# SpringMVC概要

## Spring MVC的总控：DispatcherServlet

web.xml中的声明：

<context-param>

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

<param-value>classpath:spring/app-\*.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<servlet>

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

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

<init-param>

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

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

</init-param>

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

</servlet>

<servlet-mapping>

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

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

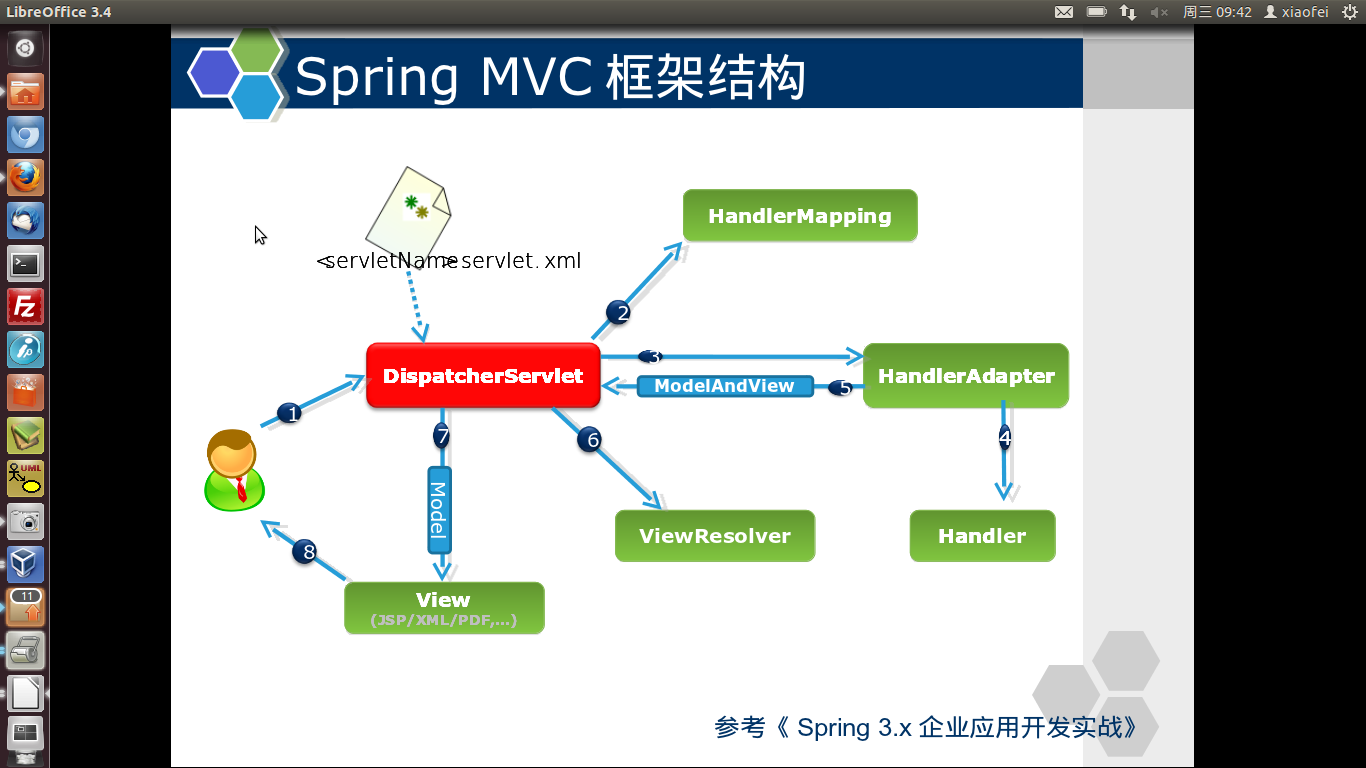
</servlet-mapping>

1、ContextLoaderListener创建root webApplicationContext

2、DispatcherServlet创建servlet-specific 的webApplicationContext，继承并覆盖root的定义。init-param可选，缺省查找/WEB-INF/{servletName}-servlet.xml。示例中该文件放在classpath中便于使用maven resources插件进行环境变量替换。

3、DispatcherServlet会默认初始化一些特定类型的bean，见/org/springframework/web/servlet/DispatcherServlet.properties中的说明。

4、这些特定类型的Bean，可在\*-servlet.xml中声明，覆盖默认定义，或者声明多个不同实现，spring根据他们的order属性来做优先级处理。这些类构建了spring mvc的整个处理流程的主体。见下图（图片来源《Spring3.x企业应用开发实战》）



{servletName}-servlet.xml配置, 初始化默认的框架级别的bean。

<mvc:annotation-driven />

spring 3.1 mvc默认实现变化：

HandlerMapping和HandlerAdapter ，spring3.1的这两个接口的默认实现：

* **org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping**

Creates RequestMappingInfo instances from type and method-level @RequestMapping annotations in @Controller classes.

