**Struts2第2天课程**

**【学习目标】**

## 1，能够独立使用Servlet相关API

1. 按照解耦合方式调用ActionContext
2. 按照掌握接口注入方式操作Servlet API
3. 通过ServletActionContext 类的静态方法，直接获取Servlet API

## 2，能够接收请求参数

1. 独立使用Action作为model通过成员变量属性和setter方法接收参数
2. 独立使用Action中创建独立的model对象和setter、getter方法赋值，页面通过ognl表达式接收参数。
3. 独立使用ModelDriven接口封装请求的数据。
4. 了解集合类型(Collection)的数据封装

## 3，能够理解参数类型转换机制

1. 独立编写代码实现注册案例
2. 了解内置转换器

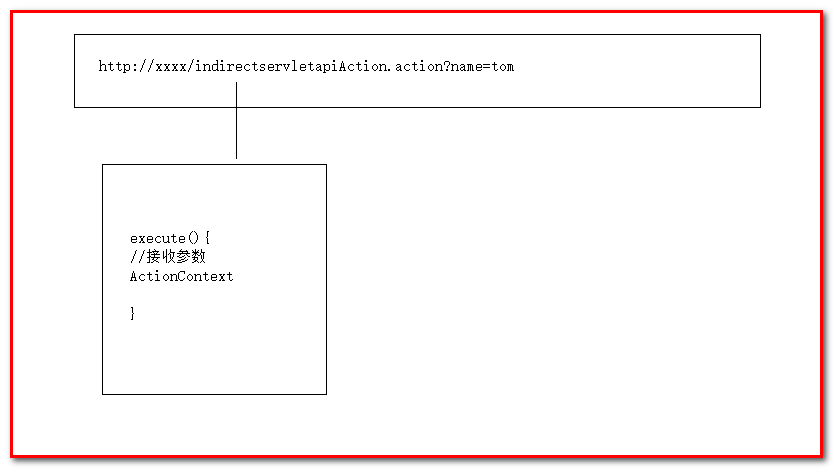
## 4，能够独立使用result结果集

1. 区别局部结果集
2. 区别全局结果集
3. 说出结果集的类型

## 5，能够理解struts2的下载功能

1. 独立编写代码实现文件下载

# Action使用Servlet相关API



【需求分析】为了简化开发，struts2默认情况下将servlet api（比如：request对象、response对象、session对象、application对象）都隐藏起来了。但是在某些情况下，我们还需要调用servlet api，比如，登录的时候将用户信息保存到session域中。

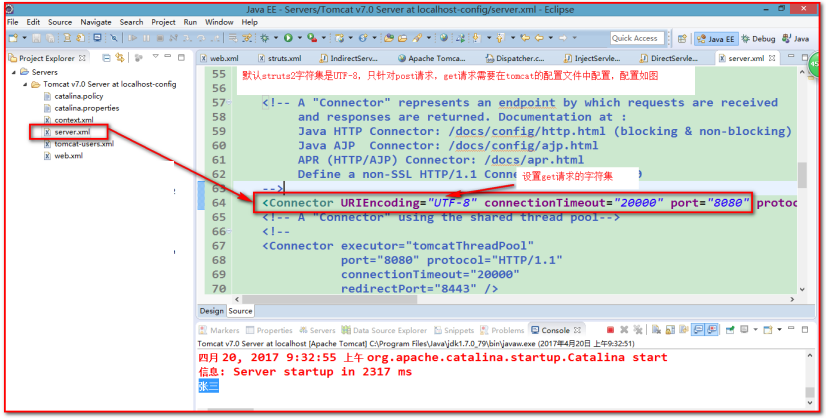
Struts2中提供了3种方式来获取servlet的api

第一种方式：解耦方式获取：借助ActionContext获取，方便单元测试Junit（这个是struts2官方的推荐的方式，但是不利于理解，所以我们不推荐）

第二种方式：接口注入方式操作Servlet API（了解）

第三种方式：通过ServletActionContext 类的静态方法直接获取Servlet API（掌握）

设置GET请求的字符集



## 解耦合方式调用（间接调用）-了解

第一点：耦合的概念

举个例子：

以前我们开发项目使用Serlvet的时候，调用的doGet方法和doPost(HttpServletRequest, HttpServletReponse) 方法，代码中直接关联request和 response对象，这就是耦合。

第二点：解耦的概念

解耦的概念就是代码中不直接依赖于request对象和response对象。

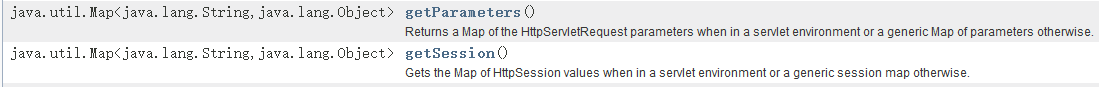
Struts2设计思想就是与Servlet API 解耦合，Action 中不再依赖Servlet的API。

那如何调用这些API呢？

struts2为我们提供了一个API，可间接调用servlet api，这个api叫做：ActionContext 类 （Action 上下文 ）。（可以理解该类就是一个工具类）

该API提供大量间接操作Servlet API 方法:



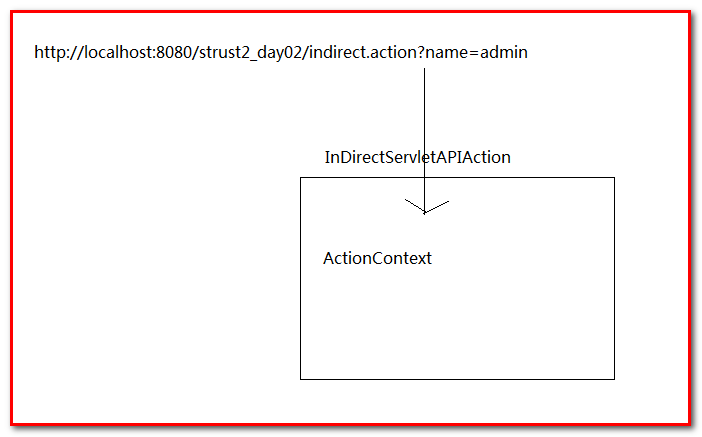




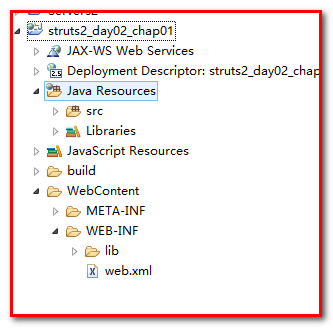




提示：间接操作是指操作ActionContext中的map集合，而不是直接操作相关对象。



【准备工作】创建项目struts2\_day02\_chap01，快速搭建运行环境



【说明】通过解耦合的方式间接操作ServletAPI

【第一步】：创建com.igeek\_01\_servletapi,创建IndirectServletAPIAction.java，具体代码如下：

|  |
| --- |
| **package** com.igeek\_01\_servletapi;  **import** java.util.Map;  **import** com.opensymphony.xwork2.ActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** IndirectServletAPIAction **extends** ActionSupport{    @Override  **public** String execute() **throws** Exception {  //获取工具类  ActionContext actionContext = ActionContext.*getContext*();  //获取页面的参数信息  Map<String, Object> parameters = actionContext.getParameters();  //接收的结果是一个数组？此处为什么要设计为数组接收页面参数？  //因为通过数组的方式，如果页面传递的是一个值，数组也可以接收，  //如果页面传递的是两个值，数组也可以接收  Object object = parameters.get("name");    System.*out*.println(object);    **return** *NONE*;  }    } |

