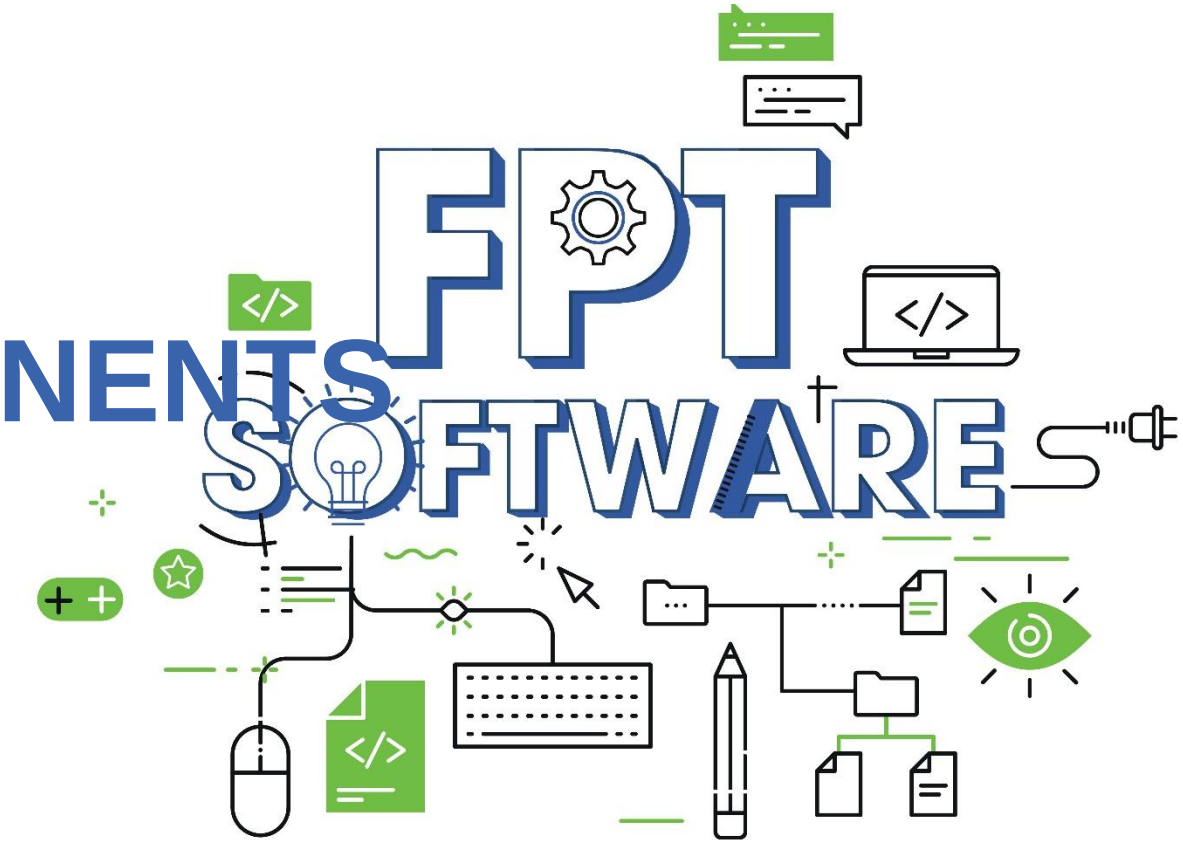
# SPRING MVC COMPONENTS




Design by: DieuNT1

# Agenda

1. HttpSession, @SessionAttributes, @SessionAttribute
2. Spring Expression Language (SpEL)
3. RedirectView and RedirectAttributes
4. Question and Answer

# Lesson Objectives

1. Understand the concept of HTTP sessions
2. Implement session-based authentication
3. Able to use SpEL, RedirectView and RedirectAttributes

Section 1

# Session Management

# What is Session Management?

**Session management** is the process of securely **handling multiple requests** to a web-based application or service from a single user or entity.

