Spring Mvc

The Spring Web model-view-controller (MVC) framework is designed around a DispatcherServlet that dispatches requests to handlers, with configurable handler mappings, view resolution, locale, time zone and theme resolution as well as support for uploading files. The default handler is based on the @Controller and@RequestMapping annotations, offering a wide range of flexible handling methods. With the introduction of Spring 3.0, the @Controller mechanism also allows you to create RESTful Web sites and applications, through the @PathVariable annotation and other features.

## **The DispatcherServlet**



源码解读：

/\*\*

\* Initialize the strategy objects that this servlet uses.

\* <p>May be overridden in subclasses in order to initialize further strategy objects.

\*/

**protected** **void** initStrategies(ApplicationContext context) {

initMultipartResolver(context);

initLocaleResolver(context);

initThemeResolver(context);

initHandlerMappings(context);

initHandlerAdapters(context);

initHandlerExceptionResolvers(context);

initRequestToViewNameTranslator(context);

initViewResolvers(context);

initFlashMapManager(context);

}

非Xml方式配置

**public** **class** MyWebApplicationInitializer **implements** WebApplicationInitializer {

@Override

**public** **void** onStartup(ServletContext container) {

ServletRegistration.Dynamic registration = container.addServlet("example", **new** DispatcherServlet());

registration.setLoadOnStartup(1);

registration.addMapping("/example/\*");

}

}

XML方式

<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/config/mvc-config.xml</param-value>

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>dispatcherServlet</servlet-name>

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

</servlet-mapping>

Mvc的相关配置你可以选择放在mvc的config里面单独配置也可以通过委派给WebApplicationContext 配置，即放在单独的\*-servlet.xml 配置到\*-Application.xml中去。

### **Special Bean Types In the WebApplicationContext**

| **Bean type** | **Explanation** |
| --- | --- |
| [HandlerMapping](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-handlermapping" \o "18.4 Handler mappings) | Maps incoming requests to handlers and a list of pre- and post-processors (handler interceptors) based on some criteria the details of which vary by HandlerMapping implementation. The most popular implementation supports annotated controllers but other implementations exists as well. |
| HandlerAdapter | Helps the DispatcherServlet to invoke a handler mapped to a request regardless of the handler is actually invoked. For example, invoking an annotated controller requires resolving various annotations. Thus the main purpose of a HandlerAdapter is to shield the DispatcherServlet from such details. |
| [HandlerExceptionResolver](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-exceptionhandlers" \o "18.11 Handling exceptions) | Maps exceptions to views also allowing for more complex exception handling code. |
| [ViewResolver](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-viewresolver" \o "18.5 Resolving views) | Resolves logical String-based view names to actual View types. |
| [LocaleResolver](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-localeresolver" \o "18.8 Using locales) &[LocaleContextResolver](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-timezone" \o "18.8.1 Obtaining Time Zone Information) | Resolves the locale a client is using and possibly their time zone, in order to be able to offer internationalized views |
| [ThemeResolver](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-themeresolver" \o "18.9 Using themes) | Resolves themes your web application can use, for example, to offer personalized layouts |
| [MultipartResolver](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-multipart" \o "18.10 Spring’s multipart (file upload) support) | Parses multi-part requests for example to support processing file uploads from HTML forms. |
| [FlashMapManager](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-flash-attributes" \o "18.6 Using flash attributes) | Stores and retrieves the "input" and the "output" FlashMap that can be used to pass attributes from one request to another, usually across a redirect. |

### **@Controller**

<context:component-scan base-package="org.springframework.samples.petclinic.web"/>

#### **@Controller and AOP Proxying**

In some cases a controller may need to be decorated with an AOP proxy at runtime. One example is if you choose to have @Transactional annotations directly on the controller. When this is the case, for controllers specifically, we recommend using class-based proxying. This is typically the default choice with controllers. However if a controller must implement an interface that is not a Spring Context callback (e.g. InitializingBean, \*Aware, etc), you may need to explicitly configure class-based proxying. For example with <tx:annotation-driven/>, change to <tx:annotation-driven proxy-target-class="true"/>.

