestWrapper 对象传到页面上,  而这个类重写了 getAttribute() 方法. |

## 异常处理: exception-mapping 元素

|  |
| --- |
|  |

# 通用标签

## property 标签

|  |
| --- |
| property 标签用来输出一个值栈属性的值    示例:  - 输出 Action 属性 customerId 的值: <s:property value=“customerId”/>  - 输出 session 属性 userName 的值: <s:property value=“#session.userName”/>  如果 value 属性没有给出, ValueStack 值栈栈顶对象的值被输出  在许多情况下, JSP EL 可以提供更简洁的语法 |

## url 标签

|  |
| --- |
| url 标签用来动态地创建一个 URL    Portlet是基于java的web组件，由portlet容器管理，并由容器处理请求，生产动态内容。Portals使用portlets作为可插拔用户接口组件，提供信息系统的表示层。作为利用servlets进行web应用编程的下一步，portlets实现了web应用的模块化和用户中心化。 portlet规范，即jsr（Java Standardization Request ）168，是为了实现portal和portlet的互操作。它定义了portlet和portlet容器之间的和约，让portlet实现个性化、表示和安全的api集。规范还定义了怎样在portlets应用中打包portlets。 |

## param 标签

|  |
| --- |
| param 标签用来把一个参数传递给包含着它的那个标签    1.无论在给出 value 值时有没有使用 %{}, Struts 都会对它进行 ognl 求值  2.如果想传递一个 String 类型的字符串作为参数值, 必须把它用单引号括起来.  3.可以把 value 属性的值写在开始标签和结束标签之间. 利用这种方式来传递一个 EL 表达式的值 |

## set 标签

|  |
| --- |
| set 标签用来在以下 Map 对象里创建一个键值对:  -ValueStack 值栈的 ContextMap 值栈  -Map 类型的 session 对象  -Map 类型的 application 对象  -Map 类型的 request 对象  -Map 类型的 page 对象 |

## push 标签

|  |
| --- |
| push 标签的功能和 set 标签类似.  push 标签将把一个对象压入 ValueStack 而不是压入 ContextMap.  push 标签在标签起始时把一个对象压入栈, 标签结束时将对象弹出栈. |

## if, else 和 elseif 标签

|  |
| --- |
| 这三个标签用来进行条件测试, 它们的用途和用法类似于 if, else 和 elseif 关键字.  其中 if 和 elseif 必须有 test 属性 |

## iterator 标签

|  |
| --- |
| iterator 标签用来遍历一个数组, Collection 或一个 Map, 并把这个可遍历对象里的每一个元素依次压入和弹出 ValueStack 栈    在开始执行时, iterator 标签会先把 IteratorStatus 类的一个实例压入 ContextMap, 并在每次遍历循环时更新它. 可以将一个指向 IteratorStatus 对象的变量赋给 status 属性.  iterator 标签的 status 属性的属性值 |

## sort 标签

|  |
| --- |
| sort 标签用来对一个可遍历对象里的元素进行排序. |

## date 标签

|  |
| --- |
| date 标签用来对 Date 对象进行排版    format 属性的值必须是 java.text.SimpleDateFormat 类里定义的日期/时间格式之一. |

## a 标签

|  |
| --- |
| a 标签将呈现为一个 HTML 连接. 这个标签可以接受 HTML 语言中的 a 元素所能接受的所有属性. |

## action 标签

|  |
| --- |
| action 标签用在页面上来执行一个 action.  action 标签还会把当前 Action 对象压入 ValueStack 值栈的 ContextMap 子栈. |

## bean 标签

|  |
| --- |
| bean 标签将创建一个 JavaBean, 并把它压入 ValueStack 值栈的 ContextMap 子栈. 这个标签的功能与 JSP 中的 useBean 动作元素很相似 |