* **org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter**

An AbstractHandlerMethodAdapter that supports HandlerMethods with the signature -- method argument and return types, defined in @RequestMapping.

多了几个接口：HandlerMethodArgumentResolver/HandlerMethodReturnValueHandler，因此MVC XML Namespace下的配置有些变化，可扩展点更多。暂时只需了解即可。

<beans xmlns=*"http://www.springframework.org/schema/beans"*

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

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:util=*"http://www.springframework.org/schema/util"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.1.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.1.xsd*

*"*>

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

<mvc:argument-resolvers>

<bean/>

<bean/>

</mvc:argument-resolvers>

<mvc:message-converters register-defaults=*"true"*>

<bean/>

</mvc:message-converters>

<mvc:return-value-handlers>

<bean/>

</mvc:return-value-handlers>

</mvc:annotation-driven>

## Restful MVC

### URL规划

|  |  |  |  |
| --- | --- | --- | --- |
| URL | HTTP方法 | MIME | 含义 |
| /return/order | GET | text/html | 退货单首页 |
| /return/order/new | GET | text/html | 显示新增页 |
| /return/order | POST |  | 创建退货单 |
| /return/order/{id}/edit | GET | text/html | 显示修改页 |
| /return/order/{id} | PUT |  | 修改退货单 |
| /return/order/{id} | GET | text/html | 显示退货单 |
| /return/order/{id} | DELETE |  | 删除退货单 |
| /return/order/query | GET | text/html | 查询结果 |
| /return/order/query | GET | application/json | 查询结果 |

### HTTP方法，表达了针对资源的动作

* GET：检索资源。具有幂等性，可以充分使用客户端的缓存。
* POST：创建一个新资源，不具有幂等性（多次操作会产生多个新资源）。建议使用Post/Redirect/Get模式。
* PUT：更新资源数据，如果资源不存在的话，则可根据此URI创建一个新的资源。具有幂等性。
* DELETE，删除一个资源。具有幂等性，可以多次操作它，直到得出结果，删除不存在的东西没有任何问题。
* HEAD/OPTIONS，略。

ajax框架可指定http方法，浏览器一般只能GET/POST，spring通过一个表单隐藏域来模拟其他方法。

<form action="#springUrl('')/return/order/$id" method="POST">

...

<input name="\_method" type="hidden" value="PUT" />

<input type="submit" value="保存修改"/>

<input type="button" value="删除" onclick="javascript:this.form['\_method'].value='DELETE';this.form.submit();" />

</ins>

</form>

web.xml

<filter>

<filter-name>HiddenHttpMethodFilter</filter-name>

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

</filter>

<filter-mapping>

<filter-name>HiddenHttpMethodFilter</filter-name>

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

</filter-mapping>

# Controller

## 使用注解编写Restful Controller：

@Controller

@RequestMapping("/return/order")

**public** **class** ReturnOrderController {

@Autowired

ReturnOrderService service;

@RequestMapping(value = "/{id}/edit", method = RequestMethod.*GET*)

**public** String edit(@PathVariable("id") Long id, Model view) {

// load from backend by id

ReturnOrder order = service.getById(id);

// add to view model to render edit page

view.addAttribute("order", order);

**return** "return/order-edit";

}

...

}

spring扫描并管理：

<context:component-scan base-package=*"com.jd.bluedragon.web.controller"* />

## controller类声明：

1、@Controller注解bean，由spring ioc管理，默认为单例。其value属性可指定bean name，默认为首字母小写的类名。

2、@RequestMapping("/return/order")，将http请求限定到类，映射资源url的起始路径。

## controller类方法签名：

1、@RequestMapping注解，将http请求限定到类方法：映射从起始路径开始的url路径，http方法, http头、请求参数等等。

2、方法名，无特殊规定，建议有意义的命名。

3、允许的方法参数，详尽@RequestMapping的API文档，稍后举例。

4、允许的返回值，详见@RequestMapping的API文档，稍后举例。

5、异常，无特殊规定。

### controller类方法签名：@RequestMapping的作用

mapping and narrow request to method.