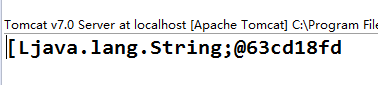
【第二步】：struts.xml

|  |
| --- |
| <?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=*"default"* extends=*"struts-default"* namespace=*"/"*>  <action name=*"indirect"* class=*"com.igeek\_01\_servletapi.IndirectServletAPIAction"*></action>  </package>  </struts> |

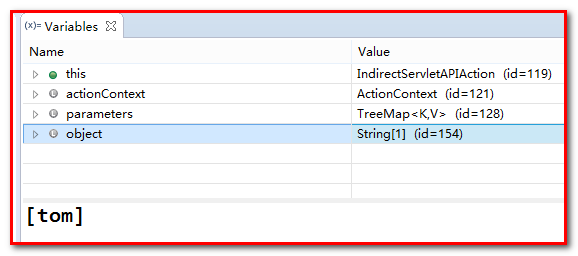
【第三步】：访问请求url：

<http://localhost:8080/struts2_day02_chap01/indirect.action?name=tom>

运行结果：



控制台一直打印的是对象的hashcode，很奇怪，通过断点发现：



原因分析：map.get("name");--->get出来的数组String[]的对象数组，所以才会一直打印hashcode的值

将代码修改为：

|  |
| --- |
| **package** com.igeek\_01\_servletapi;  **import** java.util.Map;  **import** com.opensymphony.xwork2.ActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** IndirectServletAPIAction **extends** ActionSupport{    @Override  **public** String execute() **throws** Exception {  //获取工具类  ActionContext actionContext = ActionContext.*getContext*();  //获取页面的参数信息  Map<String, Object> parameters = actionContext.getParameters();  //接收的结果是一个数组？此处为什么要设计为数组接收页面参数？  //因为通过数组的方式，如果页面传递的是一个值，数组也可以接收，  //如果页面传递的是两个值，数组也可以接收  // Object object = parameters.get("name");  String[] values = (String[]) parameters.get("name");    System.*out*.println(values[0]);    **return** *NONE*;  }    } |

## Servlet API直接调用（耦合方式）

### 接口注入方式操作Servlet API（了解）



\* ServletRequestAware        -- 获取到request对象  
\* ServletResponseAware        -- 获取到response对象  
\* ServletContextAware        -- 获取到ServletContext对象

用法：

【第一步】：在com.igeek\_01\_servletapi包中，创建InjectServletAPIAction.java类， （Inject注入）

|  |
| --- |
| **package** com.igeek\_01\_servletapi;  **import** javax.servlet.http.HttpServletRequest;  **import** org.apache.struts2.interceptor.ServletRequestAware;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** InjectServletAPIAction **extends** ActionSupport **implements** ServletRequestAware{  //定义一个全局的request对象接收  **private** HttpServletRequest request;    @Override  **public** String execute() **throws** Exception {  // /接收页面的参数  String value = request.getParameter("name");    System.*out*.println(value);  **return** *NONE*;  }  /\*\*  \* 在实现了ServletRequestAware接口之后，系统会自动的的让你实现setServletRequest方法，  \* 给request对象赋值  \*/  @Override  **public** **void** setServletRequest(HttpServletRequest request) {  **this**.request = request;  }    } |

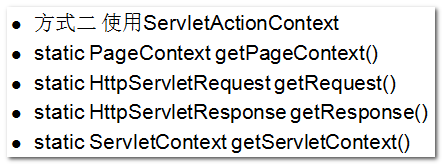
【第二步】：配置struts.xml:

|  |
| --- |
| <!-- 接口注入的方式调用ServletAPI,直接调用 -->  <action name=*"inject"* class=*"com.igeek\_01\_servletapi.InjectServletAPIAction"*></action> |

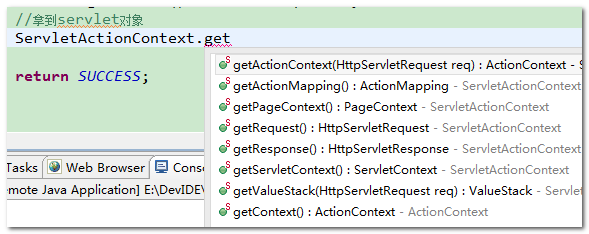
【第三步】：测试

【注意】在action初始化的时候，会自动将request注入赋值给action

### 1.2.2通过ServletActionContext 类的静态方法直接获取Servlet API （推荐）







【第一步】：创建DirectServletAPIAction 类，代码如下：

|  |
| --- |
| **package** com.igeek\_01\_servletapi;  **import** javax.servlet.http.HttpServletRequest;  **import** org.apache.struts2.ServletActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** DirectServletAPIAction **extends** ActionSupport{  @Override  **public** String execute() **throws** Exception {  //1.获取页面参数  HttpServletRequest request = ServletActionContext.*getRequest*();  String name =request.getParameter("name");  System.***out***.println(name);  //2.向request域设置值  request.setAttribute("msg", "rose");  **return** ***SUCCESS***;  }  } |

【第二步】:在WebContent下新建01\_servletapi文件夹，在文件夹中编写result.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  ${msg}  </body>  </html> |

【第三步】：在struts.xml进行如下配置

|  |
| --- |
| <!-- 配置直接获取ServletAPI的Action -->  <action name=*"direct"* class=*"com.igeek\_01\_servletapi.DirectServletAPIAction"*>  <result name=*"success"*>/01\_servletapi/result.jsp</result>  </action> |

