# Spring MVC 背景介绍

Spring 框架提供了构建 Web [应用程序](http://baike.baidu.com/view/330120.htm)的全功能 MVC 模块。使用 Spring 可插入的 MVC 架构，可以选择是使用内置的 Spring Web 框架还是 Struts 这样的 Web 框架。通过策略接口，Spring 框架是高度可配置的，而且包含多种[视图](http://baike.baidu.com/view/71981.htm)技术，例如 JavaServer Pages（JSP）技术、Velocity、Tiles、iText 和 POI。Spring MVC 框架并不知道使用的视图，所以不会强迫您只使用 JSP 技术。Spring MVC 分离了控制器、模型[对象](http://baike.baidu.com/view/2387.htm)、分派器以及处理程序对象的角色，这种分离让它们更容易进行定制。

# 常见MVC框架比较

**运行性能上：**

Jsp+servlet>struts1>spring mvc>struts2+freemarker>>struts2,ognl,值栈。

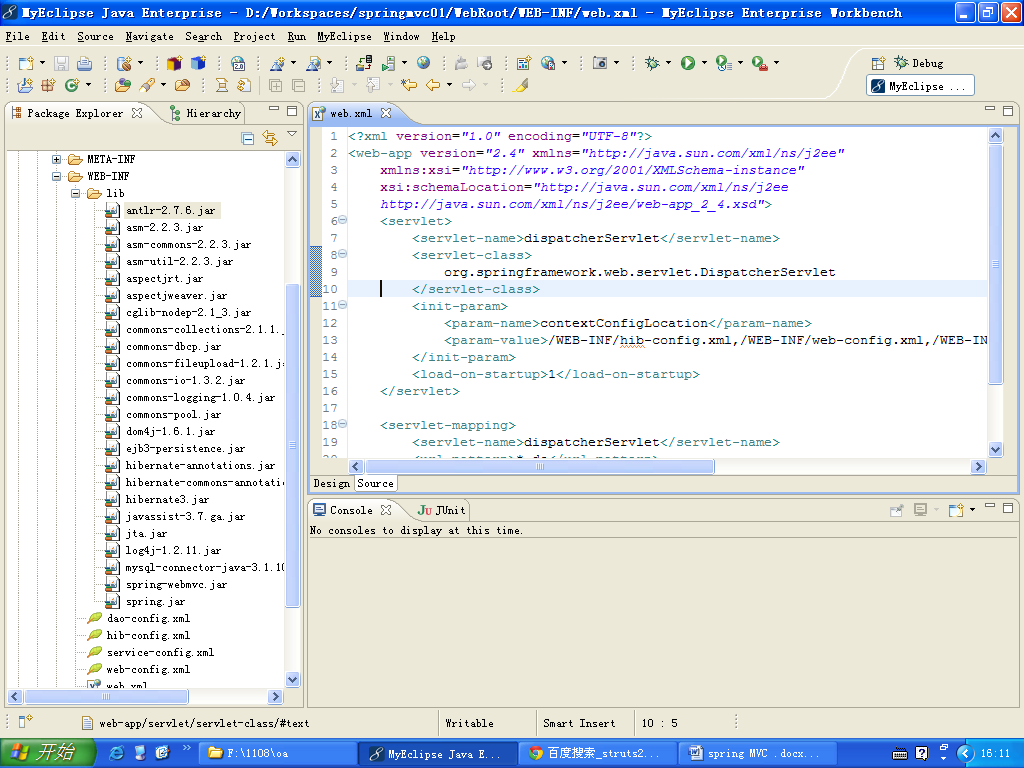
**开发效率上,基本正好相反。值得强调的是，spring mvc开发效率和struts2不相上下。**

Struts2的性能低的原因是因为OGNL和值栈造成的。所以，如果你的系统并发量高，可以使用freemaker进行显示，而不是采用OGNL和值栈。这样，在性能上会有相当大得提高。

# 基于spring2.5的采用XML配置的spring MVC项目

**注：本项目全部基于XML配置。同时，集成了hibernate。采用的是：spring MVC+hibernate+spring的开发架构。**

1. 建立web项目
2. 导入jar包(spring.jar, spring-webmvc.jar, commons-logging.jar。其他jar包为hibernate相关jar包)



1. 修改web.xml如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app version=*"2.5"*  xmlns=*"http://java.sun.com/xml/ns/javaee"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*  *http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*>>  <servlet>  <servlet-name>dispatcherServlet</servlet-name>  <servlet-class>  org.springframework.web.servlet.DispatcherServlet  </servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>/WEB-INF/hib-config.xml,/WEB-INF/web-config.xml,/WEB-INF/service-config.xml,/WEB-INF/dao-config.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>dispatcherServlet</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping>  </web-app> |

1. 增加web-config.xml(这里包含spring mvc相关的相关配置)

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"*>    <!-- Controller方法调用规则定义 -->  <bean id=*"paraMethodResolver"*  class=*"org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver"*>  <property name=*"paramName"* value=*"action"*/>  <property name=*"defaultMethodName"* value=*"list"*/>  </bean>    <!-- 页面View层基本信息设定 -->  <bean id=*"viewResolver"*  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"viewClass"*  value=*"org.springframework.web.servlet.view.JstlView"*/>  <!--<property name="prefix" value="/myjsp/"/>-->  <property name=*"suffix"* value=*".jsp"*/>  </bean>  <!-- servlet映射列表,所有控制层Controller的servlet在这里定义 -->  <bean id=*"urlMapping"*  class=*"org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"*>  <property name=*"mappings"*>  <props>  <prop key=*"user.do"*>userController</prop>  </props>  </property>  </bean>  <bean id=*"userController"* class=*"com.sxt.action.UserController"*>  <property name=*"userService"* ref=*"userService"*></property>  </bean>  </beans> |

1. 在WEB-INF下增加service-config.xml(这里包含service层类的相关配置)

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"*>  <bean id=*"userService"* class=*"com.sxt.service.UserService"*>  <property name=*"userDao"* ref=*"userDao"*></property>  </bean>    </beans> |

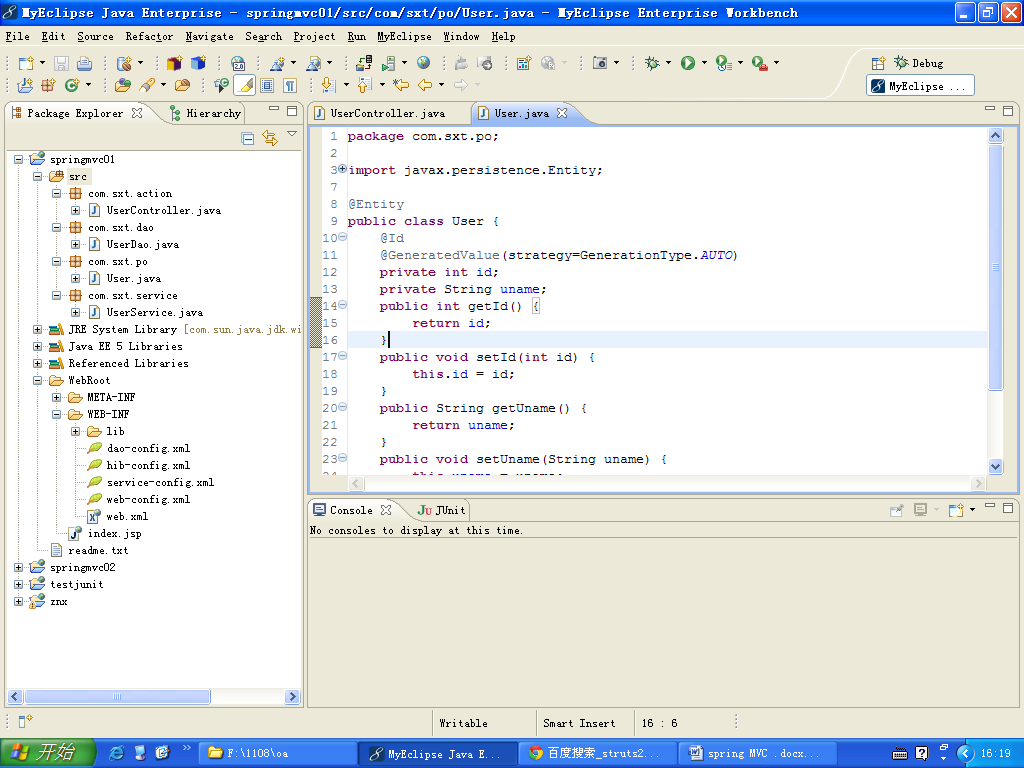
1. 在WEB-INF下增加hib-config.xml(这里包含spring集成hibernate相关的配置)

