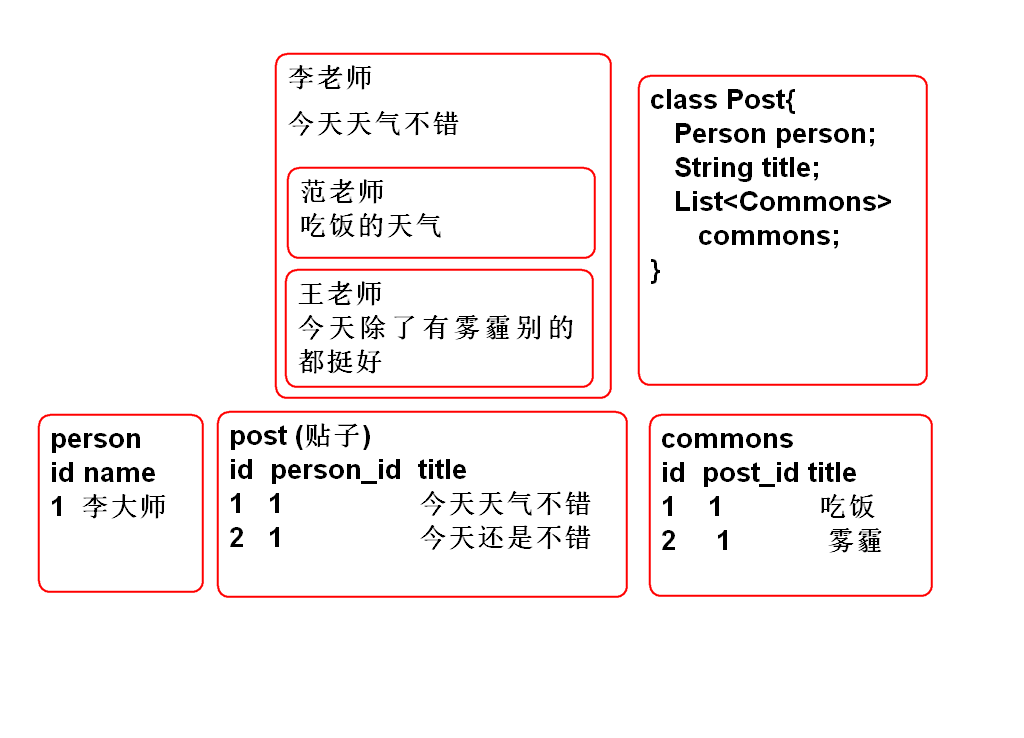
**SSH**

**MyBatis 关联映射**

MyBatis可以处理赋值关系映射:

业务描述:



数据库SQL:

create table p\_person( id int not null AUTO\_INCREMENT, name varchar(100), primary key(id) ); insert into p\_person (name) values ('李老师'); insert into p\_person (id, name) values (null,'李老师'); create table p\_comment( id int not null AUTO\_INCREMENT, title varchar(100), post\_id int, primary key(id) ); create table p\_post( id int not null AUTO\_INCREMENT, title varchar(100), person\_id int, primary key(id) ); insert into p\_post (id, title, person\_id) values (null, '今天天气不错', 1); insert into p\_post (id, title, person\_id) values (null, '高考又来了', 1); insert into p\_comment(id, title, post\_id) values ( null, '少穿了哪一件呀?', 1); insert into p\_comment(id, title, post\_id) values (null, '冻成狗', 1); insert into p\_comment(id, title, post\_id) values (null, '喜欢下雪', 1);

**关联映射步骤**

1. 声明实体类:

public class Person implements Serializable { private static final long serialVersionUID = -2365398342302306276L; private Integer id; private String name; public Person() { } public Person(Integer id, String name) { super(); this.id = id; this.name = name; } public Integer getId() { return id; } public void setId(Integer id) { this.id = id; } public String getName() { return name; } public void setName(String name) { this.name = name; } @Override public String toString() { return "Person [id=" + id + ", name=" + name + "]"; } @Override public int hashCode() { final int prime = 31; int result = 1; result = prime \* result + ((id == null) ? 0 : id.hashCode()); return result; } @Override public boolean equals(Object obj) { if (this == obj) return true; if (obj == null) return false; if (getClass() != obj.getClass()) return false; Person other = (Person) obj; if (id == null) { if (other.id != null) return false; } else if (!id.equals(other.id)) return false; return true; } } public class Comment implements Serializable { private static final long serialVersionUID = -5881249634665160256L; private Integer id; private String title; public Comment() { } public Comment(String title) { super(); this.title = title; } public Integer getId() { return id; } public void setId(Integer id) { this.id = id; } public String getTitle() { return title; } public void setTitle(String title) { this.title = title; } @Override public int hashCode() { final int prime = 31; int result = 1; result = prime \* result + ((id == null) ? 0 : id.hashCode()); return result; } @Override public boolean equals(Object obj) { if (this == obj) return true; if (obj == null) return false; if (getClass() != obj.getClass()) return false; Comment other = (Comment) obj; if (id == null) { if (other.id != null) return false; } else if (!id.equals(other.id)) return false; return true; } @Override public String toString() { return "Comment [id=" + id + ", title=" + title + "]"; } } public class Post implements Serializable{ private static final long serialVersionUID = -4722438109030592372L; private Integer id; private String title; /\*\* 发帖人 \*/ private Person person; /\*\* 当前帖子收到的回复 \*/ private List<Comment> comments = new ArrayList<Comment>(); public Post() { } public Integer getId() { return id; } public void setId(Integer id) { this.id = id; } public String getTitle() { return title; } public void setTitle(String title) { this.title = title; } public Person getPerson() { return person; } public void setPerson(Person person) { this.person = person; } public List<Comment> getComments() { return comments; } public void setComments(List<Comment> comments) { this.comments = comments; } @Override public String toString() { return "Post [id=" + id + ", title=" + title + ", person=" + person + ", comments=" + comments + "]"; } @Override public int hashCode() { final int prime = 31; int result = 1; result = prime \* result + ((id == null) ? 0 : id.hashCode()); return result; } @Override public boolean equals(Object obj) { if (this == obj) return true; if (obj == null) return false; if (getClass() != obj.getClass()) return false; Post other = (Post) obj; if (id == null) { if (other.id != null) return false; } else if (!id.equals(other.id)) return false; return true; } }

1. 添加映射接口:

public interface PostDao { Post findPostById(Integer id); }

1. 添加映射基本属性映射: NoteMapper.xml

<mapper namespace="cn.tedu.note.dao.PostDao"> <!-- 复杂映射必须使用resultMap 进行处理 --> <resultMap type="cn.tedu.note.entity.Post" id="postMap"> <!-- 逐一映射每个属性 --> <!-- 数据库主键, 使用id映射 --> <id column="id" property="id"/> <!-- 非主键, 使用 result 映射--> <result column="title" property="title"/> </resultMap> <select id="findPostById" parameterType="int" resultMap="postMap"> select id, title, person\_id from p\_post where id=#{id} </select> </mapper>

提示: MyBatis不自动支持复杂的映射关联, 复杂的关联关系必须使用resultMap进行手动映射.

1. 测试:

public class PostDaoTest extends BaseTest { PostDao dao; @Before public void initDao(){ dao = ctx.getBean( "postDao", PostDao.class); } @Test public void testFindPostById(){ Post post = dao.findPostById(1); System.out.println(post); } }

测试结果说明, 可以映射基本属性 id和name, 但是无法映射person属性和commons属性.

1. 映射 person 属性, 使用关联映射: 重构PersonMapper.xml

<mapper namespace="cn.tedu.note.dao.PostDao"> <!-- 复杂映射必须使用resultMap 进行处理 --> <resultMap type="cn.tedu.note.entity.Post" id="postMap"> <!-- 逐一映射每个属性 --> <!-- 数据库主键, 使用id映射 --> <id column="id" property="id"/> <!-- 非主键, 使用 result 映射--> <result column="title" property="title"/> <!-- 映射 person 属性使用association(关联)映射 --> <association property="person" javaType="cn.tedu.note.entity.Person"> <id column="person\_id" property="id"/> <result column="name" property="name"/> </association> </resultMap> <select id="findPostById" parameterType="int" resultMap="postMap"> select p\_post.id, title, person\_id, p.name from p\_post left outer join p\_person p on p.id=person\_id where p\_post.id=#{id} </select> </mapper>

首先使用关联SQL查询, 查询出person对象的属性值, 再利用association标签将查询结果映射到Person对象的属性.

1. 测试:

经过测试可用发现已经成功映射了person属性.

1. 映射 comments 属性, 使用collection标签: 重构PostMapper.xml

<mapper namespace="cn.tedu.note.dao.PostDao"> <!-- 复杂映射必须使用resultMap 进行处理 --> <resultMap type="cn.tedu.note.entity.Post" id="postMap"> <!-- 逐一映射每个属性 --> <!-- 数据库主键, 使用id映射 --> <id column="id" property="id"/> <!-- 非主键, 使用 result 映射--> <result column="title" property="title"/> <!-- 映射 person 属性使用association(关联)映射 --> <association property="person" javaType="cn.tedu.note.entity.Person"> <id column="person\_id" property="id"/> <result column="name" property="name"/> </association> <!-- 属性是一个集合, 使用collection 进行 映射处理, 其中 column="id" 是查询参数--> <collection property="comments" select="findCommentsByPostId" column="id"> </collection> </resultMap> <select id="findCommentsByPostId" parameterType="int" resultType="cn.tedu.note.entity.Comment"> select id, title from p\_comment where post\_id=#{id} </select> <select id="findPostById" parameterType="int" resultMap="postMap"> select p\_post.id, title, person\_id, p.name from p\_post left outer join p\_person p on p.id=person\_id where p\_post.id=#{id} </select> </mapper>

首先利用collection表示将comments属性委托到SQL查询 findCommentsByPostId , 再定义SQL查询findCommentsByPostId, 将comment属性的数据查询并且映射到Comment对象.