【第三步】：测试

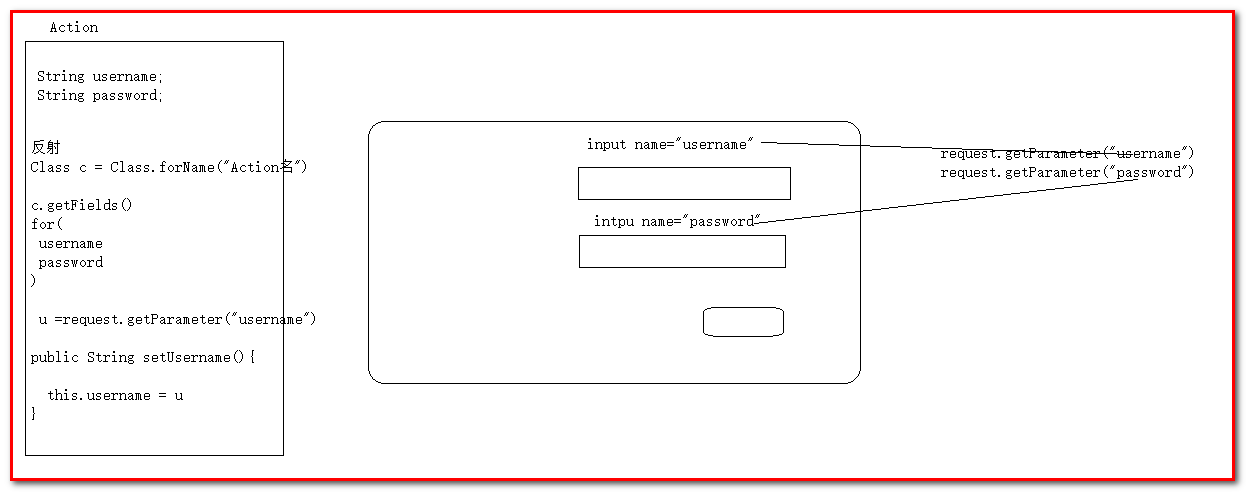
【总结】

1>使用ActionContext来获取request对象，与servlet解耦，但是比较麻烦。（Struts官方推荐，主要为了单元测试方便）

2>使用接口注入，必须要实现ServletRequestAware接口，使用麻烦

3>使用ServletActionContext类的静态方法获取request对象，比较方便（个人推荐使用）

# 请求参数的接收机制（重点）



【概念了解】

1、什么是请求参数？

答：比如登录、注册功能的时候，客户端需要向服务器提交登录或者注册信息，这些信息都是请求参数

Serlvet接收参数：request.getParameter(“XXX”);

【理论知识】

作为MVC框架，Struts2要负责解析HTTP请求参数，并将其自动封装到Model对象中。

表现层一个核心职责 ， 负责接收客户端提交请求数据

* 表现层 负责和客户端交互 （围绕 request 、response ）
* 业务层 负责数据业务逻辑处理
* 持久层 负责和数据库进行增删改查

Struts2提供两大类（属性驱动封装、模型驱动封装）、三种数据封装的方式：

* 属性驱动方式：
  + Action本身作为表单model对象，通过成员变量的setter方法进行封装。(使用非常多)
  + Action中创建独立的表单model对象和提供对应的setter、getter方法，页面通过ognl表达式进行封装。
* 模型驱动方式：
  + 使用ModelDriven接口（模型驱动），对请求的数据进行封装。(使用非常多)

## 属性驱动

### Action作为model，通过成员变量属性和setter方法接收参数进行封装（方式一）

【示例说明】新建项目struts2\_day02\_chap02写一个登陆页面login1.jsp，Action中接收页面传递的信息。

【第一步】：在WebContent下面创建01\_model文件夹，在该文件夹中创建login1.jsp页面：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>My JSP 'LOGIN.jsp' starting page</title>  </head>  <body>  <h3>方式一：Aciton作为表单模型，提供属性和属性对应的getter方法和setter方法</h3>  <form action=*"**${pageContext.request.contextPath}/login1.action"* method=*"post"*>  username:<input type=*"text"* name=*"username"*><br />  password:<input type=*"password"* name=*"pwd"*><br />  <input type=*"submit"* value=*"登录"*>  </form>  </body>  </html> |

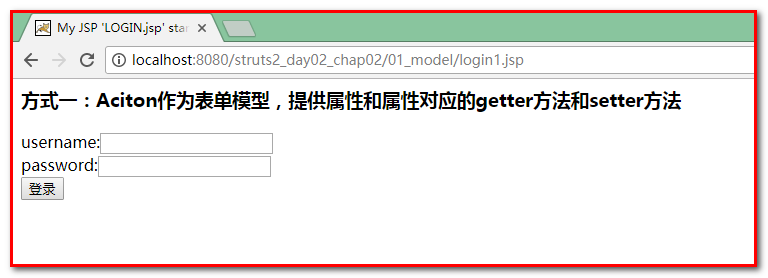
【第二步】：创建一个包com.igeek\_01\_model，并且，新建LoginAction1类，在Action 提供username、pwd成员变量生成setter方法，具体代码如下：

|  |
| --- |
| **package** com.igeek\_01\_model;  **import** com.opensymphony.xwork2.ActionSupport;  /\*\*  \* 1.1.1.Action作为model，通过成员变量属性和setter方法接收参数进行封装（方式一）  \*  \* **@author** jerry  \*  \*/  **public** **class** LoginAction1 **extends** ActionSupport{  //提供私有属性，  //私有属性的名字必须和表单的name一致(虽然struts2允许不一致，但是没人这么干)  //但是最后的赋值依赖的是setter方法  //为属性提供getter和setter方法  **private** String username;  **private** String pwd;  @Override  **public** String execute() **throws** Exception {  System.*out*.println(toString());  **return** *NONE*;  }  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getPwd() {  **return** pwd;  }  **public** **void** setPwd(String pwd) {  **this**.pwd = pwd;  }  @Override  **public** String toString() {  **return** "LoginAction1 [username=" + username + ", pwd=" + pwd + "]";  }  } |

【第三步】：在struts.xml中配置Action对象：

|  |
| --- |
| <?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=*"default"* extends=*"struts-default"* namespace=*"/"*>  <action name=*"login1"* class=*"com.igeek\_01\_model.LoginAction1"*></action>  </package>  </struts> |

