**HANDLER ADAPTER:**

**HandlerAdapter**

HandlerAdapter is responsible for actually invoking the handler. The handler is of type Object and hence the dispatcher servlet can handle any Handler type using the HandlerAdapter. Spring provides certain HandlerAdapters that can handle specific handler interfaces.

**SimpleServletHandlerAdapter**

This adapter allows using the Servlet Interface. The adapter calls the service method of the Servlet. This adapter is not registered by default and needs to be registered in the DispatcherServlet as a bean. All handler beans that implement the Servlet interface are automatically handled by this adapter.

**SimpleControllerHandlerAdapter**

Adapter for handlers that implement the Controller interface

**HttpRequestHandlerAdapter**

Handles org.springframework.web.HttpRequestHandler interface. The HttpRequestHandler is used for handler that process HttpRequests. This interface is generally used to generate binary responses and does not have a ModelAndView return type. The spring remoting classes such as HttpInvokerServiceExporter and HessianServiceExporter implement this interface.

**AnnotationMethodHandlerAdapter**

This adapter maps Handler methods based on HTTP paths, HTTP methods and request parameters. The request parameters are bound through @RequestParam annotation. Note that this Adapater is deprecated in spring 3.2 in favour of RequestMappingHandlerAdapter.

**RequestMappingHandlerAdapter**

This adapter supports Handler of type HandlerMethod. This adapter is used with RequestMappingHandlerMapping and is the new class introduced in Spring 3.1. custom Resolvers can be set using

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public void setCustomArgumentResolvers(List<handlermethodargumentresolver> argumentResolvers) {          this.customArgumentResolvers = argumentResolvers;  }        </handlermethodargumentresolver> |

**HANDLER MAPPING INTERFACE:**

**HandlerMapping**

HandlerMapping is an interface that is implemented by all Objects that map the request to the corresponding Handler Object. A HandlerExecutionChain always wraps the Handler and may also contain one or more HandlerInterceptors. The default Implementations used by the DispatcherServlet are BeanNameUrlHandlerMapping and DefaultAnnotationHandlerMapping.

**BeanNameUrlHandlerMapping**

This is the default implemenatation used by DispatcherServlet. This maps a URL to a bean name. for example the URL "/app" would map to a Handler named "/app". This also supports "\*" wildcard mapping, so "/a\*" matches both "/app" and "/apple". The class does and ant style matching so "/app/\*" matches all paths in the app directory and "/app/\*\*" matches all paths below app.

**DefaultAnnotationHandlerMapping**

Mappings declared via the @RequestMapping annotation are mapped via this Handler. The annotation may be expressed at the method level or type level. This mapping is registered by default, however if you register a custom Handler then the defaults are overridden. @RequestMappings may be defined at the type level and the method level. At the type level they map requests to the handler bean. The method level mappings help in narrowing requests. To identify that a class is a Controller use the @Controller annotation at type level. However if you have used @RequestMapping at the type level then the @Controller annotation may be ommited. However, if the @RequestMapping annotation is present at the method level and not at the type level then the annotation is required. The mappings expressed at the method level are processed via the AnnotationMethodHandlerAdapter and the mappings expressed at the type level are processed via specific Handlers.

**ControllerBeanNameHandlerMapping**

This is similar to BeanNameUrlHandlerMapping in that it mapps a URL path to a bean name by looking at the registered Controller beans or annotated beans. However, unlike BeanNameUrlHanlderMapping, the bean name may not be a URL. for example, it can map the URL "/app" to a Handler named "app". You can also specify a prefix and a suffix to the bean name to map the URL.

**ControllerClassNameHandlerMapping**

Here the mapping is generated by by comparing the URL path to the class name of the registered or annotated controllers. For simple controllers, the logic is to use ClassUtils.getShortName to obtain the class name, remove 'Controller' suffix if present, convert the remaining string to lower case and add '/' at the beginning. so LoginController maps to /login\*

**SimpleUrlHandlerMapping**

This class maps URL to Handler classes using an xml based "urlMap" map property. The key of the map is the URL and the value is the handler bean name. The mappings can also be passed using the "mappings" property in such a way that the java.util.Properties class can read it. the key can use a pattern instead of exact path. The pattern should follow org.springframework.util.AntPathMatcher.

**RequestMappingHandlerMapping**

This was introduced in spring 3.1. Earlier the DefaultAnnotationHandlerMapping decided which controller to use and the AnnotationMethodHandlerAdapter selected the actual method that handled the request. RequestMappingHandlerMapping does both the tasks. Therefore the request is directly mapped right to the method. The following types can be passed to method handlers.

1. org.springframework.web.bind.support.SessionStatus - Signals that the session processing is complete. (SessionStatusMethodArgumentResolver)
2. java.util.Map arguments annotated with @RequestParam but where the request parameter name is not specified. (RequestParamMapMethodArgumentResolver)
3. java.util.Map arguments annotated with @RequestHeader. The map contains all the name-value pairs from the header. (RequestHeaderMapMethodArgumentResolver)
4. org.springframework.ui.Model argument as well as return values. This handler should be configured before configuring any other handler that supports return type annotated with @ModelAttribute or @ResponseBody. (ModelMethodProcessor)
5. org.springframework.web.bind.annotation.ModelAttribute argument as well as return value. (ModelAttributeMethodProcessor)
6. java.util.Map argument and return types. This handler should be registered after handlers that support @ModelAttribute or @ResponseBody. (MapMethodProcessor)
7. org.springframework.validation.Errors. This should be passed immediately after the model attribute. It expects that the last two variables in the model be the model attribute and its org.springframework.validation.BindingResult.(ErrorsMethodArgumentResolver)
8. org.springframework.web.bind.annotation.CookieValue. (AbstractCookieValueMethodArgumentResolver)
9. org.springframework.beans.factory.annotation.Value. (ExpressionValueMethodArgumentResolver)
10. org.springframework.web.bind.annotation.RequestHeader (RequestHeaderMethodArgumentResolver)
11. org.springframework.web.bind.annotation.RequestParam (RequestParamMethodArgumentResolver)
12. org.springframework.web.util.UriComponentsBuilder. This is a builder for URIComponents which maps URI components to String values. (UriComponentsBuilderMethodArgumentResolver)
13. ServletResponse, java.io.OutputStream and java.io.Writer (ServletResponseMethodArgumentResolver)
14. org.springframework.web.context.request.WebRequest, ServletRequest, MultipartRequest, HttpSession, java.security.Principal, Locale, java.io.InputStream, java.io.Reader (ServletRequestMethodArgumentResolver)
15. org.springframework.web.servlet.mvc.support.RedirectAttributes. This is an extension of the Model and has attributes for a redirect scenario (RedirectAttributesMethodArgumentResolver)
16. java.util.Map argument annotated with @PathVariable where the path variable name is not specified. the map contains the uri template name,value pairs. (PathVariableMapMethodArgumentResolver). e.g. consider a path uri "GET /app;a=10;b=11".

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @RequestMapping(value = "/app", method = RequestMethod.GET)      public void findPet(@MatrixVariable int a,@MatrixVariable int b) {      // a == 10      // b == 11  } |

1. org.springframework.web.bind.annotation.MatrixVariable. The path segment may have name-value pairs. The MatrixVariable maps such name-value pairs within a path segment.