

Application Delivery Fundamentals 2.0 B: Java

Spring MVC



High performance. Delivered.

consulting | technology | outsourcing

Module Objectives

At the end of this section, you should be able to:

- Describe Model View Controller (MVC) Design Pattern
- Explain Spring MVC implementation
- Create a Spring MVC-based application



What You Need

- JDK 1.8 or later
- Gradle 4+ or Maven 3.2+
- You can also import the code straight into your IDE:
 - Spring Tool Suite (STS)
 - IntelliJ IDEA
 - Eclipse 2019

Model View Controller Design Pattern

Model View Controller (MVC): A fundamental design pattern for separating user interface logic from business logic

Separates the domain (Model), the user interface (View), and the actions based on user input (Controller) into three separate categories:

Model

Handles the behavior and data of the application

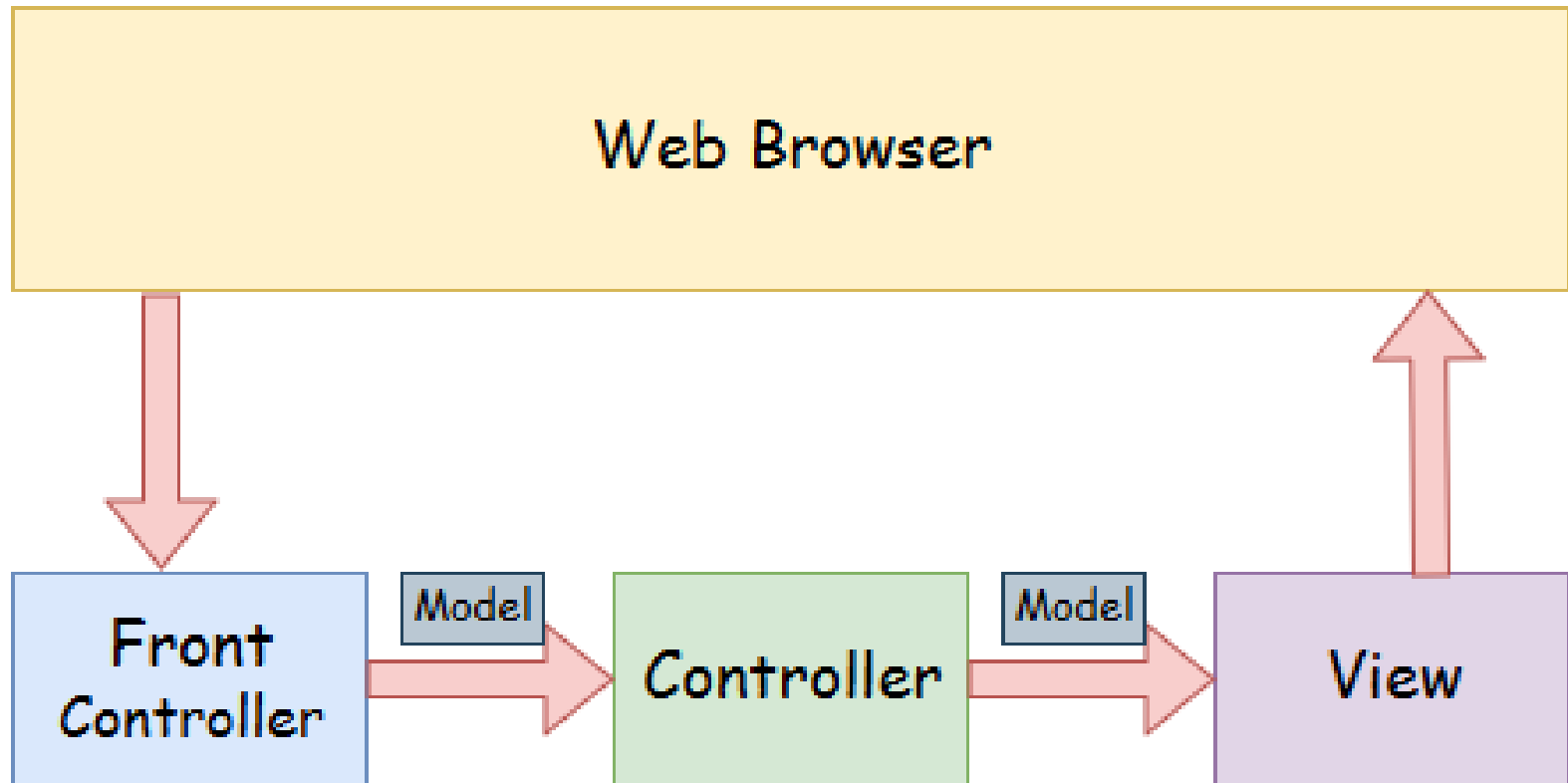
View

Handles the display of information

Controller

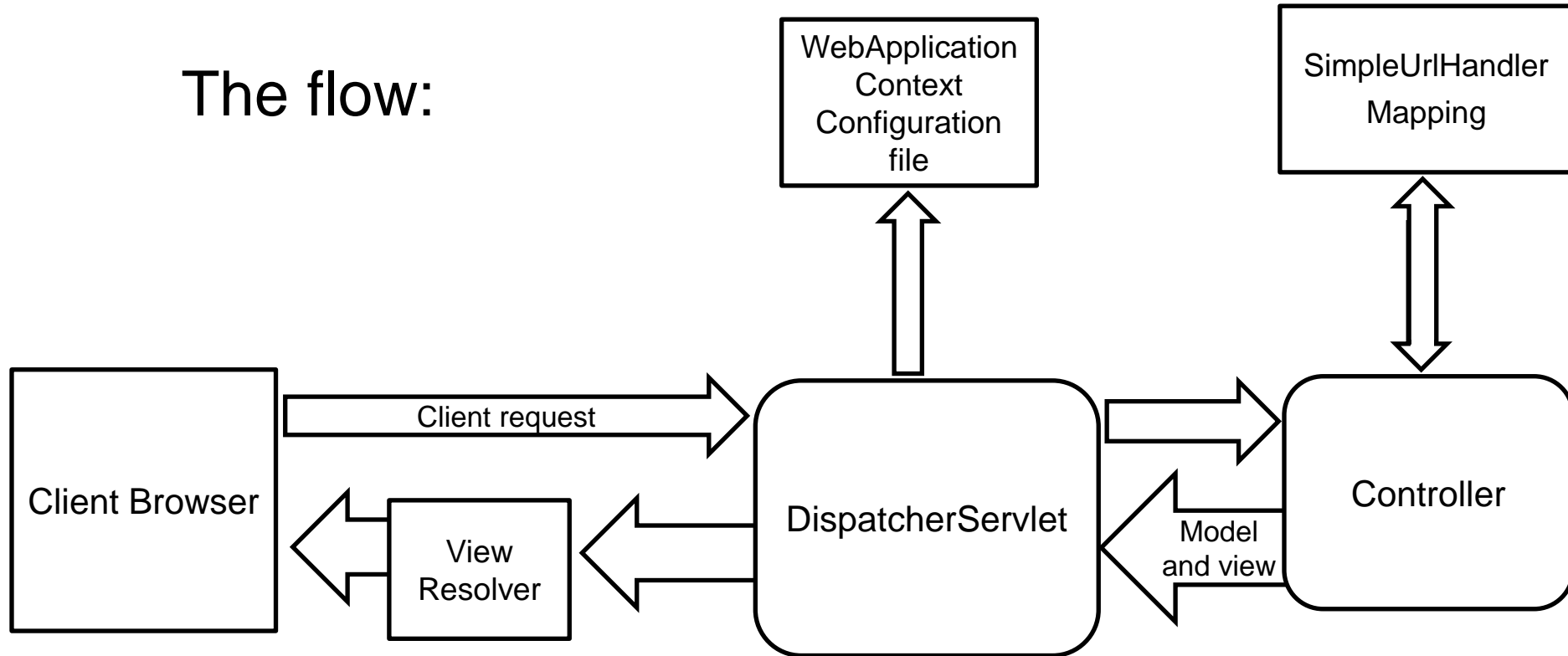
Interprets the user inputs, informing the model and/or the view to change as appropriate

Model View Controller Design Pattern



Spring MVC Flow

The flow:



Spring MVC

Spring @MVC vs. Spring <MVC/>

Earlier versions on Spring (Before 2.5) relied more on xml configurations

Spring 2.5 introduced a simplified, annotation-based model for developing Spring MVC applications informally referred to as “Spring @MVC”

Spring Framework uses annotations like @Controller, @RequestMapping, and so on, to mark a class as a Controller or to map a request to its handler respectively

Dispatcher Servlet

- Spring MVC, as many other web frameworks, is designed around the front controller pattern.
- A central Servlet, the DispatcherServlet, provides a shared algorithm for request processing.
- Actual work is performed by configurable delegate components.
- This model is flexible and supports diverse workflows.
- The DispatcherServlet, as any Servlet, needs to be declared and mapped according to the Servlet specification by using Java configuration or in web.xml.
- In turn, the DispatcherServlet uses Spring configuration to discover the delegate components it needs for request mapping, view resolution, exception handling, and more.

