Adv. Java means DURGA SIR..

ADV.JAVA With SCWCD / OCWCD

JSP Material

3. Building JSP Pages Using the Expression Language(EL)

DURGA M.Tech

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students for last 14 years across INDIA

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

Building JSP Pages Using Expression Language (EL)

Agenda:

1) Introduction

2) EL Implicit Objects :
   - pageScope, requestScope, sessionScope, applicationScope
   - param & paramValues
   - header & headerValues
   - initParam
   - cookie
   - pageContext
   - handling errors

3) EL Operators :
   - property access operator (.)
   - collection access operator []
   - Arithmetic operators (+, -, *, /, %)
   - Relational operators (lt, gt, eq, ne) OR (<, >, <=, >=, ==, =!)
   - Logical operators (and, or, not)
   - conditional operator (?:)
   - empty operator (empty)
   - EL operator presidency
   - EL reserved words
   - EL Vs null

4) EL Functions :
   - Write a java class
   - Write a TLD file : (tag library descriptor file)
   - Write a taglib directive
   - Write EL function call
   - Work Flow : (in EL function application)

Introduction

- E-L introduced in Jsp 2.0v , the main objective of EL is to eliminate java code from the jsp.
- In general we can use EL with JSTL and custom tags for complete elimination of java code from the jsp.
- EL expressions are always with in { } and prefixed with $ (dollar) sign
- syntax:
- ${leftVatiable.rightVariable}

- The leftVariable can be any EL implicit objects/attributes stored in some scopes (page scope, request scope, session scope, application scope)

1. EL Implicit Objects (11)
2. EL Operators
3. EL Functions

To print the value of request parameter "user" in jsp, write the code ?

```
<%= request.getparameter("user") %>

// is equivalent code

${param.user}
```

To print the value of session scoped attribute "x" in jsp ?

```
<%= session.getAttribute("x") %>

// is equivalent code

${sessionScope.x}
```

To use any variable x in EL compulsory it should be an attribute of some scope ?

```
${x}
```

It prints the value of x attribute present in any scope(pageScope, requestScope, sessionScope, applicationScope) and if there is no such attribute then it prints BlankSpace but not null

Ex 1:

```
<%!
        int x = 10;
%>
The value of x is : ${x}

output :
The value of x is :   //BlankSpace
```

Ex 2:

```
<%@page isELIgnored="false" %>
<%! int x=10; %>
<% pageContext.setAttribute("x",x); %>
The value of x is : ${x}

output :
The value of x is : 10
```

EL Implicit Objects (11) :

EL contains 11 implicit objects

The power of EL is just because of these implicit objects only.

The following is the list of EL implicit objects

1. pageScope 2. requestScope 3. sessionScope 4. applicationScope	---->	map objects to retrieve scoped attributes
5. param 6. paramValues	---->	map objects to retrieve form parameters
7. header 8. headerValues	---->	map objects to retrieve request headers
9. cookie	---->	map objects to retrieve cookies
10. initParam	---->	map objects to retrieve context parameters
11. pageContext	---->	java bean object (it is not a map)

pageScope, requestScope, sessionScope, applicationScope

1. pageScope	---->	pageContext.getAttribute()
2. requestScope	---->	request.getAttribute()
3. sessionScope	---->	session.getAttribute()
4. applicationScope	---->	application.getAttribute()

Ex : To print the value of session scoped attribute "x"

```
${sessionScope.x}
```

Ex : To print the value of request scoped attribute "x"

```
${requestScope.x}
```

Ex :

```
${sessionScope.x}
```

It prints the value of session scoped attribute "x" , in sessionScope if there is no such attribute then we will get BlankSpace

Ex :

```
${x}
```

jsp engine first checks in pageScope for attribute "x" if it is available then it prints the value, then it's not available it will check in requestScope followed by sessionScope and applicationScope, it simply act as pageContext.findAttribute(x);

scopes.jsp

Note : ${leftVariable.rightVariable}

- If leftVariable is a map then rightVariable should be "key", If the specified key is not available then we will get BlankSpace as a output.
- If leftVariable is a bean then rightVariable should be "property", If the specified property is not available then we will get "PropertyNotFoundException".

```
${person.empId}
// here person attribute stored in some scope
```

