# 开始

## 学习Struts2需具备的知识

* Java
* Filters、JSP、Tag Libraries
* JavaBeans
* HTML and HTTP
* Web Containers（比如Tomcat）
* XML

## Struts的发展历程

Struts1一经推出就得到了世界上JavaWeb开发者的拥护。经过这么多年的发展，它已经成为了世界上使用最广泛的MVC框架，拥有丰富的开发人群。

随着Web技术的不断发展和Web应用的不断扩大，Struts1的出现了诸多缺陷和局限性，主要表现为以下几点：

1、所支持的表现层技术比较单一。

只支持JSP作为表现层，没有提供对目前流行的FreeMarker、Velocity等表现层的支持。

2、与ServletAPI耦合严重，难预测试。

它完全基于ServletAPI，业务逻辑控制器内充满了大量的ServletAPI。一旦脱离Web容器，Action的测试将非常困难。

3、属于侵入式设计

影响了代码重用，一旦系统需要重构，这种侵入式设计将会带来很多的问题。

## Struts2概述

Struts2的体系与Struts1体系的差别非常大，因为Struts2使用了WebWork的设计核心，而不是Struts1的设计核心。Struts2中大量使用拦截器来处理用户的请求，从而允许用户的业务逻辑控制器与Servlet API分离。



Struts2对Struts1进行了巨大的改进。主要表现在如下几个方面：

1、在Action的实现方面：Struts1要求必须统一扩展自Action类，而Struts2中可以是一个普通的POJO。

2、线程模型方面：Struts1的Action工作在单例模式，一个Action的实例处理所有的请求。Struts2的Action是一个请求对应一个实例。没有线程安全方面的问题。

3、Servlet依赖方面：Struts1的Action依赖于Servlet API，比如Action的execute方法的参数就包括request和response对象。这使程序难于测试。Struts2中的Action不再依赖于Servlet API，有利于测试。

4、封装请求参数：Struts1中强制使用ActionForm对象封装请求的参数。Struts2可以选择使用POJO类来封装请求的参数，或者直接使用Action的属性。

5、表达式语言方面：Struts1中整合了EL，但是EL对集合和索引的支持不强，Struts2整合了OGNL（Object Graph NavigationLanguage）。

6、绑定值到视图技术：Struts1使用标准的JSP，Struts2使用“ValueStack”技术。

7、类型转换：Struts1中的ActionForm基本使用String类型的属性。Struts2中使用OGNL进行转换，可以更方便的使用。

8、数据校验：Struts1中支持覆盖validate方法或者使用Validator框架。Struts2支持重写validate方法或者使用XWork的验证框架。

9、Action执行控制的对比：Struts1支持每一个模块对应一个请求处理，但是模块中的所有Action必须共享相同的生命周期。Struts2支持通过拦截器堆栈为每一个Action创建不同的生命周期。

# 一个简单的示例

## 搭建开发环境

1. 从<http://struts.apache.org/>下载发行包。或者点击以下链接[http://archive.apache.org/dist/struts/binaries/struts-2.1.](http://archive.apache.org/dist/struts/binaries/struts-2.1.8-all.zip)[8-all.zip](http://archive.apache.org/dist/struts/binaries/struts-2.1.8-all.zip)直接下载struts2的2.1.8完整发行版本。完整发行版本中包含了struts2-core-\*.jar、依赖的其他jar包、示例程序、一份HTML格式的文档和源代码。
2. Struts2依赖

* Servlet API 2.4
* JSP API 2.0
* Java 5

1. 设置你的web应用文件结构：

|  |
| --- |
| **文件目录结构** |
| /project/  /project/META-INF/  /project/WEB-INF/  /project/WEB-INF/classes/  /project/WEB-INF/lib/  /project/WEB-INF/lib/minimum JARs + any plugin JARs + plugin dependencies  /project/WEB-INF/web.xml |

拷贝相应的jar包到/project/WEB-INF/lib/目录下：

* + 必要的JARs（运行struts2最小jar包集合）
  + 其他的与struts集成的插件
  + 集成插件所依赖的jar包

1. 安装必要的JARs及配置文件

|  |  |
| --- | --- |
| **文件名** | **描述** |
| struts2-core-\*.jar |  |
| xwork-core-\*.jar |  |
| ognl.jar | 框架所采用的表达式语言。Object Graph Navigation Language(OGNL) |
| freemarker.jar | UI标签的模板 |
| commons-logging.jar | 日志 |
| commons-fileupload-1.2.1.jar |  |
| commons-io-1.4.jar |  |

1. 配置web.xml。注册一个过滤器来使你的框架起作用。

|  |
| --- |
| **web.xml** |
| <?xml version="1.0"?>  <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  "http://java.sun.com/dtd/web-app\_2\_3.dtd">  <web-app>  <display-name>My Application</display-name>  <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> |

1. 在/project/WEB-INF/classes/下创建一个struts2的配置文件骨架，名字为struts.xml

|  |
| --- |
| **struts.xml** |
| <!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"  "http://struts.apache.org/dtds/struts-2.0.dtd">  <struts><!-- Configuration for the default package. -->  <package name="default" extends="struts-default">    </package>  </struts> |

## 示例应用“Hello World”

1、我们需要一个JSP页面来显示信息

|  |
| --- |
| **HelloWorld.jsp** |
| <%@ taglib prefix="s" uri="/struts-tags" %>  <html>  <head>  <title>Hello World!</title>  </head>  <body>  <h2><s:property value="message" /></h2>  </body>  </html> |

2、创建一个动作类来建立消息

|  |
| --- |
| **HelloWorld.java** |
| package cn.itcastl;  import com.opensymphony.xwork2.ActionSupport;  public class HelloWorld {  public static final String MESSAGE = "Struts2已经启动并正常运行";  public String execute() throws Exception {  setMessage(MESSAGE);  return “success”;  }  private String message;  public void setMessage(String message){  this.message = message;  }  public String getMessage() {  return message;  }  } |

