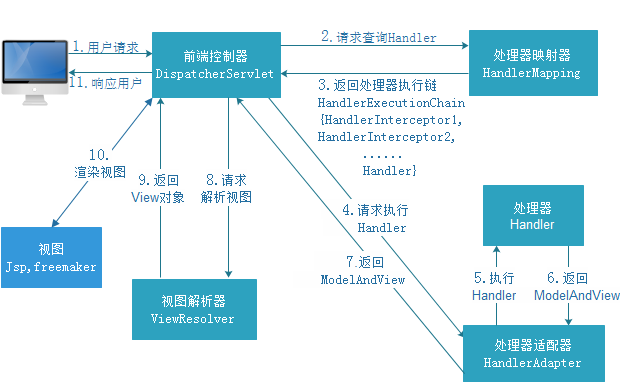
## Springmvc的整体框架



客户端请求到mvc的前端控制器(DispatcherServlet),然后由前端控制器去请求处理器映射器,处理器映射器会根据请求,返回需要调用的controller的包名方法名,然后前端控制器根据映射器返回的信息去找处理器适配器,适配器找到对应的controller中的方法,即处理器来处理数据,和视图,将处理后的结果返回到处理器适配器,再由处理器适配器返回到前端控制器,然后前端控制器根据返回的数据和视图名称交由视图解析器来处理数据获取页面,最后渲染视图.

## Springmvc的搭建

创建springmvc的配置文件springmvc.xml

### 开启组件扫描:

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

### 在web.xml中添加springmvc的servlet前端控制器

<servlet>

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

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

</servlet-class>

<!-- 指定SpringMVC配置文件 -->

<!-- SpringMVC的配置文件的默认路径是/WEB-INF/${servlet-name}-servlet.xml --> // param 参数

<init-param>

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

<param-value>classpath:springmvc.xml</param-value>

</init-param>

</servlet>

<servlet-mapping> // servlet 小程序

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

<!-- 设置所有以action结尾的请求进入SpringMVC -->

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

</servlet-mapping>

### 创建controoler层

需要打注解标签@controller

方法上需要添加 @RequestMapping("/itemList.action")

### 定义处理器映射器

<!-- 配置处理器映射器 -->

<bean

class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping" />

作用: 注解式处理器映射器，对类中标记了@ResquestMapping的方法进行映射。根据@ResquestMapping定义的url匹配@ResquestMapping标记的方法，匹配成功返回HandlerMethod对象给前端控制器。

### 配置处理器适配器

<!-- 配置处理器适配器 -->

<bean

class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter" />

作用: 注解式处理器适配器，对标记@ResquestMapping的方法进行适配。

### 注解驱动

<!-- 注解驱动 -->

<mvc:annotation-driven />

### 视图解析器

<bean

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<!-- 配置逻辑视图的前缀 -->

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

<!-- 配置逻辑视图的后缀 -->

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

</bean>

## @RequestParam

用于处理简单类型的参数绑定,当提交的参数name和接受的name不一致的时候使用该参数. Value是传入的参数的name, required是表示请求的参数中时候必须要有该参数,默认为必须, defaultValue是默认值

public String queryItemById(@RequestParam(value = "itemId", required = true, defaultValue = "1") Integer id, ModelMap modelMap)

## 解决post乱码问题

<filter>

<filter-name>encoding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<!-- 设置编码参是UTF8 -->

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encoding</filter-name>

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

</filter-mapping>

设置编码过滤器.

## Get提交乱码

<Connector URIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>

在webxml中添加编码配置

## Springmvc的转换器

Springmvc不能把字符串转换成时间类型,需要在处理器适配器上定义转换器,进行参数的绑定.

