JSP Note

1. JSP life cycle

Với request đầu tiên sau khi được deploy, JSP phải trải qua 2 step là translation và compilation
- Translation: translate jsp sang servlet. JSP syntax được kiểm tra tại step này
- Compilation: container compile servlet sang bytecode (.class file). Java language/syntax được kiểm tra tại step này.

2. Generated Servlet API

JSP translate thành Servlet bao gồm 3 methods sau:
jspInit(): được gọi bởi init() method (từ HttpServlet), method này có thể được override
jspDestroy(): được gọi bởi destroy() method, method này cũng có thể được override
_jspService(): được gọi bởi service() method, method này không nên override

Chú ý: Những method bắt đầu bằng underscore (_) có nghĩa không nên override

3. Standard action

a. Bean declaration
Standard action tag cho bean là <jsp:useBean>
Có các attribute:
- id attribute:
chỉ ra tên của attribute, được set trong servlet với setAttribute() method
<jsp:useBean id="titi" class="" scope=".."> ⇔ page.setAttribute("titi",obj)

-scope attribute
Chỉ ra scope mà attribute được lưu. Có các scope sau:
page (default scope), request, session, application

<jsp:useBean id="titi" scope="session" > ⇔ session.setAttribute("titi",obj)

-class attribute
-type attribute
<jsp:useBean id="", class="", scope="">
chỉ tạo instant của bean nếu như không có object nào của class có tên như id được lưu trong scope
<jsp:useBean> có thể có body và body chỉ được thực thi khi và chỉ khi bean được tạo mới.
b.Bean property tags

setProperty tag có 4 attributes:
- name: bean name
- class: bean type
- property: name of the bean property
- value: the value to give to the bean property
- param: name of the request parameter
param và value là loại trừ lẫn nhau. Sử dụng param attribute khi muốn set property giá trị từ request parameter
<jsp:setProperty name="titi" property="userName" param="fName">:

Lấy giá trị từ fName request parameter và gán cho userName property của titi bean

<jsp:setProperty name="titi" property="userName" >

(không có param attribute) sử dụng khi request parameter và bean property có cùng tên. Trong trường hợp này là set value cho userName property từ userName request parameter.
<jsp:setProperty name="titi" property="*" > dùng khi set tất cả các property trong bean (kể cả các property được kế thừa) với giá trị của tất các request parameter nếu chúng có cùng tên.
Chú ý: String value của request parameter sẽ tự động convert sang primitive data type của bean propery, nhưng container sẽ không tự động convert trong trường hợp sử dụng scripting như sau:
<jsp:setProperty name="n" property="age" value="<%= request.getParameter("age") %>" />

4.Expression Language (EL)

- EL syntax: luôn có dạng như sau ${firstArg.secondArg}
o firstArg là một trong 2 loại sau:
• EL implicit object
• Attribute của 1 trong 4 scope
o secondArg là một trong 2 loại sau:
• map key

• bean property
- EL operators:
EL cung cấp 2 operator:
o dot (.) operator: có dạng như sau leftVariable. rightVariable
• leftVariable là một trong 2 loại sau:
• a Map
• a bean
• rightVariable là một trong 2 loại sau:
• a map key
• a bean property
Tên của rightVariable phải theo quy tắc của Java naming convention
o [] operator: có dạng sau leftVariable [“content”]
• leftVariable là một trong các loại sau:
• a Map
• a bean
• a List
• an array
• content là một trong các loại sau
• a map key
• a bean property
• an index into a List
• an index into an array
Với index của List/array, chúng ta có thể viết thành 2 cách:
myList[3] hoặc myList[“3”] (string index tự động được convert sang int)
Nếu content là String và không có dấu “” thì EL cho rằng đó là attribute và sẽ được thay thế bằng giá trị của nó hoặc null nếu không tìm thấy. Vd: toto[titi]: titi được coi là attribute và sẽ được evaluted
- EL Implicit objects
EL có vài implicit objects sau:
o Các map lưu attribute trong các scope
• pageScope
• requestScope
• sessionScope
• applicationScope
o Các map của request parameter
• param
• paramValues: một param name có nhiều value
o Các map của request header
• header
• headerValues: một header name có nhiều value
o Các map của cookie value
• cookie
o Các map của context init parameter
• initParam: chỉ cho context init parameter
o Tham chiếu đến pageContext
• pageContext: factory obj dùng để tham chiếu đến các obj khác, vd: ${pageContext.request.method} lấy http method của request