### **@RequestMapping**

**Since4.3**

* @GetMapping
* @PostMapping
* @PutMapping
* @DeleteMapping
* @PatchMapping

@PathVariable

#### **URI Template Patterns with Regular Expressions**

@RequestMapping("/spring-web/{symbolicName:[a-z-]+}-{version:\\d\\.\\d\\.\\d}{extension:\\.[a-z]+}")**public** **void** handle(@PathVariable String version, @PathVariable String extension) {

*// ...*

}

#### **Supported method argument types**

The following are the supported method arguments:

* Request or response objects (Servlet API). Choose any specific request or response type, for example ServletRequest or HttpServletRequest.
* Session object (Servlet API) of type HttpSession. An argument of this type enforces the presence of a corresponding session. As a consequence, such an argument is never null.

|  |
| --- |
| IMG_256 |
| Session access may not be thread-safe, in particular in a Servlet environment. Consider setting the RequestMappingHandlerAdapter's "synchronizeOnSession" flag to "true" if multiple requests are allowed to access a session concurrently. |

* org.springframework.web.context.request.WebRequest or org.springframework.web.context.request.NativeWebRequest. Allows for generic request parameter access as well as request/session attribute access, without ties to the native Servlet/Portlet API.
* java.util.Locale for the current request locale, determined by the most specific locale resolver available, in effect, the configured LocaleResolver /LocaleContextResolver in an MVC environment.
* java.util.TimeZone (Java 6+) / java.time.ZoneId (on Java 8) for the time zone associated with the current request, as determined by aLocaleContextResolver.
* java.io.InputStream / java.io.Reader for access to the request’s content. This value is the raw InputStream/Reader as exposed by the Servlet API.
* java.io.OutputStream / java.io.Writer for generating the response’s content. This value is the raw OutputStream/Writer as exposed by the Servlet API.
* org.springframework.http.HttpMethod for the HTTP request method.
* java.security.Principal containing the currently authenticated user.
* @PathVariable annotated parameters for access to URI template variables. See [the section called “URI Template Patterns”](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-ann-requestmapping-uri-templates" \o "URI Template Patterns).
* @MatrixVariable annotated parameters for access to name-value pairs located in URI path segments. See [the section called “Matrix Variables”](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-ann-matrix-variables" \o "Matrix Variables).
* @RequestParam annotated parameters for access to specific Servlet request parameters. Parameter values are converted to the declared method argument type. See [the section called “Binding request parameters to method parameters with @RequestParam”](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-ann-requestparam" \o "Binding request parameters to method parameters with @RequestParam).
* @RequestHeader annotated parameters for access to specific Servlet request HTTP headers. Parameter values are converted to the declared method argument type. See [the section called “Mapping request header attributes with the @RequestHeader annotation”](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-ann-requestheader" \o "Mapping request header attributes with the @RequestHeader annotation).
* @RequestBody annotated parameters for access to the HTTP request body. Parameter values are converted to the declared method argument type usingHttpMessageConverters. See [the section called “Mapping the request body with the @RequestBody annotation”](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-ann-requestbody" \o "Mapping the request body with the @RequestBody annotation).
* @RequestPart annotated parameters for access to the content of a "multipart/form-data" request part. See [Section 18.10.5, “Handling a file upload request from programmatic clients”](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-multipart-forms-non-browsers" \o "18.10.5 Handling a file upload request from programmatic clients) and [Section 18.10, “Spring’s multipart (file upload) support”](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-multipart" \o "18.10 Spring’s multipart (file upload) support).
* @SessionAttribute annotated parameters for access to existing, permanent session attributes (e.g. user authentication object) as opposed to model attributes temporarily stored in the session as part of a controller workflow via @SessionAttributes.
* @RequestAttribute annotated parameters for access to request attributes.
* HttpEntity<?> parameters for access to the Servlet request HTTP headers and contents. The request stream will be converted to the entity body usingHttpMessageConverters. See [the section called “Using HttpEntity”](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-ann-httpentity" \o "Using HttpEntity).
* java.util.Map / org.springframework.ui.Model / org.springframework.ui.ModelMap for enriching the implicit model that is exposed to the web view.
* org.springframework.web.servlet.mvc.support.RedirectAttributes to specify the exact set of attributes to use in case of a redirect and also to add flash attributes (attributes stored temporarily on the server-side to make them available to the request after the redirect). See [the section called “Passing Data To the Redirect Target”](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-redirecting-passing-data" \o "Passing Data To the Redirect Target) and [Section 18.6, “Using flash attributes”](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-flash-attributes" \o "18.6 Using flash attributes).
* Command or form objects to bind request parameters to bean properties (via setters) or directly to fields, with customizable type conversion, depending on@InitBinder methods and/or the HandlerAdapter configuration. See the webBindingInitializer property on RequestMappingHandlerAdapter. Such command objects along with their validation results will be exposed as model attributes by default, using the command class name - e.g. model attribute "orderAddress" for a command object of type "some.package.OrderAddress". The ModelAttribute annotation can be used on a method argument to customize the model attribute name used.
* org.springframework.validation.Errors / org.springframework.validation.BindingResult validation results for a preceding command or form object (the immediately preceding method argument).
* org.springframework.web.bind.support.SessionStatus status handle for marking form processing as complete, which triggers the cleanup of session attributes that have been indicated by the @SessionAttributes annotation at the handler type level.
* org.springframework.web.util.UriComponentsBuilder a builder for preparing a URL relative to the current request’s host, port, scheme, context path, and the literal part of the servlet mapping.

