MVC Architecture

Dispatcher servelet configuration :

web.xml - . NOTE :metadata-complete = true indicate JAR files in /WEB-INF/lib doesn't need to be scanned for Servlet 3.0 specific annotations

web-fragment.xml - has to be in the META-INF directory of a jar file or in a jar file

ServletContainerInitializer

(servlet 3.0 spec) Interface - Servlet 3 spec way of boot straping spring mvc

WebApplicationInitializer

Interface -

Code base configuration :

WebApplicationInitializer -

AbstractDispatcherServletInitializer -

AbstractAnnotationConfigDispatcherServletInitializer

The WebApplicationContext is bound in the ServletContext, and by using static methods on the RequestContextUtils class you can always

look up the WebApplicationContext if you need access to it.

contextClass - default is XmlWebApplicationContext

contextConfigLocation - location of [AAA]/[bbb]/[xx]-[yy].xml

namespace - Defaults to [servlet-name]-servlet (Namespace of the WebApplicationContext)

contextId - Provides the Id of the application context.

publishEvents - Indicates whether to fire an event after request processing

publishContext - Indicates whether the servlet’s (dispatcherServlet) application context is being published to the javax.servlet.ServletContex

environment -

dispatchOptionsRequest - by default it will not DispatcherServlet will not handle Option. set it to true to handle the option request.

dispatchTraceRequest - similar to dispatchOptionRequest

environment - Configures the org.springframework.core.env.Environment to use for this servlet. The environment specifies which profile is active andcan hold properties specific for this environment.

detectAll

[HandlerAdapters]

[HandlerExceptionResolvers]

[HandlerMappings]

[ViewResolvers]

Dispatcher Configuration :

Same root context and more than one webAppContext :

===============================================

<context-param>

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

<param-value>/WEB-INF/root-context.xml</param-value>

</context-param>

<listener>

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

</listener>

<servlet>

<servlet-name>dispatcher2</servlet-name>

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

<init-param>

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

<param-value>/WEB-INF/AppServlet-context.xml</param-value>

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>dispatcher2</servlet-name>

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

</servlet-mapping>

<servlet>

<servlet-name>dispatcher1</servlet-name>

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

<init-param>

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

<param-value>/WEB-INF/myservlet-context.xml</param-value>

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>dispatcher1</servlet-name>

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

</servlet-mapping>

Root context and webAppContext in single XML:

=============================================

<context-param>

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

<param-value>/WEB-INF/root-context.xml</param-value>

</context-param>

<servlet>

<servlet-name>dispatcher</servlet-name>

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

<init-param>

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

<param-value></param-value>

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

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

</servlet-mapping>

<listener>

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

</listener>

Special bean types in the WebApplicationContext :

Bean type - Explanation

HandlerMapping - 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.

(RequestMappingHandlerMapping)

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

RequestMappingHandlerMapping is enabled by default when <mvc:annotation-driven>

HandlerMapping component is responsible for routing request URI’s to handlers which are the controller methods

annotated with @RequestMapping annotation

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- Maps exceptions to views also allowing for more complex exception handling code.

ViewResolver - Resolves logical String-based view names to actual View types.

LocaleResolver &

LocaleContextResolver - Resolves the locale a client is using and possibly their time zone, in order to be able to offer internationalized views

ThemeResolver - Resolves themes your web application can use, for example, to offer personalized layouts

MultipartResolver - Parses multi-part requests for example to support processing file uploads from HTML forms.

FlashMapManager - 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.

This information is kept in the file DispatcherServlet.properties in the package org.springframework.web.servlet

HandlerMapping & HandlerAdapater difference

Since introduction of RequestMappingHandlerMapping and RequestMappingHandlerAdapter in Spring 3.1 the distinction is even simpler:

RequestMappingHandlerMapping finds the appropriate handler method for the given request.

RequestMappingHandlerAdapter executes this method, providing it with all the arguments.

Java based Configuration :

@EnableWebMvc (class must be annotated with @Configuration)

DispatcherServlet Processing Sequence : (Highlevel)

1. WebApplicationContext bound in Request as Attribute

2. locale resolver is bound to the request to enable elements in the process to resolve the locale to use when processing the request

3. theme resolver is bound to the request to let elements such as views determine which theme to use

4. If you specify a multipart file resolver, the request is inspected for multiparts;

5. if multiparts are found, the request is wrapped in a MultipartHttpServletRequest for further processing by other elements in the process.