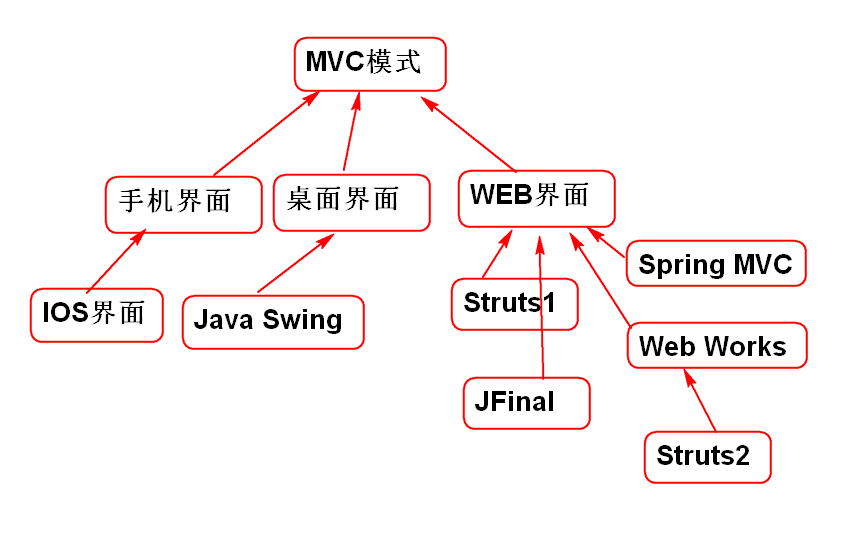
1. 测试

测试结果中将出现comments属性的值.

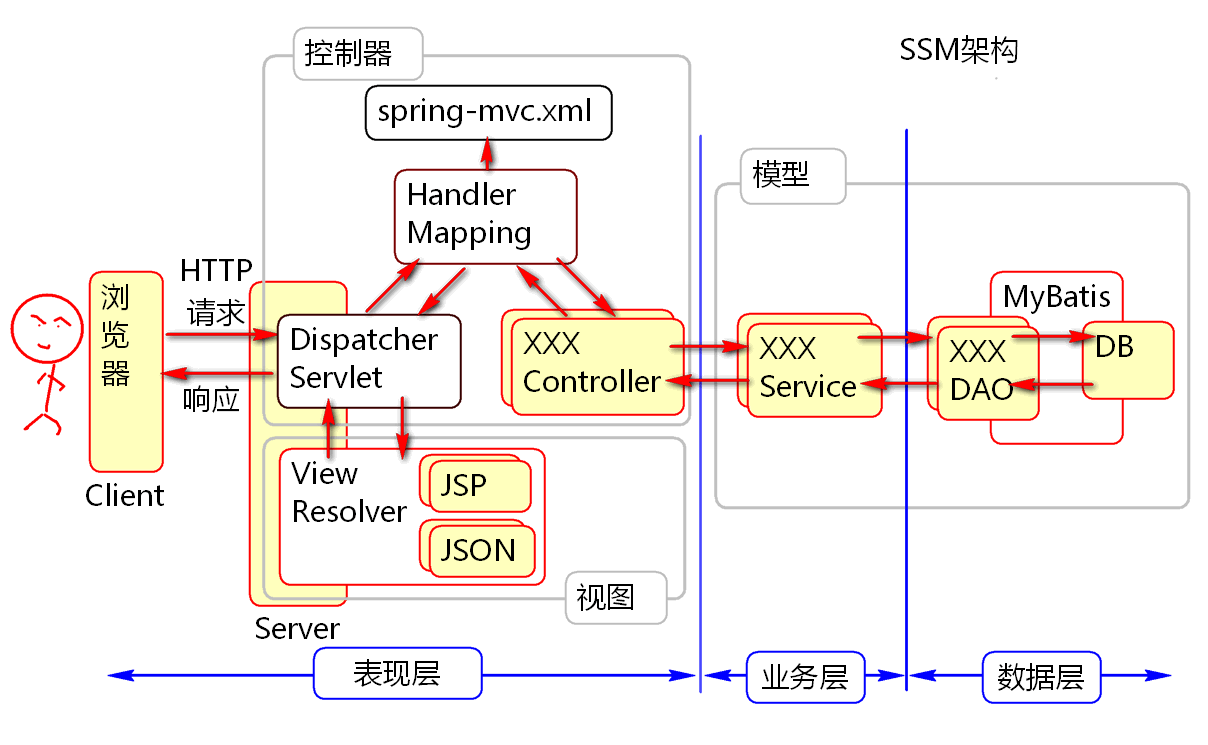
**Struts2**

**MVC**

1. MVC模式是用户界面的经典设计模式! 无论是手机还是桌面以及WEB应用界面都采用了MVC模式.
2. 在Java WEB编程中, 有很多的框架采用了MVC模式.

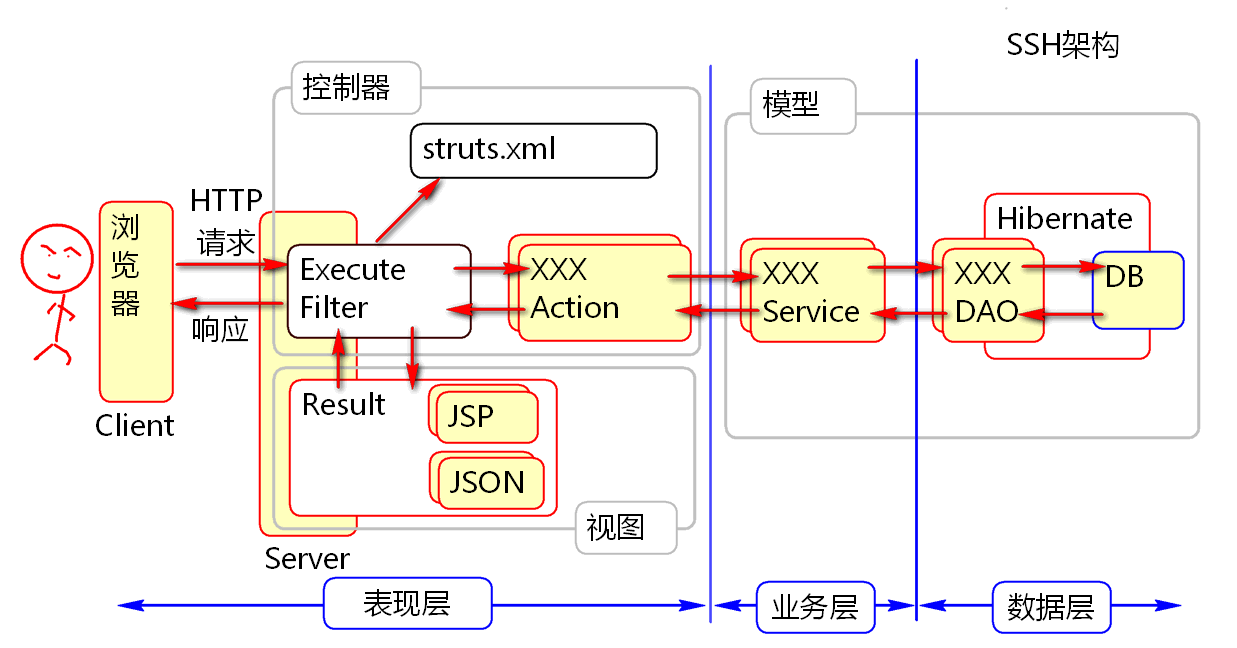


**Spring MVC**



**Struts2 简介**

1. 来自Apache基金会的软件： http://struts.apache.org
2. Struts2 与 Struts1 完全没有关系
3. Struts2 的前身是 WebWorks
4. Spring MVC\Struts2\Struts1 都是 MVC 模式的Web框架
   * MVC是非常流行的 用户界面设计模式。
   * MVC是3层架构中的表现层。



**配置Struts2**

配置步骤：

1. 创建项目, 导入Struts2:

<dependency> <groupId>org.apache.struts</groupId> <artifactId>struts2-core</artifactId> <version>2.3.8</version> </dependency>

1. 配置Struts2 主控制器 web.xml

<filter> <display-name>StrutsPrepareAndExecuteFilter</display-name> <filter-name>StrutsPrepareAndExecuteFilter</filter-name> <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class> </filter> <filter-mapping> <filter-name>StrutsPrepareAndExecuteFilter</filter-name> <url-pattern>/\*</url-pattern> </filter-mapping>

1. 添加Struts2的配置文件 struts.xml:

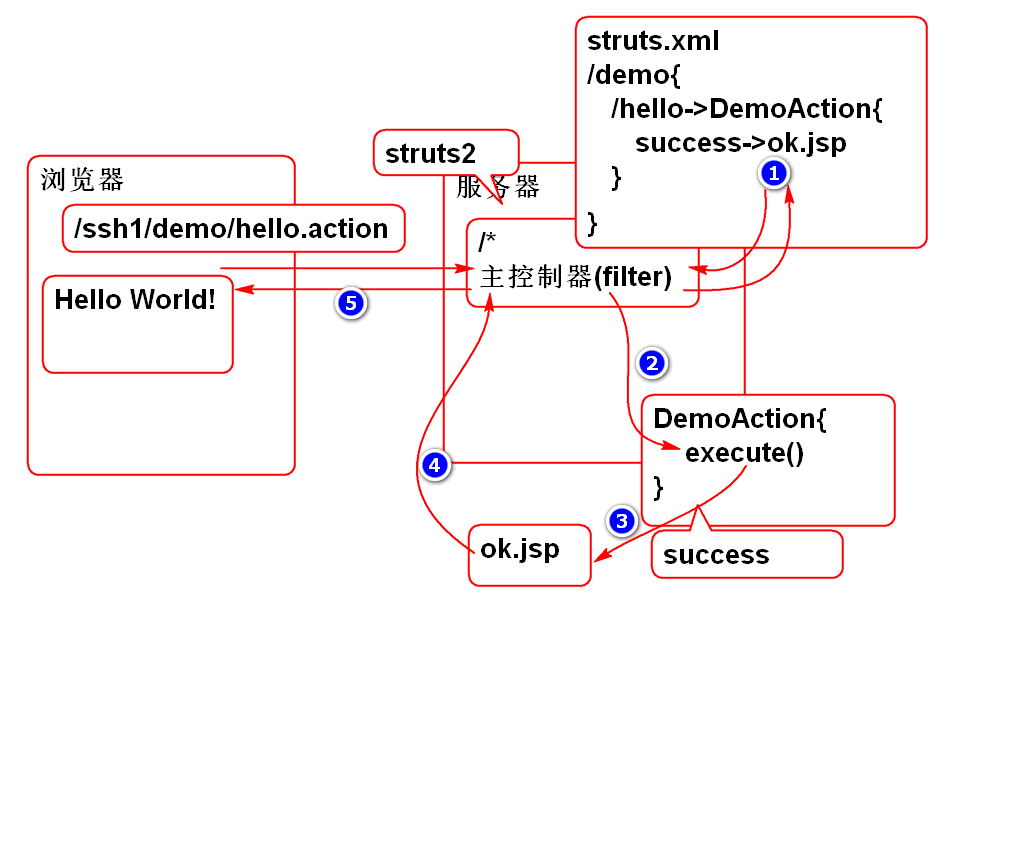
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN" "http://struts.apache.org/dtds/struts-2.3.dtd"> <struts> </struts>

提示: DTD 信息可以从 struts2-core-2.3.8.jar 包的struts-2.3.dtd 文件中找到.