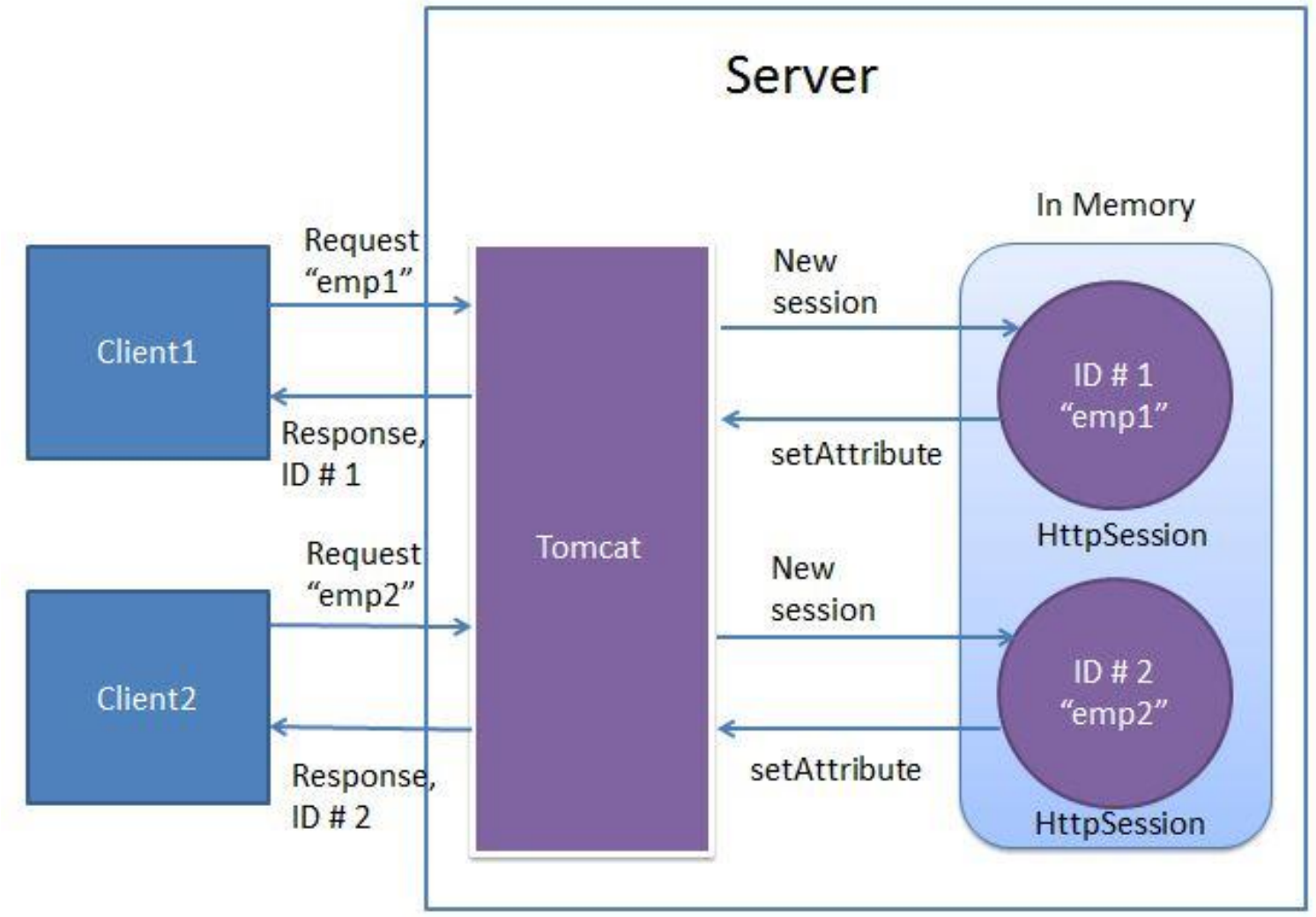
- HTTP is used to communicate between websites and browsers, and a session is a series of HTTP requests and transactions created by the same user.
- As **HTTP protocol is stateless**, and to keep track of customer behavior, we need session management.
- Session Management is a web container framework used to store session data for a specific user.

# What is Session Management?

You can handle the session in one of the following ways-:

- **A Cookies:**
  - ✓ is a data sent from a website and saved by the user's web browser on the user's computer as the user browses.
- **Hidden form field:**
  - ✓ is a hidden data, which will not be shown to user and can not be modified. However, when the user submits the form, hidden data would be sent.
- **URL Rewriting:**
  - ✓ is the method of modifying the URL parameters.
- **HttpSession:**
  - ✓ enables data to be associated with individual visitors.

