

Spring MVC

- ❑ The Spring Web model-view-controller (MVC) framework is designed around a **DispatcherServlet** that dispatches requests to handlers and then to view resolver for view resolution.
- ❑ The default handler is based on the **@Controller** and **@RequestMapping** annotations, offering a wide range of flexible handling methods.
- ❑ Annotation **@PathVariable** provides support for restful web services.

DispatcherServlet

- ❑ DispatcherServlet is a servlet configured in web.xml
- ❑ It uses special beans to process request and render views
- ❑ It supports the following init-param elements for configuration
- ❑ By default it look for a file with name {servlet-name}-servlet.xml

contextClass

Class that implements `WebApplicationContext`, which instantiates the context used by this Servlet. By default, the `XmlWebApplicationContext` is used.

contextConfigLocation

String that is passed to the context instance to indicate where context(s) can be found. The string consists potentially of multiple strings (using a comma as a delimiter) to support multiple contexts. In case of multiple context locations with beans that are defined twice, the latest location takes precedence.

namespace

Namespace of the `WebApplicationContext`. Defaults to `[servlet-name]-servlet`.

Special Beans

HandlerMapping

Maps incoming requests to handlers and a list of pre- and post-processors (handler interceptors) based on some criteria the details of which vary by HandlerMapping implementation.

ViewResolver

Resolves logical String-based view names to actual View types.

LocaleResolver

Resolves the locale a client is using and possibly their time zone, in order to be able to offer internationalized views

ThemeResolver

Resolves themes your web application can use

MultipartResolver

Parses multi-part requests for example to support processing file uploads from HTML forms.

FlashMapManager

Stores and retrieves the "input" and the "output" FlashMap that can be used to pass attributes from one request to another, usually across a redirect.