## include 标签

|  |
| --- |
| include 标签用来把一个 Servlet 或 JSP 页面的输出包含到当前页面里来. |

## append, merge 标签

|  |
| --- |
| append 标签用来合并可遍历对象.  merge 标签用来交替合并可遍历对象.    示例: |

## generator 标签

|  |
| --- |
| generator 标签用来生成一个可遍历对象并把它压入 ValueStack 栈.  generator 标签结束标记将弹出遍历对象    如果在一个 generator 标签里给出了 converter 属性, 新生成的可遍历对象里的每一个元素都会传递到该属性所指定的方法进行必要的转换.  示例. |

## subset 标签

|  |
| --- |
| subset 标签用来创建一个可遍历集合的子集.  subset 标签通过 decider 属性来创建一个可遍历集合的子集.    示例: |

# 表单标签

## 概述

|  |
| --- |
| 表单标签将在 HTML 文档里被呈现为一个表单元素  使用表单标签的优点:  -表单回显  -对页面进行布局和排版  标签的属性可以被赋值为一个静态的值或一个 OGNL 表达式. 如果在赋值时使用了一个 OGNL 表达式并把它用 %{} 括起来, 这个表达式将会被求值. |

## 表单标签的共同属性

|  |
| --- |
| \* 该属性只在没有使用 simple 主题时才可以使用. |

## form 标签

|  |
| --- |
| form 标签用来呈现 HTML 语言中的表单元素    默认情况下, form 标签将被呈现为一个表格形式的 HTML 表单. 嵌套在 form 标签里的输入字段将被呈现为一个表格行. 每个表格行由两个字段组成, 一个对应着行标, 一个对应着输入元素. 提交按钮将被呈现为一个横跨两列单元格的行 |

## textfield, password, hidden 标签

|  |
| --- |
| textfield 标签将被呈现为一个输入文本字段, password 标签将被呈现为一个口令字段, hidden 标签将被呈现为一个不可见字段.    password 标签扩展自 textfield 标签, 多了一个 showPassword 属性. 该属性时布尔型. 默认值为 false, 它决定着在表单回显时是否显示输入的密码. |

## submit 标签

|  |
| --- |
| submit 标签将呈现为一个提交按钮. 根据其 type 属性的值. 这个标签可以提供 3 种呈现效果:  input: <input type=“submim” …/>  button: <input type=“button” …/>  image: <input type=“image” /> |

## textarea 标签

|  |
| --- |
| textarea 标签将呈现为一个 HTML 文本域元素 |

## checkbox 标签

|  |
| --- |
| 1.checkbox 标签将呈现为一个 HTML 复选框元素. 该复选框元素通常用于提交一个布尔值  2.当包含着一个复选框的表单被提交时, 如果某个复选框被选中了, 它的值将为 true, 这个复选框在 HTTP 请求里增加一个请求参数.但如果该复选框未被选中, 在请求中就不会增加一个请求参数.  3.checkbox 标签解决了这个局限性, 它采取的办法是为单个复选框元素创建一个配对的不可见字段        checkbox 标签有一个 fieldValue 属性, 该属性指定的值将在用户提交表单时作为被选中的单选框的实际值发送到服务器. 如果没有使用 fieldValue 属性, 单选框的值将为 true 或 false.    示例: |

## list, listKey 和 listValue 属性

|  |
| --- |
| list, listKey, listValue 这 3 个属性对 radio, select, checkboxlist 等标签非常重要    可以把一个 String, 一个数组, 一个 Enumeration, Iterator, Map 或 Collection 赋给 list 属性.    赋值为一个Map:      赋值为一个 Collection(或一个对象数组): 把数组或 Collection 赋值给 list 属性, 把用来提供选项值的对象属性赋给 listKey 属性, 把用来提供选项行标的对象属性赋给 listValue 属性 |

## radio 标签

|  |
| --- |
| radio 标签将呈现为一组单选按钮, 单选按钮的个数与程序员通过该标签的 list 属性提供的选项的个数相同.  一般地, 使用 radio 标签实现 “多选一”, 对于 “真/假” 则该使用 checkbox 标签.    示例: |