3、利用配置文件将它们整合。编辑struts.xml，输入以下信息

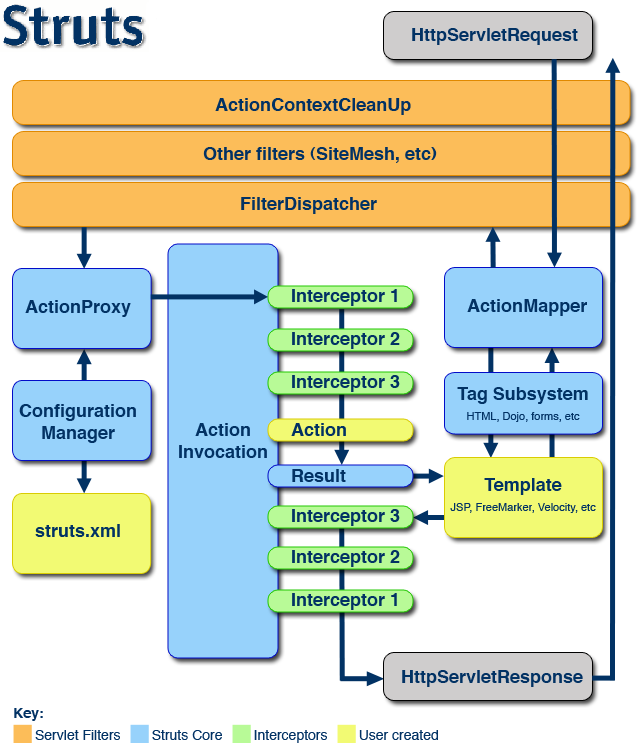
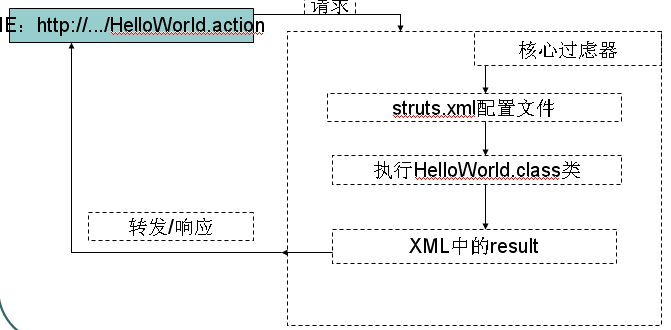
|  |
| --- |
| **struts.xml** |
| <!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"  "http://struts.apache.org/dtds/struts-2.0.dtd">  <struts>  <package name="tutorial" extends="struts-default">  <action name="HelloWorld" class=" cn.itcast.HelloWorld">  <result name=”success”>/HelloWorld.jsp</result>  </action>  <!-- Add your actions here -->  </package>  </struts> |

4、启动你的服务器，访问<http://localhost:8080/project/HelloWorld.action> 页面上会输出"Hello World"

## 示例应用程序分析

当你的浏览器向服务器发送了一个<http://localhost:8080/project/HelloWorld.action>请求：

1. 容器从客户端收到请求HelloWorld.action，按照web.xml中的设置，容器发现所有的请求都必须经过org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter过滤，当然\*.action的请求也在过滤范围之内。过滤器是进入Struts2框架的入口点。
2. Struts框架发现一个名字为HelloWorld的动作请求，正好与struts.xml配置文件中的HelloWorld动作相匹配，于是框架实例化对应的动作类cn.itcast.HelloWorld 并调用其execute方法。
3. Execute方法设置了消息的内容并返回success字符串。框架检查struts.xml文件来决定返回success时该有哪个结果页面来呈现，因此框架呈现了HelloWorld.jsp页面。
4. 当HelloWorld.jsp被处理时，<s:property value="message" />标签将调用cn.itcast.HelloWorld类中的getter方法getMessage,并将该方法的返回值显示在页面上。
5. 一个纯粹的HTML页面相应给客户端（浏览器）。



图表 1Struts2架构图

# Struts2配置

绝大多数struts2应用程序都包含两个配置文件：

* struts.xml
* struts.properties

这些文件是我们对struts2应用程序里的各种动作进行配置的地方。struts.properties是可选的，因此struts.xml对我们来说显得重要一些，控制器的配置，一般就是指配置struts.xml。

## Struts2的动作处理流程

Struts2的控制器是由一个过滤器来担当的，因为struts支持的功能非常多，过滤器程序因此也会变得很复杂。Struts2采用了一种思路来简化这种复杂性：在它的过滤器调度程序里把整个任务分解成一系列被称为“拦截器”的组件。

### 动作方法

在请求某个动作时，执行的动作类里面的那个方法就是动作方法。

动作方法的命名规则很简单，只要它是一个合法的Java方法名称即可。

动作方法还应该返回一个String类型的值，这个值告诉struts2应该把控制权转交到哪里。

如果一个动作的配置中没有明确指定要运行的动作方法，则struts2中默认的动作方法名为execute作为默认方法名。

注：开发阶段，我们会经常修改struts.xml文件，因为struts.xml是在第一启动服务器时就被加载了，因此日后的改动在没有重起服务器的情况下不会生效，这对于开发程序来说不是很方便。

解决办法，就是我们希望在每次修改完struts.xml文件后，在不重起服务器的前提下让其改动生效。我们应该在strtus.xml中增加一参数，如下：

<struts>

…

<constant name=”struts.devMode” value=”true”/>

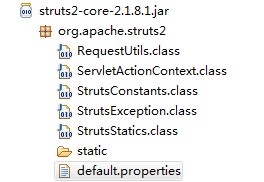
…

</struts>

### Struts.xml

Struts配置文件支持继承，默认的配置文件被打包在struts2-core-\*.jar（\*为版本号）中，名字为strut-default.xml，里面定义了很多bean、拦截器和结果类型（后面章节会详细讲解）。