Dispatcher Servlet

```
public class MyWebApplicationInitializer implements WebApplicationInitializer {  
    @Override  
    public void onStartup(ServletContext servletCxt) {  
        // Load Spring web application configuration  
        AnnotationConfigWebApplicationContext ac = new  
AnnotationConfigWebApplicationContext();  
        ac.register(AppConfig.class);  
        ac.refresh();  
        // Create and register the DispatcherServlet  
        DispatcherServlet servlet = new DispatcherServlet(ac);  
        ServletRegistration.Dynamic registration = servletCxt.addServlet("app", servlet);  
        registration.setLoadOnStartup(1);  
        registration.addMapping("/app/*");  
    }  
}
```

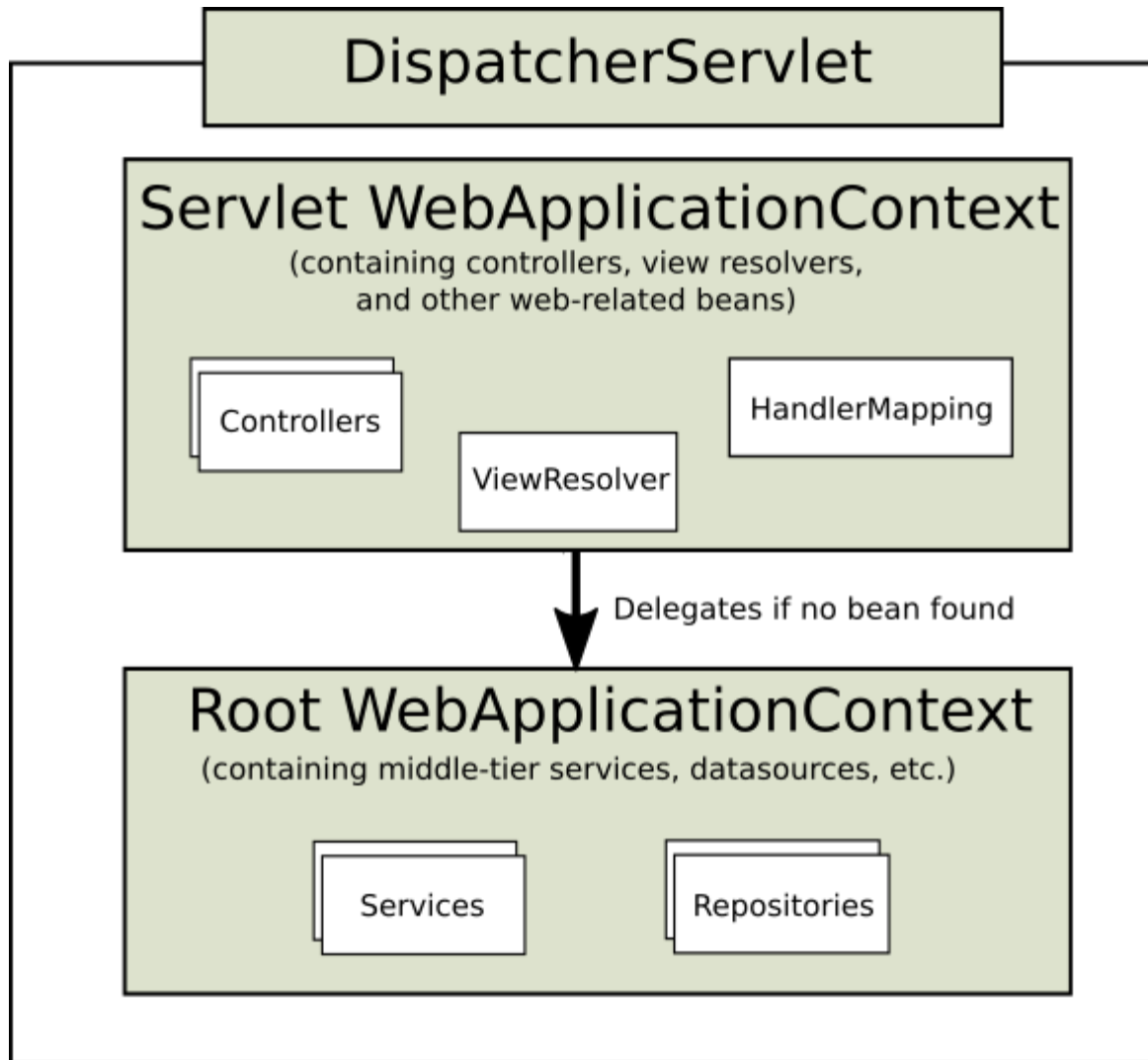
Dispatcher Servlet

```
<web-app>
  <listener>
    <listener-
class>org.springframework.web.context.ContextLoaderListe
ner</listener-class>
  </listener>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/app-context.xml</param-
value>
  </context-param>
```

Dispatcher Servlet

```
<servlet>
  <servlet-name>app</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value></param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>app</servlet-name>
  <url-pattern>/app/*</url-pattern>
</servlet-mapping>
</web-app>
```

Context Hierarchy



Web Application Context Hierarchy

```
public class MyWebAppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class<?>[] { RootConfig.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class<?>[] { App1Config.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/app1/*" };
    }
}
```

JAVA

Dispatcher Servlet Process

- The Dispatcher Servlet processes requests as follows:
- The WebApplication Context 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 `DispatcherServlet.WEB_APPLICATION_CONTEXT_ATTRIBUTE` key.
- The locale resolver is bound to the request to let elements in the process resolve the locale to use when processing the request (rendering the view, preparing data, and so on).
- Locale resolver is optional.
- The theme resolver is bound to the request to let elements such as views determine which theme to use.
- Themes also optional.
- In a multipart file resolver, the request is inspected for multipart.
- If multipart are found, the request is wrapped in a `MultipartHttpServletRequest` for further processing by other elements in the process.

Dispatcher Servlet 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.
- Alternatively, for annotated controllers, the response can be rendered (within the HandlerAdapter) instead of returning a view.
- If a model is returned, the view is rendered.
- If no model is returned (maybe due to a preprocessor or postprocessor intercepting the request, perhaps for security reasons), no view is rendered, because the request could already have been fulfilled.

DispatcherServlet initialization parameters

Parameter	Explanation
contextClass	Class that implements ConfigurableWebApplicationContext, to be instantiated and locally configured by this Servlet. By default, XmlWebApplicationContext is used.
contextConfigLocation	String that is passed to the context instance (specified by contextClass) to indicate where contexts can be found. The string consists potentially of multiple strings (using a comma as a delimiter) to support multiple contexts. In the case of multiple context locations with beans that are defined twice, the latest location takes precedence.

