目标：

1. Struts 访问通配符使用
2. Struts常量
3. Struts核心业务

请求数据自动封装

类型换行

Struts中数据处理

文件上传

# 1. Struts2配置

## 1.1Struts2的Action开发的几种模式

方法一：继承ActionSupport （a shen sipio t常用）

|  |
| --- |
| **public** **class** UserAction **extends** ActionSupport {  } |

如果用struts的数据校验功能，必须继承此类！

方法二：实现Action接口

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

方式三：不继承也不实现

|  |
| --- |
| **public** **class** UserAction {  //自己传入参数  **private** String userName;  **public** **void** setUserName(String userName){  **this**.userName = userName;  }  } |

## 1.2通配符

|  |
| --- |
| <package name=*"config"* extends=*"struts-default"* namespace=*"/"* >  <!-- 使用通配符  \*： 值是用户访问路径决定 {1}：获取前面 第一个\*的值 然后再类中查找对应的方法 -->  <action name=*"user\_\*"* class=*"cn.itcast.config.UserAction"* method=*"{1}"*>    <!-- 根据返回的name 跳转不同的页面 -->  <result name=*"login"*>/index.jsp</result>  <result name=*"register"*>/index.jsp</result>    <!-- 返回第一个值 跳转页面 -->  <result name=*"{1}"*>/{1}.jsp</result> |

## 1.3路径匹配原则

|  |
| --- |
| <package name="config" namespace="/user" extends="struts-default" namespace="/" >  <action name="user\_\*" class="cn.itcast.config.UserAction" method="{1}">  <result name="{1}">/{1}.jsp</result>  </action>  </package> |
| 访问路径：<http://localhost:8080/struts2/user/user_login> 可以 |
| 访问路径：<http://localhost:8080/struts2/user/a/b/user_login> 可以 |
| 访问路径：<http://localhost:8080/struts2/a/b/user/user_login> 不可以 |

Tomcat

Localhost 找到访问的机器

8080 找到tomcat

Struts2 找到项目名称

/user/a/b 找这个路径的名称空间 找到就返回，没有向下

/user/a 找这个路径的名称空间 找到就返回，没有向下

/user 找这个路径的名称空间 找到就返回，没有向下

/ 默认名称空间， 找到就返回，没有就报错！！！！

## 1.4 Strust常量

Struts中默认访问后缀

Struts1 .do

Strust2 .action

如何修改默认后缀

1. Struts2 后缀定义： Struts-core-2.3.4-1.jar/org.apache.struts/default.properties

Struts.action.extension=action,,

1. struts.xml中 修改

|  |
| --- |
| <struts>  <!-- 全局配置 -->  <!-- 修改struts默认访问路径的后缀 value 中也可以指定没有后缀也可以访问 -->  <constant name=*"struts.action.extendion"* value=*"action,do，"*></constant>  <!-- 总配置文件： 引入其他所有配置文件 -->  <include file=*"cn/itcast/config/struts.xml"*></include>  </struts> |

1. 其他常量

|  |
| --- |
| 指定默认编码集,作用于HttpServletRequest的setCharacterEncoding方法 和freemarker 、velocity的输出  <constant name="struts.i18n.encoding" value="UTF-8"/>    自定义后缀修改常量  <constant name="struts.action.extension" value="do"/>  设置浏览器是否缓存静态内容,默认值为true(生产环境下使用),开发阶段最好关闭  <constant name="struts.serve.static.browserCache" value="false"/>  当struts的配置文件修改后,系统是否自动重新加载该文件,默认值为false(生产环境下使用),开发阶段最好打开  <constant name="struts.configuration.xml.reload" value="true"/>  开发模式下使用,这样可以打印出更详细的错误信息  <constant name="struts.devMode" value="true" />  默认的视图主题  <constant name="struts.ui.theme" value="simple" />  与spring集成时，指定由spring负责action对象的创建  <constant name="struts.objectFactory" value="spring" />    该属性设置Struts 2是否支持动态方法调用，该属性的默认值是true。  如果需要关闭动态方法调用，则可设置该属性为 false  <constant name="struts.enable.DynamicMethodInvocation" value="false"/>  上传文件的大小限制  <constant name="struts.multipart.maxSize" value=“10701096"/> |