6. Appropriate handler is searched for. If a handler is found, the execution chain associated with the handler (preprocessors, postprocessors, and controllers)

is executed in order to prepare a model or rendering.

7. model is returned, the view is rendered.

8. Handler exception resolvers that are declared in the WebApplicationContext pick up exceptions that are thrown during processing of the request.

DispatcherServlet receives the request.

DispatcherServlet dispatches the task of selecting an appropriate controller to HandlerMapping (RequestMappingHandlerMapping). HandlerMapping selects the controller which is mapped to the incoming request URL and returns the (selected Handler) and Controller to DispatcherServlet.

DispatcherServlet dispatches the task of executing of business logic of Controller to HandlerAdapter.

HandlerAdapter calls the business logic process of Controller.

Controller executes the business logic, sets the processing result in Model and returns the logical name of view to HandlerAdapter.

DispatcherServlet dispatches the task of resolving the View corresponding to the View name to ViewResolver. ViewResolver returns the View mapped to View name.

DispatcherServlet dispatches the rendering process to returned View.

View renders Model data and returns the response.

Controller :

class level annotation

method level annotation

Prior to Spring 3.1, type and method-level request mappings were examined in two separate stages

-- a controller was selected first by the DefaultAnnotationHandlerMapping and the actual method to invoke

was narrowed down second by the AnnotationMethodHandlerAdapter.

With the new support classes in Spring 3.1, the RequestMappingHandlerMapping is the only place where a decision is made about

which method should process the request. Think of controller methods as a collection of unique endpoints

with mappings for each method derived from type and method-level @RequestMapping information.

Controller Types :

==================

1. AbstractCommandController

2. AbstractController

3. AbstractFormController

4. AbstractUrlViewController

\*5. AbstractWizardFormController - deprecated in -favour of annotation-based controllers.

6. BaseCommandController

\*7. CancellableFormController - deprecated in -favour of annotation-based controllers.

8. ComponentControllerSupport

9. Controller

10. EventAwareController -

\*11. MultiActionController - deprecated in -favour of annotation-based controllers.

12. ParameterizableViewController - A controller that returns a configured view name [<mvc:view-controller path="/" view-name="home"/>]

immediately forwards to a view when invoked.Use it in static cases when there is no Java controller

logic to execute before the view generates the response.

<mvc:view-controller path="/" view-name="home"/>

14. PortletWrappingController -

15. ResourceAwareController -

16. ServletForwardingController - A controller implementation that forwards the request to a named servlet. The named servlet can be a servlet without any mapping.

This is useful if you want to use the Spring MVC infrastructure for dispatching requests and to apply interceptors.

"servlet-name" in web.xml rather than a URL path mapping.

17. ServletWrappingController -

18. SimpleControllerHandlerAdapter -

\*19. SimpleFormController - deprecated in -favour of annotation-based controllers.

20. UrlFilenameViewController - A controller implementation that takes the path of a URL and transforms that into a view name.

It can be configured to append a prefix and/or suffix to the view name

Controller

1 • Controller

2 • AbstractCommandController

3 • SimpleFormController

4 • WizardFormController

5 • MultiActionController

URI Template Patterns :

=======================

can be used for convenient access to selected parts of a URL in a @RequestMapping method

@RequestMapping(value="/owners/{ownerId}", method=RequestMethod.GET)

public String findOwner(@PathVariable("ownerId") String theOwner, Model model) {

// implementation omitted

}

Matrix variables :

// GET /owners/42;q=11;r=12/pets/21;q=22;s=23

@RequestMapping(value = "/owners/{ownerId}/pets/{petId}", method = RequestMethod.GET)

public void findPet(

@MatrixVariable Map<String, String> matrixVars,

@MatrixVariable(pathVar="petId"") Map<String, String> petMatrixVars)

{

// matrixVars: ["q" : [11,22], "r" : 12, "s" : 23]

// petMatrixVars: ["q" : 11, "s" : 23]

