JSP

What is JSP? (Java server pages)
Dynamic Web page (having typically HTML 5 markup) , can embed Java code directly.
Dynamic web component , whose life-cycle is managed by WC(JSP container/Servlet container/Servlet engine)

WHY JSP?
1. JSP allows developer to separate presentation logic(dyn resp generation) from Business logic or data manipulation logic.
Typically JSPs -- used for P.L(presentation logic)
Java Beans or Custom Tags(actions) --- will contain Business logic.
2. Ease of development --- JSP pages are auto. translated by W.C in to servlet & compiled & deployed.
3. Can use web design tools -- for faster development (RAD --rapid application development) tools.

JSP API : is a part of Java EE specs
jsp-api.jar --- <tomcat>/lib : specs

Contains JSP API implementation classses. : jasper.jar

0. javax.servlet.Servlet -- super i/f
1. javax.servlet.jsp.JspPage -- extends Servlet i/f
1.1 public void jspInit()
1.2 public void jspDestroy()

Can be overridden by JSP page author

2. Further extended by javax.servlet.jsp.HttpJspPage
2.1 public void _jspService(HttpServletRequest rq,HttpServletResponse rs) throws ServletExc,IOExc.

Never override _jspService ---JSP container auto translates JSP tags (body) into _jspService.

JSP life-cycle

1. Clnt sends the 1st request to the JSP (test.jsp)
2. Web-container invokes the life cycle for JSP
3. Translation Phase : handled by the JSP container.
   I/p : test.jsp O/p : test_jsp.java (name : specific to the Tomcat container)
   Meaning : .jsp is translated into corresponding servlet page(.java)
   Translation time errs : syntactical errs in using JSP syntax.
   In case of errs : life-cycle is aborted.
4. Compilation Phase : handled by the JSP container.
   I/p : Translated servlet page(.java) O/p : Page Translation class(.class)
   Meaning : servlet page auto. compiled into .class file
   Compilation time errs: syntacticle errs in generated Java syntax.
5. Request processing phase / Run time phase. : typically handled by the Servlet Container.
6. S.C : will try to locate,load,instantiate the generated servlet class.

7. The 1st it calls : public void jspInit() : one time inits can be performed.(jspInit availble from javax.servlet.jsp.JspPage)
8. Then it will call follwing method using thrd created per clnt request :
   public void _jspService(HttpServlet Rq,HttpServletResponse) throws ServletException,IOException(API avlble from javax.servlet.jsp.HttpJspPage)
   When _jspService rets , thread's run method is over & thrd rets to the pool, where it can be used for servicing some other or same clnt's req.
9.. At the end ...(server shutting down or re-deployment of the context) : the S.C calls

public void jspDestroy()
After this : translated servlet page class inst. will be GCEd....

10 For 2nd req onwards ..... : SC will invoke step 8 onwards.

JSP 2.0/2.1/2.2/2.3 syntax
---
1. JSP comments

1.1 server side comment
syntax : <%-- comment text --%>
significance : JSP translator & compiler ignores the commented text.

1.2 clnt side comment
syntax : <!-- comment text -->
significance : JSP translator & compiler does not ignore the commented text BUT clnt browser will ignore it.

2. JSP's implicit objects (available only to _jspService) -- avlable to scriptlets,exprs
2.1 out - javax.servlet.jsp.JspWriter : represents the buffered writer stream connected to the clnt via HttpServletResponse(similar to your PrintWriter in servlets)
Has the same API as PW(except printf)
usage eg : out.print("some text sent to clnt");

2.2 request : HttpServletRequest (same API)

2.3 response : HttpServletResponse

2.4 config : ServletConfig (used for passing init params)

2.4 session : HttpSession (By def. all JSPs participate in session tracking i.e session obj is created)

2.5 exception : java.lang.Throwable (available only to err handling pages)

2.6 pageContext : current page environment : javax.servlet.jsp.PageContext(this class stores references to page specific objects viz -- exception,out,config,session)

2.7 application : ServletContext(used for Request dispatching, server side logging, for creating context listeners,to avail context params, to add/get context scoped attrs)
2.8 page --- current translated page class instance created for 'this' JSP

