## Java EE框架整合开发入门到实战——Spring+Spring MVC+MyBatis（微课版）

### 习题1

1．Spring的核心容器有哪些模块组成？

答：Spring的核心容器是其他模块建立的基础，由Spring-core、Spring-beans、Spring-context、Spring-Context-support和Spring-expression（Spring表达式语言）等模块组成。

2．如何找到Spring框架的官方API？

答：将下载到的ZIP文件解压缩，解压缩后的目录结构中，docs目录包含Spring的API文档和开发规范。

### 习题2

1．举例说明IoC容器的实现方式有哪些？

答：Sping提供了两种类型的IOC 容器实现：BeanFactory和ApplicationContext。

BeanFactory接口有多个实现类，其中比较常用的是org.springframework.beans.factory.xml.XmlBeanFactory，该类会根据XML配置文件中的定义来装配Bean。创建BeanFactory实例时，需要提供XML文件的绝对路径。例如：

BeanFactory beanFac = new XmlBeanFactory(

new FileSystemResource("D:\\eclipse-workspace\\ch1\\src\\applicationContext.xml")

);

ApplicationContext是BeanFactory的子接口，也称为应用上下文，由org.springframework.context.ApplicationContext接口定义。创建ApplicationContext接口实例通常有三种方法：通过ClassPathXmlApplicationContext创建、通过FileSystemXmlApplicationContext创建、通过Web服务器实例化ApplicationContext容器。

1）通过ClassPathXmlApplicationContext创建

ClassPathXmlApplicationContext将从类路径classPath目录（src根目录）寻找指定的XML配置文件，例如：

ApplicationContext appCon = new ClassPathXmlApplicationContext("applicationContext.xml");

2）通过FileSystemXmlApplicationContext创建

FileSystemXmlApplicationContext将从指定文件的绝对路径中寻找XML配置文件，找到并装载完成ApplicationContext的实例化工作。例如：

ApplicationContext appCon =

new FileSystemXmlApplicationContext("D:\\eclipse-workspace\\ch1\\src\\applicationContext.xml");

3）通过Web服务器实例化ApplicationContext容器

Web服务器实例化ApplicationContext容器时，一般使用基于org.springframework.web.context.ContextLoaderListener的实现方式（需要将spring-web-5.0.2.RELEASE.jar复制到WEB-INF/lib目录中），此方法只需在web.xml中添加如下代码：

<context-param>

<!-- 加载src目录下的applicationContext.xml文件 -->

<param-name>contextConfigLocation</param-name>

<param-value>

**classpath:applicationContext.xml**

</param-value>

</context-param>

<!-- 指定以ContextLoaderListener方式启动Spring容器 -->

<listener>

<listener-class>

**org.springframework.web.context.ContextLoaderListener**

</listener-class>

</listener>

2．在Spring框架中，什么是控制反转？什么是依赖注入？使用控制反转与依赖注入有什么优点？

答：控制反转（Inversion of Control，IoC）是一个比较抽象的概念，是Spring框架的核心，用来消减计算机程序的耦合问题。依赖注入（Dependency Injection，DI）是IoC的另外一种说法，只是从不同的角度，描述相同的概念。

当某个Java对象（调用者，比如您）需要调用另一个Java对象（被调用者，即被依赖对象，比如面包）时，在传统编程模式下，调用者通常会采用“new 被调用者”的代码方式来创建对象（比如您自己制作面包）。这种方式会增加调用者与被调用者之间的耦合性，不利于后期代码的升级与维护。

当Spring框架出现后，对象的实例不再由调用者来创建，而是由Spring容器（比如面包店）来创建。Spring容器会负责控制程序之间的关系（比如面包店负责控制您与面包的关系），而不是由调用者的程序代码直接控制。这样，控制权由调用者转移到Spring容器，控制权发生了反转，这就是Spring的控制反转。

从Spring容器角度来看，Spring容器负责将被依赖对象赋值给调用者的成员变量，相当于为调用者注入它所依赖的实例，这就是Spring的依赖注入。

综上所述，控制反转是一种通过描述（在Spring中可以是XML或注解）并通过第三方去产生或获取特定对象的方式。在Spring中实现控制反转的是IoC容器，其实现方法是依赖注入。