}

Request Parameters and Header Values

====================================

@RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET, params="myParam=myValue", headers="myHeader=myValue")

public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {

// implementation omitted

}

@RequestMapping(value = "/home", consumes = {MediaType.APPLICATION\_JSON\_VALUE,MediaType.APPLICATION\_FORM\_URLENCODED\_VALUE},

produces = MediaType.APPLICATION\_JSON\_VALUE, method = RequestMethod.POST)

public ModelAndView welcom(HttpServletRequest request)

value , method , params [param-name=param-value, param-name!=param-value]

headers [header-name=header-value, header-name!=header-value ]

consumes

produces

Supported method argument types

===============================

ServletRequest or HttpServletRequest

HttpSession

WebRequest or NativeWebRequest

java.util.Locale

java.io.InputStream / java.io.Reader

java.io.OutputStream / java.io.Writer

java.security.Principal containing the currently authenticated user.

@PathVariable, @MatrixVariable & @RequestParam

@RequestHeader

@RequestBody (converted to the declared method argument type using HttpMessageConverters)

@RequestPart ( RequestPart annotation is put on a method argument of the type javax.servlet.http.Part, org.springframework.web.multipart.MultipartFile (or on a collection or array of the latter,) then we will get the content of that file (or group of files) injected.) if not found fall back to httpMessage converters

HttpEntity<?> parameters for access to the Servlet request HTTP headers and contents. request stream converted to the entity body using HttpMessageConverters.

Map or org.springframework.ui.Model or org.springframework.ui.ModelMap - for enriching the implicit model that is exposed to the web view.

Command or form objects

org.springframework.validation.Errors / org.springframework.validation.BindingResult validation results for a preceding command or form object

(the immediately preceding method argument)

SessionStatus

UriComponentsBuilder a builder for preparing a URL relative to the current request's host, port, scheme, context path

Supported method return types

===============================

ModelAndView

Model

Map

View

void if the method handles the response itself

@ResponseBody

-- The return value will be converted to the declared method argument type using HttpMessageConverters.

HttpEntity<?> or ResponseEntity<?> object to provide access to the Servlet response HTTP headers and contents.

Controller method annotations

=============================

@RequestParam

@RequestBody method parameter annotation indicates that a method parameter should be bound to the value of the HTTP request body

@RequestMapping("/displayHeaderInfo.do")

public void displayHeaderInfo(@CookieValue("JSESSIONID") String cookie) {

//...

}

@RequestMapping("/displayHeaderInfo.do")

public void displayHeaderInfo(@RequestHeader("Accept-Encoding") String encoding,

@RequestHeader("Keep-Alive") long keepAlive) {

//...

}

Binding :

==========

Global configuration -

Per controller configuration -

Binder - allows for setting property values onto a target object, including support for validation and binding result analysis.

WebDataBinder - Special DataBinder for data binding from web request parameters to JavaBean objects. Designed for web environments,

but not dependent on the Servlet API;

ServletRequestDataBinder - perform data binding from servlet request parameters to JavaBeans, including support for multipart files

method with @InitBinder- support all arguments that @RequestMapping supports, except for command/form objects and corresponding validation result objects

must not have a return value.

identify the methods which initialize the WebDataBinder. also its place where customEditor & customValidato can be set

ConfigurableWebBindingInitializer : allow reusing pre-configured initializers with multiple controller/handlers.

allow to configure

ConversionService

MessageCodesResolver

PropertyEditorRegistrar

PropertyEditorRegistrar[]

Validator

interface Validator :

not coupled to validating only objects in the web tier, the data-access tier, or the whatever-tier

@Component

public class UserValidator implements Validator {

@Override

public boolean supports(Class<?> clazz) {

return User.class.isAssignableFrom(clazz);

}

@Override

public void validate(Object target, Errors errors) {

User user = (User)target;

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "","Username is empty");

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "password", "", "Password is empty");

if (user.getName().length()<5)

{

errors.rejectValue("name","", "Username length is less than 5");

}

}

}

@InitBinder

public void dataBinding(WebDataBinder binder) {

binder.addValidators(userValidator, emailValidator);

SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");

dateFormat.setLenient(false);

binder.registerCustomEditor(Date.class, "dob", new CustomDateEditor(dateFormat, true));

}

for multiple command/ form backing objects

@InitBinder("user")

protected void initUserBinder(WebDataBinder binder) {

binder.setValidator(new UserValidator());

}

@InitBinder("payment")

protected void initPaymentBinder(WebDataBinder binder) {

binder.setValidator(new CustomerPaymentValidator());

}

Spring will return 400 Bad Request, if there is type conversion failure/failed to bind the values in object because of type mismatch.

