Struts2

作者：任亮

讲师：任亮

# 系统开发流程分层

我们所学习的struts2是mvc层，让我们开发变得更简单

Mvc层

Jsp（视图V）

Model（模型M）

Servlet（控制层C）

Service（业务逻辑层）

Dao（数据访问层）

浏览器

# 什么是struts2



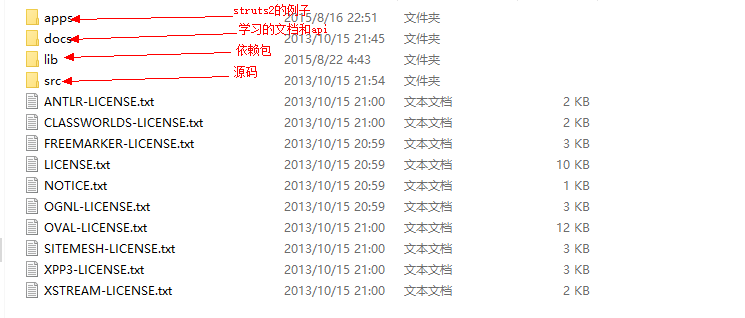
Struts 2是Struts的下一代产品，是在 struts 1和WebWork的技术基础上进行了合并的全新的Struts 2框架。其全新的Struts 2的体系结构与Struts 1的体系结构差别巨大。Struts2以WebWork为核心，采用拦截器的机制来处理用户的请求，这样的设计也使得业务逻辑控制器能够与ServletAPI完全脱离开，所以Struts 2可以理解为WebWork的更新产品。虽然从Struts 1到Struts 2有着太大的变化，但是相对于WebWork，Struts 2的变化很小。

Struts：框架。框架就是一个半成品。

WebWork：设计非常先进。

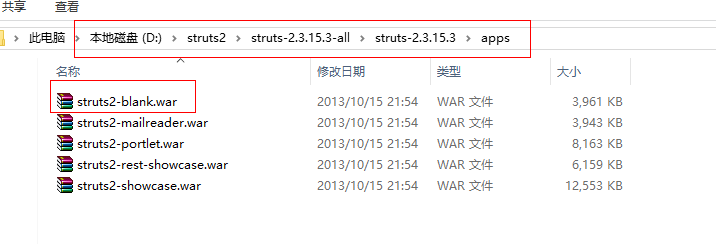
Struts2是一个位于表现层的非常优秀的MVC框架。它的核心是WebWork。

下载地址<http://struts.apache.org/download.cgi> 我们学习2.3



# 3.struts2的第一个例子

解压struts2-blank.war



## 创建WEB 工程

## 导入必要jar包

* struts2-core-2.3.1.1.jar：Struts 2框架的核心类库
* xwork-core-2.3.1.1.jar：Command模式框架,WebWork和Struts2都基于xwork
* ognl-3.0.3.jar：对象图导航语言(Object Graph Navigation Language),
* struts2框架通过其读写对象的属性
* freemarker-2.3.18.jar：Struts 2的UI标签的模板使用FreeMarker编写
* commons-logging-1.1.x.jar：ASF出品的日志包，Struts 2框架使用这个日志
* 包来支持Log4J和JDK 1.4+的日志记录。
* commons-fileupload-1.2.2.jar： 文件上传组件，2.1.6版本后需要加入此文件
* commons-io-2.0.1.jar：传文件依赖的jar包
* commons-lang-2.5.jar：对java.lang包的增强
* javassist-3.11.0.GA.jar是一个开源的分析、编辑和创建Java字节码的类库

## 编写JSP 页面

## 编写Action 服务器端处理逻辑

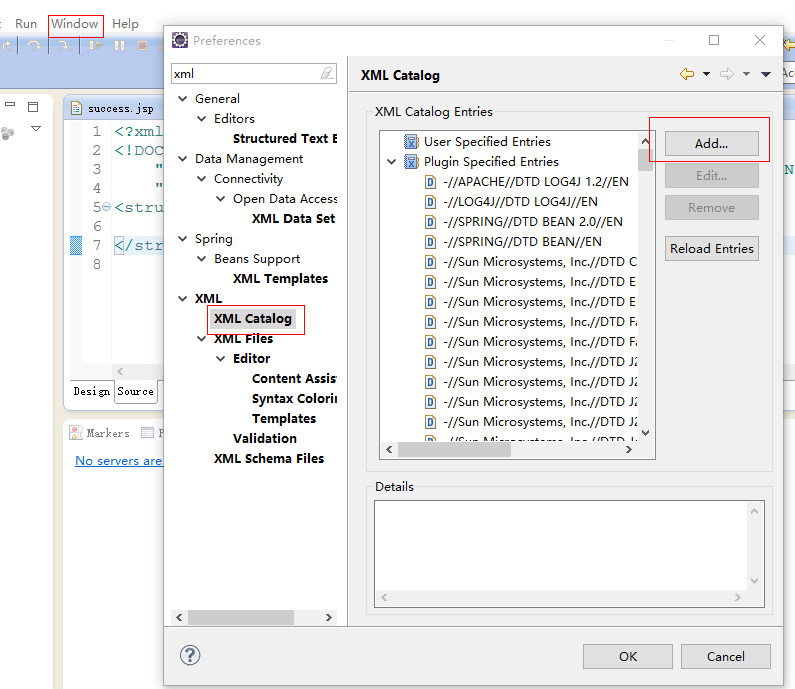
Action就是处理request请求的动作类，类似servlet

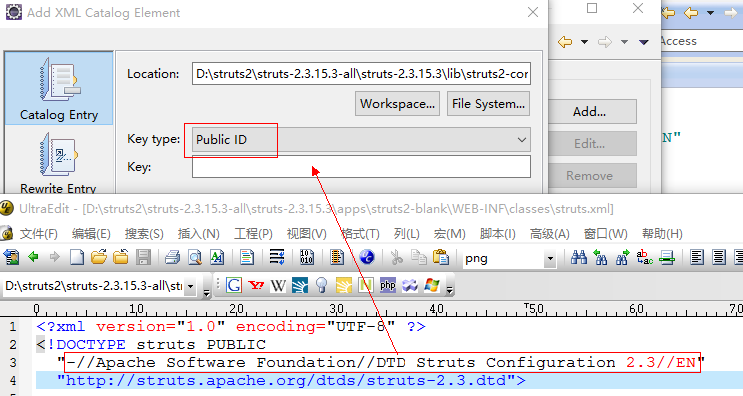
|  |
| --- |
| **public** **class** HelloAction {    **public** String hello(){  **return** "success";  }  } |

## 进行框架配置web.xml、struts.xml

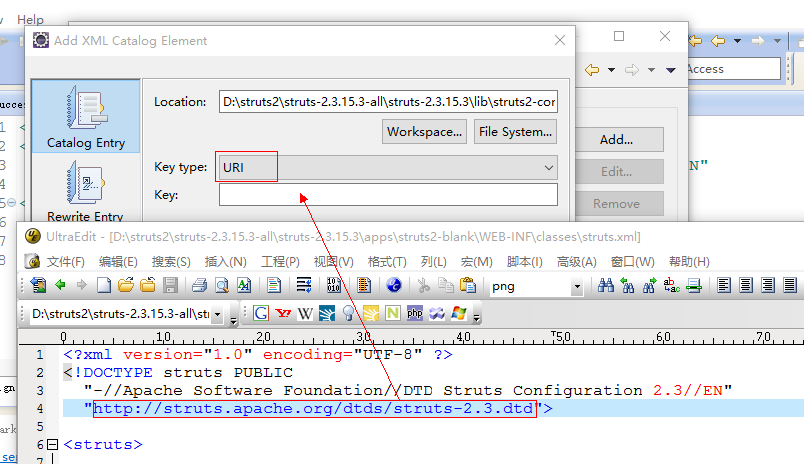
从例子中拷贝struts.xml到工程中的classpath下，流下头信息，其余清空

配置dtd的提示信息





或者



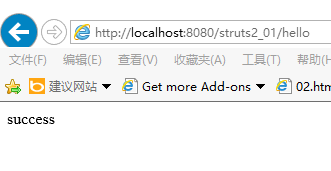
Struts.xml

|  |
| --- |
| <?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 name=*"hello"* extends=*"struts-default"*>  <action name=*"hello"* class=*"com.rl.action.HelloAction"* method=*"hello"*>  <result name=*"success"*>/success.jsp</result>  </action>  </package>  </struts> |

Web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app id=*"WebApp\_9"* version=*"2.4"* xmlns=*"http://java.sun.com/xml/ns/j2ee"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"*>    <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> |

## 运行测试



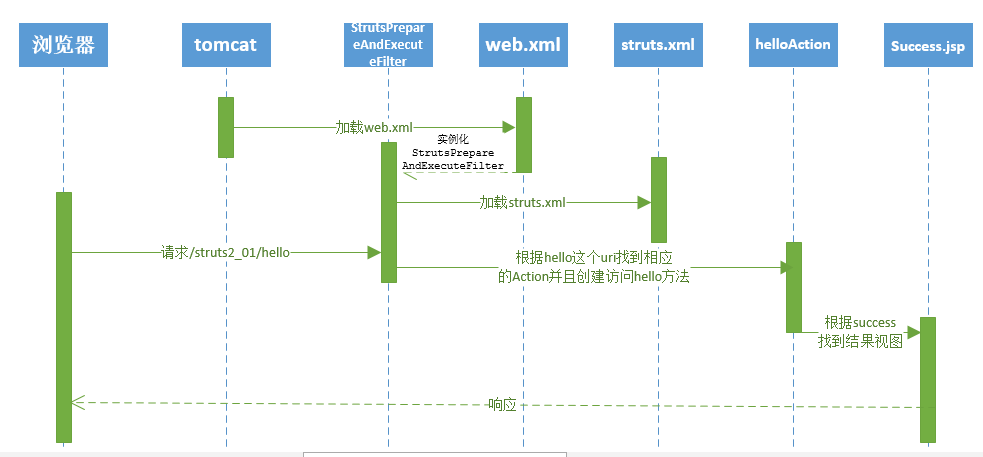
# 4.struts2的运行流程

1.启动服务，加载web.xml实例化StrutsPrepareAndExecuteFilter过滤器

2.在实例化StrutsPrepareAndExecuteFilter的时候会执行过滤器中的init方法加载struts.xml

3.浏览器发起请求，会被StrutsPrepareAndExecuteFilter拦截到，根据请求的uri（hello）找到相应的Action类，并且创建Action对象，执行相应的hello方法。

