# SpringMVC

**SpringMVC简介：**

SpringMVC是一个表示层框架，搭建真实环境

SpringMVC的使用方法：注解和配置。（**注解为主**）

**SpringMVC是Spring3.x的一个模块**，其实就是用MVC提供的表示层框架。

**SpringMVC对视图组件没有必然要求，不一定要使用jsp，struts1,struts2规定了只能用jsp。**

SpringMVC的**映射器对象**，相当于struts1 的ModuleConfig

SpringMVC的转向：可以自动的封装结果数据。相当于struts1的ActionForward.

**SpringMVC优点：**

使用简单，学习成本低

容易写出性能高的程序

灵活性好

结构清晰

编码效率：jsp+servlet—> struts1🡪  **SpringMVC🡪struts2（越来越高）**

执行效率：jsp+servlet—> struts1🡪  **SpringMVC🡪struts2（越来越低）**

Struts1：编码效率低，执行效率高，配置量大，结构清晰

Struts2：编码效率高，运行效率低，需要维护ognl

表示层框架：接收用户请求，封装请求数据，将结果转发到对应的jsp页面。

**SpringMVC的（开发）环境搭建：**

1. 创建web项目
2. 导入jar包（dist下的所有jar包）基本的jar：org.springframework.aop-3.0.5.RELEASE.jar ：与Aop 编程相关的包

org.springframework.beans-3.0.5.RELEASE.jar ：提供了简捷操作bean 的接口

org.springframework.context-3.0.5.RELEASE.jar ：构建在beans 包基础上，用来处理资源文件及国际化。

org.springframework.core-3.0.5.RELEASE.jar ：spring 核心包

org.springframework.web-3.0.5.RELEASE.jar ：web 核心包，提供了web 层接口

org.springframework.web.servlet-3.0.5.RELEASE.jar ：web 层的一个具体实现包，DispatcherServlet也位于此包中。

**建议在搭建环境中导入spring3.0 的所有jar 包(所有jar 包位于dist 目录下)。**

1. 编写配置文件,SpringMVC的中央控制器DispatcherServlet,和struts1配置ActionServlet一样。在web.xml中配置

<servlet>

//spmvc命名和springMVC的主配置文件名相关

< servlet-name>**spmvc**</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

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

<servlet>

<servlet-mapping>

< servlet-name>**spmvc**</servlet-name>

< url-pattern >\*.do</url-pattern>

</servlet-mapping>

1. 配置SpringMVC的主配置文件：**spmvc**-servlet.xml servlet-name 作为前缀，文件放在WEB-INF下

从springmvc到struts2使用的tomcat版本至少是6.0及以上。

spmvc-servlet.xm文件中的配置信息

<beans>

<bean id = *"simpleUrlHandlerMapping"*

class =*"org.springframework.web.servlet.*

*handler. SimpleUrlHandlerMapping"*>

<property name =*"mappings"*>

<props>

<prop key =*"/login.do"*>loginControl</prop>

</props>

</property>

</bean>

<bean id =*"loginControl"*

class =*"com.asm.spmvclogin"*/>

<bean id=*"viewResolver"* class=*"org.springframework.web.servlet.view.*

*InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/page/"* />

<property name=*"suffix"* value=*".jsp"* />

</bean>

</beans>

**SpringMVC的流程：**

1、Tomcat一启动，读取web.xml，

2、根据web.xml的配置创建DispatchServlet对象，

3、执行该Servlet对象执行init()方法，读取整个springmvc的配置文件（spmvc-servlet.xml）。

4、一读取主配置文件spmvc-servlet.xml创建WebFactory（web 工厂），

5、在web工厂中创建以下几个对象：

**创建分派器（也叫映射器）对象***SimpleUrlHandlerMapping：将请求交给对应的Action*，分派器有个属性 mappings,是一个props，key和value都是String类型

**创建处理请求的Action对象**

**创建视图定位器对象***InternalResourceViewResolver：给转向信息加上前缀和后缀*

**通过配置方式登录示例：**

1、发现\*.do请求，交给DispatcherServlet处理

2、截取请求路径，截取结果包含.do（/login.do）

3、到web工厂中找到分派器，把请求交给分派器

4、分派器到分派器对象的mappings属性的props配置中找key（/login.do）对应的value.（loginControl）

5、根据value（loginControl）取出id标识为loginControl的Action对象。（实现了Controller的接口的类标识）

6、该Action类实现了handleRequest()方法（handleRequest是接口中的方法）

handleRequest()方法中：

从request中拿到请求参数，创建业务层对象，调用业务对象的方法。

//封装转向信息

ModelAndView mav2= new ModelAndView("failure");

mav2.addObject("username", username); //封装结果信息

mav2.addObject("password", password);

return mav2;

