Spring Web on Servlet Stack

By Vo Van Hai

vovanhai@iuh.edu.vn

1

Introduction

- Servlet-stack web applications
  - Built on the Servlet API and deployed to Servlet containers.
  - Includes:
    - Spring MVC,
    - View Technologies,
    - CORS (Cross-Origin Resource Sharing) Support,
    - WebSocket Support.
- *Reactive-stack web applications*
  - *Spring WebFlux*

2

2

Spring Web MVC Framework

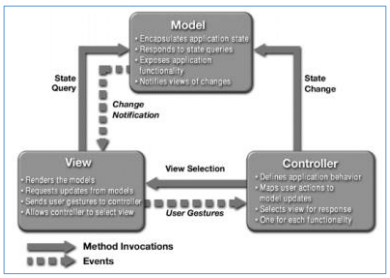
- Referred to as Spring MVC, (as many other web frameworks,) is designed around the front controller pattern where a central Servlet, (called DispatcherServlet,) provides a shared algorithm for request processing, while actual work is performed by configurable delegate components.
- The DispatcherServlet, (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 other tasks.

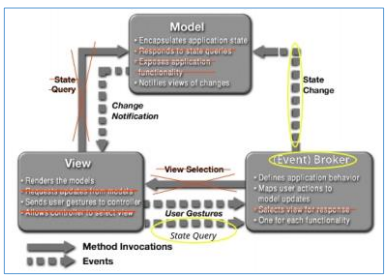
3

3

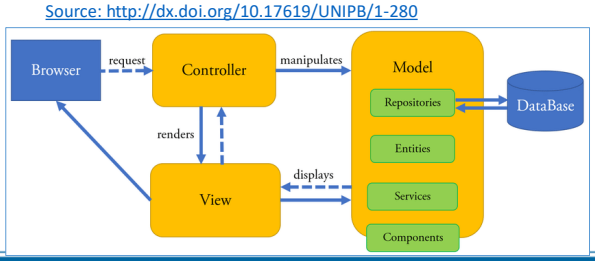
The MVC Model

First introduced by Trygve Reenskaug, Xerox Part company, 1979





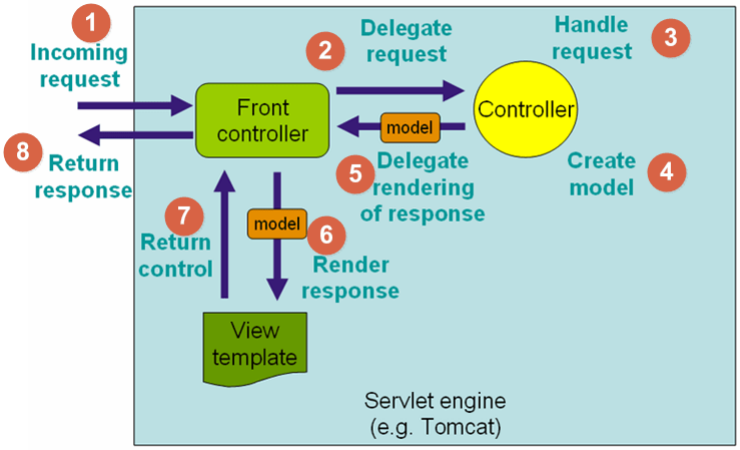
Source: http://dx.doi.org/10.17619/UNIPB/1-280



4

4

The request processing workflow in Spring Web MVC



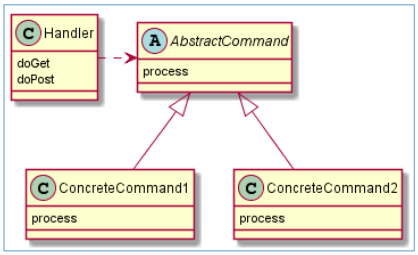
Source: https://docs.spring.io/

5

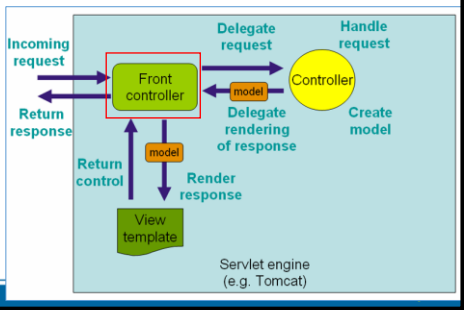
5

The DispatcherServlet

- The request processing workflow of the Spring Web MVC DispatcherServlet.
- The DispatcherServlet is an expression of the "Front Controller" design pattern.
- All request are handled by the DispatcherServlet.
- It is the entry (central) point of the Spring MVC application.
- Front controller determine a suitable **handler** for the request processing.



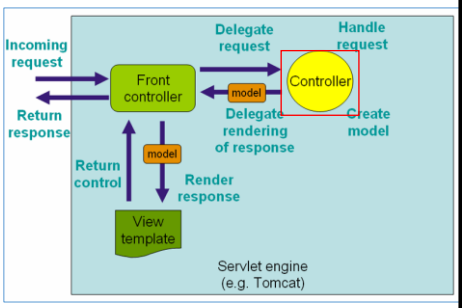
The Font Controller Design Pattern class diagram



6

The Controller

- Controllers provide access to the application behavior that you typically define through a service interface.
- Controllers interpret user input and transform it into a model that is represented to the user by the view.



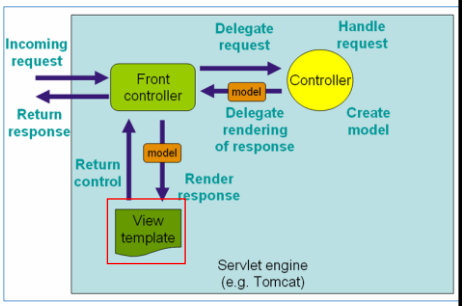
- Spring implements a controller in a very abstract way, which enables you to create a wide variety of controllers.

7

7

The View

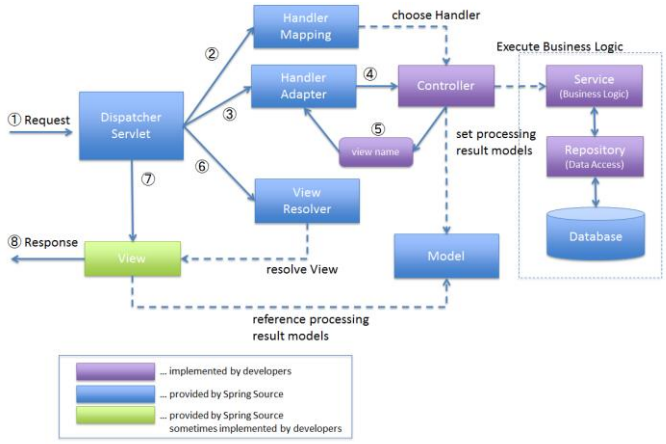
- When the Controller completes the request's processing, the Front Controller selects a relevant **view** to render.
- The Font Controller also pass the model(s) to the rendered view on the browser.