The Errors or BindingResult parameters have to follow the model object that is being bound immediately as the method signature might have more than one model object and Spring will create a separate BindingResult instance for each of them so the following sample won’t work:

**Invalid ordering of BindingResult and @ModelAttribute.**

@PostMapping**public** String processSubmit(****@ModelAttribute("pet") Pet pet****, Model model, ****BindingResult result****) { ... }

Note, that there is a Model parameter in between Pet and BindingResult. To get this working you have to reorder the parameters as follows:

@PostMapping**public** String processSubmit(****@ModelAttribute("pet") Pet pet****, ****BindingResult result****, Model model) { ... }

|  |
| --- |
| IMG_257 |
| JDK 1.8’s java.util.Optional is supported as a method parameter type with annotations that have a required attribute (e.g. @RequestParam,@RequestHeader, etc). The use of java.util.Optional in those cases is equivalent to having required=false. |

@MatrixVariable

@RequestParam

@RequestHeader

@RequestBody

@RequestPart

@SessionAttribute

@RequestAttribute

**@ModelAttribute.**

#### **Supported method return types**

The following are the supported return types:

* A ModelAndView object, with the model implicitly enriched with command objects and the results of @ModelAttribute annotated reference data accessor methods.
* A Model object, with the view name implicitly determined through a RequestToViewNameTranslator and the model implicitly enriched with command objects and the results of @ModelAttribute annotated reference data accessor methods.
* A Map object for exposing a model, with the view name implicitly determined through a RequestToViewNameTranslator and the model implicitly enriched with command objects and the results of @ModelAttribute annotated reference data accessor methods.
* A View object, with the model implicitly determined through command objects and @ModelAttribute annotated reference data accessor methods. The handler method may also programmatically enrich the model by declaring a Model argument (see above).
* A String value that is interpreted as the logical view name, with the model implicitly determined through command objects and @ModelAttribute annotated reference data accessor methods. The handler method may also programmatically enrich the model by declaring a Model argument (see above).
* void if the method handles the response itself (by writing the response content directly, declaring an argument of type ServletResponse /HttpServletResponse for that purpose) or if the view name is supposed to be implicitly determined through a RequestToViewNameTranslator (not declaring a response argument in the handler method signature).
* If the method is annotated with @ResponseBody, the return type is written to the response HTTP body. The return value will be converted to the declared method argument type using HttpMessageConverters. See [the section called “Mapping the response body with the @ResponseBody annotation”](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-ann-responsebody" \o "Mapping the response body with the @ResponseBody annotation).
* An HttpEntity<?> or ResponseEntity<?> object to provide access to the Servlet response HTTP headers and contents. The entity body will be converted to the response stream using HttpMessageConverters. See [the section called “Using HttpEntity”](http://docs.spring.io/spring/docs/5.0.0.M2/spring-framework-reference/htmlsingle/" \l "mvc-ann-httpentity" \o "Using HttpEntity).
* An HttpHeaders object to return a response with no body.
* A Callable<?> can be returned when the application wants to produce the return value asynchronously in a thread managed by Spring MVC.
* A DeferredResult<?> can be returned when the application wants to produce the return value from a thread of its own choosing.
* A ListenableFuture<?> can be returned when the application wants to produce the return value from a thread of its own choosing.
* A ResponseBodyEmitter can be returned to write multiple objects to the response asynchronously; also supported as the body within a ResponseEntity.
* An SseEmitter can be returned to write Server-Sent Events to the response asynchronously; also supported as the body within a ResponseEntity.
* A StreamingResponseBody can be returned to write to the response OutputStream asynchronously; also supported as the body within a ResponseEntity.
* Any other return type is considered to be a single model attribute to be exposed to the view, using the attribute name specified through @ModelAttribute at the method level (or the default attribute name based on the return type class name). The model is implicitly enriched with command objects and the results of@ModelAttribute annotated reference data accessor methods.