In Servlet code :

```
package foo;

import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ELServletDemo extends HttpServlet {

        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {
```

```
                foo.Person p = new foo.Person();
                p.setName("Akshay");

                foo.Dog d = new Dog();
                d.setName("puppy");

                p.setDog(d);

                request.setAttribute("person", p);
                request.setAttribute("arun", "SCWCD");

                RequestDispatcher rd = request.getRequestDispatcher("demo.jsp");
                rd.forward(request, response);
        }
}
```

in Jsp code :

```
${requestScope.arun}
${requestScope.person.dog.name}

<%-- ${person.dog.name} --%>
```

Note :

- use requestScope implicit object to get request scoped attributes but not request parameters.
- If we want request properties then we should go for pageContext EL implicit object.

param & paramValues :

We can use these implicit objects to retrieve form parameter values

1. param	---->	request.getParameter()
2. paramValues	---->	request.getParameterValues()

${param.x} // it prints the value of form parameter x, if the specified parameter is not available then we will get BlankSpace
If parameter associated with multiple values then we will get first value.

param :
Map

String	String
uname	ashok

course	scwcd
place	hyd

paramValues :
Map

String	String[]
uname	{ashok, arun, akshay}
course	{scjp, scwcd}
place	{hyd}

${paramValues.uname}
// String[] will be returns on that Object class toString() is called.

${ paramValues.uname[0] }	--->	ashok
${ paramValues.uname[100] }	--->	BlankSpace
${ paramValues.user }	--->	BlankSpace

Note : EL handles null and ArrayIndexOutOfBoundException very nicely and prints BlankSpace.

form.html

```
<h3>Enter Your Details :</h3>
<form action="formParam.jsp" method="post">
        Enter Name :<input type="text" name="name" > <br>
        Enter Mail :<input type="text" name="mail" > <br>
        Enter Age:<input type="text" name="age" > <br>
        Enter Food 1 :<input type="text" name="food" > <br>
        Enter Food 2 :<input type="text" name="food" > <br>
        <input type="submit" value="Submit"> <br>
</form>
```

formParam.jsp

```
<%@page isELIgnored="false"%>
```

```
<h3>Form Parameter Values :</h3>
Name : ${param.name} <br/>
Mail : ${param.mail} <br/>
Age : ${param.age} <br/>
Food : ${param.food} <br/>
Sex : ${param.sex} <br/>

ParamValues.name : ${paramValues.name} <br/>
ParamValues.name[0] : ${paramValues.name[0]} <br/>
ParamValues.name[1] : ${paramValues.name[1]} <br/>

ParamValues.food : ${paramValues.food} <br/>
ParamValues.food[0] : ${paramValues.food[0]} <br/>
ParamValues.food[1] : ${paramValues.food[1]} <br/>

ParamValues.food[100] : ${paramValues.food[100]} <br/>
```

header & headerValues :

These are exactly similar to param and paramValues except that these are retrieving request headers.

1. header	---->	request.getHeader()
2. headerValues	---->	request.getHeaders()

If the specified request header is not available you will get BlankSpace as a output.

${ header.x }	--->	BlankSpace
${ header.host }	--->	localhost:2020
${ headerValues.host }	--->	java.lang.String@1234

header.jsp

```
<%@page isELIgnored="false"%>

<h3>Header Information :</h3>

Accept : ${header.accept} <br/>
Accept Values [0] : ${headerValues.accept[0]} <br/>
Accept Values [1] : ${headerValues.accept[1]} <br/>

Host : ${header.host} <br/>
Host [0] : ${headerValues.host[0]} <br/>

Some x is : ${header.x} <br/>
Accept Language : ${header['accept-language']} <br/>
```

initParam :

By using initParam implicit object we can access ServletContext parameters but not servlet initialization parameters.

initParam ---> application.getInitParameter()

${initParam.user} // it prints the value associated associated with context parameter user.