## select 标签

|  |
| --- |
| select 标签将呈现一个 select 元素.    示例: |

## optiongroup 标签

|  |
| --- |
| optiongroup 标签对 select 元素所提供的选项进行分组. 每个选项有它自己的来源.    示例: |

## checkboxlist 标签

|  |
| --- |
| checkboxlist 标签将呈现一组多选框.    checkbox 标签被映射到一个字符串数组或是一个基本类型的数组. 若它提供的多选框一个也没有被选中, 相应的属性将被赋值为一个空数组而不是空值.  示例: |

# 主题

|  |
| --- |
| 默认情况下, form 标签将呈现为一个 HTML form 元素和一个 table 元素.    每一种输入标签都将呈现为一个带标号的输入元素, 而这个输入元素将被包含在一个 tr 元素和 td 元素的内部 |

## 示例代码

|  |
| --- |
|  |

# ModelDriven 和 Preparable 拦截器

## 示例代码

|  |
| --- |
|  |

## Params 拦截器

|  |
| --- |
| Parameters 拦截器将把表单字段映射到 ValueStack 栈的栈顶对象的各个属性中.  如果某个字段在模型里没有匹配的属性, Param 拦截器将尝试 ValueStack 栈中的下一个对象 |

## 把 Action 和 Model 隔开

|  |
| --- |
| 在使用 Struts 作为前端的企业级应用程序时把 Action 和 Model 清晰地隔离开是有必要的: 有些 Action 类不代表任何Model 对象, 它们的功能仅限于提供显示服务 |

## ModelDriven 拦截器

|  |
| --- |
| 当用户触发 add 请求时, ModelDriven 拦截器将调用 EmployeeAction 对象的 getModel() 方法, 并把返回的模型(Employee实例)压入到 ValueStack 栈.  接下来 Parameters 拦截器将把表单字段映射到 ValueStack 栈的栈顶对象的各个属性中. 因为此时 ValueStack 栈的栈顶元素是刚被压入的模型(Employee)对象, 所以该模型将被填充. 如果某个字段在模型里没有匹配的属性, Param 拦截器将尝试 ValueStack 栈中的下一个对象 |

## Preparable 拦截器

|  |
| --- |
| Struts 2.0 中的 modelDriven 拦截器负责把 Action 类以外的一个对象压入到值栈栈顶  而 prepare 拦截器负责准备为 getModel() 方法准备 model |

## PrepareInterceptor拦截器用方法

|  |
| --- |
|  |

## 使用 paramsPrepareParamsStack 拦截器栈

|  |
| --- |
|  |

# 类型转换

|  |
| --- |
| 1.类型转换概述  2.类型转换出错时如何进行处理  - 转到哪个页面  - 显示什么错误消息  3.自定义类型转换器  **4.类型转换与复杂对象配合使用** |

## 概述

|  |
| --- |
| 1.从一个 HTML 表单到一个 Action 对象, 类型转换是从字符串到非字符串.  HTTP 没有 “类型” 的概念. 每一项表单输入只可能是一个字符串或一个字符串数组. 在服务器端, 必须把 String 转换为特定的数据类型  2.在 struts2 中, 把请求参数映射到 action 属性的工作由 Parameters 拦截器负责, 它是默认的 defaultStack 拦截器中的一员.  Parameters 拦截器可以自动完成字符串和基本数据类型之间转换. |

## 类型转换错误

|  |
| --- |
| **如果类型转换失败:**  1.若 Action 类没有实现 ValidationAware 接口： Struts 在遇到类型转换错误时仍会继续调用其 Action 方法, 就好像什么都没发生一样.  2.若 Action 类实现 ValidationAware 接口：Struts 在遇到类型转换错误时将不会继续调用其 Action 方法: Struts 将检查相关 action 元素的声明是否包含着一个 name=input 的 result. 如果有, Struts 将把控制权转交给那个 result 元素; 若没有 input 结果, Struts 将抛出一个异常 |

