Understanding the flow of Spring Web MVC



Working With SpringMvc We have Different Types of Coponents :

* DispatherServlet
* HandlerMapping
* Controller
* ViewResolver

**DispatherServlet--->**

**DispatherServlet Act as front controller**

DispatcherServlet taking the request coming from view component and give the request to HandlerMappering

**HandlerMapping**

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

The HandlerMapping is resonsible for mapping the incoming request to the handler that can handle the request

When the DispatcherServlet receives the request it delegates the request to the HandlerMapping.Which Identifies the appropriate HandlerExecutionChain that can handle the request.

The spring built in HandlerMappings

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

BeanNameUrlHandlerMapping

SimpleUrlHandlerMapping

ControllerClassNameHandlerMapping

CommonsPathMapHandlerMapping

Note: No more declaration for the Handler Mapping from spring 3.x

like

BeanNameUrlHandlerMapping

SimpleUrlHandlerMapping

ControllerClassNameHandlerMapping

CommonsPathMapHandlerMapping

All are replaced with a standard @RequestMapping annotation

If the @RequestMapping is applied at class level(can apply at method level for multi-actionscontroller)it required to put a RequestMethod to indicate which method to handle the mapping request

**Types of Controllers**

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

Controller/Handler classes are act as helperclasses to interact with Model component...

Controller

AbstractCommandController

SimpleFormController

WizardFormController

MultiActionController

In Annotation SpringMvc,The controller class is no longer need to extends basedController like AbstractController or SimplrFormController ,just simply annotate the classs with @Controller annotation

ModelAndView

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

ModelandView represented by the class org.springframework.web.servlet.MOdleAndvie is

return by the controller object back to the DispatcherServlet

This class is just a Container class for holding the model and view information.

The Model object represents some piece of information that can be used by the view to display the information

Any kind of view Technlogy can be plugged into the f/w with ease for eg:

Excel,Jasper Reports,pdf,xslt,freeMarker,html,tiles,velocity...

are the supported frameworks as of now

The Model object(represented by org.sf.ui.ModelMap) internally matained as a Map for Stroing the information

eg:

ModelAndView mvnew ModeladnView("successview","greetingMsg",grettingMessage);

The ModelAndView is a value object designed to hold model and view making it possible for a handler to return both model and view in a single return value.

The ModelAndview Ojbect represents a model and view specified by the handler

SpringMvcHandler Interceptors

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

SpringMvc allow to intercept web request through handler interceptors.

The handlerInterceptor hv to implement the HandleInterceptor interface,which contains three methods,

preHandler()

Called before the Handlerexexution,returns a boolean value

true : contineu the handle exeuction chain,

false : stop the execution chain and return it

postHandle()

called after the handler execution,allow manipulate the MJodelandview object before render it to view page

afterCompletion()

called after the complete request has finished

**ViewResolver**

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

The ViewResolver object is responsible to locate the view object

that can render a view for this request.

The DispatcherServlet uses the ViewResolver to resolve the logicalview name described by the ModelAndView Object to locate the view Object.

Spring provides various built-in ViewResolver implementations that meet most of our requirement in locating the view The most commonly used Spring provided built-in viewResolver implementations are...

UrlBasedViewResolver

IntenalResourceViewResolver

ResourceBundleViewResolver

BeanNameViewResolver

XmlViewResolver

Each one of these viewResolver implementations presents a unique way to transform the logical view name to view objects capable of rendering the view for the request.