1. **value**：url限定，比如方法上的/{id}/edit和/new**。**
2. **method**：http方法限定。GET/POST/PUT/DELETE等。当url一样时，通过http方法来区分。

**@RequestMapping(value = "/{id}", method = RequestMethod.*PUT*)**

**public** String update(...

@RequestMapping(value = "/{id}", method = RequestMethod.*GET*)

**public** String show(...

1. **consumes或produces**：根据请求或响应的ContentType来区分。当url和http方法一样，以此来区分。

**@RequestMapping(value = "/query", method = RequestMethod.*GET*,**

produces = "text/html")

**public** String query(...

@RequestMapping(value = "/query", method = RequestMethod.*GET*,

produces = "application/json")

@ResponseBody

**public** Model queryJson(...

1. params/header限定请参考API。

### controller类方法签名：允许的方法参数类型

1. spring替你准备好这些参数，不用管抽取、类型转换等细节。
2. @PathVariable注解的参数，抽取url占位符代表的值

**@RequestMapping(value = "/{id}/edit", method = RequestMethod.*GET*)**

**public** String edit(@PathVariable("id") Long id, Model view) {

1. Model/Map等视图模型，被spring自动注入，如上例。将它暴露到视图层。

**public String edit(@PathVariable("id") Long id, Model view) {**

// add to view model to render edit page

view.addAttribute("order", order);

**return** "return/order-edit";

1. @RequestParam注解的参数，抽取http request中相应参数的值。

**public Model queryJson(**

@RequestParam(value = "page", required = **false**, defaultValue = "1") **int** page,

@RequestParam(value = "rows", required = **false**, defaultValue = "20") **int** pageSize**,**

**…**

1. 表单绑定对象(command/form object)/输入校验结果对象(BindingResult/Errors)

**public** String create(@Valid @ModelAttribute("order") ReturnOrderDTO command,

BindingResult bindingResult) {

**if** (bindingResult.hasErrors()) {

**return** "return/order-new";

}

**ReturnOrder order = new ReturnOrder();**

order.setOrg(command.getOrg());

1. HttpServletRequest/HttpServletResponse等，如：

**@RequestMapping(value = "/index", method = RequestMethod.*GET*)**

**public** String index(

@RequestParam(value = "locale", required = **false**) Locale locale,

HttpServletRequest request,

HttpServletResponse response)

1. 其他可使用的参数类型见@RequestParam类API文档。
2. spring3.1的 HandlerMethodArgumentResolver扩展，应该可以允许更多的自定义类型参数。

### controller类方法签名：允许的返回值类型

1. String，表示视图的逻辑名，例如下例将使用/WEB-INF/view/return/order-edit.vm为模板视图：

**public** String edit(@PathVariable("id") Long id, Model view) {

...

view.addAttribute("order", order);

**return** "return/order-edit";

}

又如，下例将重定向到/return/order

**public String update( …**) {

// redirect to index

**return** "redirect:/return/order";

**}**

1. ModelAndView，比如：

**@ExceptionHandler**

**public** ModelAndView notfound(NoSuchEntityException exception) {

**return** **new** ModelAndView("return/order-notfound")

.addObject("exception", exception);

}

1. @ResponseBody 注解的，返回值将被序列化，比如下例将返回值转换成json：

**@ResponseBody**

**public** Model queryJson(

@RequestParam(value = "page", required = **false**, defaultValue = "1") **int** page,

@RequestParam(value = "rows", required = **false**, defaultValue = "20") **int** pageSize,

Model view) {

PagedList<ReturnOrder> orders = service

.findReturnOrders(page, pageSize);

view.addAttribute("rows", orders);

view.addAttribute("total", orders.getItemCount());

**return** view;

// return new PagesBar<ReturnOrder>(orders, 7);

}

1. void，由应用自己处理响应，比如只返回http状态：

**public** **void** create(..., HttpServletResponse response){

...