同时打包的还有适用于所有struts应用程序设置的另一个配置文件，即default.properties，在struts2-core-\*.jar（\*为版本号）中的org.apache.struts2包中，如下图：

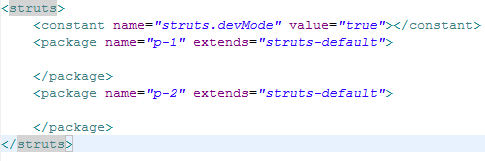


default.properties中的配置都是名值对的形式，可以在struts.xml中通过

<constant name=”*key*” value=”*value*”/>的形式来更改其默认行为。

#### package元素

struts在设计时就已经考虑到了模块化的问题，所以可以把各种动作分门别类地组织成不同的包。你不妨把一个这样的包想象为一个模块。一个典型的struts.xml文件可以有一个或多个包。

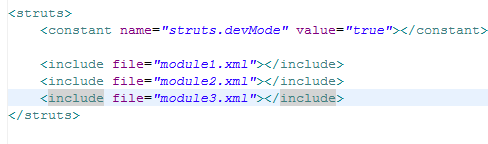


##### 属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 必须的 | 描述 |
| name | string | 无 | yes | 包的名称，它在同一个struts.xml文件里必须是唯一的 |
| extends | string | 无 | no | 这个包所扩展的父包名称.一般需要继承struts-default |
| namespace | string | / | no | 这个包的命名空间 |
| abstract | boolean | false | no | 用来表明这是不是一个抽象包（包中要求没有动作配置） |

#### include元素

一个大型的应用程序可能有许多包。为了让大型应用程序的strut.xml文件更易于管理，可以把它划分成几个较小的文件（最好是让每个小文件只包含一个包或只包含几个彼此相关的包），然后用include元素来引用那些小文件。

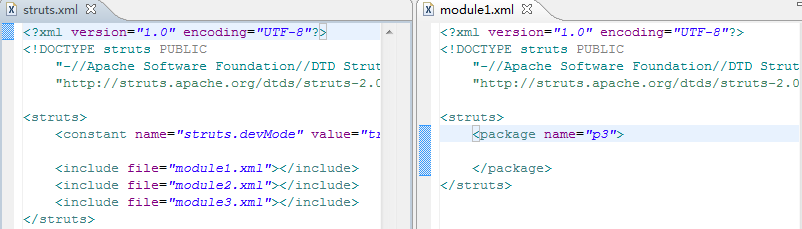


##### 属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 必须的 | 描述 |
| file | string | 无 | yes | 要包含的其它配置文件名称 |

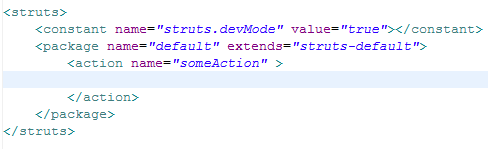
注：

被包含的每个配置小文件，都应该有相同的DOCTYPE元素和一个struts根元素。下面是一个例子：



#### action元素

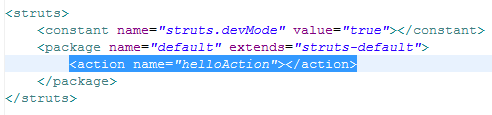
action元素嵌套在package元素内部，它表示一个动作。每个动作都必须有一个名字。一个包中可以有零个或者多个action元素。



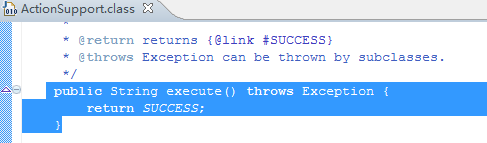
##### 属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 必须的 | 描述 |
| name | string | 无 | yes | 动作的名称 |
| class | string | com.opensymphony.xwork2.ActionSupport  （在xwork-core-\*.jar中） | no | 动作对应的动作类。若没有指定该属性，struts会自动实例化默认的类作为动作类。 |
| method | string | execute | no | 动作方法 |
| converter | string | 无 | no | 动作的类型转换器 |

最简单的写法如下：



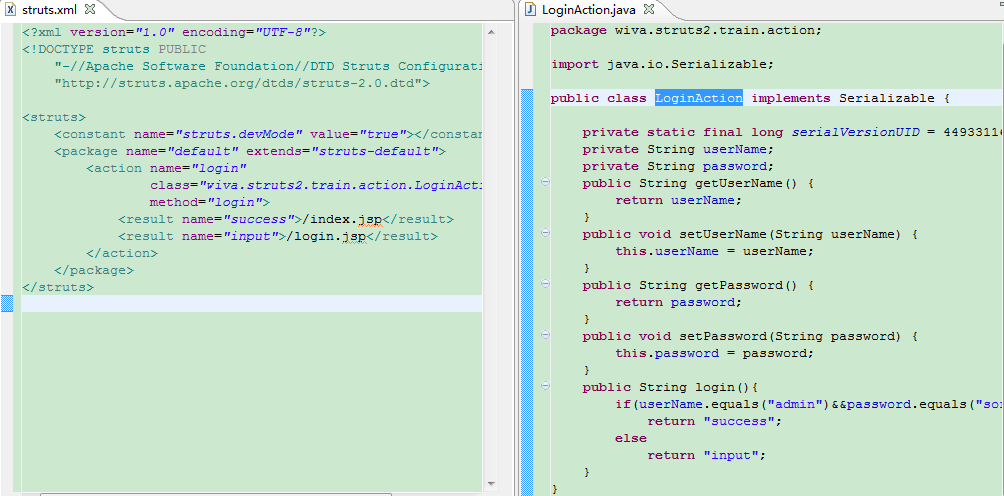
因未指定class和method属性，在struts在调用该动作的时候，会实例化com.opensymphony.xwork2.ActionSupport，并调用execute方法，该方法实现很简单，如下图：