1. 部署到tomcat 进行测试.

**Hello World**

工作原理为:



1. 编写控制器类:

public class DemoAction { /\*\* \* Struts2 控制器中的默认处理方法名为execute \* @return 返回视图的名称 \*/ public String execute(){ System.out.println("Hello World!"); return "success"; } }

默认的控制器方法为execute!

1. 编写jsp: /WEB-INF/jsp/ok.jsp

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%> <!DOCTYPE html> <html> <head> <meta charset="UTF-8"> <title></title> </head> <body> <h1>Struts2</h1> <p>Hello World!</p> </body> </html>

1. 编写配置文件

<struts> <!-- 定义WEB请求的第一层路径 namespace="/demo" --> <package name="demo" namespace="/demo" extends="struts-default" > <action name="hello" class="cn.tedu.controller.DemoAction"> <result name="success"> /WEB-INF/jsp/ok.jsp </result> </action> <!-- 请求 /demo/hello.action 时候, 执行 DemoAction的execute方法, 当方法返回值是 success时候, 转发到ok.jsp --> </package> </struts>

package 的 name 属性任意定义, 不重复即可

1. 部署测试:

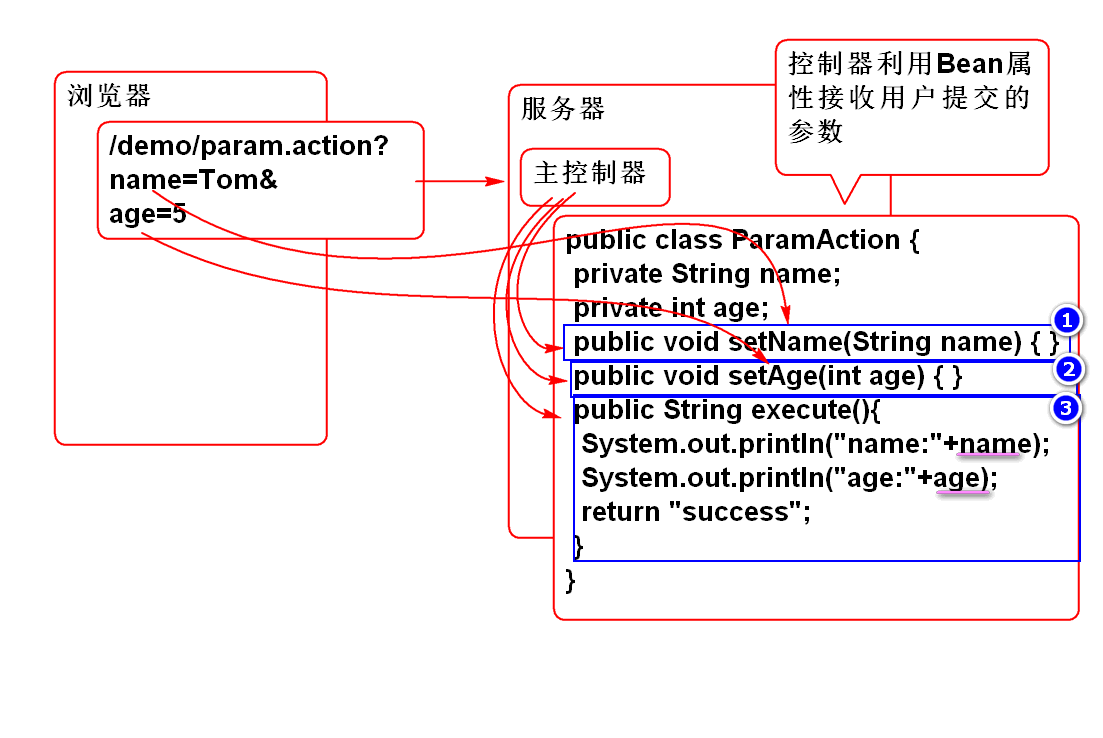
http://localhost:8080/ssh1/demo/hello.action

**接收用户提交的数据**

利用 控制器 的 "Bean属性" 接收用户提交的数据

Bean属性: 是指类上声明的getXXX和setXXX方法

原理为:



案例:

1. 声明控制器

/\*\* \* 利用Bean属性接收用户提交的参数 \*/ public class ParamAction { private String name; private int age; public void setName(String name) { this.name = name; } public void setAge(int age) { this.age = age; } public String execute(){ System.out.println("name:"+name); System.out.println("age:"+age); return "success"; } }

1. 编写JSP: param.jsp

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%> <!DOCTYPE html> <html> <head> <meta charset="UTF-8"> <title>控制器接收参数</title> </head> <body> <h1>Struts2</h1> <p>接收参数</p> </body> </html>

1. 配置控制器:

<!-- 请求路径: /demo/param.action?name=Tom&age=5 --> <action name="param" class="cn.tedu.controller.ParamAction"> <result name="success"> /WEB-INF/jsp/param.jsp </result> </action>

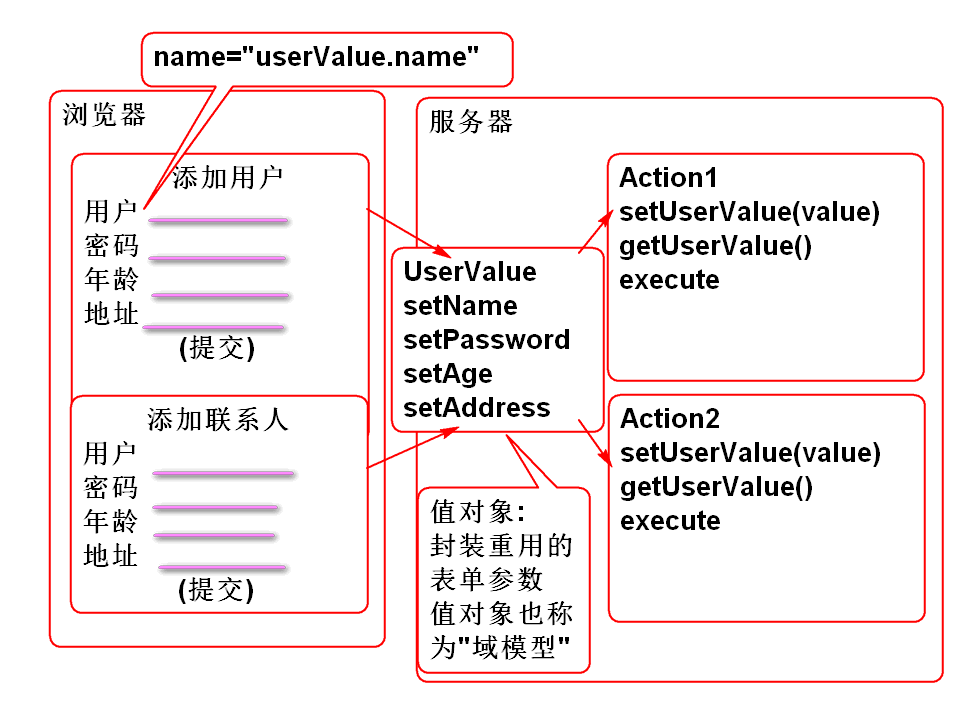
1. 测试:

http://localhost:8080/ssh1/demo/param.action?name=Tom&age=5

参数: Tom 和 5 将出现在 服务器的控制台上

**利用领域模型(值对象)封装用户提交的参数**

当用户提交的表单数据很多, 并且表单项目会重用时候, 可以利用领域模型(值对象)封装用户提交的表单:



案例:

1. 声明领域模型(值对象)

public class UserValue implements Serializable { private static final long serialVersionUID = 232L; private String name; private String password; private Integer age; private String address; public UserValue() { } public String getName() { return name; } public void setName(String name) { this.name = name; } public String getPassword() { return password; } public void setPassword(String password) { this.password = password; } public Integer getAge() { return age; } public void setAge(Integer age) { this.age = age; } public String getAddress() { return address; } public void setAddress(String address) { this.address = address; } @Override public String toString() { return "UserValue [name=" + name + ", password=" + password + ", age=" + age + ", address=" + address + "]"; } }

1. 定义控制器, 接收领域模型参数:

public class UserAction { private UserValue userValue; public void setUserValue(UserValue userValue) { this.userValue = userValue; } public UserValue getUserValue() { return userValue; } public String execute(){ System.out.println(userValue); return "success"; } }

1. 开发JSP页面, 用于提交表单参数:

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%> <!DOCTYPE html> <html> <head> <meta charset="UTF-8"> <title></title> </head> <body> <h1>提交用户信息</h1> <form action="demo/user.action" method="post"> <div> 用户名:<input type="text" name="userValue.name"> </div> <div> 密码:<input type="password" name="userValue.password"> </div> <div> 年龄:<input type="text" name="userValue.age"> </div> <div> 地址: <input type="text" name="userValue.address"> </div> <div> <input type="submit" value="保存"> </div> </form> </body> </html>

表单中的input元素name属性必须写为 name="userValue.address" 这样才能传递参数.

1. 配置控制器 struts.xml

<!-- 利用值对象打包传递参数 --> <action name="user" class="cn.tedu.controller.UserAction"> <result name="success"> /WEB-INF/jsp/ok.jsp </result> </action>

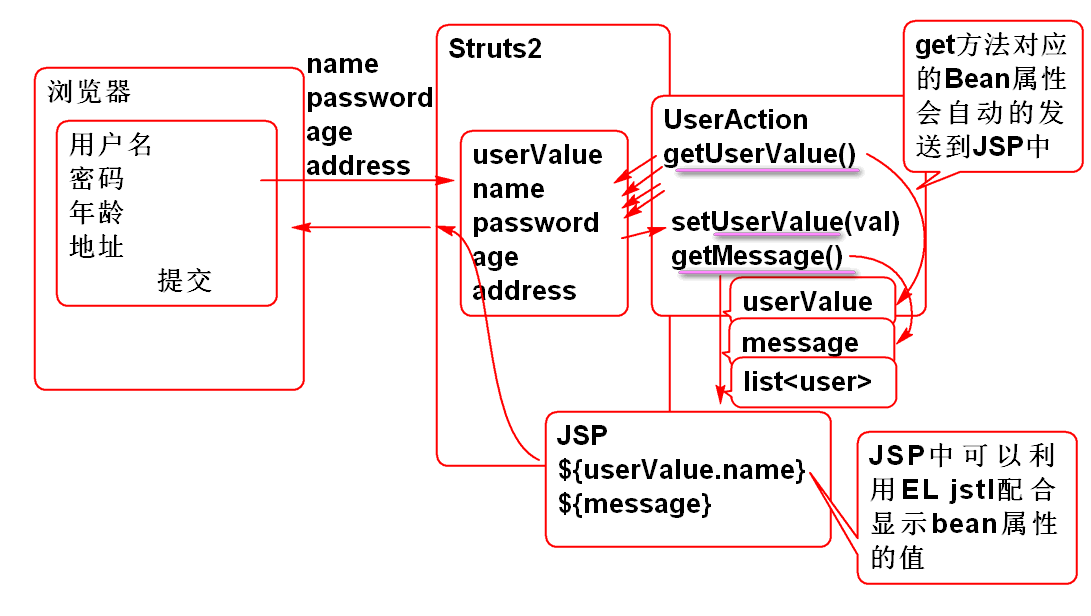
1. 测试

利用表单提交参数, 服务端手动打包的van是

**将服务端的数据显示到页面上**

控制器中的bean属性getXXX方法会自动传递到JSP中, 其底层是利用request传递的.

