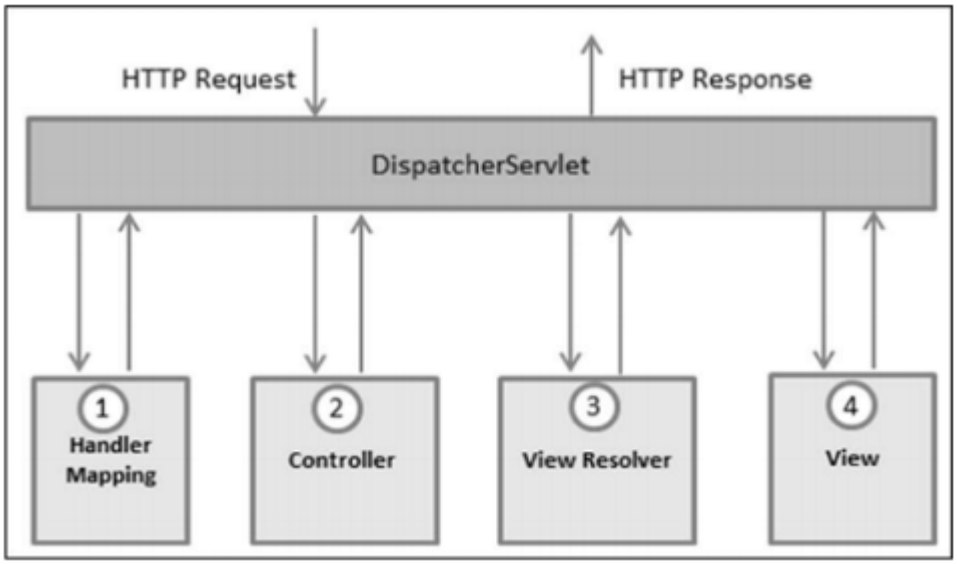
مروری بر MVC Framework

Spring Web MVC framework با توجه به معماری مدل – نمایش – کنترلر و اجزای آماده ای که دارد، با کمک آن می توان برنامه های تحت وب منعطف و با وابستگی کمی را توسعه داد. با استفاده از الگوی MVC می توان جنبه های مختلف برنامه (منطق ورودی، منطق تجاری و منطق رابط کاربری) را از هم تفکیک کرد و در عین حال جفت شدگی این عناصر را به حداقل رساند.

- Model اطلاعات برنامه را در بر می گیرد و در کل متشکل از POJO است.
- View مسئول نمایش اطلاعات مدل و در کل تولید خروجی ای است که مرورگر کلاینت می تواند آن را تفسیر کند.
- Controller مسئول پردازش درخواست های کاربر و ساختن مدل مناسب و دادن آن به view برای نمایش است.

DispatcherServlet

Spring Web model-view-controller (MVC) framework پیرامون DispatcherServlet طراحی شده است که تمامی پاسخ ها و درخواست های HTTP را مدیریت می کند. جریان کار پردازش درخواست Spring Web MVC DispatcherServlet را می توانید در شکل زیر مشاهده کنید.



در ادامه می توانید دنباله ی رویدادهای مربوط به یکی از درخواست های HTTP ورودی به DispatcherServlet را مشاهده کنید:

- DispatcherServlet پس از دریافت یکی از درخواست های HTTP به HandlerMapping می گوید تا کنترلر مناسب را فراخوانی کند.
- کنترلر این درخواست را می گیرد و بر اساس متد GET یا POST، متدهای سرویس مناسب را فراخوانی می کند. این متد سرویس بر اساس منطق تجاری تعریف شده داده های مدل را تنظیم می کند و اسم view را به DispatcherServlet برمی گرداند.
- DispatcherServlet برای برداشتن view تعریف شده ی این درخواست از ViewResolver کمک می گیرد.
- DispatcherServlet بعد از نهایی شدن view داده های مدل را به view می دهد که خود درنهایت بر روی مرورگر رندر می شود.

تمامی مؤلفه های بیان شده در بالا، یعنی HandlerMapping، Controller و ViewResolver، بخش هایی از WebApplicationContext هستند که خود افزونه ای برای ApplicationContext ساده بوده که برخی از امکانات اضافی لازم برای برنامه های وب به آن اضافه شده است.

پیکربندی لازم

ما باید درخواست هایی که DispatcherServlet قرار است به آن ها رسیدگی کند را نگاشت کنیم که این کار با استفاده از نگاشت یک URL در فایل web.xml انجام می شود. در مثال زیر اعلان و نگاشت HelloWeb DispatcherServlet نشان داده شده است.

```
<web-app id = "WebApp_ID" version = "2.4"
  xmlns = "http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
```

```
 <display-name>Spring MVC Application</display-name>


 <servlet>
  <servlet-name>HelloWeb</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
 </servlet>


 <servlet-mapping>
  <servlet-name>HelloWeb</servlet-name>
  <url-pattern>*.jsp</url-pattern>
 </servlet-mapping>
</web-app>
```

فایل web.xml در دایرکتوری WebContent/WEB-INF برنامه ی وب شما نگهداری می شود. بعد از مقداردهی اولیه ی HelloWeb DispatcherServlet ، فریمورک سعی می کند زمینه ی برنامه را از فایلی به نام [servlet-name]-servlet.xml را واقع در دایرکتوری WebContent/WEB-INF برنامه بارگذاری کند. در این حالت فایل ما HelloWeb-servlet.xml خواهد بود.

سپس تگ <servlet-mapping> نشان می دهد که کدام یک از URL ها قرار است توسط کدام یک از DispatcherServlet ها مدیریت شود. در اینجا تمامی درخواست های HTTP ای که با jsp. تمام می شوند، توسط HelloWeb DispatcherServlet مدیریت خواهند شد.