EL implicit obj khác với JSP implicit obj: requestScope không phải là request, tương tự cho sessionScope/session, applicationScope/application
- Scope objects và [] operator
Scope objects và [] operator có tác dụng khi attribute name không theo đúng java naming convention. Vd:
session.setAttribute(“vn.ac.aptech.batchName”,”C0808L”);
thì ta không thể sử dụng EL như sau: ${vn.ac.aptech.batchName}
mà phải dùng ${sessionScope[“vn.ac.aptech.batchName”]}

5.JSTL

- **c:set**

Syntax 1: Set the value of a scoped variable using attribute value
<c:set value=”value” var=”varName” [scope=”{page|request|session|application}”]/>
Syntax 2: Set the value of a scoped variable using body content
<c:set var=”varName” [scope=”{page|request|session|application}”]> body content </c:set>
Syntax 3: Set a property of a target object using attribute value
<c:set value=”value” target=”target” property=”propertyName”/>
Syntax 4: Set a property of a target object using body content
<c:set target=”target” property=”propertyName”> body content </c:set>
o var version: (Attribute)

• var attribute là tên của attribute của 1 trong 4 scope
• nếu attribute không tồn tại và value attribute khác null thì attribute sẽ được tạo mới
• nếu attribute tồn tại và value attribute là null thì attribute sẽ bị remove
o target version: (Bean or Map)
• target attribute value không thể null, nó phải tham chiếu đến object thật sự (bằng EL)
- c:url
Tự động url rewriting (session tracking) và url encoding khi sử dụng với c:param

6.Custom tag

-Tag files
Allows to create custom tags for content inclusion.
Tagfiles are JSP fragments renamed with the .tag extension.
Made accessible in a JSP with the taglib directive, using tagdir attribute.
<%@ taglib prefix="..." tagdir="..." %>
tagdir attribute : directory containin the Tag files.
Usage :
<prefix:TagFileName/>

TagFileName is the tag file name minus the .tag extension.
Tag attributes
Tag may be sent attributes using two ways :
+ tag attributes;
Attributes are declared in the tag file itself, using the attribute directive.
<%@ attribute name="attrName" required="true" rtexprvalue="true" %>
Note : if required is set to true and attribute is not present, a runtime error occurs.
In Jsp :
<prefix:tagName attrName="blah" />

+ tag body contents.
Big values may be given to tag files using their body :

<prefix:tagName>

This is the tag value passed in its body

</prefix:tagName>

In the tag file, body value is displayed using the doBody standard action :
 <jsp:doBody/>
The kind of value a tag's body content may receive can be declared with the tag directive (like the page directive, but for tag files). tag directive offers same attributes as page directive, plus one : body-content.
<%@ tag body-content="scriptless|empty|tagdependant" %>

Default value is scriptless.

Note : A tag body CANNOT use scripting elements.
Storage
Tag files may be deployed in :

+ WEB-INF/tags/ directory (or one of its subdirectories);
+ META-INF/tags (or one of its subdirectories) in a JAR file in the WEB-INF/lib/ directory.
Note : If a tag file is deployed in a JAR file, it MUST be declared in a tld file.
Tag file TLD entry
In the tag lib descriptor file, tag files may be described like this :
<tag-file> <name>tagFileName</name> <path>/META-INF/tags/ tagFileName.tag</path> </tag-file>
- Tag Handlers
Tag files implement their functionality in another page (JSP).
Tag handlers implement their functionality in a special java class.
Simple tags
Extends SimpleTagSupport, implements SimpleTag
Lifecycle
Container :
+ Loads the simple tag handler class;

+ Instantiate class (no-args cstr);
+ Calls setJspContext(JspContext);
+ If tag is nested, calls the setParent(JspTag) method;
+ If attributes, calls the attributes setter methods;
+ If tag declared to have a body and is NOT empty : calls setJspBody(JspFragment) method
+ Calls doTag() method.
Note : Simple tags handlers are NEVER reused; each call has a NEW instance.
Response writer
Writer can be obtained with :

getJspContext().getOut()

Note : Return value of getJspContext is a JspContext. Needs a cast to access PageContext objects/methods.

Body content processing
To process tag's body contents, call (in doTag()) :
getJspBody().invoke(java.io.Writer)

Note : invoke argument is null, output goes to response.

