**1. Overview**

In this article, we’ll focus on what is handler adapter and various handler adapters implementations available in the Spring framework.

## ****2. What is a****HandlerAdapter****?****

The *HandlerAdapter*is basically an interface which facilitates the handling of HTTP requests in a very flexible manner in Spring MVC.

It’s used in conjunction with the *HandlerMapping*, which maps a method to a specific URL.

**The *DispatcherServlet*then uses a *HandlerAdapter*to invoke this method. The servlet doesn’t invoke the method directly – it basically serves as a bridge between itself and the handler objects, leading to a loosely coupling design.**

Let’s take a look at various methods available in this interface:

public interface HandlerAdapter {

    boolean supports(Object handler);

    ModelAndView handle(

      HttpServletRequest request,

      HttpServletResponse response,

      Object handler) throws Exception;

    long getLastModified(HttpServletRequest request, Object handler);

}

The *supports* API is used to check if a particular handler instance is supported or not. This method should be called first before calling the *handle()*method of this interface, in order to make sure whether the handler instance is supported or not.

The *handle* API is used to handle a particular HTTP request. This method is responsible for invoking the handler by passing the *HttpServletRequest*and *HttpServletResponse* object as the parameter. The handler then executes the application logic and returns a *ModelAndView* object, which is then processed by the *DispatcherServlet*.

SimpleServletHandlerAdapter

Handler

Adapter

(HandlerAdapter)Passing

HttpServletRequestand,

HttpServletResponse

Receiving from SSHA,RMHA

ModelAndView

RequestMappingHandlerAdapter

Passing

HttpServletRequestand,

HttpServletResponse

Receiving

ModelAndView

Dispatcher

Servlet

(HandlerAdapter) Passing

HttpServletRequestand,

HttpServletResponse

Receiving from HRHA

Void

HttpRequestHandlerAdapter

Lets have one example for RequestMappingHandlerAdapter

|  |
| --- |
| @Controller  public class RequestMappingHandler {        @RequestMapping("/requestName")      public ModelAndView getEmployeeName() {          ModelAndView model = new ModelAndView("Greeting");          model.addObject("message", "Madhwal");          return model;      }  } |

As stated in above figure Dispatcher Servlet neither directly communicate with **RequestMappingHandlerAdapter** nor **other HandlerMappings** because RequestMappingHandlerAdapter returns ModelAndView and The return type of HttpRequestHandlerAdapter method is void ,it doesn’t generate ModelAndView.

The handler then executes the application logic communicate with handlers and returns a ModelAndView object as contracted with Dispathcher servlet. If handler received ModelAndView then simply send to to Dispathcher servlet else it creates the same do some logic and send to the Dispathcher servlet.

## ****Below are some in-depth explanation and types of HandlerAdapter (can be skipped if not required)****

## ****3. Maven Dependency****

Let’s start with the Maven dependency that needs to be added to *pom.xml*:

|  |  |
| --- | --- |
|  | <dependency>      <groupId>org.springframework</groupId>      <artifactId>spring-webmvc</artifactId>      <version>4.3.4.RELEASE</version>  </dependency> |

## ****4. Types of HandlerAdapter****

### **4.1.**SimpleControllerHandlerAdapter

This is the default handler adapter registered by Spring MVC. It deals with classes implementing *Controller* interface and is used to forward a request to a controller object.

If a web application uses only controllers then we don’t need to configure any *HandlerAdapter* as the framework uses this class as the default adapter for handling a request.