ModelAndView：封装转向信息，也封装结果数据。

ModelAndView中有个map ，

mavl.addObject(“username”,username);// key是String类型，value是Object类型。

7、**DispatcherServlet的作用：**

1. 从ModelAndView中拿出存储的数据，放到request内置对象中（key作为属性名，value作为属性值）
2. 从ModelAndView中拿到转向字符串，找到视图定位器，加上前缀和后缀，找到确定的转向页面

8、**结果显示：**

在加上前缀和后缀指定位置的failure.jsp页面中：使用EL表达式直接输出登录失败的用户名和密码：${username}， ${password}

**使用注解SpringMVC（annocation）完成登录：**

在web.xml中的配置

<servlet>

//命名和主配置文件名相关，主配置文件名为：spring-servlet.xml

< servlet-name>**spring**</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

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

<servlet>

<servlet-mapping>

< servlet-name>**spring**</servlet-name>

< url-pattern >\*.do</url-pattern>

</servlet-mapping>

//配置监听器，在该监听器中创建BeanFactory

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

<!-- 指定Spring ，Bean的配置文件所在目录（业务层对象在这个bean中通过IOC创建）。默认配置在WEB-INF目录下 -->

<context-param>

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

<param-value>/WEB-INF/applicationContext.xml

</param-value>

</context-param>

</web-app>

**Tomcat一启动：**

1. 先创建Listener监听器对象
2. 一创建监听器对象对application的状态监听，创建Application对象时，调用监听器的contextInitialized()方法
3. 在contextInitialized()方法中通过application内置对象拿到初始化参数信息创建BeanFactory（BeanFactory：处理业务）
4. 创建DispatcherServlet对象，调用init（）方法，读取springmvc主配置文件信息 spring-servlet.xml
5. 根据springmvc的主配置文件spring-servlet.xml

创建webFactory(web 工厂：处理请求)

6、 创建视图定位器对象；

创建分派器对象（分派器也叫映射器）

分派器自动扫描项目，到指定的包下找Action类(Action类有被@Controller注解标识)，自动创建Action对象。

7、 在Action类中找@Resource(name=”loginService”)：一创建Action对象，就自动到BeanFactory中找到bean中id为loginService的对象,自动注入业务层对象。

Spring-servlet.xml(springMVC主配置文件中的信息)

<!-- 启用spring mvc 注解 -->

<context:annotation-config />

<!-- 设置使用注解的类所在的jar包 -->

<context:component-scan base-package= *"com.asm"*> </context:component-scan>

<!-- 完成请求和注解POJO的映射 -->

<bean class=*"org.springframework.web.servlet.mvc.*

*annotation.AnnotationMethodHandlerAdapter"* />

<!-- 对转向页面的路径解析。prefix：前缀， suffix：后缀 -->

<bean class=*"org.springframework.web.servlet.view.*

*InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/page/"* />

<property name=*"suffix"* value=*".jsp"* />

</bean>

**@Controller**：注解标识，标识这个类就是Action（用在类上）

**@RequestMapping("/login.do")**  // 请求url地址映射，类似Struts的action-mapping（用在方法上）

**@Resource(name=”loginService”)**：一创建Action对象，就自动到BeanFactory中找到bean中id为loginService对象,自动注入业务层对象。（用在属性上）

**@RequestParam(value="username")String name**：一般用于当表单的请求参数名和方法中形参不一致时，将指定的请求参数赋值给方法中的形参

可以简写为：**@RequestParam("username")**（用在参数上）

SpringMVC注解方式登录示例：

1、 \*.do请求，交给DispatcherServlet处理

2、 截取请求路径，/login.do

3、 到web工厂中找到分派器，把请求交给分派器

4、 分派器找到Action类（@Controller标识的类）,根据注解@requestMapping找到对应的方法

5、 找到对应方法后，收集表单数据：

如果方法的形参名字和表单中参数名一样，会自动设定

如果不一样就通过注解设定@RequestParam(value="username")String name//把表单中参数名为username的参数值设到此方法中的name参数中。