4.返回视图标志，展示success.jsp



# 5.struts2的配置文件

|  |  |  |  |
| --- | --- | --- | --- |
| 配置文件的名称 | 位置 | 存储的内容 | 说明 |
| Default.properties | Struts2-core-2.3.jar/org/apache/struts2/default.properties | 通过属性的形式来配置struts2的参数 | 不能直接修改（可以间接修改） |
| Struts-default.xml | Struts2-core-2.3.jar/ struts-default.xml | 是struts的核心配置文件 | 不能直接修改（可以间接修改） |
| Struts-plugin.xml | \*\*\*-plugin.jar中 | 插件相关配置 | 不能直接修改 |
| **Struts.xml** | 放在classpath中 | 应用配置文件 | 开发人员使用，可以修改 |
| Struts.properties | 放在classpath中 | 应用配置文件（较少使用） | 开发人员使用，可以修改 |

以上的配置文件服务器启动时会被加载，按着配置文件的加载的顺序，后面文件和前面文件相同的配置，后面的会把前面的文件的值覆盖。

## 1.default.properties

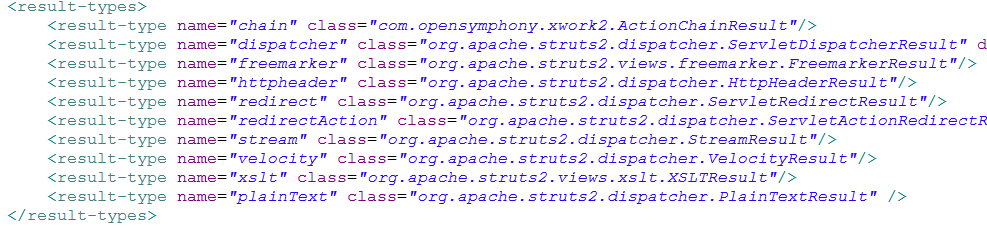
|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数默认值 | 说明 |
| struts.i18n.encoding | UTF-8 | 框架使用的编码 |
| struts.action.extension | action,, | 请求的后缀名 |
| struts.serve.static.browserCache | True | 是否开启浏览器的静态资源的缓存 |
| struts.configuration.xml.reload=false | False | 每当struts.xml被修改的时候我们是否要热部署，是否需要重启服务器 |
| struts.devMode | False | 是否是开发者模式 |

|  |
| --- |
| <!--  通过constant配置来间接修改default.properties的属性，修改请求的后缀  -->  <constant name=*"struts.action.extension"* value=*"do,,"*></constant> |

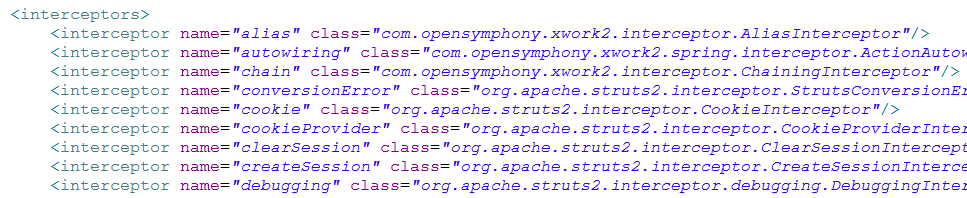
## 2.struts-default.xml

该文件是struts的核心文件，里面提供了结果的返回类型和拦截器还有业务bean

Result-type



拦截器



## 3 struts.xml重点

这个文件是开发人员要编写的文件

* **包**：

package是我们按着项目的模块来划分的一种单元，我们可以在开发中一个模块一个package

属性：

Name：必须的属性，包的名称

Extends:包的继承，默认情况下我们必须要继承struts-default,否则我们无法使用struts框架

Abstract：抽象包，在该包中不能有Action，其他的都可以有

Namespace：包的命名空间，值必须要有/,namespace用于请求访问时指定某一个包的路径，目的区分不同包的相同的Action名字

* 动作

Action是每次请求所访问的方法

属性：

Name：必须要有的<http://localhost:8080/struts2_02/person/hello.action>,后缀不需要指定，我们访问方法时就是根据name来访问

Class：要访问的方法的所在类

Method：方法名

|  |
| --- |
| <action name=*"hello"* class=*"com.rl.action.HelloAction"* method=*"hello"*>  <result name=*"success"*>/success.jsp</result>  </action> |

* 结果

Result就是要跳转的视图

属性：

Name：result的名称，是唯一的，到底Action返回哪个视图是由Action的方法的返回值来决定，返回值和result视图的name属性值做匹配，从而返回相应的视图。

Type：跳转视图的方式。默认情况跳转方式是请求转发，地址栏不变。

# 6.struts动作类（Action）

## 1.创建动作类

1.使用普通方式javaBean作为Action动作类，不需要继承任何父类，也不需要实现接口。

* 方法一定是public公用的，
* 返回值是字符串用来决定跳转到哪个视图
* 不需要参数
* 方法名自定义，如果不自定义的话，有个默认的方法名execute

|  |
| --- |
| **public** **class** HelloAction {    **public** String hello(){  System.*out*.println("任亮讲师");  **return** "success";  }  /\*\*  \* 当前请求没有绑定方法的时候自动找execute默认的方法  \* **@return**  \*/  **public** String execute(){  System.*out*.println("任亮是一个好讲师");  **return** "success";  }  } |

2.创建动作类实现接口com.opensymphony.xwork2.Action

Action接口中提供了一些常量

|  |  |  |
| --- | --- | --- |
| 常量 | 值 | 说明 |
| *SUCCESS* | Success | 返回成功页面 |
| *NONE* | None | 不返回任何页面 |
| *ERROR* | Error | 返回错误提示页面 |
| *INPUT* | Input | 当提交表单时发生错误，就跳回表单页面 |
| *LOGIN* | Login | 返回登录页面 |

1. 创建动作类继承父类com.opensymphony.xwork2.Action（推荐使用）

## 2.配置动作类

|  |
| --- |
| <!--  name:必须指定在一个包中唯一  class：当前action所在的动作类  method：当前action要访问的方法  -->  <action name=*"as"* class=*"com.rl.action.HelloAction2"* method=*"helloAS"*>  <result name=*"success"*>/success.jsp</result>  </action> |

## 3.action动作类的生命周期

创建：Action动作类每次请求的时候都会创建一个实例对象

销毁：当前action动作类的请求响应完后就消失了

跟javaweb中的HttpServletRequest的生命周期是一样的，struts2是多例的，线程安全的

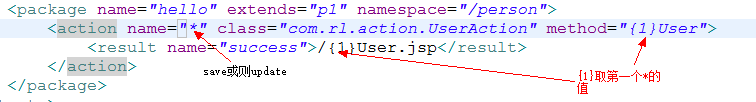
## 4.action动作类的访问

* 通配符

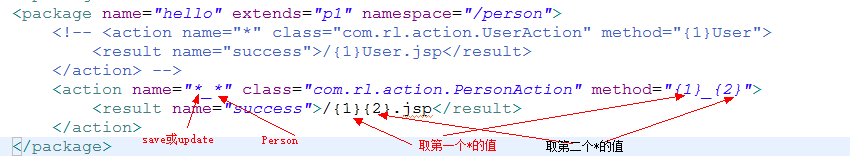
原则：约定优于配置

在action中每一个方法有一定的规则：都以User结尾

|  |
| --- |
| **public** **class** UserAction **extends** ActionSupport{  **public** String saveUser(){  System.*out*.println("啊..我被保存了");  **return** **super**.*SUCCESS*;  }    **public** String updateUser(){  System.*out*.println("啊..我被修改了");  **return** **super**.*SUCCESS*;  }  } |



|  |
| --- |
| **public** **class** PersonAction **extends** ActionSupport{  **public** String save\_Person(){  System.*out*.println("啊..我被保存了");  **return** **super**.*SUCCESS*;  }    **public** String update\_Person(){  System.*out*.println("啊..我被修改了");  **return** **super**.*SUCCESS*;  }  } |



注意：同一个包中不能有多个通配符的action

* 动态方法调用

首先要开启动态方法调用的开关

|  |
| --- |
| <!--  启用动态方法调用  -->  <constant name=*"struts.enable.DynamicMethodInvocation"* value=*"true"*/> |

这是我们的action的动作类不需要再指定方法method属性了，而是在url上指定

语法：！方法名[.后缀名]

<http://localhost:8080/struts2_05/person/renliang!hello3.do>

## 5 结果视图的配置

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

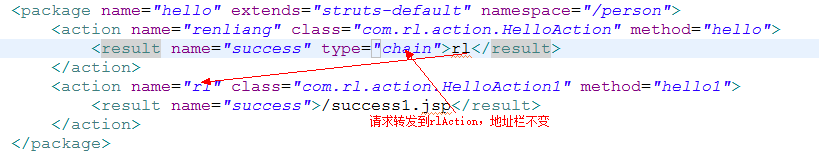
Result：要返回的视图

属性：

Name: action动作类要返回的值，如果返回的值匹配上了当前的name的值就跳转到相应的页面

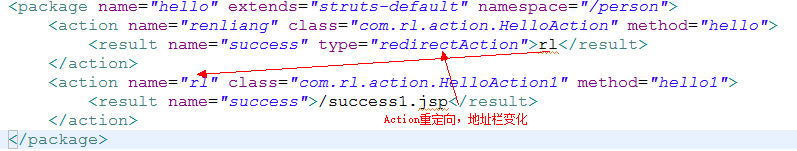
Type：跳转方式

1. Dispatcher:请求转发，相当于javaweb中的forward，地址栏不变，在我们不知道type的时候默认是dispatcher
2. Redirect:页面重定向，地址栏变化
3. Chain:请求转发到一个action动作类，地址栏不发生变化



1. redirectAction：重定向到一个action的动作类，地址栏变化

同包内的重定向



不同包的重定向，result需要通过param来指定namespace和actionName

|  |
| --- |
| <package name=*"hello"* extends=*"struts-default"* namespace=*"/person"*>  <action name=*"renliang"* class=*"com.rl.action.HelloAction"* method=*"hello"*>  <result name=*"success"* type=*"redirectAction"*>  <!--  指定命名空间  -->  <param name=*"namespace"*>/person1</param>  <!--  重定向的action的name  -->  <param name=*"actionName"*>rl</param>  </result>  </action>    </package>  <package name=*"hello1"* extends=*"struts-default"* namespace=*"/person1"*>  <action name=*"rl"* class=*"com.rl.action.HelloAction1"* method=*"hello1"*>  <result name=*"success"*>/success1.jsp</result>  </action>  </package> |