## 类型转换错误消息的定制

|  |
| --- |
| 1. 作为默认的 default 拦截器的一员, ConversionError 拦截器负责添加与类型转换有关的出错消息(前提: Action 类必须实现了 ValidationAware 接口)和保存各请求参数的原始值.   2.若字段标签使用的不是 simple 主题, 则非法输入字段将导致一条有着以下格式的出错消息:  3.覆盖默认的出错消息    4.定制出错消息的样式:  每一条出错消息都被打包在一个 HTML span 元素里, 可以通过覆盖其行标为 errorMessage 的那个 css 样式来改变出错消息的格式.  5.显示错误消息: 如果是 simple 主题, 可以通过  <s:fielderror fieldName=“filedname”></s:fielderror> 标签显示错误消息 |

## 定制类型转换器

|  |
| --- |
| 自定义类型转换器必须实现 ongl.TypeConverter 接口或对这个接口的某种具体实现做扩展 |

## 扩展 StrutsTypeConverter 类

|  |
| --- |
| 1.在大多数类型转换器里, 需要提供从 String 类型到非 String 类型和与此相反的转换功能  2.在 StrutsTypeConverter 中有两个抽象方法: |

## 配置自定义的类型转换器

|  |
| --- |
| 1.在应用程序里使用一个自定义的类型转换器之前, 必须先对它进行配置. 这种配置既可以基于字段, 也可以基于类型  基于字段配置: 可以为某个 Model（该 Model 类也可能是 Action） 的各个属性分别配置一个自定义的转换器.    基于类型配置:  - 在 WEB-INF/classes/ 目录下创建 xwork-conversion.properties 文件.  - 在 xwork-conversion.properties 文件里把每一个需要进行类型转换的类与一个类型转换器关联起来 |

## 示例代码

|  |
| --- |
| 实现自定义的时间类型转换器: 时间 pattern 需要以 web 应用的初始化参数配置在 web.xml 中    若类型转换失败，给出自定义的信息。 |

## 类型转换与复杂属性配合使用

|  |
| --- |
|  |

## 类型转换与 Collection 配合使用

|  |
| --- |
|  |

# 消息处理与国际化

## 概述

|  |
| --- |
|  |

## 配置国际化资源文件

|  |
| --- |
| 1. Action 范围资源文件：在Action类文件所在的路径建立名为ActionName\_language\_country.properties 的文件   2.包范围资源文件：在包的根路径下建立文件名为package\_language\_country.properties 的属性文件，一旦建立，处于该包下的所有 Action 都可以访问该资源文件。  注意：包范围资源文件的 baseName 就是package，不是Action所在的包名。  3.全局资源文件    4.临时指定资源文件：<s:i18n.../> 标签的 name 属性指定临时的国际化资源文件 |

## 加载资源文件的顺序

|  |
| --- |
|  |

## 访问国际化消息

|  |
| --- |
|  |

## 利用超链接实现动态加载国际化资源文件

|  |
| --- |
| 1. Struts2 使用 i18n 拦截器 处理国际化，并且将其注册在默认的拦截器中   2. i18n拦截器在执行Action方法前，自动查找请求中一个名为request\_locale 的参数。如果该参数存在，拦截器就将其作为参数，转换成Locale对象，并将其设为用户默认的Locale(代表国家/语言环境)。并把其设置为 session 的 WW\_TRANS\_I18N\_LOCALE 属性  3.若 request 没有名为request\_locale 的参数，则 i18n 拦截器会从 Session 中获取 WW\_TRANS\_I18N\_LOCALE 的属性值，若该值不为空，则将该属性值设置为浏览者的默认Locale  4.若 session 中的 WW\_TRANS\_I18N\_LOCALE 的属性值为空，则从 ActionContext 中获取 Locale 对象。 |

