Advanced Java

Agenda

- Applet vs Servlets
- CGI vs Servlets
- JSP
  - Introduction
  - Syntax
  - Life cycle
  - Implicit objects
  - Standard actions
  - Java beans
  - ~~Expression lanaguage~~
  - ~~JSTL~~

Movie Review System using Servlets

- AddUserServlet
  - HTML control type="date" always sends date in form "yyyy-MM-dd".
  - If using java.util.Date

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
String birthDate = req.getParameter("birth");
Date birth = sdf.parse(birthDate);
```

  - If using java.sql.Date

```
String birthDate = req.getParameter("birth");
Date birth = Date.valueOf(birthDate);
```

- LoginServlet -- same as VotingSystem
- AddReviewServlet -- same as VotingSystem's CandidateEditServlet GET & POST.
- EditReviewServlet -- same as VotingSystem's CandidateEditServlet GET & POST -- Keep "id" control invisible instead of readonly. Hidden controls are not visible in browser, but visible in View Source.

```
out.printf("<input type='hidden' value='%s'><br/>", r.getId());
```

- DeleteReviewServlet -- same as VotingSystem's CandidateDeleteServlet.
- LogoutServlet -- same as VotingSystem's LogoutServlet.
- ReviewsServlet (url-pattern /reviews) -- similar to VotingSystem's ResultServlet.

```
HttpSession session = req.getSession();
User curUser = (User)session.setAttribute("curUser");

out.println("<a href='reviews?type=all'>All Reviews</a> | ");
out.println("<a href='reviews?type=my'>My Reviews</a> | ");
out.println("<a href='reviews?type=shared'>Shared Reviews</a>");

String type = req.getParameter("type");
List<Review> list = new ArrayList<>();
if(type==null || type.equals("all"))
    list = reviewDao.findAll();
else if(type.equals("my"))
    list = reviewDao.findByUserId(curUser.getId());
else if(type.equals("shared"))
    list = reviewDao.getSharedReviewsWithUser(curUser.getId());
```

```
out.println("<table>");
// print table header
out.println("<tbody>");
for(Review r:list) {
    out.println("<tr>");
    out.printf("<td>%d</td>", r.getId());
    //out.printf("<td>%d</td>", r.getMovieId());
    Movie m = movieDao.findById(r.getMovieId());
    out.printf("<td>%s</td>", m.getTitle());
    out.printf("<td>%d</td>", r.getUserId());
    out.printf("<td>%d</td>", r.getRating());
    out.printf("<td>%s</td>", r.getReview());
    out.printf("<td>%s</td>", r.getModified());
    out.println("<td>");
    if(r.getUserId() == curUser.getId())
        out.printf("<a href='editreview?id=%s'>Edit</a><a href='delreview?id=%s'>Edit</a>...", r.getId(), r.getId());
    out.println("<td>");
    out.println("</tr>");
}
out.println("</tbody>");
out.println("</table>");
```

- ShareReviewServlet -- Refer hint from day03.pdf (at the end).

Applet vs Servlets

- Applet is a Java class that is downloaded into client machine (when request is made) and executed in client browser JRE/JVM plugin.
- Applet Life Cycle: init(), start(), paint(), stop(), destroy() called by JRE in browser (a.k.a. Applet container)
- Servlet is a Java class that is executed "in web-server" when request is received from client and produces response that is sent back to the client.
- Servlet Life Cycle: init(), service() and destroy() called by JRE in web-server (a.k.a. Web container).

CGI vs Servlets

- CGI stands for Common Gateway Interface -- Standard of how a program can communicate with web.
- Program can be written in any language (that supports CGI) e.g. C, Fortran. These programs executed in OS (native applications) i.e. platform dependent.
- For each request from the client a new process is created, that execute the request handling program and produces response that is sent back to the client.
- Less efficient (Processes are heavy-weight).
- Servlet is standard of implementing Java program that can handle HTTP requests.
- Servlet is written in Java and runs on Web container (JVM) i.e. platform independent.
- For each request from the client a new thread is created, that execute the service() method and produces response that is sent back to the client.
- More efficient (Threads are light-weight).

JSP

- Servlet = Business logic* + Presentation logic
- JSP = Presentation logic* + Business logic
- **JSP is converted into the servlet while execution.**
- JSP is outdated.

JSP syntax

- Directive <%@ ... %>
  - Instructs JSP engine to process the jsp.
  - @page -- servlet creation/translation.
  - @include -- include a jsp/html into another jsp.
  - @taglib -- to use custom/third party tags in jsp.
- Declaration <%! ... %>
  - To declare fields and methods in generated servlet (other than service()).
- Scriptlet <% ... %>
  - For Java statements to be executed for each request (in jspService()).
- Expression <%= ... %>
  - For Java expressions whose output is to be embedded in produced response. Executes for each request (in jspService()).
- Comment <%-- ... --%>
  - Server side comment -- discarded while processing.

Example Servlet --> JSP

- Generated servlet