Let’s define a simple controller class, using the older style of controller (implementing the *Controller* interface):

|  |  |
| --- | --- |
|  | public class SimpleController implements Controller {      @Override      public ModelAndView handleRequest(        HttpServletRequest request,        HttpServletResponse response) throws Exception {            ModelAndView model = new ModelAndView("Greeting");          model.addObject("message", "Dinesh Madhwal");          return model;      }  } |

The similar XML configuration:

|  |  |
| --- | --- |
|  | <beans ...>      <bean name="/greeting.html"        class="com.baeldung.spring.controller.SimpleControllerHandlerAdapterExample"/>      <bean id="viewResolver"        class="org.springframework.web.servlet.view.InternalResourceViewResolver">          <property name="prefix" value="/WEB-INF/" />          <property name="suffix" value=".jsp" />      </bean>  </beans> |

The *BeanNameUrlHandlerMapping* class is the mapping class for this handler adapter.

**Note**: If a custom handler adapter is defined in *BeanFactory,*then this adapter is not automatically registered. Thus, we need to define it explicitly in the context. If it is not defined and we have defined a custom handler adapter, then we will get an exception that says that no adapter for a handler is specified.

### **4.2.**SimpleServletHandlerAdapter

This handler adapter allows the use of any *Servlet* to work with *DispatcherServlet*for handling the request. It forwards the request from *DispatcherServlet*to the appropriate *Servlet* class by calling its *service()*method*.*

The beans which implement the *Servlet*interface are automatically handled by this adapter. It is not registered by default and we need to register it like any other normal bean in the configuration file of *DispatcherServlet*:

|  |  |
| --- | --- |
|  | <bean name="simpleServletHandlerAdapter"    class="org.springframework.web.servlet.handler.SimpleServletHandlerAdapter" /> |

### **4.3.**AnnotationMethodHandlerAdapter

This adapter class is used to execute the methods that are annotated with *@RequestMapping*annotation. It is used to map the methods based on HTTP methods and HTTP paths.

The mapping class for this adapter is *DefaultAnnotationHandlerMapping,*which is used to process the *@RequestMapping* annotation at the type level and *AnnotationMethodHandlerAdaptor*is used to process at a method level.

These two classes are already registered by the framework when the *DispatcherServlet* is initialized. However, if the other handler adapters are already defined, then we need to define it as well in the configuration file.

Let’s define a controller class:

|  |  |
| --- | --- |
|  | @Controller  public class AnnotationHandler {      @RequestMapping("/annotedName")      public ModelAndView getEmployeeName() {          ModelAndView model = new ModelAndView("Greeting");          model.addObject("message", "Dinesh");          return model;      }  } |

The *@Controller* annotation indicates that this class serves the role of *controller.*

The *@RequestMapping*annotation maps the *getEmployeeName()* method to the URL */name.*

There are 2 different ways of configuring this adapter depending on whether the application uses Java based configuration or XML based configuration. Let us look at the first way using Java configuration:

|  |  |
| --- | --- |
|  | @ComponentScan("com.baeldung.spring.controller")  @Configuration  @EnableWebMvc  public class ServletConfig extends WebMvcConfigurerAdapter {      @Bean      public InternalResourceViewResolver jspViewResolver() {          InternalResourceViewResolver bean = new InternalResourceViewResolver();          bean.setPrefix("/WEB-INF/");          bean.setSuffix(".jsp");          return bean;      }  } |

If the application uses XML configuration, then there are two different approaches for configuring this handler adapter in web application context XML. Let us take a look at the first approach defined in the file *spring-servlet\_AnnotationMethodHandlerAdapter.xml*:

|  |  |
| --- | --- |
|  | <beans ...>      <context:component-scan base-package="com.baeldung.spring.controller" />      <bean        class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"/>      <bean        class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter"/>      <bean id="viewResolver"        class="org.springframework.web.servlet.view.InternalResourceViewResolver">          <property name="prefix" value="/WEB-INF/" />          <property name="suffix" value=".jsp" />      </bean>  </beans> |

The *<context:component-scan />* tag is used to specify the package to scan for *controller* classes.