|  |
| --- |
| <?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:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-2.5.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-2.5.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-2.5.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-2.5.xsd*  *"*>  <context:component-scan base-package=*"com.sxt"*/>  <!-- 支持aop注解 -->  <aop:aspectj-autoproxy />      <bean id=*"dataSource"*  class=*"org.apache.commons.dbcp.BasicDataSource"*>  <property name=*"driverClassName"*  value=*"com.mysql.jdbc.Driver"*>  </property>  <property name=*"url"* value=*"jdbc:mysql://localhost:3306/myhib"*></property>  <property name=*"username"* value=*"root"*></property>  <property name=*"password"* value=*"123456"*></property>  </bean>  <bean id=*"sessionFactory"*  class=*"org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean"*>  <property name=*"dataSource"*>  <ref bean=*"dataSource"* />  </property>  <property name=*"hibernateProperties"*>  <props>  <!-- key的名字前面都要加hibernate. -->  <prop key=*"hibernate.dialect"*>  org.hibernate.dialect.MySQLDialect  </prop>  <prop key=*"hibernate.show\_sql"*>true</prop>  <prop key=*"hibernate.hbm2ddl.auto"*>update</prop>  </props>  </property>  <property name=*"packagesToScan"*>  <value>com.sxt.po</value>  </property>  </bean>  <bean id=*"hibernateTemplate"* class=*"org.springframework.orm.hibernate3.HibernateTemplate"* >  <property name=*"sessionFactory"* ref=*"sessionFactory"*></property>  </bean>  <!--配置一个JdbcTemplate实例-->  <bean id=*"jdbcTemplate"* class=*"org.springframework.jdbc.core.JdbcTemplate"*>  <property name=*"dataSource"* ref=*"dataSource"*/>  </bean>  <!-- 配置事务管理 -->  <bean id=*"txManager"* class=*"org.springframework.orm.hibernate3.HibernateTransactionManager"* >  <property name=*"sessionFactory"* ref=*"sessionFactory"*></property>  </bean>  <tx:annotation-driven transaction-manager=*"txManager"* />  <aop:config>  <aop:pointcut expression=*"execution(public \* com.sxt.service.impl.\*.\*(..))"* id=*"businessService"*/>  <aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"businessService"* />  </aop:config>  <tx:advice id=*"txAdvice"* transaction-manager=*"txManager"* >  <tx:attributes>  <tx:method name=*"find\*"* read-only=*"true"* propagation=*"NOT\_SUPPORTED"* />  <!-- get开头的方法不需要在事务中运行 。  有些情况是没有必要使用事务的，比如获取数据。开启事务本身对性能是有一定的影响的-->  <tx:method name=*"\*"*/> <!-- 其他方法在实务中运行 -->  </tx:attributes>  </tx:advice>  </beans> |

1. 在WEB-INF下增加dao-config.xml(这里包含dao层类的相关配置)

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"*>    <bean id=*"userDao"* class=*"com.sxt.dao.UserDao"*>  <property name=*"hibernateTemplate"* ref=*"hibernateTemplate"*></property>  </bean>  </beans> |

1. 建立相关类和包结构，如下图所示：



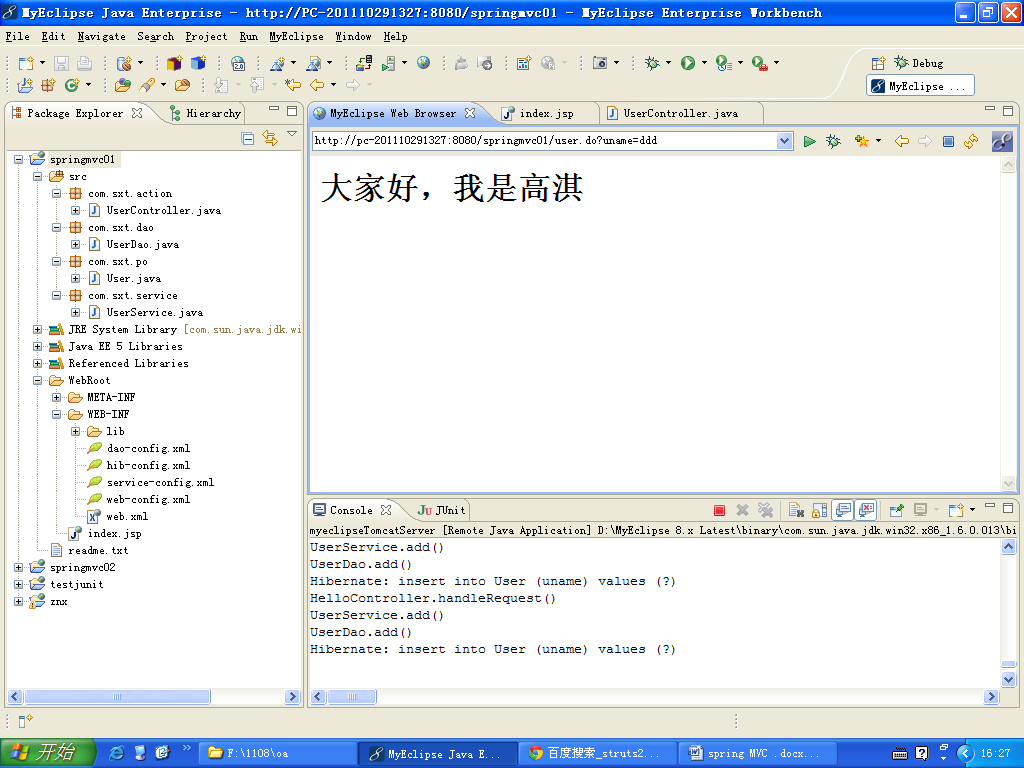
1. 各类代码如下：

|  |
| --- |
| **package** com.sxt.po;  **import** javax.persistence.Entity;  **import** javax.persistence.GeneratedValue;  **import** javax.persistence.GenerationType;  **import** javax.persistence.Id;  @Entity  **public** **class** User {  @Id  @GeneratedValue(strategy=GenerationType.*AUTO*)  **private** **int** id;  **private** String uname;  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** String getUname() {  **return** uname;  }  **public** **void** setUname(String uname) {  **this**.uname = uname;  }  } |
| package com.sxt.dao;  import org.springframework.orm.hibernate3.HibernateTemplate;  import com.sxt.po.User;  public class UserDao {  private HibernateTemplate hibernateTemplate;    public void add(User u){  System.out.println("UserDao.add()");  hibernateTemplate.save(u);  }  public HibernateTemplate getHibernateTemplate() {  return hibernateTemplate;  }  public void setHibernateTemplate(HibernateTemplate hibernateTemplate) {  this.hibernateTemplate = hibernateTemplate;  }    } |
| package com.sxt.service;  import com.sxt.dao.UserDao;  import com.sxt.po.User;  public class UserService {    private UserDao userDao;    public void add(String uname){  System.out.println("UserService.add()");  User u = new User();  u.setUname(uname);  userDao.add(u);  }  public UserDao getUserDao() {  return userDao;  }  public void setUserDao(UserDao userDao) {  this.userDao = userDao;  }    } |
| package com.sxt.action;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import org.springframework.web.servlet.ModelAndView;  import org.springframework.web.servlet.mvc.Controller;  import com.sxt.service.UserService;  public class UserController implements Controller {  private UserService userService;    @Override  public ModelAndView handleRequest(HttpServletRequest req,  HttpServletResponse resp) throws Exception {  System.out.println("HelloController.handleRequest()");  req.setAttribute("a", "aaaa");  userService.add(req.getParameter("uname"));  return new ModelAndView("index");  }  public UserService getUserService() {  return userService;  }  public void setUserService(UserService userService) {  this.userService = userService;  }    } |