Interceptors :

HandlerInterceptor - Interface

HandlerInterceptorAdapter - implementation class

interceptor will be called after determining the handlerMapping.

preHandle(HttpServletRequest request,HttpServletResponse response,Object handler) throws Exception

postHanlde (HttpServletRequest request,HttpServletResponse response,Object handler,ModelAndView modelAndView)throws Exception

afterCompletion(HttpServletRequest request,HttpServletResponse response,Object handler,Exception ex)throws Exception

WebRequestInterceptor Interface

preHandle(WebRequest request)

postHandle(WebRequest request, ModelMap model)

afterCompletion(WebRequest request, Exception ex)

<beans>

<bean id="handlerMapping" class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping">

<property name="interceptors">

<list>

<ref bean="officeHoursInterceptor"/>

</list>

</property>

</bean>

<bean id="officeHoursInterceptor" class="samples.TimeBasedAccessInterceptor"> (HandlerInterceptorAdapter)

<property name="openingTime" value="9"/>

<property name="closingTime" value="18"/>

</bean>

<beans>

or

<mvc:interceptors>

<mvc:interceptor>

<mvc:mapping path="" />

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

<ref bean="" />

</mvc:interceptor>

</mvc:interceptors>

HandlerMapping

define a mapping between requests and handler objects. getHandler will identify the Handler Object.

A handler will always be wrapped in a HandlerExecutionChain instance, optionally accompanied by some HandlerInterceptor instances.

1 • BeanNameUrlHandlerMapping - Maps controllers to URLs that are based on the controllers’ bean names.

you'll need to use an alias name="/foo" in the bean definition, as the XML id may not contain slashes.

alias or bean name must contain /

supports ant pattern matching (/viewMod\* -> viewModules or viewModuleTest)

name should match prefixed with /. id is not used for mapping

2 • SimpleUrlHandlerMapping - this requires explicit mapping

map from URLs to request handler beans

setUrlMap(Map<String,?> urlMap)

setMappings(Properties mappings)

registerHandlers(Map<String,Object> urlMap)

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

<property name="mappings">

<props>

<prop key="/simpleUrlOne.htm">firstController</prop>

<prop key="/simpleUrlTwo.htm">secondController</prop>

</props>

</property>

</bean>

<bean id="firstController" class="com.candidjava.springmvc.FirstController" />

<bean id="secondController" class="com.candidjava.springmvc.SecondController" />

3 • ControllerClassNameHandlerMapping -

simple Controller implementations (those that handle a single request type),

the convention is to take the short name of the Class, remove the 'Controller' suffix if it exists and return the remaining text,

lower-cased, as the mapping, with a leading /.

For example: WelcomeController -> /welcome\* HomeController -> /home\*

MultiActionController MultiActionControllers and @Controller beans, a similar mapping is registered, except that all sub-paths are registered using the trailing wildcard pattern /\*.

For example: WelcomeController -> /welcome, /welcome/\* ,CatalogController -> /catalog, /catalog/\*

5 • DefaultAnnotationHandlerMapping - Maps request to controller and controller methods that are annotated with @RequestMapping.

registered by default

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

Not registered by default. @EnableMvc or mvc:annotation-driven

7. ControllerBeanNameHandlerMapping - BeanNameUrlHandlerMapping but doesn't expect bean names to follow the URL convention:

It turns plain bean names into URLs by prepending a slash and optionally applying a specified prefix and/or suffix.

it doesn’t require the bean name to start with a /.

Optionally, it can also apply a suffix to the generated URL mapping

2 • <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">

<property name="mappings">

<props>

<prop key="/saveEmployee.htm">saveEmployeeController</prop>

<prop key="/deleteEmployee.htm">deleteEmployeeController</prop>

</props>

</property>

</bean>

<bean id="saveEmployeeController" class="com.answersz.employee.SaveEmployeeController" />

<bean id="deleteEmployeeController" class="com.answersz.employee.DeleteEmployeeController" />

HandlerAdapter :

HandlerAdapter is the glue between the dispatcher servlet and the selected handler.