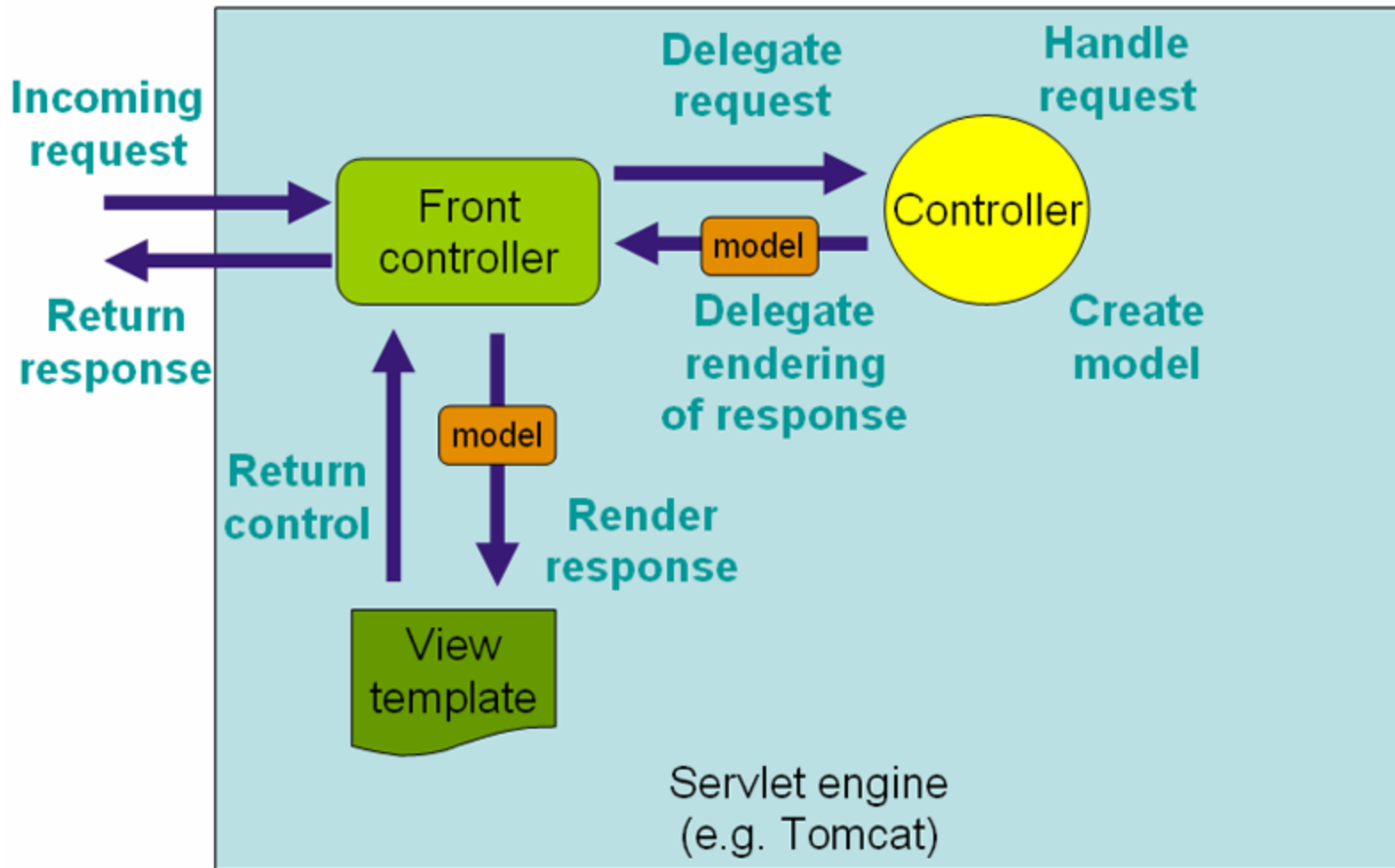
DispatcherServlet Process

- ☐ The `WebApplicationContext` is searched for and bound in the request as an attribute that the controller and other elements in the process can use. It is bound by default under the key `DispatcherServlet.WEB_APPLICATION_CONTEXT_ATTRIBUTE`.
- ☐ The locale resolver is bound to the request to enable elements in the process to resolve the locale to use when processing the request
- ☐ The theme resolver is bound to the request to let elements such as views determine which theme to use.
- ☐ If you specify a multipart file resolver, the request is inspected for multipart; if multipart is found, the request is wrapped in a `MultipartHttpServletRequest` for further processing by other elements in the process.
- ☐ An appropriate handler is searched for. If a handler is found, the execution chain associated with the handler (preprocessors, postprocessors, and controllers) is executed in order to prepare a model or rendering.
- ☐ If a model is returned, the view is rendered. If no model is returned, no view is rendered, because the request could already have been fulfilled.