# Struts2 运行流程分析

|  |
| --- |
|  |

## 相关的几个 API

|  |
| --- |
|  |

## Struts2 运行流程分析

|  |
| --- |
|  |

# 输入验证

## 概述

|  |
| --- |
| 一个健壮的 web 应用程序必须确保用户输入是合法、有效的.  Struts2 的输入验证  -基于 XWork Validation Framework 的声明式验证：Struts2 提供了一些基于 XWork Validation Framework 的内建验证程序. 使用这些验证程序不需要编程, 只要在一个 XML 文件里对验证程序应该如何工作作出声明就可以了. 需要声明的内容包括:  **a.哪些字段需要进行验证**  **b.使用什么验证规则**  **c.在验证失败时应该把什么样的出错消息发送到浏览器端**    -编程验证：通过编写代码来验证用户输入 |

## 声明式验证

|  |
| --- |
| - 声明式验证程序可以分为两类:  字段验证: 判断某个字段属性的输入是否有效  非字段验证: 不只针对某个字段，而是针对多个字段的输入值之间的逻辑关系进行校验。例如：对再次输入密码的判断。  - 使用一个声明式验证程序需要 3 个步骤:  1. 确定哪些 Action 字段需要验证  2. 编写一个验证程序配置文件. 它的文件名必须是以下两种格式之一:  **若一个 Action 类的多个 action 使用同样的验证规则: ActionClassName-validation.xml**  **若一个 Action 类的多个 action 使用不同的验证规则: ActionClass-alias-validation.xml, 例如 UserAction-User\_create-validation.xm**  3. 确定验证失败时的响应页面: 在 struts.xml 文件中定义一个 <result name=“input”> 的元素. |

## Struts2 内建的验证规则

|  |
| --- |
|  |

## 验证程序的配置

|  |
| --- |
|  |

## Struts2 声明式验证原理解析

|  |
| --- |
|  |

## Struts2 的验证规则和验证器

|  |
| --- |
| **每个验证规则都对应一个具体的验证器** |

## 配置文件与验证器属性

|  |
| --- |
|  |

## Struts2 内建的验证程序

|  |
| --- |
| required: 确保某给定字段的值不是空值 null “”  requiredstring: 确保某给定字段的值既不是空值 null, 也不是空白.  trim 参数. 默认为 true, 表示 struts 在验证该字段值之前先剔除前后空格.  stringlength: 验证一个非空的字段值是不是有足够的长度.  minLength: 相关字段的最小长度. 若没有给出这个参数, 该字段将没有最小长度限制  maxLength:相关字段的最大长度. 若没有给出这个参数, 该字段将没有最大长度限制  trim: 在验证之前是否去除前后空格  date: 确保某给定日期字段的值落在一个给定的范围内  max:相关字段的最大值. 若没给出这个参数, 该字段将没有最大值限制  min:相关字段的最小值. 若没给出这个参数, 该字段将没有最小值限制  email: 检查给定 String 值是否是一个合法的 email  url: 检查给定 String 值是否是一个合法的 url  regex: 检查某给定字段的值是否与一个给定的正则表达式模式相匹配.  expresssion\*: 用来匹配的正则表达式  caseSensitive: 是否区分字母的大小写. 默认为 true  trim: 是否去除前后空格. 默认为 true  int: 检查给定整数字段值是否在某一个范围内  min: 相关字段的最小值. 若没给出这个参数, 该字段将没有最小值限制  max: 相关字段的最大值. 若没给出这个参数, 该字段将没有最大值限制  conversion: 检查对给定 Action 属性进行的类型转换是否会导致一个转换错误. 该验证程序还可以在默认的类型转换消息的基础上添加一条自定义的消息  expression 和 fieldexpression: 用来验证给定字段是否满足一个 OGNL 表达式.  前者是一个非字段验证程序, 后者是一个字段验证程序.  前者在验证失败时将生成一个 action 错误, 而后者在验证失败时会生成一个字段错误  expression\*: 用来进行验证的 OGNL 表达式 |