DispatcherServlet initialization parameters

Parameter	Explanation
namespace	Namespace of the WebApplicationContext. Defaults to [servlet-name]-servlet.
throwExceptionIfNoHandlerFound	<p>Whether to throw a <code>NoHandlerFoundException</code> when no handler was found for a request. The exception can then be caught with a <code>HandlerExceptionResolver</code> (for example, by using an <code>@ExceptionHandler</code> controller method) and handled as any others.</p> <p>By default, this is set to false, in which case the <code>DispatcherServlet</code> sets the response status to 404 (<code>NOT_FOUND</code>) without raising an exception.</p>

Spring MVC Config

- The MVC Java configuration and the MVC XML namespace provide default configuration suitable for most applications and a configuration API to customize it.
- `@Configuration`
- `@EnableWebMvc`
- `public class WebConfig {`
- `}`

Spring MVC Config

In XML configuration, you can use the `<mvc:annotation-driven>` element to enable MVC configuration, as the following example shows:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           https://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/mvc
           https://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <mvc:annotation-driven/>

</beans>
```

Validation

- By default, if Bean Validation is present on the classpath (for example, Hibernate Validator), the LocalValidatorFactoryBean is registered as a global Validator for use with @Valid and Validated on controller method arguments.
- In Java configuration, you can customize the global Validator instance, as the following example shows:
- **@Configuration @EnableWebMvc public class WebConfig implements WebMvcConfigurer {**
- **@Override public Validator getValidator()**
- **{ // ... }**
- **}**

Validation

- **<?xml version="1.0" encoding="UTF-8"?>** <beans xmlns="http://www.springframework.org/schema/beans" xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/mvc https://www.springframework.org/schema/mvc/spring-mvc.xsd">
- **<mvc:annotation-driven validator="globalValidator"/>**
</beans>

Validation

@Bean

```
public MessageSource messageSource() {  
    ResourceBundleMessageSource source = new  
ResourceBundleMessageSource();  
    source.setBasename("messages");  
    return source;  
}
```

@Override

```
public Validator getValidator() {  
    LocalValidatorFactoryBean validator = new LocalValidatorFactoryBean();  
    validator.setValidationMessageSource(messageSource());  
    return validator;  
}
```

Special Bean Types

Bean type	Explanation
HandlerMapping	<p>Map a request to a handler along with a list of interceptors for pre- and post-processing.</p> <p>The two main HandlerMapping implementations are RequestMappingHandlerMapping (which supports @RequestMapping annotated methods) and SimpleUrlHandlerMapping (which maintains explicit registrations of URI path patterns to handlers).</p>
HandlerAdapter	<p>The interface HandlerAdapter is responsible to invoke a handler method and to return the response as ModelAndView to the DispatcherServlet.</p> <p>The DispatcherServlet then uses a HandlerAdapter to invoke this method.</p> <p>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.</p>

Special Bean Types

Bean type	Explanation
HandlerExceptionHandlerResolver	Strategy to resolve exceptions, possibly mapping them to handlers, to HTML error views, or other targets.
ViewResolver	Resolve logical String-based view names returned from a handler to an actual View with which to render to the response.
LocaleResolver , LocaleContextResolver	Resolve the Locale a client is using and possibly their time zone, in order to be able to offer internationalized views.
ThemeResolver	Resolve themes your web application can use — for example, to offer personalized layouts.

Special Bean Types

Bean type	Explanation
MultipartResolver	Abstraction for parsing a multi-part request (for example, browser form file upload) with the help of some multipart parsing library.
FlashMapManager	Store and retrieve the “input” and the “output” FlashMap that can be used to pass attributes from one request to another, usually across a redirect.

Locale and Theme Resolver

- spring-mvc-locale-resolver-example (refer project)
- Fileupload(Multipart resolver)

Interceptors

- All HandlerMapping implementations support handler interceptors.
- Interceptors must implement HandlerInterceptor from the org.springframework.web.servlet package.
- It has three methods that should provide enough flexibility to do all kinds of pre-processing and post-processing:
- **HandlerInterceptor** interface defined 3 methods.
- **preHandle(request, response, handler)** – Used to intercept the request before handed over to the handler method. Here handler is the chosen handler object to handle the request.
- **postHandle(request, response, handler, modelAndView)** – Used to intercept the request after handler has completed request processing but **DispatcherServlet** is yet to render the view.
- **afterCompletion(request, response, handler, exception)**

Interception

- The `preHandle(..)` method returns a boolean value.
- It can be used to break or continue the processing of the execution chain.
- When this method returns true, the handler execution chain continues.
- When it returns false, the `DispatcherServlet` assumes the interceptor itself has taken care of requests.
- It does not continue executing the other interceptors and the actual handler in the execution chain.

Interceptors

@Configuration

@EnableWebMvc

```
public class WebConfig implements WebMvcConfigurer {  
    @Override  
    public void addInterceptors(InterceptorRegistry registry) {  
        registry.addInterceptor(new LocaleChangeInterceptor());  
        registry.addInterceptor(new  
ThemeChangeInterceptor()).addPathPatterns("/**").excludePathPatterns("/admin/**");  
        registry.addInterceptor(new  
SecurityInterceptor()).addPathPatterns("/secure/*");  
    }  
}
```

Interceptors

```
<mvc:interceptors>
```

```
<bean  
class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"/>
```

```
</mvc:interceptor>
```

```
<mvc:mapping path="/**"/>
```

```
<mvc:exclude-mapping path="/admin/**"/>
```

```
<bean  
class="org.springframework.web.servlet.theme.ThemeChangeInterceptor"/>
```

```
</mvc:interceptor>
```

```
<mvc:interceptor>
```

```
<mvc:mapping path="/secure/*"/> <bean  
class="org.example.SecurityInterceptor"/>
```

```
</mvc:interceptor>
```

```
</mvc:interceptors>
```

Interceptors

- firstSpringApplication(refer project)

Content Types

@Configuration

@EnableWebMvc

public class WebConfig implements WebMvcConfigurer {
@Override

public void
configureContentNegotiation(ContentNegotiationConfigurer
configurer)

{

configurer.mediaType("json", MediaType.APPLICATION_JSON);

configurer.mediaType("xml", MediaType.APPLICATION_XML);

}

}

Content Types

```
<mvc:annotation-driven content-negotiation-  
manager="contentNegotiationManager"/>
```

```
<bean id="contentNegotiationManager"  
class="org.springframework.web.accept.ContentNegotiationMan  
agerFactoryBean">
```