原理:



案例:

1. 重构 UserAction, 添加 bean 属性 message:

private String message; public String getMessage() { return message; } public String execute(){ System.out.println(userValue); message = "成功!"; //转发目标JSP收到两个Bean对象: // message 和 userValue //对应着 Action 对象的 getXXX方法 //在JSP中可以使用 EL 表达式访问这些属性 return "success"; }

在控制器方法中为message属性赋值

1. 编写数据展示页面 success.jsp

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%> <!DOCTYPE html> <html> <head> <meta charset="UTF-8"> <title></title> </head> <body> <h1>表单结果</h1> <p>收到了两个bean对象: message, userValue</p> <p>消息: ${message}</p> <p>用户名: ${userValue.name}</p> <p>密码: ${userValue.password}</p> <p>年龄: ${userValue.age}</p> <p>地址: ${userValue.address}</p> </body> </html>

在JSP中可以利用EL表达式和 JSTL 配合显示控制器传递来的属性.

1. 重构配置文件, UserAction 控制器处理结束后转发到 success.jsp

<!-- 利用值对象打包传递参数 --> <action name="user" class="cn.tedu.controller.UserAction"> <result name="success"> /WEB-INF/jsp/success.jsp </result> </action>

1. 测试

控制器的数据可以成功的显示到JSP页面上.

**Struts 2 控制器中访问 Session对象**

在 Struts 2 控制器用两种访问Session的方式:

1. 利用ActionContext 获取Session.
   * 代表当前控制器的工作环境: 被那个主控制器调用, 当前的request, 当前的resopnse, 当前session等信息, 都可以从 ActionContext 获得.

ActionContext ctx = ActionContent.getContext(); Map<String, Object> session=ctx.getSession(); //session 是 map类型的, 如果把数据保存到map中, struts 2 会 // 自动的将数据存储到 HttpSession对象中 session.put("loginUser", "Tom"); //转发到 JSP 后可以使用EL读取session ${loginUser}

案例:

1. 重构UserAction, 在转发JSP之前在session 中添加属性:

public String execute(){ System.out.println(userValue); message = "成功!"; //转发目标JSP收到两个Bean对象: // message 和 userValue //对应着 Action 对象的 getXXX方法 //在JSP中可以使用 EL 表达式访问这些属性 //将数据保存到 session ActionContext context = ActionContext.getContext(); Map<String, Object> session = context.getSession(); session.put("loginUser", "Tom"); //struts 2 会自动的将 map 中的数据同步到 // HttpSession 中 //转发到JSP 后可以使用EL获取 session 中的数据 // ${loginUser} return "success"; }

1. 重构 success.jsp 利用EL 显示session中的属性:

<p>Session: ${loginUser} </p>

1. 测试

可以再success.jsp显示session中的数据

**作业**

1. 完成MyBatis的复杂关联查询
2. 搭建Struts2项目, 完成Struts2 案例.

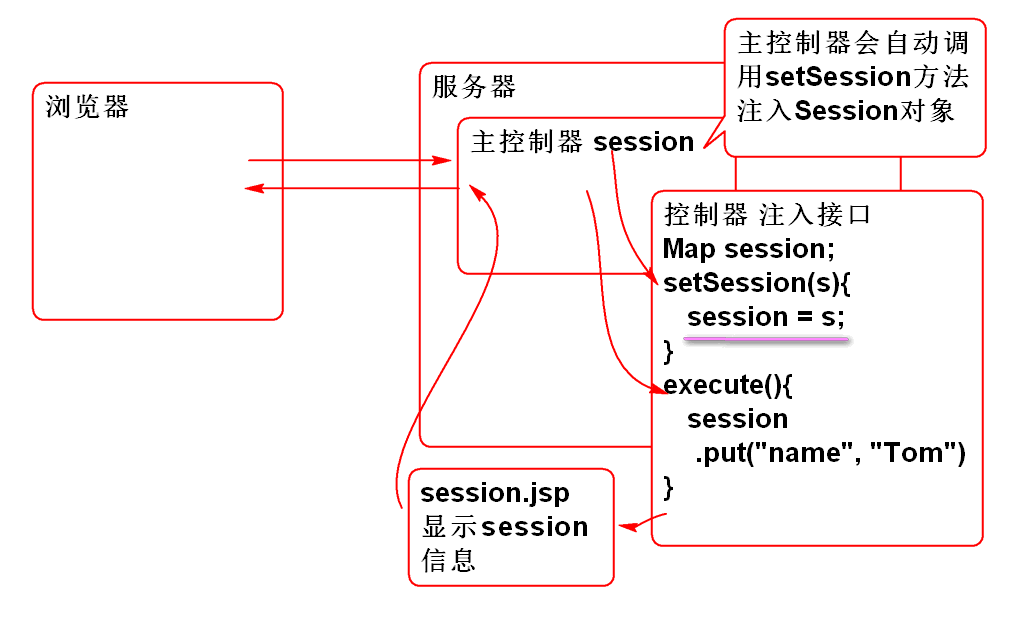
**SSH**

**Struts2 中访问 session**

**利用SessionAware接口注入Session对象**

Struts 2 的控制器只要实现 SessionAware接口, 实现其Bean属性注入方法 setSession, Struts 2 主控制器就会在执行控制器方法之前将Session对象注入到控制器中.

原理:



案例:

1. 声明控制器, 实现SessionAware接口:

/\*\* \* 自动注入Session对象. \* Struts主控制器在发现控制器类, SessionController \* 实现了SessionAware时候, 会自动的将Session对象 \* 注入到 控制器对象中. 注入时候调用的方法是setSession \*/ public class SessionAction implements SessionAware{ private Map<String, Object> session; public void setSession( Map<String, Object> session) { this.session=session; } public String demo(){ session.put("name", "Tom"); return "success"; } }

1. 编写success.jsp测试, session中存储的数据:

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%> <!DOCTYPE html> <html> <head> <meta charset="UTF-8"> <title>Session Demo</title> </head> <body> <h1>Session</h1> <p>session数据: ${name}</p> </body> </html>

1. 编辑配置文件, struts.xml

<action name="session" class="cn.tedu.controller.SessionAction" method="demo"> <result name="success"> /WEB-INF/jsp/session.jsp </result> </action>

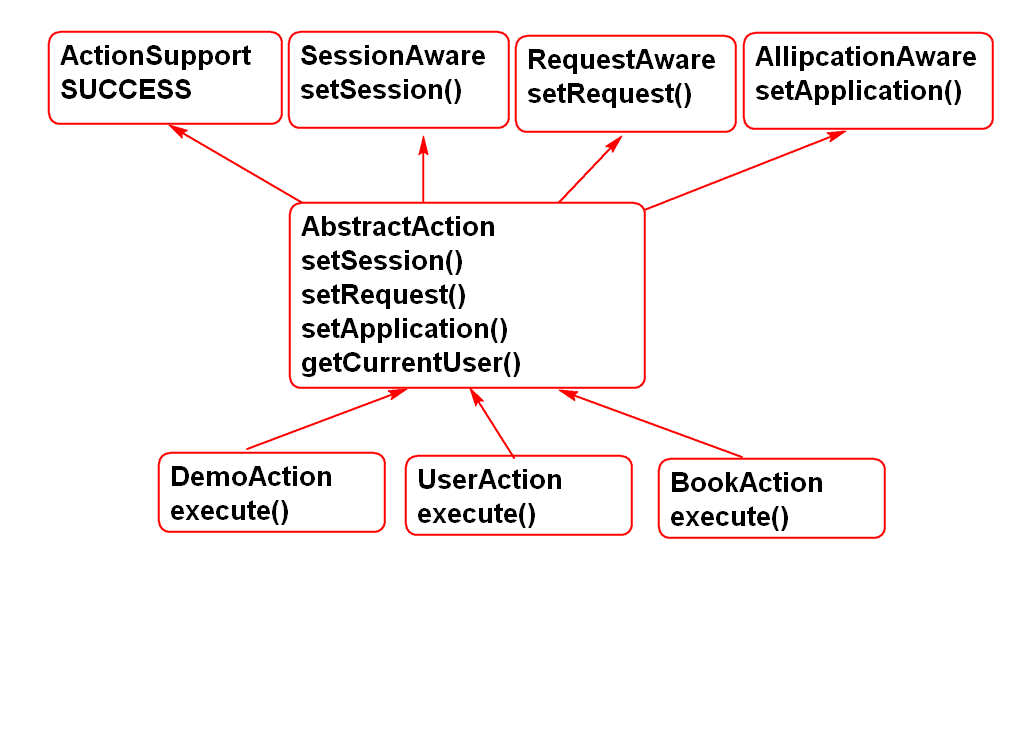
1. 测试

http://localhost:8080/ssh2/demo/session.action

**抽取AbstractAction类, 简化控制器**

为了重用SessionAware注入代码, 简化控制器编写, 在实际工作中经常抽取AbstractAction, 封装控制器的公共代码, 简化控制器的开发.

Struts 2 也提供了一个 ActionSupport 类, 建议作为Action的父类, 其作用也是封装了 Action常用功能, 简化Action的开发.



