

Expression Language implicit variables(case sensitive)

1. `pageContext` : `PageContext` object (`javax.servlet.jsp.PageContext`) asso. 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`. `paramname` 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)`).

All Available scopes to JSP

1. page scope => smallest scope = current page only.

Represented by PageContext object. Attributes added to the page are visible only to the current page.

(no direct equivalence in servlet API)

2. request scope => current request only

Attributes added to req. scope are visible to all pages from the same web-app, chained by the RD , ONLY for the current rq.

(Servlet API equivalence : RD's forward & include scenario)

3. session scope => current session only

Attributes added to the session scope are visible to all pages from same web-app, for multiple rqs BUT coming from the same clnt.

(Servlet API : HttpSession)

4. application scope => current web-application only.

Attributes added to appln scope are visible to all pages from SAME web-app for any rqs coming from any clnt.

widest possible scope for web-components

(Servlet API equivalent : ServletContext)

Info Available to error handling page.

Error Stack Trace `${pageContext.exception.stackTrace}`

Error Code `${pageContext.errorData.statusCode}`

Error Causing Page `${pageContext.errorData.requestURI}`

JSP Actions

Commands/messages for JSP container, but it will be used both in translation time + req. processing time.

Syntax :

```
<jsp:actionName attribute="attrVal" />
```

RD related actions

1. Forward action

```
<jsp:forward page="Relative URI of the forwarded page" />
```

OR

```
<jsp:forward page="Relative URI" >
```

```
<jsp:param name="pName" value="pValue" />
```

one or more params can be passed.

....

```
</jsp:forward>
```

100% same as RD's forward scenario.

2. Include action

```
<jsp:include page="Relative URI of the forwarded page" />
```

OR

```
<jsp:include page="Relative URI" >
```

```
<jsp:param name="pName" value="pValue" />
```

one or more params can be passed.

....

```
</jsp:include>
```

100% same as RD's include scenario.

Diff bet. include directive and include action

1. Via include dir., contents are included @ translation time.

Whereas , in include action : included @ runtime(req. processing time)

2. include dir. is typically used for including static contents(eg. HTML content). Vs include action typically used for including dyn cont.(eg . servlet,JSP)

3. Via include dir : the included page can access the page scoped vars or attrs. of the 1st page.

Via include action : scope = req(equivalent to RD's include scenario.) So cant

access page scoped vars or attributes from the included page.

3. To add request params : there is no API in servlets. But possible in JSP. How?

```
<jsp:param name="pName" value="pValue" />
```

child action which can be embedded into `<jsp:forward>`, `<jsp:include>` or `<jsp:plugin>` actions.

setters & getters syntax

//setters

```
public void setPropName(Type val)
```

Type : data type of the property.

eg. In EmpBean

```
private String name;
```

```
private Address empAdr;
```

//setters

```
public void setName(String val)
```

```
public void setEmpAdr(Address a)
```

//getters

```
public Type getPropName()
```

eg :

```
public String getName()
```

```
public Address getEmpAdr()
```


JSP using Java Beans via standard JSP actions

1. `<jsp:useBean id="Bean Ref Name" class="Fully qualified Bean class name" scope="page|request|session|application" />`
 default scope =page

2.1 `<jsp:setProperty name="Bean Ref Name" property="propname" value="fixed/dyn value" />`

Meaning : WC invokes

Bean Ref name.setPropName(value attr value)

eg. `<jsp:SetProperty name="emp" property="sal" value="${param.f1}" />`

WC invokes : `emp.setSal(request.getParameter("f1"))`

2.2 2.1 `<jsp:setProperty name="Bean Ref Name" property="propname" param="paramName" />`

param : rq. param.

URL : `http://localhost:9080/test_jsp/actions/test_bean.jsp?f1=aa`

`<jsp:setProperty name="emp" property="name" param="f1" />`

WC invokes : `emp.setName(request.getParameter("f1"))`

2.3 `<jsp:setproperty name="Bean Ref Name" property="*" />`

eg.

URL : `http://localhost:9080/test_jsp/actions/test_bean.jsp?name=aa&f2=101&sal=1000`

`<jsp:setProperty name="emp" property="*" />`

EmpBean private data members are : id,name, sal

How many setters : 2, `emp.setName("aa")` & `emp.setsal(1000)`

3. `<jsp:getProperty name="Bean Ref Name" property="propName" />`

eg : `<jsp:getProperty name="emp" property="sal" />`

WC invokes : `emp.getSal()`----> converted to String & sent to clnt Browser.

Better alternative : EL syntax : better suited for properties based on ref. types or collection(List,Map,Set).

eg : `${sessionScope.emp.adr.city}` : return the city name from Address Bean

NOTE :

In req/session/application scope, if u r trying to display prop. values from different pages via : `<jsp:getProperty>` , u have to add `<jsp:useBean>` ow u will get Null Object exc.