【第四步】：测试

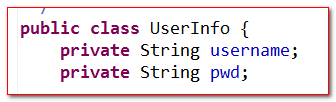


注意点：

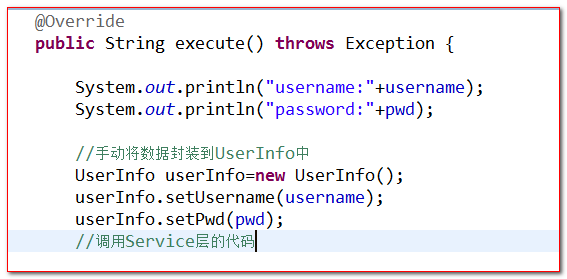
特点 ： Action本身就是 表单model对象

缺点： Action 往往需要手动将数据封装到一个新的model对象，传递给业务层（service层）使用。

例如：定义UserInfo类，提供username和pwd的属性



username和pwd属性封装到UserAction类中，用来获取username和pwd的值，将获取的值要重新封装到UserInfo类中。



### Action中创建独立的model对象,提供setter、getter方法，页面通过ognl表达式进行封装（方式二）

【第一步】：在login2.jsp中编写页面代码

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  <h3>Action中创建独立的model对象,提供setter、getter方法，页面通过ognl表达式进行封装（方式二）</h3>  <form action=*"*${pageContext.request.contextPath }*/login2.action"* method=*"post"*>  username:<input type=*"text"* name=*"userInfo.username"* /><br>  password:<input type=*"password"* name=*"userInfo.pwd"* /><br>  <input type=*"submit"* value=*"登录"* />  </form>  </body>  </html> |

【第二步】：在com.igeek.mode包中，创建表单模型UserInfo类：

|  |
| --- |
| **package** com.igeek.mode;  /\*\*  \*专门用来封装页面表单数据  \*首先提供与页面表单的name一致的属性，并且为这些属性提供getter和setter 方法  \*  \* **@author** jerry  \*  \*/  **public** **class** UserInfo {    **private** String username;  **private** String pwd;    **public** UserInfo()  {  System.***out***.println("UseInfo的构造方法");  }    **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  System.***out***.println("setUsername....");  **this**.username = username;  }  **public** String getPwd() {  **return** pwd;  }  **public** **void** setPwd(String pwd) {  System.***out***.println("setPwd...");  **this**.pwd = pwd;  }  @Override  **public** String toString() {  **return** "UserInfo [username=" + username + ", pwd=" + pwd + "]";  }  } |

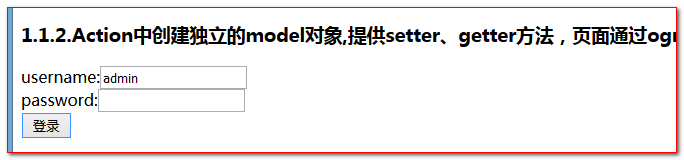
【第三步】：在com.igeek\_01\_model包中创建LoginAction2类，继承ActionSupport，具体代码如下：

|  |
| --- |
| **package** com.igeek\_01\_model;  **import** com.igeek.mode.UserInfo;  **import** com.opensymphony.xwork2.ActionSupport;  /\*\*  \* 1.1.2.Action中创建独立的model对象,提供setter、getter方法，页面通过ognl表达式进行封装（方式二）  \* 操作步骤：  \* 1 创建独立的model对象，在这个对象中提供私有的属性(属性名字和表单的name一致)，并且为属性提供setter方法  \* 2 model对象作为属性放入Action中，并且提供getter和setter方法  \* **@author** jerry  \*  \*/  **public** **class** LoginAction2 **extends** ActionSupport{    //独立的model  //这个模型对象什么时候被创建？  //答：第一个属性进行赋值的时候，系统会自动创建这个对象，并且封装到Action中  **private** UserInfo userInfo ;    @Override  **public** String execute() **throws** Exception {  System.***out***.println(userInfo.toString());  **return** ***NONE***;  }  **public** UserInfo getUserInfo() {  System.***out***.println("getUserInfo...");  **return** userInfo;  }  **public** **void** setUserInfo(UserInfo userInfo) {  System.***out***.println("setUserInfo...");  **this**.userInfo = userInfo;  }  } |

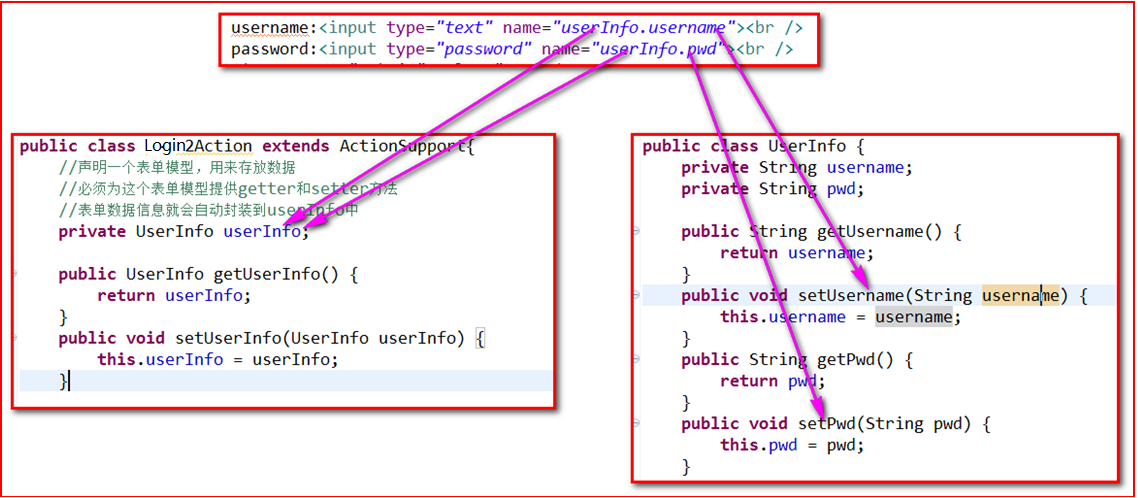
【第四步】：配置

|  |
| --- |
| <!-- 配置参数封装方式二的Action -->  <action name=*"login2"* class=*"com.igeek\_01\_model.LoginAction2"* /> |

【第五步】：测试



【1、分析】



【2、原理】 （必须同时提供model getter和setter 方法）



阶段小结：

1、第一种和第二种封装方式类似，都是使用的是Action的属性来封装数据。因此，这两种都统称为 属性驱动封装数据

2、第二种方法也有缺点，你需要在页面中采用如下的方式编写表单：

username:<input type=*"text"* name=*"userInfo.username"*><br />

password:<input type=*"password"* name=*"userInfo.pwd"*><br />

## 模型驱动（方式三）(重点)

使用ModelDriven接口（模型驱动），对请求的数据进行封装。

【第一步】：在login3.jsp页面中编写如下代码：

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  <H3>方式三：通过ModelDriven驱动封装数据</H3>  <form action=*"*${pageContext.request.contextPath}*/login3.action"* method=*"post"*>  username:<input type=*"text"* name=*"username"*><br />  password:<input type=*"password"* name=*"pwd"*><br />  <input type=*"submit"* value=*"登录"* />  </form>  </body>  </html> |