It removes the actual execution logic from the dispatcher servlet

it does the below configuration :

WebBindingInitializer

ArgumentResolvers

MessageConverters

ModelAndViewResolvers

ReturnValueHandlers

Compare Last-Modified & IfModified-Since.If there was a modification, the content will be regenerated and resent

HttpRequestHandlerAdapter - mostly used by spring remoting

SimpleControllerHandlerAdapter - know how to execute org.springframework.web.servlet.mvc.Controller implementations

handleRequest method return ModelAndView.

SimpleServletHandlerAdapter - configure javax.servlet.Servlet instances in the application context and put them behind the dispatcher servlet.

It knows how to execute the javax.servlet.Servlet, and it always return null because it expects

the servlet to handle the response itself

AnnotationMethodHandlerAdapter - used to execute method annotated with @RequestMapping . Return value of the method is converted or added to the

org.springframework.web.servlet.ModelAndView internally created by this handler adapter

RequestMappingHandlerAdapter - similar to AnnoatationMethodHandlerAdapter

The RequestMappingHandlerAdapter provides a flag called "ignoreDefaultModelOnRedirect"

that can be used to indicate the content of the default Model should never be used

if a controller method redirects.

Instead the controller method should declare an attribute of type RedirectAttributes or

if it doesn't do so no attributes should be passed on to RedirectView

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

<property name="cacheSeconds" value="0" />

<property name="webBindingInitializer">

<bean class="org.springframework.samples.petclinic.web.ClinicBindingInitializer" />

</property>

</bean>

MultipartResolver :

MultipartResolver interface used to determine multipart file request. if so wraps with

org.springframework.web.multipart.MultipartHttpServletRequest.

isMultipart()

resolveMultipart ()

cleanupMultipart () called after request processing completed

CommonsMultipartResolver : -- its an apache common file upload library

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

<property name="maxUploadSize" value="100000"/>

</bean>

StandardServletMultipartResolver : Servlet 3.0 spec way of handling multipart forms. Support for setting MultipartConfigElement

<servlet>

<servlet-name>SpringDispatcher</servlet-name>

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

<multipart-config>

<location>/tmp</location>

<max-file-size>5242880</max-file-size><!--5MB-->

<max-request-size>20971520</max-request-size><!--20MB-->

<file-size-threshold>0</file-size-threshold>

</multipart-config>

</servlet>

MultipartFile or MultipartHttpServletRequest used in controller method.

MultipartException will be thrown if there is any error occurred during handling the multipart request.

ExceptionResolver :

Interface HandlerExceptionResolver

ExceptionHandlerExceptionResolver

AnnotationMethodHandlerExceptionResolver - Searches the current controller for methods annotated with @ExceptionHandler and selects the best exception-handling method to handle the exception.

This exception resolver is enabled by default in the DispatcherServlet.

This is also a drawback because this pair of exception resolvers only operates on methods in the currently assigned controller.

doesn't support global exception handling

DefaultHandlerExceptionResolver - Translates well-known (Spring exceptions) exceptions to a proper appropriate HTTP Response Code for the client.

Returns an empty ModelAndView

This exception resolver is enabled by default in the DispatcherServlet.

(4XX & 5XX status code)

SimpleMappingExceptionResolver - Maps exceptions to view names by the exception class name or part (substring) of that class name.

This implementation can be configured either globally or for certain controllers

SimpleMappingExceptionResolver exceptionResolver;

exceptionResolver = new SimpleMappingExceptionResolver();

Properties mappings = new Properties();

mappings.setProperty("AuthenticationException", "login");

Properties statusCodes = new Properties();

mappings.setProperty("login",String.valueOf(HttpServletResponse.SC\_UNAUTHORIZED));

exceptionResolver.setExceptionMappings(mappings);

exceptionResolver.setStatusCodes(statusCodes);

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

<property name="exceptionMappings">

<props>

<prop key="com.howtodoinjava.demo.exception.AuthException">

error/authExceptionView

</prop>

</props>

</property>

<property name="defaultErrorView" value="error/genericView"/>

</bean>

ResponseStatusExceptionResolver - checks if the thrown exception is annotated with an org.springframework.web.bind.annotation.ResponseStatus annotation

@ResponseStatus(value = HttpStatus.FORBIDDEN ,reason =" exception reason")

in Spring 3.2 :

@ControllerAdvice and a method with @ExceptionHandler. This method will be called whenever an unhandled exception occurs.