案例:

public abstract class AbstractAction extends ActionSupport implements SessionAware { protected Map<String, Object> session; public void setSession( Map<String, Object> session) { this.session=session; } }

Struts2 还提供了 RequestAware 和 ApplicationAware, 用于注入 request和application, 为了方便控制器的编程, 可以将这些接口在AbstractAction上实现:

public abstract class AbstractAction extends ActionSupport implements SessionAware, RequestAware, ApplicationAware{ protected Map<String, Object> request; protected Map<String, Object> session; protected Map<String, Object> application; public void setSession( Map<String, Object> session) { this.session=session; } public void setRequest( Map<String, Object> request) { this.request=request; } public void setApplication( Map<String, Object> application) { this.application=application; } }

这样控制器可以简化为:

public class DemoAction extends AbstractAction{ public String execute(){ request.put("myName", "Wang"); session.put("name", "Andy"); application.put("test", "熊大"); return SUCCESS; } }

配置控制器:

<action name="demo" class="cn.tedu.controller.DemoAction"> <result name="success"> /WEB-INF/jsp/session.jsp </result> </action>

session.jsp

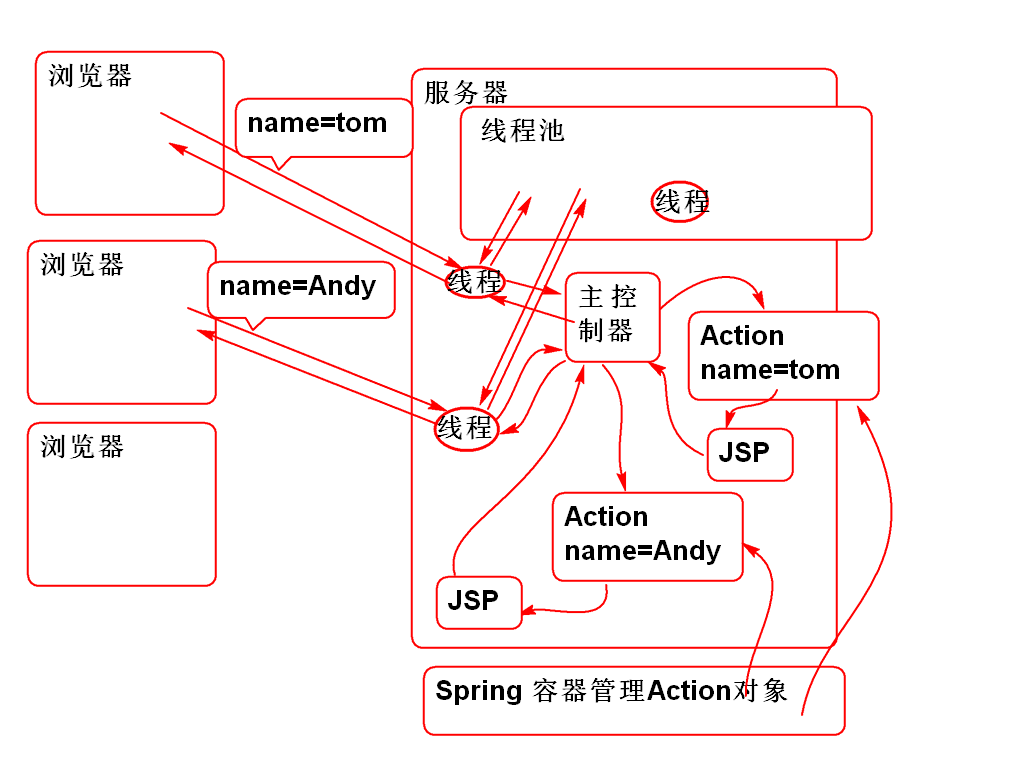
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%> <!DOCTYPE html> <html> <head> <meta charset="UTF-8"> <title>Session Demo</title> </head> <body> <h1>Session</h1> <p>request数据: ${myName}</p> <p>session数据: ${name}</p> <p>application数据: ${test}</p> </body> </html>

可以体会到在控制器中访问 session request 和 application的便捷性.

**控制器的线程安全(重要)**

Struts 2 为了保护控制器的线程安全, 每个用户请求创建一个新的 控制器(Action)对象.

原理:



**Spring 和 Struts 2 整合**

Spring 将接管 Struts 2 控制器对象的管理, 帮助创建 Struts 2 控制器Bean对象.

整合步骤:

1. 导入包 Struts 2 提供的整合包

<dependency> <groupId>org.apache.struts</groupId> <artifactId>struts2-spring-plugin</artifactId> <version>2.3.8</version> </dependency>

导入以后, 无需配置, 自动工作!

1. 导入Spring包, 和配置Spring容器.

Spring 包已经自动依赖, 无需再次导入

<listener> <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class> </listener> <context-param> <param-name>contextConfigLocation</param-name> <param-value>classpath:spring-\*.xml</param-value> </context-param>

其中 ContextLoaderListener 类要求配置参数 contextConfigLocation

添加Spring配置文件 spring-web.xml

<?xml version="1.0" encoding="UTF-8"?> <beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context" xmlns:jdbc="http://www.springframework.org/schema/jdbc" xmlns:jee="http://www.springframework.org/schema/jee" xmlns:tx="http://www.springframework.org/schema/tx" xmlns:aop="http://www.springframework.org/schema/aop" xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:util="http://www.springframework.org/schema/util" xmlns:jpa="http://www.springframework.org/schema/data/jpa" xsi:schemaLocation=" http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-3.0.xsd http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd http://www.springframework.org/schema/data/jpa http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.0.xsd"> </beans>

当前Spring版本是3.0版本, 配置文件中的Schema文件版本也要改成3.0版本的! 否则无法使用.

部署到容器进行测试...

1. 利用Spring 管理 控制器Bean组件, 注意这些bean不能是单例的, 否则有线程安全问题!!

@Controller @Scope("prototype") public class HelloAction extends AbstractAction { public String execute(){ System.out.println("Hello World!"); return SUCCESS; } }

在spring 配置文件中配置组件扫描功能 spring-web.xml

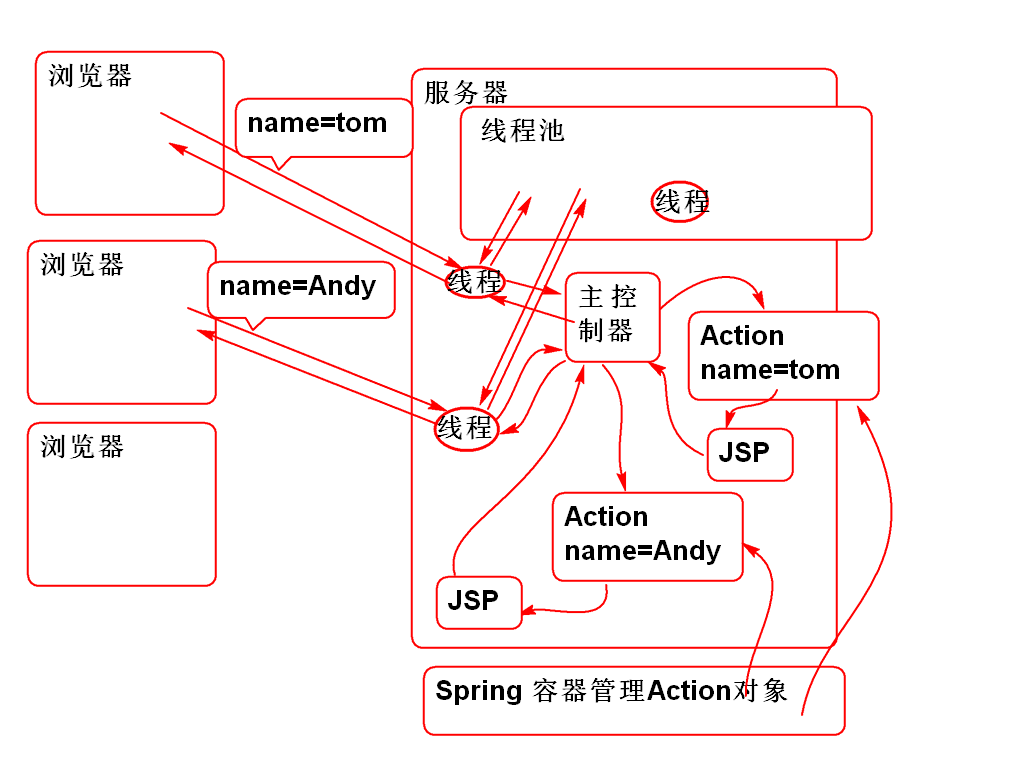
<context:component-scan base-package="cn.tedu.controller"/>

1. 在Struts2 配置文件中, class 属性替换为 Spring Bean ID 即可使用Spring 创建的Bean组件了.

<!-- 利用Spring作为控制器Action的容器 只需要使用class="Bean ID"--> <action name="hello" class="helloAction"> <result type="dispatcher" name="success"> /WEB-INF/jsp/ok.jsp </result> </action>

**为什么要将 Spring 和 Struts2 整合**

1. Spring 框架核心功能: IOC/DI AOP
   * IOC/DI 管理对象, 管理对象和对象的关系, 可以解除耦合
   * AOP
2. Struts2 可以简化WEB编程.
3. 整合以后:
   * 简化了Web开发
   * 简化业务层和持久层组件的开发.



**Result**

Struts 提供了丰富的Result支持, 在struts的默认配置文件struts-default.xml中声明了这些result:

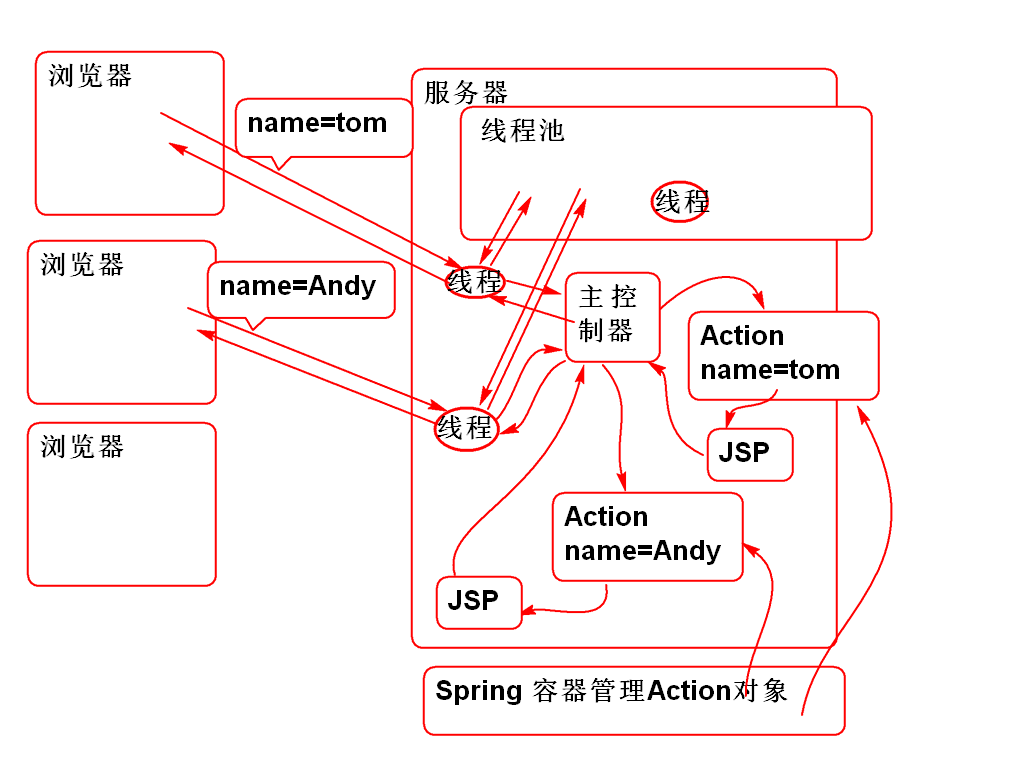
<package name="struts-default" abstract="true"> <result-types> <result-type name="chain" class="com.opensymphony.xwork2.ActionChainResult"/> <result-type name="dispatcher" class="org.apache.struts2.dispatcher.ServletDispatcherResult" default="true"/> <result-type name="freemarker" class="org.apache.struts2.views.freemarker.FreemarkerResult"/> <result-type name="httpheader" class="org.apache.struts2.dispatcher.HttpHeaderResult"/> <result-type name="redirect" class="org.apache.struts2.dispatcher.ServletRedirectResult"/> <result-type name="redirectAction" class="org.apache.struts2.dispatcher.ServletActionRedirectResult"/> <result-type name="stream" class="org.apache.struts2.dispatcher.StreamResult"/> <result-type name="velocity" class="org.apache.struts2.dispatcher.VelocityResult"/> <result-type name="xslt" class="org.apache.struts2.views.xslt.XSLTResult"/> <result-type name="plainText" class="org.apache.struts2.dispatcher.PlainTextResult" /> </result-types> ... > 可以看到在struts-default包中定义了这些Result类型, 只要继承struts-default既可以继承这些Result类型. > 每个Result类型对应一个类, Result类型的具体功能是由这个类提供算法支持. 如: dispatcher 类型的功能, 由 org.apache.struts2.dispatcher.ServletDispatcherResult 类来处理. > 这个配置也说明: 如果在使用Struts2时候, 对其提供的Result类型不满意, 可以自行添加类进行扩展, 扩展出自己的Result类型.

常用的类型有:

1. dispatcher
2. redirect
3. redirectAction
4. stream
5. json

**dispatcher**

dispatcher 类型是默认的结果类型, 其处理规则是将控制器转发到目标JSP.

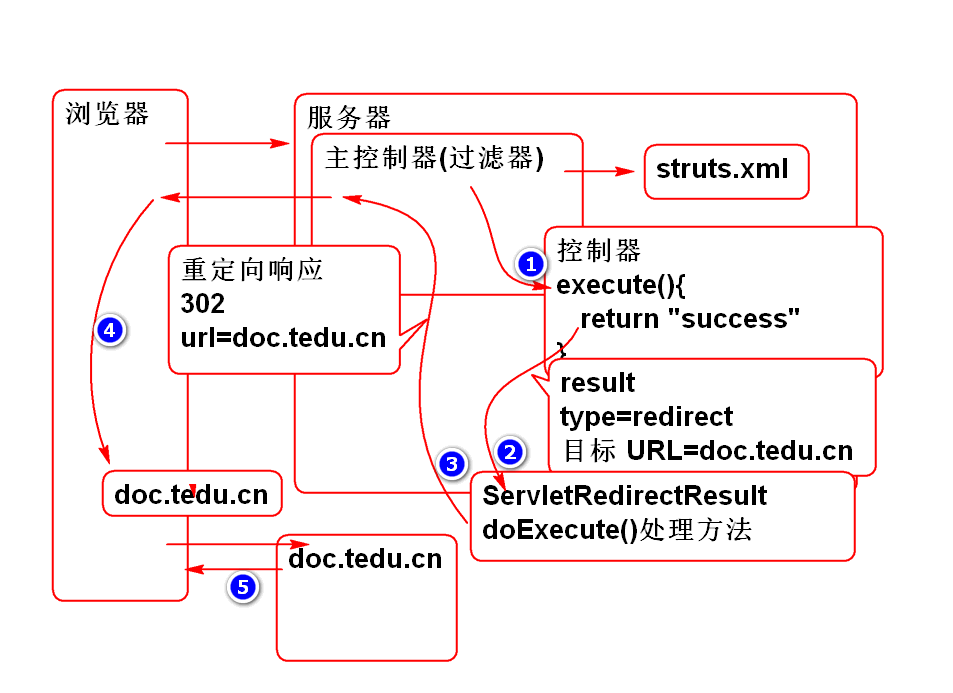


案例:

<action name="demo" class="cn.tedu.controller.DemoAction"> <result name="success"> /WEB-INF/jsp/session.jsp </result> </action>

**redirect**

就是重定向, 控制器处理以后, 重定向到其他URL目标:



案例:

1. 编写控制器

@Controller @Scope("prototype") public class RedirectDemoAction extends AbstractAction{ public String execute() { System.out.println("重定向请求"); return SUCCESS; } }

1. 配置 struts.xml

<!-- 重定向演示 --> <action name="doc" class="redirectDemoAction"> <result type="redirect" name="success"> <param name="location"> http://doc.tedu.cn </param> </result> </action>

1. 测试:

http://localhost:8080/ssh2/demo/doc.action

**redirectAction**

与 redirect 类型, 只是重定向的目标不是url而是 action

案例:

1. 编写控制器类

@Controller @Scope("prototype") public class RedirectActionDemoAction extends AbstractAction{ public String execute(){ System.out.println("重定向到其他控制器"); return SUCCESS; } }

1. 配置 struts.xml

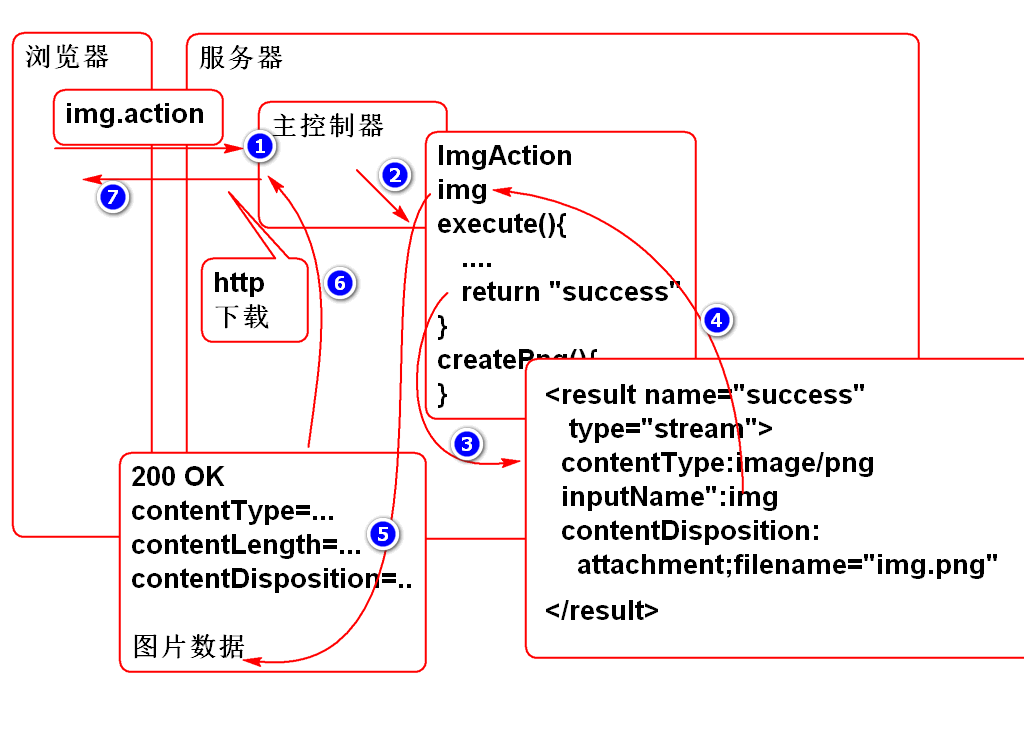
<!-- 重定向到其他的控制器 --> <action name="redirect" class="redirectActionDemoAction"> <result type="redirectAction" name="success"> <param name="actionName">hello</param> <param name="namespace">/demo</param> </result> </action>

1. 测试:

http://localhost:8080/ssh2/demo/redirect.action

**stream**

stream 可以用于HTTP下载 文件:



将一个图片显示发送到客户端:

1. 编写控制器:

@Controller @Scope("prototype") public class ImgAction extends AbstractAction { private InputStream img; public InputStream getImg() { return img; } public void setImg(InputStream img) { this.img = img; } private byte[] createPng() throws IOException{ BufferedImage img = new BufferedImage(200, 60, BufferedImage.TYPE\_3BYTE\_BGR); Random r = new Random(); for(int i=0; i<100; i++){ int x = r.nextInt(img.getWidth()); int y = r.nextInt(img.getHeight()); int rgb=r.nextInt(0xffffff); img.setRGB(x, y, rgb); } ByteArrayOutputStream out = new ByteArrayOutputStream(); ImageIO.write(img, "png", out); out.close(); return out.toByteArray(); } public String execute() throws IOException{ byte[] png=createPng(); img = new ByteArrayInputStream(png); return SUCCESS; } }

1. 配置 struts.xml

<!-- 将图片发送到浏览器 --> <action name="img" class="imgAction"> <result name="success" type="stream"> <param name="inputName">img</param> <param name="contentType">image/png</param> </result> </action>

注意: img 是控制器ImgAction的bean属性, 其类型必须是inputStream

1. 测试:

http://localhost:8080/ssh2/demo/img.action

将图片下载到浏览器:

1. 控制器, 重用ImgAction
2. 配置struts.xml

<!-- 浏览器端自动下载图片 --> <action name="download" class="imgAction"> <result name="success" type="stream"> <param name="inputName">img</param> <param name="contentType"> application/octet-stream </param> <param name="contentDisposition"> attachment; filename="demo.png" </param> </result> </action>

根据Http协议, 设置contentType 和 contentDisposition, HTTP协议具体内容请参考RFC2616.