# What is Session Management?



# What is Session Management?

- When developing web applications, we often need to refer to the same attributes in several views.
- **For example**, we may have shopping cart contents that need to be displayed on multiple pages.
- A good location to *store those attributes is in the user's session*.
- Have **some strategies for working with a session attribute**:
  - ✓**Directly add one attribute to session**
  - ✓**Using a scoped proxy**
  - ✓**Using the @*SessionAttributes* annotation**

# Session attributes in Spring MVC

- **Directly add one attribute to session:**

```
@RequestMapping(method = RequestMethod.GET)
public String testMestod(HttpServletRequest request) {
        ShoppingCart cart = (ShoppingCart) request.getSession()
                                                .setAttribute("cart", value);
        return "testJsp";
}
```

- **Using @Scopes("session"):**

```
@Component
@Scope("session")
public class User {
        String userName;
        ...
        /* setter getter*/
}
```

*// then inject class in each controller that you want*

```
@Autowired
private User user
```

# Session attributes in Spring MVC

- **The AOP proxy injection : in spring -xml:**

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.1.xsd">

    <bean id="user" class="com.User" scope="session">
        <aop:scoped-proxy/>
    </bean>

</beans>
```

*// then inject class in each controller that you want*

```
@Autowired
private User user
```

# Session attributes in Spring MVC

- **Make ModelAttribute in session By @SessionAttributes("ShoppingCart"):**

```
public String index (@ModelAttribute("ShoppingCart") ShoppingCart shoppingCart,
                     SessionStatus sessionStatus) {
        //Spring v4 you can modify session status by sessionStatus.setComplete();
}
```

// or you can add Model to entire Controller class like:

```
@Controller
@SessionAttributes("ShoppingCart")
@RequestMapping("/req")
public class MyController {
    @ModelAttribute("ShoppingCart")
    public ShoppingCart getShopCart (....) {
        return new ShoppingCart(....); //get From DB Or Session
    }
}
```

# @SessionAttributes example

- **@SessionAttributes** annotation is used to store the model attribute in the session. This annotation is used at controller class level.

```
@SessionAttributes("user")
public class LoginController {
        @ModelAttribute("user")
        public User setUpUserForm() {
                return new User();
        }
}
```

- **@SessionAttribute** annotation is used to retrieve the existing attribute from session that is managed globally and it is used at method parameter as shown follows.

```
@GetMapping("/info")
public String userInfo(@SessionAttribute("user") User user) {
        //... //... return "user";
}
```

# @SessionAttributes example

- **Controller** class:

```
@Controller
@SessionAttributes("user")
public class UserController {
    /**
     * Add user in model attribute.
    */
    @ModelAttribute("user")
    public User setUpUserForm() {
        return new User();
    }

    @PostMapping("/dologin")
    public String doLogin(@ModelAttribute("user") User user, Model model) {

        // Implement your business logic
        if ("admin".equals(user.getUsername())
                && "admin".equals(user.getPassword())) {
            return "index";
        } else {
            model.addAttribute("message", "Login failed. Try again.");
            return "login";
        }
    }
}
```

# @SessionAttribute example

- **Controller** class:

```
    /*
     * Get user from session attribute
     */
    @GetMapping("/info")
    public String userInfo(@SessionAttribute("user") User user) {

        System.out.println("User Name: " + user.getUsername());

        return "index";
    }
}
```

# View

- Login.jsp

```
<form:form action="${pageContext.request.contextPath}/dologin" method="post" modelAttribute="user">
        <h2 class="text-center">Log in</h2>
        <label style="color: red">${errorMessage}</label><!-- JSP Expression -->
        <div class="form-group">
            <input type="text" name="username" class="form-control"

           placeholder="Username" required="required">
        </div>
        <div class="form-group">
            <input type="password" name="password" class="form-control"

           placeholder="Password" required="required">
        </div>
        <div class="form-group">
            <button type="submit" class="btn btn-primary btn-block">Log in</button>
        </div>
        <div class="clearfix">
            <label class="float-left form-check-label"><input type="checkbox"> Remember me</label>
            <a href="#" class="float-right">Forgot Password?</a>
        </div>
  </form:form>
```

Section 2

# SPRING EXPRESSION LANGUAGE (SpEL)

# Introduction

- **Spring Expression Language** (SpEL) is a powerful expression language, which can be used for querying and manipulating an object graph at runtime.
  - ✓ **SpEL** supports standard *mathematical operators*, *relational operators*, *logical operators*, *conditional operators*, *collections* and *regular expressions*, etc.
  - ✓ It can be used to *inject a bean* or *a bean property* into another bean.
  - ✓ *Method invocation of a bean* is also supported.

# SpEL Examples

- **Example 1**: The logical operators, (&&) or (||) and not (!), are supported. The textual equivalents can also be used.
  - ✓ (1) First let's define the **MyOtherGlass** POJO:

```
package fa.training.entities;

public class MyOtherGlass {
    private boolean empty;
    private boolean halfEmpty;
    private int volume;
    private int maxVolume;
    private boolean largeGlass;

    public MyOtherGlass() {

    }
    // getter and setter moethod
}
```

# SpEL Examples

- (2) Let's now create our spring configuration file where we define the **smallGlass** bean and the **largeGlass** bean.