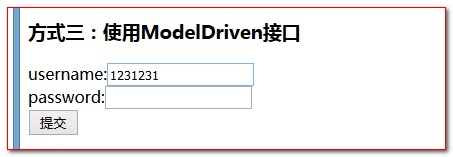
【第二步】：创建LoginAction3类， 继承ActionSupport类，实现 ModelDriven接口 ，实现getModel方法

|  |
| --- |
| **package** com.igeek\_01\_model;  **import** com.igeek.mode.UserInfo;  **import** com.opensymphony.xwork2.ActionSupport;  **import** com.opensymphony.xwork2.ModelDriven;  /\*\*  \* 方式三：模型驱动的方式  \* Action需要实现ModelDriven接口  \* 当Action实现了ModelDriven接口之后，会报错，那么什么原因？我们需要实现ModelDriven接口中的getModel方法  \* **@author** jerry  \*  \*/  **public** **class** LoginAction3 **extends** ActionSupport **implements** ModelDriven<UserInfo>{  //定义一个UserInfo  //必须手动的实例化UserInfo，系统是不会自动实例化UserInfo的  **private** UserInfo userInfo = **new** UserInfo();  //属性驱动：模型驱动优于属性驱动，如何理解这句话：      @Override  **public** String execute() **throws** Exception {  System.***out***.println(userInfo.toString());  **return** ***NONE***;  }  /\*\*  \* 当你实现了接口之后，必须要实现getModel方法  \* 这个getModel返回最终数据要封装的对象，而且这个对象不能为null  \*  \*/  @Override  **public** UserInfo getModel() {  System.***out***.println("getModel...");  **return** userInfo;  }    } |

【第三步】：配置

|  |
| --- |
| <!-- 配置参数封装的方式三的Action -->  <action name=*"login3"* class=*"com.igeek\_01\_model.LoginAction3"* /> |

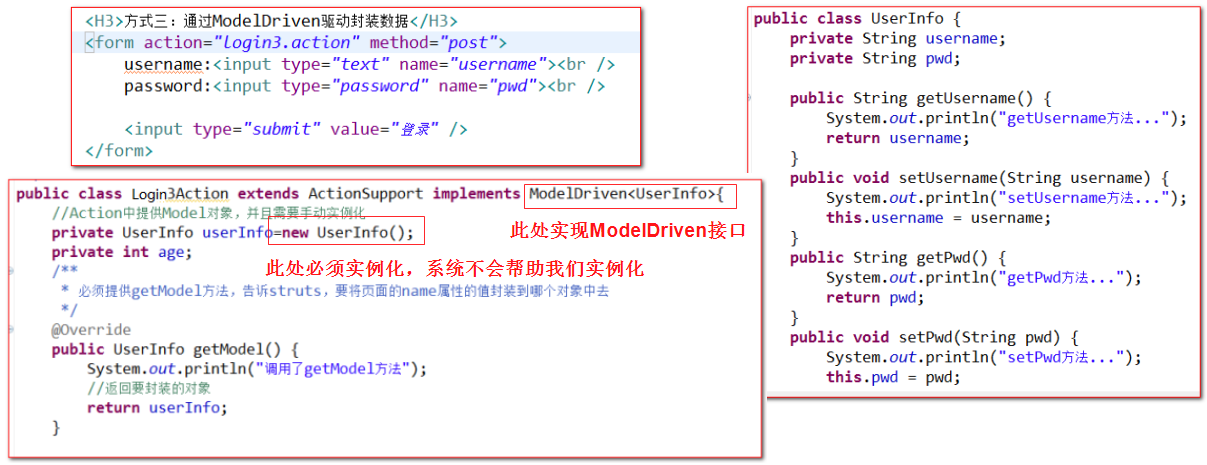
【第四步】：访问



【第五步】：

原理分析：

1>图解

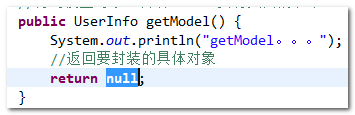


2>：具体解释



注意：

1. 你返回的模型对象必须是可用对象，即是个对象，且对象中有对应name属性的setter方法。下面的示例就无法封装数据。



【小结】：

在企业级的开发中，各种方式都有可能会用，建议初学者先选择一种方式进行熟练运行

第一种和第三种必须掌握，第二种了解

## Struts2的复杂数据的封装:

### 封装到List集合:

页面:product1.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  <form action=*"*${ pageContext.request.contextPath }*/product1Action.action"* method=*"post"*>  商品名称:<input type=*"text"* name=*"list[0].name"*/><br/>  商品价格:<input type=*"text"* name=*"list[0].price"*/><br/>  商品名称:<input type=*"text"* name=*"list[1].name"*/><br/>  商品价格:<input type=*"text"* name=*"list[1].price"*/><br/>  商品名称:<input type=*"text"* name=*"list[2].name"*/><br/>  商品价格:<input type=*"text"* name=*"list[2].price"*/><br/>  <input type=*"submit"* value=*"批量导入"*/>  </form>  </body>  </html> |

Javabean：

|  |
| --- |
| **package** com.igeek.mode;  **public** **class** Product {  **private** String name;  **private** **double** price;    **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **double** getPrice() {  **return** price;  }  **public** **void** setPrice(**double** price) {  **this**.price = price;  }  @Override  **public** String toString() {  **return** "Product [name=" + name + ", price=" + price + "]";  }  } |

Action：

|  |
| --- |
| **package** com.igeek\_01\_model;  **import** java.util.List;  **import** com.igeek.mode.Product;  **import** com.opensymphony.xwork2.ActionSupport;  /\*\*  \* 封装复杂的数据到List集合中。  \* **@author** jt  \*/  **public** **class** ProductAction1 **extends** ActionSupport{  **private** List<Product> list;    **public** List<Product> getList() {  **return** list;  }  **public** **void** setList(List<Product> list) {  **this**.list = list;  }  @Override  **public** String execute() **throws** Exception {  **for** (Product product : list) {  System.***out***.println(product);  }  **return** ***NONE***;  }  } |

### 封装到Map集合

页面:product2.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  <h1>批量插入商品:封装到Map集合</h1>  <form action=*"*${ pageContext.request.contextPath }*/productAction2.action"* method=*"post"*>  商品名称:<input type=*"text"* name=*"map['one'].name"*/><br/>  商品价格:<input type=*"text"* name=*"map['one'].price"*/><br/>  商品名称:<input type=*"text"* name=*"map['two'].name"*/><br/>  商品价格:<input type=*"text"* name=*"map['two'].price"*/><br/>  商品名称:<input type=*"text"* name=*"map['three'].name"*/><br/>  商品价格:<input type=*"text"* name=*"map['three'].price"*/><br/>  <input type=*"submit"* value=*"批量导入"*/>  </form>  </body>  </html> |