1. 运行测试：

<http://locahost:8080/springmvc01/user.do?uname=zhangsan>。

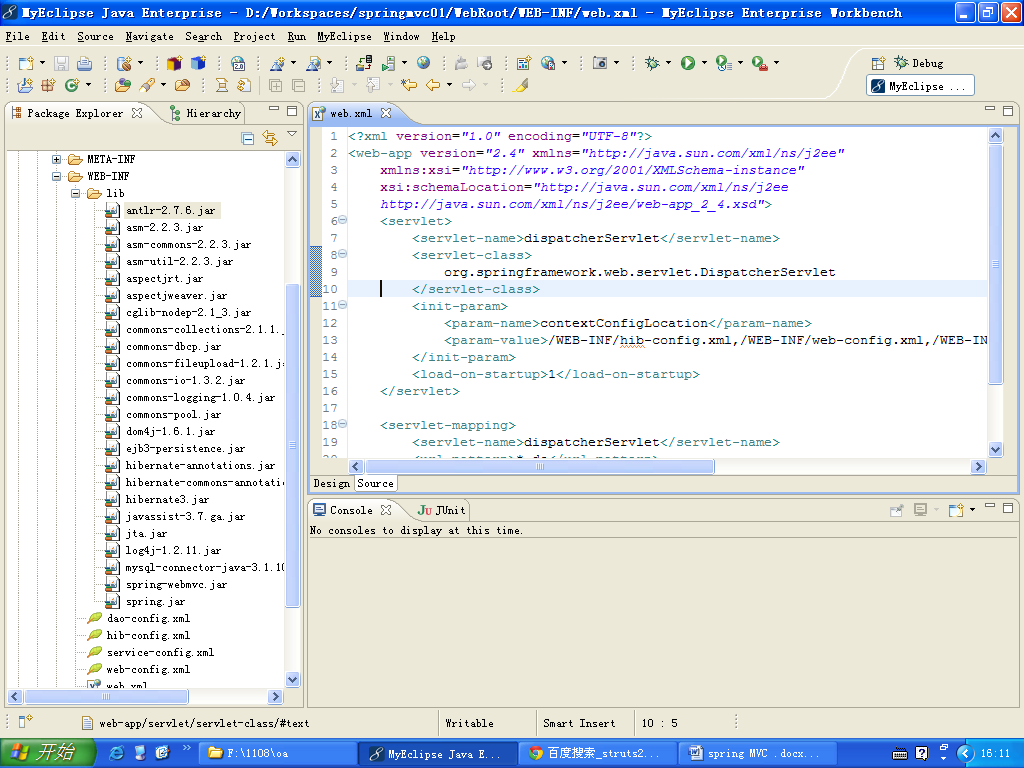
结果：数据库中增加zhangsan的记录。页面跳转到index.jsp上，显示：



# 基于spring2.5注解实现的spring MVC项目

我们采用sprng MVC开发项目时，通常都会采用注解的方式，这样可以大大提高我们的开发效率。实现零配置。下面我们从零开始重新做一个spring MVC的配置。这个项目完全采用注解的方式开发。同时，我们以后的spring MVC项目也都会采用注解的方式。

1. 建立web项目
2. 导入jar包(spring.jar, spring-webmvc.jar, commons-logging.jar。其他jar包为hibernate相关jar包)



1. 修改web.xml，文件内容如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app version=*"2.5"*  xmlns=*"http://java.sun.com/xml/ns/javaee"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*  *http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*>  <servlet>  <servlet-name>springmvc</servlet-name>  <servlet-class>  org.springframework.web.servlet.DispatcherServlet  </servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>/WEB-INF/hib-config.xml,/WEB-INF/springmvc-servlet.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>springmvc</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping>  </web-app> |

1. springmvc-servlet.xml配置内容如下：

|  |
| --- |
| <?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:p=*"http://www.springframework.org/schema/p"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-2.5.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-2.5.xsd"*>    <!-- 对web包中的所有类进行扫描，以完成Bean创建和自动依赖注入的功能 -->  <context:component-scan base-package=*"com.sxt"*/>  <!-- 启动Spring MVC的注解功能，完成请求和注解POJO的映射 -->  <bean class=*"org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter"*/>  <!--对模型视图名称的解析，即在模型视图名称添加前后缀 -->  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*  p:suffix=*".jsp"*/>  </beans> |

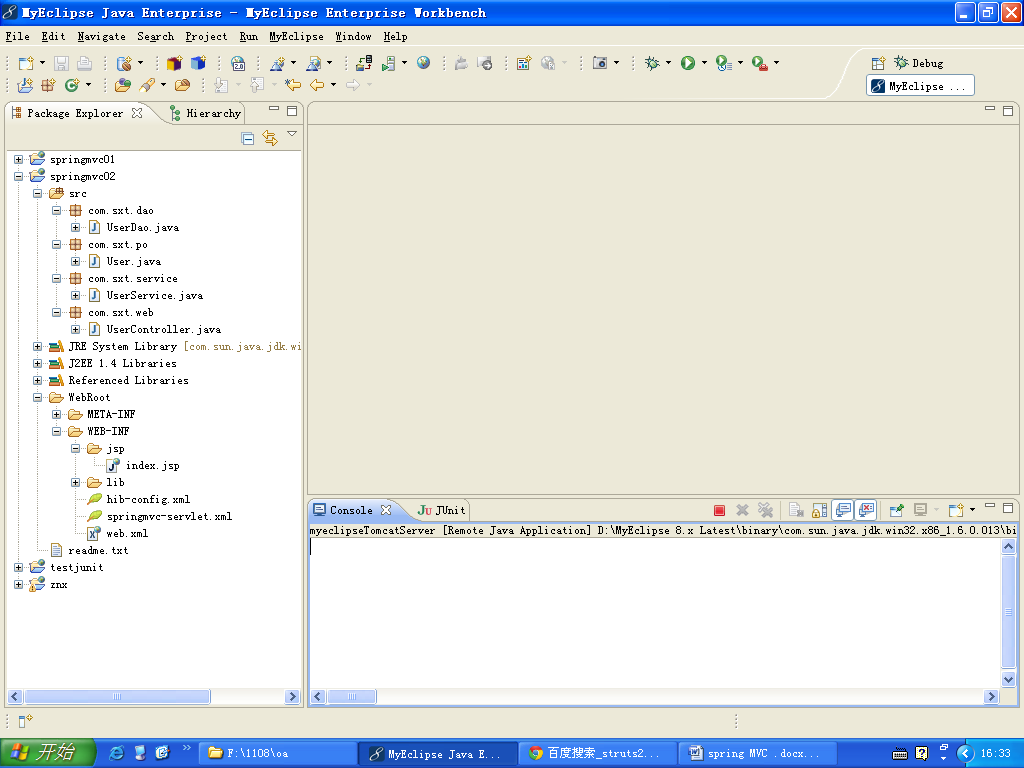
1. hib-config.xml(配置了spring集成hibernate)