8

8

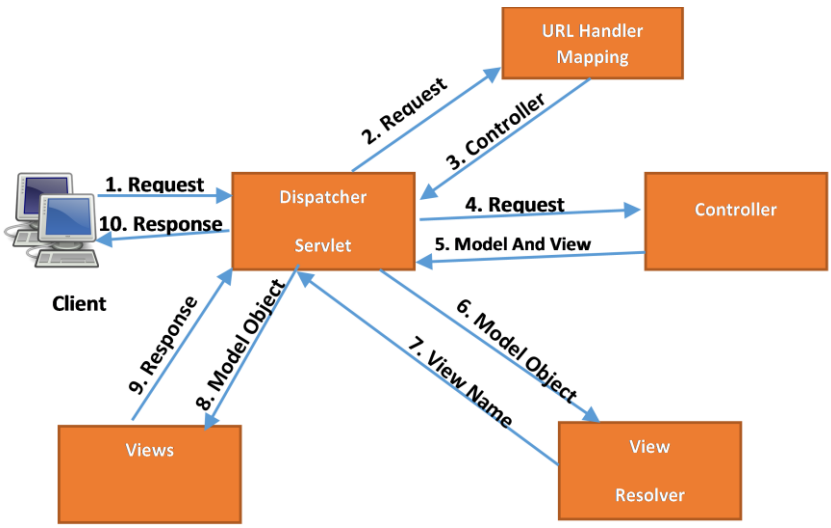
The Spring MVC Processing Sequence



Source: Tomasz Nurkiewicz

9

Spring MVC request flow



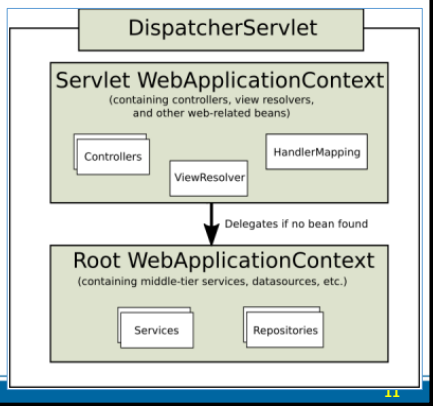
10

Context hierarchy in Spring Web MVC

- The ApplicationContext instances in Spring can be scoped.
  - In the Web MVC framework, each DispatcherServlet has its own WebApplicationContext, which inherits all the beans already defined in the root WebApplicationContext.
  - These inherited beans can be overridden in the servlet-specific scope, and you can define new scope-specific beans local to a given Servlet instance.

The context package also provides the following functionality:

- Access to messages in i18n-style, through the MessageSource interface.
- Access to resources, such as URLs and files, through the ResourceLoader interface.
- Event publication to beans implementing the ApplicationListener interface, through the use of the ApplicationEventPublisher interface.
- Loading of multiple (hierarchical) contexts, allowing each to be focused on one particular layer, such as the web layer of an application, through the HierarchicalBeanFactory interface.



11

Spring MVC Configuration

12

XML Configuration

1. Configure the DispatcherServlet

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-mvc-context.xml</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

2. Setting up a URL mapping to the DispatcherServlet

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context">
    <context:component-scan base-package="vn.edu.iuh"/>
</beans>
```

Read more: https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-servlet.html

13

13

XML Configuration

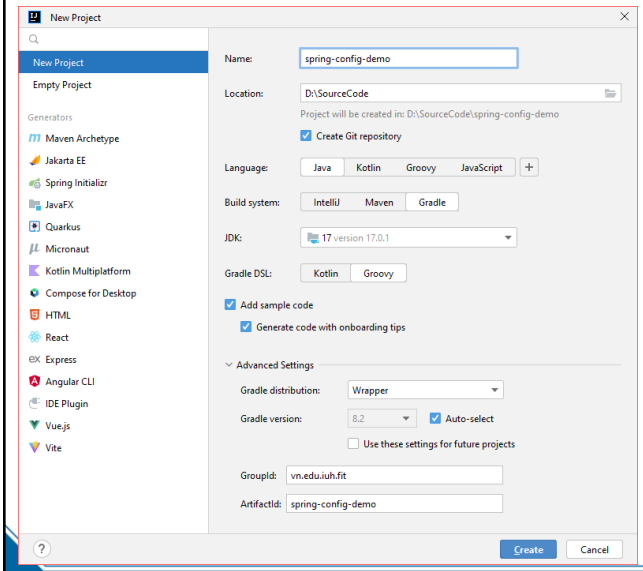
dispatcher-config.xml

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

    <!--3. Components Scanning-->
    <context:component-scan base-package="vn.edu.iuh"/>

    <!-- 4. Add support for conversion, formatting, and validation-->
    <mvc:annotation-driven/>

    <!--5. Configure the Spring MVC view resolver-->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/views/"/>
      <property name="suffix" value=".jsp"/>
    </bean>
</beans>
```

14

14

Java-based Configuration



15

15

Java-based Configuration

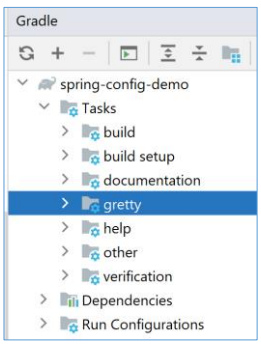
Dependencies

Gretty

Gradle plugin for running web-apps on servlet containers.

```
dependencies {
    compileOnly 'jakarta.servlet:jakarta.servlet-api:6.0.0'
    implementation 'org.springframework:spring-webmvc:6.0.9'
    //JSTL
    implementation 'jakarta.servlet.jsp.jstl:jakarta.servlet.jsp.jstl-api:3.0.0'
    implementation 'org.glassfish.web:jakarta.servlet.jsp.jstl:3.0.1'

    //auto-load
    implementation 'org.gretty:gretty:4.1.0'

    testImplementation platform('org.junit:junit-bom:5.9.1')
    testImplementation 'org.junit.jupiter:junit-jupiter'
}

gretty {
    httpPort = 8080
    contextPath = '/'
//    servletContainer = 'jetty9.4'
    servletContainer = 'tomcat10'
}
```

```
plugins {
    id 'java'
    id 'org.gretty' version '4.1.0'
}
```