---

3. Scripting elements : To include the java content within JSP : to make it dynamic.

3.1 Scriptlets : can add the java code directly . AVOID scriptlets . (Use only till you learn Javabeans & custom tags or JSTL,). we will use use the scriptlets to add : Req. processing logic, B.L & P.L)
syntax : <% java code..... %> : within <body> tag.
location inside the translated page : within _jspService
usage : till Java beans / JSTL or cust. tags are introduced : scriptlets used for control flow/B.L/req. proc. logic

3.2 JSP expressions :
syntax : <%= expr to evaluate %>
--Evaluates an expression --converts it to string --send it to clnt browser.
eg : <%= new Date() %>

expr to evaluate : java method invocation which rets a value OR
const expr or attributes(getAttribute) or variables(instance vars or method local)
location inside the translated page : within _jspService
significance : the expr gets evaluated---> to string -> automatically sent to clnt browser.

eg <%= new Date() %>
eg <%= request.getAttribute("user_dtls") %>
<%= 12*34*456 %>
<%= session.getAttribute("user_dtls") %>
<%= session.setAttribute("nm",1234) %> -- compiler error
<%= session.getId() %>

Better alternative to JSP Expressions : EL syntax (Expression Language : avlble from JSP 1.2 onwards)
syntax : ${expr to evaluate} (to be added directly in ,<body> tag)
EL syntax will evaluate the expr ---to String --sends it clnt browser.

JSP implicit object --- request,response,session....---accessible from scriptlets & JSP exprs. ---

EL implicit objects --- can be accessible only via EL syntax
param =Name of the map , created by WC : containing request parameters
pageScope=Name of the map , created by WC : containing page scoped attrs
requestScope =map of request scoped attrs
sessionScope=map of session scoped attrs

applicationScope=map of application(=context) scoped attrs
pageContext --- instance of PageContext's sub class
cookie -- map of cookies(cookie objects)
initParam -- map of context params.

---avlable ONLY to EL syntax ${...}
---to be added directly within <body> ...</body>

eg : ${param.user_nm} ---param.get("user_nm") --value --to string ---> clnt
request.getParameter("user_nm") --value --to string ---> clnt

${requestScope.abc} ---request.getAttribute("abc") ---to string --sent to clnt
browser.

eg : suppose ctx scoped attr --- loan_scheme
${applicationScope.loan_scheme} ---
getServletContext().getAttribute("loan_scheme") ---to string --sent to clnt

${abc} ---
pageContext.getAttribute("abc") ---not null -- to string -clnt
 null
--request.getAttribute("abc") -- not null -- to string -clnt
null
session.getAttribute("abc") ---
null
getServletContext().getAttirbute("abc") --not null -- to string -clnt
null ---BLANK to clnt browser.

eg : ${sessionScope.nm} OR ${nm}

${pageContext.session.id}
--pageContext.getSession().getId() --- val of JessionId cookie w/o java code.

${pageContext.request.contextPath} ---/day5_web

${pageContext.session.maxInactiveInterval}

${param}
{user_nm=asdf, user_pass=123456}

eg : ${param.f1} ---> request.getParameter("f1").toString()---> sent to browser

param ----map of req parameters.

param : req. param map

${requestScope.abc} ----- out.print(request.getAttribute("abc").toString())

${abc} -----pageCotext.getAttribute("abc")----null ---request
---session---application ---null ---EL prints blank.

3.3 JSP declarations (private members of the translated servlet class)
syntax : <%! JSP declaration block %> (outside <body>)

Usage : 1. for creating page scoped java variables & methods (instance vars & methods/static members)
        2. Also can be used for overriding life cycle methods (jspInit,jspDestroy)
           location inside the translated page : outside of _jspService (directly within JSP's translated class
---
JSP Directives --- commands/messages for JSP Engine(=JSP container=WC) -- to be used @Translation time.

Syntax ---
<%@ Directive name attrList %>
1. page directive
--- all commands applicable to current page only.
Syntax
<%@ page import="comma separated list of pkgs" contentType="text/html" %>
eg -- <%@ page import="java.util.*,java.text.SimpleDateFormat" contentType="text/html" %>
Imp page directive attributes
1. import --- comma separated list of pkgs
2. session --- boolean attribute. default=true.
To disable session tracking, spectify session="false"

3. errorPage="URI of err handling page" ---
tells WC to forward user to err handler page.
4. isErrorPage="true|false" def = false
If you enable this to true--- one can access 'exception' implicit object from this page.