1. 测试

http://localhost:8080/ssh2/demo/download.action

下载Excel文件:

1. 导入POI API

<dependency> <groupId>org.apache.poi</groupId> <artifactId>poi</artifactId> <version>3.13</version> </dependency>

1. 编写控制器

@Controller @Scope("prototype") public class ExcelAction extends AbstractAction { private InputStream excel; public InputStream getExcel() { return excel; } public void setExcel(InputStream excel) { this.excel = excel; } private byte[] createExcel() throws IOException{ //创建工作簿 HSSFWorkbook workbook=new HSSFWorkbook(); //创建工作表 HSSFSheet sheet= workbook.createSheet("Demo"); //在工作表中创建数据行 HSSFRow row = sheet.createRow(0); //创建行中的格子 HSSFCell cell = row.createCell(0); cell.setCellValue("Hello World!"); //将Excel文件保存为 byte 数组 ByteArrayOutputStream out = new ByteArrayOutputStream(); workbook.write(out); out.close(); return out.toByteArray(); } public String execute() throws IOException{ byte[] buf = createExcel(); excel = new ByteArrayInputStream(buf); return SUCCESS; } }

1. 配置 struts.xml

<!-- 浏览器端自动下载Excel --> <action name="excel" class="excelAction"> <result name="success" type="stream"> <param name="inputName">excel</param> <param name="contentType"> application/octet-stream </param> <param name="contentDisposition"> attachment; filename="demo.xls" </param> </result> </action>

1. 测试

http://localhost:8080/ssh2/demo/excel.action

**json**

json Result 不是Struts2 内嵌结果类型, 需要导入 struts2-json-plugin:

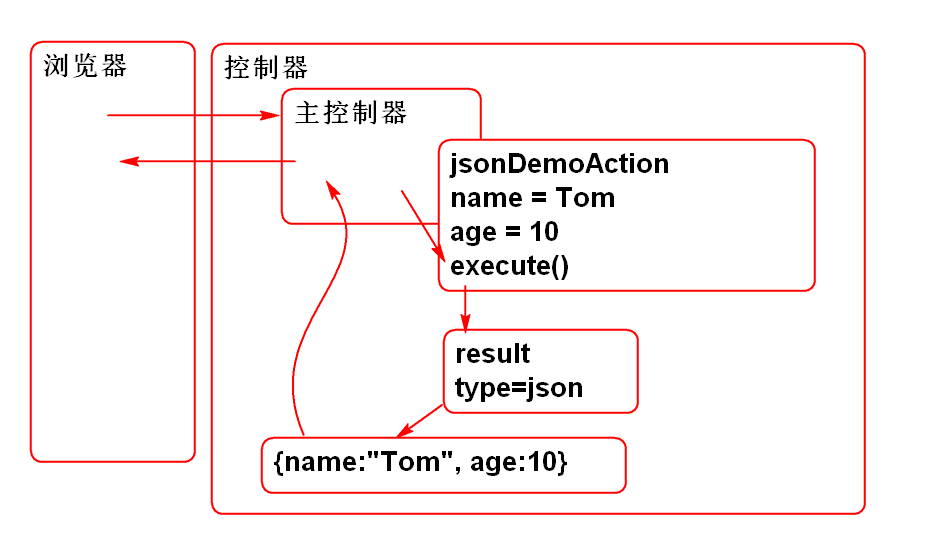
<dependency> <groupId>org.apache.struts</groupId> <artifactId>struts2-json-plugin</artifactId> <version>2.3.8</version> </dependency>

在 struts2-json-plugin-2.3.8.jar 包中的struts-plugin.xml文件中声明了 json类型的result:

<package name="json-default" extends="struts-default"> <result-types> <result-type name="json" class="org.apache.struts2.json.JSONResult"/> </result-types> ...

使用时候需要将 package 继承于 json-default 就可以使用 json Result, 由于json-default继承于struts-default, 这样也可以使用struts-default中定义的result了.

Json Result 原理:



使用Json Result

1. 编写控制器

@Controller @Scope("prototype") //转换为json: {name:"Tom", age:10} public class JsonDemoAction { private String name; private int age; public String getName() { return name; } public void setName(String name) { this.name = name; } public int getAge() { return age; } public void setAge(int age) { this.age = age; } public String execute(){ name = "Tom"; age = 10; return "success"; } }

1. 配置:

<package name="json" namespace="/json" extends="json-default"> <action name="jsonDemo" class="jsonDemoAction"> <result name="success" type="json"> </result> </action> </package>

1. 测试

http://localhost:8080/ssh2/json/jsonDemo.action

将jsonDemoAction的整体作为Java Bean 转换为json字符串发送到浏览器 jsonDemoAction中的Bean属性会转换为 json字符串中的属性

以上案例问题是不能精确控制器哪些Bean属性转换为json字符串, 所以一般将指定属性转换为JSON

1. 编写控制器:

@Controller @Scope("prototype") public class RootDemoAction extends AbstractAction{ private Object jsonResult; public void setJsonResult(Object jsonResult) { this.jsonResult = jsonResult; } public Object getJsonResult() { return jsonResult; } public String execute(){ String[] ary={"Tom", "Andy"}; jsonResult = ary; return SUCCESS; } }

1. 配置:

<!-- 指定Action中的一个Bean属性, 转换为 json字符串, 发送到客户端, jsonResult 是Action中的Bean属性名 --> <action name="rootDemo" class="rootDemoAction"> <result name="success" type="json"> <param name="root">jsonResult</param> </result> </action>

1. 测试:

http://localhost:8080/ssh2/json/rootDemo.action

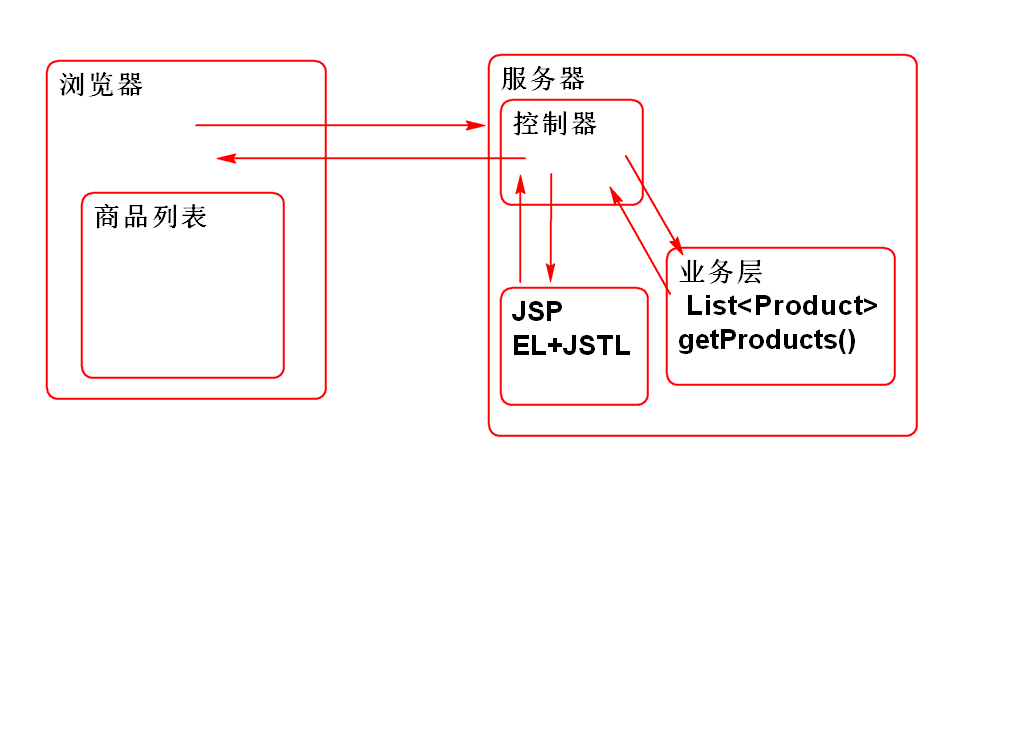
**作业**

1. 搭建Struts2 + Spring + JSON 整合开发环境
2. 实现控制器将List作为JSON返回值

**SSH**

**Struts 简单应用**

实现一个简单的列表功能



1. 创建项目导入相关的包:

<dependency> <groupId>org.apache.struts</groupId> <artifactId>struts2-core</artifactId> <version>2.3.8</version> </dependency> <dependency> <groupId>org.apache.struts</groupId> <artifactId>struts2-spring-plugin</artifactId> <version>2.3.8</version> </dependency> <dependency> <groupId>jstl</groupId> <artifactId>jstl</artifactId> <version>1.2</version> </dependency>

导入目标服务器运行环境 Tomcat Runtime

1. 配置Struts2 和 Spring: web.xml

<filter> <display-name>StrutsPrepareAndExecuteFilter</display-name> <filter-name>StrutsPrepareAndExecuteFilter</filter-name> <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class> </filter> <filter-mapping> <filter-name>StrutsPrepareAndExecuteFilter</filter-name> <url-pattern>/\*</url-pattern> </filter-mapping> <listener> <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class> </listener> <context-param> <param-name>contextConfigLocation</param-name> <param-value>classpath:spring-\*.xml</param-value> </context-param>

1. 添加 Spring 和 Struts2 的配置文件

spring-web.xml

<?xml version="1.0" encoding="UTF-8"?> <beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context" xmlns:jdbc="http://www.springframework.org/schema/jdbc" xmlns:jee="http://www.springframework.org/schema/jee" xmlns:tx="http://www.springframework.org/schema/tx" xmlns:aop="http://www.springframework.org/schema/aop" xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:util="http://www.springframework.org/schema/util" xmlns:jpa="http://www.springframework.org/schema/data/jpa" xsi:schemaLocation=" http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-3.0.xsd http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd http://www.springframework.org/schema/data/jpa http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.0.xsd"> <context:component-scan base-package="cn.tedu.controller"/> <context:component-scan base-package="cn.tedu.service"/> </beans>

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> </struts>

1. 创建业务层:

@Service public class ProductService { List<Product> products; @PostConstruct private void init(){ products = new ArrayList<Product>(); products.add(new Product( 1,"大黄蜂",24.5,"变形金刚经典系列")); products.add(new Product( 2,"擎天柱",88.5,"变形金刚经典系列")); products.add(new Product( 3,"光头强",12.5,"大笨熊你粗来!")); products.add(new Product( 4,"李老师",22.5,"胡说派诗人!")); products.add(new Product( 5,"范传奇",23.5,"变形金刚消费者!")); } public List<Product> getProducts(){ return products; } }

1. 创建控制器

@Controller @Scope("prototype") public class ProductAction { @Resource private ProductService productService; private List<Product> products; public List<Product> getProducts() { return products; } public void setProducts(List<Product> products) { this.products = products; } private String name; public String getName() { return name; } public void setName(String name) { this.name = name; } //控制器方法 public String list(){ name = "list"; products = productService.getProducts(); ActionContext context = ActionContext.getContext(); context.getSession().put( "loginUser", "老王"); return "success"; } }