****1. @Controller****

    Controller控制器是通过服务接口定义的提供访问应用程序的一种行为，它解释用户的输入，将其转换成一个模型然后将试图呈献给用户。Spring MVC 使用 @Controller 定义控制器，它还允许自动检测定义在类路径下的组件并自动注册。如想自动检测生效，需在XML头文件下引入 spring-context:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <?**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.xsd          http://www.springframework.org/schema/context          http://www.springframework.org/schema/context/spring-context.xsd">        <**context:component-scan** base-package="org.springframework.samples.petclinic.web"/>        <!-- ... --></**beans**> |

****2. @RequestMapping****

    我们可以 @RequestMapping 注解将类似 “/favsoft”这样的URL映射到整个类或特定的处理方法上。一般来说，类级别的注解映射特定的请求路径到表单控制器上，而方法级别的注解只是映射为一个特定的HTTP方法请求（“GET”，“POST”等）或HTTP请求参数。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | @Controller  @RequestMapping("/favsoft")  **public** **class** AnnotationController {        @RequestMapping(method=RequestMethod.GET)  **public** String get(){  **return** "";      }        @RequestMapping(value="/getName", method = RequestMethod.GET)  **public** String getName(String userName) {  **return** userName;      }        @RequestMapping(value="/{day}", method=RequestMethod.GET)  **public** String getDay(Date day){          DateFormat df = **new** SimpleDateFormat("yyyy-MM-dd");  **return** df.format(day);      }        @RequestMapping(value="/addUser", method=RequestMethod.GET)  **public** String addFavUser(@Validated FavUser favUser,BindingResult result){  **if**(result.hasErrors()){  **return** "favUser";          }          //favUserService.addFavUser(favUser);  **return** "redirect:/favlist";      }        @RequestMapping("/test")      @ResponseBody  **public** String test(){  **return** "aa";      }    } |

    @RequestMapping 既可以作用在类级别，也可以作用在方法级别。当它定义在类级别时，标明该控制器处理所有的请求都被映射到 /favsoft 路径下。@RequestMapping中可以使用 method 属性标记其所接受的方法类型，如果不指定方法类型的话，可以使用 HTTP GET/POST 方法请求数据，但是一旦指定方法类型，就只能使用该类型获取数据。

    @RequestMapping 可以使用 @Validated与BindingResult联合验证输入的参数，在验证通过和失败的情况下，分别返回不同的视图。

    @RequestMapping支持使用URI模板访问URL。URI模板像是URL模样的字符串，由一个或多个变量名字组成，当这些变量有值的时候，它就变成了URI。

****3. @PathVariable****

    在Spring MVC中，可以使用 @PathVariable 注解方法参数并将其绑定到URI模板变量的值上。如下代码所示：

|  |
| --- |
| String findOwner( String , Model model) {      FavUser favUser = favUserService.findFavUser();      model.addAttribute(       ;  } |

 URI模板 “favusers/{favUserId}"指定变量的名字 favUserId ,当控制器处理这个请求的时候， favUserId的值会被设定到URI中。比如，当有一个像“favusers/favccxx”这样的请求时，favUserId的值就是 favccxx。

    @PathVariable 可以有多个注解，像下面这样：

@RequestMapping(value="/owners/{ownerId}/pets/{petId}", method=RequestMethod.GET)public String findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {

    Owner owner = ownerService.findOwner(ownerId);

    Pet pet = owner.getPet(petId);

    model.addAttribute("pet", pet);    return "displayPet";

}

    @PathVariable中的参数可以是任意的简单类型，如int, long, Date等等。Spring会自动将其转换成合适的类型或者抛出 TypeMismatchException异常。当然，我们也可以注册支持额外的数据类型。

    如果@PathVariable使用Map<String, String>类型的参数时， Map会填充到所有的URI模板变量中。

    @PathVariable支持使用正则表达式，这就决定了它的超强大属性，它能在路径模板中使用占位符，可以设定特定的前缀匹配，后缀匹配等自定义格式。

    @PathVariable还支持矩阵变量，因为现实场景中用的不多，这就不详细介绍了，有需要的童鞋请查看官网的文档。

****4. @RequestParam****

    @RequestParam将请求的参数绑定到方法中的参数上，如下面的代码所示。其实，即使不配置该参数，注解也会默认使用该参数。如果想自定义指定参数的话，如果将@RequestParam的 required 属性设置为false（如@RequestParam（value="id",required=false））。