```
<bean id="smallGlass" class="fa.training.entities.MyOtherGlass">
        <constructor-arg name="volume" value="5" />
        <constructor-arg name="maxVolume" value="10" />
        <property name="largeGlass"
                value="#{smallGlass.maxVolume ge 20 and smallGlass.maxVolume le 30}" />
</bean>

<bean id="largeGlass" class="fa.training.entities.MyOtherGlass">
        <constructor-arg name="volume" value="5" />
        <constructor-arg name="maxVolume" value="30" />
        <property name="largeGlass"
                value="#{largeGlass.maxVolume ge 20 and largeGlass.maxVolume le 30}" />
</bean>
```

# SpEL Examples

- **(3) Test**

```
LogUtils.getLogger().info(smallGlass.isLargeGlass());

LogUtils.getLogger().info(largeGlass.isLargeGlass());
```

- **Results:**

```
[INFO ] 2020-11-14 15:12:47.678 [http-nio-8080-exec-2] LogUtils - false
[INFO ] 2020-11-14 15:12:47.680 [http-nio-8080-exec-2] LogUtils - true
```

# SpEL Examples

- **Example 2:**

```
<bean id="officeAddress" class="fa.training.entities.Address">
        <property name="number" value = "101" />
        <property name="street" value = "#{'M I Road'}" />
        <property name="city" value = "Jaipur" />
        <property name="state" value = "Rajasthan" />
        <property name="pinCode" value = "#{'302001'}" />
    </bean>

    <bean id="employee" class="fa.training.entities.Employee">
        <property name="empId" value = "1001" />
        <property name="empName" value = "Ram" />
        <!--  Bean reference through SpEL -->
        <property name="officeAddress" value = "#{officeAddress}" />
        <property name="officeLocation" value = "#{officeAddress.city}" />
        <!--  Method invocation through SpEL -->
        <property name="employeeInfo" value = "#{officeAddress.getAddress('Ram')}" />
    </bean>
```

# SpEL - properties files

- Create **DBConfig.properties** file:

```
driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
url=jdbc:sqlserver://localhost:1433;databaseName=DBName
username=sa
password=12345678
```

- **dispatcher-servlet.xml** file:

```
<!- Properties Spring -->
<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations">
                <list>
                        <value>classpath:DBConfig.properties</value>
                </list>
   </property>
</bean>
```

# SpEL - properties files

- **dispatcher-servlet.xml** file:
  - ✓ The **context:property-placeholder** tag is used to externalize properties in a separate file.
  - ✓ It automatically configures **PropertyPlaceholderConfigurer**, which replaces the ${} placeholders, which are resolved against a specified properties file (as a Spring resource location).

```
<context:property-placeholder
        location="classpath:data.properties, classpath:DBConfig.properties"
        ignore-unresolvable="true" />
```

*Default resource location: src/main/resources.*

# SpEL - properties files

- **dispatcher-servlet.xml** file:

```
<!-- DataSourse -->
<bean id="dataSource"
                class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="${driver}" />
        <property name="url" value="${url}" />
        <property name="username" value="${username}" />
        <property name="password" value="${password}" />
</bean>
```

# SpEL Using Annotation

- SpEL expressions begin with the # symbol, and are wrapped in braces: **#{expression}**.
- Properties can be referenced in a similar fashion, starting with a $ symbol, and wrapped in braces: **${property.name}**.

```
@Value("#{19 + 1}") // 20
private double add;

@Value("#{'String1 ' + 'string2'}") // "String1 string2"
private String addString;

@Value("#{20 - 1}") // 19
private double subtract;

@Value("#{10 * 2}") // 20
private double multiply;

@Value("#{36 / 2}") // 19
private double divide;
```

```
@Value("#{36 div 2}") // 18, the same as for / operator
private double divideAlphabetic;

@Value("#{37 % 10}") // 7
private double modulo;

@Value("#{37 mod 10}") // 7, the same as for % operator
private double moduloAlphabetic;

@Value("#{2 ^ 9}") // 512
private double powerOf;

@Value("#{(2 + 2) * 2 + 9}") // 17
private double brackets;
```

# SpEL Using Annotation

- **Example**: The **@Component** annotation for registering the bean and **@Value** for setting values into bean properties.