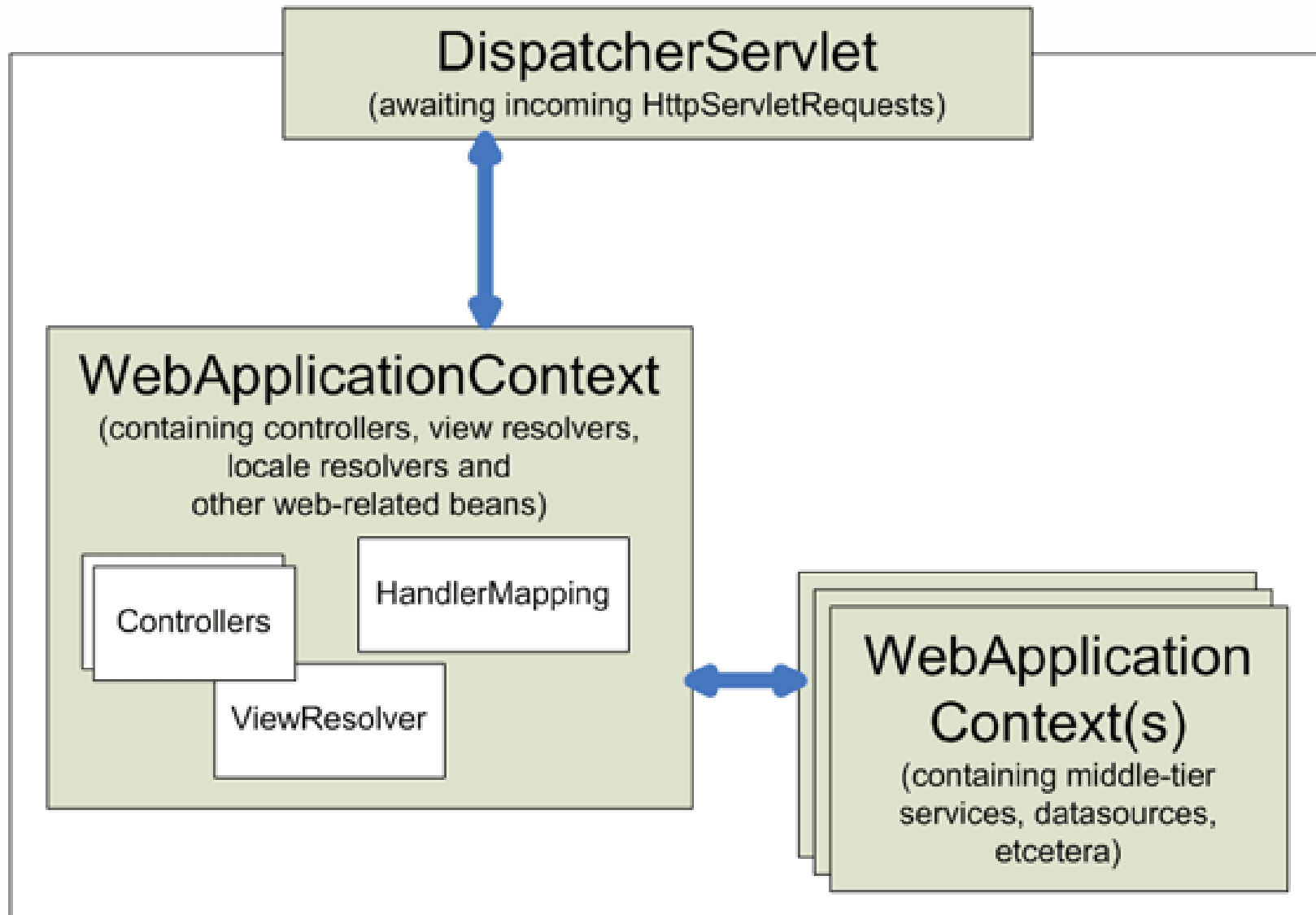
Spring MVC Components

- ❑ A **Controller** is typically responsible for preparing a model Map with data and selecting a view name.
- ❑ View name resolution is handled by **View Resolver**.
- ❑ The **model** is a **Map** interface, which allows for the complete abstraction of the view technology.
- ❑ You can integrate directly with template based rendering technologies such as JSP, Velocity and Freemarker.
- ❑ The model Map is simply transformed into an appropriate format, such as JSP request attributes.
- ❑ It is also possible to generate XML or JSON and send it to client.

Request Processing Workflow



Context Hierarchy



Controller

- ❑ Controllers provide access to the application behavior
- ❑ Controllers interpret user input and transform it into a model that is represented to the user by the view.
- ❑ A controller is a Spring bean defined using **@Controller** annotation.
- ❑ It has methods that are mapped to requests using **@RequestMapping** annotations.
- ❑ We have to place controllers in packages that are specified using **<component-scan>** element in configuration file.
- ❑ Controllers need not extend any class or implement any interface.
- ❑ Methods in controller can have flexible signatures .

@RequestMapping

- ❑ You use the **@RequestMapping** annotation to map URLs such as **/login** onto an entire class or a particular handler method.
- ❑ Typically the class-level annotation maps a specific request path (or path pattern)
- ❑ Additional method-level annotations narrowing the primary mapping for a specific HTTP method request method ("GET", "POST", etc.) or an HTTP request parameter condition
- ❑ It has two properties - value and method.
- ❑ Method can be set to one of the options in RequestMethod enumeration representing HTTP request methods

Controller Example

```
@Controller
@RequestMapping("/books")
public class BooksController {
    @Autowired
    public BooksController(Catalog cat) {
    }
    @RequestMapping(method = RequestMethod.GET)
    public String get() {
        // process
    }
    @RequestMapping(value="/new", method = RequestMethod.GET)
    public Book newBook() {
        // process
    }
    @RequestMapping(method = RequestMethod.POST)
    public String addBook(@Valid Book book, BindingResult result) {
        // process
    }
}
```

@RequestMapping Handler methods

An @RequestMapping handler method can have a very flexible signatures. Most arguments can be used in arbitrary order with the only exception of BindingResult arguments.

@RequestMapping Handler method Parameters

- ❑ Request or response objects (Servlet API).
- ❑ Session object (Servlet API): of type HttpSession.
- ❑ java.io.InputStream / java.io.Reader for access to the request's content.
- ❑ java.io.OutputStream / java.io.Writer for generating the response's content.
- ❑ @PathVariable annotated parameters for access to URI template variables.
- ❑ @RequestParam annotated parameters for access to specific Servlet request parameters
- ❑ @RequestHeader annotated parameters for access to specific Servlet request HTTP headers.
- ❑ @RequestBody annotated parameters for access to the HTTP request body.
- ❑ @RequestPart annotated parameters for access to the content of a "multipart/form-data" request part.
- ❑ java.util.Map / org.springframework.ui.Model /org.springframework.ui.ModelMap for enriching the implicit model that is exposed to the web view.