16

16

Java-based Configuration

Configuration

```
@EnableWebMvc
@Configuration
//@ComponentScan({ "vn.edu.iuh.fit.controller" })
@ComponentScan(basePackages = { "vn.edu.iuh.fit"})
public class SpringWebConfig implements WebMvcConfigurer {

   @Override
   public void addResourceHandlers(ResourceHandlerRegistry registry) {
      registry.addResourceHandler("/resources/**").addResourceLocations("/resources/");
      registry.addResourceHandler("/webjars/**").addResourceLocations("/webjars/");
   }

   @Bean
   public InternalResourceViewResolver viewResolver() {
      InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
      viewResolver.setViewClass(JstlView.class);
      viewResolver.setPrefix("/WEB-INF/views/jsp/");
      viewResolver.setSuffix(".jsp");
      return viewResolver;
   }
}
```

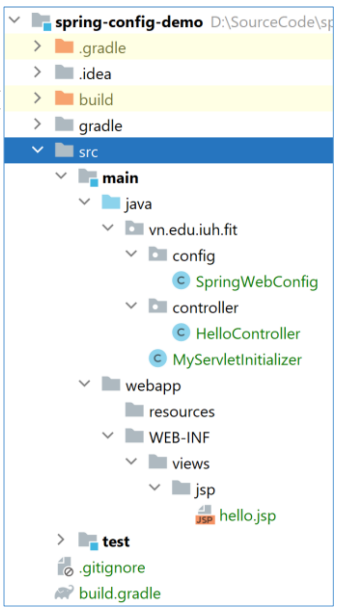
17

17

Java-based Configuration

Configuration (cont.)

```
public class MyServletInitializer extends
            AbstractAnnotationConfigDispatcherServletInitializer {

   // services and data sources
   @Override
   protected Class<?>[] getRootConfigClasses() {
      return new Class[0];
   }

   // controller, view resolver, handler mapping
   @Override
   protected Class<?>[] getServletConfigClasses() {
      return new Class[]{
                  SpringWebConfig.class
            };
   }
   @Override
   protected String[] getServletMappings() {
      return new String[]{"/"};
   }
}
```



18

18

Java-based Configuration

Controllers

```
@Controller
public class HelloController {

   //@RequestMapping(value = "/", method = RequestMethod.GET)
   @GetMapping("/")
   public String defaultPage(ModelMap model) {
      return "hello";
   }

   //@RequestMapping(value = "/hello/{name:.+}", method = RequestMethod.GET)
   @GetMapping("/hello/{name:.+}")
   public ModelAndView hello(@PathVariable("name") String name) {

      ModelAndView model = new ModelAndView();
      model.setViewName("hello");
      model.addObject("message", name);

      return model;
   }
}
```

19

19

Java-based Configuration

Views

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Spring Web MVC and JSP</title>
</head>
<main role="main" class="container">
   <div class="starter-template">
      <h1>Spring Web MVC JSP Example</h1>
      <h2>
         <c:if test="${not empty message}">
            Hello ${message}
         </c:if>

         <c:if test="${empty message}">
            Welcome!
         </c:if>
      </h2>
   </div>
</main>
</body>
</html>
```

20

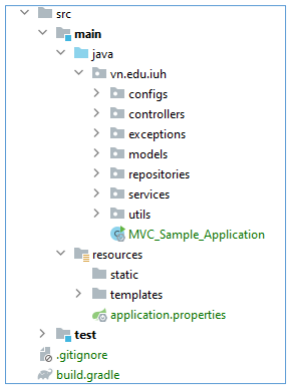
20

Spring Web on Servlet Stack with Springboot

21

Spring MVC with

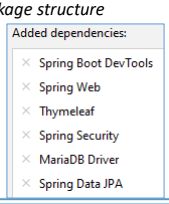
- Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".
- Features
  - Create stand-alone Spring applications
  - Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
  - Provide opinionated 'starter' dependencies to simplify your build configuration
  - Automatically configure Spring and 3rd party libraries whenever possible
  - Provide production-ready features such as metrics, health checks, and externalized configuration
  - Absolutely no code generation and no requirement for XML configuration



Sample package structure

Change default Tomcat server

```
configurations {
    implementation.exclude module: "spring-boot-starter-tomcat"
}
dependencies {
//    implementation ("org.springframework.boot:spring-boot-starter-jetty")
    implementation ("org.springframework.boot:spring-boot-starter-undertow")
```



22

22

Spring MVC Annotations

- Spring MVC provides an annotation-based programming model where @Controller and @RestController components use annotations to express request mappings, request input, exception handling, and more.
- @RestController is a composed annotation that is itself meta-annotated with @Controller and @ResponseBody to indicate a controller whose every method inherits the type-level @ResponseBody annotation and, therefore, writes directly to the response body versus view resolution and rendering with an HTML template

23

23

Spring MVC Annotations (cont.)

- @Controller
  - Stereotype used as "Controller" of the MVC
  - Scanned Request mappings
- @RequestMapping
  - Used for mapping web requests to specific handler classes and/or handler methods.

name	Assign a name to this mapping.
value	The primary mapping expressed by this annotation.
method	The HTTP request methods to map to, narrowing the primary mapping: GET, POST, HEAD, OPTIONS, PUT, PATCH, DELETE, TRACE.
headers	The headers of the mapped request, narrowing the primary mapping.
consumes	The consumable media types of the mapped request, narrowing the primary mapping.
produces	The producible media types of the mapped request, narrowing the primary mapping.
path	In a Servlet environment only: the path mapping URIs
params	The parameters of the mapped request, narrowing the primary mapping.

Table: Attributes of @RequestNapping annotation

24

24

Spring MVC Annotations (cont.)

@RequestMapping cont.

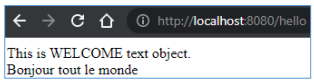
```
@Controller
@RequestMapping("/hello")
public class HomeController {
    @RequestMapping(value = {"","/home","/welcome"}, method = RequestMethod.GET)
    public ModelAndView welcome(){
        ModelAndView model =new ModelAndView( viewName: "welcome");
        model.addObject( attributeName: "text", attributeValue: "This is WELCOME text object.");

        Map<String, Object> mm = model.getModel();
        mm.put("sample","Bonjour tout le monde");
        return model;
    }
}
```

welcome.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Wellcome</title>
</head>
<body>
<label th:text="${text}"></label>
<br/>
<label th:text="${sample}"></label>
</body>
</html>
```

With this example,

- All requests start from "***/hello"*** will be routed to this controller.
- The ***welcome()*** method is accessed using HTTP GET method only.
- Three URLs:
  http://localhost:8080/hello
  http://localhost:8080/hello/home
  http://localhost:8080/hello/welcome
  are routed to the same "welcome.html" page.