1. stream：文件的上传和下载。

## 6.局部结果视图和全局的结果视图

在一个action中配置的result是局部结果视图，外部的action是不能使用这个result

包内的全局的结果视图配置对于包内的每一个action动作类都有效

|  |
| --- |
| <package name=*"hello1"* extends=*"struts-default"* namespace=*"/person1"*>  <!--  包级别的全局结果视图  -->  <global-results>  <result name=*"error"*>/error.jsp</result>  </global-results>  <action name=*"rl"* class=*"com.rl.action.HelloAction1"* method=*"hello1"*>  <result name=*"success"*>/success1.jsp</result>  <!-- <result name="error">/error.jsp</result> -->  </action>  <action name=*"rl1"* class=*"com.rl.action.HelloAction2"* method=*"hello2"*>  <result name=*"success"*>/success1.jsp</result>  <!-- <result name="error">/error.jsp</result> -->  </action>  </package> |

在整个系统中的全局的结果视图，通过继承的方式来实现

|  |
| --- |
| <!-- 定义一个公用的包，继承struts-default -->  <package name=*"basePackage"* extends=*"struts-default"* abstract=*"true"*>  <global-results>  <result name=*"error"*>/error.jsp</result>  </global-results>  </package>    <package name=*"hello"* extends=*"basePackage"* namespace=*"/person"*>  <action name=*"renliang"* class=*"com.rl.action.HelloAction"* method=*"hello"*>  <result name=*"success"*>/success.jsp</result>  </action>  </package>  <package name=*"hello1"* extends=*"basePackage"* namespace=*"/person1"*>  <action name=*"rl"* class=*"com.rl.action.HelloAction1"* method=*"hello1"*>  <result name=*"success"*>/success1.jsp</result>  </action>  <action name=*"rl1"* class=*"com.rl.action.HelloAction2"* method=*"hello2"*>  <result name=*"success"*>/success1.jsp</result>  </action>  </package> |

# 7.struts动作类获得ServletAPI

1.使用ServletActionContext获得servletAPI

|  |
| --- |
| **public** String hello(){  //获得page域对象获得不到(很少使用)  PageContext page = ServletActionContext.*getPageContext*();  //获得request  HttpServletRequest request = ServletActionContext.*getRequest*();  //获得response  HttpServletResponse response = ServletActionContext.*getResponse*();  //获得session  HttpSession session = request.getSession();  //获得application  ServletContext sc = ServletActionContext.*getServletContext*();  System.*out*.println(page);  System.*out*.println(request);  System.*out*.println(response);  System.*out*.println(session);  System.*out*.println(sc);  **return** **super**.*SUCCESS*;  } |

2.通过实现接口的方式来获得servletAPI

实现

org.apache.struts2.interceptor.ServletRequestAware;

org.apache.struts2.interceptor.ServletResponseAware;

org.apache.struts2.interceptor.SessionAware;

org.apache.struts2.util.ServletContextAware;

|  |
| --- |
| **public** **class** HelloAction1 **extends** ActionSupport **implements**  ServletRequestAware,  ServletResponseAware, ServletContextAware{  HttpServletRequest request;    HttpServletResponse response;    ServletContext sc;    **public** String hello(){  System.*out*.println(request);  String result = request.getParameter("name");  System.*out*.println(result);  System.*out*.println(request.getSession());  System.*out*.println(response);  System.*out*.println(sc);  **return** **super**.*SUCCESS*;  }  @Override  **public** **void** setServletRequest(HttpServletRequest request) {  **this**.request = request;  }  @Override  **public** **void** setServletResponse(HttpServletResponse response) {  **this**.response = response;  }  @Override  **public** **void** setServletContext(ServletContext context) {  **this**.sc = context;  }  } |

# 8.参数的封装

## 1.静态封装

在运行器不发生变化的数据，或者一些配置相关的数据可以做静态封装

Action动作类

|  |
| --- |
| **public** **class** PersonAction **extends** ActionSupport{    /\*\*  \* 用于接收静态的参数，必须提供set和get方法  \*/  **private** String name = "任亮";    **private** String desc;    **public** String getDesc() {  **return** desc;  }  **public** **void** setDesc(String desc) {  **this**.desc = desc;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** String hello(){  System.*out*.println(name);  System.*out*.println(desc);  **return** **super**.*SUCCESS*;  }  } |

Struts.xml配置

|  |
| --- |
| <package name=*"person"* extends=*"struts-default"* namespace=*"/person"*>  <action name=*"renliang"* class=*"com.rl.action.PersonAction"* method=*"hello"*>  <!--  param的name在Action中必须要有相应的属性  -->  <param name=*"name"*>任亮老师</param>  <param name=*"desc"*>讲javaee的互联网的技术</param>  <result name=*"success"*>/success.jsp</result>  </action>  </package> |

## 2.动态参数的封装

系统云期间用户提交表单，ajax，url访问。

### 动作类充当模型对象

动作类和模型和为一体，我们可以在动作类中定义要接收的属性的值，对每个属性必须要提供set和get方法，动作类model中的属性的名字和表单中的name一定要一致，否则无法注入。

Jsp页面

|  |
| --- |
| <form action=*"*${pageContext.request.contextPath }*/person/renliang1"* method=*"post"*>  <table>  <tr>  <td>ID</td>  <td><input type=*"text"* name=*"id"*></td>  </tr>  <tr>  <td>姓名</td>  <td><input type=*"text"* name=*"name"*></td>  </tr>  <tr>  <td>性别</td>  <td>  <input type=*"radio"* name=*"gender"* value=*"1"* checked=*"checked"*>男  <input type=*"radio"* name=*"gender"* value=*"2"*>女  </td>  </tr>  <tr>  <td>地址</td>  <td><input type=*"text"* name=*"address"*></td>  </tr>  </table>  <input type=*"submit"* value=*"提交"*>  </form> |

动作类model

|  |
| --- |
| **public** **class** PersonAction1 **extends** ActionSupport{    /\*\*  \* 在Action的内部定义model的属性  \* **@return**  \*/  **private** **int** id;    **private** String name;    **private** **int** gender;    **private** String address;    **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getGender() {  **return** gender;  }  **public** **void** setGender(**int** gender) {  **this**.gender = gender;  }  **public** String getAddress() {  **return** address;  }  **public** **void** setAddress(String address) {  **this**.address = address;  }  **public** String hello(){  System.*out*.println(id);  System.*out*.println(name);  System.*out*.println(gender);  System.*out*.println(address);  **return** **super**.*SUCCESS*;  }  } |

## 动作类和模型对象分离封装参数（推荐使用）

动作类充当模型对象Action既是是C又是M，可读性差，重用性差，很难维护，不推荐使用第一种。

这种方式接受参数对提交的表单中的name有要求，name需要使用Action中的model属性的名字加点再加上要接收的属性名：如 person.name

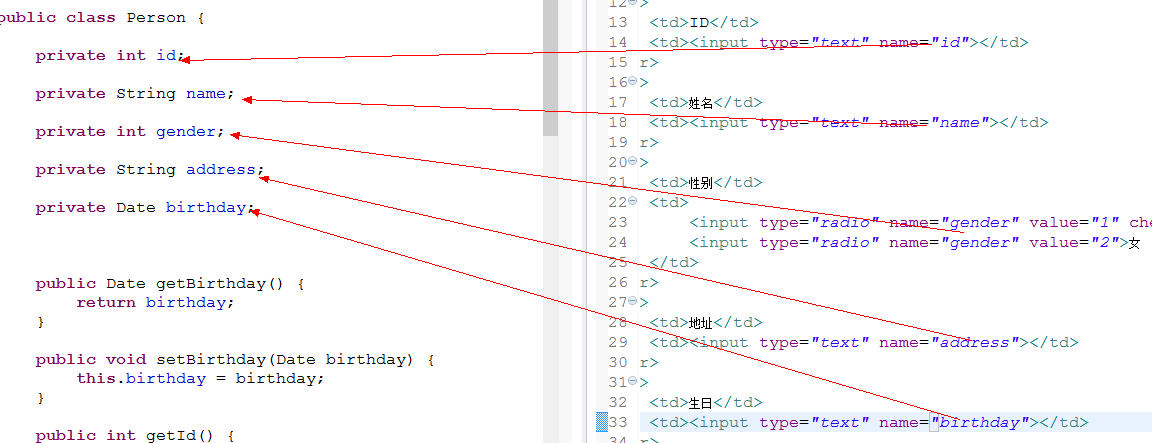
|  |
| --- |
| <form action=*"*${pageContext.request.contextPath }*/person/renliang2"* method=*"post"*>  <table>  <tr>  <td>ID</td>  <td><input type=*"text"* name=*"person.id"*></td>  </tr>  <tr>  <td>姓名</td>  <td><input type=*"text"* name=*"person.name"*></td>  </tr>  <tr>  <td>性别</td>  <td>  <input type=*"radio"* name=*"person.gender"* value=*"1"* checked=*"checked"*>男  <input type=*"radio"* name=*"person.gender"* value=*"2"*>女  </td>  </tr>  <tr>  <td>地址</td>  <td><input type=*"text"* name=*"person.address"*></td>  </tr>  </table>  <input type=*"submit"* value=*"提交"*>  </form> |

动作类里面需要定义model的属性并且提供set和get

|  |
| --- |
| **package** com.rl.action;  **import** com.opensymphony.xwork2.ActionSupport;  **import** com.rl.model.Person;  /\*\*  \* action动作类充当模型对象  \* **@author** renliang  \*  \*/  **public** **class** PersonAction2 **extends** ActionSupport{    /\*\*  \* 定义model类的对象用于接收参数  \* 必须提供set和get  \*/  **private** Person person;    **public** Person getPerson() {  **return** person;  }  **public** **void** setPerson(Person person) {  **this**.person = person;  }  **public** String hello(){  System.*out*.println(person);  **return** **super**.*SUCCESS*;  }  } |

## 模型驱动方式封装参数（推荐使用）

第二种方式对页面上的文本域的name有要求必须要用model的属性的名称.属性名，这样的话页面和Action动作类有侵入性，模型驱动方式解决了这个问题，我们需要实现一个ModelDriven接口指定要接收的model 的类型提供getModel的方法，Action类中的model对象必须要手动的创建，否则无法注入属性值。



页面：

|  |
| --- |
| <form action=*"*${pageContext.request.contextPath }*/person/renliang3"* method=*"post"*>  <table>  <tr>  <td>ID</td>  <td><input type=*"text"* name=*"id"*></td>  </tr>  <tr>  <td>姓名</td>  <td><input type=*"text"* name=*"name"*></td>  </tr>  <tr>  <td>性别</td>  <td>  <input type=*"radio"* name=*"gender"* value=*"1"* checked=*"checked"*>男  <input type=*"radio"* name=*"gender"* value=*"2"*>女  </td>  </tr>  <tr>  <td>地址</td>  <td><input type=*"text"* name=*"address"*></td>  </tr>  </table>  <input type=*"submit"* value=*"提交"*>  </form> |