## 短路验证器

|  |
| --- |
| <validator …/> 元素和 <field-validator …/> 元素可以指定一个可选的 short-circuit 属性，该属性指定该验证器是否是短验证器，默认值为 false。  对同一个字段内的多个验证器，如果一个短路验证器验证失败，其他验证器不会继续校验 |

## 非字段验证示例

|  |
| --- |
|  |

## 字段验证 vs 非字段验证

|  |
| --- |
| 1.字段验证字段优先，可以为一个字段配置多个验证规则  2.非字段验证验证规则优先  3.大部分验证规则支持两种验证器，但个别的验证规则只能使用非字段验证，  例如表达式验证。 |

## 错误消息的重用性

|  |
| --- |
| 多个字段使用同样的验证规则，可否使用同一条验证消息 ？ |

## 自定义验证器

|  |
| --- |
|  |

## 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 |

# 文件的上传下载

## 表单的准备

|  |
| --- |
| **要想使用 HTML 表单上传一个或多个文件**  须把 HTML 表单的 enctype 属性设置为 multipart/form-data  须把 HTML 表单的method 属性设置为 post  需添加 <input type=“file”> 字段. |

## Struts 对文件上传的支持

|  |
| --- |
| 在 Struts 应用程序里, FileUpload 拦截器和 Jakarta Commons FileUpload 组件可以完成文件的上传.  步骤:  1. 在 Jsp 页面的文件上传表单里使用 file 标签. 如果需要一次上传多个文件, 就必须使用多个 file 标签, 但它们的名字必须是相同的  2. 在 Action 中新添加 3 个和文件上传相关的属性. 这 3 个属性的名字必须是以下格式  [File Name] : File -被上传的文件。例如：data  [File Name]ContentType : String -上传文件的文件类型。例如：dataContentType  [File Name]FileName : String -上传文件的文件名。例如：dataFileName    3.如果上上传多个文件, 可以使用 List |

## 配置 FileUpload 拦截器

|  |
| --- |
| FileUpload 拦截器有 3 个属性可以设置  -maximumSize: 上传单个文件的最大长度(以字节为单位), 默认值为 2 MB  -allowedTypes: 允许上传文件的类型, 各类型之间以逗号分隔  -allowedExtensions: 允许上传文件扩展名, 各扩展名之间以逗号分隔  -可以在 struts.xml 文件中覆盖这 3 个属性    Commons FileUpload 组件默认接受上传文件总的最大值为 2M， 可以通过在 struts 配置文件中配置常量的方式修改  与文件上传有关的出错消息在 struts-messages.properties 文件里预定义. 可以在文件上传 Action 相对应的资源文件中重新定义错误消息 |

## 文件下载概述

|  |
| --- |
| 在某些应用程序里, 可能需要动态地把一个文件发送到用户的浏览器中, 而这个文件的名字和存放位置在编程时是无法预知的 |

## Stream 结果类型

|  |
| --- |
| Struts 专门为文件下载提供了一种 Stream 结果类型. 在使用一个 Stream 结果时, 不必准备一个 JSP 页面.  Stream 结果类型可以设置如下参数:  contentType：被下载的文件的 MIME 类型。默认值为 text/plain  contentLength：被下载的文件的大小，以字节为单位  contentDisposition： 可以设置下载文件名的ContentDispositon 响应头，默认值为 inline，通常设置为如下格式： attachment;filename="document.pdf".  inputName：Action 中提供的文件的输入流。默认值为 inputStream  bufferSize：文件下载时缓冲区的大小。默认值为 1024  allowCaching ：文件下载时是否允许使用缓存。默认值为 true  contentCharSet：文件下载时的字符编码。  **Stream 结果类型的参数可以在 Action 以属性的方式覆盖** |