#### result元素

<result>元素是<action>的一个子元素,它告诉struts你想在动作完成后把控制权转交到哪里。result元素对应着动作方法的返回值。

因为动作方法在不同的情况下可能会返回不同的值，所以同一个action元素可能会有多个result元素。如下图：



##### 属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 必须的 | 描述 |
| name | string | success | no | 对应动作方法的返回值 |
| type | string | dispatcher | no | 结果类型。struts定义的结果类型有10种。 |

最简单的写法如下：

<result>/index.jsp</result>

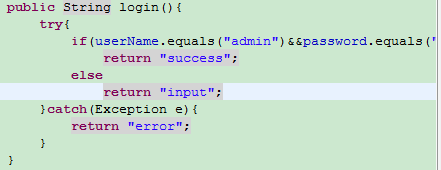
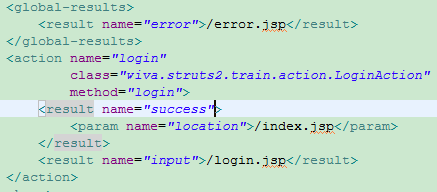
注：结果类型为dispatcher的result元素还有另外一种写法：



param的name取值来源于dispatcher结果类型的实现类的某个具有setter方法的属性。

#### global-results

一个package元素可以包含一个global-results元素，其中定义的结果是一些通用的结果。如果某个动作在它的动作声名里无法找到一个匹配结果，struts会搜索该元素中定义的结果。



##### 属性：

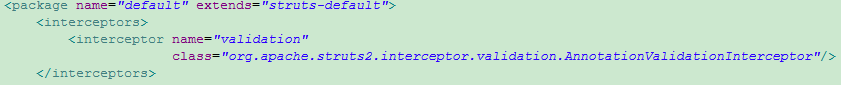
无，一旦使用该元素，必须在其内部定义一个或多个result元素。

注：如果在其他包中的动作也适用该包中定义的全局结果类型的话，需要采用继承的形式。

#### 与拦截器有关的元素

有5个与拦截器有关的元素可能出现在struts.xml文件里，它们是：interceptors、interceptor、interceptor-stack、interceptor-ref和default-interceptor-ref。

action元素必须包含一系列负责对动作对象进行处理的拦截器。在使用一个拦截器之前，你必须先在interceptors标签内使用interceptor注册一个拦截器，如下图：



图中定义了一个名字叫做validation的拦截器，并通过class属性指向该拦截器的处理类。

##### 属性

interceptors属性无

interceptor属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 必须的 | 描述 |
| name | string | 无 | yes | 拦截器名 |
| class | string | 无 | yes | 实现类的完整限定名 |

interceptor-stack：用于将定义的拦截器进行分组，它必须出现在interceptors元素中，其属性只有一个name，用于为该组起名，must。

interceptor-ref：用于定义某个拦截器组中的组员，它必须出现在interceptor- stack元素中，其属性只有一个name，用于引用之前已经定义的拦截器或小组的名称，must。

default-interceptor-ref：如果动作标签没有指定任何拦截器，则会默认使用该标签指定的拦截器或拦截器组，其属性只有一个name，用于引用之前已经定义的拦截器或小组的名称，must，默认值是defaultStack。

#### constant元素

用于在struts.xml文件里覆盖default.properties文件里的某个设置。

##### 属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 默认值 | 必须的 | 描述 |
| name | string | 无 | yes | default.properties文件中的key值 |
| value | string | 无 | yes | 新的value值 |

# 动作与结果

## 动作类

应用程序可以完成的每一个操作都可以称为一个动作。而动作要处理的业务逻辑需要在动作类中来完成。

一个示例：我们来检查用户输入的用户名和密码。加入这2项输入的内容为空，我们返回INPUT，否则返回SUCCESS，以用于验证用户是否输入了用户名和密码。代码如下：

|  |
| --- |
| **Logon.java** |
| package cn.sdxhce.tutorial;  import com.opensymphony.xwork2.ActionSupport;  public class Logon extends ActionSupport {  private String username;  private String password;  public String getUsername(){  return username;  }  public void setUsername(String username){  this.username = username;  }  public String getPassword(){  return password;  }  public void setPassword (String password){  this. password = password;  }  private boolean isInvalid(String value){  return (value == null||value.length()==0);  }  public String execute(){  if(isInvalid(username)) return INPUT;  if(isInvalid(password)) return INPUT;  return SUCCESS;  }  } |

动作类其实就是普通的Java类，它们可以由属性和方法，同时必须遵守下面这些规则：

* 每个属性必须有一个getter或setter方法。动作属性的名字必须遵守与JavaBean属性名同样的命名规则。
* 动作类必须有一个不带任何参数的构造方法。
* 每个动作类至少要有一个动作方法。
* 每个动作类中可以包含多个动作方法。

注：动作类并不是非得扩展自某个特定的父类或是必须实现某种接口。话虽如此，编写的绝大多数动作类都将扩展一个名为com.opensymphony.xwork2.ActionSupport，通过继承该类，就相当于间接实现了com.opensymphony.xwork2.Action接口。

如果实现了Action接口，那么可以直接使用以下常量（在Action接口中定义）：

|  |  |  |
| --- | --- | --- |
| 常量名 | 对应值 | 使用场景 |
| SUCCESS | success | 执行成功 |
| NONE | none | 执行成功但不响应任何视图 |
| ERROR | error | 执行不成功，出错 |
| INPUT | input | 输入验证失败时应该将输入视图再次返回给用户 |
| LOGIN | login | 用户没有登陆，并应该将登录视图显示给用户 |