Action类

|  |
| --- |
| **public** **class** PersonAction3 **extends** ActionSupport **implements** ModelDriven<Person>{    /\*\*  \* 定义model类的对象用于接收参数  \* 必须提供set和get,model的对象必须手动的创建，否则框架无法注入属性  \*/  **private** Person person = **new** Person();    **public** Person getPerson() {  **return** person;  }  **public** **void** setPerson(Person person) {  **this**.person = person;  }  **public** String hello(){  System.*out*.println(person);  **return** **super**.*SUCCESS*;  }  @Override  **public** Person getModel() {  **return** person;  }  } |

# 9.类型转换

## 1.类型转换认识

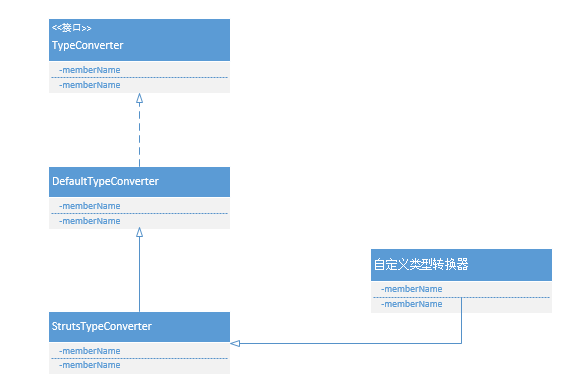
页面所提交过来的数据都是字符串的类型，而model里面的数据类型并不都是字符串，有各种各样的类型int ，Integer， float， String，date…,我们使用servlet来接收数据时都是自己手动转换，struts2可以为我们自动转换，转换的前提是前台提交的字符串和model中的相应的数据类型可以转换。

提交表单时：字符串-------🡪其他类型

页面展示：其他类型--------🡪字符串

表单提交时数据类的转换struts给我做了绝大多数，但是时间类型往往需要根据我们自己的需求来转换的。

## 2.如果自定义转换数据类型呢



日期转换：

默认情况下，struts2解析的时间的格式：“yyyy-MM-dd”

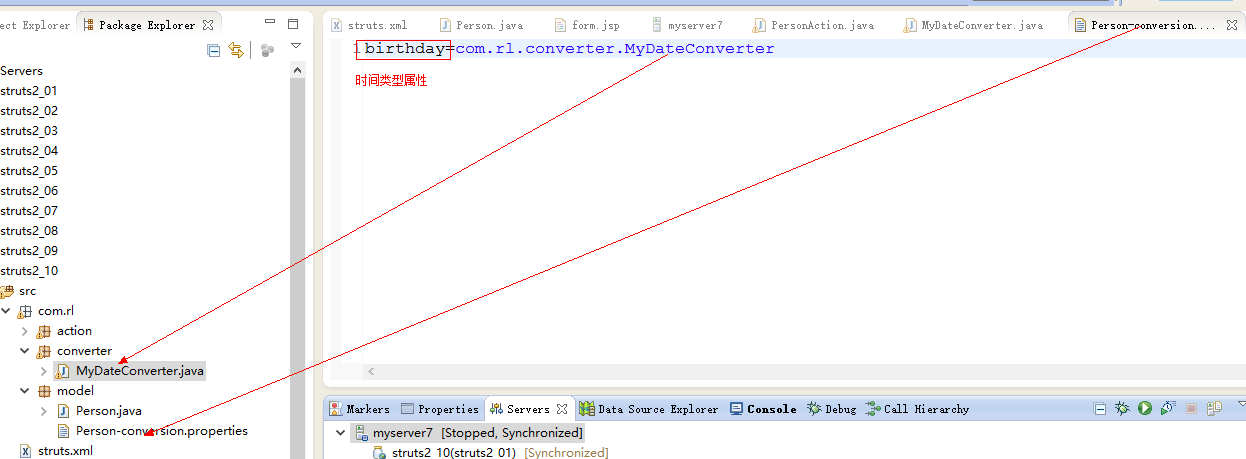
### 1.通过自定义类型转换器继承StrutsTypeConverter做日期转换器

|  |
| --- |
| **public** **class** MyDateConverter **extends** StrutsTypeConverter {  @Override  **public** Object convertFromString(Map context, String[] values, Class toClass) {  Date date = **null**;  **if**(values != **null** && values.length > 0){  **if**(toClass == Date.**class**){  **try** {  date = **new** SimpleDateFormat("yyyy/MM/dd").parse(values[0]);  } **catch** (ParseException e) {  e.printStackTrace();  }  }  }  **return** date;  }  @Override  **public** String convertToString(Map context, Object o) {  String dateStr = **null**;  **if**(o != **null** && o.getClass() == Date.**class**){  dateStr = **new** SimpleDateFormat("yyyy/MM/dd").format((Date)o);  }    **return** dateStr;  }  } |

### 2.注册日期类型转换器

#### 局部注册

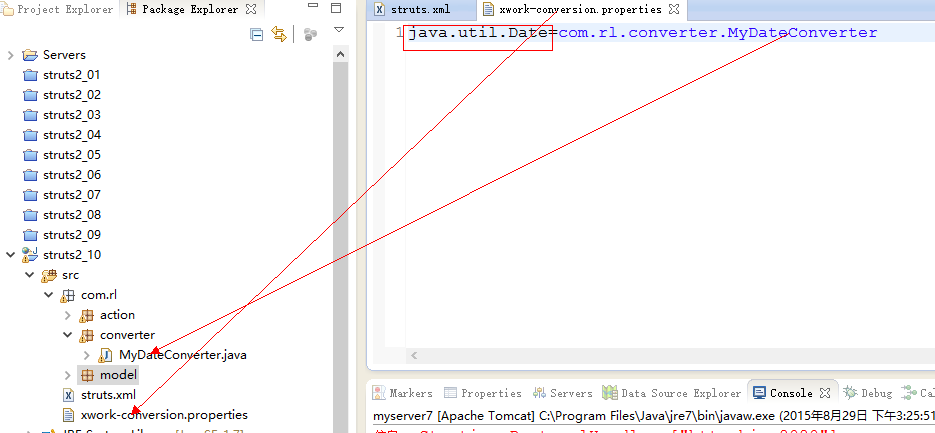
第一步：在要转换的model类的同级包中建立一个属性文件，命名要求：model类名-conversion.properties，在文件的内部key是要转换的属性名，value是转换器的全路径



#### 全局注册

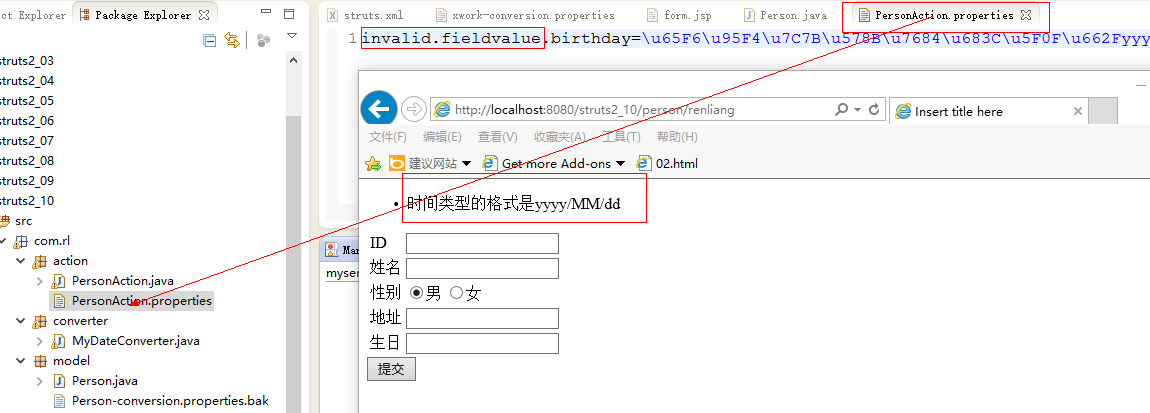
如果有很多的model类都有时间类型，那么使用局部类型转换器就需要创建很多注册文件，这样不合理。需要全局注册

方法：在classpath下一个属性文件：xwork-conversion.properties



如果把错误提示信息转换成中文呢

1. 在Action类的同级包下创建一个属性文件, 规则：和Action类同名.properties
2. Key: invalid.fieldvlaue.[要转换的表单中文本域的name：birthday]，value是中文的提示信息，中文在properties文件中以Unicode的编码的方式



# 10.struts2数据验证

验证的方式

1.客户端校验：使用js结合正则表达式来校验，不和服务器打交道，开发简单，安全性差

2.服务器端校验：请求web服务器，开发量比较大，安全性好，如果需要和数据库到交到的就必须要使用服务器端校验

实际开发中，客户端校验比较多，最好是客户端和服务端都校验。

Struts2的校验属于服务器端校验

## struts2编程式验证

编程校验：重写父类的validate方法，在这个方法中对每一个字段来做校验，但是校验的前提是提交过来的参数是能相互转换的，如果转换不了的话会由param，modelDriven拦截器来负责。如果参数不符合正则，我们可以添加提示信息addFieldError("id", "id必须是1到5位的数字")

|  |
| --- |
| @Override  **public** **void** validate() {    **if**(person.getId() != 0 && !(person.getId()+"").matches("\\d{1,3}")){  addFieldError("id", "id必须是1到5位的数字");  }  **if**(!person.getName().matches("[a-zA-Z]{3,8}")){  //添加提示信息，第一个参数表单中文本域的name，第二个是提示信息  addFieldError("name", "姓名只能是3到8位大小字母");  }  } |
|  |

跳出校验

1. 在编程式校验中validate会对当前的Action中的每一个方法做校验，那么有一些方法时不需要校验的，我们可以通过@skipValidation来跳过校验。

|  |
| --- |
| @SkipValidation//跳出校验  **public** String delete(){  System.*out*.println(person);  **return** **super**.*SUCCESS*;  } |

2.我们在Action中定义一个验证方法，命名规则是validate+要验证的方法名，那么这个方法就会被校验如validateSave，save方法就会被校验，其余的方法都不会被校验