@ExceptionHandler({ResourceNotFoundException.class,IOException.class})

@ResponseStatus(HttpStatus.NOT\_FOUND)

public String handleIOException(IOException ex, HttpServletRequest request)

{

return ClassUtils.getShortName(ex.getClass());

}

@ExceptionHandler

public ModelAndView handleIOException(DataAccessException ex, Principal principal, WebRequest request)

{

ModelAndView mav = new ModelAndView("db-error");

mav.addObject("username", principal.getName());

mav.addAllObjects(request.getParameterMap());

for(Iterator<String> names = request.getHeaderNames(); names.hasNext(); )

{

String name = names.next();

String[] value = request.getHeaderValues(name);

mav.addObject(name, value);

}

return mav;

}

ResponseStatusException (Spring 5 and Above)

catch (MyResourceNotFoundException exc) {

throw new ResponseStatusException( HttpStatus.NOT\_FOUND, "Foo Not Found", exc);

}

ViewResolver :

Interface ViewResolver

View resolveViewName(String viewName, Locale locale) throws Exception;

interface View

String getContentType()

void render(Map<String,?> model,HttpServletRequest request,HttpServletResponse response) throws Exception

Types :

BeanNameViewResolver - one of the default ones configured. takes the name of the view and looks in the

org.springframework.context.ApplicationContext to see if there is a View with that name

big drawback in that each view needs to be configured in the application context

XmlViewResolver - Similar to BeanNameViewResolver.

By default it will load the file named /WEB-INF/views.xml

location - The location of the XML file defining the view beans. Defaults to /WEB-INF/views.xml.

order - The order in which this view resolver is called in the chain. The higher the number, the lower the order in the chain.

views.xml

<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="index" class="org.springframework.web.servlet.view.JstlView">

<property name="url" value="/WEB-INF/views/index.jsp"/>

</bean>

</beans>

ResourceBundleViewResolver - Very similar to the XmlViewResolver, this implementation uses a properties file instead of an XML file to hold the view configuration.

have different views for different languages. load from views.properties

The ResourceBundleViewResolver has to be provided with a property file containing viewnames and viewclasses

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

<property name="basename" value="views"/>

</bean>

# And a sample properties file is uses (views.properties in WEB-INF/classes):

welcome.(class)=org.springframework.web.servlet.view.JstlView

welcome.url=/WEB-INF/jsp/welcome.jsp

productList.(class)=org.springframework.web.servlet.view.JstlView

productList.url=/WEB-INF/jsp/productlist.jsp

UrlBasedViewResolver - expects the view name to map directly to a URL,

without an explicit mapping definition.

The UrlBasedViewResolver instantiates the given viewClass for each view it has to resolve.

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

<property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>

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

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

</bean>

InternalResourceViewResolver - subclass of UrlBasedViewResolver that supports InternalResourceView.

adds some additional support to include beans in the model so that they are available in the view for rendering

VelocityViewResolver / - Subclass of UrlBasedViewResolver

FreeMarkerViewResolver

ContentNegotiatingViewResolver - resolve views by name and content-type.

It works by first determining which contenttype is requested,

which it does by checking the file extension,checking the Accept header.

After the content-type is determined, the resolver consults all configured view resolvers

to collect the candidate views by name

ContentNegotiatingViewResolver does not perform the view resolution itself but instead delegates to a list of view resolvers that

you specify through the bean property ViewResolvers

Note : If ContentNegotiatingViewResolver's list of ViewResolvers is not configured explicitly,

it automatically uses any ViewResolvers defined in the application context.

Use a distinct URI for each resource

fred.pdf requests a PDF representation

fred.xml requests an XML representation

Accept-Header

application/pdf requests a PDF representation

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

<property name="mediaTypes">

<map>

<entry key="atom" value="application/atom+xml"/>

<entry key="html" value="text/html"/>

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

</map>

</property>

<property name="viewResolvers">

<list>

<bean class="org.springframework.web.servlet.view.BeanNameViewResolver"/>

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

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

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

</bean>

</list>

</property>

<property name="defaultViews">

<list>

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

</list>

</property>

</bean>

<bean id="content" class="com.springsource.samples.rest.SampleContentAtomView"/>

Serving Static resource :

Allows static resource requests following a particular URL pattern to be served by a ResourceHttpRequestHandler from any of a list of Resource locations.

provides a convenient way to serve static resources from locations other than the web application root,including locations on the classpath.