需要传送数据的时候，在方法中也要写上request参数，就会自动注入request内置对象

然后再 调用request.setAttribute();方法，设定属性名和属性值

6、 返回字符串，success

到WebFactory中拿到视图解析器，加上前缀和后缀，完成转向。

7、 在jsp页面中通过EL表达式输出存在request中的属性值

${username}

SpringMVC对Action类没有要求，通过注解将请求映射到方法上面。返回值可以为String,也可以为ModelAndView.不能无返回值。

知识点解析：

表单中获取的参数类型都是String类型

当前方法中的形参age为int类型，SpringMVC完成自动转换。

return new ModelAndView(“success”);**//默认为转发，需要用到视图解析器加上前缀和后缀**

**如果需要重定向(不需要使用视图解析器)：**

return new ModelAndView(new RedirectView (“./index.jsp”));// 重定向的第一种方法

return new ModelAndView(“redirect: ../index.jsp”));//重定向的第二种方法

**引入ActionForm User收集表单数据**

表单bean，不需要继承任何的类，和普通的bean一样属性私有构造方法公有。

pubic String testLogin3(User user){}

1、创建User对象

2、自动的将表单中的参数设到User对象中（这个User对象的属性名和表单中的参数名必须保持一致），没有对应的参数名时 就取对象的默认值。String类型取null，int类型取0.

3、自动将收集完数据的User对象设到request内置对象中，属性名为user.

在jsp页面中，直接通过el表达式输出：${user.username}

// 获取applicationContext.xml中bean的id为loginService的，并注入

**@Resource(name = "loginService")**

private LoginService loginService; //等价于spring传统注入方式写get和set方法，这样的好处是简洁工整，省去了不必要得代码

# SpringMVC的国际化

国际化使用的技术：国际化资源文件

相应的国际化标签

SpringMVC几乎不支持动态国际化，只支持静态国际化。

国际化资源文件：基名+Locale(语言名 zh+国家名CN),示例：MessageResources\_zh\_CN.properties

中文国际化资源文件：转换为Unicode码

**根据浏览器显示语言实现国际化步骤：**

1. 在src下添加国际化资源文件
2. 在applicationContext.xml文件中配置国际化资源文件

<bean id=”messageSource” class=”^”>

1. 使用标签处理国际化

必须导入标签库prefix，uri

<spring:message code=”国际化标签的key”/>

Tomcat启动：

先创建Listener对象，对application对象的创建做监听

Application对象一创建就调用监听器的contextInitialized()方法

创建中央控制器DispatcherServlet，执行init()方法，读取主配置文件

创建请求分派器

创建后端控制器

创建拦截器LocaleChangeInterceptor（处理国际化）

创建国际化处理器对象SessionLocaleResolver

发送请求执行项目，默认调用index.jsp，

1、在index.jsp页面发送login.do的请求

2、Tomcat创建request接收请求字符串，把http协议中的locale信息也封装到request。（Locale,Cookie,IP）

3、Tomcat自动将locale信息封装到locale对象设到session中

4、发现是/login.do请求，tomcat把请求交给springmvc

5、springmvc截取请求路径

6、先到webFactory中有没有拦截器拦截此请求

7、没有就在webFactory中找到请求分派器对象

8、通过分派器找到对应的后端控制器

9、执行后端控制器的handleRequest方法，返回ModeAndView对象

10、找到视图解析器，加上前缀和后缀，完成转向

在jsp页面使用国际化资源标签：

1. 到BeanFactory中找到国际化资源对象拿到国际化资源文件的基名
2. 到Session通过属性名中拿到Locale
3. (基名+locale)找到相应的国际化资源文件，根据code指定的key,找到国际化资源文件中对应的value进行显示。

Springmvc把国际化基名放在beanFactory中

可以人为的设定locale信息，控制显示语言

**手动控制显示语言实现国际化步骤：**

1. 在src下添加国际化资源文件

2、在applicationContext.xml文件中配置国际化资源文件

<bean id=”messageSource” class=”^”>

3、使用标签处理国际化

必须导入标签库prefix，uri

<spring:message code=”国际化标签的key”/>

**请求参数必须是locale,参数值必须是按照格式填写**

**Chinese.do?locale=zh\_CN**

Tomcat启动：

先创建Listener对象，对application对象的创建做监听

Application对象一创建就调用监听器的contextInitialized()方法

创建中央控制器DispatcherServlet，执行init()方法，读取主配置文件

创建请求分派器

创建后端控制器

创建拦截器LocaleChangeInterceptor（处理国际化）

创建国际化处理器对象SessionLocaleResolver

发送请求执行项目，默认调用index.jsp，

1、在index.jsp页面发送Chinese.do的请求

2、Tomcat创建request接收请求字符串，把http协议中的locale信息也封装到request。（Locale,Cookie,IP）

3、Tomcat自动将locale信息封装到locale对象设到session中

4、发现是/Chinese.do请求，tomcat把请求交给springmvc

5、springmvc截取请求路径/Chinese.do

6、先到webFactory中有没有拦截器拦截此请求

7、发现拦截请求，就找到拦截器对象LocaleChangeInterceptor，执行拦截器对象的preHandle方法，从request中拿到locale信息，更改locale信息

