

Web Component Developer for the J2EE Platform

Study Notes for the Beta Exam

Sources:

Java Servlet Specification and JavaDoc
Java Server Pages Specification and JavaDoc
Professional JSP 2nd Edition

Section 1: The Servlet Model

- 1.1 doGet, doPost, doPut. Params are: (HttpServletRequest req, HttpServletResponse res)
- 1.2 GET, POST, HEAD – what triggers cause a browser to do this??
- GET – user enters url into browser.
<FORM METHOD="GET" ACTION="servlet_name">
 - POST – user fills in html form and clicks submit, form uses POST method to pass form values.
<FORM METHOD="POST" ACTION="servlet_name">
 - HEAD – request only page headers. A way to check to see if a document has been updated since the last request.
<FORM METHOD="HEAD" ACTION="servlet_name">
- 1.3 a) HttpServletRequest request.getParameters() to get list of parameters.
b) ServletConfig.getInitParameter()
c) HttpServletRequest.getHeader(String headerName) : String – other methods: getHeaderNames(), getHeaders : Enumeration.
d) General purpose HttpServletResponse.setHeader(name, value). Also, can use setDateHeader(String, long) and setIntHeader(String, int).
ServletResponse.setContentType(String) “text/html”.
e) PrintWriter out = response.getWriter();
f) OutputStream out = response.getOutputStream()
g) HttpServletResponse.sendRedirect(String url)
- 1.4 Request – ServletRequest.setAttribute(String, Object), deleteAttribute(String), getAttribute(String), getAttributeNames().
Session - HttpSession.setAttribute(String, Object), deleteAttribute(String), getAttribute(String), getAttributeNames().
Context – ServletContext.getAttribute(String), getAttributeNames(), removeAttribute(name), setAttribute(name)

- 1.5 init – call when first created and not called again for each user request. One time setup code goes here. Servlet container controls life-cycle.
service – Server receives request and spawns separate thread for each user request. Invokes service method which, by default, hand's off to doXXX methods doGet, doPost
destroy – executed prior to server unloading the servlet. Good for closing db connections, etc.

- 1.6 RequestDispatcher -

```
RequestDispatcher rd;  
ServletContext sc = getServletContext();  
rd = sc.getRequestDispatcher("/mainmenu.jsp");  
rd.forward(req, res);
```

void	<u>forward</u> (ServletRequest request, ServletResponse response) Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
void	<u>include</u> (ServletRequest request, ServletResponse response) Includes the content of a resource (servlet, JSP page, HTML file) in the response.

Section 2: The Structure and Deployment of Modern Servlet Web Applications

2.1 Web Application Structure

\	Jsp, html, images (optional)
\WEB-INF	Web.xml, tld files, other xml config files.
\WEB-INF \classes*	Compiled classes
\WEB-INF \lib*.jar	Jarred classes

NOTE: Enters in WEB-INF and below are not visible outside the container. Root is visible.

2.2

- Servlet instance – The servlet element contains the declarative data of a servlet
<!ELEMENT servlet (icon?, servlet-name, display-name?, description?, (servlet-class|jsp-file), init-param, load-on-startup?, security-role-ref*)>*
- Servlet name – logical name used within dd *<!ELEMENT servlet-name (#PCDATA)>*
- Servlet class – fully qualified package name of servlet class. *<!ELEMENT servlet-class (#PCDATA)>*
- Initialization parms – init parms for servlet. *<!ELEMENT init-param (param-name, param-value, description?)>*
- URL to named servlet mapping - map logical servlet name to URL that server will listen for. *<!ELEMENT servlet-mapping (servlet-name, url-pattern)>*

Section 3: The Servlet Container Model

Application Events

Event Listeners instantiation and registration is done by the Web Container and must be declared in the web.xml deployment descriptor.