response.setStatus(201);

}

1. 其他的返回值类型可参考@RequestParam类API文档。
2. spring3.1的 HandlerMethodReturnValueHandler 扩展，应该可以允许更多的自定义返回值类型。

# 表单对象绑定/输入验证/自定义转换器

假设一个场景，输入错误后返回输入页面

**public** String create(@Valid @ModelAttribute("order") ReturnOrderDTO command,

BindingResult bindingResult) {

**if** (bindingResult.hasErrors()) {

**return** "return/order-new";

}

…

**}**

**/\*\***

\* DTO (or VO, or else), demonstrate JSR 303 validation in spring mvc.

\*/

**public** **static** **class** ReturnOrderDTO {

@Min(1)

**private** **int** orderQty;

//getter/setter

**<form id="orderform" action="#springUrl('')/return/order" method="POST">**

<p>

<label for="orderQty">订单数量</label>

#springFormInput('order.orderQty', 'class="easyui-validatebox" required="true"') #springShowErrors('<br/>','')

</p>

1. 输出：#springFormInput是spring提供的velocity宏，”order.orderQty”表示视图模型中有一个名称为order的对象，orderQty为其属性。
2. 输入：参数@ModelAttribute("order") ReturnOrderDTO command接收输入，当返回时，自动将其内容暴露给视图模板。
3. 服务端验证：在表单绑定对象加上@Valid注解，即可自动验证表单对象属性设置的各种JSR303约束规则。
4. 客户端验证：js框架,class="easyui-validatebox" required="true"。
5. 自定义转换器，详见demo代码。

# 视图层（以velocity为例）

## 默认视图处理器

**<!-- template view -->**

<bean id=*"velocityConfig"*

class=*"org.springframework.web.servlet.view.velocity.VelocityConfigurer"*>

<property name=*"resourceLoaderPath"* value=*"/WEB-INF/views/"* />

<property name=*"velocityProperties"*>

<props>

<prop key=*"input.encoding"*>UTF-8</prop>

<prop key=*"output.encoding"*>UTF-8</prop>

<prop key=*"contentType"*>text/html;charset=UTF-8</prop>

<prop key=*"velocimacro.library"*>macro.vm</prop>

</props>

</property>

</bean>

<bean id=*"velocityViewResolver"* class=*"org.springframework.web.servlet.view.velocity.VelocityLayoutViewResolver"*>

<property name=*"layoutUrl"* value=*"layout/default.vm"* />

<property name=*"cache"* value=*"false"* />

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

<property name=*"exposeSpringMacroHelpers"* value=*"true"* />

<property name=*"dateToolAttribute"* value=*"dateTool"* />

<property name=*"numberToolAttribute"* value=*"numberTool"* />

<!-- Merge urlBuilderMap to view context for convenience. You can put your tools which must be thread safe. -->

<property name=*"attributesMap"* ref=*"\_urlBuilderMap"* />

<property name=*"contentType"* value=*"text/html;charset=UTF-8"* />

</bean>

1. 模板中使用spring宏:

见上面的例子

1. 使用自定义的线程安全的对象方法：

**#set($pageUrl = $homeModule.forPath('/return/order/query').put({"id":$!id}))**

1. 布局

**见/WEB-INF/views/layout/default.vm**

## 内容协商视图：{servletName}-servlet.xml配置

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

<property name=*"defaultContentType"* value=*"text/html"* />

<!-- not by accept header -->

<property name=*"ignoreAcceptHeader"* value=*"true"*/>

<!-- by extension -->

<property name=*"mediaTypes"*>

<map>

<entry key=*"xml"* value=*"application/xml"* />

<entry key=*"json"* value=*"application/json"* />

</map>

</property>

<property name=*"viewResolvers"*>

<list>

<ref bean=*"velocityViewResolver"*/>

</list>

</property>

<property name=*"defaultViews"*>

<list>

<!-- for application/json -->

<bean class=*"org.springframework.web.servlet.view.json.MappingJacksonJsonView"* />

</list>

</property>

</bean>

* 以扩展名来获取json格式的资源：

return/order/query.json

* 还可以特定参数、accept 头的设定来制定同一资源的不同表示。

# 本地化

## {servletName}-servlet.xml配置:

**<!-- locale related -->**

<bean id=*"localeResolver"* class=*"org.springframework.web.servlet.i18n.CookieLocaleResolver"*>