8、找到国际化处理器对象SessionLocaleResolver，将locale信息设定到Session中

9、在webFactory中找到请求分派器对象

10、通过分派器找到对应的后端控制器

11、执行后端控制器的handleRequest方法，返回ModeAndView对象

12、找到视图解析器，加上前缀和后缀，完成转向

在jsp页面使用国际化资源标签：

到BeanFactory中找到国际化资源对象拿到国际化资源文件的基名

到Session通过属性名中拿到Locale

(基名+locale)找到相应的国际化资源文件，根据code指定的key,找到国际化资源文件中对应的value进行显示。

# Spring中的拦截器

作用：用来拦截Spring的相关请求

学习springmvc的拦截器目的：了解拦截器的特点，为国际化做准备

**拦截器的语法**

1、实现接口：**HandlerInterceptor**

必须实现接口中的方法：

afterCompletion：对结果进行拦截，最后执行，用于释放资源，处理异常，

postHandle：在执行完后端处理器的相应方法，进行拦截（找到视图定位器，生成视图之前）对结果数据和页面信息做处理。

preHandle：在请求进入后端处理器之前调用，处理国际化问题，处理编码问题（请求拦截）

2、继承适配器类：**HandlerInterceptorAdapter**

适配器是对接口中的方法进行了空实现，这里可以不实现三个方法，需要用到哪个就实现哪个方法。

**类的继承和实现接口有什么不同：**

接口和类不是同一个概念

**语法：**定义接口是interface,实现是implements ；使用class定义类继承是extends，

一个子接口能继承多个接口

一个子类只能继承一个接口

**语义：**接口是功能规范

父类对外不提供构造方法：构造方法私有

接口中没有构造方法

**接口数据结构的特点：**

接口中的属性：public static final

接口中的方法：public abstract

**创建子类对象：**会调用父类的构造方法为父类的私有属性初始化，采用动态绑定的机制；不会创建父类对象。

**子接口继承多个父接口，父接口中有相同的方法，一个类实现子接口后，该实现哪个父类的方法？**

使用预运算符，：,指定具体的方法。

**Java提出的面向接口编程：**

降低耦合性，提高可维护性

3、拦截器的配置：

<mvc:interceptors>

<mvc:interceptor>

//指定拦截的请求，如果path=”/\*”表示拦截所有请求

<mvc:mapping path=”/login.do”/>

//配置拦截器的位置

<bean class=”cn.springmvc.myInterceptor”/>

</mvc:interceptor>

</mvc:interceptors>

**Tomcat启动：（配置方式）**

1、Tomcat一启动,创建DispatcherServlet对象

2、执行init()方法，创建web工厂，

3、在web工厂中会创建以下几个对象

创建**分派器对象**：（只有一个，在webFactory中）

**后端处理器对象**：(配置方式有多个，注解方式只有一个，在webFactory中)

**视图定位器对象**

**拦截器对象**（配置了几个拦截器就创建几个拦截器对象），并且设定拦截范围

发送请求：

4、发送请求：/login.do,交给tomcat处理

5、tomcat创建request内置对象接收请求（请求中包含Locale，Cookie，IP地址）

request.getRequestURL()；拿到请求路径

request.getLocale();拿到请求中的locale信息

request.getRemoveAddr();拿到请求的ip地址

request.getCookies();拿到cookies,是个数组

6、将请求交给DispatcherServlet 处理，截取请求

7、先到webFactory中查看有没有拦截器对象，如果有就找到拦截器指定拦截的请求。

8、如果当前请求符合就拦截要求，拦截器对象就会调用preHandle()方法处理请求，该方法的返回的结果为boolean类型，为true的时候（**如果还有下一个拦截器就交给下一个拦截器，如果没有就请求下传**）为false的时候就直接返回，不会将请求下传。

9、请求下传后到web工厂中找到分派器，将请求交给分派器

10、**分派器根据请求找到相应的后端控制器**。（分派器到mappings的prop中找到id对应的后端控制器）

11、执行后端控制器的handlerRequest()方法，注入request和response内置对象，调用业务层方法，返回值一定是ModelAndView（因为是配置方式，接口中的方法是固定的）

12、handlerRequest()方法执行完后，SpringMVC将调用拦截器的postHandle()对返回值ModelAndView进行拦截。

13、postHandle()方法执行完后，SpringMVC拿到ModelAndView对象做以下这些事情：

根据ModelAndView对象解析出结果数据并且设到request内置对象中，

从该对象中拿到转向信息，到webFactory中拿到视图定位器，加上前缀和后缀，生成完成的转向信息。转向相应的结果页面。

14、在结果页面执行完，即将回到浏览器显示结果的时候，调用afterCompletion()方法，进行最后一次拦截。（处理异常，关闭资源，懒加载关闭Session等操作）