@RequestMapping Handler method return types

- ☐ A ModelAndView object
- ☐ A Model object
- ☐ A Map object for exposing a model
- ☐ A View object
- ☐ A String value that is interpreted as the logical view name
- ☐ void if the method handles the response itself (by writing the response content directly, declaring an argument of type `ServletResponse` / `HttpServletResponse` for that purpose) or if the view name is supposed to be implicitly determined through a `RequestToViewNameTranslator`.
- ☐ Any other return type is considered to be a single model attribute to be exposed to the view, using the attribute name specified through `@ModelAttribute` at the method level (or the default attribute name based on the return type class name). The model is implicitly enriched with command objects and the results of `@ModelAttribute` annotated reference data accessor methods

@ModelAttribute

- ❑ @ModelAttribute can be used on methods or on method arguments.
- ❑ It indicates the argument should be retrieved from the model. If not present in the model, the argument should be instantiated first and then added to the model.
- ❑ Once present in the model, the argument's fields should be populated from all request parameters that have matching names.
- ❑ This is known as data binding in Spring MVC

```
@RequestMapping ( value="/jobs/edit",  
                  method = RequestMethod.POST)  
public String processSubmit(@ModelAttribute Job job)  
{  
}
```

@PathVariable

- ❑ A URI Template is a URI-like string, containing one or more variable names
Variables are enclosed in {}
- ❑ We can use @PathVariable annotation on a method argument to bind it to the value of a URI template variable

```
@RequestMapping(value="/books/{id}",  
                method=RequestMethod.GET)  
public String findBook(@PathVariable("id") String bookId,  
                      Model model) {  
  
}  
  
@RequestMapping(value="/books/{id}/chapters/{chno}",  
                method=RequestMethod.GET)  
public String findChapter(@PathVariable String id,  
                        @PathVariable String chno,  
                        Model model) {  
  
}
```

Other Annotations related to Request Handler

- ☐ **@SessionAttributes**
- ☐ **@RequestParam**
- ☐ **@CookieValue**
- ☐ **@RequestHeader**
- ☐ **@ModelAttribute**
- ☐ **@RequestPart**
- ☐ **@RequestBody**

Available View Resolvers

ResourceBundleViewResolver

Uses bean definitions in a `ResourceBundle`, specified by the bundle base name. Typically you define the bundle in a properties file, located in the classpath. The default file name is `views.properties`.

UrlBasedViewResolver

Effects the direct resolution of logical view names to URLs, without an explicit mapping definition. This is appropriate if your logical names match the names of your view resources in a straightforward manner, without the need for arbitrary mappings.

InternalResourceViewResolver

Convenient subclass of `UrlBasedViewResolver` that supports `InternalResourceView` (in effect, Servlets and JSPs) and subclasses such as `JstlView` and `TilesView`. You can specify the view class for all views generated by this resolver by using `setViewClass(..)`

Prefixes

redirect : prefix

If a view name is returned that has the prefix `redirect:`, the `UrlBasedViewResolver` (and all subclasses) will recognize this as a special indication that a redirect is needed. The rest of the view name will be treated as the redirect URL.

forward: prefix

It is also possible to use a special `forward:` prefix for view names that are ultimately resolved by `UrlBasedViewResolver` and subclasses. This creates an `InternalResourceView` (which ultimately does a `RequestDispatcher.forward()`) around the rest of the view name, which is considered a URL.

```
@RequestMapping(value = "/files",  
                method = RequestMethod.POST)  
public String upload(...) {  
    // ...  
    return "redirect:files";  
}
```

Spring Form Tags

- ❑ Spring provides comprehensive set of data binding-aware tags for handling form elements
- ❑ Form tags are bundled with `spring-webmvc.jar`
- ❑ Register form tags as follows :

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
```

Spring Form Tags

- ☐ `<form:form commandName="" ">`
- ☐ `<form:input path="" />`
- ☐ `<form:checkbox path="" value="" />`
- ☐ `<form:radio button path="" value="" />`
- ☐ `<form:checkboxes path="" items="" />`
- ☐ `<form:radio buttons path="gender" items="{genderOptions}" />`
- ☐ `<form:select path="skills" items="{skills}" />`
- ☐ `<form:option value="">`
- ☐ `<form:options items="" itemValue="" itemLabel="" />`
- ☐ `<form:textarea path="" rows="" cols="" />`
- ☐ `<form:hidden path="" />`
- ☐ `<form:errors path="" />`

path can be *, field. If omitted then object errors are displayed.