Uninitialized EL expressions
If tag body contents has an EL expression resolving to an unknown variable, this variable may be set in the doTag() method of the simple tag handler. Iterations may be made to process multiples values for the same EL variable.
Tag attributes
Tag attributes must be defined in the TLD file with the attribute block :

<attribute> <name>tagAttribute</name> <required>...</required> <rtexprvalue>...</rtexprvalue> </attribute>

Tag Handler setter method respect the JavaBean naming convention :
setTagAttribute(...)
Stop page processing
Simple tag handlers may stop further page processing by throwing a SkipPageException.
The rest of the page will not be processed. However, everything that was already processed WILL appear in output.

Note :

For included JSP invoking simple tags : if tag throws a SkipPageException, ONLY the included page processing is stopped. The including page continues its nomal processing.
JSP Fragments
JSP fragment MUST NOT containt ANY scripting element.
JSPFragment defines two methods :
getJspContext():JspContext
invoke(java.io.Writer) /!\ if writer is null, output goes to the response
Classis tags
Extends TagSupport or BodyTagSupport, implements Tag, IterationTag, BodyTag.
Lifecycle
Container :
+ Loads the classic tag handler class;
+ Instantiate class (no-args cstr);
+ Calls setPageContext(PageContext);
+ If tag is nested, calls the setParent(Tag) method;
+ If attributes, calls the attributes setter methods;
+ Calls the doStartTag() method
+ If tag declared to have a body and is NOT empty and doStartTag returns EVAL_BODY_INCLUDE : evaluates body
+ If body was evaluated, calls doAfterBody() method;
+ Calls doEndTag() method.

Note : Classic tags handlers MAY be reused; one instance my be pooled and used more than once.
Note : doStartTag() and doEndTag() are ALWAYS called once and ONLY ONCE.
doAfterTag() may be called zero to many times.

Response writer
Writer can be obtained with :
pageContext.getOut()
Body content processing
To process tag's body contents, doStartTag MUST return :
EVAL_BODY_INCLUDE
Stop page processing
Classic tag handlers may stop further page processing by returning a SKIP_PAGE value in the doEndTag() method.
Methods return types
Classic tag methods return types are int values.

doStartTag()
EVAL_BODY_INCLUDE Tag body is evaluated
SKIP_BODY Tag body is NOT evaluated (default)
doAfterTag()
EVAL_BODY_AGAIN Tag body is re-evaluated
SKIP_BODY calls doEndTag (default)
doEndTag()
SKIP_PAGE Stops page processing (like SkipPageException)
EVAL_PAGE Goes on with page processing (default)
NOTE : doStartTag() MUST be overriden if body is to be evaluated, because default return value of doStartTag() is : SKIP_BODY.
NOTE : if tld file declares that the tag body content is empty, then doStartTag() MUST return SKIP_BODY.

7.Servlet Descriptor

- URL Pattern
Ba cách <ur-pattern>
o Exact match:
• Bắt đầu với slash(/)
• Có thể có extension (không bắt buộc)
Vd: < url-pattern>/faces/userRegister.do < /url-pattern>
o Directory match:
• Bắt đầu với slash(/)
• Kết thúc với "/afkdf/adfd/*"
Vd: < url-pattern>/faces/*< /url-pattern>
o Extension match:
• Không bắt đầu với slash(/)
• Bắt đầu với "*"
• Theo sau là phần mở rộng
Vd: <url-pattern>*.do
Các quy tắc matching:
+: Container sẽ match request với url-pattern theo quy tắc từ exact match đến extension match
+: Nếu không có exact match, containser sẽ chọn longest mapping
Vd: /calc/customer/userRegister.do sẽ match với /calc/customer/* mặc dù nó cũng match với /calc/*
- Startup loading
Để load servlet tại thời điểm container start-up (hoặc wep app reload), sử dụng <load-at-startup> tag trong tag <servlet> của web.xml file
<servlet>
...
<load-at-startup>x</load-at-startup>
</servlet>

Giá trị của x có thể là:
+ <= 0 : không preloading; load tại thời điểm có request đầu tiên;
+ >= 1 : servlet loaded at startup.

Chú ý :

Nếu các servlet khác nhau có giá trị x khác nhau, thì servlet có giá trị x nhỏ nhất (mà > 0) sẽ load trước (load theo thứ tự tăng dần của x). Nếu các servlet khác nhau có cùng giá trị x, thì servlet được load dựa theo thứ tự khai báo trong web.xml file