|  |
| --- |
| <?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:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-2.5.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-2.5.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-2.5.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-2.5.xsd*  *"*>  <context:component-scan base-package=*"com.sxt"*/>  <!-- 支持aop注解 -->  <aop:aspectj-autoproxy />      <bean id=*"dataSource"*  class=*"org.apache.commons.dbcp.BasicDataSource"*>  <property name=*"driverClassName"*  value=*"com.mysql.jdbc.Driver"*>  </property>  <property name=*"url"* value=*"jdbc:mysql://localhost:3306/myhib"*></property>  <property name=*"username"* value=*"root"*></property>  <property name=*"password"* value=*"123456"*></property>  </bean>  <bean id=*"sessionFactory"*  class=*"org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean"*>  <property name=*"dataSource"*>  <ref bean=*"dataSource"* />  </property>  <property name=*"hibernateProperties"*>  <props>  <!-- key的名字前面都要加hibernate. -->  <prop key=*"hibernate.dialect"*>  org.hibernate.dialect.MySQLDialect  </prop>  <prop key=*"hibernate.show\_sql"*>true</prop>  <prop key=*"hibernate.hbm2ddl.auto"*>update</prop>  </props>  </property>  <property name=*"packagesToScan"*>  <value>com.sxt.po</value>  </property>  </bean>  <bean id=*"hibernateTemplate"* class=*"org.springframework.orm.hibernate3.HibernateTemplate"* >  <property name=*"sessionFactory"* ref=*"sessionFactory"*></property>  </bean>  <!--配置一个JdbcTemplate实例-->  <bean id=*"jdbcTemplate"* class=*"org.springframework.jdbc.core.JdbcTemplate"*>  <property name=*"dataSource"* ref=*"dataSource"*/>  </bean>  <!-- 配置事务管理 -->  <bean id=*"txManager"* class=*"org.springframework.orm.hibernate3.HibernateTransactionManager"* >  <property name=*"sessionFactory"* ref=*"sessionFactory"*></property>  </bean>  <tx:annotation-driven transaction-manager=*"txManager"* />  <aop:config>  <aop:pointcut expression=*"execution(public \* com.sxt.service.impl.\*.\*(..))"* id=*"businessService"*/>  <aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"businessService"* />  </aop:config>  <tx:advice id=*"txAdvice"* transaction-manager=*"txManager"* >  <tx:attributes>  <tx:method name=*"find\*"* read-only=*"true"* propagation=*"NOT\_SUPPORTED"* />  <!-- get开头的方法不需要在事务中运行 。  有些情况是没有必要使用事务的，比如获取数据。开启事务本身对性能是有一定的影响的-->  <tx:method name=*"\*"*/> <!-- 其他方法在实务中运行 -->  </tx:attributes>  </tx:advice>  </beans> |

1. WEB-INF下建立jsp文件夹，并且将index.jsp放入该文件夹下。Index.jsp的内容如下：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"gbk"*%>  <%  String path = request.getContextPath();  String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";  %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <base href=*"*<%=basePath%>*"*>    <title>My JSP 'index.jsp' starting page</title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  <meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>  <meta http-equiv=*"description"* content=*"This is my page"*>  <!--  <link rel="stylesheet" type="text/css" href="styles.css">  -->  </head>    <body>  <h1>\*\*\*\*\*\*\*\*\*\*${params.uname}</h1>  <h1>\*\*\*\*\*\*\*\*\*\*${requestScope.u}</h1>  <h1>\*\*\*\*\*\*\*\*\*\*${requestScope.user}</h1>  </body>  </html> |

1. 建立整个项目的包结构和相关类。如下图所示：



1. User、UserDao、UserService、UserController类的代码如下：

|  |
| --- |
| package com.sxt.po;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.GenerationType;  import javax.persistence.Id;  @Entity  public class User {  @Id  @GeneratedValue(strategy=GenerationType.AUTO)  private int id;  private String uname;  private String pwd;      public String getPwd() {  return pwd;  }  public void setPwd(String pwd) {  this.pwd = pwd;  }  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getUname() {  return uname;  }  public void setUname(String uname) {  this.uname = uname;  }      } |
| package com.sxt.dao;  import javax.annotation.Resource;  import org.springframework.orm.hibernate3.HibernateTemplate;  import org.springframework.stereotype.Repository;  import com.sxt.po.User;  @Repository("userDao")  public class UserDao {  @Resource  private HibernateTemplate hibernateTemplate;    public void add(User u){  System.out.println("UserDao.add()");  hibernateTemplate.save(u);  }  public HibernateTemplate getHibernateTemplate() {  return hibernateTemplate;  }  public void setHibernateTemplate(HibernateTemplate hibernateTemplate) {  this.hibernateTemplate = hibernateTemplate;  }    } |
| package com.sxt.service;  import javax.annotation.Resource;  import org.springframework.stereotype.Service;  import com.sxt.dao.UserDao;  import com.sxt.po.User;  @Service("userService")  public class UserService {  @Resource  private UserDao userDao;    public void add(String uname){  System.out.println("UserService.add()");  User u = new User();  u.setUname(uname);  userDao.add(u);  }  public UserDao getUserDao() {  return userDao;  }  public void setUserDao(UserDao userDao) {  this.userDao = userDao;  }    } |
| package com.sxt.web;  import javax.annotation.Resource;  import org.springframework.stereotype.Controller;  import org.springframework.ui.ModelMap;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.bind.annotation.SessionAttributes;  import com.sxt.po.User;  import com.sxt.service.UserService;  @Controller("userController")  @RequestMapping("/user.do")  public class UserController {  @Resource  private UserService userService;    @RequestMapping(params="method=reg")  public String reg(String uname) {  System.out.println("HelloController.handleRequest()");  userService.add(uname);  return "index";  }    public UserService getUserService() {  return userService;  }  public void setUserService(UserService userService) {  this.userService = userService;  }    } |

1. 运行测试：

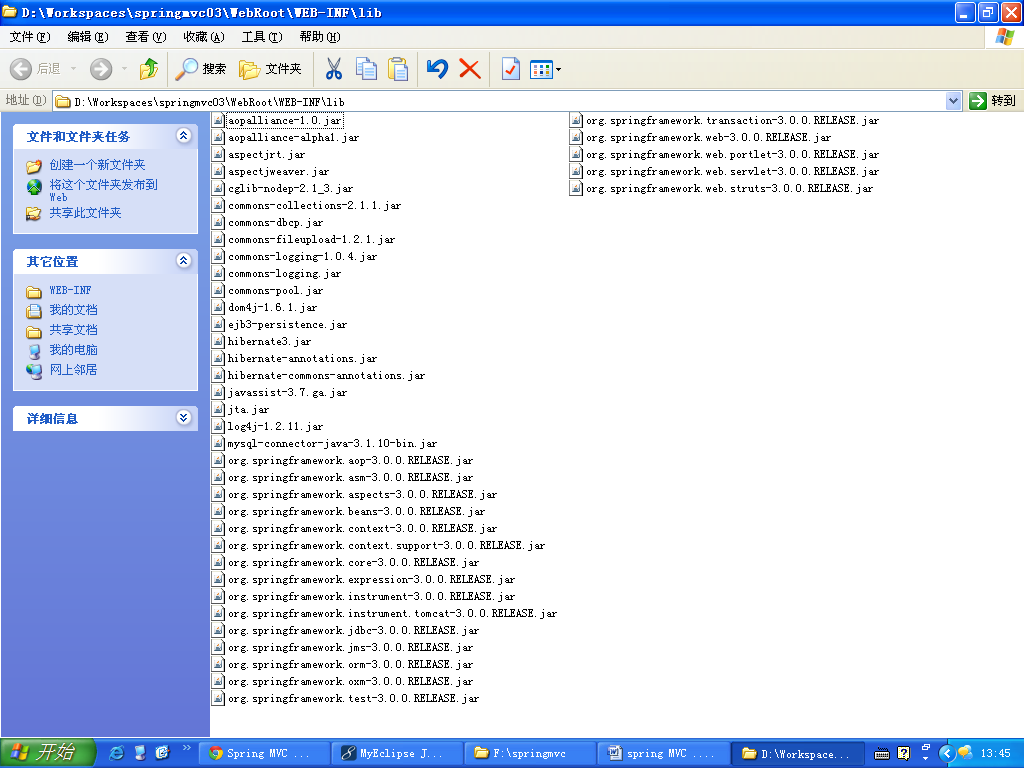
http://pc-201110291327:8080/springmvc02/user.do?method=reg&uname=gaoqi