```
<property name="mediaTypes">
```

```
<value> json=application/json xml=application/xml </value>  
</property>
```

```
</bean>
```

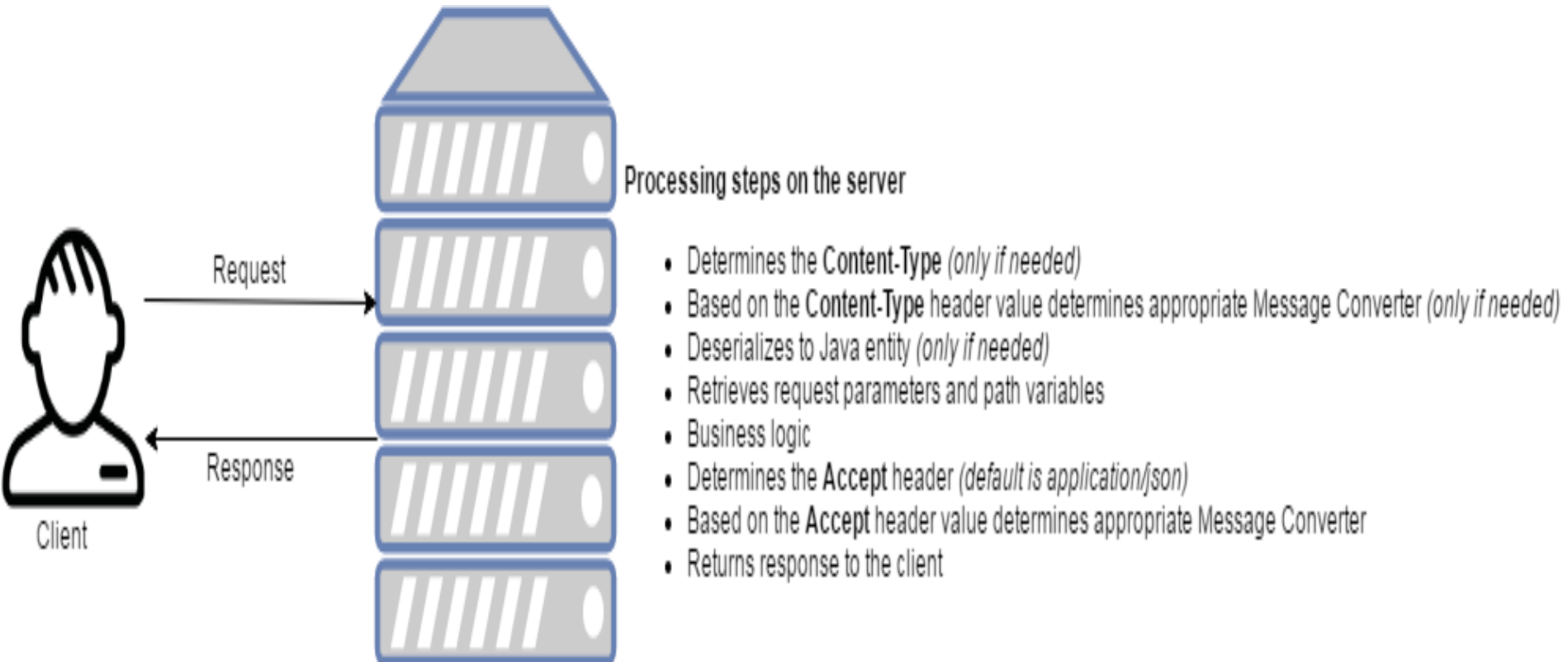
Message Converters

- Message converters are to marshal and unmarshal object in a different format (like JSON, XML etc.).
- Spring MVC uses the `HttpMessageConverter` interface to convert HTTP requests and responses.

How do the Message Converters work?

- When receiving a new request, Spring will make use of the “Accept” header to determine the media type that it needs to respond with.
- Spring then tries to find a registered converter that is capable of handling that specific media type – and it will use it to convert the entity and send back the response.
- For receiving a request which contains JSON information – the framework will use the “Content-Type” header to determine the media-type in the request body.
- Spring searches for the appropriate `HttpMessageConverter` that can convert the body sent by the client to a Java Object.

How do the Message Converters work?



Further digging into the process

- `@ResponseBody` means that the returned value of the method will constitute the body of the HTTP response.
- Of course, an HTTP response can't contain Java objects.
- The processed data on the server is transformed to a format suitable for REST applications, typically JSON or XML.
- As discussed above, the “Accept” header specified by the Client will be used to choose the appropriate HTTP Converter to marshal/transform the processed entity to the required format in response.
- Now, let's assume that there is no the “Accept” header, the default attempt that Spring makes is to convert the response to JSON.
- But what if there is no corresponding `MappingJackson2HttpMessageConverter` JAR included in the classpath.
- Then the application throws an exception –

Further digging into the process

exception

```
org.springframework.web.util.NestedServletException: Request processing failed; nested exception is java.lang.IllegalArgumentException
    org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:982)
    org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:861)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:624)
    org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:846)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:731)
    org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
```

root cause

```
java.lang.IllegalArgumentException: No converter found for return value of type: class com.jcombat.bean.Employee
    org.springframework.util.Assert.isTrue(Assert.java:68)
    org.springframework.web.servlet.mvc.method.annotation.AbstractMessageConverterMethodProcessor.writeWithMessageConverters(AbstractMessageConverterMethodProcessor.java:111)
    org.springframework.web.servlet.mvc.method.annotation.AbstractMessageConverterMethodProcessor.writeWithMessageConverters(AbstractMessageConverterMethodProcessor.java:103)
    org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleReturnValue(RequestMappingHandlerAdapter.java:203)
    org.springframework.web.servlet.mvc.method.support.HandlerMethodReturnValueHandlerComposite.handleReturnValue(HandlerMethodReturnValueHandlerComposite.java:70)
```

Further digging into the process

But if the JAR is present in the classpath, the HTTP Response body contains –

```
1 {  
2  "empld":"1234",  
3  "name":"Abhimanyu"  
4 }
```

Further digging into the process

- **@RequestBody** is often used as argument in a Controller method.
- It indicates to Spring that the body of the HTTP Request needs to be deserialized into its corresponding Java entity.
- As discussed previously, the “**Content-Type**” header specified by the Client will be used to determine the appropriate converter for this.

```
1 @RequestMapping(value = "/{name}", method
2 = RequestMethod.GET)
3 public @ResponseBody void
4 updateEmployee(@PathVariable("id") String id,
5 @RequestBody Employee employee) {
    employeeService.update(employee);
}
```


Further digging into the process

When we consume the service with “**Content-Type**” as *application/json* and HTTP request body as JSON data (as mentioned below), the **@RequestBody** determines the appropriate converter and deserializes the incoming JSON request data into its corresponding Java entity as specified in the controller argument.

```
1 {  
2  "empId":"1234",  
3  "name":"Abhimanyu"  
4 }
```

The Default Message Converters

- By default, the following `HttpMessageConverters` instances are pre-enabled:
- `ByteArrayHttpMessageConverter` – converts byte arrays
- `StringHttpMessageConverter` – converts Strings
- `ResourceHttpMessageConverter` – converts `org.springframework.core.io.Resource` for any type of octet stream
- `SourceHttpMessageConverter` – converts `javax.xml.transform.Source`
- `FormHttpMessageConverter` – converts form data to/from a `MultiValueMap<String, String>`.
- `Jaxb2RootElementHttpMessageConverter` – converts Java objects to/from XML (added only if JAXB2 is present on the classpath)

The Default Message Converters

- MappingJackson2HttpMessageConverter – converts JSON (added only if Jackson 2 is present on the classpath)
- MappingJacksonHttpMessageConverter – converts JSON (added only if Jackson is present on the classpath)
- AtomFeedHttpMessageConverter – converts Atom feeds (added only if Rome is present on the classpath)
- RssChannelHttpMessageConverter – converts RSS feeds (added only if Rome is present on the classpath)
- Message Converters

Message Converters

@Configuration

@EnableWebMvc

public class WebConfiguration implements WebMvcConfigurer {

 @Override

 public void configureMessageConverters(List<HttpMessageConverter<?>>
converters) {

 Jackson2ObjectMapperBuilder builder = new Jackson2ObjectMapperBuilder()

 .indentOutput(true)

 .dateFormat(new SimpleDateFormat("yyyy-MM-dd"))

 .modulesToInstall(new ParameterNamesModule());

 converters.add(new MappingJackson2HttpMessageConverter(builder.build()));

 converters.add(new
MappingJackson2XmlHttpMessageConverter(builder.createXmlMapper(true).build()));
 }
}

Message Converters

```
<mvc:annotation-driven>
```

```
<mvc:message-converters>
```

```
<bean
```

```
class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">
```

```
<property name="objectMapper" ref="objectMapper"/>
```

```
</bean>
```

```
<bean
```

```
class="org.springframework.http.converter.xml.MappingJackson2XmlHttpMessageConverter">
```

```
<property name="objectMapper" ref="xmlMapper"/> </bean>
```

```
</mvc:message-converters>
```

Message Converters

```
</mvc:annotation-driven>
```

```
<bean id="objectMapper"  
class="org.springframework.http.converter.json.Jackson2Object  
MapperFactoryBean" p:indentOutput="true"  
p:simpleDateFormat="yyyy-MM-dd"  
p:modulesToInstall="com.fasterxml.jackson.module.paramnames.  
ParameterNamesModule"/>
```

```
<bean id="xmlMapper" parent="objectMapper"  
p:createXmlMapper="true"/>
```

Message Converters

- `http-message-converter-master`(Refer Project)

CORS