Let us take a look at the second approach:

|  |  |
| --- | --- |
|  | <beans ...>      <mvc:annotation-driven/>      <context:component-scan base-package="com.baeldung.spring.controller" />      <bean id="viewResolver"        class="org.springframework.web.servlet.view.InternalResourceViewResolver">          <property name="prefix" value="/WEB-INF/" />          <property name="suffix" value=".jsp" />      </bean>  </beans> |

The *<mvc:annotation-driven>* tag will automatically register these two classes with spring MVC. This adapter was deprecated in Spring 3.2 and a new handler adapter called *RequestMappingHandlerAdapter*was introduced in Spring 3.1.

### **4.4.**RequestMappingHandlerAdapter

This adapter class was introduced in Spring 3.1, deprecating the *AnnotationMethodHandlerAdaptor*handler adapter in Spring 3.2.

It’s used with *RequestMappingHandlerMapping*class, which **executes methods annotated with***@RequestMapping*.

The *RequestMappingHandlerMapping* is used to maintain the mapping of the request URI to the handler. Once the handler is obtained, the *DispatcherServlet*dispatches the request to the appropriate handler adapter, which then invokes the *handlerMethod().*

The type-level and method-level mappings were processed in two different stages in the Spring version prior to 3.1.

The first stage was to select the controller by *DefaultAnnotationHandlerMapping*and the second stage was to invoke the actual method by *AnnotationMethodHandlerAdapter*.

From Spring version 3.1, there is only one stage, which involves identifying the controller as well as which method needs to be invoked to process the request.

Let’s define a simple controller class:

|  |  |
| --- | --- |
|  | @Controller  public class RequestMappingHandler {        @RequestMapping("/requestName")      public ModelAndView getEmployeeName() {          ModelAndView model = new ModelAndView("Greeting");          model.addObject("message", "Madhwal");          return model;      }  } |

There are 2 different ways of configuring this adapter depending on whether the application uses Java based configuration or XML based configuration.

Let’s look at the first way using Java configuration:

|  |  |
| --- | --- |
|  | @ComponentScan("com.baeldung.spring.controller")  @Configuration  @EnableWebMvc  public class ServletConfig extends WebMvcConfigurerAdapter {      @Bean      public InternalResourceViewResolver jspViewResolver() {          InternalResourceViewResolver bean = new InternalResourceViewResolver();          bean.setPrefix("/WEB-INF/");          bean.setSuffix(".jsp");          return bean;      }  } |

If the application uses XML configuration, then there are two different approaches for configuring this handler adapter in web application context XML. Let us take a look at the first approach defined in the file *spring-servlet\_RequestMappingHandlerAdapter.xml*:

|  |  |
| --- | --- |
|  | <beans ...>      <context:component-scan base-package="com.baeldung.spring.controller" />        <bean        class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>        <bean        class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/>        <bean id="viewResolver"        class="org.springframework.web.servlet.view.InternalResourceViewResolver">          <property name="prefix" value="/WEB-INF/" />          <property name="suffix" value=".jsp" />      </bean>  </beans> |

And here’s the second approach:

|  |  |
| --- | --- |
|  | <beans ...>      <mvc:annotation-driven />        <context:component-scan base-package="com.baeldung.spring.controller" />        <bean id="viewResolver"        class="org.springframework.web.servlet.view.InternalResourceViewResolver">          <property name="prefix" value="/WEB-INF/" />          <property name="suffix" value=".jsp" />      </bean>  </beans> |

This tag will automatically register these two classes with Spring MVC.

If we need to customize the *RequestMappingHandlerMapping,*then we need to remove this tag from the application context XML and manually configure it in the application context XML.

### **4.5.**HttpRequestHandlerAdapter

This handler adapter is used for the handlers that process *HttpRequest*s. It implements the *HttpRequestHandler*interface, which contains a single *handleRequest()*method for processing the request and generating the response.

The return type of this method is void and it doesn’t generate *ModelAndView*return type as produced by other handler adapters. It’s basically used to generate binary responses and it doesn’t generate a view to render.