# 防止表单重复提交

## 概述

|  |
| --- |
|  |

## Struts2 解决表单重复提交

|  |
| --- |
|  |

## 标记管理

|  |
| --- |
| Struts 提供的 token 标签可以用来生成一个独一无二的标记. 这个标签必须嵌套在 form 标签的内部使用, 它将在表单里插入一个隐藏字段并把标记值（隐藏域的字段的值）保存在HttpSession 对象里.  Token 标签必须与 Token 或 TokenSession 拦截器配合使用, 这两个拦截器都能对标记进行处理.  Token 拦截器在遇到重复提交情况时, 会返回 invalid.token 结果并加上一个 Action 错误. 这种错误默认的消息是: The form has already been processed or no token was supplied, please try again.  TokenSession 拦截器采取的做法只是阻断后续的提交, 用户将看到同样的响应，但实际上并没有重复提交 |

# 自定义拦截器

## Struts2 拦截器

|  |
| --- |
| 1.拦截器（Interceptor）是 Struts 2 的核心组成部分。  2.Struts2 很多功能都是构建在拦截器基础之上的，例如文件的上传和下载、国际化、数据类型转换和数据校验等等。  3.Struts2 拦截器在访问某个 Action 方法之前或之后实施拦截  4.Struts2 拦截器是可插拔的, 拦截器是 AOP（面向切面编程） 的一种实现  ．  5.拦截器栈(Interceptor Stack): 将拦截器按一定的顺序联结成一条链. 在访问被拦截的方法时, Struts2 拦截器链中的拦截器就会按其之前定义的顺序被依次调用 |

## Struts2 自带的拦截器

|  |
| --- |
|  |

## Interceptor 接口

|  |
| --- |
| 每个拦截器都是实现了接口的 Java 类:    -init: 该方法将在拦截器被创建后立即被调用, 它在拦截器的生命周期内只被调用一次. 可以在该方法中对相关资源进行必要的初始化  -interecept: 每拦截一个请求, 该方法就会被调用一次.  -destroy: 该方法将在拦截器被销毁之前被调用, 它在拦截器的生命周期内也只被调用一次. |

## 自定义拦截器

|  |
| --- |
| 定义自定义拦截器的步骤  -自定义拦截器  -在 struts.xml 文件中配置自定义的拦截器 |

# 零配置

## Convention 插件

|  |
| --- |
| 从 Struts 2.1 开始, Struts 可以使用 Convention 插件来支持零配置:  Convention 插件完全抛弃配置信息, 不仅不需要使用 struts.xml 文件进行配置, 甚至不需要使用 Annotation 进行配置. 而是完全根据约定来自动配置.  安装 Conversion 插件: 复制 struts-2.2.1\lib\struts2-convention-plugin-2.2.1.jar 到当前当前 WEB 应用的 WEB-INF 的 lib 目录下. |

## 搜索 Action

|  |
| --- |
|  |

## 按约定映射命名空间

|  |
| --- |
|  |

## 按约定映射 Action

|  |
| --- |
|  |

## 按约定映射 Result

|  |
| --- |
|  |

## Action 链的约定

|  |
| --- |
|  |

## Conversion 插件的常用常量

|  |
| --- |
|  |

## Convention 的 Annotation

|  |
| --- |
| Conversion 插件使用 Annotation 来管理拦截器, 异常处理等相关配置. Conversion 还允许使用 Annotation 管理 Action 和 Result 的配置, 从而覆盖 Conversion 的约定. |

## Action 配置相关的 Annotation

|  |
| --- |
|  |

## Result 配置相关的 Annotation

|  |
| --- |
|  |

## 包和命名空间相关的 Annotation

|  |
| --- |
|  |

## 异常相关的 Annotation

|  |
| --- |
|  |

## 拦截器相关的 Annotation

|  |
| --- |
|  |

# 整合 Spring

## 概述

|  |
| --- |
|  |

## 让 Spring 管理控制器

|  |
| --- |
|  |

## 自动装配

|  |
| --- |
|  |