اگر با اسم پیش فرض [servlet-name]-servlet.xml و مکان پیش فرض WebContent/WEB-INF راحت نیستید، می توانید اسم و مکان این فایل را با اضافه کردن شنونده ی سرولت ContextLoaderListener موجود در فایل web.xml به دلخواه خود تغییر دهید، مانند زیر:

```
<web-app…>

 <!-------- DispatcherServlet definition goes here----->
 ….
 <context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/HelloWeb-servlet.xml</param-value>
 </context-param>

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

حالا بیایید پیکربندی لازم برای فایل HelloWeb-servlet.xml را که در دایرکتوری WebContent/WEB-INF برنامه وجود دارد، چک کنیم.

```
<beans xmlns = "http://www.springframework.org/schema/beans"
 xmlns:context = "http://www.springframework.org/schema/context"
 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-3.0.xsd

   http://www.springframework.org/schema/context

   http://www.springframework.org/schema/context/spring-context-3.0.xsd">


   <context:component-scan base-package = "com.tutorialspoint" />


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

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

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

   </bean>


</beans>
```

در ادامه برخی از نکات مهم درباره ی فایل HelloWeb-servlet.xml را می توانید مشاهده کنید:

- با استفاده از فایل [servlet-name]-servlet.xml می توان beans تعریف شده را ایجاد کرد و تعاریف تمامی beans های تعریف شده با اسم یکسان را در محدوده ی سراسری اوررایدکرد.
- با استفاده از تگ <context:component-scan...> می توان قابلیت اسکن حاشیه نویسی Spring MVC را فعال کرد و از این طریق از حاشیه نویسی هایی مثل @Controller و @RequestMapping و ... استفاده کرد.
- InternalResourceViewResolver جهت حل و فصل اسم view ها مقررات تعیین شده ای خواهد داشت. همان طور که در مقررات تعریف شده در بالا مشخص است، یک view منطقی به نام hello به پیاده سازی view واقع در /WEB-INF/jsp/hello.jsp سپرده می شود.

بیایید به چگونگی ایجاد اجزای واقعی یعنی کنترلر ، مدل و view بپردازیم.

تعریف یک کنترلر

DispatcherServlet درخواست را به کنترلرها می سپارد تا قابلیت مختص به آن را اجرا کنند. حاشیه نویسی @Controller نشان می دهد که کلاس خاصی نقش یک کنترلر را ایفا می کند. حاشیه نویسی @RequestMapping در نگاشت یک URL به کل کلاس یا متد هندلر مشخص کاربرد دارد.

```
@Controller
@RequestMapping("/hello")
public class HelloController{


  @RequestMapping(method = RequestMethod.GET)
  public String printHello(ModelMap model) {
    model.addAttribute("message", "Hello Spring MVC Framework!");
    return "hello";
  }


}
```

حاشیه نویسی @Controller کلاس را به صورت یک کنترلر Spring MVC تعریف می کند. در اینجا اولین کاربرد @RequestMapping نشان می دهد که تمامی متدهای handling این کنترلر به مسیر /hello وابسته هستند.

حاشیه نویسی بعدی یعنی @RequestMapping (method = RequestMethod.GET) در اعلان متد printHello() کاربرد دارد. این کاربرد شبیه به کاربرد متد سرویس پیش فرض کنترلر برای رسیدگی به درخواست HTTP GET است. می توانیم برای رسیدگی به درخواست های POST در همین URL متد دیگری را تعریف کنیم.

همچنین می توانیم کنترلر بالا را به صورت دیگری بنویسیم. به این صورت که صفات جدیدی را مانند زیر به @RequestMapping اضافه کنیم.

```
@Controller

public class HelloController{


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

  public String printHello(ModelMap model) {

    model.addAttribute("message", "Hello Spring MVC Framework!");

    return "hello";

  }


}
```

صفت value بیانگر URL ای است که متد هندلر به آن نگاشت شده است و صفت method، متد سرویسی را تعریف می کند که کار آن رسیدگی به درخواست HTTP GET است.

در ادامه برخی از نکات مهم مربوط به کنترلر تعریف شده در بالا را می توانید مشاهده کنید:

- منطق تجاری لازم داخل متد سرویس تعریف می شود. می توانید متد دیگری را داخل این متد و در صورت نیاز فراخوانی کنید.
- بر اساس منطق تجاری تعریف شده، شما مدلی را داخل این متد ایجاد می کنید و صفت های مدل مختلفی را تنظیم کرده و به این صفات می توانید توسط view دسترسی پیدا کنید تا در نهایت نتیجه ارائه شود. در این مثال مدلی به همراه صفت "message" آن ایجاد می شود.
- متدهای سرویس تعریف شده می توانند رشته ای را برگشت دهند که شامل اسم view ای است که قرار است مدل را نمایش دهد. در این مثال "hello" به عنوان اسم منطقی view برگشت داده می شود.

ایجاد JSP View

Spring MVC از انواع زیادی از view ها به ازای فناوری های نمایشی مختلف پشتیبانی می کند که از جمله ی آن ها می توان به JSPs, HTML, PDF, Excel Worksheets, XML, Velocity Templates, XSLT, JSON, Atom و RSS feeds, JasperReports و ... اشاره کرد. با این حال رایج ترین آن ها قالب های JSP ای هستند که به کمک JSTL نوشته شده اند. با این حساب بیایید در /WEB-INF/hello/hello.jsp یک ویوی hello ساده بنویسیم.

```
<html>
  <head>
    <title>Hello Spring MVC</title>
  </head>
  <body>
    <h2>${message}</h2>
  </body>
</html>
```

در اینجا ${message} صفتی است که ما داخل کنترلر قرار داده ایم. می توانید صفات مختلفی را داخل view خود نمایش دهید.