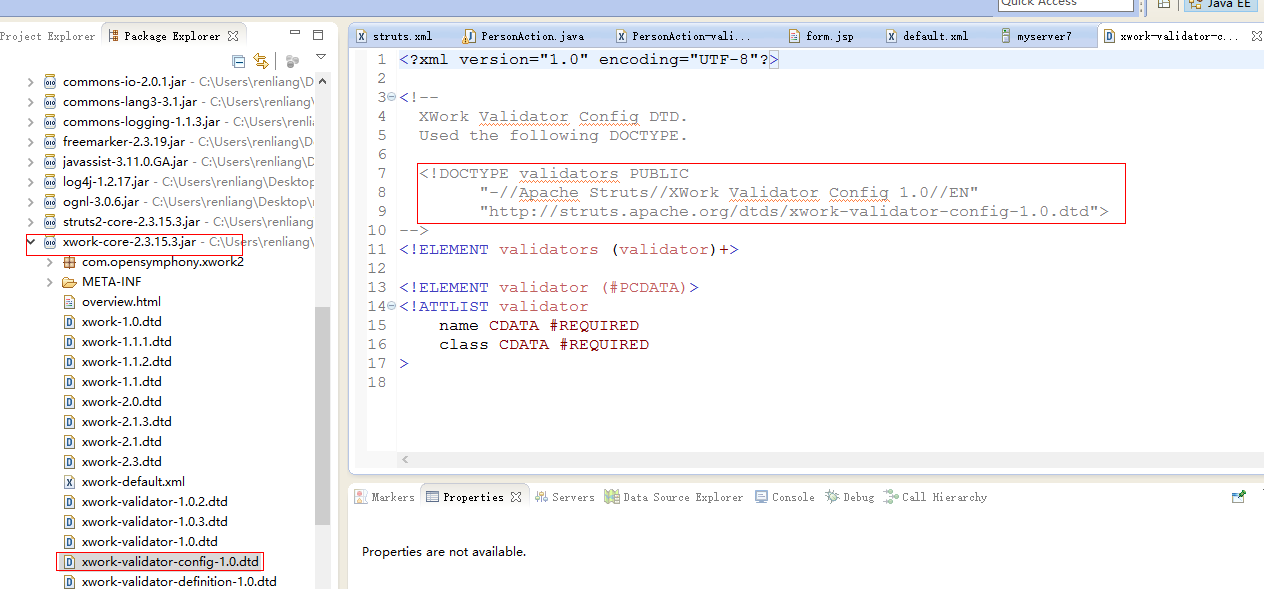
|  |
| --- |
| **public** **void** validateSave() {    **if**(person.getId() != 0 && !(person.getId()+"").matches("\\d{1,3}")){  addFieldError("id", "id必须是1到5位的数字");  }  **if**(!person.getName().matches("[a-zA-Z]{3,8}")){  //添加提示信息，第一个参数表单中文本域的name，第二个是提示信息  addFieldError("name", "姓名只能是3到8位大小字母");  }  } |

## struts2声明式验证

为了解决Action和验证逻辑的高耦合，我们可以使用声明式验证，我们要把验证规则配置在xml中。

规则：在Action的同级包下创建一个xml，名称规范：Action名称-validation.xml

打开xwork-validator1.0.3.dtd,把头信息拷贝到我们的xml中



我们对验证做配置

|  |
| --- |
| <?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：表单中要提交的文本域  name：文本域的name  -->  <field name=*"name"*>  <!--  文本域的验证器  -->  <field-validator type=*"requiredstring"*>  <message>姓名不能为空</message>  </field-validator>  </field>  </validators> |

Struts2框架给我们提供了很多验证器，这些验证器的位置在xwork-core.2.3.15.jar/com/opensymphony/xwork2/validator/validators/default.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE validators PUBLIC  "-//Apache Struts//XWork Validator Definition 1.0//EN"  "http://struts.apache.org/dtds/xwork-validator-definition-1.0.dtd">  <!-- START SNIPPET: validators-default -->  <validators>  <validator name=*"required"* class=*"com.opensymphony.xwork2.validator.validators.RequiredFieldValidator"*/>  <validator name=*"requiredstring"* class=*"com.opensymphony.xwork2.validator.validators.RequiredStringValidator"*/>  <validator name=*"int"* class=*"com.opensymphony.xwork2.validator.validators.IntRangeFieldValidator"*/>  <validator name=*"long"* class=*"com.opensymphony.xwork2.validator.validators.LongRangeFieldValidator"*/>  <validator name=*"short"* class=*"com.opensymphony.xwork2.validator.validators.ShortRangeFieldValidator"*/>  <validator name=*"double"* class=*"com.opensymphony.xwork2.validator.validators.DoubleRangeFieldValidator"*/>  <validator name=*"date"* class=*"com.opensymphony.xwork2.validator.validators.DateRangeFieldValidator"*/>  <validator name=*"expression"* class=*"com.opensymphony.xwork2.validator.validators.ExpressionValidator"*/>  <validator name=*"fieldexpression"* class=*"com.opensymphony.xwork2.validator.validators.FieldExpressionValidator"*/>  <validator name=*"email"* class=*"com.opensymphony.xwork2.validator.validators.EmailValidator"*/>  <validator name=*"url"* class=*"com.opensymphony.xwork2.validator.validators.URLValidator"*/>  <validator name=*"visitor"* class=*"com.opensymphony.xwork2.validator.validators.VisitorFieldValidator"*/>  <validator name=*"conversion"* class=*"com.opensymphony.xwork2.validator.validators.ConversionErrorFieldValidator"*/>  <validator name=*"stringlength"* class=*"com.opensymphony.xwork2.validator.validators.StringLengthFieldValidator"*/>  <validator name=*"regex"* class=*"com.opensymphony.xwork2.validator.validators.RegexFieldValidator"*/>  <validator name=*"conditionalvisitor"* class=*"com.opensymphony.xwork2.validator.validators.ConditionalVisitorFieldValidator"*/>  </validators> |

系统提供的校验器如下：

* required (必填校验器,要求被校验的属性值不能为null)
* 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)
* stringlength(字符串长度校验器，要求被校验的属性值必须在指定的范围内，否则校验失败,minLength参数指定最小长度，maxLength参数指定最大长度，trim参数指定校验field之前是否去除字符串前后的空格)
* 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，expression参数指定正则表达式，caseSensitive参数指

|  |
| --- |
| <field name=*"name"*>  <!--  文本域的验证器  -->  <field-validator type=*"requiredstring"*>  <message>姓名不能为空</message>  </field-validator>  <field-validator type=*"regex"*>  <param name=*"regex"*>[a-zA-Z]{3,8}</param>  <message>姓名必须是3到8位的字符</message>  </field-validator>  </field> |

第二种验证配置方式

|  |
| --- |
| <validator type=*"regex"*>  <!-- 配置验证的字段 -->  <param name=*"fieldName"*>password</param>  <!-- 验证的正则表达式 -->  <param name=*"regex"*>\d{6,8}</param>  <message>请输入6到8位数字的密码</message>  </validator> |

* int(整数校验器，要求field的整数值必须在指定范围内，min指定最小值，max指定最大值)
* 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,min指定最小值,max指定最大值)
* fieldexpression(字段OGNL表达式校验器,要求field满足一个ognl表达式，expression参数指定ognl表达式,该逻辑表达式基于ValueStack进行求值,返回true时校验通过，否则不通过)
* 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)
* 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)
* date(日期校验器,要求field的日期值必须在指定范围内,min指定最小值,max指定最大值)
* conversion(转换校验器，指定在类型转换失败时，提示的错误信息)
* visitor(用于校验action中复合类型的属性，它指定一个校验文件用于校验复合类型属性中的属性)
* expression(OGNL表达式校验器，它是一个非字段校验器， expression参数指定ognl表达式,该逻辑表达式基于ValueStack进行求值,返回true时校验通过，否则不通过，该校验器不可用在字段校验器风格的配置中)

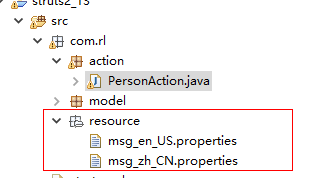
# 11.struts2国际化

1.先来建立独立的包的资源文件

命名的规范：以msg开头后面加上国家的语言和国家名称的简写

msg\_en\_US.properties

msg\_zh\_CN.properties



2.配置国际化资源文件的加载

|  |
| --- |
| <!--  加载资源文件做国际化，value指的是资源文件的包的路径后面加上msg，用/分隔  -->  <constant name=*"struts.custom.i18n.resources"* value=*"com/rl/resource/msg"*></constant> |

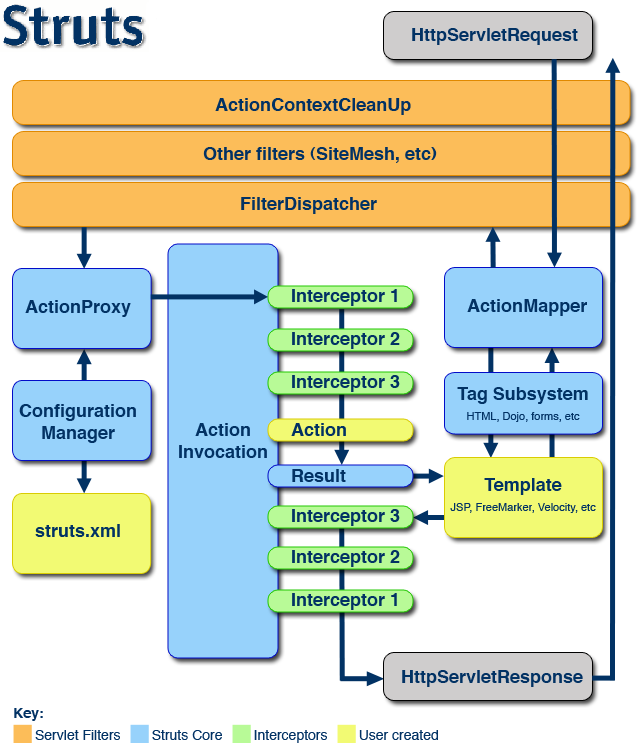
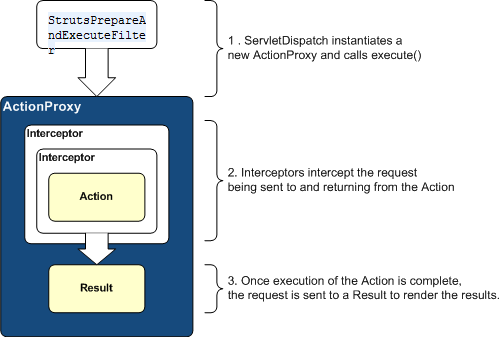
1. 在页面上使用国际化的资源<s:text name=’属性文件的key’>

|  |
| --- |
| <form action=*"*${pageContext.request.contextPath }*/person/renliang"* method=*"post"*>  <table>  <tr>  <td>ID</td>  <td><input type=*"text"* name=*"id"*></td>  <td><s:fielderror fieldName=*"id"*></s:fielderror></td>  </tr>  <tr>  <td><s:text name=*"pname"*/></td>  <td><input type=*"text"* name=*"name"*></td>  <td><s:fielderror fieldName=*"name"*></s:fielderror></td>  </tr>  <tr>  <td><s:text name=*"pgender"*/></td>  <td>  <input type=*"radio"* name=*"gender"* value=*"1"* checked=*"checked"*>男  <input type=*"radio"* name=*"gender"* value=*"2"*>女  </td>  <td></td>  </tr>  <tr>  <td><s:text name=*"paddr"*/></td>  <td><input type=*"text"* name=*"address"*></td>  <td><s:fielderror fieldName=*"address"*></s:fielderror></td>  </tr>    <tr>  <td><s:text name=*"pbirth"*/></td>  <td><input type=*"text"* name=*"birthday"*></td>  <td><s:fielderror fieldName=*"birthday"*></s:fielderror></td>  </tr>  </table>  <input type=*"submit"* value=*"*<s:text name=*'submit'*/>*"*>  </form> |