    5. ****@RequestBody****

    @RequestBody是指方法参数应该被绑定到HTTP请求Body上。

@RequestMapping(value = "/something", method = RequestMethod.PUT)public void handle(@RequestBody String body, Writer writer) throws IOException {

    writer.write(body);

}

   如果觉得@RequestBody不如@RequestParam趁手，我们可以使用 HttpMessageConverter将request的body转移到方法参数上， HttMessageConverser将 HTTP请求消息在Object对象之间互相转换，但一般情况下不会这么做。事实证明，@RequestBody在构建REST架构时，比@RequestParam有着更大的优势。

  6. ****@ResponseBody****

    @ResponseBody与@RequestBody类似，它的作用是将返回类型直接输入到HTTP response body中。@ResponseBody在输出JSON格式的数据时，会经常用到，代码见下图：

@RequestMapping(value = "/something", method = RequestMethod.PUT)@ResponseBodypublic String helloWorld() {    return "Hello World";

}

    7. ****@RestController****

    我们经常见到一些控制器实现了REST的API，只为服务于JSON，XML或其它自定义的类型内容，@RestController用来创建REST类型的控制器，与@Controller类型。@RestController就是这样一种类型，它避免了你重复的写@RequestMapping与@ResponseBody。

|  |
| --- |
| @RestController  public class FavRestfulController {    @RequestMapping(value="/getUserName",method=RequestMethod.POST)  public String getUserName(@RequestParam(value="name") String name){  return name;  }  } |

    8. ****HttpEntity****

    HttpEntity除了能获得request请求和response响应之外，它还能访问请求和响应头，如下所示：

@RequestMapping("/something")public ResponseEntity<String> handle(HttpEntity<byte[]> requestEntity) throws UnsupportedEncodingException {

    String requestHeader = requestEntity.getHeaders().getFirst("MyRequestHeader"));    byte[] requestBody = requestEntity.getBody();    // do something with request header and body

    HttpHeaders responseHeaders = new HttpHeaders();

    responseHeaders.set("MyResponseHeader", "MyValue");    return new ResponseEntity<String>("Hello World", responseHeaders, HttpStatus.CREATED);

}

****9. @ModelAttribute****

    @ModelAttribute可以作用在方法或方法参数上，当它作用在方法上时，标明该方法的目的是添加一个或多个模型属性（model attributes）。该方法支持与@RequestMapping一样的参数类型，但并不能直接映射成请求。控制器中的@ModelAttribute方法会在@RequestMapping方法调用之前而调用，示例如下：

@ModelAttribute

public Account addAccount(@RequestParam String number) {

    return accountManager.findAccount(number);

}

@ModelAttribute

public void populateModel(@RequestParam String number, Model model) {

    model.addAttribute(accountManager.findAccount(number));

    // add more ...

}

    @ModelAttribute方法用来在model中填充属性，如填充下拉列表、宠物类型或检索一个命令对象比如账户（用来在HTML表单上呈现数据）。

    @ModelAttribute方法有两种风格：一种是添加隐形属性并返回它。另一种是该方法接受一个模型并添加任意数量的模型属性。用户可以根据自己的需要选择对应的风格。

****@ModelAttribute作用在方法参数上****

    当@ModelAttribute作用在方法参数上时，表明该参数可以在方法模型中检索到。如果该参数不在当前模型中，该参数先被实例化然后添加到模型中。一旦模型中有了该参数，该参数的字段应该填充所有请求参数匹配的名称中。这是Spring MVC中重要的数据绑定机制，它省去了单独解析每个表单字段的时间。

    @ModelAttribute是一种很常见的从数据库中检索属性的方法，它通过@SessionAttributes使用request请求存储。在一些情况下，可以很方便的通过URI模板变量和类型转换器检索属性。

    注解的出现终结了XML配置文件漫天飞的年代，它让程序拥有更高的可读性，可配置性与灵活性。当然，也有一些人说注解不如配置文件显的结构清晰，个人觉得所谓的结构应该是一个统一的规范，而不是将一堆文件结构糅合在一起。这就好比是面向对象与面向结构，你能说面向对象的逻辑不清晰吗？