## 1.5全局跳转视图配置、配置的各项默认值

|  |
| --- |
| <!-- 配置全局跳转视图 -->  <global-results>  <result name=*"success"*>/index.jsp</result>  </global-results>    <!-- 配置各项默认值  name 默认执行的action在struts-default有配置  method 默认为execute  默认execute返回值为success，对应的页面会去全局视图找  -->  <action name=*"test"*></action>  <!-- 什么情况下不配置class 即处理的action    只需要跳转到WEB-INF下资源的时候 跳转WEB-INF下的资源 只能转发不能重定向  -->  <action name=*"test"*></action>  <result name=*"success"*>/index.jsp</result>  </action> |

# 2. Struts2中数据处理

方式一：耦合方式 直接拿到servletApi 需要导包

|  |
| --- |
| **public** **class** DataAction **extends** ActionSupport {  **public** String execute() **throws** Exception{  //1.请求数据封装 2.调用service处理业务 拿到数据结果  //3.数据保存到域中    //Struts中对数据操作 方式1: 直接拿到servletApi，执行操作  HttpServletRequest request = ServletActionContext.*getRequest*();  HttpSession session = request.getSession();  ServletContext application = ServletActionContext.*getServletContext*();  //操作 保存  request.setAttribute("request\_data", "request\_data");  session.setAttribute("session\_data", "session\_data");  application.setAttribute("application\_data", "application\_data");  **return** ***SUCCESS***; |

方式二： 解耦方式 通过ActionContext类操作数据（常用）

|  |
| --- |
| **public** **class** DataAction **extends** ActionSupport {  **public** String execute() **throws** Exception{  //1.请求数据封装 2.调用service处理业务 拿到数据结果  //3.数据保存到域中  //Struts中对数据操作 方式2：通过ActionContext类  ActionContext ac = ActionContext.*getContext*();  //得到Struts对httpServletRequest对象进行封装，封装为一个Map  //拿到表示request对象的 map  Map<String, Object> request = ac.getContextMap();  //拿到session对象的map  Map<String, Object> session = ac.getSession();  //拿到servletContext对象map  Map<String, Object> application = ac.getApplication();    //数据  request.put("request\_data", "request\_data");  session.put("session\_data", "session\_data");  application.put("application\_data", "application\_data");    **return** ***SUCCESS***; |

方式三： 实现接口的方式

|  |
| --- |
| //struts运行时候吗会代表request的map对象注入  **public** **class** DataAction **extends** ActionSupport **implements** RequestAware, SessionAware ,ApplicationAware {  **private** Map<String, Object> request;  **private** Map<String, Object> session;  **private** Map<String, Object> application;  //注入  **public** **void** setRequest(Map<String, Object> arg0) {  **this**.request = request;  }  //注入  **public** **void** setSession(Map<String, Object> arg0) {  **this**.session = session;  }  //注入  **public** **void** setApplication(Map<String, Object> arg0) {  **this**.application = application;  }  **public** String execute() **throws** Exception{  //数据  request.put("request\_data", "request\_data");  session.put("session\_data", "session\_data");  application.put("application\_data", "application\_data");    **return** ***SUCCESS***; |

# 3. 请求数据自动封装

实现原理： 参数拦截器

<interceptor

name=”params” class=”com.opensymphpony.xwork2.interceptor.Parameterslnterceptor”/>

方式1：jsp表单数据填充到action中的属性

|  |
| --- |
| Register.Jsp |
| <body>  <form action=*"*${pageContext.request.contextPath }*/user\_register.action"* method=*"post"*>  用户名:<input type=*"text"* name=*"name"*><br/>  密码:<input type=*"text"* name=*" pwd"*><br/>  年龄:<input type=*"text"* name=*" age"*><br/>  生日:<input type=*"text"* name=*" birth"*><br/>    <input type=*"submit"* value=*"注册"*>  </form>  </body> |
| **public** **class** User {  // 封装请求数据  **private** String name; // 必须给set / get可以不用给  **private** String pwd;  **private** **int** age;  **private** Date birth;  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **void** setPwd(String pwd) {  **this**.pwd = pwd;  }  **public** **void** setAge(**int** age) {  **this**.age = age;  }  **public** **void** setBirth(Date birth) {  **this**.birth = birth;  }  // 处理注册请求  **public** String register() {  System.***out***.println(name);  System.***out***.println(pwd);  System.***out***.println(age);  System.***out***.println(birth);  **return** "success";  }  } |