# 12.sruts2拦截器

## 1.struts拦截器介绍

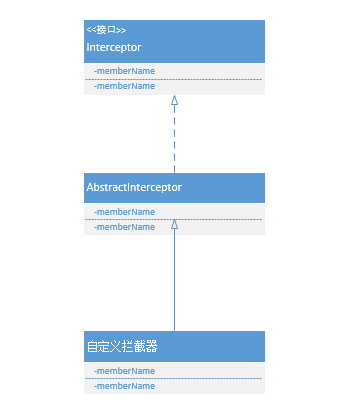
Struts2的核心就是拦截器，param，modelDriven，validation，servletAPI等等这些都是拦截器完成的功能。



服务器启动时实例化StrustPrepareAndExecuteFilter，读取struts所有的配置文件，把struts.xml中的每一个Action实例化，当一个请求访问的时候，StrustPrepareAndExecuteFilter会把请求拦截下来，匹配一个ActionMapper，然后ActionMapper创建ActionProxy，然后去执行默认拦截器栈，在执行Action，然后返回结果，然后默认拦截器再反向执行（做一些运行期的监控和清理工作，不是每一个拦截器都执行后置拦截），最后响应结果返回给页面。

## 自定义拦截器

自定义拦截器我们需要继承AbstractInterceptor



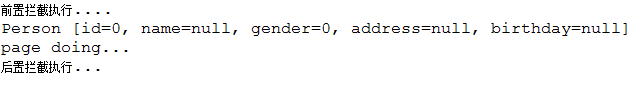
创建拦截器类

|  |
| --- |
| **public** **class** MyInterceptor **extends** AbstractInterceptor {  @Override  **public** String intercept(ActionInvocation invocation) **throws** Exception {  System.*out*.println("前置拦截执行....");  //让拦截器向下走并且返回结果代码  String result = invocation.invoke();  System.*out*.println("后置拦截执行...");  **return** result;  }  } |

拦截器的配置

|  |
| --- |
| <package name=*"person"* extends=*"struts-default"* namespace=*"/person"*>  <!-- 拦截器配置 -->  <interceptors>  <interceptor name=*"myInterceptor"* class=*"com.rl.interceptor.MyInterceptor"*></interceptor>  </interceptors>  <action name=*"renliang"* class=*"com.rl.action.PersonAction"* method=*"save"*>  <result name=*"success"*>/success.jsp</result>  <result name=*"input"*>/form.jsp</result>  </action>  <action name=*"delete"* class=*"com.rl.action.PersonAction"* method=*"delete"*>  <!-- 引用拦截器 -->  <interceptor-ref name=*"myInterceptor"*></interceptor-ref>  <result name=*"success"*>/success.jsp</result>  <result name=*"input"*>/form.jsp</result>  </action>  </package> |

我们从结果上可以看懂执行的顺序：前置拦截--🡪action---🡪页面执行--🡪后置拦截



## 3.拦截器和拦截器栈的配置

|  |
| --- |
| <package name=*"myStruts-Default"* extends=*"struts-default"* abstract=*"true"*>  <interceptors>  <!--  配置我们自己的拦截器  -->  <interceptor name=*"myInterceptor"* class=*"com.rl.interceptor.MyInterceptor"*></interceptor>  <!--  拦截器栈：是把多个拦截器集中到一起统一被引用  -->  <interceptor-stack name=*"rlStack"*>  <!--  拦截器栈中要引用已经配置好的拦截器或者拦截器栈  -->  <interceptor-ref name=*"defaultStack"*></interceptor-ref>  <interceptor-ref name=*"myInterceptor"*></interceptor-ref>  </interceptor-stack>  </interceptors>  <!--  把rlStack作为默认拦截器，只要继承了myStruts-Default，执行Action时就会自动先执行rlStack中的所有拦截器  -->  <default-interceptor-ref name=*"rlStack"*/>    </package>  <package name=*"person"* extends=*"myStruts-Default"* namespace=*"/person"*>    <!-- 拦截器配置 -->  <action name=*"renliang"* class=*"com.rl.action.PersonAction"* method=*"save"*>  <result name=*"success"*>/success.jsp</result>  <result name=*"input"*>/form.jsp</result>  </action>  <action name=*"delete"* class=*"com.rl.action.PersonAction"* method=*"delete"*>  <result name=*"success"*>/success.jsp</result>  <result name=*"input"*>/form.jsp</result>  </action>  </package>  <package name=*"person1"* extends=*"myStruts-Default"* namespace=*"/person1"*>    <!-- 拦截器配置 -->  <action name=*"renliang1"* class=*"com.rl.action.PersonAction1"* method=*"save"*>  <result name=*"success"*>/success.jsp</result>  <result name=*"input"*>/form.jsp</result>  </action>    </package> |

## 4.排除Action的拦截

要想排除被拦截的Action的执行方法，拦截器需要继承MethodFilterInterceptor

|  |
| --- |
| **public** **class** MyInterceptor1 **extends** MethodFilterInterceptor {  @Override  **protected** String doIntercept(ActionInvocation invocation) **throws** Exception {  System.*out*.println("前置拦截执行1....");  //让拦截器向下走并且返回结果代码  String result = invocation.invoke();  System.*out*.println("后置拦截执行1...");  **return** result;  }  } |

配置Action的排除

|  |
| --- |
| <package name=*"myStruts-Default"* extends=*"struts-default"* abstract=*"true"*>  <interceptors>  <!--  配置我们自己的拦截器  -->  <interceptor name=*"myInterceptor"* class=*"com.rl.interceptor.MyInterceptor"*></interceptor>  <interceptor name=*"myInterceptor1"* class=*"com.rl.interceptor.MyInterceptor1"*></interceptor>  <!--  拦截器栈：是把多个拦截器集中到一起统一被引用  -->  <interceptor-stack name=*"rlStack"*>  <!--  拦截器栈中要引用已经配置好的拦截器或者拦截器栈  -->  <interceptor-ref name=*"defaultStack"*></interceptor-ref>  <interceptor-ref name=*"myInterceptor"*></interceptor-ref>  <interceptor-ref name=*"myInterceptor1"*></interceptor-ref>  </interceptor-stack>  </interceptors>  <!--  把rlStack作为默认拦截器，只要继承了myStruts-Default，执行Action时就会自动先执行rlStack中的所有拦截器  -->  <default-interceptor-ref name=*"rlStack"*/>    </package>  <package name=*"person"* extends=*"myStruts-Default"* namespace=*"/person"*>    <!-- 拦截器配置 -->  <action name=*"renliang"* class=*"com.rl.action.PersonAction"* method=*"save"*>  <result name=*"success"*>/success.jsp</result>  <result name=*"input"*>/form.jsp</result>  </action>  <action name=*"delete"* class=*"com.rl.action.PersonAction"* method=*"delete"*>  <!-- 当前Action需要主动管理默认拦截器栈 -->  <interceptor-ref name=*"rlStack"*>  <!--  指定要排除的拦截器  name：要排除的拦截器的名字.exludeMethods  元素：要排除的方法  -->  <param name=*"myInterceptor1.excludeMethods"*>delete</param>  </interceptor-ref>  <result name=*"success"*>/success.jsp</result>  <result name=*"input"*>/form.jsp</result>  </action>  </package> |

# 13.struts2的上传和下载

## 1.单文件上传

上传文件对表单的要求：

* 表单要使用post方式提交
* 表单的enctype是multipart/form-data
* 表单中要有file类型的input文本域

Struts上传也是基于拦截器，底层还是使用commons-fileupload组件

Struts上传的步骤

1. 建立表单

|  |
| --- |
| <form action=*"*${pageContext.request.contextPath }*/upload/renliang"* method=*"post"* enctype=*"multipart/form-data"*>  姓名：<input type=*"text"* name=*"username"*><br>  文件：<input type=*"file"* name=*"upload"*><br>  <input type=*"submit"*>  </form> |

（2）创建Action

注意：

接收的文件属性：和表单中文本域file类型input的name一致

接收的文件名字：file的name+FileName固定写法

接收的文件MIME类型：file的name+ContentType

|  |
| --- |
| **public** **class** UploadAction **extends** ActionSupport{    **private** String username;    /\*\*  \* 要接收的文件，命名需要和表单中的file类型的input的name一致  \*/  **private** File upload;    /\*\*  \* 文件名的接收 File属性名FileName:固定写法  \*/  **private** String uploadFileName;    /\*\*  \* 获得上传文件的MIME类型，File属性名字ContentType：固定写法  \*/  **private** String uploadContentType;  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** File getUpload() {  **return** upload;  }  **public** **void** setUpload(File upload) {  **this**.upload = upload;  }  **public** String getUploadFileName() {  **return** uploadFileName;  }  **public** **void** setUploadFileName(String uploadFileName) {  **this**.uploadFileName = uploadFileName;  }  **public** String getUploadContentType() {  **return** uploadContentType;  }  **public** **void** setUploadContentType(String uploadContentType) {  **this**.uploadContentType = uploadContentType;  }      **public** String upload() **throws** Exception{  //获得servletContext  ServletContext sc = ServletActionContext.*getServletContext*();  //获得服务的绝对路径  String realPath = sc.getRealPath("/");  realPath = realPath + "upload\\"+uploadFileName;  //定义输入输出流  InputStream in = **new** FileInputStream(upload);  OutputStream out = **new** FileOutputStream(realPath);  **int** len = -1;  **byte**[] bs = **new** **byte**[1024];  **while**((len = in.read(bs)) != -1){  out.write(bs, 0, len);  }  out.close();  in.close();  **return** **super**.*SUCCESS*;  }        } |

## 2.文件上传类型的验证

在上传时要验证文件类型和大小

1. 验证文件的类型

|  |
| --- |
| <package name=*"upload"* extends=*"struts-default"* namespace=*"/upload"*>    <!-- 拦截器配置 -->  <action name=*"renliang"* class=*"com.rl.action.UploadAction"* method=*"upload"*>  <!--  主动引用默认拦截器栈  -->  <interceptor-ref name=*"defaultStack"*>  <!-- 设置上传拦截器fileUpload.allowedExtensions, 不要使用allowedExtensionsSet -->  <param name=*"fileUpload.allowedExtensions"*>.png,.txt</param>  </interceptor-ref>  <result name=*"success"*>/success.jsp</result>  <result name=*"input"*>/form.jsp</result>  </action>    </package> |