25

25

Spring MVC Annotations (cont.)

- Since Spring MVC 4.3, we can use
  - @GetMapping,
  - @PostMapping,
  - @PutMapping,
  - @DeleteMapping,
  - @PatchMapping

equivalently for @RequestMapping with specific HTTP methods.

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

//    @RequestMapping(value = {"", "/home", "/welcome"}, method = RequestMethod.GET)
    @GetMapping(value = {"", "/home", "/welcome"})
    public ModelAndView welcome() {
        ModelAndView model = new ModelAndView( viewName: "welcome");
        model.addObject( attributeName: "text",  attributeValue: "This is WELCOME text object.");
```

26

26

URI patterns

- @RequestMapping methods can be mapped using URL patterns. There are two alternatives:
  - PathPattern — a pre-parsed pattern matched against the URL path also pre-parsed as PathContainer. Designed for web use, this solution deals effectively with encoding and path parameters, and matches efficiently.
  - AntPathMatcher — match String patterns against a String path. This is the original solution also used in Spring configuration to select resources on the classpath, on the filesystem, and other locations. It is less efficient, and the String path input is a challenge for dealing effectively with encoding and other issues with URLs.

Some example patterns:

- `"/resources/ima?e.png"` - match one character in a path segment
- `"/resources/*.png"` - match zero or more characters in a path segment
- `"/resources/**"` - match multiple path segments
- `"/projects/{project}/versions"` - match a path segment and capture it as a variable
- `"/projects/{project:[a-z]+}/versions"` - match and capture a variable with a regex

27

27

URI patterns (cont.)

- Captured URI variables can be accessed with @PathVariable.

```
@GetMapping("/owners/{ownerId}/pets/{petId}")
public Pet findPet(@PathVariable Long ownerId, @PathVariable Long petId) {
    // ...
    return pet;
}
```

- It can declare URI variables at the class and method levels.

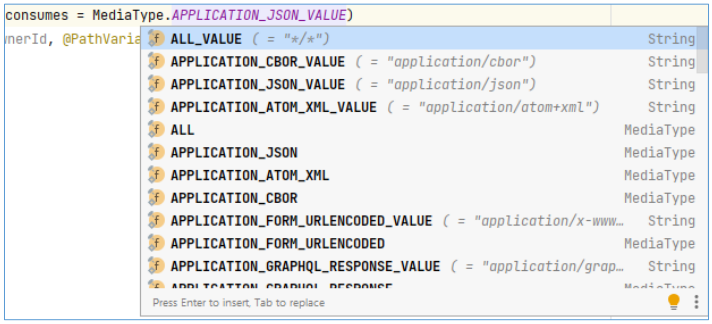
```
@Controller
@RequestMapping("/owners/{ownerId}")
public class OwnerController {
    @GetMapping("/pets/{petId}")
    public Pet findPet(@PathVariable Long ownerId, @PathVariable Long petId) {
        // ...
        return pet;
    }
}
```

28

28

Consumable/Producible Media Types

- Use consume/produces attribute to narrow the mapping by the content type
- The org.springframework.http.MediaType class provides constants for commonly used media types.



29

29

Spring MVC Annotations (cont.)

- @RequestParam
  - Annotates that a method parameter should be bound to a web request parameter
  - Can be passed through the URL query parameters.

Attributes of @RequestParam annotation

defaultValue	The default value to use as a fallback when the request parameter is not provided or has an empty value.
Name	The name of the request parameter to bind to.
Value	It is alias for name attribute.
required	Whether the parameter is required.

```
@Controller
public class AuthenticateController {
    @GetMapping("login")
    public ModelAndView login(
            @RequestParam(value = "username", required = true) String userName,
            @RequestParam("password") String password) {
        ModelAndView model = new ModelAndView();
        if (!do_login(userName, password))
            model.addObject( attributeName: "error", attributeValue: "Invalid username or password");
            //model.setView((model1, request, response) -> response.getWriter().println("<h1>Login Successfully</h1>"));
        model.setViewName("afterLoginProcess");
        return model;
    }
```

30

30

Spring MVC Annotations (cont.)

- @ModelAttribute
  - On a method argument in @RequestMapping methods to create or access an Object from the model and to bind it to the request through a WebDataBinder.
  - As a method-level annotation in @Controller or @ControllerAdvice classes that help to initialize the model prior to any @RequestMapping method invocation.
  - On a @RequestMapping method to mark its return value is a model attribute

```
@GetMapping("/addUser")
public ModelAndView addUser(@ModelAttribute("sampleUser") User user){
    //add user to the data source; check if success or not,...

    ModelAndView model=new ModelAndView();
    model.setViewName("listUser");
    return model;
}
}
```

```
@ModelAttribute
public Account addAccount(@RequestParam String number) {
    return accountRepository.findAccount(number);
}
@GetMapping("/accounts/{id}")
@ModelAttribute("myAccount")
public Account handle() {
    // ...
    return account;
}
```

31

31

Spring MVC Annotations (cont.)

- @InitBinder
  - Bind request parameters (that is, form or query data) to a model object.
  - Convert String-based request values (such as request parameters, path variables, headers, cookies, and others) to the target type of controller method arguments.
  - Format model object values as String values when rendering HTML forms.

```
@Controller
public class FormController {
    @InitBinder
    protected void initBinder(WebDataBinder binder) {
        binder.addCustomFormatter(new DateFormatter( pattern: "yyyy-MM-dd"));
    }
// ...
}
```

32

32

Exceptions

- @ExceptionHandler
  - Handle exceptions from controller methods

```
@Controller
public class SomeHowController {

    //Others @RequestMapping methods

    @ExceptionHandler({ConfigDataResourceNotFoundException.class})
    public ResponseEntity<String> handleException(IOException ex) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Resource not found");
    }