```
@Component
public class Address {
    @Value("100")
    private String houseNo;
    @Value("The Mall")
    private String street;
    @Value("Shimla")
    private String city;
    @Value("HP")
    private String state;
    // As SpEL literal
    @Value("#{'171004'}")
    private String pinCode;

            // getter and setter methods
}
```

```
@Component
public class Person {
    @Value("#{'Suresh'}")
    private String name;
    @Value("34")
    private int age;
    // SpEL Bean reference
    @Value("#{address}")
    private Address address;
    @Value("#{address.city}")
    private String personCity;
    // SpEL Method invocation
    @Value("#{person.getInfo()}")
    private String personInfo;

            // getter and setter methods
}
```

# SpEL Using Annotation - properties files

- Create a **data.properties** file:

```
technic_name=Java Web,Android,.Net,C/C++,Angular,React
MSG1=Sorry, your username or password is incorrect. Please try again!
MSG2=Username must be not empty!
MSG3=Password must be not empty!
MSG4=You must input all required fields!
MSG5=Wrong format!
```

- We will add some fields to read the configuration from **employee.properties** using @Value annotation.
- **Example**:

```
@Controller
@SessionAttributes("user")
@PropertySource(value = "classpath:data.properties")
public class UserController {

    @Value("#{'${MSG1}'}")
    private String msg1;
    @Value("#{'${MSG2}'}")
    private String msg2;
    @Value("#{'${MSG3}'}")
    private String msg3;
    @Value("#{'${MSG4}'}")
    private String msg4;
    @Value("#{'${MSG5}'}")
    private String msg5;

    @Value("#{'${technic_name}'.split(',')}")
    private List<String> technics;
}
```

Section 3

# RedirectAttributes and RedirectView

# RedirectAttributes class

- A specialization of the Model interface that controllers can use to select attributes for a redirect scenario.
- This interface also provides a way to **add flash attributes** and they will be automatically propagated to the "output" FlashMap of the current request.
- A **RedirectAttributes model is empty** when the method is called and is never used unless the method returns a redirect view name or a RedirectView.
- **After the redirect**, flash attributes are automatically added to the model of the controller that **serves the target URL**.

# Methods

- **addFlashAttribute**:("key", "value")
  - ✓ Flash Attributes are attributes which lives in session for short time.
  - ✓ It is used to propagate values from one request to another request and then automatically removed.
  - ✓ Handling flash attributes are achieved using **FlashMap** and **FlashMapManager**.
  - ✓ But in annotated spring MVC controller, it can be achieved with **RedirectAttributes**.
- **addAttribute**(“attributeName”,  “attributeValue”)
  - ✓ Add the supplied attribute under the supplied name.

# Add Flash Attributes

```
@RequestMapping(value = "mybook", method = RequestMethod.GET)
public ModelAndView book() {
    return new ModelAndView("book", "book", new Book());
}

@RequestMapping(value = "/save", method = RequestMethod.POST)
public RedirectView save(@ModelAttribute("book") Book book, RedirectAttributes redirectAttrs) {
    redirectAttrs.addAttribute("msg", "Hello World!");
    redirectAttrs.addFlashAttribute("book", book.getBookName());
    redirectAttrs.addFlashAttribute("writer", book.getWriter());

    RedirectView redirectView = new RedirectView();
    redirectView.setContextRelative(true);
    redirectView.setUrl("/hello/{msg}");
    return redirectView;
  }
```

# Fetch Flash Attributes

- To fetch flash attributes we have two approaches.
  - ✓The first one is by using **Model** as an argument in the @**RequestMapping** method and fetch the flash attribute as below.

```
model.asMap().get("key");
```

```
@RequestMapping(value = "/hello/{msg}", method = RequestMethod.GET)
  public String hello(Model model, RedirectAttributes redirectAttrs,
                @PathVariable("msg") String msg, HttpServletRequest request) {
    System.out.println("Message:" + msg);

    System.out.println("Fetch Flash Attributes By using Model");
    System.out.println("Book Name:" + model.asMap().get("book"));
    System.out.println("Writer:" + model.asMap().get("writer"));

    return "redirect:/success.jsp";
  }
```

# Fetch Flash Attributes

- Another approach is by using **RequestContextUtils** . The static method ***getInputFlashMap***() accepts HttpServletRequest as an argument and it returns a Map. Now using keys we can fetch flash attributes.

```
Map<String, ?< flashMap = RequestContextUtils.getInputFlashMap(request);
flashMap.get("key");
```

# Summary

- HttpSession, @SessionAttributes, @SessionAttribute
- Spring Expression Language (SpEL)
- RedirectView and RedirectAttributes

# THANK YOU!