If the specified parameter is not available then we will get BlankSpace.

```
${initParam.x} //BlankSpace
```

initParam.jsp

```
<%@page isELIgnored="false" %>

<h3>Init Parameters are :</h3>

Init Param Name : ${initParam.name} <br/>
Init Param Mail : ${initParam.mail} <br/>
Init Param Age : ${initParam.age}   <br/>

The X value is : ${initParam.x} <br/>
The Y value is : ${initParam.y} <br/>
```

web.xml

```
<display-name>EL Implicit Objects</display-name>

<context-param>
 <param-name>name</param-name>
 <param-value>Ashok</param-value>
</context-param>
```

```
<context-param>
 <param-name>mail</param-name>
 <param-value>admin@jobs4times.com</param-value>
</context-param>
<context-param>
 <param-name>age</param-name>
 <param-value>2022</param-value>
</context-param>
<context-param>
 <param-name>x</param-name>
 <param-value>12345</param-value>
</context-param>
<context-param>
 <param-name>y</param-name>
 <param-value>54321</param-value>
</context-param>
```

To print the value of "userName" cookie ?

```
<%
        Cookie[] c=request.getCookies();
        for(int i=0;i<c.length;i++) {
             if(c[i].getName().equals("userName")) {
                    out.println(c[i].getValue());
             }
       }
%>

(OR)

<%@page isELIgnored="false" %>

Cookie value is : ${cookie.userName.value} <br/>
```

cookie :

By using cookie implicit object we can retrieve cookies associated with the request

cookie ----> request.getCookies()

cookie :
Map

String	String

JSESSSIONID	1234
c1	SCJP

cookie.jsp

```
<%@page isELIgnored="false" %>

<%
Cookie c1=new Cookie("c1","SCJP");
Cookie c2=new Cookie("c2","SCWCD");

response.addCookie(c1);
response.addCookie(c2);
%>

<h3>The Cookie Id is :</h3>
${cookie.JSESSIONID.name} <br/>
${cookie.JSESSIONID.value} <br/>
${cookie.JSESSIONID} <br/>

Cookie c1 is : ${cookie.c1} <br/>
Cookie c1 name is : ${cookie.c1.name} <br/>
Cookie c1 value is : ${cookie.c1.value} <br/>
Cookie c2 name is : ${cookie.c2.name} <br/>
Cookie c2 value is : ${cookie.c2.value} <br/>
```

pageContext :

- This is only one EL implicit object which matches with jsp implicit object.
- This is only one EL implicit object which is a non-map object by using this implicit object we can bring all jsp implicit objects into EL.

```
<%@page isELIgnored="false" %>

<% session.setAttribute("course","SCJP"); %>

${pageContext.request.method} <br/>
${pageContext.session.id} <br/>
${pageContext.request.Cookies[0]} <br/>
${course} <br/>
```

```
${request.method} <br/>
${session.id} <br/>
```

handling errors :

- There is an exception and it was not caught then in this case the request will be forwarded along with exception to the error page URL, here the error page is specified errorPage attribute of page directive and by specified URL.
- This is so simple because the error page itself is identified by using the implicit object exception, more over in addition to exception object an error page also has the following request attributes available.
- javax.servlet.error.status-code
- javax.servlet.error.request-URI
- javax.servlet.error.servlet-name
- When ever we use the pageContext.getErrorData() method, it is possible to get an instance of javax.servlet.jsp.Error-Data class
- This will provide to find a simple way to access the above attributes. (status-code, request-URI, servlet-name)

index.jsp

```
<%@page isELIgnored="false" %>

<%@page errorPage="error.jsp" %>
<%
out.println(10/0);
%>
```

error.jsp

```
<%@page isErrorPage="false"%>

The Raised Exception is : ${pageContext.exception.message} <br/>
The Status code is : ${pageContext.errorData.statusCode} <br/>
The Request URI is : ${pageContext.errorData.requestURI} <br/>
The Servlet Name is : ${pageContext.errorData.servletName} <br/>
```

All EL implicit objects are map object type except pageContext.

EL Operators :

EL contains it's own specific operators the following is list of all possible operators

1. property access operator (.)
2. collection access operator []
3. Arithmetic operators (+, -, *, /, %)
4. Relational operators (lt, gt, eq, ne) OR (<, >, <=, >=, ==, =!)
5. Logical operators (and, or, not)

6. conditional operator (?:)
7. empty operator (empty)

property access operator :

```
${leftVariable.rightVariable}
 //  map -----key
 //  bean ----property
 // left attribute stored in some scope
```

- If the expression has a variable followed by . (dot) , the left variable must be a map or bean.
- The thing to the right of the . (dot) must be a key or bean property.