## 动作属性值

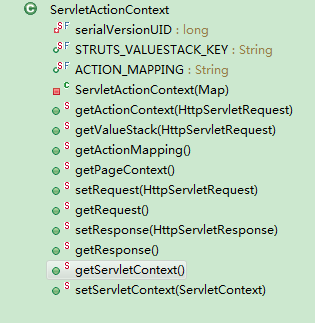
在struts.xml文件里的action元素可以包含任意个param元素。每个param元素对应一个动作类的属性值。

## 访问Servlet资源

由于struts2采用的是低侵入式设计，那么在struts.xml的动作类里要访问serlvet的相关资源可以通过以下2种方式：

* 通过使用org.apache.struts2.ServletActionContext类
* 通过实现\*Aware接口

## ServletActionContext：



其中定义的方法均为静态的，因此我们可以很方便地去获取到与Servlet有关的资源。

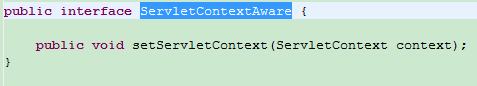
## Aware接口

struts2提供了4个分别用来访问ServletContext、HttpServletRequest、HttpServletResponse和HttpSession对象的接口，只要根据自己的具体需要有选择地实现它们就可以了，这4个接口分别是：

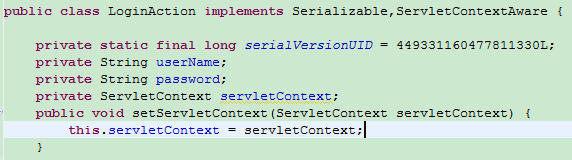
* org.apache.struts2.util.ServletContextAware
* org.apache.struts2.interceptor.ServletRequestAware
* org.apache.struts2.interceptor.ServletResponseAware
* org.apache.struts2.interceptor.SessionAware

### ServletContextAware

如果要从动作类内部访问ServletContext对象，可以实现该接口，它的定义如下：

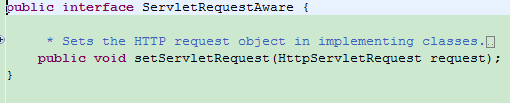


在调用一个动作时，struts将检查相关的动作类是否实现了该接口。如果是，struts将在填充动作属性之前先调用该动作的setServletContext方法，并传递ServletContext对象后再执行动作方法。在setServletContext方法内，你需要把ServletContext对象赋给一个类变量，如下图：



### ServletRequestAware

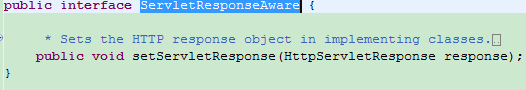
如果要从动作类内部访问HttpServletRequest对象，可以实现该接口，它的定义如下：



略

### ServletResponseAware

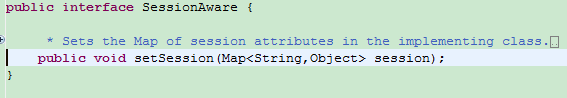
如果要从动作类内部访问HttpServletResponse对象，可以实现该接口，它的定义如下：



略

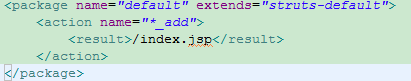
### SessionAware

SessionAware接口与以上3个接口有些小区别。实现该接口并不能将当前的HttpSession对象提供给你，你得到的是一个java.util.Map类型。



## 通配符映射及动态方法调用

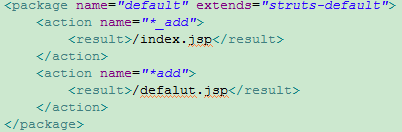
1、“\*”即为struts2中的通配符。



以上说明，只要你请求的动作名以“\_add”结尾，都会转向index.jsp

注：

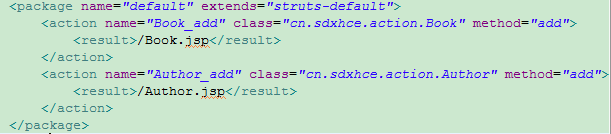
如果struts找到的带有通配符的匹配不止一个，它最匹配的那个将胜出。



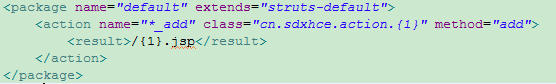
若请求动作studet\_add，则第一个动作将胜出。

如果找到了多个匹配，没有使用通配符的那个将胜出。

2、被通配符匹配到的字符串的子串可以用{1}来引用。如下图：

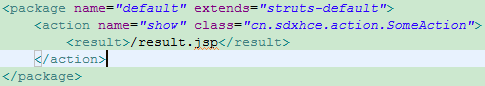


可以被替换为



### 动态方法调用

动态方法调用即使用“！”（棒字符）来动态地调用某个动作方法。



上图中没有给定任何动作方法，这样做的后果之一是SomeAction的execute方法将被调用。利用“棒字符”，你可以在同一个动作里调用不同的方法。例如你请求的URL是：show!edit.action，那么将调用edit方法。

注：最好不要使用动态方法调用，因为这会引发安全问题。你不会希望用户能够调用你没有公开的方法。

在默认的情况下，struts的动态方法调用处于开启状态，你可以通过改变default.properties文件里的struts.enable.DynamicMethodInvocation = false的形式来将其禁用。

## 使用内建的结果类型

Struts2框架提供了几个实现了com.opensymphony.xwork2.Result接口的结果类型,它们是:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 结果类型 | 取值 | 描述 | 常用参数 |
| 1 | Chain Result | chain |  |  |
| 2 | Dispatcher Result | dispatcher |  |  |
| 3 | FreeMarker Result | freemarker |  |  |
| 4 | HttpHeader Result | httpheader |  |  |
| 5 | Redirect Result | redirect |  |  |
| 6 | Redirect Action Result | redirectAction |  |  |
| 7 | Stream Result | stream |  |  |
| 8 | Velocity Result | velocity |  |  |
| 9 | XSL Result | xslt |  |  |
| 10 | PlainText Result | plainText | 常用于显示个别页面的源码 | locatioin:要显示的页面  charSet:编码 |