<property name=*"cookieName"* value=*"\_clientlocale"*/>

<property name=*"defaultLocale"* value=*"zh\_CN"*/>

<property name=*"cookieMaxAge"* value=*"2147483647"*/>

</bean>

**<!-- Access resource bundles with the specified basename -->**

<bean id=*"messageSource"*

class=*"org.springframework.context.support.ReloadableResourceBundleMessageSource"*>

<property name=*"basenames"* value=*"/WEB-INF/i18n/content,/WEB-INF/i18n/validation,/WEB-INF/i18n/asset,/WEB-INF/i18n/enum"* />

<property name=*"cacheSeconds"* value=*"5"* />

</bean>

## 配置一个localeResolver和相关的本地化资源文件，资源文件不同用途使用不同的命名规则。

## 一个允许用户更改并存储locale的入口：

**public class IndexController {**

@Autowired

LocaleResolver localeResolver;

@RequestMapping(value = "/index", method = RequestMethod.*GET*)

**public** String index(

@RequestParam(value = "locale", required = **false**) Locale locale,

HttpServletRequest request,

HttpServletResponse response) {

**if** (locale != **null**) {

localeResolver.setLocale(request, response, locale);

}

**…**

## 页面上显示本地化信息，使用spring宏：

1. **内容：#springMessage('return.order.col.id')**
2. **验证错误：#springShowErrors， *validation.properties中，key以typeMismatch.开头。***

# 异常映射

## {servletName}-servlet.xml配置全局的:

**<!-- exception related -->**

<!-- all exception handler will be registered by DispatcherServlet default (detectAllHandlerExceptionResolvers=true) -->

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

<property name=*"defaultErrorView"* value=*"error/uncaught"*/>

<property name=*"exceptionMappings"*>

<props>

<prop key=*"com.jd.common.hrm.IllegalHrmPrivilegeException"*>error/access-denied</prop>

</props>

</property>

</bean>

## 注解，定义controller特定的异常处理

**@ExceptionHandler**

**public** ModelAndView notfound(NoSuchEntityException exception) {

**return** **new** ModelAndView("return/order-notfound")

.addObject("exception", exception);

}

# 拦截器，以京东SSO为例说明

## 静态资源映射，不使用拦截器：{servletName}-servlet.xml

**<mvc:default-servlet-handler />**

<!-- static resources -->

<mvc:resources location=*"/static/"* mapping=*"/static/\*\*"*

cache-period=*"864000"*/><!-- 24 \* 3600 \* 10 -->

使用servlet容器默认的servlet，而不使用spring的DispatcherServlet来处理，快速，还可加缓存时间。在生产环境中，有可能被前端的proxy给拦截掉。

## spring mvc的拦截器，类似于servlet的filter。

比filter好的地方是，interceptor可以被spring ioc容器管理。

## URL映射和拦截器配置:{servletName}-servlet.xml

**<!-- url mapping with interceptor -->**

<mvc:interceptors>

<mvc:interceptor>

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

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

<ref bean=*"\_DotnetTicketContextInterceptor"* />

</mvc:interceptor>

<mvc:interceptor>

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

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

<ref bean=*"\_HrmDotnetTicketLoginContextInterceptor"* />

</mvc:interceptor>

<mvc:interceptor>

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

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

<ref bean=*"\_LoginContextInterceptor"* />

</mvc:interceptor>

<mvc:interceptor>

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

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

<ref bean=*"\_DotnetTicketRequiredInterceptor"* />

</mvc:interceptor>

<mvc:interceptor>

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

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

<ref bean=*"\_HrmPrivilegeInterceptor"* />

</mvc:interceptor>

</mvc:interceptors>

## 权限相关：在controller方法上加上自定义的注解

**/\*\***

\* demonstrate **@HrmPrivilege** handling failed.

\*/

@HrmPrivilege("ceo,cto")

@RequestMapping(value = "/access-denied", method = RequestMethod.*GET*)

**public** **void** denied() {

// throw in interceptor

}

***讲解***

***com.jd.common.springmvc.interceptor.DotnetTicketRequiredInterceptor***

***com.jd.common.hrm.support.HrmPrivilegeInterceptor***