配置多个拦截器：

<mvc:interceptors>

<mvc:interceptor>

//指定拦截的请求，如果path=”/\*”表示拦截所有请求

<mvc:mapping path=”/login.do”/>

<bean class=”cn.springmvc.myInterceptor1”/>

</mvc:interceptor>

<mvc:interceptor>

<mvc:mapping path=”/\*”/>

<bean class=”cn.springmvc.myInterceptor2”/>

</mvc:interceptor>

</mvc:interceptors>

**执行过程：**

请求login.do,到拦截器对象中看看是否拦截这个请求

如果符合多个拦截器请求，拦截顺序是按照配置顺序拦截（依次执行）

**preHandle方法的执行：**

先调用第一个拦截器的preHandle()方法，发现还有下一个拦截器就继续调用第二个拦截器的preHandle()方法，如果还有下一个拦截器就继续调用下一个拦截器的preHandle()方法，知道直到没有后就将请求下传。

**handleRequest的方法的执行：**

找到分派器对象，分派器找到对应的后端处理器，执行完handleRequest()方法。

**postHandle()方法的执行：**

handleRequest()方法执行完后先调用最后一个拦截器的postHandle()方法，依次往前，最后执行第一个拦截器的postHandle()方法。

所有的拦截器的postHandle()方法执行完后， SpringMVC拿到ModelAndView对象做以下这些事情：

根据ModelAndView对象解析出结果数据并且设到request内置对象中，

从该对象中拿到转向信息，到webFactory中拿到视图定位器，加上前缀和后缀，生成完成的转向信息。转向相应的结果页面。

**afterCompletion方法的执行：**

在结果页面执行完，即将回到浏览器显示结果的时候，先调用最后一个拦截器的afterCompletion()方法，做结果拦截，然后依次往上，最后执行第一个拦截器的afterCompletion()方法。（处理异常，关闭资源，懒加载关闭Session等等操作）

**多个拦截器存在时**，只要有一个拦截器的preHandle()方法返回的是false,请求不会就传到后端控制器，也不会再继续执行preHandle()以外的方法，直接将请求返回。

# SpringMVC的上传

SpringMVC的上传和struts1的原理类似，更加智能

1. 必须导入两个包 apache-commons-fileupload.jar

apache-commons-io.jar

1. 在主配置文件中添加 上传控制器 ConmmonMultipartResolver

defaultEncoding value=”gbk” 默认编码

maxInMemorySize value=”1000” 上传文件大小

uploadTempDir value=”/upload/” 上传文件夹，该文件夹不存在就会自动创建

maxUploadSize value=”-1” 指定上传文件的大小，-1表示不限制上传文件的大小

**指定的上传文件夹是在创建上传控制器对象的时候的创建，tomcat一启动就创建了，由上传控制器创建。**

**SpringMVC的主配置文件的配置：**

<!-- 配置启用springmvc的注解 -->

<context:annotation-config/>

<!-- 配置注解扫描的包 -->

<context:component-scan base-package=*"cn.springmvc"*></context:component-scan>

<!-- 配置视图解析器 -->

<bean id=*"viewResolver"* class=*" org.springframework. web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/jsp/"/*>

<property name=*"suffix"* value=*".jsp"/*>

</bean>

**文件上传的配置：**

<!-- 文件上传的配置信息 -->

<bean id=*"multipartResolver"* class=*"org.springframework.*

*web.multipart.commons.CommonsMultipartResolver"* >

<!-- 默认编码 (ISO-8859-1) -->

<property name=*"****defaultEncoding****"* value=*"gbk"*/>

<!-- 最大内存大小 (10240)-->

<property name=*"****maxInMemorySize****"* value=*"10240"*/>

<!-- 上传后的目录名 -->

<property name=*"****uploadTempDir****"* value=*"/upload/"*/>

<property name=*"****maxUploadSize****"* value=*"-1"*/> <!-- 最大文件大小，-1为无限止(-1) -->

</bean>

**上传文件的表单：**

请求方式为post, enctype=”multipart/form-data”,上传文件为file类型

<form action=*"upload.do"* method=*"post"* enctype=*"multipart/form-data"*>

上传文件：<input **type=*"file"*** name=*"myFile"*/><br><br>

<input type=*"submit"* value=*"上传"*/>

</form>

**文件上传的类：**

@Controller

**public** **class** UploadController **implements** ServletContextAware{

//注入application内置对象，用来拿到上传的文件存储的路径

**private** ServletContext servletContext;

**public** **void** setServletContext(ServletContext servletContext) {

**this**.servletContext=servletContext;

}

@RequestMapping("/upload.do")