## 定制结果类型

# OGNL表达式

## 概述

OGNL是Object Graph Navigation Language(对象图形导航语言)的缩写。如要查看ONGL的全部文档，请访问<http://www.ognl.org>。

Struts2框架采用一个ognl上下文来解析OGNL表达式。在这个上下文中，有一个**根对象**栈（root object）。在使用ONGL表达式时，可以**直接通过名字**而不用任何其他标记的形式访问根对象栈中对象及其属性，而要访问上下文中的其他对象时必须使用标记**“#”**。

|  |
| --- |
| |  |--application  |  |--session  context map---|  |--value stack(root)  |  |--request  |  |--parameters  |  |--attr (searches page, request, session, then application scopes)  | |

## 基本用法

1、示例1：假设有2个对象在ognl上下文中：”foo”🡪foo 和”bar”🡪bar，foo存在于根对象栈中，并且是唯一的一个存在根对象栈内的对象。以下代码说明了OGNL进行处理的三种方式:

|  |
| --- |
| #foo.blah // returns foo.getBlah()  #bar.blah // returns bar.getBlah()  blah // returns foo.getBlah() because foo is the root |

2、示例2：假设在根对象栈内有2个对象Animal 和Person。这2个对象都拥有一个“name”属性，Animal对象还有一个“species”属性，同时Person对象也有一个“salary”属性。Animal对象位于栈的顶部，Person对象紧接其后。以下代码演示了发生的情况：

|  |
| --- |
| species // call to animal.getSpecies()  salary // call to person.getSalary()  name // call to animal.getName() because animal is on the top |

3、示例3：针对示例2，你也许有一些疑问，其实在对象栈中，OGNL支持对象的索引，位于栈顶的对象的索引为0，紧接其后的为1，依次类推，因此上面的代码等同于以下代码：

|  |
| --- |
| [0].name // call to animal.getName()  [1].name // call to person.getName() |

4、Struts2中的Action对象一般会放到ValueStack中，并且是位于OGNL的根中（即以上提到的“根对象栈”），因此访问动作对象的属性时，可以不用任何标记。但是，访问其他对象时，需要用到“#”。

|  |
| --- |
| **打印一个动作对象的属性** |
| <s:property value="postalCode"/> |

其他的（非根）在ActionContext中的对象访问需要用到“#”

|  |
| --- |
| <s:property value="#session.mySessionPropKey"/> or  <s:property value="#session['mySessionPropKey']"/> or  <s:property value="#request['myRequestPropKey']"/> or  <s:property value=’#request[“myRequestPropKey”]’/> |

ActionContext对象可以通过一个静态方法获得：

|  |
| --- |
| ActionContext.getContext().getSession().put(“mySessionPropKey”,mySessionObject); |

## 处理集合

我们经常在框架中使用到集合（Collections🡪Maps,Lists,and Sets），以下代码通过select标签展示了一些示例。

1、list创建语法：{e1,e2,e3}。以下代码创建了一个list，它包含“nam1”，“name2”,“name3”三个元素，并且“name2”是默认值：

|  |
| --- |
| <s:select label=”label” name=”name” list=”{‘name1’,’name2’,’name3’}” value=”%{‘name2’}” |

2、map创建语法:#{key1:value1,key2:value2}，以下代码创建了一个map，并且用“foo”与“foovalue”、“bar”与字符串“barvalue”进行了绑定：

|  |
| --- |
| <s:select label=”label” name=”name” list=”#{‘foo’:’foovalue’,’bar’:’barvalue’}” |

## 问题及处理

早期的JSP中的EL表达式可能使用#标记，这时struts2框架可能会发生错误，若要快速地修正，最好的办法就是让JSP禁止使用EL，我们来修改以下web.xml设置如下：

|  |
| --- |
| <jsp-config>  <jsp-property-group>  <url-pattern>\*.jsp</url-pattern>  <el-ignored>true</el-ignored>  </jsp-property-group>  </jsp-config> |

# Struts2标签

## 通用标签(Generic Tags)

通用标签分为控制标签和数据标签。共22个。

### 控制标签(Control Tags)

控制标签一般是用于控制程序执行流程的，包含以下几个标签：if、elseif、else、append、generator、iterator、merge、sort、subset共9个。

#### if

##### 功能描述

##### 标签属性

##### 示例

#### elseif

##### 功能描述

##### 标签属性

##### 示例

#### else

##### 功能描述

##### 标签属性

##### 示例

#### append

##### 功能描述

##### 标签属性

##### 示例

#### generator

##### 功能描述

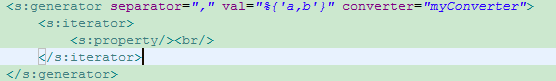
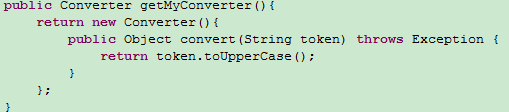
用来生成一个可遍历对象并把它压入ValueStack根栈顶，在该标签结束时会将压入的对象弹出，所以要作的事情必须在它的起始标记和结束标记之间完成。

还有一个用法是为那个可遍历的对象设定一个变量，以方便以后在该页面中使用。

##### 标签属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 必须的 | 默认值 | 类型 | 描述 |
| id | false |  | String | 弃用，用var代替 |
| var | false |  | String | 变量名，该变量存储可遍历对象，并放到page范围 |
| val | true |  | String | 可遍历对象的各个元素的来源 |
| separator | true |  | String | 分隔符 |
| count | false |  | Integer | 可遍历对象的容量 |
| converter | false |  | org.apache.struts2.util.IteratorGenerator.Converter | 转换器 |