    @ExceptionHandler({HttpServerErrorException.GatewayTimeout.class})
    public ResponseEntity<String> handleGatewayTimeoutException(IOException ex) {
        return ResponseEntity.status(HttpStatus.GATEWAY_TIMEOUT).body("Gateway timeout");
    }
}
```

Recommend: declaring primary root exception mappings on a @ControllerAdvice prioritized with a corresponding order. While a root exception match is preferred to a cause, this is defined among the methods of a given controller or @ControllerAdvice class.

33

33

Session Management in Spring MVC

34

34

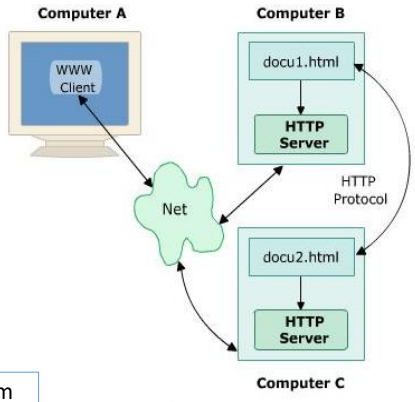
Session Tracking

Introduction

Protocol

- Is a set of rules, which governs the syntax, semantics and synchronization of communication
- Stateless Protocol: not tracked
- HTTP Protocol
  - Client – server Model
  - Request – response
  - Stateless Protocol

Sometimes, we want/need to keep (store) data from user, what will we do?



Client Server Model using HTTP Protocol

The session tracking mechanism serves the purpose tracking the client identity and other state information required throughout the session

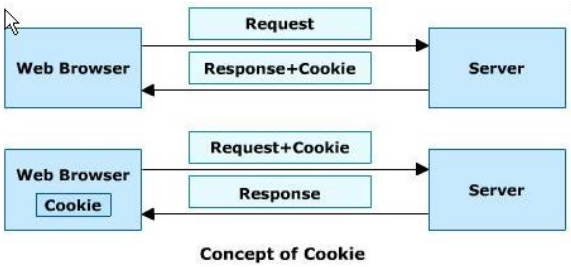
35

35

Session Tracking

Traditional techniques

- URL rewriting
- Hidden form field
- Cookies
  - Is a small piece of information sent by the web server to the client to keep track of users.
  - Cookie has values in the form of key-value pairs



Concept of Cookie

36

36

Session management

Fundamental

- Session management is the process of tracking user interactions within a web application.
- It allows applications to maintain state across multiple requests and responses, which is essential for implementing features like user authentication, shopping carts, and personalized user experiences.
- Spring Session has the simple goal of free up session management from the limitations of the HTTP session stored in the server.
- Here are some common use cases for Spring Session:
  - Distributed web applications: If you have a web application distributed across multiple servers, managing user sessions can be challenging. Spring Session can help by storing the session data in a shared database or Redis, allowing all servers to access and update session data.
  - Session scalability: In a large web application with many concurrent users, storing sessions in memory on the server can lead to scalability issues. Spring Session allows you to store session data in a persistent store, improving scalability and reducing the risk of out-of-memory errors.
  - Session backup and recovery: Storing session data in a persistent store can also provide a mechanism for backing up and recovering session data in case of server failure or downtime.

```
implementation 'org.springframework.session:spring-session-core'
```

37

37

Session management

Using HttpSession

- In Spring Boot, session management revolves around the HttpSession interface, which provides a way to store and retrieve user-specific information during the course of a session.
- Spring Boot's session management builds on top of the underlying servlet container's session handling mechanisms.
  - HttpContextServlet
  - HttpSession

```
@GetMapping("/set")
public String usingHSR(HttpServletRequest request, Model model){
   HttpSession session = request.getSession();
   session.setAttribute("att_name", "Att value");
   return "target or object";
}
```

```
@GetMapping("/set")
public String usingHSR(HttpSession session, Model model) {
   session.setAttribute("att_name", "Att value");
   return "target or object";
}
```

Application scope

```
ServletContext ctx =request.getServletContext();
ctx.setAttribute("app_scope", object);
```

38

38

Session management

Using HttpSession example

```
@RestController
public class UsingHttpSessionController {
    @GetMapping("/set-v1")
    public String set(HttpSession session, Model model){
        session.setAttribute( name: "task1", new Task( taskName: "Sample task"));
        return "task created";
    }

    @GetMapping("/get-v1")
    public Task get(HttpSession session, Model model){
        return (Task) session.getAttribute( name: "task1");
    }

    @GetMapping("/del")
    public String invalidate(HttpSession session){
        session.invalidate();
        return "session was invalidated";
    }
}
```

39

39

Session management

Using SessionAttribute annotation

- @SessionAttribute
  - Replacing of using the HttpSession object (since v4.3).
  - Retrieves the existing attribute from session to a handler method parameter.
  - Attribute: require={true|false}, default is true → exception if attribute value not present

```
@GetMapping("/get-v1a")
public Task getA(@SessionAttribute("task1") Task task, Model model){
    return task;
}
```

40

40

Session management

Using a Scoped Proxy

```
@Configuration
public class TaskProxy {
    @Bean
    @Scope(value = WebApplicationContext.SCOPE_SESSION, proxyMode = ScopedProxyMode.TARGET_CLASS)
    public Tasks tasks() {
        return new Tasks();
    }
}
```

```
@RestController
public class UsingScopeController {
    @Autowired
    private Tasks tasks;

    @GetMapping("/set-v2")
    public String sett(Model model) {
        Task task = null;
        if (!tasks.isEmpty()) {...} else {...}
        tasks.add(task);
        return "task was add";
    }
```

```
@GetMapping("/get-v2")
private Task get() {
    Task task = null;
    if (!tasks.isEmpty())
        task = tasks.peekLast();
    return task;
}

@GetMapping("/getall-v2")
private Tasks getAll() { return tasks; }
```

41

41

Session management

Using SessionAttributes annotation

- @SessionAttributes
  - @SessionAttributes is used to store model attributes in the HTTP Servlet session between requests. It is a type-level annotation that declares the session attributes used by a specific controller.
  - This typically lists the names of model attributes or types of model attributes that should be transparently stored in the session for subsequent requests to access.
  - On the first request, when a model attribute with the name, user, is added to the model, it is automatically promoted to and saved in the HTTP Servlet session. It remains there until another controller method uses a SessionStatus method argument to clear the storage,

```
@Controller
@SessionAttributes("user") ❶
public class EditUserForm {
  @ModelAttribute("user")
  public User aUser(){return new User("admin","secret");}

  @PostMapping("/users/{id}")
  public String handle(User user, BindingResult errors, SessionStatus status) {
    if (errors.hasErrors()) { /* ... */}
    status.setComplete(); ❷
    // ...
    return "...";
  }
}
```

Storing the User value in the Servlet session.

Clearing the User value from the Servlet session.

42

42

Session management

Using the @SessionAttributes Annotation

```
RestController
SessionAttributes("tasks")
ublic class UsingAttributeController {

    @ModelAttribute("tasks")
    public Task getTasks() { return new Task( taskName: "Sample Task"); }

    @GetMapping("/set-v3")
    public String set(Model model,
                      @ModelAttribute("tasks") Tasks tasks,
                      RedirectAttributes attributes) {
        Task task = null;
        if (!tasks.isEmpty()) {...} else {...}
        tasks.add(task);
        model.addAttribute( attributeName: "tasks", tasks);

        attributes.addFlashAttribute( attributeName: "tasks", tasks);
        return "task added";
    }
```

By using addFlashAttribute we are telling the framework that we want our TASKS to survive the redirect without needing to encode it in the URL.