The thing on the right must follow normal java naming rules for the identifies.

```
${person.name}

// name must starts with letter or underscore or $
// after the first character we can include the numbers
// can't be a java keyword
${initParam.age}

//map ---- key
${person.dog}

//bean ---- property
${person.1}

//javax.el.ELExpeption:ErrorParsing
```

collection access operator :

```
${leftVariable.rightVariable}
 //  Map -----key      //quotes are mandatory
 //  Bean ----property //quotes are mandatory

 // List ---- index   //quotes are optional
 // Array ---- index  //quotes are optional
```

When we use the . operator the thing on the left can be only a Map or Bean and thing on the right must follow java naming rules for the identifies but with collection access operator, the thing on the left can also be a List or Array that also meaning the thing on the right can be a number or any thing that resolves to a number or an identifier that doesn't java naming rules.

Ex :

```
${musicMap[MusicTypes[0]]}
```

```
(OR)

${musicMap[MusicTypes["0"]]}
```

But you can't

```
${musicMap[MusicTypes["zero"]]}
                // It can't convert into int
```

Map Example :

```
${initParam['mail']}

${initParam['age']}

${initParam['name']}

${initParam[age]}
 <%-- blank space --%>
```

Note : If we are not using code symbol then it is treated as attribute if the specified attribute is not available then we will get blank space.

Bean in servlet code :

```
package foo;

import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ELServletDemo extends HttpServlet {
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                            throws ServletException, IOException {

                foo.Person p = new foo.Person();
                p.setName("Akshay");

                foo.Dog d = new Dog();
                d.setName("puppy");

                p.setDog(d);

                request.setAttribute("person", p);
                RequestDispatcher rd = request.getRequestDispatcher("view.jsp");
                rd.forward(request, response);
        }

}
```

view.jsp

```
<%@ page isELIgnored="false"%>

${person['dog']['name']}

${person[dog][name]}   //blankspace
```

List Example :

Servlet code :

```
package foo;

import java.io.IOException;
import java.util.ArrayList;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ELServletDemo extends HttpServlet {

        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {

                java.util.ArrayList list=new ArrayList();
                list.add("Ashok");
                list.add("Arun");
                list.add("Akshay");
                list.add("Saicharan");

                request.setAttribute("x", list);
                request.setAttribute("index", "2");

                RequestDispatcher rd = request.getRequestDispatcher("view.jsp");
                rd.forward(request, response);
```

```
    }

}
```

view.jsp

```
<%@ page isELIgnored="false"%>
The first name is : ${x[0]} <br/>
The second name is : ${x[1]} <br/>

List Names are : ${x}
```

Array Example :

servlet code :

```
package foo;

import java.io.IOException;
import java.util.HashMap;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ELServletDemo extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

        java.util.Map musicMap = new HashMap();
        musicMap.put("melody", "song1");
        musicMap.put("DJ", "song2");
        musicMap.put("fastBeat", "song3");
        musicMap.put("slowBeat", "song4");

        request.setAttribute("musicMap", musicMap);
        String[] musicTypes = {"melody","DJ","fastBeat","slowBeat"};
        request.setAttribute("musicType", musicTypes);

        RequestDispatcher rd = request.getRequestDispatcher("view.jsp");
        rd.forward(request, response);
    }

}
```

view.jsp

```
<%@ page isELIgnored="false"%>
```

```
${ musicMap[musicType[1]] }

${ musicMap["DJ"] }
```

note : first inner most brackets will be evaluated.