##### 示例



#### iterator

##### 功能描述

控制标签中最重要的.它可以用来遍历一个数组、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Value Stack的根栈顶部。

##### 标签属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 必须的 | 默认值 | 类型 | 描述 |
| id | false |  | String | 弃用，用var代替 |
| var | false |  | String |  |
| value | false |  | String |  |
| begin | false | 0 | Integer |  |
| end | false | “values”大小，如果step是负数，则为0 | Integer |  |
| step | false | 1 | Integer |  |
| status | false |  | String | org.apache.struts2.views.jsp.IteratorStatus对象的引用变量 |

###### org.apache.struts2.views.jsp.IteratorStatus对象属性

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 描述 |
| index | Integer | 各次遍历的下标值，从0开始 |
| count | Integer | 各次遍历计数值，从1开始 |
| first | boolean | 如果当前元素是可遍历对象的第一个元素，则这个值为true |
| last | boolean | 如果当前元素是可遍历对象的最后一个元素，则这个值为true |
| even | boolean | 如果count属性的值是一个偶数，这个值为true |
| odd | boolean | 如果count属性的值是一个奇数，这个值为true |
| modulus | Integer | 如: modulus(2)这个属性需要输入一个参数，返回值是count属性值除以该参数的余数 |

##### 示例

略

#### merge

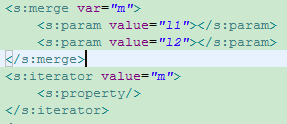
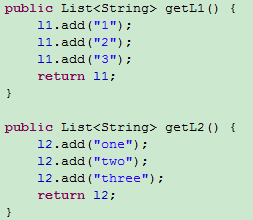
##### 功能描述

合并两个列表，该标签将从每个列表交替读取一个元素。

##### 标签属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 必须的 | 默认值 | 类型 | 描述 |
| id | false |  | String | 弃用，用var代替 |
| var | false |  | String | 变量名，该变量存储可遍历对象 |

##### 示例



#### sort

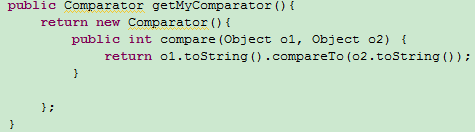
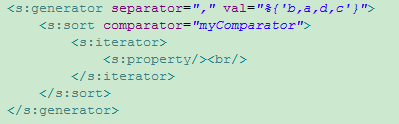
##### 功能描述

用来对一个可遍历对象里的元素进行排序。虽然在模型层或数据库层的SQL语句里使用ORDER BY子句对数据进行排序可能会更容易些，但把排序安排在表现层里实现的做法是一种更好的选择，这是一个比较不错的设计决策。

##### 标签属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 必须的 | 默认值 | 类型 | 描述 |
| id | false |  | String | 弃用，用var代替 |
| var | false |  | String | 变量名，该变量存储可遍历对象 |
| source | false |  | String | 将要被排序的可遍历对象 |
| comparator | true |  | java.util.Comparator | 比较器 |

##### 示例



#### subset

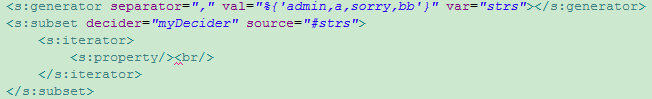
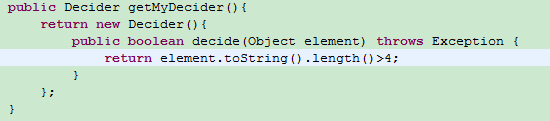
##### 功能描述

创建一个可遍历对象的子集。Decider类是org.apache.struts2.util. SubsetIteratorFilter类的一个内部接口,其具体子类将告诉该标签该如何来创建一个子集。

##### 标签属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 必须的 | 默认值 | 类型 | 描述 |
| id | false |  | String | 弃用，用var代替 |
| var | false |  | String | 变量名，该变量存储可遍历对象 |
| source | false |  | String | 来源 |
| count | false |  | Integer | 起始下标，新建子集将从这个下标开始从source属性中的可遍历对象筛选元素 |
| start | false |  | Integer | 新建子集的元素个数 |
| decider | false |  | org.apache.struts2.util. SubsetIteratorFilter.Decider | 新建子集的依据 |

##### 示例



### 数据标签(Data Tags)

数据标签一般是用于数据处理，包含以下几个标签：a、action、bean、date、debug、i18n、include、param、property、push、set、text、url共13个。

#### a

##### 功能描述

将呈现为一个HTML链接。

##### 标签属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 必须的 | 默认值 | 类型 | 描述 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

##### 示例

#### action

##### 功能描述

用来执行一个动作和该动作的结果。它还会把当前动作压入上下文栈中。

##### 标签属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 必须的 | 默认值 | 类型 | 描述 |
| id | false |  | String | 弃用，用var代替 |
| var | false |  | String | 变量名 |
| executeResult | false | false | Boolean | 是否执行/呈现动作结果 |
| flush | false | true | Boolean | 动作结束后是否刷新缓冲区 |
| ignoreContextParams | false | false | Boolean | 触发该动作时是否要包含请求参数 |
| name | true |  | String | 动作名称，不要使用.action后缀 |
| namespace | false |  | String | 动作所在的命名空间 |
| rethrowException | false |  | Boolean | 目标动作若出现异常，是否再扔出去 |

##### 示例

#### bean

##### 功能描述

创建一个JavaBean，并把它压入ContextMap子栈。这个标签和JSP中的<jsp:userBean/>很相似。

##### 标签属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 必须的 | 默认值 | 类型 | 描述 |
| id | false |  | String | 弃用，用var代替 |
| var | false |  | String | 变量名 |
| name | true |  | String | 将要创建的Bean的完整限定名 |