**public** String upload(@RequestParam("myFile") CommonsMultipartFile file,HttpServletRequest request){

//拿到上传的文件存储的路径

String path=**this**.servletContext.getRealPath("/upload/");

//拿到上传文件的文件名

String fileName=file.getOriginalFilename();

request.setAttribute("filename",fileName);

//创建新的文件夹并且指定文件路径和文件名

File file1=**new** File(path,fileName);

**try**{

**byte**[] b=file.getBytes();//将需要上传的文件存到字节数组对象

//创建文件输出流对象

FileOutputStream fos=**new** FileOutputStream(file1);

fos.write(b);//将字节数组对象中的数据写入缓存中

fos.flush();//将缓存中数据写入文件

fos.close();//关闭文件输出流

}**catch**(Exception e){

**throw** **new** RuntimeException("文件上传错误");

}

**return** "upload";

}

}

**上传流程： ----注解方式**

1、Tomcat一启动,创建DispatcherServlet对象

2、执行init()方法，创建web工厂，

3、在web工厂中会创建以下几个对象

创建**视图定位器对象**

创建**上传处理器对象**,先看有没有上传文件目录，没有的话就在项目的根下创建目录。

创建**分派器对象**（映射器），到指定包下扫描后端处理器（@Controller标识的类）创建该类对象。

一创建**后端处理器对象**就把application内置对象注入。

@Controller

public class FileUploadController implements ServletContextAware{

private ServletContext servletContext;//注入appllication对象

public void setServletContext(ServletContext context) {

this.servletContext = context;

}

}

4、发送请求：/login.do,交给tomcat处理

5、tomcat创建request内置对象接收请求

6、将请求交给DispatcherServlet 处理，截取请求

7、到web工厂中找到分派器

8、分派器找到请求对应的后端控制器

9、传递参数：第一个参数String name,直接传递

第二个参数 表单中参数是file类型（传入的是字节数组对象）参数名myFile,方法中CommonsMultipartFile file, 形参的名为 file,类型为 CommonsMultipartFile.

自动创建CommonsMultipartFile类型的对象来封装表单中传过来的字节数组文件。

上传方法：

@RequestMapping(value="/upload.do")

public String handleUploadData(String name, @RequestParam("myFile") CommonsMultipartFile file){

**上传处理：**

String path=this.getServletContext().getRealPath(“/upload/”);//通过application找到上传文件夹的绝对路径

String fileName=file.getOringinalName();//得到原来的文件名

String fileType=fileName.substring(fileName.lastIndexOf(.)+1);//使用substring()截取得到上传文件的后缀名

**下载处理：**

File file1 = new File(path,fileName); //新建一个文件

byte[] b=file.getBytes();//将上传的文件存入字节数组对象中

FileOutputStream fos = new FileOutputStream(file1);//创建文件字节输出流对象

fos.write(b);//将文件写入文件字节输出流对象所指的文件中

fos.flush();

fos.close();

return “redirect:upload\_ok.jsp”;//重定向，不需要视图定位器。

# SpringMVC的异常处理

主要的类：**SimpleMappingExceptionResolver**

接口：**AbstractHandlerExceptionResolver**

Exception异常对象的数据结构:

有个私有属性private String message,

只能通过有参的构造方法设值，只提供了公有的getMessage(),

e.getMessage();显示异常信息

e.toString():显示异常类型和异常信息

e.pringStackTrace():调用toString(),显示出异常的类,方法。

SpringMVC的异常类：

**异常类的定义：**继承RutimeException,复写有参的构造方法

**public** **class** **UserException** **extends** RuntimeException {

**public** UserException(String msg){

**super**(msg);

}

}

**SpringMVC的主配置文件： springmvc-servlet.xml**

<!-- 配置启用springmvc的注解 -->

<context:annotation-config/>

<!-- 配置注解扫描的包 -->

<context:component-scan base-package=*"cn.springmvc"/*>

<!-- 配置视图解析器 -->

<bean id=*"viewResolver"* class=*" org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/jsp/"*></property>

<property name=*"suffix"* value=*".jsp"*></property>

</bean>

**异常信息的配置：**

<!-- 配置异常处理信息 -->

<bean id=*"exceptionResolver"*class=*"org.springframework.web.servlet.handler.SimpleMappingExceptionResolver"*>

<property name=*"exceptionMappings"*>

<props>

<prop key=*"cn.springmvc.exception.UserException"*>

UserExceptionPage</prop>

</props>

<!-- 配置默认异常处理转向 -->

</property>

<property name=*"defaultErrorView"*>

<value>error</value>

</property>

</bean>

在主配置文件中引入异常处理类SimpleMappingExceptionReslover

**key:**是异常类型（自定义异常类路径名）

**value:**异常转向页面，通过视图解析器自动加上前缀和后缀，然后完成跳转。（相当于 struts1的input属性）

配置在exceptionMappings外的异常处理信息，表示找不到指定的key时，会调用这个默认的key（*defaultErrorView*）所指定的转向页面value完成转向。