方式2：

|  |
| --- |
| <body>  <form action=*"*${pageContext.request.contextPath }*/user\_register.action"* method=*"post"*>  用户名:<input type=*"text"* name=*"user.name"*><br/>  密码:<input type=*"text"* name=*"user.pwd"*><br/>  年龄:<input type=*"text"* name=*"user.age"*><br/>  生日:<input type=*"text"* name=*"user.birth"*><br/>    <input type=*"submit"* value=*"注册"*>  </form>  </body> |
| **public** **class** UserAction {    // 对象类型，一定给get方法  **private** User user;  **public** **void** setUser(User user) {  **this**.user = user;  }  **public** User getUser() {  **return** user;  }  // 处理注册请求  **public** String register() {  System.***out***.println(user.getName());  System.***out***.println(user.getPwd());  System.***out***.println(user.getAge());  System.***out***.println(user.getBirth());  **return** "success";  }  } |

# 类型转换

Strust中jsp提交的数据 struts会自动转换为action中的属性类型

对基本数据类型 and 日期类型会自动转换 日期只支持 yyyy-MM-dd 格式

其他的格式就要 自定义类型转换器

Struts 转换器 API

|TypeConverter 转换器类接口

| DefaultTypeConverter 默认类型装换器

|StrutsTypeConverter 用户编写的转换器类，继承此类就可

## 局部类型转换器

转换器开发步骤

1. 写转换器类
2. 配转换器类（告诉struts应用自己的装换器类）

-->在同包的action，目录下，新建一个properties文件

-->命名规则：ActionClassName-conversion.properties

例：cn.itcast.type/UserAction-conversion.properies

1. 内容

User.birth=转换器类全路径（cn.itcast.type.MyConverter）

(限制类型) 类.属性 = 全路径 (限制原则 在MyConverter 编写)

## 全局类型转换器

给全部的action用

配置全局类型转换器

-->src/xwork-conversion.properties

-->内容

Java.util.Date=转换器类( cn.itcast.type.MyConverter)

# 5 文件上传

Servlet 中文件上传

前台：

1. 提交方式POST
2. 表单类型 multipart / form-data
3. input type=file

后台：

Apache 提供的FileUpload 组件

核心类：

FileltemFactory Fileltem 的工厂

ServletFileUpload servlet 中文件上传的核心类

Fileltem 封装了上传的文件项的信息

## Struts2 中文件上传

文件上传拦截器帮助我们完成类文件上传功能

|  |
| --- |
| Upload.jsp |
| <form action=*"*${pageContext.request.contextPath }*/fileUploadAction"* method=*"post"* enctype=*"multipart/form-data"*>  用户名:<input type=*"text"* name=*"userName"*><br/>  文件:<input type=*"file"* name=*"file1"*><br/>    <input type=*"submit"* value=*"上传"*>  </form> |
| FileUpload.java |
| **public** **class** FileUpload **extends** ActionSupport {  // 对应表单：<input type="file" name="file1">  **private** File file1;  // 文件名  **private** String file1FileName;  // 文件的类型(MIME)  **private** String file1ContentType;  **public** **void** setFile1(File file1) {  **this**.file1 = file1;  }  **public** **void** setFile1FileName(String file1FileName) {  **this**.file1FileName = file1FileName;  }  **public** **void** setFile1ContentType(String file1ContentType) {  **this**.file1ContentType = file1ContentType;  }    @Override  **public** String execute() **throws** Exception {  /\*\*\*\*\*\*拿到上传的文件，进行处理\*\*\*\*\*\*/  // 把文件上传到upload目录    // 获取上传的目录路径  String path = ServletActionContext.*getServletContext*().getRealPath("/upload");  // 创建目标文件对象  File destFile = **new** File(path,file1FileName);  // 把上传的文件，拷贝到目标文件中  FileUtils.*copyFile*(file1, destFile);    **return** ***SUCCESS***;  } |