3.1 Interfaces, Uses, and Methods

- a) ServletContext.getInitParameter(), getInitParameterNames(). These are specified in the web.xml.
- b) Servlet context listeners are used to manage resources or state held at a VM level for the application. ServletContextListener.contextDestroyed() – server shut down. ContextInitialized – server up and ready for requests.
- c) When attributes are added to the application environment context. ServletContextAttributeListener attributeAdded(), attributeRemoved(), attributeReplaced()
- d) Fine grained control over contents of a Session. HttpSessionAttributeListener ServletContextAttributeListener attributeAdded(), attributeRemoved(), attributeReplaced()

3.2 Corresponding Deployment Descriptor Elements

- a) Element: <context-param> Subelements: <param-name> <param-value>
- b, c, d) <listener> <listener-class>com.something.classname</listener-class></listener> (Session & Context dd declaration is the same)

3.3 A Web Application marked as Distributable – servlet container to distribute to multiple JVM's for clustering, scalability, and failover.

- The context exists locally in the VM in which they were created and placed. This prevents the ServletContext from being used as a distributed shared memory store, Instead place it in a Session, DB, or EJB.
 - Session is scoped to vm servicing session requests.
 - Context is scoped to the web container's vm.
 - Container is not required to propagate context or session events to other vm's.
- a) each container will have it's own separate copy of this.
 - b) each container will have it's own and events are not propagated to the other.
 - c) same as b
 - d) each vm will have it's own and events are not propagated to the other.

Section 4: Designing and Developing Servlets to Handle Server-side Expectations

4.1 sendError, setStatus

- `HTTPServletResponse.sendError(int), (int, string)`. Sends HTTP Error code to browser. If an error-page declaration has been made for the web application corresponding to the status code passed in, it will be served back in preference to the suggested msg parameter.
- `HTTPServletResponse.setStatus(int)`, set HTTP status code when there is an error, does not invoke web app error page.

4.2 Deployment Descriptor for specifying an error page for the web application for a given HTTP error code or java exception.

```
<error-page>  
    <error-code>404</error-code>  
    <location>/404.html</location>  
</error-page>
```

OR `<exception-type>java.lang.NullPointerException</exception-type>`
Instead of `<error-code>`

Request Dispatcher could be used to manually forward a request to an error page so long as the request attributes (`status_code`, `exception`, `request_uri`, `servlet_name`) are contained in the request.

4.3 Servlet/WebApp Log File ServletContext (interface), GenericServlet methods:

void	<code>log</code> (java.lang.String msg) Writes the specified message to a servlet log file, usually an event log.
void	<code>log</code> (java.lang.String message, java.lang.Throwable throwable) Writes an explanatory message and a stack trace for a given Throwable exception to the servlet log file.

Section 5: Designing and Developing Servlets Using Session Management

5.1

- a) `HttpServletRequest.getSession() : HttpSession`
- b) `HttpSession.setAttribute(name, object)` (`putValue` is deprecated)
- c) `HttpSession.getAttribute(name) : Object`
- d) `HttpSessionAttributeListener attributeAdded(HttpSessionBindingEvent), attributeRemoved(evt), attributeReplaced(evt)`
- e) `HttpSessionListener.sessionCreated(HttpSessionEvent), sessionDestroyed(evt)`
- f) `HttpSession.invalidate()`

5.2 Invalidate Session. A session will be invalidated when the session times out (container/server invalidates) or is invalidated in code by the use of `HttpSession.invalidate()`.

5.3 URL Rewriting

URL rewriting is the lowest common denominator of session tracking. When a client will not accept a cookie, URL rewriting may be used by the server as the basis for session tracking. URL rewriting involves adding data, a session id, to the URL path that is interpreted by the container to associate the request with a session. The session id must be encoded as a path parameter in the URL string. The name of the parameter must be `jsessionid`. Here is an example of a URL containing encoded path information:
`http://www.myserver.com/catalog/index.html;jsessionid=1234`

Section 6: Designing and Developing Secure Web Applications

6.1

- a) authentication – verify user is who they claim to be. See 6.3.
- b) authorization – authenticated user is allowed to access a particular resource.
- c) data integrity – the means to prove that information has not been modified by a third party while in transit.
- d) Auditing – record a permanent log of user activities within the application.
- e) malicious code – Code aimed at breaking the security rules of a web application (buffer overflow mainly).
- f) web site attacks – external attacks by hackers to the web application.