3．Spring框架采用Java的哪种机制进行依赖注入？（ A ）

A．反射 B．异常 C．事件 D．多态

### 习题3

1．Bean的实例化有哪几种常见的方法？

答：Spring框架实例化Bean有三种方式：构造方法实例化、静态工厂实例化和实例工厂实例化。

2．简述基于注解的装配方式的基本用法。

答：1）使用注解创建Bean的实现类，如：

@Component()

/\*\*相当于@Component("annotationUser")或@Component(value = "annotationUser")，annotationUser为Bean的id，默认为首字母小写的类名\*\*/

public class AnnotationUser {

@Value("chenheng")//只注入了简单的值，复杂值的注入目前使用该方式还解决不了

private String uname;

/\*\*省略setter和getter方法\*\*/

}

2）配置注解

需要在配置文件中配置注解，注解配置方式如下：

<context:component-scan base-package="Bean所在的包路径"/>

3．@Autowired与@Resource的区别是什么？

答：该注解与@Autowired功能一样。区别在于，该注解默认是按照名称来装配注入的，只有当找不到与名称匹配的Bean才会按照类型来装配注入；而@Autowired默认按照Bean的类型进行装配，如果想按照名称来装配注入，则需要结合@Qualifier注解一起使用。

4．Bean的默认作用域是（ C ）。

A．page B．request C．singleton D．prototype

5．下面代码片段中，使用@Controller 注解装配了Bean，而Bean的id是（ B ）。

@Controller

public class TestController {

......

}

A．TestController B．testController C．无id D．任意名称

### 习题4

1．什么是AOP？有哪些术语？为什么要学习AOP编程？

答：AOP（Aspect-Oriented Programming），即面向切面编程。

在Spring AOP框架中，涉及以下常用术语：切面、连接点、切入点、通知（增强处理）、引入、目标对象、代理、组入。

AOP采取横向抽取机制，即将分散在各个方法中的重复代码提取出来，然后在程序编译或运行阶段，再将这些抽取出来的代码应用到需要执行的地方。这种横向抽取机制，采用传统的OOP是无法办到的，因为OOP实现的是父子关系的纵向重用。但是AOP不是OOP的替代品，而是OOP的补充，它们相辅相成。

2．在Java中，有哪些常用的动态代理技术？

答：在Java中，有多种动态代理技术，如JDK、CGLIB、Javassist、ASM，其中最常用的动态代理技术有JDK和CGLIB。

3．AspectJ框架的AOP开发方式有哪几种？

答：使用AspectJ实现Spring AOP的方式有两种：一是基于XML配置开发AspectJ，一是基于注解开发AspectJ。

### 习题5

1．什么是编程式事务管理？在Spring中有哪几种编程式事务管理？

答：在代码中显式调用beginTransaction()、commit()、rollback()等事务处理相关的方法，这就是编程式事务管理。

在Spring中有基于底层API的编程式事务管理和基于TransactionTemplate的编程式事务管理。

2．简述声明式事务管理的处理方式。

答：Spring的声明式事务管理，是通过AOP技术实现的事务管理，其本质是对方法前后进行拦截，然后在目标方法开始之前创建或者加入一个事务，在执行完目标方法之后根据执行情况提交或者回滚事务。

Spring的声明式事务管理可以通过两种方式来实现，一是基于XML的方式，一是基于@Transactional注解的方式。

基于XML方式的声明式事务管理是通过在配置文件中配置事务规则的相关声明来实现的。Spring框架提供了tx命名空间来配置事务，<tx:advice>元素来配置事务的通知。配置<tx:advice>元素时，一般需要指定id和transaction-manager属性，其中id属性是配置文件中的唯一标识，transaction-manager属性指定事务管理器。另外，还需要<tx:attributes>子元素，该子元素可配置多个<tx:method>子元素指定执行事务的细节。

@Transactional注解可以作用于接口、接口方法、类以及类方法上。当作用于类上时，该类的所有public方法将都具有该类型的事务属性，同时，也可以在方法级别使用该注解来覆盖类级别的定义。

### 习题6

1．简述MyBatis的工作原理。

答：1）读取MyBatis配置文件mybatis-config.xml。mybatis-config.xml为MyBatis的全局配置文件，配置了MyBatis的运行环境等信息，如数据库连接信息。