Action：

|  |
| --- |
| **package** com.igeek\_01\_model;  **import** java.util.Map;  **import** com.igeek.mode.Product;  **import** com.opensymphony.xwork2.ActionSupport;  /\*\*  \* 复杂类型数据封装：封装到Map集合  \* **@author** jt  \*  \*/  **public** **class** ProductAction2 **extends** ActionSupport {  **private** Map<String,Product> map;    **public** Map<String, Product> getMap() {  **return** map;  }  **public** **void** setMap(Map<String, Product> map) {  **this**.map = map;  }  @Override  **public** String execute() **throws** Exception {  **for** (String key : map.keySet()) {  Product product = map.get(key);  System.***out***.println(key+" "+product);  }  **return** ***NONE***;  }  } |

# 请求参数的类型转换机制（了解）

【问题】从上面的例子中，大家看到，我们在Action中使用了int类型接收了age属性。

大家要明确一点：页面传递过来的肯定是String类型，为什么我们没有需要进行类型转换呢？

【解答】Struts2提供了功能强大的类型转换器，用于将请求数据封装到model对象。

对于大部分常用类型，开发者根本无需创建自己的转换器。

只有当内置的转换类型不够用的时候，可以自定义参数类型转换器。

下面分别讲解。

## 内置的参数类型的转换（了解）

### 内置的转换器

Struts2内置了常见数据类型多种转换器，如下：

* boolean 和 Boolean
* char和 Character
* int 和 Integer
* long 和 Long
* float 和 Float
* double 和 Double
* Date 可以接收 yyyy-MM-dd格式字符串
* 数组 可以将多个同名参数，转换到数组中
* 集合 支持将数据保存到 List 或者 Map 集合

### 编写注册案例 （包含文本、数字、日期、数组）

【需求】编写注册案例，使用struts2的内置转换器

【第一步】：新建项目struts2\_day02\_chap03,在WebContent下面，创建conversion文件夹，然后创建regist.jsp页面：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>My JSP 'Regist.jsp' starting page</title>  </head>  <body>  <h3>用户注册--内部类型转换测试</h3>  <form action=*"*${pageContext.request.contextPath }*/regist.action"* method=*"post"*>  username:<input type=*"text"* name=*"username"*><br />  age:<input type=*"text"* name=*"age"*><br />  <h1>日期格式默认使用yyyy-MM-dd</h1>  birthday:<input type=*"text"* name=*"birthday"*><br />  hobby:<input type=*"checkbox"* name=*"hobby"* value=*"音乐"*>音乐  <input type=*"checkbox"* name=*"hobby"* value=*"游戏"*>游戏  <input type=*"checkbox"* name=*"hobby"* value=*"旅游"*>旅游<br />  <input type=*"submit"* value=*"注册"*>  </form>  </body>  </html> |

【第二步】：创建com.igeek.conversion包，然后创建RegistAction.java类，继承ActionSupport类：具体代码如下：

|  |
| --- |
| **package** com.igeek.conversion;  **import** java.util.Arrays;  **import** java.util.Date;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** RegistAction **extends** ActionSupport{  **private** String username;  **private** **int** age;  **private** Date birthday;  **private** String[] hobby;  @Override  **public** String execute() **throws** Exception {    System.*out*.println(toString());    **return** *NONE*;  }  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **this**.age = age;  }  **public** Date getBirthday() {  **return** birthday;  }  **public** **void** setBirthday(Date birthday) {  **this**.birthday = birthday;  }  **public** String[] getHobby() {  **return** hobby;  }  **public** **void** setHobby(String[] hobby) {  **this**.hobby = hobby;  }  @Override  **public** String toString() {  **return** "RegistAction [username=" + username + ", age=" + age  + ", birthday=" + birthday + ", hobby="  + Arrays.*toString*(hobby) + "]";  }    } |

【第三步】：配置

|  |
| --- |
| <!-- 配置RegistAction -->  <action name=*"regist"* class=*"com.igeek.conversion.RegistAction"*></action> |

【第四步】：测试:

# result结果集的使用（重点）

【为什么要介绍结果集】

在前面的过程中，我们已经能够熟练的操作result标签，并且返回jsp视图页面到客户端，

但是大家有没有想象，如果我们要跳转的路径又是一个Action，使用我们之前的方式能够跳转吗？

带着这个问题我们继续深入学习result对象。

【概念】

结果集：可以把它看成是响应，通过配置方式实现页面路径的跳转

【理解结果集】

针对于结果集，我们主要学习以下三个知识点：

1. 局部结果集
2. 全局结果集
3. 结果集类型

## 局部结果集

在<action> 标签内部配置的<result>元素。

作用范围：局部结果集只能给当前的action使用，只对当前Action有效

【测试】当响应的结果集名字不存在的时候，会报错：

【第一步】：新建struts2\_day02\_chap04，创建com.igeek.result包，编写LocalResultAction类，代码如下：

|  |
| --- |
| **package** com.igeek.result;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** LocalResultAction **extends** ActionSupport{    @Override  **public** String execute() **throws** Exception {  System.***out***.println("局部结果集");  **return** ***SUCCESS***;  }    } |

【第二步】：编写struts.xml文件和result.jsp：

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  local  </body>  </html> |

|  |
| --- |
| <!-- 配置局部结果集 -->  <action name=*"local"* class=*"com.igeek.result.LocalResultAction"*>  <!-- name属性可以省略，默认值是success  在此处可以配置多个result结果集视图，多个结果集视图的name不能一致  -->  <result>/result.jsp</result>  </action> |

【第三步】：测试

## 全局结果集

在包的标签中<global-results>中配置

作用范围：对package内所有Action生效

【示例】

【第一步】：在com.igeek.result包中，并在包中创建GlobalResultAction类，

|  |
| --- |
| **package** com.igeek.result;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** GlobalResultAction **extends** ActionSupport{    @Override  **public** String execute() **throws** Exception {  System.*out*.println("全局结果集");  **return** *SUCCESS*;  }    } |

【第二步】：struts.xml中先配置Action的访问路径

|  |
| --- |
| <!-- 全局结果集 -->  <action name=*"global"* class=*"com.igeek.result.GlobalResultAction"*>    </action> |

然后配置全局跳转路径

|  |
| --- |
| <global-results>  <result name=*"**success"*>/global\_result.jsp</result>  </global-results> |

编写global\_result.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  global  </body>  </html> |