6.2 a) Security Constraint Example

```
<security-constraint>
    <webresource-collection>
        <web-resource-name>Entire application</web-resource-name>
        <url-pattern>/*</url-pattern></web-resource-collection>
    <auth-constraint> <role-name>manager</role-name></auth-constraint>
</security-constraint>
```

Web resource – A web resource collection is a set of URL patterns and HTTP methods that describe a set of resources to be protected. All requests that contain a request path that matches a URL pattern described in the web resource collection is subject to the constraint. The container matches URL patterns defined in security constraints using the same algorithm described in this specification for matching client

- b) requests to servlets and static resources.

```
<web-resource-collection>
    <web-resource-name>SalesInfo</web-resource-name>
    <url-pattern>/salesinfo/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
</web-resource-collection>
```

- c) Determine what kind of user authentication is required

```
<login-config> <auth-method>BASIC|DIGEST|FORM</auth-method>
    <realm-name>HTTP Realm Name </realm-name>
</login-config>
```

d) Determine security roles

```
<security-role-ref>
  <role-name>XYZ</role-name>
  <role-link>customer service</role-link>
</security-role-ref>

...
<security-role>
  <role-name>administrator</role-name>
  <role-name>customerservice</role-name>
</security-role>
```

6.3 Authentication Types

BASIC – HTTP Basic Authentication – Web server/container prompts web client for username/pwd. User sends Base64 encoded string. Browser accomplishes this by popping up a dialog. Browser caches authentication info for subsequent requests.

DIGEST – HTTP Digest Authentication – More secure form of BASIC authentication. Browser sends MD-5 of username/pwd as, url, & http method. Only IE 5.x supports this form of authentication. Not required in spec.

FORM – Form Based Authentication – Developer creates custom login/error screens. Server manages the display of these. Form action: j_security_check, parms: j_username, j_password. Must be configured in DD:

```
<form-login-config>
  <form-login-page>/loginpage.jsp</form-login-page>
  <form-error-page>/errorpage.jsp</form-error-page>
</form-login-config>
```

CLIENT-CERT – HTTPS Client Authentication – Uses SSL. Uses public-key encryption to establish a secure session to transmit data between client and server. Authenticate server and optionally the client. Provide encrypted connection to exchange messages.

Section 7: Designing and Developing Thread-safe Servlets

7.1 Thread Safety Issues

- a) Y
- b) N
- c) N
- d) Y – only within the request thread
- e) N
- f) N

7.2 Single Thread Model guarantees one thread will execute through a servlet's service method at a time. Objects (ie. Session) may be accessible by multiple instances however. Multiple Thread Model usually has a single instance to handle all requests. A container with Single Thread Model may instantiate multiple servlet instances to handle the load.

7.3 Implementing SingleThreadModel interface marks the servlet (to the container) as being capable of only executing one request at a time.

Section 8: The JSP Model

8.1 Valid Opening and Closing Tags for different tag types.

- a) Directive - `<%@ Directive {attribute="value"}* %>`
- b) Declaration - `<%! JavaDeclaration %>`
- c) Scriptlet - `<% JavaStatements %>`
- d) Expression - `<%= JavaExpression %>`

8.2 Purpose of the different tag types

- a) Directive – controls translation and compilation phase.
- b) Declaration – declare a java variable or method. No output produced. Class level declarations.
- c) Scriptlet – valid java statements. No output produced unless `out` is used
- d) Expression – evaluation of expression to a String and then included in the output.

8.3 XML Version of Tag Types

- a) `<jsp:directive.page .../>`
- b) `<jsp:declaration> ... </jsp:declaration>`
- c) `<jsp:scriptlet> ... </jsp:scriptlet>`
- d) `<jsp:expression> ... </jsp:expression>`

8.4 Specified Page Directive examples

- a) `<%@ page import="java.util.Date" %>`
- b) `<%@ session="true"`
- c) `<%@ errorPage="errorPage.jsp" %>`
- d) `<%@ page isErrorPage="true" %>`

8.5 sequence in order as per requirements document.