For EL syntax : no such need.

MVC advantages

1. Division of responsibilities among the various components.
2. One component is not TOO MUCH burdened with TOO many jobs.
3. Cleaner separation between request processing, navigation, business logic and presentation logic.
4. Reusability of Business logic components across various envs.
5. Each comp. can be implemented completely inde. of others.
eg . Can generate fresh/attractive/new set of views(display renderers) keeping same B.L & navigation logic.
6. Single model can be made to support multiple views & which view to select can be decided dyn.

Implementation of MVC : using Front Servlet Controller Pattern

1. Thin clnt sends the HTTP rq. to a servlet controller.
2. Servlet/s reads the request params & performs initial processing if reqd.
3. Servlet controller will instantiate the Model componet/s (Java Beans)
2 ways : can directly invoke the parameterised constr. to instantiate & load the state of the JB's.

OR

Can invoke the def. constr & then invoke setters similar to ur JSP scenario.

Now JB's hold the data/state of the web appln reflecting the clnt state.

4. Servlet controller sends rq. to JB's to execute B.L

As a result of B.L , the servlet controller dyn. forwards/redirects the client to view (consisting of 1 or more JSPs)

5. JSPs contact JB's to load the state(using <jsp:getProperty> or EL syntax) .

6. Using the state info. JSPs genrate dyn contents & send the resp to the clnt.

Developing Custom Tags (via Simple tags)

Refer doc of : `javax.servlet.jsp.tagext.SimpleTag` i/f & its imple. class `SimpleTagSupport`.

Need : JSP std. actions may not be sufficient to take care of all needs of application. So extend it by adding ur own actions or tags.

Steps

1. Writing Custom Tag Handler :

1.1 Java class which extends `SimpleTagSupport`

1.2 Override public void `doTag()` throws `JSPException`, `IOException` which contains the exec. logic of the tag.

1.3 To get the reference of the invoking JSP page :

Use the API : `JSPContext.getContext()`

From `JSPContext` : to get the `JSPWriter` use method :

`JSPWriter.getOut()`

& then generate & send the dyn. contents to clnt browser using `print/println` methods.

Life-cycle of Simple Tags

ref: javax.servlet.jsp.tagext.SimpleTag i/f(jsp-api.jar)

When JSP author invokes the Simple tag(eg: <my:welcome/>) simple tag life-cycle begins.

- 1.WC locates,loads & instantiates the Tag handler class.
- 2.Invokes public void setJspContext(JspContext ctx) on the tag instance.
- 3.Invokes setParent() if our custom tag is the nested tag(optional)
- 4.Invokes setters for all the attributes supplied by the JSP author (skipped if no-attributes)
- 5.Invokes public void setJspBody(JspFragment jspf) (skipped if body content empty)
- 6.Finally invokes doTag() method on the tag instance, from where exec. logic of the tag gets executed.(mandatory, never skipped) , tag instance is destroyed.

```
for (AccountBean ac : accts)  
SoP(ac.id.....)
```


Struts flow of Control(refer to Block diag.)

- 1.Thin client sends the rq. using post method & action=*.do
2. On the struts enabled web-app, it reaches ActionServlet (controller element supplied by Struts : from org.apache.struts.action.ActionServlet). ActionServlet performs basic rq. & processing & forwards the rq.(i.e client) to Request Processor(Controller element from Struts : same pkg)

2.R.P does follo.

- 2.1 Instantiates FormBeans if any.(FormBeans will be supplied by Prog. by subclassing org.apache.struts.action.ActionForm)

Form Beans : i/p Beans : to transfer data(rq. params from controller to model) . default scope=request. purpose : validating P.L. +data xfer.

- 2.2 Invokes the reset & validate methods on the formbean.

- 2.3 In case of presentation logic validation err: client can be auto. forwarded to the orig. page(eg : login.html) with highlighted errs.

- 3.In absence of P.L validation errs, R.P instantiates Action class(singleton instance will be maintained in a web-app) Action class supplied by prog. by

sub-classing org.apache.struts.Action class.

4. In action class override the execute method.

```
public ActionForward execute(ActionMapping a, ActionForm f,  
HttpServletRequest rq, HttpServletResponse rs)
```

This method will be auto. called by R.P.

5. Return value of the execute contains the Forward name.

This forward name will be resolved by R.P using the navigation controller (/web-inf/struts-config.xml) & client will be auto. forwarded(default scenario, can be replaced by redirect) to appro. View JSp

6. View JSP can load the state from the DTO(data xfer objs, result beans or Form beans) & generate the dyn. cont.

7. This FINALLY!!!! finishes 1st rq. via struts flow of control.