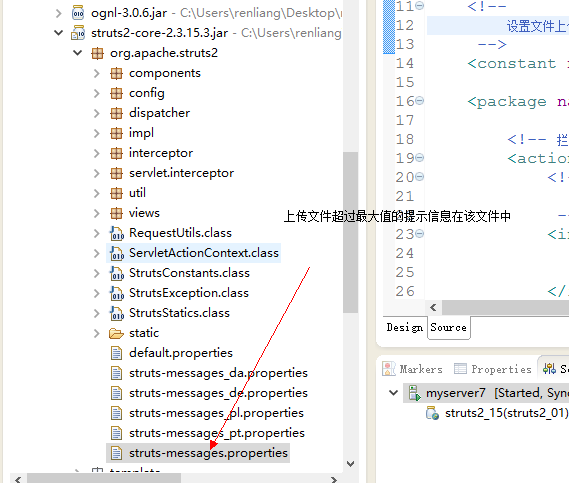
## 3.文件上传的大小验证

Struts2默认的上传文件的配置在default.properties的文件中

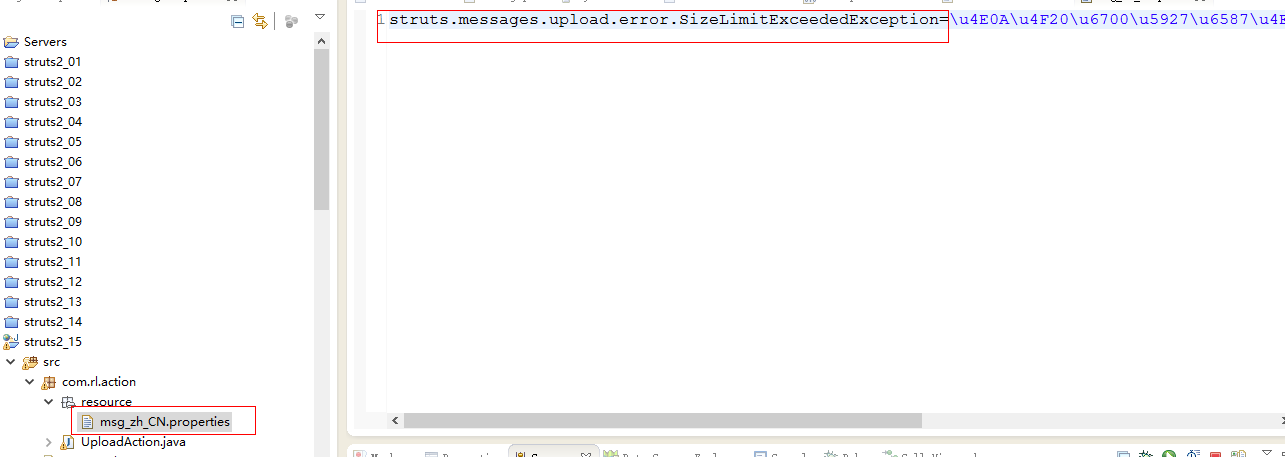
由struts.multipart.maxSize=2097152控制

我们发现展示并不是中文，我们使用国际化来处理

文件超过最大值的提示信息是在struts\_message.propertise中，我们只要通过国际化文件覆盖它即可



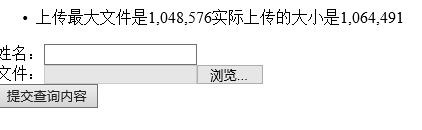
我们创建国际化文件



在国际化文件中{0}是去文件最大值，{1}取我们上次的文件的大小

然后再struts.xml中加载资源文件

|  |
| --- |
| <constant name=*"struts.custom.i18n.resources"* value=*"com/rl/action/resource/msg"*></constant> |



## 4.多文件的上传

多文件上传Action中把File属性以及和file属性相关的上传文件都变成数组，在execute中循环上传即可

|  |
| --- |
| **public** **class** UploadsAction **extends** ActionSupport{    **private** String username;    /\*\*  \* 要接收的文件，命名需要和表单中的file类型的input的name一致  \*/  **private** File[] upload;    /\*\*  \* 文件名的接收 File属性名FileName:固定写法  \*/  **private** String[] uploadFileName;    /\*\*  \* 获得上传文件的MIME类型，File属性名字ContentType：固定写法  \*/  **private** String[] uploadContentType;    **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** File[] getUpload() {  **return** upload;  }  **public** **void** setUpload(File[] upload) {  **this**.upload = upload;  }  **public** String[] getUploadFileName() {  **return** uploadFileName;  }  **public** **void** setUploadFileName(String[] uploadFileName) {  **this**.uploadFileName = uploadFileName;  }  **public** String[] getUploadContentType() {  **return** uploadContentType;  }  **public** **void** setUploadContentType(String[] uploadContentType) {  **this**.uploadContentType = uploadContentType;  }    **public** String upload() **throws** Exception{  //获得servletContext  ServletContext sc = ServletActionContext.*getServletContext*();  //获得服务的绝对路径  String realPath = sc.getRealPath("/");  **for**(**int** i = 0; i < upload.length; i++){  realPath = realPath + "upload\\"+uploadFileName[i];  //定义输入输出流  FileUtils.*copyFile*(upload[i], **new** File(realPath));  }  **return** **super**.*SUCCESS*;  }    } |

## 5.文件下载

Struts文件下载对动作类Action的要求：

在Action之中必须提供三个属性：

//提供一个输入流的属性，名字叫inputStream固定

private InputStream inputStream;

//定义文件的大小

private int filelength;

//定义文件名

private String fileName;

|  |
| --- |
| **public** **class** DownAction **extends** ActionSupport{    //提供一个输入流的属性，名字叫inputStream固定  **private** InputStream inputStream;    //定义文件的大小  **private** **int** filelength;    //定义文件名  **private** String fileName;  **public** InputStream getInputStream() {  **return** inputStream;  }  **public** **void** setInputStream(InputStream inputStream) {  **this**.inputStream = inputStream;  }  **public** **int** getFilelength() {  **return** filelength;  }  **public** **void** setFilelength(**int** filelength) {  **this**.filelength = filelength;  }  **public** String getFileName() {  **return** fileName;  }  **public** **void** setFileName(String fileName) {  **this**.fileName = fileName;  }    **public** String down() **throws** Exception{  ServletContext sc = ServletActionContext.*getServletContext*();  String realPath = sc.getRealPath("/upload/1.png");  //实例化输入流  inputStream = **new** FileInputStream(realPath);  //给fileName赋值  fileName = FilenameUtils.*getName*(realPath);  //对文件名做编码  fileName = URLEncoder.*encode*(fileName, "UTF-8");  //给文件大小赋值  filelength = inputStream.available();  **return** **super**.*SUCCESS*;  }  } |

Action的配置

|  |
| --- |
| <action name=*"renliang"* class=*"com.rl.action.DownAction"* method=*"down"*>  <result name=*"success"* type=*"stream"*>  <!-- 指定Action中输入流变量 -->  <param name=*"inputName"*>inputStream</param>  <!-- 设置响应的消息头 Content-Disposition -->  <param name=*"contentDisposition"*>attachment;filename=${fileName}</param>  <!-- 使用下载的方式来返回结果 -->  <param name=*"contentType"*>application/octet-stream</param>  <!-- 配置文件的大小 -->  <param name=*"contentLength"*>${filelength}</param>  </result>  </action> |

# 14.ognl表达式

OGNL是Object Graphic Navigation Language（对象图导航语言）的缩写，

它是一个开源项目。 Struts2框架使用OGNL作为默认的表达式语言。

作用：取值，获取javaBean中的属性，获取List或者数组元素，获得map的键值对，还可以执行逻辑运算。

要求：我们必须把ognl表达式写在struts的标签中。

## ognl对普方法的调用

|  |
| --- |
| <%-- 在<s:properties >的value属性中""内部是ognl表达式，如果要输出字符串要加''  --%>  <s:property value=*"'renliang'"*/>  <%-- 在<s:properties >的value属性中""内部是ognl表达式，可以使用java的api  --%>  <s:property value=*"'renliang'.toUpperCase()"*/> |

## ognl对静态变量和静态方法的调用

静态变量和静态方法的调用都要使用@类的全路径@[静态变量或静态方法]，但是如果是静态方法的调用必须要先开启

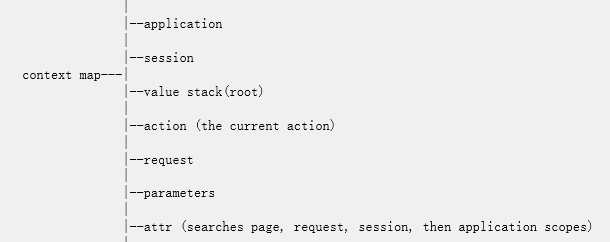
|  |
| --- |
| <!-- 开启ognl对静态方法的调用 -->  <constant name=*"struts.ognl.allowStaticMethodAccess"* value=*"true"*></constant> |

先须态变量和静态方法的调用ertiseptorilter

|  |
| --- |
| <s:property value=*"@java.lang.Integer@MAX\_VALUE"*/>  <s:property value=*"@java.lang.Math@abs(-100)"*/> |

## ActionContext

context map:是每次请求访问时存储数据的对象，每一次请求都会创建context map，我们可以把数据来存放到map中。



|  |  |  |
| --- | --- | --- |
| Key | Value | 说明 |
| Value stack | List集合 | 以栈的方式来存储数据 |
| Request | Map<String,Object>结构 | 以键值对的方式存储请求范围的数据 |
| session | Map<String,Object>结构 | 以键值对的方式存储会话范围的数据 |
| Application | Map<String,Object>结构 | 以键值对的方式存储应用范围的数据 |
| Action | Object类型 | 当前访问的Action动作类对象 |
| Parameters | Map<String,Object>结构 | 存储请求的参数 |
| Attr | Map<String,Object>结构 | 根据key从page，request，session，application范围依次查找属性的值。 |

每一个请求访问时都会创建一个contextMap的对象，ValueStack和contextmap的数据是关联的可以相互转化

我们的数据采用两个存储的结构来存储一个valueStack（set（map）），contextMap（map）

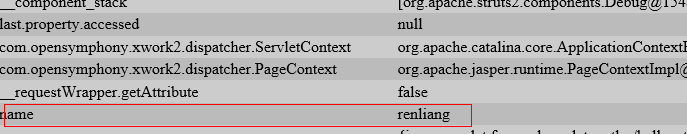
### 1在context map中存取数据

在请求范围内存储数据

|  |
| --- |
| @Override  **public** String execute() **throws** Exception {  System.*out*.println(username);  //获得动作类的上下文ActionContext包含了contextmap和valuestack  ActionContext ac = ServletActionContext.*getContext*();  //在contextmap中存储数据，默认的相当于request,因为ActionContext的生命周期和request一样都是一次请求  ac.put("name", "renliang");  **return** **super**.execute();  } |