##### 示例

<s:bean name="java.lang.String" var="myObj"></s:bean>

#### date

##### 功能描述

用来对一个Java中的Date对象进行排版。

##### 标签属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 必须的 | 默认值 | 类型 | 描述 |
| id | false |  | String | 弃用，用var代替 |
| var | false |  | String | 变量名 |
| name | true |  | String | 将要创建的Date对象 |
| format | false |  | String | 日期的格式 |
| nice | false |  | Boolean | 是否进行高级排版 |

##### 示例

#### debug

##### 功能描述

打印出ValueStack中的内容。开发阶段非常有用。

##### 标签属性

无

##### 示例

<s:debug/>

#### i18n

##### 功能描述

##### 标签属性

##### 示例

#### include

##### 功能描述

用于将一个Servlet或JSP包含到当前页面。

##### 标签属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 必须的 | 默认值 | 类型 | 描述 |
| value | true |  | String | 被包含的目标 |

##### 示例

<s:include value="anotherJsp.jsp"></s:include>

<s:include value="anotherJsp.jsp">

<s:param name=”p1”>admin</s:param>

</s:include>

#### param

##### 功能描述

用于把一个参数传递给它的父标签。

##### 标签属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 必须的 | 默认值 | 类型 | 描述 |
| name | false |  | String | 参数名称 |
| value | false |  |  | 参数值 |

##### 示例

#### property

##### 功能描述

输出一个动作属性的值

##### 标签属性

##### 示例

#### push

##### 功能描述

把对象压入ValueStack的根栈中。起始标签压入对象，结束标签时弹出对象。

##### 标签属性

##### 示例

#### set

##### 功能描述

把对象压入ValueStack的上下文子栈中。

##### 标签属性

##### 示例

#### text

##### 功能描述

呈现一个I18N的消息

##### 标签属性

##### 示例

#### url

##### 功能描述

动态地创建一个URL。

##### 标签属性

##### 示例

## UI标签

分为Form标签和非表单标签。共25个。

### 表单标签

主要是对HTML表单标签的包装，以方便更好的利用struts2的功能，另外struts2还提供了一些更高级的标签，主要包括：checkbox、checkboxlist、combobox、doubleselect、head、file、form、hidden、label、optiontransferselect、optgroup、password、radio、reset、select、submit、textarea、textfield、token、updownselect。共20个。

#### checkbox

##### 功能描述

该标签呈现为一个HTML中的type为checkbox的input标签。

<input type=”checkbox”/>

##### 标签属性

是否允许动态属性：是

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **属性名** | **是否必须** | **默认值** | **值类型** | **描述** |
| accesskey | false |  | String | 快捷键。若指定值为c，这采用ALT+c的组合键可以改变其值。 |
| fieldValue | false | true | String | 选择框所对应的值 |
| value | false |  | String | 默任状态 |

##### 示例

|  |
| --- |
| **StudentAction.java** |
| public class StudentAction extends ActionSupport {  private boolean married;  public boolean isMarried() {  return married;  }  public void setMarried(boolean married) {  this.married = married;  }    } |

|  |
| --- |
| **jsp** |
| <s:checkbox name="married" label="已婚" value="true" fieldValue="true"></s:checkbox> |
| 说明：  value：value=”true”说明复选框默认是被选中的。  fieldValue：fieldValue=”false”说明客户端不管有没有选中该复选框，动作类的married属性永远为false；若fieldValue=”true”，客户端不选中复选框，动作类的married属性为false，反之为true。 |

#### checkboxlist

##### 功能描述

##### 标签属性

##### 示例

#### combobox

##### 功能描述

##### 标签属性

##### 示例

#### doubleselect

##### 功能描述

##### 标签属性

##### 示例

#### head

##### 功能描述

##### 标签属性

##### 示例

#### file

##### 功能描述

##### 标签属性

##### 示例

#### form

##### 功能描述

##### 标签属性

##### 示例

#### hidden

##### 功能描述

##### 标签属性

##### 示例

#### label

##### 功能描述

##### 标签属性

##### 示例

#### optiontransferselect

##### 功能描述

##### 标签属性

##### 示例

#### optgroup

##### 功能描述

##### 标签属性

##### 示例

#### password

##### 功能描述

##### 标签属性

##### 示例

#### radio

##### 功能描述

##### 标签属性

##### 示例

#### reset

##### 功能描述

##### 标签属性

##### 示例

#### select

##### 功能描述

##### 标签属性

##### 示例

#### submit

##### 功能描述

##### 标签属性

##### 示例

#### textarea

##### 功能描述

##### 标签属性

##### 示例

#### textfield

##### 功能描述

##### 标签属性

##### 示例

#### token

##### 功能描述

##### 标签属性

##### 示例

#### updownselect

##### 功能描述

##### 标签属性

##### 示例

### 非表单标签

根表单没有什么关系，大部分的是用于消息的显示。主要有：actionerror、actionmessage、component、div、fielderror，共5个。

#### actionerror

##### 功能描述

##### 标签属性

##### 示例

#### actionmessage

##### 功能描述

##### 标签属性

##### 示例

#### component

##### 功能描述

##### 标签属性

##### 示例

#### div

##### 功能描述

##### 标签属性

##### 示例

#### fielderror

##### 功能描述

##### 标签属性

##### 示例

# 转换与验证

## 概述转换和验证过程

## 使用内建的转换器

## 使用内建的验证器

## 编写自定义转换器和验证器

# 消息处理与国际化

# 拦截器

## 理解拦截器

## 使用内建的拦截器

## 自定义拦截器

## 深入拦截器

# 文件的上传与下载

# 防止重复提交

# 插件的使用

# S2SH整合

# 用户在线注册系统