This exception obj is stored under current page ---i.e under pageContext (type=javax.servlet.jsp.PageContext -- class which represents curnt JSP)
EL expresssion to display error mesg
${pageContext.exception.message}
-- evals to pageContext.getException().getMessage()

Additional EL syntax

EL syntax to be used in error handling pages

ERR causing URI : ${pageContext.errorData.requestURI }
 ERR code : ${pageContext.errorData.statusCode}
 ERR Mesg : ${pageContext.exception.message}
 Throwable : ${pageContext.errorData.throwable}
 Throwable Root cause: ${pageContext.errorData.throwable.cause}

5. isThreadSafe="true|false" default=true. "true" is recommended
true=>informing WC--- JSP is already written in thrd -safe manner ---- DON'T apply

thrd safety.

false=>informing WC --- apply thrd safety.

(NOT recommended) ---WC typically marks entire service(servlet scenario) or _jspService in JSP scenarion --- synchronized. --- this removes concurrent handling of multiple client request --so not recommended.
What is recommended? --- isThreadSafe=true(def.) --- identify critical section(i.e code prone to race condition among threads)--guard it in synchronized block.
eg ---Context scoped attrs are inherently thrd -un safe. So access them always from within synched block.

Equivalent step in Servlet
Servlet class can imple. tag i/f -- javax.servlet.SingleThreadModel(DEPRECATED) -- WC ensures only 1thread (representing clnt request) can invoke service method.
--NOT recommended.
---
2. include directive
<%@ include file="URI of the page to be included" %>
Via include directive ---- contents are included @ Translation time.--- indicates page scope(continuation of the same page).
Typically used -- for including static content (can be used to include dyn conts)
eg ---one.jsp
....<%@ include file="two.jsp" %>
two.jsp.....

JSP actions ---- commands/messages meant for WC
to be interpreted @ translation time & applied @ req. processing time.(run time)

Syntax ---standard actions --specifications are present in
jsp-api.jar.(implementations in jasper jar)

<jsp:actionName attribute list>Body of the tag/action
</jsp:actionName>
OR
<jsp:actionName attr list />

JSP Using Java beans(JB)
Why Java Beans
1. allows prog to seperate B.L in Javabeans(Req processing logic, Page navigation & resp generation will be still part of JSP)

Javabeans can store conversational state of clnt(Javabeans 's properties will reflect clnt state) + supplies Business logic methods.

2. simple sharing of JBS across multiple web pages---gives rise to re-usability.

3. Automatic translation between req. params & JB props(string--->primitive data types automatically done by WC)

What is JB?
1. pkged public Java class
It's actually an attribute automatically created by WC.(trigger : jsp:useBean)
& WC will automatically store it under the specified scope
2. Must have def constr.(MUST in JSP using JB scenario)
3. Properties of JBs --- private, non-static , non-transient Data members ---
equivalent to request params sent by clnt.(Prop names MUST match with req params
for easy usage)
In proper words --- Java bean properties reflect the conversational state of the
clnt.
4. per property -- if RW
naming conventions of JB
supply getter & setter.
Rules for setter (Java Bean Naming convention) : strict
public void setPropertyName(Type val)
Type -- prop type.
eg -- private double regAmount;
public void setRegAmount(double val)
{...}
Rules for getter
public Type getPropertyName()
Type -- prop type.
eg -- public double getRegAmount(){...}

5. Business Logic --- methods
public methods --- no other restrictions

Using Java Beans from JSP Via standard actions

1. <jsp:useBean id="BeanRef name" class="F.Q. Bean class name"
scope="page|request|session|application/>

default = page scope.

pre-requisite --- JB class exists under <WEB-INF>/classes.
 JB = server side obj (attribute), attr name --- bean id,attr val -- bean inst.,can
be added to any scope using scope atribute.

eg :
eg --- beans.Userbean
props --- email,pass
setters/getters
B.L mehod -- for validation

Usage ---
<jsp:useBean id="user" class="beans.UserBean" scope="session"/>

W.C invokes JB life-cycle
1. WC chks if specified Bean inst alrdy exists in specified scope
java api --- request.getAttribute("user")