## 文件上传细节处理

1. 文件大小限制

|  |
| --- |
| <!-- 修改上传文件的最大 大小为30M -->  <constant name=*"struts.multipart.maxSize"* value=*"31457280"*/> |

1. 限制文件上传允许的类型

需求：只运行txt/jpg后缀的文件

拦截器：注入参数 从而限制文件上传类型

|  |
| --- |
| <!-- 限制运行上传的文件的类型 -->  <interceptor-ref name=*"defaultStack"*>    <!-- 限制运行的文件的扩展名 -->  <param name=*"fileUpload.allowedExtensions"*>txt,jpg,jar</param>    <!-- 限制运行的类型 【与上面同时使用，取交集】  <param name="fileUpload.allowedTypes">text/plain</param> -->  </interceptor-ref>    <result name=*"success"*>/e/success.jsp</result> |

1. 错误提示

当文件上传出错时，strust内部会放回input视图（错误视图），需要我们在strust.xml中配置input视图对应的错误页面

|  |
| --- |
| <result name=*"success"*>/e/success.jsp</result>  <!-- 配置错误视图 -->  <result name=*"input"*>/e/error.jsp</result> |

# 6.Struts的文件下载

## Struts文件下载方式：

方式1：通过response对象向浏览器写入字节流数据，设置下载的响应头

方式2：struts的方式

|  |
| --- |
| Java |
| /\*\*  \* 文件下载  \* 1. 显示所有要下载文件的列表  \* 2. 文件下载  \* **@author** Jie.Yuan  \*/  **public** **class** DownAction **extends** ActionSupport {  /\*\*\*\*\*\*\*\*\*\*\*\*\*1. 显示所有要下载文件的列表\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  **public** String list() **throws** Exception {    //得到upload目录路径  String path = ServletActionContext.*getServletContext*().getRealPath("/upload");  // 目录对象  File file = **new** File(path);  // 得到所有要下载的文件的文件名  String[] fileNames = file.list();  // 保存  ActionContext ac = ActionContext.*getContext*();  // 得到代表request的map (第二种方式)  Map<String,Object> request= (Map<String, Object>) ac.get("request");  request.put("fileNames", fileNames);  **return** "list";  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*2. 文件下载\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  // 1. 获取要下载的文件的文件名  **private** String fileName;  **public** **void** setFileName(String fileName) {  // 处理传入的参数中问题(get提交)  **try** {  fileName = **new** String(fileName.getBytes("ISO8859-1"),"UTF-8");  } **catch** (UnsupportedEncodingException e) {  **throw** **new** RuntimeException(e);  }  // 把处理好的文件名，赋值  **this**.fileName = fileName;  }    //2. 下载提交的业务方法 (在struts.xml中配置返回stream)  **public** String down() **throws** Exception {  **return** "download";  }    // 3. 返回文件流的方法  **public** InputStream getAttrInputStream(){  **return** ServletActionContext.*getServletContext*().getResourceAsStream("/upload/" + fileName);  }  // 4. 下载显示的文件名（浏览器显示的文件名）  **public** String getDownFileName() {  // 需要进行中文编码  **try** {  fileName = URLEncoder.*encode*(fileName, "UTF-8");  } **catch** (UnsupportedEncodingException e) {  **throw** **new** RuntimeException(e);  }  **return** fileName;  }  } |
| Jsp |
| <html>  <head>  <title>下载列表</title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  </head>    <body>  <table border=*"1"* align=*"center"*>  <tr>  <td>编号</td>  <td>文件名</td>  <td>操作</td>  </tr>  <%@taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"* %>  <c:forEach var=*"fileName"* items=*"*${fileNames}*"* varStatus=*"vs"*>  <tr>  <td>${vs.count }</td>  <td>${fileName }</td>  <td>  <!-- 构建一个url -->  <c:url var=*"url"* value=*"down\_down"*>  <c:param name=*"fileName"* value=*"*${fileName}*"*></c:param>  </c:url>    <a href=*"*${url }*"*>下载</a>  </td>  </tr>  </c:forEach>  </table>  </body>  </html> |

图解：下载执行流程图