2）加载映射文件。映射文件即SQL映射文件，文件中配置了操作数据库的SQL语句，需要在MyBatis配置文件mybatis-config.xml中加载。mybatis-config.xml文件可以加载多个映射文件，每个文件对应数据库中一张表。

3）构造会话工厂。通过MyBatis的环境等配置信息，构建会话工厂SqlSessionFactory。

4）创建SqlSession对象。由会话工厂创建SqlSession对象，该对象中包含执行SQL语句的所有方法。

5）MyBatis底层定义了一个Executor接口来操作数据库，它将根据SqlSession传递的参数动态地生成需要执行的SQL语句，同时负责查询缓存的维护。

6）在Executor接口的执行方法中，有一个MappedStatement类型的参数，该参数是对映射信息的封装，用于存储要映射的SQL语句的id、参数等信息。

7）输入参数映射。输入参数类型可以是Map、List等集合类型，也可以是基本数据类型和POJO类型。输入参数映射过程类似于JDBC对preparedStatement对象设置参数的过程。

8）输出结果映射。输出结果类型可以是Map、List等集合类型，也可以是基本数据类型和POJO类型。输出结果映射过程类似于JDBC对结果集的解析过程。

2．简述MyBatis与Spring的整合过程。

答：1、实现MyBatis与Spring的整合，需要导入相关JAR包，包括MyBatis、Spring以及其他JAR包：1）MyBatis框架所需的JAR包；2）Spring框架所需的JAR包；3）MyBatis与Spring整合的中间JAR包；4）数据库驱动JAR包；5）数据源所需的JAR包。2、在Spring中配置MyBatis工厂。3、使用Spring管理MyBatis的数据操作接口。整合示例参加6.5.4节。

3．除MyBatis持久化框架外，您还知道哪些持久化框架？

答：Hibernate。

### 习题7

1．MyBatis实现查询时，返回的结果集有几种常见的存储方式？请举例说明。

答：可以使用Map存储，也可以使用POJO存储。

<!-- 查询所有用户信息存到Map中 -->

<select id="selectAllUserMap" resultType="map">

select \* from user

</select>

<!-- 使用自定义结果集类型 -->

<resultMap type="com.pojo.MapUser" id="myResult">

<!-- property是com.pojo.MapUser类中的属性-->

<!-- column是查询结果的列名，可以来自不同的表 -->

<id property="m\_uid" column="uid"/>

<result property="m\_uname" column="uname"/>

<result property="m\_usex" column="usex"/>

</resultMap>

2．在MyBatis中针对不同的数据库软件，<insert>元素如何将主键回填？

答：MySQL、SQL Server等数据库的表格可以采用自动递增的字段作为主键。自动回填示例如下：

<!-- 添加一个用户，成功后将主键值回填给uid（po类的属性）-->

<insert id="addUser" parameterType="com.po.MyUser"

keyProperty="uid" useGeneratedKeys="true">

insert into user (uname,usex) values(#{uname},#{usex})

</insert>

如果实际工程中使用的数据库不支持主键自动递增（如Oracle），或者取消了主键自动递增的规则时，可以使用MyBatis的<selectKey>元素来自定义生成主键。具体配置示例代码如下：

<insert id="insertUser" parameterType="com.po.MyUser">

<!-- 先使用selectKey元素定义主键，然后再定义SQL语句 -->

<selectKey keyProperty="uid" resultType="Integer" order="BEFORE">

select if(max(uid) is null, 1 , max(uid)+1) as newUid from user

</selectKey>

insert into user (uid,uname,usex) values(#{uid},#{uname},#{usex})

</insert>

3．在MyBatis中，如何给SQL语句传递参数？

答：1）传递一个基本数据参数，如：

<!-- 根据uid查询一个用户信息 -->

<select id="selectUserById" parameterType="Integer"

resultType="com.po.MyUser">

select \* from user where uid = #{uid}

</select>

2）使用Map接口传递多个参数，如：

<!-- 查询陈姓男性用户信息 -->

<select id="selectAllUser" resultType="com.po.MyUser" parameterType="map">

select \* from user

where uname like concat('%',#{u\_name},'%')

and usex = #{u\_sex}

</select>

上述SQL文件中参数名u\_name和u\_sex是Map的key。

3）使用Java Bean传递多个参数，如：

package com.pojo;

public class SeletUserParam {

private String u\_name;

private String u\_sex;