<mvc:resources mapping="/resources/\*\*" location="/public-resources/"/>

<mvc:resources mapping="/resources/\*\*" location="/, classpath:/META-INF/public-web-resources/"/>

application.version=1.0.0

<util:properties id="applicationProps" location="/WEB-INF/spring/application.properties"/>

<mvc:resources mapping="/resources-#{applicationProps['application.version']}/\*\*" location="/public-resources/"/>

If the default Servlet has been custom configured with a different name, or if a different Servlet container is being used where the default

Servlet name is unknown, then the default Servlet's name must be explicitly provided

<mvc:default-servlet-handler default-servlet-name="myCustomDefaultServlet"/>

LocaleResolver :

strategy that is used to detect which Locale to use.

FixedLocaleResolver : Always resolves to a fixed locale. changing the locale isn’t supported.

SessionLocaleResolver : Resolves (and stores) the locale in the user’s HttpSession.

as well as the default locale to use if no locale is present.

AcceptHeaderLocaleResolver :Uses the http accept header to resolve the locale to use.

In general, this is the locale of the operating system of the user, so changing the locale isn’t supported.

Default localeResolver.

CookieLocaleResolver : Uses a cookie to store the user’s locale.

LocaleChangeInterceptor : If we want our users to be able to change the locale , configure LocaleChangeInterceptor.

checks whether there is a parameter named locale on the request.

name can be configured.

<bean id="messageSource"

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

<property name="basename" value="classpath:validation" />

</bean>

<bean id="localeResolver"

class="org.springframework.web.servlet.i18n.CookieLocaleResolver">

<property name="defaultLocale" value="en"></property>

<property name="cookieMaxAge" value="120"></property>

<property name="cookieName" value="com.rdp.local"></property>

</bean>

<mvc:interceptors>

<mvc:interceptor>

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

<bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">

<property name="paramName" value="lang"></property>

</bean>

</mvc:interceptor>

</mvc:interceptors>

Defaulters :

Default HttpMessageConverters

=============================

ByteArrayHttpMessageConverter converts byte arrays.

StringHttpMessageConverter converts strings.

FormHttpMessageConverter converts form data to/from a MultiValueMap<String, String>.

SourceHttpMessageConverter converts to/from a javax.xml.transform.Source.

Table 4-4. The DispatcherServlet’s Default Components

Component Default implementation(s)

=======================================================================================

MultipartResolver No default, explicit configuration is required.

LocaleResolver org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver

ThemeResolver org.springframework.web.servlet.theme.FixedThemeResolver

HandlerMapping(detect multiple) org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping

org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping

org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping (@MVC DefaultAnnotationHandl.Mapp replaced with this)

HandlerAdapter(detect multiple) org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter

org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter

org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter

org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter (@MVC AnnotationMethodHandlerAdapter replaced with this)

HandlerExceptionResolver org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerExceptionResolver

(detect multiple) org.springframework.web.servlet.mvc.annotation.ResponseStatusExceptionResolver

org.springframework.web.servlet.mvc.support.DefaultHandlerExceptionResolver

org.springframework.web.servlet.mvc.method.annotation.ExceptionHandlerExceptionResolver (@MVC AnnotationMethodHandlerExceptionResolver replaced with this)

RequestToViewNameTranslator org.springframework.web.servlet.view.DefaultRequestToViewNameTranslator

ViewResolver org.springframework.web.servlet.view.InternalResourceViewResolver

(detect multiple)

FlashMapManager org.springframework.web.servlet.support.SessionFlashMapManager

http://answersz.com/spring-mvc-interview-questions-for-experienced/

http://terasolunaorg.github.io/guideline/1.0.1.RELEASE/en/Overview/SpringMVCOverview.html

Spring 3.1 introduces a new set of support classes for processing requests with annotated controllers:

RequestMappingHandlerMapping

RequestMappingHandlerAdapter

ExceptionHandlerExceptionResolver

These classes are a replacement for the existing:

DefaultAnnotationHandlerMapping

AnnotationMethodHandlerAdapter

AnnotationMethodHandlerExceptionResolver