【第三步】：测试

最终发现，系统能够跳入到全局配置的URL路径中去

【注意】同名的局部结果集会覆盖全局结果集。先走局部，如果局部不存在，再去走全局，如果还不存在，则报错

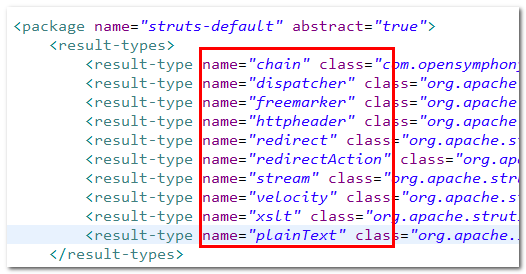
## 结果集的类型type

作用：控制响应的方式（转发、重定向）

配置<result> 元素时， name是逻辑视图名称， type是结果集类型。

Struts2提供的常用结果集类型都定义在struts-default.xml 中：

内置的结果集类型：



|  |  |  |
| --- | --- | --- |
| 序号 | 结果集类型名 | 描述 |
| 1 | dispatcher | 默认结果类型，用来呈现JSP页面（请求跳转至另外一个jsp） |
| 2 | chain | 将action和另外一个action链接起来(请求跳转至另外一个Action) |
| 3 | redirect | 将用户重定向到一个已配置好的jsp |
| 4 | redirectAction | 将用户重定向到一个已定义好的action |
| 5 | stream | 将原始数据作为流传递回浏览器端，该结果类型对下载的内容和图片非常有用 |
| 6 | freemarker | 用于FreeMarker整合的结果类型 |
| 7 | httpheader | 返回一个已配置好的HTTP头信息响应 |
| 8 | velocity | 呈现Velocity模板 |
| 9 | xslt | 呈现XML到浏览器，该XML可以通过XSL模板进行转换 |
| 10 | plainText | 返回普通文本内容 |

【示例】

演示请求跳转和重定向跳转的区别

【第一步】：新建ResultType1Action和ResultType2Action类

|  |
| --- |
| **package** com.igeek.result;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** ResultType1Action **extends** ActionSupport{  @Override  **public** String execute() **throws** Exception {  System.***out***.println("result type 1");  **return** ***SUCCESS***;  }  } |

|  |
| --- |
| **package** com.igeek.result;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** ResultType2Action **extends** ActionSupport{  @Override  **public** String execute() **throws** Exception {  System.***out***.println("result type 2");  **return** ***SUCCESS***;  }  } |

在WebContent下新建resultType1.jsp和resultType2.jsp页面

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  result type 1  </body>  </html> |

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  result type 2  </body>  </html> |

【第二步】：配置struts.xml文件，

dispatcher请求跳转到jsp页面：

|  |
| --- |
| <!-- type="dispatcher":这是默认值 ，请求跳转至另外一个jsp页面，无法跳转至另外一个action-->  <action name=*"rs1"* class=*"com.igeek.result.ResultType1Action"*>  <result type=*"dispatcher"*>/resultType1.jsp</result>  </action> |

测试，输入 [http://localhost:8080/struts2\_day02\_chap04/rs1.action](http://localhost:8080/struts2_day02/rs1.action)，观察地址栏没有变化，说明dispatcher使用forword转发，去掉type=”dispatcher”后观察地址栏没变化，说明struts2默认使用dispacher跳转

chain：请求跳转到Action，修改struts.xml配置如下

|  |
| --- |
| <!-- type="chain": 跳转至另外一个action -->  <action name=*"rs1"* class=*"com.igeek.result.ResultType1Action"*>  <result type=*"chain"*>rs2</result>  </action>  <action name=*"rs2"* class=*"com.igeek.result.ResultType2Action"*>  <result>/resultType2.jsp</result>  </action> |

测试，输入 [http://localhost:8080/struts2\_day02\_chap04/rs1.action](http://localhost:8080/struts2_day02/rs1.action)，观察地址栏没有变化，说明chain使用forword转发，观察跳转的结果页面发生变化

redirect：重定向跳转到jsp页面,修改struts.xml配置如下：

|  |
| --- |
| <!-- type="redirect"：重定向跳转，action中的参数无法传递 -->  <action name=*"rs1"* class=*"com.igeek.result.ResultType1Action"*>  <result type=*"redirect"*>/resultType2.jsp</result>  </action> |

测试，输入 [http://localhost:8080/struts2\_day02\_chap04/rs1.action](http://localhost:8080/struts2_day02/rs1.action)

结果：观察地址栏发生变化，说明redirect使用sendRedirect重定向

redirectAction：重定向到Action

|  |
| --- |
| <!-- redirectAction:重定向到另外一个Action -->  <action name=*"rs1"* class=*"com.igeek.result.ResultType1Action"*>  <result type=*"redirectAction"*>rs2</result>  </action>  <action name=*"rs2"* class=*"com.igeek.result.ResultType2Action"*>  <result>/resultType2.jsp</result>  </action> |

测试：输入 [http://localhost:8080/struts2\_day02\_chap04/rs1.action](http://localhost:8080/struts2_day02/rs1.action)

结果：观察地址栏发生变化，说明redirectAction使用sendRedirect重定向

【总结】

* dispatcher（默认值）：请求转发。（最常用）

作用：服务器内部是同一次请求，可采用request传递数据，URL不变，forword方式跳转到jsp页面

* chain（了解）

作用：请求转发到另外一个Action中，forword方式跳转到另一个action

* redirect

作用：重定向到某个jsp页面，服务器发起了一次新的请求，不能通过request传递参数，URL改变为新的地址。

应用场景举例：登录后重定向到网站的主页面，注销用户是重定向到登录页面。

重定向到jsp页面

* redirectAction

作用：重定向到另外一个Action

# Struts2的文件的下载

## 回顾文件的下载

|  |
| --- |
| 要点：两头一流   * + 使用response输出流，将文件信息打印到客户端   + 设置Content-Type头信息， 通过   ServletActionContext.getServletContext().getMimeType(fileName); 获取   * + 设置Content-Disposition 头信息 ， attachment;filename=文件名 |

## Struts2的文件下载的实现

【第一步】：创建项目struts2\_day02\_chap05,新建com.igeek.download包，在包中创建DownloadAction类：

|  |
| --- |
| **package** com.igeek.download;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.InputStream;  **import** org.apache.struts2.ServletActionContext;  **import** com.opensymphony.xwork2.ActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **import** cmo.igeek.util.FileUtils;  **public** **class** DownloadAction **extends** ActionSupport{    @Override  **public** String execute() **throws** Exception {  String path = "d://";  String filename="你好.txt";  //下载需要两头一流  //contentType:文件类型  String contentType = ServletActionContext.*getServletContext*().getMimeType(filename);  //inputStream:流  InputStream inputStream = **new** FileInputStream(**new** File(path+filename));    //如果下载的文件命中含有中文，咋们就进行中文编码  String agent = ServletActionContext.*getRequest*().getHeader("user-agent");  filename = FileUtils.*encodeDownloadFilename*(filename, agent);      //contentDisposition:设置文件的打开方式，和打开的文件的名字  String contentDisposition = "attachment;filename="+filename;      //将两头一流放入值栈中  ActionContext.*getContext*().put("contentType", contentType);  ActionContext.*getContext*().put("contentDisposition", contentDisposition);  ActionContext.*getContext*().put("inputStream", inputStream);    **return** ***SUCCESS***;  }      } |

