Beginning Spring MVC

Spring MVC is—as its name suggests—an implementation of the model-view-controller design pattern. The MVC pattern is fundamental to lose coupling between the data, its visual representation and manipulation. In MVC terms, the data is the model, the visual representation of the data is the view and the data manipulator is the controller.

The MVC pattern is not restricted to just the web; you may find the MVC pattern in Java Swing applications and, outside Java for example in Objective-C Cocoa and Carbon applications.

The clear separation between the M, the V and the C in the pattern means that it is easy to make changes to each of the components with as little impact on the other components. This is the principle of loose coupling and, therefore, it should not be a great surprise that the MVC pattern gets extensive support in the Spring Framework. Even though the MVC pattern does not impose any environment, Spring MVC deals only with web applications.

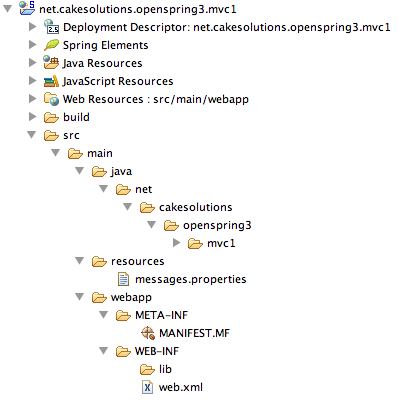
In this chapter, we will explore how the Spring Framework implements the MVC design pattern. To fully understand this chapter, you will need to understand dependency injection, you need to be fluent in Java annotations and you should be able to understand HTML.

Java EE web application revisited

We begin by looking at Java EE web applications. Unlike the command-line applications we have been implementing so far, web applications do not have any public static void main(String[]) method. This method is in the servlet container (Tomcat, for example). The servlet container expects the web application to follow a predefined structure; it looks for descriptor files and, using the information in the descriptor files, the servlet container can run the web application.

In practice, the Java EE web application’s structure must follow the one shown in Figure 1.

Figure 1. Java EE web application structure



Now, the servlet container will read the contents of the web.xml file and construct any servlets it defines. In our case, the servlet is Spring’s DispatcherServlet. The DispatcherServlet will construct the Spring application context by reading a file whose name follows the servletname-context.xml convention. This Spring application context file defines all controllers, view resolvers, annotation handlers, validators and many other components we will cover in this chapter. In real applications, these web components usually need the rest of the application components to operate: the services, which in turn use the repositories, and so on. These components must be ready before the DispatcherServlet attempts to construct the web components. Spring web applications use the ContextLoaderListener to construct the bowels of the application, even before the DispatcherServlet sees the first request. Listing 1 shows the most important elements of our web.xml file.

Listing 1. Web.xml file

<?xml version="1.0" encoding="utf-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee"

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

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

version="2.5">

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

<context-param>

<param-name>contextClass</param-name>

<param-value>

com.springsource.server.web.dm.ServerOsgiBundleXmlWebApplicationContext

</param-value>

</context-param>

<servlet>

<servlet-name>mvc1</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

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

</servlet>

<servlet-mapping>

<servlet-name>mvc1</servlet-name>

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

</servlet-mapping>

</web-app>

The two most important elements in Listing 1 are the ContextLoaderListener (with its context-param set to ServerOsgiBundleXmlWebApplicationContext) and, the DispatcherServlet. It is time to explore how the DispatcherServlet handles the requests.

The story of the film so far

If we were in Monty Python and the Holy Grail, we could say:

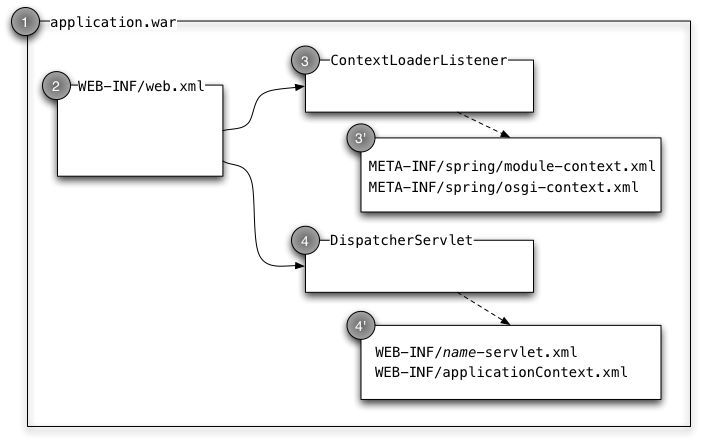
Doug and Bob are metropolitan policemen with a difference. Doug likes nothing more than slipping into little cocktail frocks, while Bob bouffants his hair for a night on duty. Still, as the art immace, no one gives their last names.