---null=>JB doesn't exist
---loc/load/inst JB class
UserBean u1=new UserBean();
--add JB inst to the specified scope
java api -- request.setAttribute("user",u1);
--- not-null -- WC continues....
---
2. JSP using JB action
2.1 <jsp:setProperty name="Bean ref Name" property="propName"
value="propVal---static/dyn" />
Usage--
<jsp:setProperty name="user" property="email"
value="a@b"/>
WC invokes --- session.getAttribute("user").setEmail("a@b");

<jsp:setProperty name="user" property="email"
value="<%= request.getParameter("f1") %>"/>

OR via EL
<jsp:setProperty name="user" property="email"
value="${param.f1}"/>

WC invokes ---
session.getAttribute("user").setEmail(request.getParameter("f1"));

2.2
<jsp:setProperty name="Bean ref Name" property="propName" param="rq. param name"/>

Usage eg --
<jsp:setProperty name="user" property="email" param="f1"/>

WC invokes ---
((Userbean)request.getAttribute("user")).setEmail(request.getParameter("f1"));

2.3
<jsp:setProperty name="Bean ref Name" property="*"/>

usage

<jsp:setProperty name="user" property="*"/>

eg -- If rq. param names are email & password(i.e matching with JB prop names) then
---matching setters(2) will get called
---
3. <jsp:getProperty name="Bean ref name" property="propName"/>
Usage --
<jsp:getProperty name="user" property="email"/>
WC ---
session.getAttribute("user").getEmail()--- toString --- sent to clnt browser.

Better equivalent -- EL syntax
${sessionScope.user.email} ---
session.getAttribute("user").getEmail()--- toString --- sent to clnt browser.

${requestScope.user.validUser.email}
request.getAttribute("user").getValidUser().getEmail()

${pageContext.exception.message}

4.JSP standard actions related to Request Dispatcher
RD's forward scenario
<jsp:forward page="dispatcher URI" />
eg : In one.jsp
<jsp:forward page="two.jsp"/>
WC invokes ---RequestDispatcher rd=reuqest.getRequestDispatcher("two.jsp");
rd.forward(request,response);

RD's include scenario
<jsp:include page="dispatcher URI" />

eg : In one.jsp
<jsp:include page="two.jsp"/>
WC invokes ---RD rd=reuqest.getRD("two.jsp");
rd.include(request,response);

Why JSTL ? JSP standard tag library
When JSP standard actions are in-sufficient to solve requirements ,
w/o writing scriptlets --- use additional standard actions --- supplied as JSTL
actions

JSP standard Tag Library
has become standard part of J2EE specs from version 1.5 onwards.
It's support exists in form of a JAR

1. jstl-1.2.jar
For using JSTL steps
1.Copy above JAR into your run-time classpath(copy jars either in <tomcat_home>/lib
OR <web-inf>/lib
2. Use taglib directive to import JSTL tag library into JSP pages.
tag=action
tag library=collection of tags
supplier = JSTL vendor(specification vendor=Sun, JAR vendor=Sun/any J2EE compliant
web/app server)
jstl.jar --- consists of Tag implementation classes
Tag libr-   TLD -- Tag library descriptor -- desc of tags -- how to use tags
<%@ taglib uri="URI of JSTL tag lib" prefix="tag prefix" %>

eg --- To import JSTL core lib
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
---

3. Invoke JSTL tag/action
3.1 eg
<c:set var="abc" value="${param.f1}" />
WC :
pageContext.setAttribute("abc",request.getParameter("f1"))

WC invokes --- session.setAttribute("abc",request.getparameter("f1"));

menaing of <c:set> sets the specified attr to specified scope.

<c:set var="details" value="${sessionScope.abc}" />
WC
pageContext.setAttribute("details",session.getAttribute("abc"));

2. <c:remove var="abc" scope="request"/>
WC ---request.removeAttribute("abc") ---removes the attr from req scope.