**How to Configure Where to Configure?**

**DispatcherServlet :**

**================**

**DispatcherServlet is a class given by springf/w its act as frontcontroller.**

**It takes all the request coming from view component/client we can configure in Web.xml.**

**web.xml**

**=========**

**<servlet>**

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

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

**<init-param>**

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

**<param-value>\WEB-INF\dispatcher-servlet.xml</param-value>**

**</init-param>**

**<load-on-startup>0</load-on-startup>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>ds</servlet-name>**

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

**</servlet-mapping>**

* + **In Spring MVC each DispatcherServlet has its own WebApplicationContext. By default, spring always looks for the context at *<dispatcherServletName-servlet.xml>.* However this can be overridden with a Servlet init-param**
  + **The *DispatcherServlet* related *WebApplicationContext* should have MVC-specific configurations such as Controllers, HandlerMappings, ViewResolvers etc…,**

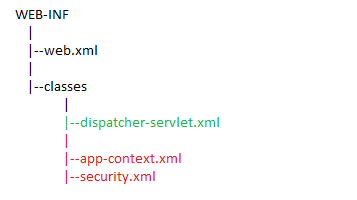
**ROOT WebApplicationContext**

**===========================**

****

* + **Other non MVC-specific configuration such as the beans for service or persistence layer should be in root *WebApplicationContext*.**
  + **In SpringMVC the root *WebApplicationContext* is bootstrapped by using *ContextLoadListener* specified as Listener in web.xml.**
  + **So the DispatcherServlet WebApplicationContext will inherit (extend) from the ROOT WebApplicationContext**

****

****

**Handler Mappings Configuration in Spring Configuration files<dispatcher-servlet>.xml**

* + **HandlerMapping is the class that helps DispatcherServlet to map an incoming request to a particular Controller class.**
  + **There are many HandlerMapping implementations in Spring, however the most used one is the annotated controllers**
  + **HandlerMapping bean has one important property – interceptors to which a user defined handler interceptor can be injected**
  + **RequestMappingHandlerMapping** 
    - **This HandlerMapping implementation automatically looks for @RequestMapping annotations on all @Controller beans**
    - **The RequestMappingHandlerMapping is the only place where a decision is made about which method should process the request**
    - **<mvc:annotation-driven />**
      * **This annotation in the DispatcherServlet WebApplicationContext (file: /WEB-INF/classes/dispatcher-servlet.xml), will automatically register the RequestMappingHandlerMapping bean**

**WEB-INF/classes/dispatcher-servlet.xml**

**==================================**

* + **Import any other context configuration files. For example to import the main ROOT application context file:**

**<import resource=*"app-context.xml"/>***

* + **Configure the @Controller programming model**

**<mvc:annotation-driven />**

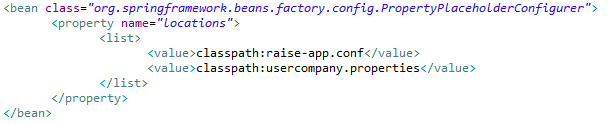
* + **Configure all the View Resolvers**
    - ***TilesViewResolver (and TilesConfigurer)***
    - ***JSPViewResolver***
    - ***MultiPartResolver***
    - ***ExceptionResolver***
    - ***CookieLocaleResolver***

**WEB-INF/classes/app-context.xml**

**===============================**

****

* + **Configure a property place holder configurer**

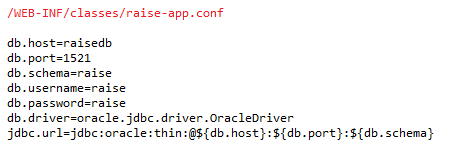
****

**Note that at run-time, these property files will typically be placed at: /WEB-INF/classes folder so they get detected in the classpath scanning**

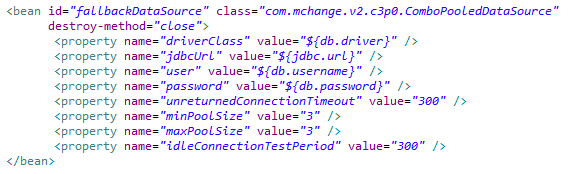
**WEB-INF/classes/app-context.xml**

**===============================**

* + **Property place holder configurer (This is a BeanFactroy PostProcessor)**
    - **Sample Contents**

****

* + - **How to use**

****