**异常抛出类**（业务类直接抛出异常，后端控制器类也直接抛出异常，最后交给SpingMVC处理的中央处理器处DispatcherServlet）**：**

**public** **class** UserImpl **implements** UserManager{

**public** String login(User user) {

**if**("DFX".equals(user.getUsername())){

**return** "main";

}

**//通过有参的构造方法抛出异常，参数为异常信息值，存到message属性中**

**throw** **new** UserException("用户名不对啊"+user.getUsername());

}

}

所有从jsp页面中取出的时候用 **${exception.message}**

**异常处理流程：**

1、Tomcat一启动,创建DispatcherServlet对象

2、执行init()方法，创建web工厂，

3、在web工厂中会创建以下几个对象

创建分派器对象

创建后端处理器对象

创建视图定位器对象

创建异常处理器对象

4、发送请求：/login.do,交给tomcat处理

5、tomcat创建request内置对象接收请求

6、将请求交给DispatcherServlet 处理，截取请求

7、到web工厂中找到分派器

8、分派器到mappings的prop中找到id对应的后端控制器

9、执行后端控制器的handlerRequest()方法

10、在方法中调用业务层方法做验证，不合法就抛出异常

**在业务层：throw new UserNotFoundException(username);**

11、后端控制器也不处理异常，继续抛出异常

12、最后把异常交给DispatcherServlet处理

异常处理DispatcherServlet：

1、拿到异常对象，设到request内置对象中，属性名为**exception**

2、通过getClass().getName()拿到异常对象的类型，到web工厂中找异常处理器对象，从异常处理器中找到和异常类型相对应的异常value(异常转向页面)。

3、异常转向找到视图定位器加上前缀和后缀，转到相应的jsp页面。

4、在异常转向jsp页面中：${exception.message}

**Throw new Exception(“haha”+password);//抛出异常信息**

拿到异常对象，设到request内置对象中，属性名为exception,属性值为异常值（message）

拿到异常对象的类型，到web工厂中找异常处理器，从异常处理器中的prop找到和异常类型相对的异常value(异常转向页面)。发现没有

就找到异常处理器默认的配置异常页面，直接到property配置中 name=”defaultErrorView”找到value(异常转向)

异常转向找到视图定位器加上前缀和后缀，转到相应的jsp页面。

在异常转向jsp页面中：${exception.message}

银行项目和物料项目使用springmvc+spring+hibernate继承

**基于注解Spring3实现：**

1. 建立web项目
2. 导入Spring3的jar包和hiberante的jar 包
3. 在WEB-INF/下配置web.xml,applicationContext.xml（配置BeanFactory，和springmvc集成）,spring-servlet.xml(注解方式：**需要配置启动注解，扫描的包**，**请求分派器**，**视图解析器**，异常处理器，拦截器，国际化等等)
4. 在src下配置hiberante.cfg.xml:指定使用数据库的信息，基本的属性是否显示sql语句，使用方言，自动创建表等等，还有配置映射文件
5. 建立项目的包结构已经相关联（视图层，模型层，控制层）

**集成环境：**

Spring3.0,Hibernate3.X

在web.xml中配置表示层SpringMVC-----DispatcherServlet。

配置webFactory:spring-servlet.xml

将annotation.jar放下项目下才能使用注解集成：该包位于tomcat/lib（还有许多不知名的包需要用的）

模型层：（hibernate）

直接对数据库进行存储和加载

持久层由hibernate完成

使用<mapping resources=””>标签将hbm.xml文件配置到hibernate.cfg.xml主配置文件中

**applicationContext.xml中的配置：**

1. Spring替Hibernate生成sessionFactory工厂

作用：生成全局性事务，创建HibernateTemplate对象

2、创建事务管理器

作用：按照事务的传播特性决定某一个方法如何使用事务

3、事务传播特性

作用：决定相应的方法如何使用事务

4、配置AOP（声明式事务）

作用：指定事务边界（一般为业务层）

**AOP：**面向切面编程，需要增加新的功能的时候不改变源代码直接增加。

**AOP主要技术：**是为实现接口的类做动态代理

**动态代理：**JDK动态代理需要生成代理对象，需要实现InvocationHandler接口，通过newProxyInstance()

**目标对象和代理对象：**实现了同一个接口

调用代理对象的方法，会调用this所指的invoke方法，织入事物。

**全局性事务：**

sessionFactory生成的是全局性事务，归所有的方法公有。

使用全局性事务的特点：使得访问数据库由**并行变串行，安全性好**

数据库只能处理串行的数据，不能接受并行访问

在spring的配置文件中直接注入hibernateTemplate对象，通过注入sessionFactory<bean id=” hibernateTemplate”class=”…”>