8.6 Implicit Objects and their purpose

- a) request – the request
- b) response – the response
- c) out – object for writing to the output stream.
- d) session – created as long as `<% page session="false" %>` is *not* used.
- e) config – Specific to a servlet instance. Same as Servlet `getServletConfig()`
- f) application – ServletContext – Available to all Servlets (application wide).
Provides access to resources of the servlet engine (resources, attributes, init context params, request dispatcher, server info, URL & MIME resources).
- g) page – this instance of the page (equivalent servlet instance 'this').
- h) pageContext – PageContext – Environment for this page. Used by Servlet engine to manage such features as error pages and params for included/forwarded pages.
- i) exception – Throwable – created only if `<%@ page isErrorPage="true" %>` this is an error page that has been forwarded to from a previous page in error.

8.7 Examples of conditional and iterative statements.

a) Conditional Statement

```
<% if (event.equals("ENTER_RECORD")) {  
%>  
    <jsp:include page="enterRecord.jsp" flush=true"/>  
%>  
} else if (event.equals ("NEW_RECORD")) {  
%>  
    <jsp:include page="newRecord.jsp" flush="true"/>  
%>  
} else {  
%>  
    <jsp:include page="default.jsp" flush = true"/>  
%> } %>
```

b) Iterative Statement

```
<% Hashtable h = (Hashtable) session.getAttribute("charges");  
if (h != null)  
{  
%>  
    <ul>  
%>  
        Enumeration charges = h.keys();  
        While (charges.hasMoreElements())  
        {  
            String proj = (String) charges.nextElement();  
            Charge ch = (Charge) h.get(proj);  
%>  
            <li>  
                name = <%= ch.getName() %>  
                , project = <%= proj %>  
                , hours = <%= ch.getHours() %>  
                , date = <%= ch.getDate() %>  
%>  
            }  
%>  
        }  
%>  
    </ul>  
%>  
}
```

Section 9: Designing and Developing Reusable Web Components

9.1 Given a description of required functionality, identify the JSP directive or standard tag in the correct format with the correct attributes required to specify the inclusion of a Web component into the JSP page

- Include Directive - Translation-time

`<%@ include file=... %>`

Content is parsed by JSP container.

- Include Action - Request-time

`<jsp:include page=... />`

Content is not parsed; it is included in place.

Custom Tags

How about scriptlets, requestdispatcher includes?

Section 10: Designing and Developing JSPs Using JavaBeans

10.1

- a) `<jsp:useBean id="connection", class="com.myco.myapp.Connection", scope="page" />`
- b) `<jsp:getProperty name="beanName" property="propName" />`
Also, this usebean tag includes one time initialization in the body
`<jsp:useBean id="beanName" scope="request" class="Popovits.name">`
`<jsp:setProperty name="id" property="givenName" value="Michele" />`
`</jsp:useBean>`
- c) see a)
- d) see a) & b)
- e) `<% String timeout = connection.getTimeout; %>`
- f) `<jsp:getProperty name="connection", property="timeout"/>`
- g) `<jsp:setProperty name="connection", property="timeout", value="33"/>`

10.2 Servlet Equivalent

request - `HttpServletRequest`

session – `HttpServletRequest.getSession() : HttpSession`

application – `GenericServlet.getServletContext() : ServletContext` or
`GenericServlet.getServletConfig().getServletContext()`

10.3 Techniques for accessing a declared java bean

A declared JavaBean can also be accessed using:

- Scriptlets
- Expressions
- Custom Tags

Section 11: Designing and Developing JSPs Using Custom Tags

11.1 web.xml tag library declaration

```
<web-app>
  <taglib>
    <taglib-uri>
      http://www.jspinsider.com/jspkit/javascript
    </taglib-uri>
    <taglib-location>
      /WEB-INF/JavaScriptExampleTag.tld
    </taglib-location>
  </taglib>
</web-app>
```

- 11.2 `<%@taglib uri=http://jakarta.apache.org/taglibs/datetime-1.0 prefix="dt"%>`
- uri is defined in web.xml, prefix is used in the jsp to reference the tag.