则会调用userController的reg方法，从而将数据内容插入到数据库中。

# 基于spring 3.0项目开发实例

spring3.0完全兼容spring2.5.因此，我们只要简单修改上面项目的类库和配置文件。类的代码保持不变。

1. 导入相关jar包，如下：



1. spring配置文件springmvc-servlet.xml修改如下：

|  |
| --- |
| <?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:p=*"http://www.springframework.org/schema/p"*  xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:util=*"http://www.springframework.org/schema/util"*  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/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"*>    <!-- 对web包中的所有类进行扫描，以完成Bean创建和自动依赖注入的功能 -->  <context:component-scan base-package=*"com.sxt.web"*/>    <mvc:annotation-driven /> <!-- 支持spring3.0新的mvc注解 -->  <!-- 启动Spring MVC的注解功能，完成请求和注解POJO的映射 -->  <bean class=*"org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter"*/>  <!--对模型视图名称的解析，即在模型视图名称添加前后缀 -->  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*  p:prefix=*"/WEB-INF/jsp/"* p:suffix=*".jsp"*>  <!-- 如果使用jstl的话，配置下面的属性 -->  <property name=*"viewClass"* value=*"org.springframework.web.servlet.view.JstlView"* />  </bean>  </beans> |

1. spring配置文件hib-config.xml内容修改如下：

|  |
| --- |
| <?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:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-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/aop*  *http://www.springframework.org/schema/aop/spring-aop-3.0.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-3.0.xsd*  *"*>  <context:component-scan base-package=*"com.sxt"*/>  <!-- 支持aop注解 -->  <aop:aspectj-autoproxy />      <bean id=*"dataSource"*  class=*"org.apache.commons.dbcp.BasicDataSource"*>  <property name=*"driverClassName"*  value=*"com.mysql.jdbc.Driver"*>  </property>  <property name=*"url"* value=*"jdbc:mysql://localhost:3306/myhib"*></property>  <property name=*"username"* value=*"root"*></property>  <property name=*"password"* value=*"123456"*></property>  </bean>  <bean id=*"sessionFactory"*  class=*"org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean"*>  <property name=*"dataSource"*>  <ref bean=*"dataSource"* />  </property>  <property name=*"hibernateProperties"*>  <props>  <!-- key的名字前面都要加hibernate. -->  <prop key=*"hibernate.dialect"*>  org.hibernate.dialect.MySQLDialect  </prop>  <prop key=*"hibernate.show\_sql"*>true</prop>  <prop key=*"hibernate.hbm2ddl.auto"*>update</prop>  </props>  </property>  <property name=*"packagesToScan"*>  <value>com.sxt.po</value>  </property>  </bean>  <bean id=*"hibernateTemplate"* class=*"org.springframework.orm.hibernate3.HibernateTemplate"* >  <property name=*"sessionFactory"* ref=*"sessionFactory"*></property>  </bean>  <!--配置一个JdbcTemplate实例-->  <bean id=*"jdbcTemplate"* class=*"org.springframework.jdbc.core.JdbcTemplate"*>  <property name=*"dataSource"* ref=*"dataSource"*/>  </bean>  <!-- 配置事务管理 -->  <bean id=*"txManager"* class=*"org.springframework.orm.hibernate3.HibernateTransactionManager"* >  <property name=*"sessionFactory"* ref=*"sessionFactory"*></property>  </bean>  <tx:annotation-driven transaction-manager=*"txManager"* />  <aop:config>  <aop:pointcut expression=*"execution(public \* com.sxt.service.impl.\*.\*(..))"* id=*"businessService"*/>  <aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"businessService"* />  </aop:config>  <tx:advice id=*"txAdvice"* transaction-manager=*"txManager"* >  <tx:attributes>  <tx:method name=*"find\*"* read-only=*"true"* propagation=*"NOT\_SUPPORTED"* />  <!-- get开头的方法不需要在事务中运行 。  有些情况是没有必要使用事务的，比如获取数据。开启事务本身对性能是有一定的影响的-->  <tx:method name=*"\*"*/> <!-- 其他方法在实务中运行 -->  </tx:attributes>  </tx:advice>  </beans> |

1. web.xml文件不变
2. 类的代码不变。
3. 运行，测试。跟上一个项目保持一致。

# Spring MVC 3.0 深入

## 核心原理

1. 用户发送请求给服务器。url：user.do
2. 服务器收到请求。发现DispatchServlet可以处理。于是调用DispatchServlet。
3. DispatchServlet内部，通过HandleMapping检查这个url有没有对应的Controller。如果有，则调用Controller。
4. Controller开始执行。
5. Controller执行完毕后，如果返回字符串，则ViewResolver将字符串转化成相应的视图对象；如果返回ModelAndView对象，该对象本身就包含了视图对象信息。
6. DispatchServlet将执视图对象中的数据，输出给服务器。
7. 服务器将数据输出给客户端。

## spring3.0中相关jar包的含义

|  |  |
| --- | --- |
| org.springframework.aop-3.0.3.RELEASE.jar | **spring的aop面向切面编程** |
| org.springframework.asm-3.0.3.RELEASE.jar | **spring独立的asm字节码生成程序** |
| org.springframework.beans-3.0.3.RELEASE.jar | **IOC的基础实现** |
| org.springframework.context-3.0.3.RELEASE.jar | **IOC基础上的扩展服务** |
| org.springframework.core-3.0.3.RELEASE.jar | **spring的核心包** |
| org.springframework.expression-3.0.3.RELEASE.jar | **spring的表达式语言** |
| org.springframework.web-3.0.3.RELEASE.jar | **web工具包** |
| org.springframework.web.servlet-3.0.3.RELEASE.jar | **mvc工具包** |

## @Controller控制器定义

和Struts1一样，Spring的Controller是Singleton的。这就意味着会被多个请求线程共享。因此，我们将控制器设计成无状态类。

在spring 3.0中，通过@controller标注即可将class定义为一个controller类。为使spring能找到定义为controller的bean,需要在spring-context配置文件中增加如下定义：

|  |
| --- |
| <context:component-scan base-package=*"com.sxt.web"*/> |

注：实际上，使用@component，也可以起到@Controller同样的作用。

## @RequestMapping

在类前面定义，则将url和类绑定。

在方法前面定义，则将url和类的方法绑定，如下所示：

|  |
| --- |
| **package** com.sxt.web;  **import** javax.annotation.Resource;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** com.sxt.service.UserService;  @Controller  @RequestMapping("/user.do")  **public** **class** UserController {  @Resource  **private** UserService userService;    //http://localhost:8080/springmvc02/user.do?method=reg&uname=zzzz  @RequestMapping(params="method=reg")  **public** String reg(String uname) {  System.*out*.println("HelloController.handleRequest()");  userService.add(uname);  **return** "index";  }    **public** UserService getUserService() {  **return** userService;  }  **public** **void** setUserService(UserService userService) {  **this**.userService = userService;  }    } |

## @RequestParam

一般用于将指定的请求参数付给方法中形参。示例代码如下：

|  |
| --- |
| @RequestMapping(params="method=reg5")  **public** String reg5(**@RequestParam("name")String uname**,ModelMap map) {  System.*out*.println("HelloController.handleRequest()");  System.*out*.println(uname);  **return** "index";  } |