Ex 2 :

```
<%@ page isELIgnored="false"%>
<%
	String[] s={"A","B","C","D"};
	pageContext.setAttribute("s",s);
%>
${s[0]} <br/>
${s[1]} <br/>
${s[100]} <br/>
```

Example :

Person.java

```
package foo;

public class Person {
	private String name;
	private Dog dog;

	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public Dog getDog() {
		return dog;
	}
	public void setDog(Dog dog) {
		this.dog = dog;
	}

}
```

Dog.java

```
package foo;

public class Dog {
	private String name;
	private Toy[] toys;

	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public Toy[] getToys() {
		return toys;
	}
	public void setToys(Toy[] toys) {
		this.toys = toys;
	}
}
```

Toy.java

```
package foo;

public class Toy {
	private String name;

	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
}
```

In Servlet code :

```
package foo;

import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ELServletDemo extends HttpServlet {

	public void doGet(HttpServletRequest request, HttpServletResponse response)
						throws ServletException, IOException {

		foo.Person p = new Person();
```

```
                p.setName("SAI");

                foo.Dog d = new Dog();
                d.setName("puppy");

                foo.Toy t = new Toy();
                t.setName("bear1");
                foo.Toy t1 = new Toy();
                t1.setName("bear2");
                foo.Toy t2 = new Toy();
                t2.setName("bear3");


                p.setDog(d);
                d.setToys(new foo.Toy[]{t,t1,t2});

                request.setAttribute("person", p);

               RequestDispatcher rd = request.getRequestDispatcher("view.jsp");
               rd.forward(request, response);
       }

}
```

view.jsp

```
<%@ page isELIgnored="false"%>

${person.name} 's dog ${person.dog.name} ,   toys are
${person.dog.toys[0].name} , ${person.dog.toys[1].name} , and
${person.dog.toys[2].name}
```

SAI 's dog puppy , toys are bear1 , bear2 , and bear3

Note : Where ever property access operator is required there we can use collection access operator but where ever collection access operator is required may not be use property access operator.

Arithmetic operators : (+, -, /, *, %)

EL doesn't support operator overloading and + operator always meant for Arithmetic addition but not for String Concatenation.
+ operator :

```
<%@ page isELIgnored="false"%>

2+3= ${2+3}
"2"+3= ${"2"+3}
"2"+'3'= ${"2"+'3'}
abc+'3'= ${abc+'3'}
```

```
"abc"+'3'= ${"abc"+'3'}
  // java.lang.NullPointerException:For input string: "abc"

""+3= ${""+'3'}
  // java.lang.NullPointerException:For input string: ""

null+'3'= ${null+'3'}
```

- operator :

```
<%@ page isELIgnored="false"%>

2-3= ${2-3}
"2"-3= ${"2"-3}
"2"-'3'= ${"2"-'3'}
abc-'3'= ${abc-'3'}

"abc"-'3'= ${"abc"-'3'}
   // java.lang.NullPointerException:For input string: "abc"

""-3= ${""-'3'}
  // java.lang.NullPointerException:For input string: " "

null-'3'= ${null-'3'}
```

<u>* operator, / operator :</u> all rules are exactly similar to + operator except that division operator always follows floating point Arithmetic.

<u>syntax :</u>

```
${a/b} or ${a div b}
```

<u>Ex :</u>

```
${10/2} = 5.0

${10/0} = Infinity

${0/0} = NaN
```

<u>% operator :</u>

All rules are exactly similar to + operator here both integral and floating Arithmetics are possible.

<u>syntax :</u>

```
${a%b} or ${a mod b}
```

Ex :

```
${10%3} // 1

${10%0} //java.lang.ArithmeticException: / by zero

${10.0%0}  //NaN

${""/'3'}
 //java.lang.NumberFormatException: empty String
```

Note : In Arithmetic operator null evaluates Zero, the presidency of Arithmetic operators *, /, %, +, -.

In side Servlet code :

```
package foo;

import java.io.IOException;
import java.util.ArrayList;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ELServletDemo extends HttpServlet {

        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                            throws ServletException, IOException {

                java.util.ArrayList numList = new ArrayList();
                numList.add("1");
                numList.add("2");
                numList.add("3");
                numList.add("4");
                request.setAttribute("numbers", numList);

               String[] s = {"melody","DJ","fast","slow"};
               request.setAttribute("musicList", s);

              RequestDispatcher rd = request.getRequestDispatcher("view.jsp");
              rd.forward(request, response);
     }

}
```

view.jsp

```
<%@ page isELIgnored="false"%>

${musicList[numbers[0]]} <br/>
${musicList[numbers[1]+2]} <br/>
${musicList[numbers[numbers[0]]+1]} <br/>
${musicList[numbers["2"]]} <br/>
```