1. 创建 list.jsp

<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%> <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> <!DOCTYPE html> <html> <head> <meta charset="utf-8"> <title>产品管理</title> <style type="text/css"> table { border-collapse: collapse; } td, th{ padding: 5px 10px; } th{ border-bottom: 2px solid #ddd; } td{ border-top: 1px solid #ccc; } </style> </head> <body> <h1>产品管理</h1> <table> <thead> <tr> <th>编号</th> <th>名称</th> <th>价格</th> <th>描述</th> </tr> </thead> <tbody> <c:forEach items="${products}" var="p"> <tr> <td>${p.id}</td> <td>${p.name}</td> <td>${p.price}</td> <td>${p.description}</td> </tr> </c:forEach> </tbody> </table> </body> </html>

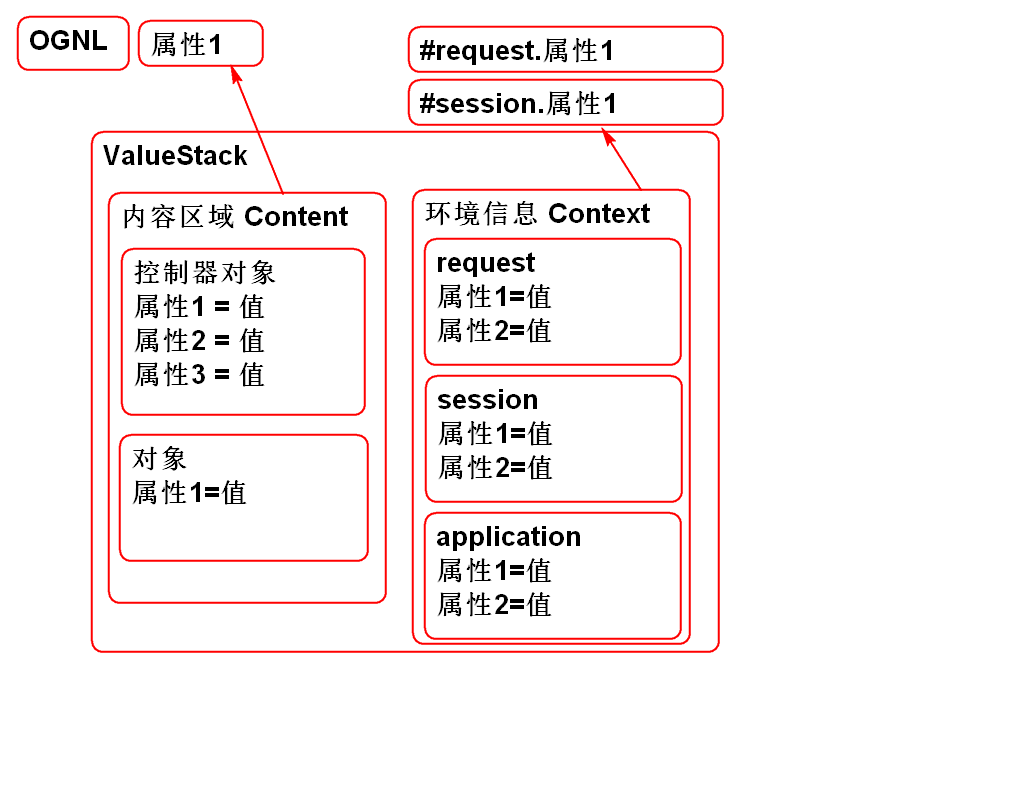
1. 配置struts.xml

<package name="product" namespace="/product" extends="struts-default"> <action name="list" class="productAction" method="list"> <result> /WEB-INF/jsp/list.jsp </result> </action> </package>

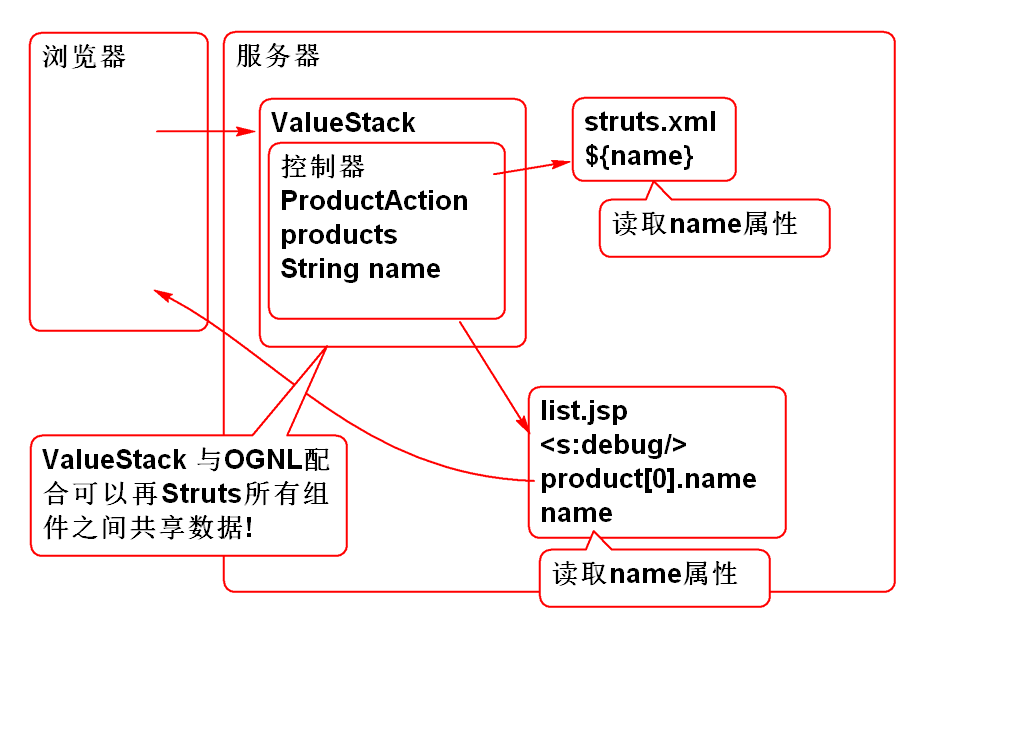
1. 测试:

http://localhost:8080/ssh3/product/list.action

**ValueStack**



Struts 中共享数据的一个机制, 利用ValueStack可以在Struts组件之间共享数据:



Struts组件包括:

* 构造器 Action
* 结果 Result
* 视图 JSP
* 配置文件 struts.xml

JSP 中使用 s:debug 标签可以显示 ValueStack 对象

案例:

1. 创建vs.jsp

<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%> <%@ taglib prefix="s" uri="/struts-tags"%> <!DOCTYPE html> <html> <head> <meta charset="utf-8"> <title>产品管理</title> <style type="text/css"> table { border-collapse: collapse; } td, th{ padding: 5px 10px; } th{ border-bottom: 2px solid #ddd; } td{ border-top: 1px solid #ccc; } </style> </head> <body> <h2>ValueStack</h2> <s:debug></s:debug> <h2>利用OGNL表的式读取ValueStack中的数据</h2> <p>读取ValueStack中的Action对象的数据</p> <s:property value="products[0].name"/> <h3>利用ValueStack 和 OGNL 显示商品列表</h3> <table> <thead> <tr> <th>ID</th> <th>名称</th> <th>价格</th> <th>描述</th> </tr> </thead> <tbody> <s:iterator value="products" var="p"> <tr> <td><s:property value="id"/></td> <td><s:property value="name"/></td> <td><s:property value="price"/></td> <td><s:property value="description"/></td> </tr> </s:iterator> </tbody> </table> </body> </html>

1. 配置struts.xml

<action name="vs" class="productAction" method="list"> <result> /WEB-INF/jsp/vs.jsp </result> </action>

重用控制器 productAction

1. 测试:

http://localhost:8080/ssh3/product/vs.action

说明使用 ValueStack 和 OGNL 也可以显示产品列表.

**使用OGNL表达式可以读取 ValueStack 中的数据**

JSP 中 可以使用 Struts 标签配合 OGNL 表达式读取 ValueStack中的数据, 由于Action对象被放到了 ValueStack中这样就可以, Action 到 JSP共享数据了.

关于 ValueStack 和 OGNL

1. Struts2 中利用ValueStack 在所有组件之间共享数据
   * 其中 控制器也被放到 ValueStack 中
2. 在任何组件中可以使用 ONGL 表达式读取ValueStack中的数据, 包含读取控制器 Action 中的数据
3. 在使用OGNL读取ValueStack数据时候是按照从顶到下顺序查找!
   * 建议不要将数据的名字重复,避免遮挡!
4. Struts2 底层接管了 EL和JSTL, 将EL数据读取转到了ValueStack上! EL的读取结果与 ONGL 一样!
   * Struts 标签和ONGL配合 与 EL和JSTL配合的结果是一样的!
5. 如果需要读取ValueStack的Context区域,就必须使用OGNL
   * 如: <s:property value="#session.loginUser"/>

案例:

1. 编写控制器

/\*\* \* 利用 ValueStack 共享数据 \*/ @Controller @Scope("prototype") public class DataAction extends AbstractAction{ //在内容区域共享数据, 利用控制器的Bean属性 private String name; private int age; //@Value("#{cfg.data}") private String target; //目标网页名 public String getName() { return name; } public void setName(String name) { this.name = name; } public int getAge() { return age; } public void setAge(int age) { this.age = age; } public String getTarget() { return target; } public void setTarget(String target) { this.target = target; } public String execute(){ //上下文区域(环境) Context 区域, 共享数据 request.put("message", "Struts 测试"); session.put("loginState", "已经登录"); application.put("count", 55); name="老王"; age = 123; target = "show.jsp";//目标视图页面 return SUCCESS; } }

1. 编写 show.jsp

<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%> <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> <%@ taglib prefix="s" uri="/struts-tags"%> <!DOCTYPE html> <html> <head> <meta charset="utf-8"> <title>ValueStack</title> </head> <body> <h1>ValueStack</h1> <h2>读取内容区域的值</h2> <s:property value="name"/> <s:property value="age"/> <s:property value="target"/> <h2>读取上下文区域的值</h2> <p>读取上下文区域使用#为开头的表达式</p> <s:property value="#request.message"/> <s:property value="#session.loginState"/> <s:property value="#application.count"/> </body> </html>

1. 配置 struts.xml

<!-- 利用 OGNL 读取ValueStack中的数据 --> <action name="data" class="dataAction"> <result> /WEB-INF/jsp/${target} </result> </action>

表达式${target} 就是利用ValueStack从控制器读取了属性值.

1. 测试

http://localhost:8080/ssh3/product/data.action