//此处省略setter和getter方法

}

<select id="selectAllUser" resultType="com.po.MyUser" parameterType="com.pojo.SeletUserParam">

select \* from user

where uname like concat('%',#{u\_name},'%')

and usex = #{u\_sex}

</select>

### 习题8

1．请简述<bind>元素的作用。

答：在模糊查询时，如果使用“${}”拼接字符串，则无法防止SQL注入问题。如果使用字符串拼接函数或连接符号，但不同数据库的拼接函数或连接符号不同，如MySQL的concat函数、Oracle的连接符号“||”。这样，SQL映射文件就需要根据不同的数据库提供不同的实现，显然是比较麻烦，且不利于代码的移植。幸运的是，MyBatis提供了<bind>元素来解决这一问题。如：

<!-- 使用bind元素进行模糊查询 -->

<select id="selectUserByBind" resultType="com.po.MyUser" parameterType="com.po.MyUser">

<!-- bind中uname是com.po.MyUser的属性名 -->

<bind name="paran\_uname" value="'%' + uname + '%'"/>

select \* from user where uname like #{paran\_uname}

</select>

2．在动态SQL元素中，类似分支语句的元素有哪些？如何使用它们？

答：<if>、<choose>、<when>、<otherwise>。

<!-- 使用if元素，根据条件动态查询用户信息 -->

<select id="selectUserByIf" resultType="com.po.MyUser" parameterType="com.po.MyUser">

select \* from user where 1=1

<if test="uname !=null and uname!=''">

and uname like concat('%',#{uname},'%')

</if>

<if test="usex !=null and usex!=''">

and usex = #{usex}

</if>

</select>

<!-- 使用choose、when、otherwise元素，根据条件动态查询用户信息 -->

<select id="selectUserByChoose" resultType="com.po.MyUser" parameterType="com.po.MyUser">

select \* from user where 1=1

<choose>

<when test="uname !=null and uname!=''">

and uname like concat('%',#{uname},'%')

</when>

<when test="usex !=null and usex!=''">

and usex = #{usex}

</when>

<otherwise>

and uid > 10

</otherwise>

</choose>

</select>

### 习题9

1．在开发Spring MVC应用时，如何部署DispatcherServlet？又如何创建Spring MVC的配置文件？

答：在开发Spring MVC应用时，还需要在web.xml中部署DispatcherServlet，代码如下所示：

<?xml version="1.0" encoding="UTF-8"?>

<web-app

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"

id="WebApp\_ID" version="3.1">

<display-name>firstSpringMVC</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

<!--部署DispatcherServlet-->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<!—处理所有URL-->

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

上述DispatcherServlet的servlet对象springmvc初始化时，将在应用程序的WEB-INF目录下查找一个配置文件，该配置文件的命名规则是“servletName-servlet.xml”，如：springmvc-servlet.xml。

2．简述Spring MVC的工作流程。

答：Spring MVC框架主要由DispatcherServlet、处理器映射、控制器、视图解析器、视图组成，其工作原理，如图9.2所示。



图9.2 Spring MVC工作原理图

从图9.2可总结出Spring MVC的工作流程如下：

1．客户端请求提交到DispatcherServlet；

2．由DispatcherServlet控制器寻找一个或多个HandlerMapping，找到处理请求的Controller；

3．DispatcherServlet将请求提交到Controller；

4．Controller调用业务逻辑处理后，返回ModelAndView；

5．DispatcherServlet寻找一个或多个ViewResoler视图解析器，找到ModelAndView指定的视图；

6．视图负责将结果显示到客户端。

### 习题10

1．在Spring MVC的控制器类中如何访问Servlet API？

答：如果需要在请求处理方法中使用Servlet API类型，那么可以将这些类型作为请求处理方法的参数类型。Servlet API参数类型示例代码如下：

package controller;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpSession;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

@Controller

@RequestMapping("/index")

public class IndexController {

@RequestMapping("/login")

public String login(**HttpSession session, HttpServletRequest request**) {

session.setAttribute("skey", "session范围的值");

request.setAttribute("rkey", "request范围的值");

return "login";