Relational operators :

<	or	lt
<=	or	le
>	or	gt
>=	or	ge
==	or	eq
=!	or	ne

Ex :

```
<%@ page isELIgnored="false"%>

Relational operators :

${3 < 4}   //true
${3 > 4}   //false
${3 >= 4}  //false
${3 <= 4}  //true
${"abc" == "abc"} //true
${2!=1}        //true
${sai == sai}   //true
${3 < sai}      //false
${String < String}  //false
```

In Relational Operators , the unknown variable evaluates as a false.

Logical operators :

&&	or	and
||	or	or

!	or	not

Ex :

```
<%@ page isELIgnored="false"%>

${true and true}   //true
${false || false}   //false
${!true}        //false
${ashok && ashok}   //false
${not ashok}   //true
```

Note : In logical operators unknown variable evaluates false.

conditional operator : (?:)

```
<%@ page isELIgnored="false"%>

${ (3 < 4)?"yes":"no" }       //yes
${ (true == false)?"Right":"wrong" }   //wrong
${ (2 < 3)?"correct":"wrong" }      //correct
```

In the case of conditional operator unknown variable treats a variable stored in some scope.

empty operator :

syntax :

```
${empty obj}
```

returns true iff object doesn't exists/ an empty array/ object is an empty collection/ object as a empty String in all other cases it returns false.

Ex :

```
<%@ page isELIgnored="false"%>
<%@page import="java.util.ArrayList"%>

<%ArrayList l=new ArrayList(); %>

${empty test}   //true
${empty "durga"}   //false
${empty null}   //true
${empty " "}   //false
${empty l}   //true
```

In Servlet code :

```
package foo;
```

```
import java.io.IOException;
import java.util.ArrayList;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ELServletDemo extends HttpServlet {

        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                throws ServletException, IOException {

                int num = 2;
                request.setAttribute("num", num);

                Integer I = new Integer(3);
                request.setAttribute("integer", i);

                java.util.ArrayList list = new ArrayList();
                list.add("true");
                list.add("false");
                list.add("5");
                list.add("6");
                request.setAttribute("list", list);

                 RequestDispatcher rd = request.getRequestDispatcher("jspDemo.jsp");
                 rd.forward(request, response);
   }

}
```

view.jsp

```
<%@ page isELIgnored="false"%>

<jsp:useBean id="myDog" class="foo.Dog">
<jsp:getProperty name="myDog" property="name" />
</jsp:useBean>

${myDog.name and true}  //false

${requestScope["integer"] ne 4 and 6 le num or false} //false
 //EL executes left to right
```

EL operator presidency :

1. unary (! , empty)
2. Arithmetic (*, /, %, +, -)
3. Relational (<, >, <=, >=, ==, =!)
4. Logical (&&, ||)
5. conditional operator (?:)

EL reserved words :

lt		true		instanceof		div
gt		false		empty		mod
le		and		null		
ge		or				
ne		not				
eq						

EL Vs null :

1. EL behavior very nicely with null, in Arithmetic operator null is treated as a zero.
2. With String evaluation null is treated as empty String (" ")
3. With logical operators null is treated as false
4. In the case of empty operator null is treated as true
5. In the case of relational operator null is treated as false

Assume that there is no attribute named with "foo" but there is an attribute named with "bar" but bar doesn't have any property or key name "foo"

- ${foo} // BlankSpace
- ${foo[bar]} // BlankSpace
- ${bar[foo]} // BlankSpace
- ${foo.bar} // BlankSpace
- ${7/foo} // Infinity
- ${7%foo} // java.lang.ArithmeticException
- ${true and false} // false
- ${true or false} // true

- ${not foo} // true

EL Functions :

1. The main objective of EL is separate code from the jsp
2. If we are having any business functionality separate in to a java class and we can invoke this functionality through EL syntax.
3. The page designer has to know only function name, tld file uri to get its functionality hence EL is alternative to custom tags.

Designing EL function application contains the following stages :

1. Writing a java class with required functionality.
2. Writing a TLD file to map a java class to Jsp
3. Write a taglib directive to make business functionality available to Jsp.
4. Write EL function call.