在业务层直接hibernateTemplate使用：

@Resource(name=”hibernateTemplate”)

private HibernateTemplate hibernateTemplate;

如果注入的对象中的方法不符合pointcut方法，就不会创建代理对象，不织入advice,拿到的对象是目标对象。

**Tomcat启动：**

读取web.xml文件，创建监听器对象，创建application内置对象，把全局初始化参数设到application中。

监听器调用contextInitialized()方法中application中拿到spring的主配置信息创建BeanFactory，存到application内置对象中

**一创建BeanFactory做以下这些事情：**

1. 先通过读取hibernate的主配置文件hiberante.cfg.xml文件，创建了SessionFactory

Configuration cfg=new Configuration().configure();

1. Spring替Hibernate管理事务，创建sessionFactory的工厂的作用？

* 根据sessionFactory**创建hiberanteTemplate对象（session），使用的时候可以直接注入。**
* 通过sessionFactory生成事务管理器，**生成全局性事务，**让hibernate的每一个session可以共享事务

1. Spring为什么要用HibernateTemplate?

Spring轻量级的封装，使得**hibernateTemplate线程安全**

**Callback技术**，把公共的东西都提取出来。

**创建DispatcherServlet对象**

执行init()方法，读取配置文件

创建WebFactory对象。

创建视图定位器

作用：给view加上前缀和后缀，找到指定页面

创建请求分派器：

请求分派器去找请求对应的后端处理器类，马上创建后端处理器对象（初始化的时候创建）

一创建后端处理器对象就会注入业务层对象:

@Resource（name=”loginService”）

private LoginService loginService;

到beanFactory中找到id为注解中name标识的属性

**发送请求：**

1、请求交给tomcat.Tomcat创建request内置对象接收请求信息

2、发现是.do请求，就交给springmvc。

3、Springmvc截取请求，将请求交给请求分派器

4、请求分派器根据请求找到对象的后端处理器对应的方法

5、收集表单数据（创建user对象，自动将表单中的数据收集到属性中，将user对象以属性名user的形式设到request中）

6、调用业务对象的方法（业务对象为代理对象）

7、调用代理对象的invoke()方法，织入advice

8、返回ModelAndView对象，找到视图定位器，转到对应的页面。

**运行时异常：出错也回滚**

编译异常：出错但是不回滚

事务边界设到业务层：因为持久层的底层数据库可能不一样

创建持久层对象，在持久层类通过注解中注入hiberanteTemplate

创建业务层对象，在业务层对象类中通过注解注入持久层对象

**为什么事务的边界需要设到业务层？**

出现这个问题的原因：DAO层没有事务hibernate是无法对数据库进行操作的，所以最需要事务的是DAO层

分析原因：以业务的角度去考虑，一个业务对应的是一系列的数据库的操作，事务的管理设置在业务层的话，其中那个操作出现问题了就这业务一起回滚是比较合理的。

假如设置到DAO层的话，每个操作进行一次事物的开启和关闭

一、为什么把事务边界设置在业务层：

1、因为持久层比较庞大，把事务边界设置在持久层不利于业务层管理，而设置到业务层便于管理

一家公司有贷款业务，和十几家银行有联系，公司有一个总经理，总经理下有业务经理，业务经理手下有多个业务员，业务员负责与银行联系。如果业务员过多，增加总经理，如果银行少，减少业务员和业务经理。

在项目中，银行就是数据库，业务员就是持久层，业务经理是业务层，合理的分层开发有利于项目的运行。

二、如何做到：利用声明式事务

1、通过session工厂创建事务管理器

2、设置事务传播特性决定哪些方法如何使用事务，让多个方法共享同一个全局事务，并行改串行。

3、通过AOP决定在哪些类的哪些方法使用事务传播特性。

**集成的第二种方法：Spring替hibernate配置hbm**

不需要配置hibernate.cfg.xml文件

也不需要配置User.hbm.xml文件

配置数据源（驱动，url用户名，密码）

配置基本属性（生成表，显示sql语句）

配置hbm文件

1. 数据源一般采用 dbcp连接池

第三方连接池：c3p0

把数据源注入Annotation

配置一些数据库的使用特性

配置映射文件：

1. 使用hibernate的映射文件 User.hbm.xml
2. 完成使用是spring使用注解

<property>

<list>

<value>cn.springmvc.bean.User</value>

</list>

</property>