The real story of the film so far:

Pucky Reginald Vas Deferens is a nuclear scientist in love with mafia boss Enrico Marx, who is himself married to Conchito Macbeth, a lively belly-dancer at the Belgian disco whose manager…

Unfortunately, we are just Java EE programmers. And so, we have to explore what happens in our servlet container (see Figure 2).

Figure 2. Java EE web application start up



We see that the servlet container deploys our application (1) and, during this process, it reads the WEB-INF/web.xml file. The web.xml file defines the ContextLoaderListener and the DispatcherServlet. The ContextLoaderListener builds the bowels of our Spring-powered application. It does this typically by looking for all module-context.xml and osgi-context.xml files in the META-INF/spring directory. So much for Spring-powered applications. But even in traditional (read old-school) servlet applications, there are usually multiple servlets, each servlet implemented in its own class and each servlet handling requests to a specific URL. In Spring web applications, there is usually just the DispatcherServlet, which handles all requests.

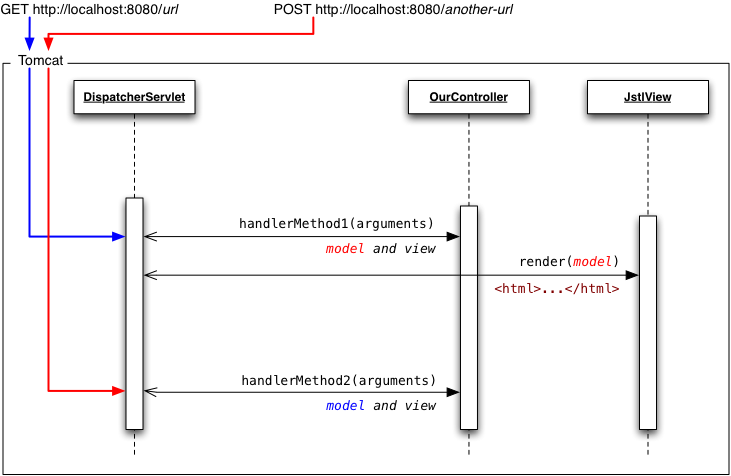
From a high above, the DispatcherServlet needs to route the requests to the appropriate controllers; the controllers react to the requests, prepare models and indicate which view should render the model. The dispatcher servlet takes the model and the view and finds the required view that will ultimately render the model. The DispatcherServlet will return whatever the view’s rendering method returns. In this sense, the DispatcherServlet is the controller, the components we call controllers in Spring are more like controller workers. In Spring-speak, they are sometimes called handlers.

Read on to find out how the DispatcherServlet does this.

The DispatcherServlet

As you have already seen, the DispatcherServlet is the central point of a Spring Framework web application. The servlet handles all requests (passed on to it by the servlet container). To handle the request, it must find one of our controllers, look up the appropriate handler method; then it must examine the handler method’s signature and prepare the argument values. It must then invoke the method and examine the returned value. Depending on the returned value, the DispatcherServlet may have to find the appropriate view. Finally, it will take the model the handler method prepared (the model may remain empty if the handler method does not add any objects to the model); it will give the prepared model to the view and pass on whatever the view returned back to the servlet container as the response to the request. The servlet container will in turn pass on this response to the client. Figure 3 shows what is happening.

Figure 3. DispatcherServlet processing requests



What can the DispatcherServlet use to route the request—and here by request we mean the HttpServletRequest? It can use its URL, HTTP method, query parameters, header parameters, and the values of the cookies. In fact, the DispatcherServlet can use any combination of these attributes of the HttpServletRequest to find the appropriate handler (and appropriate method in the handler) that maps to the request. Once the handler method completes, the DispatcherServlet will examine the returned value and locate the appropriate view and pass the model the handler method prepared to the view’s rendering method.

However, it would be bad design to include all code to handle all the request routing logic in the DispatcherServlet itself. Furthermore, there is infinite number[[1]](#footnote--1) of combinations for the arguments of the handler methods. Additionally, the handler methods should be able to signal how the rest of the framework should continue processing the request. The DispatcherServlet therefore needs these components:

* HandlerMapping  
  Finds handler chain that could handle the request (typically the “controller” beans); it maps the requests to a chain of handlers that could contain handler methods that may ultimately handle the request.
* HandlerAdapter  
  Presents a unified interface to allow the DispatcherServlet to invoke the handler method; it adapts the varying handler method signatures, thus allowing the DispatcherServlet to execute any handler method.
* ViewResolver  
  Allows the DispatcherServlet to locale an appropriate View interface implementation if the handler method does not return the ModelAndView object with its view property is not null; if all that the DispatcherServlet has is a view name (we will explore how it may obtain it later on), it will use the ViewResolver to resolve the name into a View).

Request routing

1. Really, there are infinitely many combinations of the handler methods’ arguments. For every n possible combinations you give me, I can always give you n + 1 combination. ■ [↑](#footnote-ref--1)