Advanced Java

Agenda

- JSP Custom Tags
- Filters
- Listeners

JSP Custom Tags

- Custom tags are used to combine presentation logic and business logic.
- There are two types of tags
    - Classic tags: inherited from Tag interface or TagSupport class.
    - Simple tags : inherited from SimpleTag interface or SimpleTagSupport class.
- SimpleTag implementation steps
    - step 0. Decide tag name, attributes & body type.
    - step 1. Implement tag handler class inherited from SimpleTagSupport
        - Constructor
        - Fields & getter/setter = attributes
        - setJspBody() = if there is body
        - doTag() = Logic implementation
    - step 2. Implement .tld file to define tag syntax.
    - step 3. In JSP, use <%@ taglib ... %> & tag.
- SimpleTag life cycle
    1. A new tag handler instance is created each time by the container using default constructor.
    2. The setJspContext() and setParent() methods are called.

3. The setters for each attribute defined for tag.
4. If a body exists, the setJspBody() method is called.
5. The doTag() method is called.
6. The doTag() method returns and all variables are synchronized.

Aspect Oriented Programming

- Aspect Oriented Programming enables us to implement additional functionalities (a.k.a. cross-cutting concerns) without modifying core business logic.
- Cross-cutting concerns: Security, Logging, Monitoring, Performance measurement, Transaction management, ...
- Typically AOP is done as pre-processing or post-processing or both.

Filters

- Filters is way of implementing AOP in Java EE applications. Filters are used to perform pre-processing, post-processing or both for each request.
- Multiple filters can be executed in a chain/stack before/after handling request.
- javax.servlet.Filter interface is used to implement Filters.
    - void init(FilterConfig filterConfig);
    - void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain);
    - void destroy();
- https://docs.oracle.com/javaee/7/api/javax/servlet/Filter.html
- Can be configured with @WebFilter or in web.xml (similar to servlets).

Listeners

- Listeners are used to handle application level events.
- There are many listener interfaces. Refer docs.
    - ServletContextListener -- To handle application initialized and destroy events.
        - void contextInitialized(ServletContextEvent sce);
            - Called by web container when application is started/deployed i.e. servlet context is created.
            - Example: load and register JDBC driver, initialize a connection pool, ...
        - void contextDestroyed(ServletContextEvent sce);
            - Called by web container when application is stopped i.e. when web server shutdown.

        - Example: release a connection pool, ...
      - Implement this listener to perform one time initialization and destruction for the whole application.
    - HttpSessionListener -- To handle session initialized and destroy events.
      - sessionCreated() method is called when req.getSession() is called first time for any client. You may add any session attribute in it immediately after creating session.
      - sessionDestroyed() method is called when session is invalidated or time-out.
    - ServletRequestListener -- To handle request initialized and destroy events.
    - ServletContextAttributeListener
    - HttpSessionAttributeListener
    - ServletRequestAttributeListener
- Listener class must implement one or more listener interface.
- Can be configured with @WebListener OR in web.xml.

```
<listener>
    <listener-class>pkg.MyListener</listener-class>
</listener>
```

Application architectures

Model-View architecture

- Also called as Model-1 architecture.
- Application is divided in two major parts.
  - Model: Data handling and Business logic -- Java bean.
  - View: Presentation/appearence of the data -- JSP.
- Java beans are tightly coupled with JSP pages (jsp:useBean). Also a JSP page may be tightly coupled with other JSP pages (e.g. href="...", action="..."). Any changes into bean or jsp will lead to changes in multiple other components.
- This architecture is suitable for small applications.

Model-View-Controller architecture

- Also called as Model-2 architecture.
- Application is divided in three major parts.
    - Model: Data handling and Business logic -- Java bean.
    - View: Presentation/appearence of the data -- JSP.
    - Controller: Handles communication/navigation between views and models.
- Models and views are loosely coupled with each other. Their navigation is centrally controlled by controller layer.
    - View1 --> Controller --> View2
    - View1 --> Controller --> Java Bean and View2
- This architecture is suitable for bigger applications.
- Typically controller is implemented as a servlet that forwards the request to the next component.
- There are popular frameworks which implements MVC pattern (e.g. JSF, Spring MVC, Struts MVC). Spring MVC has predefined controller called as "DispatcherServlet".

Maven

- Maven is a build tool.
- Configuration file: pom.xml
- https://jenkov.com/tutorials/maven/maven-tutorial.html
- https://youtu.be/lMXBrlVFYA0?si=okJy3QuOxGdouysj