```
@RestController
```

```
@RequestMapping("/account")
```

```
public class AccountController {
```

```
    @CrossOrigin
```

```
    @GetMapping("/{id}")
```

```
    public Account retrieve(@PathVariable Long id) {
```

```
        // ...
```

```
    }
```

```
    @DeleteMapping("/{id}")
```

```
    public void remove(@PathVariable Long id) {
```

```
        // ...
```

```
    }
```


CORS

```
@CrossOrigin(origins = "https://domain2.com", maxAge = 3600)
```

```
@RestController @RequestMapping("/account")
```

```
public class AccountController {
```

```
@GetMapping("/{id}")
```

```
public Account retrieve(@PathVariable Long id)
```

```
{ // ... }
```

```
@DeleteMapping("/{id}")
```

```
public void remove(@PathVariable Long id)
```

```
{ // ... }
```

```
}
```

Global Configuration

@Configuration

@EnableWebMvc

```
public class WebConfig implements WebMvcConfigurer {
```

```
    @Override
```

```
    public void addCorsMappings(CorsRegistry registry) {
```

```
        registry.addMapping("/api/**")
```

```
            .allowedOrigins("https://domain2.com")
```

```
            .allowedMethods("PUT", "DELETE")
```

```
            .allowedHeaders("header1", "header2", "header3")
```

```
            .exposedHeaders("header1", "header2")
```

```
            .allowCredentials(true).maxAge(3600);
```

```
        // Add more mappings...
```

```
    }
```

```
}
```

Global Configuration

```
<mvc:cors>
```

```
  <mvc:mapping path="/api/**"
```

```
    allowed-origins="https://domain1.com,  
https://domain2.com"
```

```
    allowed-methods="GET, PUT"
```

```
    allowed-headers="header1, header2, header3"
```

```
    exposed-headers="header1, header2" allow-  
credentials="true"
```

```
    max-age="123" />
```

```
  <mvc:mapping path="/resources/**"
```

```
    allowed-origins="https://domain1.com" />
```

```
</mvc:cors>
```

Exceptions

- If an exception occurs during request mapping or is thrown from a request handler (such as a `@Controller`), the `DispatcherServlet` delegates to a chain of `HandlerExceptionResolver` beans to resolve the exception.
- It provides alternative handling, which is typically an error response.

Exceptions

HandlerException Resolver	Description
SimpleMapping ExceptionHandlerResolver	A mapping between exception class names and error view names. Useful for rendering error pages in a browser application.
DefaultHandler ExceptionHandlerResolver	Resolves exceptions raised by Spring MVC and maps them to HTTP status codes. See also alternative ResponseEntityExceptionHandler and REST API exceptions .
ResponseStatus ExceptionHandlerResolver	Resolves exceptions with the <code>@ResponseStatus</code> annotation and maps them to HTTP status codes based on the value in the annotation.
ExceptionHandler ExceptionHandlerResolver	Resolves exceptions by invoking an <code>@ExceptionHandler</code> method in a <code>@Controller</code> or a <code>@ControllerAdvice</code> class.

View Resolution

- Spring MVC defines the View Resolver
- View interfaces that render models in a browser without tying to a specific view technology.
- View Resolver provides a mapping between view names and actual views.
- View addresses the preparation of data before handing over to a specific view technology.

ViewResolver implementations

ViewResolver	Description
AbstractCachingViewResolver	<p>Sub-classes of AbstractCachingViewResolver cache view instances that they resolve.</p> <p>Caching improves performance of certain view technologies.</p> <p>To turn off the cache set the cache property to false.</p> <p>Furthermore, if view is refreshed at runtime (for example, when a FreeMarker template is modified), use the removeFromCache(String viewName, Locale loc) method.</p>
XmlViewResolver	<p>Implementation of ViewResolver that accepts a configuration file written in XML with the same DTD as Spring's XML bean factories. The default configuration file is /WEB-INF/views.xml.</p>

ViewResolver implementations

ViewResolver	Description
ResourceBundleViewResolver	Implementation of ViewResolver that uses bean definitions in a ResourceBundle, specified by the bundle base name. For each view it is supposed to resolve, it uses the value of the property [viewname].(class) as the view class and the value of the property [viewname].url as the view URL.
UrlBasedViewResolver	Simple implementation of the ViewResolver interface that affects 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.

ViewResolver implementations

ViewResolver	Description
InternalResourceViewResolver	Convenient subclass of <code>UrlBasedViewResolver</code> that supports <code>InternalResourceView</code> (in effect, Servlets and JSPs) and subclasses such as <code>JstlView</code> and <code>TilesView</code> . You can specify the view class for all views generated by this resolver by using <code>setViewClass(..)</code> .
FreeMarkerViewResolver	Convenient subclass of <code>UrlBasedViewResolver</code> that supports <code>FreeMarkerView</code> and custom subclasses of them.
ContentNegotiatingViewResolver	Implementation of the <code>ViewResolver</code> interface that resolves a view based on the request file name or <code>Accept</code> header.

View Resolver

@Configuration

@EnableWebMvc

```
public class WebConfig implements WebMvcConfigurer {
```

```
    @Override
```

```
    public void configureViewResolvers(ViewResolverRegistry  
registry) {
```

```
        registry.enableContentNegotiation(new  
MappingJackson2JsonView());
```

```
        registry.jsp();
```

```
    }
```

```
}
```

View Resolver

```
<mvc:view-resolvers>
```

```
  <mvc:content-negotiation>
```

```
    <mvc:default-views>
```

```
      <bean
```

```
class="org.springframework.web.servlet.view.json.MappingJ  
ackson2JsonView"/>
```

```
    </mvc:default-views>
```

```
  </mvc:content-negotiation>
```

```
  <mvc:jsp/>
```

```
</mvc:view-resolvers>
```

ViewResolver

- MultipleViewResolversExample(Refer Project)
- Differs based on view name in controller

Static Resources

@Configuration

@EnableWebMvc

```
public class WebConfig implements WebMvcConfigurer {
```

@Override

```
public void addResourceHandlers(ResourceHandlerRegistry registry) {  
    registry.addResourceHandler("/resources/**")  
        .addResourceLocations("/public", "classpath:/static/")  
        .setCachePeriod(31556926);  
}  
}
```

Static Resources

```
<mvc:resources mapping="/resources/**" location="/public,  
classpath:/static/" cache-period="31556926" />
```

Annotated Controllers

- `@Controller`
- `public class HelloController {`
- `@GetMapping("/hello")`
- `public String handle(Model model) {`
- `model.addAttribute("message", "Hello World!");`
- `return "index";`
- `}`
- `}`

Declaration

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xmlns:p="http://www.springframework.org/schema/p"  
xmlns:context="http://www.springframework.org/schema/context"  
xsi:schemaLocation=" http://www.springframework.org/schema/beans  
https://www.springframework.org/schema/beans/spring-beans.xsd  
http://www.springframework.org/schema/context  
https://www.springframework.org/schema/context/spring-context.xsd">
```

```
<context:component-scan base-package="org.example.web"/>
```

```
<!-- ... -->
```

```
</beans>
```


Declaration

- `@Configuration`
- `@ComponentScan("org.example.web")`
- `public class WebConfig {`
- `// ...`
- `}`

Spring MVC

DispatcherServlet (1 of 3)

- DispatcherServlet:
- Is the core of Spring MVC
- Acts as a Front Controller
 - Coordinates all request handling activities
 - Dispatches to registered handlers for processing a web request.

Spring MVC

DispatcherServlet (2 of 3)

- Spring MVC's front controller (DispatcherServlet) coordinates the entire request lifecycle:
- Configured in web.xml:

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

- Is mapped using a URL Pattern. In this case, .htm

```
<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

Spring MVC

DispatcherServlet (3 of 3)

- Spring MVC's front controller (DispatcherServlet) coordinates the entire request lifecycle:
- All requests matching .htm pattern automatically will be mapped to DispatcherServlet
- DispatcherServlet loads Spring application context from an XML file* that usually contains <bean> definitions for the Spring MVC components
- **Note:** *default is <servlet-name>-servlet.xml

Spring MVC

Application Flow (1 of 4)

The application flow is:

1. DispatcherServlet receives request and coordinates business functionality
2. Looks for xml configuration file and calls appropriate controller
3. Controller maps the request to handler, handler processes request/returns instance of ModelAndView to DispatcherServlet

Spring MVC

Application Flow (2 of 4)

1. DispatcherServlet receives request and coordinates business functionality.

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

```
<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

Spring MVC

Application Flow (3 of 4)

2. DispatcherServlet looks for the xml configuration file and calls appropriate controller

- Use the following in the .xml configuration file to send the DispatcherServlet looking for Controller(s).