这样，就会将name参数的值付给uname。当然，如果请求参数名称和形参名称保持一致，则不需要这种写法。

## @SessionAttributes

将ModelMap中指定的属性放到session中。示例代码如下：

|  |
| --- |
| @Controller  @RequestMapping("/user.do")  @SessionAttributes({"u","a"}) **//将ModelMap中属性名字为u、a的再放入session中。这样，request和session中都有了。**  **public** **class** UserController {  @RequestMapping(params="method=reg4")  **public** String reg4(ModelMap map) { System.*out*.println("HelloController.handleRequest()");  map.addAttribute("u","uuuu"); **//将u放入request作用域中，这样转发页面也可以取到这个数据。**  **return** "index";  }  } |
| <body>  <h1>\*\*\*\*\*\*\*\*\*\*${requestScope.u.uname}</h1>  <h1>\*\*\*\*\*\*\*\*\*\*${sessionScope.u.uname}</h1>  </body> |

注：名字为”user”的属性再结合使用注解@SessionAttributes可能会报错。

## @ModelAttribute

[这个注解可以跟@SessionAttributes配合在一起用。可以将ModelMap](mailto:这个注解可以跟@SessionAttributes配合在一起用。可以将ModelMap)中属性的值通过该注解自动赋给指定变量。

示例代码如下：

|  |
| --- |
| **package** com.sxt.web;  **import** javax.annotation.Resource;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.ui.ModelMap;  **import** org.springframework.web.bind.annotation.ModelAttribute;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.SessionAttributes;  @Controller  @RequestMapping("/user.do")  @SessionAttributes({"u","a"})  **public** **class** UserController {    @RequestMapping(params="method=reg4")  **public** String reg4(ModelMap map) {  System.*out*.println("HelloController.handleRequest()");  map.addAttribute("u","尚学堂高淇");  **return** "index";  }    @RequestMapping(params="method=reg5")  **public** String reg5(@ModelAttribute("u")String uname,ModelMap map) {  System.*out*.println("HelloController.handleRequest()");  System.*out*.println(uname);  **return** "index";  }    } |

先调用reg4方法，再调用reg5方法。我们发现控制台打印出来：尚学堂高淇

## Controller类中方法参数的处理

## Controller类中方法返回值的处理

1. 返回string(建议)
   1. 根据返回值找对应的显示页面。路径规则为：prefix前缀+返回值+suffix后缀组成
   2. 代码如下：

|  |
| --- |
| @RequestMapping(params="method=reg4")  **public** String reg4(ModelMap map) {  System.*out*.println("HelloController.handleRequest()");  **return** "index";  } |
| 前缀为：/WEB-INF/jsp/ 后缀是：.jsp  在转发到：/WEB-INF/jsp/index.jsp |

1. 也可以返回ModelMap、ModelAndView、map、List、Set、Object、无返回值。 **一般建议返回字符串！**

## 请求转发和重定向

代码示例：

|  |
| --- |
| **package** com.sxt.web;  **import** javax.annotation.Resource;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.ui.ModelMap;  **import** org.springframework.web.bind.annotation.ModelAttribute;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.SessionAttributes;  @Controller  @RequestMapping("/user.do")  **public** **class** UserController {    @RequestMapping(params="method=reg4")  **public** String reg4(ModelMap map) {  System.*out*.println("HelloController.handleRequest()");  // return "forward:index.jsp";  // return "forward:user.do?method=reg5"; //转发  // return "redirect:user.do?method=reg5"; //重定向  **return** "redirect:http://www.baidu.com"; //重定向  }    @RequestMapping(params="method=reg5")  **public** String reg5(String uname,ModelMap map) {  System.*out*.println("HelloController.handleRequest()");  System.*out*.println(uname);  **return** "index";  }    } |

访问reg4方法，既可以看到效果。

## 获得request对象、session对象

普通的Controller类，示例代码如下：

|  |
| --- |
| @Controller  @RequestMapping("/user.do")  **public** **class** UserController {    @RequestMapping(params="method=reg2")  **public** String reg2(String uname,**HttpServletRequest req**,ModelMap map){  req.setAttribute("a", "aa");  req.getSession().setAttribute("b", "bb");  **return** "index";  }  } |

## ModelMap

是map的实现，可以在其中存放属性，作用域同request。下面这个示例，我们可以在modelMap中放入数据，然后在forward的页面上显示这些数据。通过el表达式、JSTL、java代码均可。代码如下：

|  |
| --- |
| **package** com.sxt.web;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.ui.ModelMap;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.mvc.multiaction.MultiActionController;  @Controller  @RequestMapping("/user.do")  **public** **class** UserController **extends** MultiActionController {    @RequestMapping(params="method=reg")  **public** String reg(String uname,ModelMap map){  map.put("a", "aaa");  **return** "index";  }  } |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"gbk"*%>  <%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head></head>  <body>  <h1>${requestScope.a}</h1>  <c:out value="${requestScope.a}"></c:out>  </body>  </html> |

## ModelAndView模型视图类

见名知意，从名字上我们可以知道ModelAndView中的Model代表模型，View代表视图。即，这个类把要显示的数据存储到了Model属性中，要跳转的视图信息存储到了view属性。我们看一下ModelAndView的部分源码，即可知其中关系：