【第二步】编写FileUtils类，用来给文件的文件名进行编码，放置出现乱码

|  |
| --- |
| package cmo.igeek.util;  import java.io.IOException;  import java.net.URLEncoder;  import sun.misc.BASE64Encoder;  public class FileUtils {  /\*\*  \* 下载文件时，针对不同浏览器，进行附件名的编码  \*  \* @param filename  \* 下载文件名  \* @param agent  \* 客户端浏览器  \* @return 编码后的下载附件名  \* @throws IOException  \*/  public static String encodeDownloadFilename(String filename, String agent)  throws IOException {  if (agent.contains("Firefox")) { // 火狐浏览器  filename = "=?UTF-8?B?"  + new BASE64Encoder().encode(filename.getBytes("utf-8"))  + "?=";  filename = filename.replaceAll("\r\n", "");  } else { // IE及其他浏览器  filename = URLEncoder.encode(filename, "utf-8");  filename = filename.replace("+"," ");  }  return filename;  }  } |

【第三步】配置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=*"default"* extends=*"struts-default"* namespace=*"/"*>  <!-- 配置文件下载的Action -->  <action name=*"download"* class=*"com.igeek.download.DownloadAction"*>  <!-- stream:专门用来下载用的结果集类型 -->  <result name=*"success"* type=*"stream"*></result>  </action>  </package>  </struts> |

【第四步】测试

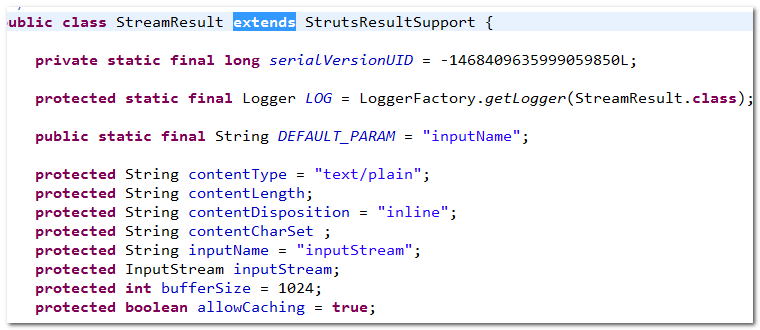
### Struts2的文件下载机制原理(了解)

我们为什么要在struts.xml中配置contentType和contentDisposition属性？

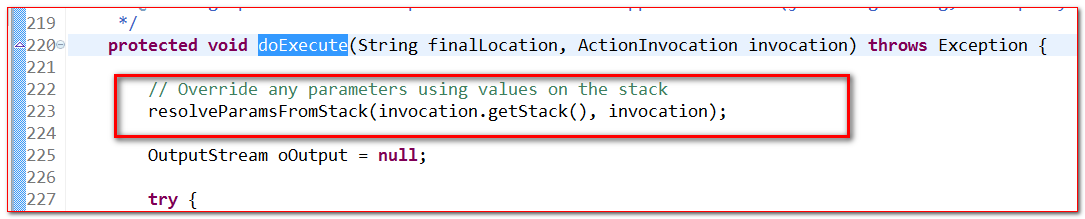
Struts2 实现文件下载，内置了 stream 类型结果集，打开struts-default.xml文件，如下：

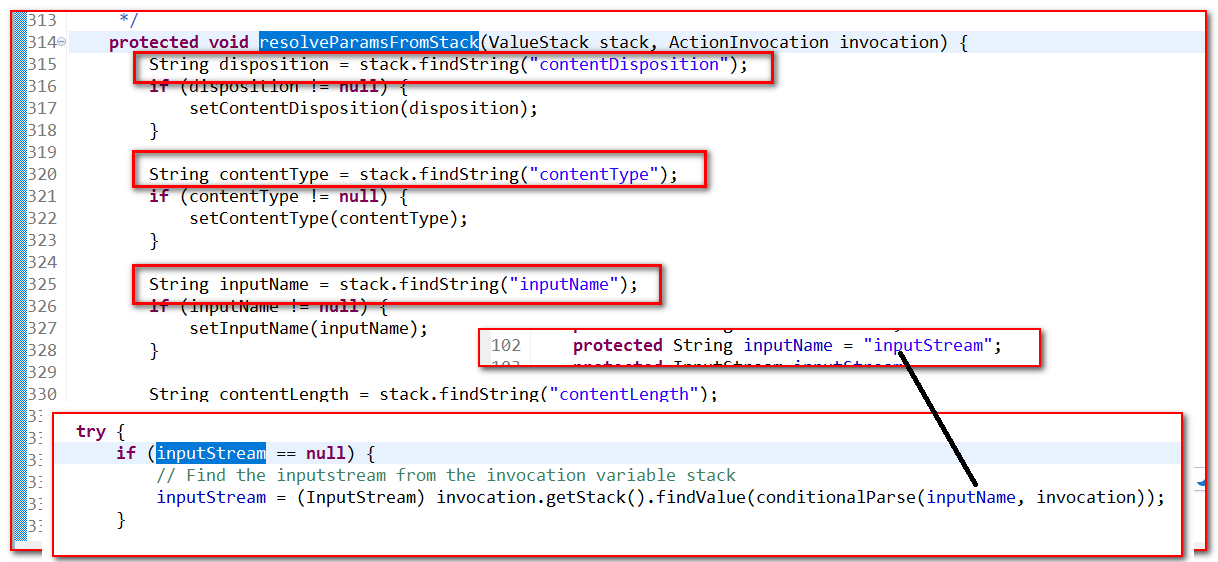


找到StreamResult类



原理：





需要三个东西

* inputName ：用来配置 文件读取输入流 方法的名称 （默认值 inputStream ， 在Action提供 getInputStream 方法）；
* contentType ： 下载文件 Mime类型；

contentDisposition ： 以附件形式下载 （attachment;filename=文件名）。IE 采用 URL编码；火狐 采用 Base64编码；

重点和小结

知识点：

1. 掌握ServletActionContext获取ServletAPI的方式
2. Action 封装请求数据 三种方式 （重点属性驱动的第一种、ModelDriven 方式）

3、 数据类型转换知识点 （了解 ）

4、 掌握结果集类型（4个）

5、 struts2文件下载（了解）