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:ctx="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-2.5.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd ">

    <ctx:annotation-config></ctx:annotation-config>
    <ctx:component-scan base-package="com.accenture.tdf.controller"></ctx:component-scan>

</beans>
```

Spring MVC

Application Flow (4 of 4)


3. Controller maps the request to appropriate handler which in turn processes the request and returns instance of ModelAndView to DispatcherServlet.

```
@Controller
public class SampleController {

    @RequestMapping("/userlogin.htm")
    public ModelAndView validateUser(HttpServletRequest request, HttpServletResponse response) throws Exception
```


Spring MVC @Controller

- **@Controller**
- Indicates that a particular class serves the role of a *controller*.
- Eliminates the need to extend any Controller base class



Request and
Response are
automatically
filled in by
Spring MVC

```
@Controller
public class SampleController {

    @RequestMapping("/userlogin.htm")
    public ModelAndView validateUser(HttpServletRequest request, HttpServletResponse response) throws Exception
```

@RequestMapping with Wildcards

```
@RequestMapping(value = "/*.request", method = {  
    RequestMethod.GET,  
    RequestMethod.POST })  
public String myHandlingMethod() {  
    return "next";  
}
```

Refer day15springjparestdemo project

```
@RequestMapping("/*.rest")  
public ModelAndView myOtherHandlingMethod() {  
    return new ModelAndView("previous", "x", "abc");  
}
```

Spring MVC @Controller

- Controllers interpret user input and convert it into a model object that can be used by view.
- @Controller is a stereotype annotation and hence, dispatcher servlet can automatically scan for these classes.
- To enable auto-detection use the following in dispatcher-servlet.xml file.

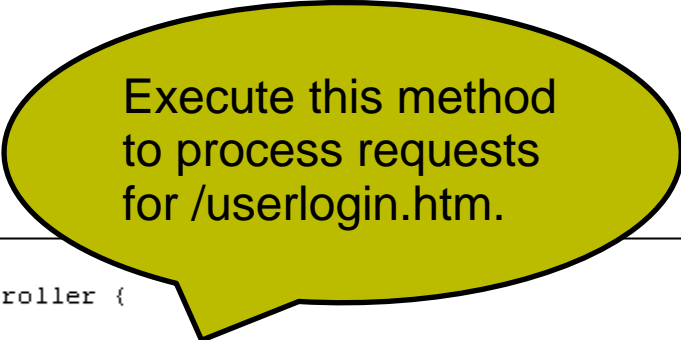
```
<context:component-scan base-package="com.accenture.controllers.*"/>
```

- The component-scan tag also allows us to exclude / include packages during auto-detection
- ```
<context:include-filter type="annotation" expression="org.aspectj.lang.annotation.Aspect"/>
```

# Spring MVC

## @RequestMapping

- **@RequestMapping**
- Is used to map URLs like '/userlogin.htm' to an entire class or a particular handler method.
- Typically, class-level annotation maps a specific request pattern



Execute this method  
to process requests  
for /userlogin.htm.

```
@Controller
public class SampleController {

 @RequestMapping("/userlogin.htm")
 public ModelAndView validateUser(HttpServletRequest request, HttpServletResponse response) throws Exception
 {
```

# Spring MVC

## @RequestMapping

- Handler methods annotated with @RequestMapping have very flexible method signatures.
- The methods can accept the following types of parameters
  - HttpServletRequest
  - HttpServletResponse
  - ModelAttribute
  - ModelMap
  - BindingResult
  - Errors
- The following return types are supported
  - ModelAndView
  - ModelMap
  - ModelAttribute
  - String

# Handler Methods – Method Arguments



# Spring MVC @RequestMapping

| Controller method argument                                          | Description                                                                                                                                                                                                                                                                                             |
|---------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WebRequest, NativeWebRequest                                        | Generic access to request parameters and request and session attributes, without direct use of the Servlet API.                                                                                                                                                                                         |
| javax.servlet.HttpServletRequest, javax.servlet.HttpServletResponse | Choose any specific request or response type — for example, HttpServletRequest, HttpServletResponse, or Spring's MultipartRequest, MultipartHttpServletRequest.                                                                                                                                         |
| javax.servlet.http.HttpSession                                      | Enforces the presence of a session. As a consequence, such an argument is never null. Note that session access is not thread-safe.<br>Consider setting the RequestMappingHandlerAdapter instance's synchronizeOnSession flag to true if multiple requests are allowed to concurrently access a session. |
| javax.servlet.http.PushBuilder                                      | Servlet 4.0 push builder API for programmatic HTTP/2 resource pushes. Note that, per the Servlet specification, the injected PushBuilder instance can be null if the client does not support that HTTP/2 feature.                                                                                       |

# Http Session

```
@RequestMapping(value = { "/login" }, method = RequestMethod.POST)
```

```
@ResponseBody
```

```
public String login(HttpSession session,String username,String password)
throws Exception {
```