3.2 JB --- ShopBean -- property --
private AL<Category> categories; --g & s

<c:forEach var="cat" items="${sessionScope.shop.listCategories()}">
${cat}
</c:forEach>

WC invokes ---

for(Category cat : session.getAttribute("shop").listCategories())
  out.print(cat);

eg :
<c:forEach var="acct" items="${sessionScope.my_bank.acctSummary}">
${acct.acctID} ${acct.type} ${acct.balance}
/c:forEach>

http://localhost:8080/day6_web/close_acct.jsp?acId=101

<input type="submit" name="btn" value="Withdraw"
                                        formaction="transactions.jsp" /></td>
                                <td><input type="submit" name="btn" value="Deposit"
                                        formaction="transactions.jsp" /></td>

<%
   request.getPrameter("btn").equals("Deposit") ---
%>
<c:if test="boolean val">
....
</c:if>
<c:if test="${param.btn eq 'Deposit'}">
  in deposit
</c:if>

```
<c:if test="${param.btn eq 'Withdraw'}">
  in withdraw
</c:if>

http://localhost:8080/day6_web/transactions.jsp?acId=102&amount=500&btn=Deposit

<c:redirect url="${sessionScope.my_bank.closeAccount()}"/>
WC --- response.sendRedirect(session.getAttribute("my_bank").closeAccount());
--------------------------------------------------------------------------------
3.3 JSTL action --- for URL rewriting
<c:url var="attr Name" value="URL to be encoded"
scope="page|request|session|application"/>

eg : <c:url var="abc" value="next.jsp" />
WC invokes --- pageContext.setAttribute("abc",resp.encodeURL("next.jsp"));

<a href="${abc}">Next</a>

How to set session tm out ?
1. programmatically --- using Java API
From HttpSession --- setMaxInactiveInterval(int secs)
2. declaratively -- either using Java annotations OR using XML config files
(web.xml)

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

Session Tracking technique :
 HttpSession + URL rewriting

Why ????
To develop a web app , independent of cookies , for session tracking.

For tracking the clnt (clnt's session) : the only information,  WC needs from the
clnt browser is JSessionID value. If clnt browser is not sending it using cookie :
Servlet/JSP prog can embed the JSessionID info in each outgoing URL .(response:
location / href /form action)

What is URL Rewriting : Encoding the URL to contain the JSessionID info.

W.C always 1st chks if JsessionID is coming from cookie, if not ---> then it will
chk in URL : if it finds JsessionID from the encoded URL : extracts its value &
proceeds in the same manner as earlier.

How to ?

API :
For URLs generated by clicking link/buttons(clnt pull I) use
HttpServletResponse method
public String encodeURL(String origURL)
```

Rets : origURL;JSESSIONID=12345

For URLs generated by sendRedirect : clnt pull II : use
HttpServletResponse method
public String encodeRedirectURL(String redirectURL)
Rets : redirectURL;JSESSIONID=12345

Expression Language implicit variables(case sensitive)

1. pageContext : PageContext object (javax.servlet.jsp.PageContext) associate with current page.
2. pageScope - a Map that contains page-scoped attribute names and their values.
3. requestScope - a Map that contains request-scoped attribute names and their values.
4. sessionScope - a Map that contains session-scoped attribute names and their values.
5. applicationScope - a Map that contains application-scoped attribute names and their values.
6. param - a Map that contains rq. parameter names to a single String parameter value (obtained by calling ServletRequest.getParameter(String name)).
7. paramValues - a Map that contains rq. param name to a String[] of all values for that parameter (similar to calling ServletRequest.getParameterValues(name)
8. initParam - a Map that contains context initialization parameter names and their String value (obtained by calling ServletContext.getInitParameter(String name)).
        eg : ${initParam.db_drvr}
9. cookie : Map.Entry of cookies. (entrySet of cookies)
eg : ${cookie.cookiename.value}

key ---cookie name
value ---javax.servlet.http.Cookie

${cookie.JSESSIONID.value}
---cookie.get("JSESSIOIND").getValue()

10. To retrieve err details from Error handling page.
 ERR causing URI : ${pageContext.errorData.requestURI }
 ERR code : ${pageContext.errorData.statusCode}
 ERR Mesg : ${pageContext.exception.message }
 Throwable : ${pageContext.errorData.throwable}
 Throwable Root cause: ${pageContext.errorData.throwable.cause}
eg :
<c:set var="abc" scope="session" value="Hello User...."/>
${sessionScope.abc}