}

}

2．控制器接收请求参数的常见方式有哪几种？

答：（1）通过一个实体bean来接收请求参数，适用于get和post提交请求方式。需要注意的是，bean的属性名称必须与请求参数名称相同。（2）通过处理方法的形参接收请求参数，也就是直接把表单参数写在控制器类相应方法的形参中，即形参名称与请求参数名称完全相同。该接收参数方式适用于get和post提交请求方式。（3）通过HttpServletRequest接收请求参数，适用于get和post提交请求方式。（4）通过@PathVariable获取URL中的参数。（5）通过@RequestParam接收请求参数，适用于get和post提交请求方式。（6）通过@ModelAttribute注解接收请求参数，适用于get和post提交请求方式。

3．如何编写基于注解的控制器类？

答：在Spring MVC中，使用org.springframework.stereotype.Controller注解类型声明某类的实例是一个控制器。Controller注解示例代码如下：

package controller;

import org.springframework.stereotype.Controller;

//“@Controller”表示IndexController的实例是一个控制器

@Controller

public class IndexController {

//处理请求的方法

}

在Spring MVC中，使用扫描机制找到应用中所有基于注解的控制器类。所以，为了让控制器类被Spring MVC框架扫描到，需要在配置文件中声明spring-context，并使用<context:component-scan/>元素指定控制器类的基本包（请确保所有控制器类都在基本包及其子包下）。

4．@ModelAttribute可实现哪些功能？

答：通过org.springframework.web.bind.annotation.ModelAttribute注解类型，可经常实现如下两个功能：

**（1）绑定请求参数到实体对象（表单的命令对象）**

该用法如：

@RequestMapping("/register")

public String register(@ModelAttribute("user") UserForm user) {

if("zhangsan".equals(user.getUname())

&& "123456".equals(user.getUpass())){

return "login";

}else{

return "register";

}

}

上述代码中“@ModelAttribute("user") UserForm user”语句的功能有两个，一是将请求参数的输入封装到user对象中；一是创建UserForm实例，以“user”为键值存储在Model对象中，与“model.addAttribute("user", user)”语句功能一样。如果没有指定键值，即“@ModelAttribute UserForm user”，那么创建UserForm实例时，以“userForm”为键值存储在Model对象中，与“model.addAttribute("userForm", user)”语句功能一样。

**（2）注解一个非请求处理方法**

被@ModelAttribute注解的方法，将在每次调用该控制器类的请求处理方法前被调用。

这种特性可以用来控制登录权限，当然控制登录权限的方法很多，例如拦截器、过滤器等。

使用该特性控制登录权限的示例代码如下：

package controller;

import javax.servlet.http.HttpSession;

import org.springframework.web.bind.annotation.ModelAttribute;

public class BaseController {

@ModelAttribute

public void isLogin(HttpSession session) throws Exception {

if(session.getAttribute("user") == null){

throw new Exception("没有权限");

}

}

}

package controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

@Controller

@RequestMapping("/admin")

public class ModelAttributeController extends BaseController{

@RequestMapping("/add")

public String add(){

return "addSuccess";

}

@RequestMapping("/update")

public String update(){

return "updateSuccess";

}

@RequestMapping("/delete")

public String delete(){

return "deleteSuccess";

}

}

上述ModelAttributeController类中的add、update、delete请求处理方法执行时，首先执行父类BaseController中的isLogin判断登录权限。可以通过地址：http://localhost:8080/ch10/admin/add测试登录权限。

### 习题11

1．在MVC框架中，为什么要进行类型转换？

答：在MVC框架中，需要收集用户请求参数，并将请求参数传递给应用的控制器组件。此时存在一个问题，所有的请求参数类型只能是字符串数据类型，但Java是强类型语言，所以，MVC框架必须将这些字符串请求参数转换成相应的数据类型。

2．Converter与Formatter的区别是什么？

答：Spring MVC框架的Formatter<T>与Converter<S, T>一样，也是一个可以将一种数据类型转换成另一种数据类型的接口。但不同的是，Formatter<T>的源数据类型必须是String类型，而Converter<S, T>的源数据类型是任意数据类型。

在Web应用中，由HTTP发送的请求数据到控制器中都是以String类型获取。因此，在Web应用中选择Formatter<T>比选择Converter<S, T>更加合理。

3．在Spring MVC框架中，如何自定义类型转换器类，又如何注册类型转换器？

答：自定义类型转换器类需要实现Converter<S, T>接口，重写convert(S)接口方法。convert(S)方法功能是将源数据类型S转换成目标数据类型T。