|  |
| --- |
| **public** **class** ModelAndView {  /\*\* View instance or view name String \*/  **private** Object view;  /\*\* Model Map \*/  **private** ModelMap model;  /\*\*  \* Indicates whether or not this instance has been cleared with a call to {@link #clear()}.  \*/  **private** **boolean** cleared = **false**;  /\*\*  \* Default constructor for bean-style usage: populating bean  \* properties instead of passing in constructor arguments.  \* **@see** #setView(View)  \* **@see** #setViewName(String)  \*/  **public** ModelAndView() {  }  /\*\*  \* Convenient constructor when there is no model data to expose.  \* Can also be used in conjunction with <code>addObject</code>.  \* **@param** viewName name of the View to render, to be resolved  \* by the DispatcherServlet's ViewResolver  \* **@see** #addObject  \*/  **public** ModelAndView(String viewName) {  **this**.view = viewName;  }  /\*\*  \* Convenient constructor when there is no model data to expose.  \* Can also be used in conjunction with <code>addObject</code>.  \* **@param** view View object to render  \* **@see** #addObject  \*/  **public** ModelAndView(View view) {  **this**.view = view;  }  /\*\*  \* Creates new ModelAndView given a view name and a model.  \* **@param** viewName name of the View to render, to be resolved  \* by the DispatcherServlet's ViewResolver  \* **@param** model Map of model names (Strings) to model objects  \* (Objects). Model entries may not be <code>null</code>, but the  \* model Map may be <code>null</code> if there is no model data.  \*/  **public** ModelAndView(String viewName, Map<String, ?> model) {  **this**.view = viewName;  **if** (model != **null**) {  getModelMap().addAllAttributes(model);  }  }  /\*\*  \* Creates new ModelAndView given a View object and a model.  \* <emphasis>Note: the supplied model data is copied into the internal  \* storage of this class. You should not consider to modify the supplied  \* Map after supplying it to this class</emphasis>  \* **@param** view View object to render  \* **@param** model Map of model names (Strings) to model objects  \* (Objects). Model entries may not be <code>null</code>, but the  \* model Map may be <code>null</code> if there is no model data.  \*/  **public** ModelAndView(View view, Map<String, ?> model) {  **this**.view = view;  **if** (model != **null**) {  getModelMap().addAllAttributes(model);  }  }  /\*\*  \* Convenient constructor to take a single model object.  \* **@param** viewName name of the View to render, to be resolved  \* by the DispatcherServlet's ViewResolver  \* **@param** modelName name of the single entry in the model  \* **@param** modelObject the single model object  \*/  **public** ModelAndView(String viewName, String modelName, Object modelObject) {  **this**.view = viewName;  addObject(modelName, modelObject);  }  /\*\*  \* Convenient constructor to take a single model object.  \* **@param** view View object to render  \* **@param** modelName name of the single entry in the model  \* **@param** modelObject the single model object  \*/  **public** ModelAndView(View view, String modelName, Object modelObject) {  **this**.view = view;  addObject(modelName, modelObject);  }  /\*\*  \* Set a view name for this ModelAndView, to be resolved by the  \* DispatcherServlet via a ViewResolver. Will override any  \* pre-existing view name or View.  \*/  **public** **void** setViewName(String viewName) {  **this**.view = viewName;  }  /\*\*  \* Return the view name to be resolved by the DispatcherServlet  \* via a ViewResolver, or <code>null</code> if we are using a View object.  \*/  **public** String getViewName() {  **return** (**this**.view **instanceof** String ? (String) **this**.view : **null**);  }  /\*\*  \* Set a View object for this ModelAndView. Will override any  \* pre-existing view name or View.  \*/  **public** **void** setView(View view) {  **this**.view = view;  }  /\*\*  \* Return the View object, or <code>null</code> if we are using a view name  \* to be resolved by the DispatcherServlet via a ViewResolver.  \*/  **public** View getView() {  **return** (**this**.view **instanceof** View ? (View) **this**.view : **null**);  }  /\*\*  \* Indicate whether or not this <code>ModelAndView</code> has a view, either  \* as a view name or as a direct {@link View} instance.  \*/  **public** **boolean** hasView() {  **return** (**this**.view != **null**);  }  /\*\*  \* Return whether we use a view reference, i.e. <code>true</code>  \* if the view has been specified via a name to be resolved by the  \* DispatcherServlet via a ViewResolver.  \*/  **public** **boolean** isReference() {  **return** (**this**.view **instanceof** String);  }  /\*\*  \* Return the model map. May return <code>null</code>.  \* Called by DispatcherServlet for evaluation of the model.  \*/  **protected** Map<String, Object> getModelInternal() {  **return** **this**.model;  }  /\*\*  \* Return the underlying <code>ModelMap</code> instance (never <code>null</code>).  \*/  **public** ModelMap getModelMap() {  **if** (**this**.model == **null**) {  **this**.model = **new** ModelMap();  }  **return** **this**.model;  }  /\*\*  \* Return the model map. Never returns <code>null</code>.  \* To be called by application code for modifying the model.  \*/  **public** Map<String, Object> getModel() {  **return** getModelMap();  }  /\*\*  \* Add an attribute to the model.  \* **@param** attributeName name of the object to add to the model  \* **@param** attributeValue object to add to the model (never <code>null</code>)  \* **@see** ModelMap#addAttribute(String, Object)  \* **@see** #getModelMap()  \*/  **public** ModelAndView addObject(String attributeName, Object attributeValue) {  getModelMap().addAttribute(attributeName, attributeValue);  **return** **this**;  }  /\*\*  \* Add an attribute to the model using parameter name generation.  \* **@param** attributeValue the object to add to the model (never <code>null</code>)  \* **@see** ModelMap#addAttribute(Object)  \* **@see** #getModelMap()  \*/  **public** ModelAndView addObject(Object attributeValue) {  getModelMap().addAttribute(attributeValue);  **return** **this**;  }  /\*\*  \* Add all attributes contained in the provided Map to the model.  \* **@param** modelMap a Map of attributeName -> attributeValue pairs  \* **@see** ModelMap#addAllAttributes(Map)  \* **@see** #getModelMap()  \*/  **public** ModelAndView addAllObjects(Map<String, ?> modelMap) {  getModelMap().addAllAttributes(modelMap);  **return** **this**;  }  /\*\*  \* Clear the state of this ModelAndView object.  \* The object will be empty afterwards.  \* <p>Can be used to suppress rendering of a given ModelAndView object  \* in the <code>postHandle</code> method of a HandlerInterceptor.  \* **@see** #isEmpty()  \* **@see** HandlerInterceptor#postHandle  \*/  **public** **void** clear() {  **this**.view = **null**;  **this**.model = **null**;  **this**.cleared = **true**;  }  /\*\*  \* Return whether this ModelAndView object is empty,  \* i.e. whether it does not hold any view and does not contain a model.  \*/  **public** **boolean** isEmpty() {  **return** (**this**.view == **null** && CollectionUtils.*isEmpty*(**this**.model));  }  /\*\*  \* Return whether this ModelAndView object is empty as a result of a call to {@link #clear}  \* i.e. whether it does not hold any view and does not contain a model.  \* <p>Returns <code>false</code> if any additional state was added to the instance  \* <strong>after</strong> the call to {@link #clear}.  \* **@see** #clear()  \*/  **public** **boolean** wasCleared() {  **return** (**this**.cleared && isEmpty());  }  /\*\*  \* Return diagnostic information about this model and view.  \*/  @Override  **public** String toString() {  StringBuilder sb = **new** StringBuilder("ModelAndView: ");  **if** (isReference()) {  sb.append("reference to view with name '").append(**this**.view).append("'");  }  **else** {  sb.append("materialized View is [").append(**this**.view).append(']');  }  sb.append("; model is ").append(**this**.model);  **return** sb.toString();  }  } |

测试代码如下：

|  |
| --- |
| **package** com.sxt.web;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.ModelAndView;  **import** org.springframework.web.servlet.mvc.multiaction.MultiActionController;  **import** com.sxt.po.User;  @Controller  @RequestMapping("/user.do")  **public** **class** UserController **extends** MultiActionController {    @RequestMapping(params="method=reg")  **public** ModelAndView reg(String uname){  ModelAndView mv = **new** ModelAndView();  mv.setViewName("index");  // mv.setView(new RedirectView("index"));    User u = **new** User();  u.setUname("高淇");  mv.addObject(u); //查看源代码，得知，直接放入对象。属性名为”首字母小写的类名”。 一般建议手动增加属性名称。  mv.addObject("a", "aaaa");  **return** mv;  }  } |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"gbk"*%>  <%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  </head>  <body>  <h1>${requestScope.a}</h1>  <h1>${requestScope.user.uname}</h1>  </body>  </html> |
| 地址栏输入：<http://localhost:8080/springmvc03/user.do?method=reg>  结果为： |

## 基于spring 3.0mvc 框架的文件上传实现

1. spring使用了apache-commons下得上传组件，因此，我们需要引入两个jar包：

1. apache-commons-fileupload.jar
2. apache-commons-io.jar

2. 在springmvc-servlet.xml配置文件中，增加CommonsMultipartResoler配置：

|  |
| --- |
| <!-- 处理文件上传 -->  <bean id=*"multipartResolver"*  class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"* >  <property name=*"defaultEncoding"* value=*"gbk"*/> <!-- 默认编码 (ISO-8859-1) -->  <property name=*"maxInMemorySize"* value=*"10240"*/> <!-- 最大内存大小 (10240)-->  <property name=*"uploadTempDir"* value=*"/tmp/"*/> <!-- 上传文件临时目录 (WebUtils#TEMP\_DIR\_CONTEXT\_ATTRIBUTE) -->  <property name=*"maxUploadSize"* value=*"-1"*/> <!-- 最大文件大小，-1为无限止(-1) -->  </bean> |

3. 建立upload.jsp页面，内容如下：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"gbk"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>测试springmvc中上传的实现</title>  </head>  <body>  <form action=*"upload.do"* method=*"post"* enctype=*"multipart/form-data"*>  <input type=*"text"* name=*"name"* />  <input type=*"file"* name=*"file"* />  <input type=*"submit"* />  </form>  </body>  </html> |