11.3

- a) `<msgBean:message/>`
- b) `<msgBean:message attrName="value" />`
- c) `<msgBean:message> <h1>This is the title</h1> </msgBean>`
- d) `<msgBean:message> <helloBean:sayHello/> </msgBean>`

Section 12: Designing and Developing A Custom Tag Library

12.1 TLD Elements

- a) `<name>`
- b) `<tag-class>`
- c) `<body-content>`
- d) `<attribute> <name>username</name> <rtexprvalue>true</rtexprvalue>
<required>false</required></attribute>`

12.2 TLD Attribute Elements

- a) `<name>`
- b) `<required>`
- c) `<rtexprvalue>` run time expression value

12.3 Valid bodycontact TLD values

- a) empty
- b) JSP
- c) tagdependent

12.4

- doStart – Process the start tag for this instance
- doAfterBody – Process body (re)evaluation – repeat for iteration tag.
- doEndTag – Process the end tag for this instance. Used to clean up resources and add any closing tags to the output as necessary

12.5 Valid Return Values

- a) doStartTag - Tag.EVAL_BODY_INCLUDE, BodyTag.EVAL_BODY_BUFFERED, SKIP_BODY
- b) doAfterBody - EVAL_BODY_AGAIN, SKIP_BODY
- c) doEndTag - EVAL_PAGE, SKIP_PAGE
- d) PageContext.getOut - javax.servlet.jsp.JspWriter

	doStart	doAfterBody	doEndTag
EVAL_BODY_INCLUDE	All		
EVAL_BODY_BUFFERED	Body tag only		
EVAL_BODY_AGAIN		Iteration	
SKIP_BODY	All	All	
EVAL_BODY_TAG	<i>deprecated</i>		
EVAL_PAGE			All
SKIP_PAGE			All

12.6

EVAL_BODY_AGAIN

Request the reevaluation of some body. Returned from doAfterBody. For compatibility with JSP 1.1, the value is carefully selected to be the same as the, now deprecated, BodyTag.EVAL_BODY_TAG,

EVAL_BODY_BUFFERED

Request the creation of new buffer, a BodyContent on which to evaluate the body of this tag. Returned from doStartTag when it implements BodyTag. This is an illegal return value for doStartTag when the class does not implement BodyTag.

EVAL_BODY_INCLUDE

Evaluate body into existing out stream. Valid return value for doStartTag.

SKIP_BODY

Skip body evaluation. Valid return value for doStartTag and doAfterBody.

EVAL_PAGE

Continue evaluating the page. Valid return value for doEndTag().

SKIP_PAGE

Skip the rest of the page. Valid return value for doEndTag.

12.7

- a) pageContext.getSession();
- b) pageContext.getRequest.getAttribute("attrName");

12.8

Tag Tag.getParent()

Tag TagSupport.getParent()

Tag TagSupport.findAncestorWithClass(Tag from, java.lang.Class class)

Section 13: Design Patterns

13.2 Benefits of these Patterns

Pattern	Benefit
Value Objects	<ul style="list-style-type: none">▪ Reduce network traffic by acting as a caching proxy of a remote object.▪ Total load on the remote object's remote interface is decreased, because data represented by the Value Object are cached on the client by the Value Object instance. All accesses to the properties of the Value Objects are local, and so those accesses cause no remote method invocations.▪ Decreased latency can improve user response time.
MVC	<ul style="list-style-type: none">▪ Clarifies design by separating data modeling issues from data display and user interaction.▪ Allows the same data to be viewed in multiple ways and by multiple users.▪ Improves extensibility by simplifying impact analysis.▪ Improves maintainability by encapsulating application functions behind well-known APIs, and decreasing code replication ("copy-paste-and-hack").▪ Enhances reusability by decoupling application functionality from presentation.▪ Makes applications easier to distribute, since MVC boundaries are natural distribution interface points.▪ Can be used to partition deployment and enable incremental updates.▪ Facilitates testability by forcing clear designation of responsibilities and functional consistency.▪ Enhances flexibility, because data model, user interaction, and data display can be made "pluggable".
Data Access Object	<ul style="list-style-type: none">▪ greater deployment flexibility▪ resource vendor independence▪ resource implementation independence▪ easier migration to CMP▪ enhanced extensibility
Business Delegate	<ul style="list-style-type: none">▪ reduced coupling, improved manageability.▪ Service exception translation▪ Simpler, uniform interface▪ Performance impact