```
 Member member=userService.authenticateUser(username, password);
```

```
 if(member!=null) {
```

```
 session.setAttribute("MEMBER", member);
```

```
 } else {
```

```
 throw new Exception("Invalid username or password");
```

```
 }
```

```
 return Utils.toJson("SUCCESS");
```

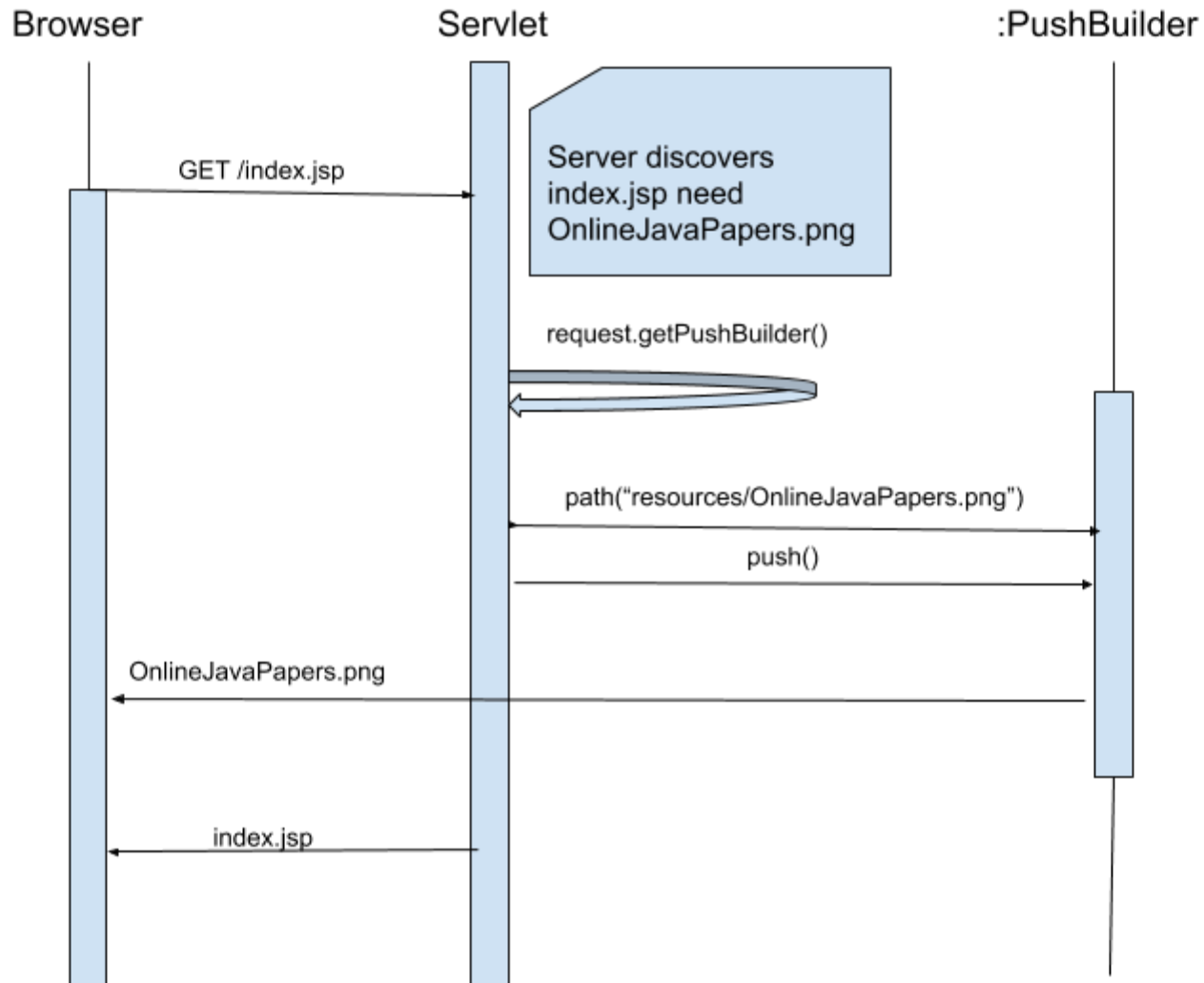
```
}
```



# Push Builder

- The most important feature of Servlet 4.0, due to HTTP/2, is the implementation of the server push capability.
- The concept behind this technique is that if the client/browser requests a certain resource, the server assumes, in advance, that some other related resources may also be requested soon.
- Because of this assumption, it pushes them into the cache (called 'cache push') before they are actually needed.
- For example, it is very much likely that when a webpage is loaded, it may eventually request a CSS file or another image.
- The server proactively starts pushing the bytes of these assets simultaneously, without the need for the client to make an explicit request.

# Push Builder



# Spring MVC @RequestMapping

| Controller method argument                                      | Description                                                                                                                                               |
|-----------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>java.security.Principal</code>                            | Currently authenticated user — possibly a specific Principal implementation class if known.                                                               |
| <code>HttpMethod</code>                                         | The HTTP method of the request.                                                                                                                           |
| <code>java.util.Locale</code>                                   | The current request locale, determined by the most specific LocaleResolver available (in effect, the configured LocaleResolver or LocaleContextResolver). |
| <code>java.util.TimeZone</code> + <code>java.time.ZoneId</code> | The time zone associated with the current request, as determined by a LocaleContextResolver.                                                              |

# Spring MVC @RequestMapping

| Controller method argument               | Description                                                        |
|------------------------------------------|--------------------------------------------------------------------|
| java.io.InputStream, java.io.<br>Reader  | For access to the raw request body as exposed by the Servlet API.  |
| java.io.OutputStream, java.io.<br>Writer | For access to the raw response body as exposed by the Servlet API. |
| @PathVariable                            | For access to URI template variables.                              |
| @MatrixVariable                          | For access to name-value pairs in URI path segments.               |

# Matrix Variable

- <http://localhost:8080/user/firstName=Sunil/lastName=Singh>
- <http://localhost:8080/user/name=sunil>
- <http://localhost:8080/employee/Mike;salary=45000;dept=HR>
- <http://localhost:8080/employee/Mike;id=12;dept=HR;/India;id=25>
- <http://localhost:8080/car/Audi;color=RED,BLACK,WHITE>
- <http://localhost:8080/person/Mike;dob=2017-02-12>
- Refer Spring matrix variable project

# Spring MVC

## @RequestMapping

| Controller method argument | Description                                                                                                                                                                                                                                                               |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| @RequestParam              | For access to the Servlet request parameters, including multipart files. Parameter values are converted to the declared method argument type. Note that use of @RequestParam is optional for simple parameter values. See “Any other argument”, at the end of this table. |
| @RequestHeader             | For access to request headers. Header values are converted to the declared method argument type.                                                                                                                                                                          |
| @CookieValue               | For access to cookies. Cookies values are converted to the declared method argument type. See <a href="#">@CookieValue</a> .                                                                                                                                              |
| @RequestBody               | For access to the HTTP request body. Body content is converted to the declared method argument type by using <code>HttpMessageConverter</code> implementations.                                                                                                           |

# Spring MVC @RequestMapping

| Controller method argument                                                   | Description                                                                                                                                                                   |
|------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HttpEntity<B>                                                                | For access to request headers and body. The body is converted with an <code>HttpMessageConverter</code> .                                                                     |
| @RequestPart                                                                 | For access to a part in a multipart/form-data request, converting the part's body with an <code>HttpMessageConverter</code> .                                                 |
| java.util.Map, org.springframework.ui.Model, org.springframework.ui.ModelMap | For access to the model that is used in HTML controllers and exposed to templates as part of view rendering.                                                                  |
| RedirectAttributes                                                           | Specify attributes to use in case of a redirect (that is, to be appended to the query string) and flash attributes to be stored temporarily until the request after redirect. |

# Spring MVC

## @RequestMapping

| Controller method argument                     | Description                                                                                                                                                                                                                                                                                        |
|------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| @ModelAttribute                                | For access to an existing attribute in the model (instantiated if not present) with data binding and validation applied. Note that use of @ModelAttribute is optional (for example, to set its attributes).                                                                                        |
| Errors, BindingResult                          | For access to errors from validation and data binding for a command object (that is, a @ModelAttribute argument) or errors from the validation of a @RequestBody or @RequestPart arguments. You must declare an Errors, or BindingResult argument immediately after the validated method argument. |
| SessionStatus + class-level @SessionAttributes | For marking form processing complete, which triggers cleanup of session attributes declared through a class-level @SessionAttributes annotation.                                                                                                                                                   |
| UriComponentsBuilder                           | For preparing a URL relative to the current request's host, port, scheme, context path, and the literal part of the servlet mapping.                                                                                                                                                               |



# Spring MVC

## @RequestMapping

| Controller method argument | Description                                                                                                                                                                                                                                              |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| @SessionAttribute          | For access to any session attribute, in contrast to model attributes stored in the session as a result of a class-level @SessionAttributes declaration.                                                                                                  |
| @ModelAttribute            | For access to request attributes.                                                                                                                                                                                                                        |
| Any other argument         | If a method argument is not matched to any of the earlier values in this table and it is a simple type (as determined by <a href="#">BeanUtils#isSimpleProperty</a> , it is resolved as a @RequestParam. Otherwise, it is resolved as a @ModelAttribute. |

# Spring MVC

## @RequestMapping

| Controller method argument | Description                                                                                                                                                                                                                                              |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| @SessionAttribute          | For access to any session attribute, in contrast to model attributes stored in the session as a result of a class-level @SessionAttributes declaration.                                                                                                  |
| @ModelAttribute            | For access to request attributes.                                                                                                                                                                                                                        |
| Any other argument         | If a method argument is not matched to any of the earlier values in this table and it is a simple type (as determined by <a href="#">BeanUtils#isSimpleProperty</a> , it is resolved as a @RequestParam. Otherwise, it is resolved as a @ModelAttribute. |

# Session Attribute

- Refer Session Scope Project

# Handler Methods – Return Values

# Spring MVC @RequestMapping

| Controller method return value                                             | Description                                                                                                                                                                                   |
|----------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| @ResponseBody                                                              | The return value is converted through <code>HttpMessageConverter</code> implementations and written to the response.                                                                          |
| <code>HttpEntity&lt;B&gt;</code> ,<br><code>ResponseEntity&lt;B&gt;</code> | The return value that specifies the full response (including HTTP headers and body) is to be converted through <code>HttpMessageConverter</code> implementations and written to the response. |
| <code>HttpHeaders</code>                                                   | For returning a response with headers and no body.                                                                                                                                            |

# Spring MVC @RequestMapping

| Controller method return value              | Description                                                                                                                                                                                                                                                           |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| String                                      | A view name to be resolved with ViewResolver implementations and used together with the implicit model — determined through command objects and @ModelAttribute methods. The handler method can also programmatically enrich the model by declaring a Model argument. |
| View                                        | A View instance to use for rendering together with the implicit model — determined through command objects and @ModelAttribute methods. The handler method can also programmatically enrich the model by declaring a Model argument.                                  |
| java.util.Map, org.springframework.ui.Model | Attributes to be added to the implicit model, with the view name implicitly determined through a RequestToViewNameTranslator.                                                                                                                                         |

# Spring MVC @RequestMapping

| Controller method return value | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| @ModelAttribute                | <p>An attribute to be added to the model, with the view name implicitly determined through a RequestToViewNameTranslator.</p> <p>Note that @ModelAttribute is optional. See "Any other return value" at the end of this table.</p>                                                                                                                                                                                                                                                                                                     |
| ModelAndView object            | <p>The view and model attributes to use and, optionally, a response status.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| void                           | <p>A method with a void return type (or null return value) is considered to have fully handled the response if it also has a ServletResponse, an OutputStream argument, or an @ResponseStatus annotation. The same is also true if the controller has made a positive ETag or lastModified timestamp check (see <a href="#">Controllers</a> for details).</p> <p>If none of the above is true, a void return type can also indicate “no response body” for REST controllers or a default view name selection for HTML controllers.</p> |

# Spring MVC @RequestMapping

| Controller method return value                                                                                                                                     | Description                                                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <code>DeferredResult&lt;V&gt;</code>                                                                                                                               | Produce any of the preceding return values asynchronously from any thread — for example, as a result of some event or callback. |
| <code>Callable&lt;V&gt;</code>                                                                                                                                     | Produce any of the above return values asynchronously in a Spring MVC-managed thread.                                           |
| <code>ListenableFuture&lt;V&gt;</code> , <code>java.util.concurrent.CompletionStage&lt;V&gt;</code> , <code>java.util.concurrent.CompletableFuture&lt;V&gt;</code> | Alternative to <code>DeferredResult</code> , as a convenience (for example, when an underlying service returns one of those).   |



# Spring MVC @RequestMapping

| Controller method return value                                                                                       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ResponseBodyEmitter, SseEmitter                                                                                      | Emit a stream of objects asynchronously to be written to the response with <code>HttpMessageConverter</code> implementations. Also supported as the body of a <code>ResponseEntity</code> . See <a href="#">Asynchronous Requests</a> and <a href="#">HTTP Streaming</a> .                                                                                                                                                                                                                                                                                                             |
| StreamingResponseBody                                                                                                | Write to the response <code>OutputStream</code> asynchronously. Also supported as the body of a <code>ResponseEntity</code> . See <a href="#">Asynchronous Requests</a> and <a href="#">HTTP Streaming</a> .                                                                                                                                                                                                                                                                                                                                                                           |
| Reactive types — <code>Reactor</code> , <code>RxJava</code> , or others through <code>ReactiveAdapterRegistry</code> | Alternative to <code>DeferredResult</code> with multi-value streams (for example, <code>Flux</code> , <code>Observable</code> ) collected to a <code>List</code> . For streaming scenarios (for example, text/event-stream, application/json+stream), <code>SseEmitter</code> and <code>ResponseBodyEmitter</code> are used instead, where <code>ServletOutputStream</code> blocking I/O is performed on a Spring MVC-managed thread and back pressure is applied against the completion of each write. See <a href="#">Asynchronous Requests</a> and <a href="#">Reactive Types</a> . |

# Spring MVC @RequestMapping

| Controller method return value                                                                                       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ResponseBodyEmitter, SseEmitter                                                                                      | Emit a stream of objects asynchronously to be written to the response with <code>HttpMessageConverter</code> implementations. Also supported as the body of a <code>ResponseEntity</code> . See <a href="#">Asynchronous Requests</a> and <a href="#">HTTP Streaming</a> .                                                                                                                                                                                                                                                                                                             |
| StreamingResponseBody                                                                                                | Write to the response <code>OutputStream</code> asynchronously. Also supported as the body of a <code>ResponseEntity</code> . See <a href="#">Asynchronous Requests</a> and <a href="#">HTTP Streaming</a> .                                                                                                                                                                                                                                                                                                                                                                           |
| Reactive types — <code>Reactor</code> , <code>RxJava</code> , or others through <code>ReactiveAdapterRegistry</code> | Alternative to <code>DeferredResult</code> with multi-value streams (for example, <code>Flux</code> , <code>Observable</code> ) collected to a <code>List</code> . For streaming scenarios (for example, text/event-stream, application/json+stream), <code>SseEmitter</code> and <code>ResponseBodyEmitter</code> are used instead, where <code>ServletOutputStream</code> blocking I/O is performed on a Spring MVC-managed thread and back pressure is applied against the completion of each write. See <a href="#">Asynchronous Requests</a> and <a href="#">Reactive Types</a> . |

# Spring MVC @RequestMapping

| Controller method return value | Description                                                                                                                                                                                                                                                                                                                                                    |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Any other return value         | Any return value that does not match any of the earlier values in this table and that is a String or void is treated as a view name (default view name selection through RequestToViewNameTranslator applies), provided it is not a simple type, as determined by <a href="#">BeanUtils#isSimpleProperty</a> . Values that are simple types remain unresolved. |

# Spring MVC Components

DispatcherServlet

Acts as a Controller

Controller Class

This class will return the Model And View

CommandClass

The class of the object that will be used to represent the user data. This is also called Form Backing Bean

ViewResolver

Customizable view resolution class Application Context Configuration file

JSP\*

Acts as View

**Note:** \*JSP is not part of Spring but it is commonly used to create a user interface in Spring applications

# Spring MVC : See-It

---

- Demonstration :

Create a Spring MVC application.

- Environment: Eclipse

- Duration: 20 min

- Steps:

1. Open Project ADFExtensionCodebaseM3\_Spring MVC\_participant
2. Navigate to Java Resources/src
3. Open package `com.accenture.adfx.newcodington.module3.sample`
4. Open `SpringappController.java` and complete the TODOs.
5. Open and review the `index.html`, `web.xml`, `Module3-servlet.xml` and `hello.jsp` files.



# Spring MVC

## ModelAndView

- **ModelAndView** contains the model (some data) and either a logical view name, or an implementation of the **View** interface.