在配置文件springmvc-servlet.xml中注册自定义类型转换器，代码如下：

<!-- 注册类型转换器GoodsConverter -->

<bean id="conversionService" class="org.springframework.context.support.ConversionServiceFactoryBean">

<property name="converters">

<list>

<bean class="converter.GoodsConverter"/>

</list>

</property>

</bean>

4．在Spring MVC框架中，如何自定义格式化转换器类，又如何注册格式化转换器？

答：自定义格式化转换器，就是编写一个实现org.springframework.format.Formatter接口的Java类。该接口声明如下：

public interface Formatter<T>

这里的T表示由字符串转换的目标数据类型。该接口有parse和print两个接口方法，自定义格式化转换器类必须覆盖它们。

public T parse(String s, java.util.Locale locale)

public String print(T object, java.util.Locale locale)

parse方法的功能是利用指定的Locale将一个String类型转换成目标类型，print方法与之相反，返回目标对象的字符串表示。

在配置文件springmvc-servlet.xml中注册格式化转换器，代码如下：

<!-- 注册MyFormatter-->

<bean id="conversionService" class="org.springframework.format.support.FormattingConversionServiceFactoryBean">

<property name="formatters">

<set>

<bean class="formatter.MyFormatter"/>

</set>

</property>

</bean>

### 习题12

1．举例说明数据绑定的优点。

答：数据绑定是将用户参数输入值绑定到领域模型的一种特性，在Spring MVC的Controller和View参数数据传递中，所有HTTP请求参数的类型均为字符串，如果模型需要绑定的类型为double或int，则需要手动进行类型转换，而有了数据绑定后，就不再需要手动将HTTP请求中的String类型转换为模型需要的类型。数据绑定的另一个好处是，当输入验证失败时，会重新生成一个HTML表单，无需重新填写输入字段。

2．Spring MVC有哪些表单标签？其中，可以绑定集合数据的标签有哪些？

答：表单标签库中有form、input、password、hidden、textarea、checkbox、checkboxes、radiobutton、radiobuttons、select、option、options、errors。

可以绑定集合数据的标签有checkbox、checkboxes、radiobuttons**、**select标签、options。

### 习题13

1．在Spring MVC框架中，如何自定义拦截器？又如何配置自定义拦截器？

答：定义一个拦截器可以通过两种方式：一种是通过实现HandlerInterceptor接口或继承HandlerInterceptor接口的实现类来定义；另一种是通过实现WebRequestInterceptor接口或继承WebRequestInterceptor接口的实现类来定义。

让自定义的拦截器生效，需要在Spring MVC的配置文件中进行配置，配置示例代码如下：

<!-- 配置拦截器 -->

<mvc:interceptors>

<!-- 配置一个全局拦截器，拦截所有请求 -->

<bean class="interceptor.TestInterceptor"/>

<mvc:interceptor>

<!-- 配置拦截器作用的路径 -->

<mvc:mapping path="/\*\*"/>

<!-- 配置不需要拦截作用的路径 -->

<mvc:exclude-mapping path=""/>

<!-- 定义在<mvc:interceptor>元素中，表示匹配指定路径的请求才进行拦截 -->

<bean class="interceptor.Interceptor1"/>

</mvc:interceptor>

<mvc:interceptor>

<!-- 配置拦截器作用的路径 -->

<mvc:mapping path="/gotoTest"/>

<!-- 定义在<mvc:interceptor>元素中，表示匹配指定路径的请求才进行拦截 -->

<bean class="interceptor.Interceptor2"/>

</mvc:interceptor>

</mvc:interceptors>

2．请简述单个拦截器和多个拦截器的执行流程。

答：在配置文件中，如果只定义了一个拦截器，程序首先将执行拦截器类中的preHandle()方法，如果该方法返回true，程序将继续执行控制器中处理请求的方法，否则中断执行。如果preHandle()方法返回true，并且控制器中处理请求的方法执行后返回视图前，将执行postHandle()方法。返回视图后，才执行afterCompletion()方法。

在需要Web应用中，通常有多个拦截器同时工作，这时它们的preHandle()方法将按照配置文件中拦截器的配置顺序执行，而它们的postHandle()方法和afterCompletion()方法则按照配置顺序的反序执行。