```
import java.util.Date;
class HelloServlet ... {
    private int count = 0;
    public void init(ServletConfig conf) ... {
        super.init(conf);
        System.out.println("init() called...");
    }
    public void destroy() {
        System.out.println("destroy() called...");
    }
    // HelloServlet.service()
    public void doGet(HttpServletRequest request, HttpServletResponse response) ... {
        processRequest(request, response);
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response) ... {
        processRequest(request, response);
    }
    public void processRequest(HttpServletRequest request, HttpServletResponse response) ... {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h3>Congratulations, Sunbeam!</h3>");
        count++;
        if(count % 2 == 0) {
            out.println("Even Count: " + count);
        } else {
```

```
            out.println("Odd Count: " + count);
        }
        Date d = new Date();
        out.println("<br/><br/>Current Time: " + d.toString());
        out.println("</body>");
        out.println("</html>");
    }
}
```

- JSP

```
<%@ page language="java" %>
<%@ page contentType="text/html" import="java.util.Date" %>
<%-- This is Hello JSP (Server side comment) --%>
<!-- This is Hello JSP (Client side/HTML comment) -->
<html>
    <head>
        <title>Hello JSP</title>
    </head>
    <body>
        <%!
            private int count = 0;
        %>
        <%!
            public void jspInit() {
                System.out.println("jspInit() called");
            }
            public void jspDestroy() {
                System.out.println("jspDestroy() called.");
            }
        %>
        <h3>Congratulations, Sunbeam!</h3>
        <% count++; %>
```

```
        <% if(count % 2 == 0) { %>
            "Even Count: " <%= count %>
        <% } else { %>
            "Odd Count: " <%= count %>
        <% } %>
        <% Date d = new Date(); %>
        <br/><br/>Current Time: <%= d.toString() %>
    </body>
</html>
```

JSP Life cycle

- JSP Engine
  - 1- Translation stage: Converts JSP into servlet java class. Check JSP syntax errors.
  - 2- Compilation stage: Compiles generated servlet java class into java byte code. Check java code errors (scriptlet, expression and declaration blocks).
- Servlet Engine
  - 3- Loading & Instantiation stage: Loads servlet class into JVM & create its object. Invokes jspInit().
  - 4- Request handling stage: Handles request & produce response. Invokes jspService(). For each request.
  - 5- Destruction stage: De-initialize the object. Invokes jspDestroy().
- For first request all stages 1 to 4 are executed.
- For subsequent requests only stage 4 is executed.

JSP @Page Directive

- `<%@page language="java"%>`
  - Server side processing lanaguage is java. Only java lanaguage is supported.
- `<%@page import="java.util.Date"%>`
  - Imports given package in generated servlet .java file.
- `<%@page contentType="text/html" %>`
  - response.setContentType("text/html");
- `<%@page session="true"%>`
  - Internally calls `session = req.getSession();`.

    - If session="false", then `session = null;`.
- `<%@page isErrorPage="false"%>`
    - This page is used only for displaying errors like 403, 404, 500 with custom error messages.
- `<%@ page errorPage="error.jsp" %>`
    - Errors produced in this page are to be displayed in error.jsp. Here error.jsp is a error page.
- `<%@page info="This is hello JSP"%>`
    - Keeps information/metadata about JSP page.
- `<%@page buffer = "8"%>`
    - JSP response is stored in a buffer. Default buffer size is 8 kb.
- `<%@page autoFlush = "false"%>`
    - Whenever buffer is full, it is flushed to the client.
- `<%@page extends = "javax.servlet.http.HttpServlet"%>`
    - Defines base class generated servlet class.
- `<%@page isELIgnored = "false"%>`
    - Do not process EL (expression language) syntax ${...} in JSP page.

JSP Implicit objects

- These objects are available for use in _jspService() i.e. scriptlets and expressions. We need not to declare them explicitly.
- Because these objects are local variables or arguments of generated _jspService() method.
- request: HttpServletRequest
- response: HttpServletResponse
- session: HttpSession
- out: JspWriter -- similar PrintWriter
- application: ServletContext
- config: ServletConfig
- pageContext: PageContext -- to store page attributes.
- page: Object -- represent current page/servlet instance (this).
- exception: Throwable -- available only in error pages.

JSP Standard Actions

- JSP Standard actions are predefined JSP tags for certain functionality. They can be used to reduce scriptlets in JSP code.
- `<jsp:forward page="subjects.jsp" />`