```
@GetMapping("/getall-v3")
private Tasks getAll(Model model, @ModelAttribute("tasks") Tasks tasks) {
    return (Tasks) model.getAttribute( attributeName: "tasks");
}
```

43

43

Session management

Session Store in database

- Spring Session provides a layer of abstraction between the application and the session management. It allows the session data to be stored in various persistent stores, such as relational databases, NoSQL databases, and others.
- With Spring Session, you can use the same API to manage sessions, regardless of the persistent store used. This makes it easier to switch between stores without changing the application code.
- Spring Session also provides features such as session expiry and cross-context communication between different web applications.

```
implementation 'org.springframework.session:spring-session-jdbc'
//implementation 'org.springframework.session:spring-session-data-mongo'
//implementation 'org.springframework.session:spring-session-data-redis'
//implementation 'org.springframework.session:spring-session-data-.....'
```

44

44

Session management

Configure session database

- Spring boot auto-detect the session implementation based on the dependencies --> add these two dependencies to your project

```
spring-boot-starter-data-jpa
spring-session-jdbc
```

- Configure the data-source in application.properties file.

→Tables will be created in your database

```
spring.datasource.url=jdbc:h2:mem:test
spring.datasource.username=root
spring.datasource.password=root
spring.h2.console.enabled=true
```

SELECT * FROM SPRING_SESSION;

PRIMARY_ID	SESSION_ID	CREATION_TIME	LAST_ACCESS_TIME	MAX_INACTIVE_INTERVAL	EXPIRY_TIME	PRINCIPAL_NAME
d85be90a-ea45-44d4-9dab-d1a8d1a9400f	5331c66f-9d8b-4b7f-b850-28c4e104f1f0	1609669322600	1609669558561	1800	1609671358561	admin
a47fc953-86be-4ea1-977d-4cc8bf539a24	bd1cd198-1006-4cd9-b388-64f5f0bd78ac	1609669573380	1609669583366	1800	1609671383366	user1
ff7f3abc-1b5a-4f48-9c38-bd29b67daa4f	e4e3919a-0897-4119-8660-11403491c3c2	1609669594659	1609669598541	1800	1609671398541	user2

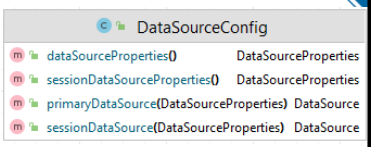
(3 rows, 1 ms)

SELECT * FROM SPRING_SESSION_ATTRIBUTES;

SESSION_PRIMARY_ID	ATTRIBUTE_NAME	ATTRIBUTE_BYTES
d85be90a-ea45-44d4-9dab-d1a8d1a9400f	SPRING_SECURITY_CONTEXT	aced00057372003d6f72672e737072696e67672616d6
a47fc953-86be-4ea1-977d-4cc8bf539a24	SPRING_SECURITY_CONTEXT	aced00057372003d6f72672e737072696e67672616d6
ff7f3abc-1b5a-4f48-9c38-bd29b67daa4f	SPRING_SECURITY_CONTEXT	aced00057372003d6f72672e737072696e67672616d6

Session management

Using multi-data-source for storing session



1. Main data-source was configured as normal
2. Session data-source should be configured difference

```
@Configuration
@EnableConfigurationProperties(JdbcSessionProperties.class)
public class DataSourceConfig {
  @Bean
  @Primary
  @ConfigurationProperties("spring.datasource")
  @Qualifier("dataSource")
  public DataSourceProperties dataSourceProperties() {
    return new DataSourceProperties();
  }
  @Bean
  @Primary
  public DataSource primaryDataSource(
      @Qualifier("dataSource") DataSourceProperties dataSourceProperties) {
    return dataSourceProperties
        .initializeDataSourceBuilder()
        .type(HikariDataSource.class)
        .build();
  }
```

```
#create session = always/embedded/never
spring.session.jdbc.initialize-schema=always
```

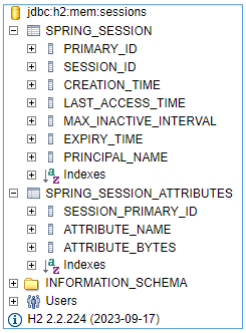
```
//Properties for primary data-source
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.datasource.url=jdbc:mariadb://localhost:3306/shopping
spring.datasource.username=root
spring.datasource.password=sapassword
```

Session management

Using multi-data-source for storing session (cont.)

```
@Bean
@ConfigurationProperties("session.datasource")
@Qualifier("sessionDataSourceProperties")
public DataSourceProperties sessionDataSourceProperties() {
   return new DataSourceProperties();
}

@Bean
@Qualifier("sessionDataSource")
@SpringSessionDataSource
public DataSource sessionDataSource(
      @Qualifier("sessionDataSourceProperties")
       DataSourceProperties sessionDataSourceProperties) {
   return sessionDataSourceProperties
         .initializeDataSourceBuilder()
         .type(HikariDataSource.class)
         .build();
}
}
```

```
# Properties for session database
session.datasource.url=jdbc:h2:mem:sessions
session.datasource.username=sa
session.datasource.password=password
```



47

47

Handling Cookie in Spring MVC

49

Handling Cookie in Spring MVC

Introduction

- An HTTP Cookie (also known as a web cookie or browser cookie) is a small piece of information stored by the server in the user's browser.
- The server sets the cookies while returning the response for a request made by the browser.
- The browser stores the cookies and sends them back with the next request to the same server.
- Cookies are generally used for session management, user-tracking, and to store user preferences.
- Cookies help server remember the client across multiple requests. Without cookies, the server would treat every request as a new client.

50

50

Handling Cookie in Spring MVC

Reading and Setting HTTP Cookie

- The Spring Framework provides the @CookieValue annotation to get the value of any HTTP cookie without iterating over all the cookies fetched from the request.
- This annotation can be used to map the value of a cookie to the controller method parameter.

```
@GetMapping("/")
public String readCookie(@CookieValue(value = "username", defaultValue = "HaiVV") String username) {
   return "Hey! My username is " + username;
}
```

- To set a cookie, we can use HttpServletResponse class's method addCookie().

```
@GetMapping("/change-username")
public String setCookie(HttpServletResponse response) {
   // create a cookie
   Cookie cookie = new Cookie("username", "Vo Van Hai");
   //add cookie to response
   response.addCookie(cookie);
   return "Username is changed!";
}
```

51

51

Handling Cookie in Spring MVC

Reading All Cookies

- Instead of using @CookieValue annotation, we can also use HttpServletRequest class as a controller method parameter to read all cookies. This class provides getCookies() method, which returns all cookies sent by the browser as an array of Cookie.