### 习题14

1．如何创建Spring验证器类？

答：创建自定义Spring验证器，需要实现org.springframework.validation.Validator接口。该接口有两个接口方法：

boolean supports(Class<?> klass)

void validate(Object object, Errors errors)

supports方法返回true时，验证器可以处理指定的Class。validate方法的功能是验证目标对象object，并将验证错误消息存入Errors对象。

往Errors对象存入错误消息的方法是reject或rejectValue方法。这两个方法的部分重载方法如下：

void reject(String errorCode)

void reject(String errorCode, String defaultMessage)

void rejectValue(String field, String errorCode)

void rejectValue(String field, String errorCode, String defaultMessage)

一般情况下，只需要给reject或rejectValue方法一个错误代码，Spring MVC框架就会在消息属性文件中查找错误代码，获取相应错误消息。具体示例如下：

if(goods.getGprice() > 100 || goods.getGprice() < 0){

errors.rejectValue("gprice", "gprice.invalid");//gprice.invalid为错误代码

}

2．举例说明JSR 303验证的标注类型的使用方法。

答：JSR 303不需要编写验证器，但需要利用它的标注类型在领域模型的属性上嵌入约束。

**（1）空检查**

@Null：验证对象是否为null。

@NotNull：验证对象是否不为null，无法查检长度为0的字符串。

@NotBlank：检查约束字符串是不是null，还有被trim后的长度是否大于0，只对字符

串，且会去掉前后空格。

@NotEmpty：检查约束元素是否为null或者是empty。

示例如下：

@NotBlank(message="{goods.gname.required}")//goods.gname.required为属性文件的错误代码

private String gname;

**（2）booelan检查**

@AssertTrue：验证boolean属性是否为true。

@AssertFalse：验证boolean属性是否为false。

示例如下：

@AssertTrue

private boolean isLogin;

**（3）长度检查**

@Size(min=, max=)：验证对象（Array，Collection，Map，String）长度是否在给定的范围之内。

@Length(min=, max=)：验证字符串长度是否在给定的范围之内。

示例如下：

@Length(min=1,max=100)

private String gdescription;

**（4）日期检查**

@Past：验证Date和Calendar对象是否在当前时间之前。

@Future：验证Date和Calendar对象是否在当前时间之后。

@Pattern：验证String对象是否符合正则表达式的规则。

示例如下：

@Past(message="{gdate.invalid}")

private Date gdate;

**（5）数值检查**

@Min：验证Number和String对象是否大等于指定的值。

@Max：验证Number和String对象是否小等于指定的值。

@DecimalMax：被标注的值必须不大于约束中指定的最大值，这个约束的参数是一个通过BigDecimal定义的最大值的字符串表示，小数存在精度。

@DecimalMin：被标注的值必须不小于约束中指定的最小值，这个约束的参数是一个通过BigDecimal定义的最小值的字符串表示，小数存在精度。

@Digits：验证Number和String的构成是否合法。

@Digits(integer=,fraction=)：验证字符串是否符合指定格式的数字，interger指定整数精度，fraction指定小数精度。

@Range(min=, max=)：检查数字是否介于min和max之间。

@Valid：对关联对象进行校验，如果关联对象是个集合或者数组，那么对其中的元素进行校验，如果是一个map，则对其中的值部分进行校验。

@CreditCardNumber：信用卡验证。

@Email：验证是否是邮件地址，如果为null，不进行验证，通过验证。

示例如下：

@Range(min=0,max=100,message="{gprice.invalid}")

private double gprice;

### 习题15

1．在JSP页面中可以通过Spring提供的（ B ）标签来输出国际化信息。

A．input B．message C．submit D．text

2．资源文件的后缀名为（ D ）。

A．txt B．doc C．property D．properties

3．什么是国际化？国际化资源文件的命名格式是？

答：国际化的目的，就是根据用户的语言环境不同，输出与之相应的页面给用户，以示友好。

资源文件的命名可以有如下3种形式：

* baseName.properties
* baseName\_language.properties
* baseName\_language\_country.properties

baseName是资源文件的基本名称，由用户自由定义。但是language和country就必须为Java所支持的语言和国家/地区代码。例如：

中国大陆： baseName\_zh\_CN.properties

美国： baseName\_en\_US.properties

### 习题16

1．简述Spring MVC框架中统一异常处理的常用方式。

答：使用Spring MVC提供的简单异常处理器SimpleMappingExceptionResolver、实现Spring的异常处理接口HandlerExceptionResolver 自定义自己的异常处理器、使用@ExceptionHandler注解实现异常处理，等3种处理方式讲解Spring MVC应用的异常统一处理。