### 自定义转换类

需要实现Converter接口

public class DateConverter implements Converter<String, Date> {

@Override

public Date convert(String source) {

try {

// 把字符串转换为日期类型

SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyy-MM-dd HH:mm:ss");

Date date = simpleDateFormat.parse(source);

return date;

} catch (ParseException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

// 如果转换异常则返回空

return null;

}

}

这里不会随便转换,是根据绑定的形参来判断的,即Converter<String, Date>

### Springmvc的配置

<!-- 配置注解驱动 -->

<!-- 如果配置此标签,可以不用配置... -->

<mvc:annotation-driven conversion-service="conversionService" />

<!-- 转换器配置 -->

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

<property name="converters">

<set>

<bean class="cn.itcast.springmvc.converter.DateConverter" />

</set>

</property>

</bean>

## Struts2与springmvc的区别

1入口不一致:springmvc入口是前端控制器servlet,而struts2是filter过滤器.

2 springmvc是基于方法的,请求参数是到形参,可以设计成单利和多例,而struts2是基于类开发,传递参数是通过类的属性.只能设计成多例.

3 strust2是通过值栈用ognl表达式来存取请求和响应的数据,springmvc是通过modelandview来封装数据和视图,通过request的域来传递到页面.

## 请求方法的限定

限定GET方法

@RequestMapping(method = RequestMethod.GET)

如果通过POST访问则报错：

HTTP Status 405 - Request method 'POST' not supported

例如：

@RequestMapping(value = "itemList",method = RequestMethod.POST)

 限定POST方法

@RequestMapping(method = RequestMethod.POST)

如果通过GET访问则报错：

HTTP Status 405 - Request method 'GET' not supported

GET和POST都可以

@RequestMapping(method = {RequestMethod.GET,RequestMethod.POST})

## Controller方法返回值

第一种: 返回modelandview,即封装数据也返回视图.

第二种: 返回void,通过request和response来返回结果.

第三种: 返回string,返回视图名称,数据用model或是modelmap来封装.

在返回字符串的时候可用在字符串中添加redirect或是forward进行请求转发或是重定向.

return "redirect:test/index.action";

## springmvc的异常处理

编写异常处理类,需要实现HandlerExceptionResolver接口

public class CustomHandleException implements HandlerExceptionResolver{

public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler,

Exception exception) {

String msg;

if(exception instanceof CustomException){

msg = exception.getMessage();

}

else{

msg="哈哈,你走错了";

}

ModelAndView modelAndView = new ModelAndView();

modelAndView.addObject("msg",msg);

modelAndView.setViewName("error");

return modelAndView;

}

}

springmvc.xml的配置文件,需要实例化这个类就可以了.

<bean class="com.itheima.springmvc.exception.CustomExceptionResolver"></bean>

## 上传图片

在配置文件springmvc.xml中:

<!-- 上传图片的实现类 -->id必须是这个名字

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

Form头需要添加enctype

<form id="itemForm" action="${pageContext.request.contextPath }/updateitem.action" enctype="multipart/form-data" method="post">

Controller中需要添加对应的形参

@RequestMapping(value = "/updateitem.action")

public String updateitem(Item item,MultipartFile pictureFile){

//修改 通过Id对商品信息进行修改

itemService.updateItemById(item);

ModelAndView mav = new ModelAndView();

mav.setViewName("success");

return "redirect:test/index.action";

}

## 拦截器

在springmvc.xml配置拦截器

<mvc:interceptor>

<!-- 配置商品被拦截器拦截 -->

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

<!-- 配置具体的拦截器 -->

<bean class="cn.itcast.ssm.interceptor.LoginHandlerInterceptor" />

</mvc:interceptor>

类的配置:实现这个接口HandlerInterceptor

public class Interceptor2 implements HandlerInterceptor{

public boolean preHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2) throws Exception {

// TODO Auto-generated method stub

System.out.println("Interceptor2:方法前");

return false;

}

public void postHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, ModelAndView arg3)

throws Exception {

// TODO Auto-generated method stub

System.out.println("Interceptor2:方法后");

}

public void afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3)

throws Exception {

// TODO Auto-generated method stub

System.out.println("Interceptor2:页面渲染后");

}

}