```
@GetMapping("/all-cookies")
public String readAllCookies(HttpServletRequest request) {

   Cookie[] cookies = request.getCookies();
   if (cookies != null) {
      return Arrays.stream(cookies)
            .map(c -> c.getName() + "="
                  + c.getValue()).collect(Collectors.joining(", "));
   }
   return "No cookies";
}
```

52

52

Handling Cookie in Spring MVC

Cookie Expiration

- If no expiration time is specified for a cookie, it lasts as long as the session is not expired. Such cookies are called session cookies. Session cookies remain active until the user closes their browser or clears their cookies. The username cookie created above is in fact a session cookie.
- But you can override this default behavior and set the cookie expiration time using setMaxAge() method of Cookie class.

```
// create a cookie
Cookie cookie = new Cookie("username", "Jovan");
cookie.setMaxAge(7 * 24 * 60 * 60); // expires in 7 days
//add cookie to response
response.addCookie(cookie);
```

- Now, instead of expiring when the browser is closed, the username cookie will remain active for the next 7 days. Such cookies, which expire at a specified date and time, are called permanent cookies.
- The expiry time passed to setMaxAge() method is in seconds. The expiry date and time is relative to the client where the cookie is being set, not the server.

53

53

Handling Cookie in Spring MVC

Secure Cookie

- A secure cookie is the one which is only sent to the server over an encrypted HTTPS connection.
- Secure cookies cannot be transmitted to the server over unencrypted HTTP connections.

```
// create a cookie
Cookie cookie = new Cookie("username", "Jovan");
cookie.setMaxAge(7 * 24 * 60 * 60); // expires in 7 days
cookie.setSecure(true);
//add cookie to response
response.addCookie(cookie);
```

54

54

Handling Cookie in Spring MVC

HttpOnly Cookie

- HttpOnly cookies are used to prevent cross-site scripting (XSS) attacks and are not accessible via JavaScript's Document.cookie API.
- When the HttpOnly flag is set for a cookie, it tells the browser that this particular cookie should only be accessed by the server.

```
// create a cookie
Cookie cookie = new Cookie("username", "Jovan");
cookie.setMaxAge(7 * 24 * 60 * 60); // expires in 7 days
cookie.setSecure(true);
cookie.setHttpOnly(true);

//add cookie to response
response.addCookie(cookie);
```

55

55

Handling Cookie in Spring MVC

Cookie Scope

- If the scope is not specified, a cookie is only sent to the server for a path that was used to set it in the browser.
- We can change this behavior using setPath() method of the Cookie class. This sets the Path directive for the cookie.

```
// create a cookie
Cookie cookie = new Cookie("username", "VoVan");
cookie.setMaxAge(7 * 24 * 60 * 60); // expires in 7 days
cookie.setSecure(true);
cookie.setHttpOnly(true);
cookie.setPath("/"); // global cookie accessible every where

//add cookie to response
response.addCookie(cookie);
```

56

56

Handling Cookie in Spring MVC

Deleting Cookie

- To delete a cookie, set the Max-Age directive to 0 and unset its value.
- You must also pass the same other cookie properties you used to set it. Don't set the Max-Age directive value to -1. Otherwise, it will be treated as a session cookie by the browser.

```
// create a cookie
Cookie cookie = new Cookie("username", null);
cookie.setMaxAge(0);
cookie.setSecure(true);
cookie.setHttpOnly(true);
cookie.setPath("/");

//add cookie to response
response.addCookie(cookie);
```

57

57

View Technologies

58

Introduction

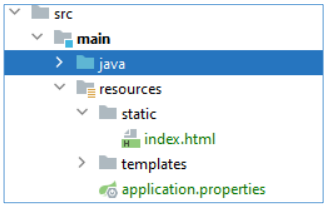
- The use of view technologies in Spring MVC is pluggable.
- Deciding to use which view technologies is primarily a matter of a configuration change.
- The views of a Spring MVC application live within the internal trust boundaries of that application.
  - Views have access to all the beans of your application context.
  - As such, it is not recommended to use Spring MVC's template support in applications that are editable by external sources since this can have security implications.

59

59

Static Content

- By default, Spring Boot serves static content from a directory called /static (or /public or /resources or /META-INF/resources) in the classpath or from the root of the ServletContext.



60

60

Template Engines

- As well as REST web services, you can also use Spring MVC to serve dynamic HTML content.
- Spring MVC supports a variety of templating technologies, including **Thymeleaf**, FreeMarker, and JSPs. Also, many other templating engines include their own Spring MVC integrations.
- Spring Boot includes auto-configuration support for the following templating engines:
  - FreeMarker
  - Groovy
  - Thymeleaf
  - Mustache

61

61

Thymeleaf

- Thymeleaf is a modern server-side Java template engine for both web and standalone environments.
- Thymeleaf's main goal is to bring elegant natural templates to your development workflow — HTML that can be correctly displayed in browsers and also work as static prototypes, allowing for stronger collaboration in development teams.
- With modules for Spring Framework, a host of integrations with your favourite tools, and the ability to plug in your own functionality, Thymeleaf is ideal for modern-day HTML5 JVM web development — although there is much more it can do.

62

62

Standard Expression Syntax

- **Simple expressions:**
  - **Variable Expressions:** ${...}
  - **Selection Variable Expressions:** *{...}
  - **Message Expressions:** #{...}
  - **Link URL Expressions:** @{...}
- **Literals**
  - **Text literals:** 'one text', 'Another one!',...
  - **Number literals:** 0, 34, 3.0, 12.3,...
  - **Boolean literals:** true, false
  - **Null literal:** null
  - **Literal tokens:** one, sometext, main,...
- **Text operations:**
  - **String concatenation:** +
  - **Literal substitutions:** The name is ${name}

63

63

Standard Expression Syntax (cont.)

- Arithmetic operations:
  - Binary operators: +, -, *, /, %
  - Minus sign (unary operator): -
- Boolean operations:
  - Binary operators: and, or
  - Boolean negation (unary operator): !, not
- Comparisons and equality:
  - Comparators: >, <, >=, <= (gt, lt, ge, le)
  - Equality operators: ==, != (eq, ne)
- Conditional operators:
  - If-then: (if) ? (then)
  - If-then-else: (if) ? (then) : (else)
  - Default: (value) ?: (defaultvalue)

64

64

Object available

	Object	Description
1	#ctx	the context object
2	#vars	the context variables
3	#locale	the context locale
	Only in Web Contexts	
5	#request	the HttpServletRequest object
6	#response	the HttpServletResponse object
7	#session	the HttpServletSession object
7	#servletContext	the ServletContext object

65

65

Creating Forms

- Requires to specify the command object by using th:object attribute in the form tag.
  - Value of the th:object attribute must be inside the expression ${…} specifying ONLY the name of a model attribute.
  - Only one th:object attribute in a form tag.