4. 建立控制器，代码如下：

|  |
| --- |
| package com.sxt.web;  import java.io.File;  import java.util.Date;  import javax.servlet.ServletContext;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.context.ServletContextAware;  import org.springframework.web.multipart.commons.CommonsMultipartFile;  @Controller  public class FileUploadController implements ServletContextAware {  private ServletContext servletContext;    @Override  public void setServletContext(ServletContext context) {  this.servletContext = context;  }    @RequestMapping(value="/upload.do", method = RequestMethod.POST)  public String handleUploadData(String name,**@RequestParam("file")**CommonsMultipartFile file){  if (!file.isEmpty()) {  String path = this.servletContext.getRealPath("/upload/"); //上传文件目录  System.out.println(path);  String fileName = file.getOriginalFilename();  String fileType = fileName.substring(fileName.lastIndexOf("."));  System.out.println(fileType);  File file2 = new File(path,new Date().getTime() + fileType); //新建一个文件  try {  file.getFileItem().write(file2); //将上传的文件写入新建的文件中  } catch (Exception e) {  e.printStackTrace();  }  return "redirect:upload\_ok.jsp";  }else{  return "redirect:upload\_error.jsp";  }  }  } |

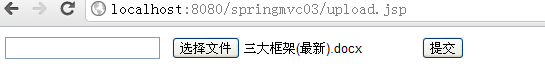
5. 建立upload\_ok.jsp页面

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"gbk"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  </head>  <body>  <h1>上传成功！</h1>  </body>  </html> |

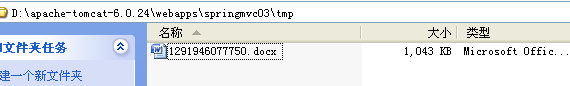
6. 建立upload\_error.jsp页面

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"gbk"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  </head>  <body>  <h1>上传失败！</h1>  </body>  </html> |

1. 发布项目，运行测试：<http://localhost:8080/springmvc03/upload.jsp>



进入项目发布后的目录，发现文件上传成功：



## ****处理ajax请求****

spring使用了jackson类库，帮助我们在java对象和json、xml数据之间的互相转换。他可以将控制器返回的对象直接转换成json数据，供客户端使用。客户端也可以传送json数据到服务器进行直接转换。使用步骤如下：

1. 项目中需要引入如下两个jar包：

jackson-core-asl-1.7.2jar

jackson-mapper-asl-1.7.2jar

2. spring配置文件中修改：

|  |
| --- |
| <mvc:annotation-driven /> <!-- 支持spring3.0新的mvc注解 -->  <!-- 启动Spring MVC的注解功能，完成请求和注解POJO的映射 -->  <bean class=*"org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter"*>  <property name=*"cacheSeconds"* value=*"0"* />  <property name=*"messageConverters"*>  <list>  <bean class=*"org.springframework.http.converter.json.MappingJacksonHttpMessageConverter"*></bean>  </list>  </property>  </bean> |

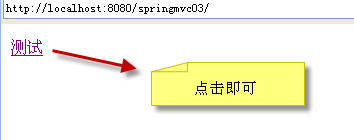
1. 客户端代码a.jsp如下：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"gbk"*%>  <%  String path = request.getContextPath();  String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";  %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <base href=*"*<%=basePath%>*"*>    <title>My JSP 'index.jsp' starting page</title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  <meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>  <meta http-equiv=*"description"* content=*"This is my page"*>  <script>  **function** createAjaxObj(){  **var** req;  **if**(window.XMLHttpRequest){  req = **new** XMLHttpRequest();  }**else**{  req = **new** ActiveXObject("Msxml2.XMLHTTP"); //ie  }  **return** req;  }    **function** sendAjaxReq(){  **var** req = createAjaxObj();  req.open("get","myajax.do?method=test2&uname=张三");  req.setRequestHeader("accept","application/json");  req.onreadystatechange = **function**(){  eval("var result="+req.responseText);  document.getElementById("div1").innerHTML=result[0].uname;  }  req.send(**null**);  }  </script>  </head>    <body>  <a href=*"javascript:void(0);"* onclick="sendAjaxReq();">测试</a>  <div id=*"div1"*></div>  </body>  </html> |

1. 服务器端代码如下：

|  |
| --- |
| package com.sxt.web;  import java.io.UnsupportedEncodingException;  import java.util.ArrayList;  import java.util.List;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestBody;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.ResponseBody;  import com.sxt.po.User;  @Controller  @RequestMapping("myajax.do")  public class MyAjaxController {    @RequestMapping(params="method=test1",method=RequestMethod.GET)  public @ResponseBody List<User> test1(String uname) throws Exception{  String uname2 = new String(uname.getBytes("iso8859-1"),"gbk");  System.out.println(uname2);  System.out.println("MyAjaxController.test1()");  List<User> list = new ArrayList<User>();  list.add(new User("高淇","123"));  list.add(new User("马士兵","456"));    return list;  }    } |

1. 测试。
   1. 启动服务器。输入：<http://localhost:8080/springmvc03/a.jsp>



## Spring中的拦截器

### 定义spring拦截器两种基本方式

1. 实现接口：org.springframework.web.servlet.HandlerInterceptor。

接口中有如下方法需要重写：

注意：参数中的Object handler是下一个拦截器。

* 1. **public** **boolean** preHandle  
     (HttpServletRequest request,HttpServletResponse response,   
     Object handler) **throws** Exception

该方法在action执行前执行，可以实现对数据的预处理，比如：编码、安全控制等。

如果方法返回true，则继续执行action。

* 1. **public** **void** postHandle  
     (HttpServletRequest request,HttpServletResponse response,   
     Object handler, ModelAndView modelAndView) **throws** Exception

该方法在action执行后，生成视图前执行。在这里，我们有机会修改视图层数据。

* 1. **public** **void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) **throws** Exception

最后执行，通常用于释放资源，处理异常。我们可以根据ex是否为空，来进行相关的异常处理。因为我们在平时处理异常时，都是从底层向上抛出异常，最后到了spring框架从而到了这个方法中。

1. 继承适配器：  
   org.springframework.web.servlet.handler.HandlerInterceptorAdapter

这个适配器实现了HandlerInterceptor接口。提供了这个接口中所有方法的空实现。

如下我们写出两个拦截器的示例代码，仅供大家参考：

|  |
| --- |
| **package** com.sxt.interceptor;  **import** javax.interceptor.Interceptors;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.web.servlet.HandlerInterceptor;  **import** org.springframework.web.servlet.ModelAndView;  **public** **class** MyInterceptor **implements** HandlerInterceptor {  @Override  **public** **void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) **throws** Exception {  System.*out*.println("最后执行！！！一般用于释放资源！！");    }  @Override  **public** **void** postHandle(HttpServletRequest request,HttpServletResponse response, Object handler, ModelAndView modelAndView) **throws** Exception {  System.*out*.println("Action执行之后，生成视图之前执行！！");  }  @Override  **public** **boolean** preHandle(HttpServletRequest request,HttpServletResponse response, Object handler) **throws** Exception {  System.*out*.println("action之前执行！！！");  **return** **true**; //继续执行action  }  } |
| **package** com.sxt.interceptor;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.web.servlet.handler.HandlerInterceptorAdapter;  **public** **class** MyInterceptor2 **extends** HandlerInterceptorAdapter {  @Override  **public** **boolean** preHandle(HttpServletRequest request,HttpServletResponse response, Object handler) **throws** Exception {  System.*out*.println("MyInterceptor2.preHandle()");  **return** **true**; //继续执行action  }  } |

1. XML中如何配置。如下为示例代码：

|  |
| --- |
| <mvc:interceptors>  <bean class=*"com.sxt.interceptor.MyInterceptor"*></bean> <!-- 拦截所有springmvc的url！ -->  <mvc:interceptor>  <mvc:mapping path=*"/user.do"* />  <!--<mvc:mapping path="/test/\*" />-->  <bean class=*"com.sxt.interceptor.MyInterceptor2"*></bean>  </mvc:interceptor>  </mvc:interceptors> |