```
<%
    RequestDispatcher rd = request.getRequestDispatcher("subjects.jsp");
    rd.forward(request, response);
%>
```

- `<jsp:include page="page2.jsp" />`

```
<%
    RequestDispatcher rd = request.getRequestDispatcher("page2.jsp");
    rd.include(request, response);
%>
```

  - Dynamic/runtime inclusion i.e. page1.jsp <===> page2.jsp
  - `<%@include file="page.jsp"%>` is static inclusion i.e. contents of page2.jsp are included in page1.jsp during translation stage.
- `<jsp:param name=... value=... />`
  - Can be used as optional param as child tag of forward or include.

```
<%-- page1.jsp --%>
<jsp:forward page="page2.jsp">
    <jsp:param name="key" value="someValue"/>
</jsp:forward>
```

```
<%-- page2.jsp --%>
<%
```

```
    String value = request.getParameter("key");
%>
```

- `<jsp:plugin type="applet" ... />`
  - Applets are java classes that gets loaded into client browser and executed there in browser's JRE (plugin). Due to severe security concerns they are deprecated.
- `<jsp:fallback .../>`
  - fallback is child tag for plugin tag to show alternate message if plugin loading is failed.

```
<jsp:plugin type="applet" class="com.sunbeam.MyApplet" ...>
    <jsp:fallback>Applet Not Loaded.</jsp:fallback>
</jsp:plugin>
```

- `<jsp:element name = "xmlElement">`
- `<jsp:attribute name = "xmlEleAttr">`
- `<jsp:body>...</jsp:body>`
- `<jsp:text>...</jsp:text>`
  - Above four are XML generation tags.
- `<jsp:useBean ... />`
- `<jsp:setProperty ... />`
- `<jsp:getProperty ... />`

Java Beans

- Java beans are simple java classes which contain constructor, fields, getters/setters and one/more business logic methods.
- Ideal JSPs do not contain scriptlets. So Java beans are used to encapsulate all business logic required for the JSP processing.
- Java beans used in JSP pages using
  - `<jsp:useBean id="var" class="pkg.BeanClass" scope="..."/>`
  - `<jsp:setProperty name="var" property="... " value="..."/>`

    - `<jsp:setProperty name="var" property="... " param="..."/>`
    - `<jsp:setProperty name="var" property="*"/>`
    - `<jsp:getProperty name="var" property="... "/>`
- Java beans objects are created & accessed using reflection. So naming conventions must be strictly followed.

Java bean scopes

- page â€“ PageContext attribute (default) -- lowest scope
    - Internally, bean object is stored in the current page context using pageContext.setAttribute("beanName", beanObject) and accessed using pageContext.getAttribute("beanName").
    - Bean is available for the current page current request only.
- request â€“ Request attribute
    - Internally, bean object is stored in the current request using request.setAttribute("beanName", beanObject) and accessed using request.getAttribute("beanName").
    - If same request is forwarded or included (using RequestDispatcher), then the bean will be accessible in next page as well.
- session â€“ HttpSession attribute
    - Internally, bean object is stored in the current user HttpSession using session.setAttribute("beanName", beanObject) and accessed using session.getAttribute("beanName").
    - The bean is accessible in all requests to all pages by the same client.
- application â€“ ServletContext attribute -- highest scope
    - Internally, bean object is stored in the current application ServletContext using ctx.setAttribute("beanName", beanObject) and accessed using ctx.getAttribute("beanName").
    - The bean is accessible in all requests to all pages by all clients.

jsp:useBean

- Check if object with given name is present in given scope (using getAttribute()). If available, access it.
- If not available, create new bean object.
- Add the object into given scope (using setAttribute()).

```
// Internals of jsp:useBean
beanObj = scope.getAttribute("beanName");
if(beanObj == null) {
    beanObj = new BeanClass();
    scope.setAttribute("beanName", beanObj);
}
```

jsp:setProperty and jsp:getProperty

- These tags internally calls setter and getter methods on the bean object.
- jsp:setProperty, jsp:getProperty must be preceded by jsp:useBean.

Assignments

1. Movie Review System
    - index.jsp --> login.jsp (authentication using LoginBean)
    - register.jsp --> registration.jsp (registration using RegistrationBean)