```
<form method="post" role="form" th:action="@{/products/saveProduct}" th:object="${product}">
    <div>
        <div><label>Product name:</label>
            <input placeholder="Product name" th:field="${product.productName}" type="text"/>
        </div>
        <div>
            <label>Product create Date</label>
            <input placeholder="Created date" th:field="${product.procedureDate}" type="date"/>
        </div>
        <div...>
        <input th:field="${product.productId}" type="hidden"/>
    </div>
</form>
```

66

66

Display model attributes

- Simple attribute
  - Using the `th:text="${attribute_name}` tag attribute to display
  - Ex: model.addAttribute("serverTime", Instant.*now*());

```
<span th:text="${serverTime}"/>
```

- Collection attribute
  - Use the `th:each` attribute for iterating.

```
<tbody>
<tr th:each="product:${products}">
    <td th:text="${product.productId}"/>
    <td th:text="${product.productName}"/>
    <td>
        <a th:href="@{/products/delete(id=${product.productId})}">Delete</a>
    </td>
    <td>
        <a th:href="@{/products/showUpdateForm(id=${product.productId})}">Update</a>
    </td>
</tr>
</tbody>
```

67

67

Conditional Evaluation

- `th:if="${condition}`
  - Display a section of the view if the condition is true
- `th:unless="${condiation}`
  - Display a section of the view if the condition is false.

```
<td>
   <span th:if="${student.gender}=='M'" th:text="Male"/>
   <span th:unless="${student.gender}=='M'" th:text="Female"/>
</td>
```

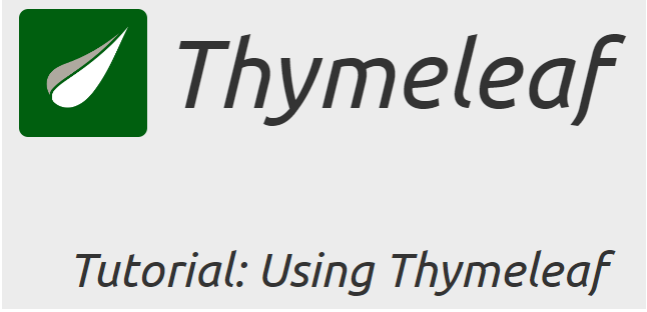
- The `th:switch` and `th:case` are used to display content conditionally suing the switch statement structure.

```
<td th:switch="${student.gender}">
   <span th:case="M" th:text="Male"/>
   <span th:case="F" th:text="Female"/>
</td>
```

68

68

READ MORE »



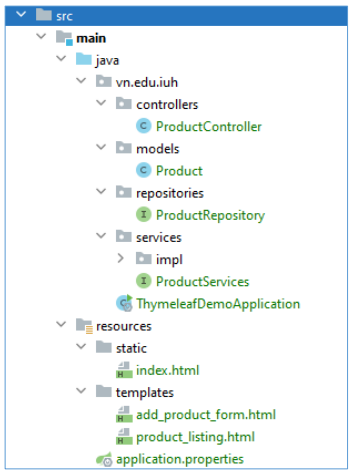
https://www.thymeleaf.org/documentation.html

69

69

Example

- An example with CRUD operators on a table.
- Use Spring MVC, Spring Data JPA, and Thymeleaf template.

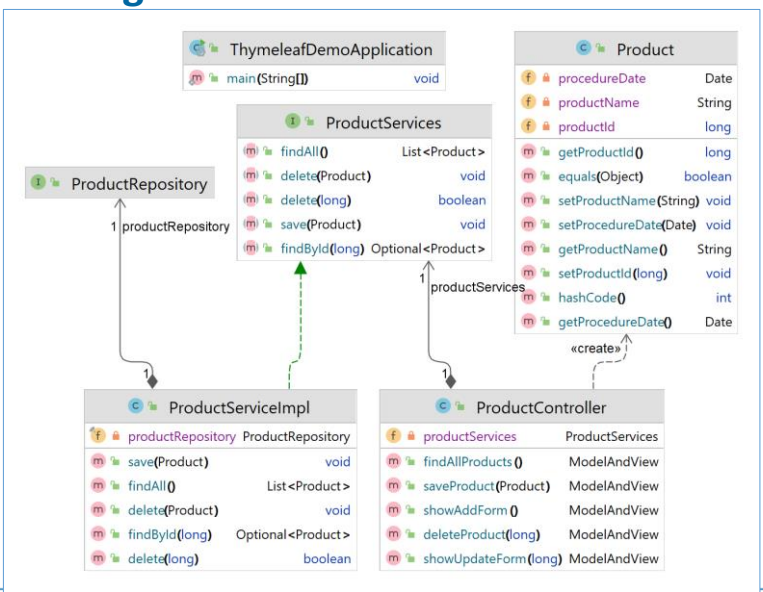



```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'org.mariadb.jdbc:mariadb-java-client'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

70

70

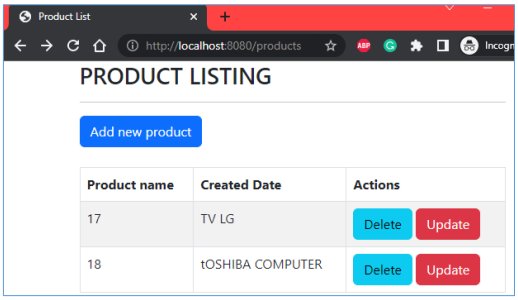
Class diagram

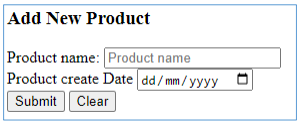


71

71

Views



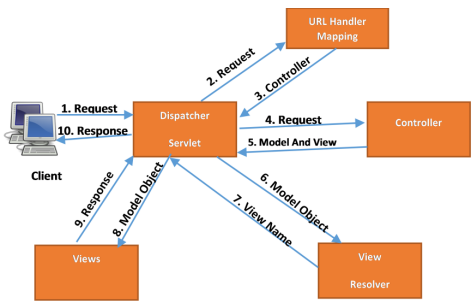


72

72

Summary

- Spring Web MVC Framework introduction
- Spring MVC Configuration
- Spring Web on Servlet Stack with Springboot
- Session Management in Spring MVC
- Handling Cookie in Spring MVC
- View Technologies



73

73

Thank you for Listening!

74

74