在contextmap中取数据要使用#

|  |
| --- |
| <s:property value=*"#name"*/>  <s:debug></s:debug> |

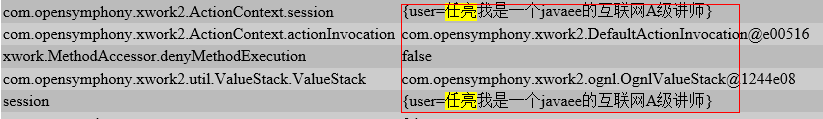


在会话范围存储数据

|  |
| --- |
| @Override  **public** String execute() **throws** Exception {  System.*out*.println(username);  //获得动作类的上下文ActionContext包含了contextmap和valuestack  ActionContext ac = ServletActionContext.*getContext*();  //在contextmap中存储数据，默认的相当于request,因为ActionContext的生命周期和request一样都是一次请求  ac.put("name", "renliang");  //把数据存储在会话范围  ac.getSession().put("user", "任亮我是一个javaee的互联网A级讲师");  **return** **super**.execute();  } |

取值时语法

|  |
| --- |
| <s:property value=*"#session.user"*/>  <s:debug></s:debug> |

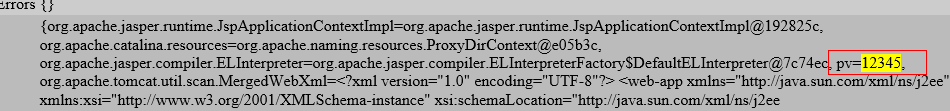


在应用级别的存储数据

|  |
| --- |
| @Override  **public** String execute() **throws** Exception {  System.*out*.println(username);  //获得动作类的上下文ActionContext包含了contextmap和valuestack  ActionContext ac = ServletActionContext.*getContext*();  //在contextmap中存储数据，默认的相当于request,因为ActionContext的生命周期和request一样都是一次请求  ac.put("name", "renliang");  //把数据存储在会话范围  ac.getSession().put("user", "任亮我是一个javaee的互联网A级讲师");  //在应用范围内存数据  ac.getApplication().put("pv", 12345);  **return** **super**.execute();  } |

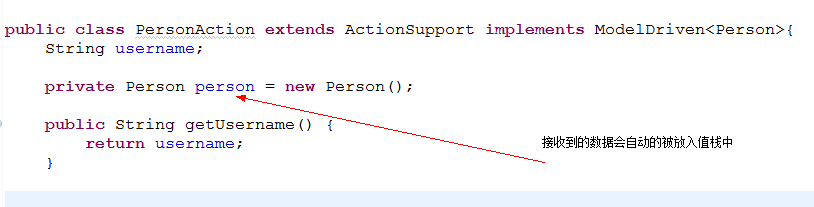
取数据

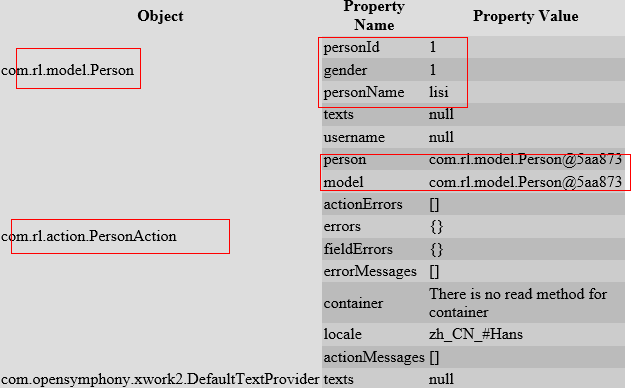
|  |
| --- |
| <s:property value=*"#application.pv"*/>  <s:debug></s:debug> |



### 在值栈中存取数据

会把请求传递的参数存储在值栈中，同时把Action对象也压入栈中，属性对象在上面Action在下面，对象的值既可以从栈顶取，也可以从Action的person属性中来取

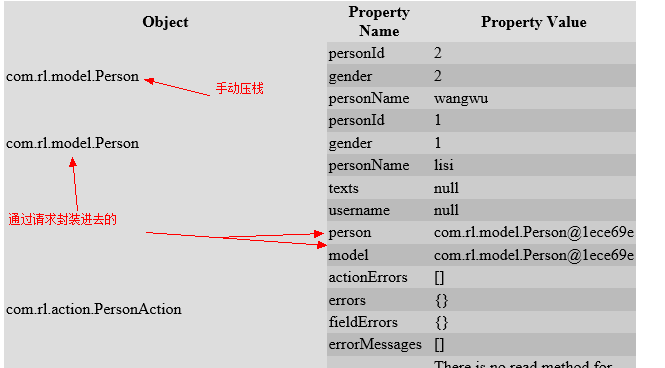




从值栈中取数据ognl表达式不需要#

|  |
| --- |
| <s:property value=*"username"*/>  <s:property value=*"person.personId"*/>  <s:property value=*"person.personName"*/>  <s:property value=*"person.gender"*/> |

手动通过栈放入的数据也会放入栈顶，但是在Action对象中不会存储在相应属性数据



# 15.struts2的标签库

## 分支判断

首先要引入struts2的标签库

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

If elseif else使用 test内部是ognl表达式取值

|  |
| --- |
| <s:if test=*"#age* < *16"*>  <s:property value=*"'小孩'"*/>  </s:if>  <s:elseif test=*"#age >= 16 && #age* < *18"*>  <s:property value=*"'未成年'"*/>  </s:elseif>  <s:else>  <s:property value=*"'成年人'"*/>  </s:else> |

## 循环

使用<s:iterator>标签

属性：

Value是从ActionContext中获取的集合key，不需要#

Var：是每次从集合中取值赋值的变量

使用的时候需要加#，#str

|  |
| --- |
| <h3>循环数组</h3>  <s:iterator value=*"arrStr"* var=*"str"*>  <s:property value=*"#str"*/>  </s:iterator>  <hr>  <h3>循环list</h3>  <s:iterator value=*"list"* var=*"str"*>  <s:property value=*"#str"*/>  </s:iterator>  <hr>  <h3>循环map</h3>  <s:iterator value=*"map"* var=*"mapObj"*>  <s:property value=*"#mapObj.key"*/>------><s:property value=*"#mapObj.value"*/><br>  </s:iterator> |

对于对象集合循环

Status属性可以给循环设置参数

变量：

Index：当前循环的索引号，从0开始

Count：当前循环的顺序号，从1开始

First：是否是第一行

List:是否是最后一行

odd：是否是奇数

even:是否是偶数

begin：从数字几开始

end：到数字几结束

step：步长

|  |
| --- |
| <table border=*"1"*>  <tr>  <th>id</th>  <th>姓名</th>  <th>性别</th>  <th>索引</th>  <th>序号</th>  <th>是否首行</th>  <th>是否尾行</th>  <th>奇数</th>  <th>偶数</th>  </tr>  <s:iterator value=*"personList"* var=*"person"* status=*"status"*>  <tr bgcolor=*'*<s:property value=*"#status.odd?'#c3f3c3':'#f3c3f3'"*/>*'*>  <td><s:property value=*"#person.personId"*/></td>  <td><s:property value=*"#person.personName"*/></td>  <td><s:property value=*"#person.gender == 1?'男':'女'"*/></td>q  <td><s:property value=*"#status.index"*/></td>  <td><s:property value=*"#status.count"*/></td>  <td><s:property value=*"#status.first"*/></td>  <td><s:property value=*"#status.last"*/></td>  <td><s:property value=*"#status.odd"*/></td>  <td><s:property value=*"#status.even"*/></td>      </tr>  </s:iterator>    </table> |



## 3.输出标签

<S:property>

属性：

Value：用于通过ognl表达式来取值

Default：如果value值是空就给一个默认值

EscapeHtml：是否被浏览器解析，默认是true不解析，false 是解析

|  |
| --- |
| <s:property value=*"#name"*/>  <s:property value=*"#name1"* default=*"空值"*/>  <s:property value=*"'*<a href>*任亮*</a>*'"* escapeHtml=*"false"*/> |

## 4.日期输出标签

<s:date>

属性：

Name：取日期的ognl表达式的值

Format：要展示的日期的格式

|  |
| --- |
| <h3>日期输出标签</h3>  <s:date name=*"#ctime"* format=*"yyyy-MM-dd HH:mm:ss"*/> |

## 5.页面动态包含

<s:action>

属性：

Name：要请求的Action

ExecuteResult：是否展示Action的执行结果，true是展示，false不展示

|  |
| --- |
| <h3>页面包含</h3>  <s:action name=*"hello1"* executeResult=*"true"*></s:action> |

6.超链接标签

<s:a>

属性：

Action：要链接的动作类的名称，标签会在Action的值后面自动的加上后缀

<s:param>是<s:a>内部的元素，param主要是给a链接赋予参数的，可以自动的对中文编码

|  |
| --- |
| <h3>超链接</h3>  <s:a action=*"hello"*>我是a链接  <s:param name=*"username"* value=*"'任亮'"*></s:param>  <s:param name=*"job"* value=*"'teacher'"*></s:param>  </s:a> |

|  |
| --- |
| http://localhost:8080/struts2\_18/hello.action?username=%E4%BB%BB%E4%BA% |

# 16.struts2对el表达式的支持

* El表达式

${name}:调用pageContext.findAttribute(“name”),从页面，请求，会话，应用范围去查找key=name对应的值

|  |
| --- |
| <table border=*"1"*>  <tr>  <th>id</th>  <th>姓名</th>  <th>性别</th>  <th>索引</th>  <th>序号</th>  <th>是否首行</th>  <th>是否尾行</th>  <th>奇数</th>  <th>偶数</th>  </tr>  <s:iterator value=*"personList"* var=*"p"* status=*"status"*>  <tr bgcolor=*"*${status.odd?'#c3f3c3':'#f3c3f3'}*"*>  <td>${p.personId }</td>  <td>${p.personName }</td>  <td>${p.gender == 1?'男':'女' }</td>  <td>${status.index}</td>  <td>${status.count}</td>  <td>${status.first}</td>  <td>${status.last}</td>  <td>${status.odd}</td>  <td>${status.even }</td>  </tr>  </s:iterator>    </table> |



Struts中el表达式${name}查找范围依次按着如下顺序查找： 请求， 值栈，contextMap, 会话，应用