Writing a java class :

1. Any java class can act as a repository for EL functions , the only requirement of the method that act as a functions it should be public and static.
2. The method can take parameter also, there are no restriction on return type even void return type is also allowed.

Writing a TLD file : (tag library descriptor file)

TLD file is a xml file which provides mapping between java class (where functionality is available) and jsp (where functionality is required)

We can configure EL function by using function tag in TLD file , it contains the following 4 steps :

1. description : It is an optional tag
2. name : By means of this name only wecan call EL functionality in jsp (this name need not be same as original method name, it is a logical name)
   There is no <function-name> tag to configure EL function .
3. function-class : The fully qualified name of java class where EL function defined.
4. function-signature : The signature of EL function which is defined in java class , here we have to specified return type, method name and argument list.

Writing a taglib directive :

We have to declare taglib directive in the jsp to provide location of the TLD file

```
<%@taglib prefix="mime" uri="www.jobs4times.com" %>
```

Writing EL function call :

```
${mime.getBalance(435678)}
```

demo.jsp

```
<%@ page isELIgnored="false"%>
<%@taglib prefix="mime" uri="DiceFunctions"%>

Your 3 - Digit Lucky Number :

${mime:rollIt()} <br/>
${mime:rollIt()} <br/>
${mime:rollIt()} <br/>
```

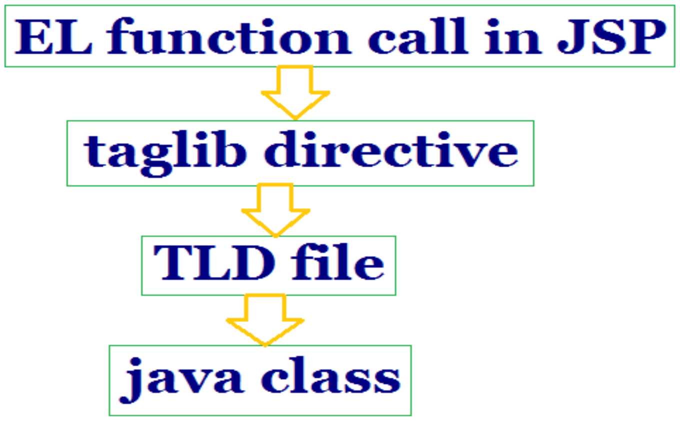
DiceRoller.java

```
package foo;

public class DiceRoller {

public DiceRoller() {
 System.out.println("DiceRoller");
}
public static int rollDice() {
 return (int) (Math.random() *6+1);
}

}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>

 <taglib version="2.1" >
 <tlib-version>1.0</tlib-version>
 <uri>DiceFunctions</uri>
 <functions>
  <name>rollIt</name>
  <function-class>foo.DiceRoller</function-class>
  <function-signature>int rollDice()</function-signature>
 </functions>
 </taglib>
```

Work Flow : (in EL function application)



1. When ever jsp engine encounters EL function call in jsp first it identifies the prefix and check for corresponding taglib directive with matched prefix.
2. From the taglib directive jsp engine identifies uri and checks for corresponding TLD file with matched uri.
3. From the TLD file jsp engine identifies the corresponding java class and method.
4. Jsp engine execute that method and returns its result to jsp.

Note :

- The common thing between EL function call and taglib directive is "prefix".
- The common thing between taglib directive and TLD file is "uri".
- The common thing between TLD file and java class is "class name" and "method" .

Write a program to invoke Math class sqrt() as a EL function call with name m1

public static double sqrt(double) //method signature

index.jsp
```
<%@ page isELIgnored="false"%>
<%@taglib prefix="mime" uri="MathFunctions"%>

The squre root of 4 is : ${mime.m1(4)}
The squre root of 5 is : ${mime.m1(5)}
```

mathFunction.tld
```
<taglib version="2.1" >
 <tlib-version>1.0</tlib-version>
 <uri>MathFunctions</uri>
 <functions>
  <name>m1</name>
  <function-class>java.lang.Math</function-class>
  <function-signature>double sqrt(double)</function-signature>
 </functions>
 </taglib>
```