2．如何使用@ExceptionHandler注解进行统一异常处理？

答：创建BaseController类，并在类中使用@ExceptionHandler注解声明异常处理方法，具体代码如下：

package controller;

import java.sql.SQLException;

import javax.servlet.http.HttpServletRequest;

import org.springframework.web.bind.annotation.ExceptionHandler;

import exception.MyException;

public abstract class BaseController {

/\*\* 基于@ExceptionHandler异常处理 \*/

@ExceptionHandler

public String exception(HttpServletRequest request, Exception ex) {

request.setAttribute("ex", ex);

// 根据不同错误转向不同页面

if(ex instanceof SQLException) {

return "sql-error";

}else if(ex instanceof MyException) {

return "my-error";

} else {

return "error";

}

}

}

将所有需要异常处理的Controller都继承BaseController类，示例代码如下：

@Controller

public class TestExceptionController extends BaseController{

……

}

使用@ExceptionHandler注解声明统一处理异常时，不需要配置任何信息。

### 习题17

1．基于表单的文件上传，应将表单的 enctype属性值设置为（ A ）。

A．multipart/form-data

B．application/x-www-form-urlencoded

C．text/plain

D．html/text

2．在Spring MVC框架中，如何限定上传文件的大小？

答：<!-- 配置MultipartResolver 用于文件上传 使用spring的CommosMultipartResolver -->

<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver"

p:defaultEncoding="UTF-8"

p:maxUploadSize="5400000"

p:uploadTempDir="fileUpload/temp"

>

<!--D:\spring mvc workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\fileUpload -->

</bean>

<!-- defaultEncoding="UTF-8" 是请求的编码格式，默认为iso-8859-1

maxUploadSize="5400000" 是允许上传文件的最大值，单位为字节

uploadTempDir="fileUpload/temp" 为上传文件的临时路径 -->

3．单文件上传与多文件上传有什么区别？

答：上传单文件时，需要模型类封装文件信息，在该类使用MultipartFile类型声明文件域属性；上传多文件是，需要模型类封装文件信息，在该类使用List<MultipartFile>类型声明文件域属性。

### 习题18

1．在Web应用程序中有以下程序代码段，执行后转发到某个JSP页面：

ArrayList<String> dogNames = new ArrayList<String>();

dogNames.add("goodDog");

request.setAttribute("dogs", dogNames);

以下（ B ）选项可以正确地使用EL取得数组中的值。

A．${ dogs .0}

B．${ dogs [0]}

C．${ dogs .[0]}

D．${ dogs "0"}

2．（ D ）JSTL标签可以实现Java程序中的if语句功能。

A．<c:set>

B．<c:out>

C．<c:forEach>

D．<c:if>

3．（ A ）不是EL的隐含对象。

A．request

B．pageScope

C．sessionScope

D．applicationScope

4．（ C ）JSTL标签可以实现Java程序中的for语句功能。

A．<c:set>

B．<c:out>

C．<c:forEach>

D．<c:if>

### 习题19

1．简述SSM框架整合所需JAR包。

答：因为Spring MVC是Spring框架中的一个子模块，所以Spring与Spring MVC之间不存在整合的问题。实际上SSM框架的整合只涉及Spring与MyBatis的整合，以及Spring MVC与MyBatis的整合。

实现SSM框架的整合，首先需要准备三个框架的JAR包以及其他整合所需要的JAR包。在第6章6.5节已经讲解了Spring与MyBatis框架整合所需要的JAR包，在这里只需再加入Spring MVC的相关JAR包（spring-web-5.0.2.RELEASE.jar和spring-webmvc-5.0.2.RELEASE.jar）即可。

2．简述SSM框架整合时，Spring MVC核心配置文件和web.xml文件的配置信息。

答：在WEB-INF目录下，创建Spring MVC的核心配置文件springmvc-servlet.xml。该文件中仅配置控制器扫描包和视图解析器。

在WEB-INF目录下，创建web.xml文件，并在该文件中实例化ApplicationContext容器、启动Spring容器、配置DispatcherServlet以及配置字符编码过滤器。