```
String userName=request.getParameter("username");
String password=request.getParameter("password");
ModelAndView mView=new ModelAndView();
if(userName.equalsIgnoreCase(password))
{
 mView.addObject("USERNAME", userName);
 mView.setViewName("/usersuccess.jsp");
}
else
{
 mView.addObject("ERROR", "Invalid Credentials");
 mView.setViewName("/sample.jsp");
}
return mView;
```

# Spring MVC

## View Resolvers

- Logical view names become view objects.

```
<bean id="viewResolver"
 class="org.springframework.web.servlet.view.InternalResourceViewResolver"
 p:prefix="/WEB-INF/jsp/" p:suffix=".jsp" />
```

- InternalResourceViewResolver resolves a logical view name...

# Spring MVC Sample Code

- Follow the steps below to see examples on Spring Core:
    1. Open Project ADFExtensionCodebaseM3\_Spring MVC\_participant
    2. Run it on Tomcat server. This is dependent on completion of the previous See It. You should be able to see a click button. Click on it. Click on Home.
  - You should see a screen like the one below (see 2a. below).
  - 3. Click the link to view sample code.
  - 4. Enter user name and password (see 2b below)
  - 5. Password should match the name. Try various combinations and see how it behaves.
- 

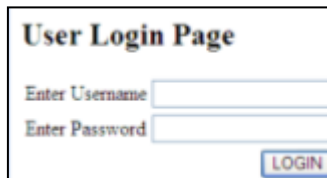
2a

Click [here](#) to view the See It Code

Click [here](#) to view Sample Code

Click [here](#) to view Activity Code

2b



A screenshot of a web form titled "User Login Page". It contains two input fields: "Enter Username" and "Enter Password". Below the password field is a blue button labeled "LOGIN".



# Activity 1: Spring MVC

## Objective:

Write/modify a web-based application using Spring MVC Framework components to insert a record in the Museum table.

## Instructions:

- Navigate to the Module 3\_Activity 1.docx document embedded in the project ADFExtensionCodebaseM3Spring MVC\_participant
- Follow the instructions provided to complete the activity.

# Questions



# Module Summary

- MVC is a fundamental design pattern for separating user interface logic from business logic
- Spring Web MVC module is based on MVC design Pattern
- Spring MVC's key components are:
  - DispatcherServlet
  - Controller Classes
  - View Resolvers