**Servlet & JSP Notes**

rgupta.mtech@gmail.com

**5 days training program**

**Day-1**

Servlet introduction
Servlet life cycle
Http protocol
Role of container
Servlet Life cycle
Hello World
Servlet API introduction
Servlet config and context parameters and use
Request Dispaching vs redirecting

Ex:
1. login application
2. counter servlet

**Day-2**

MVC design pattern
MVC application: bok advise application

Listners
- ServletcontextListner
  - loding jdbc driver at application startup time
  - connection pooling concept and implementation
- HttpSessionListner
  - Count no of active user in the application.

Attributs and uses

Session management
- Cookies, Hidden field, url rewriting, HttpSession
- Examples: simple shopping cart.

Filters
- Security login filter.

**Day-3**

JSP nuts and bolts
- scriptlet, decleration , directive etc
- hello world applicatin

JSP life cycle
Model 1 and model 2 architecture
JSP tages in details
Include action and include directive
Error page
- Ex: simple login application
  - Simple intrest calculation application

Scope: page, request,session, application

Initilization of JSP
Using context and config with JSP

**Day-4**

JSP use bean
EL
Introduction to JSTL
Core tag,formatting tag, jdbc tag, xml formatting tag

**Day-5**

DAO, DTO pattern
Desigining MVC Book mgt system application

**Day-1**

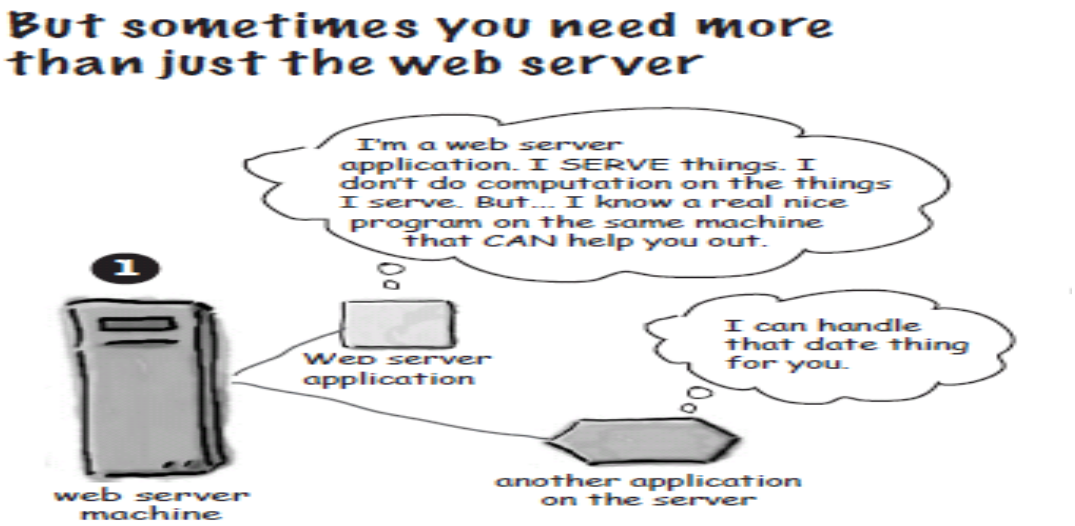
==========

**Servlet introduction**

**What is servlet?**

Enhance functionality of server

**Web server can only handle static request** form the client , for handling
dynamic request we need some dynamic scripting technologes such as CGI,perl
asp, asp.net etc and an component that can handle their life cycle and can provide them communication support.



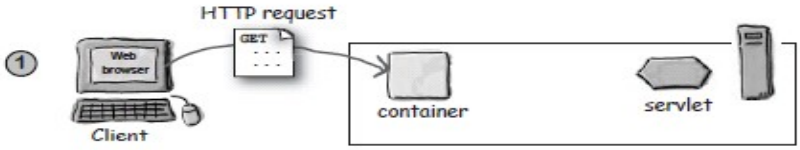

So we need something else then web server to handle dynamic request and that componet is called web container as tomcat

Web container is the component that manage life cycle of servlet/jsp

**What web container provides?**

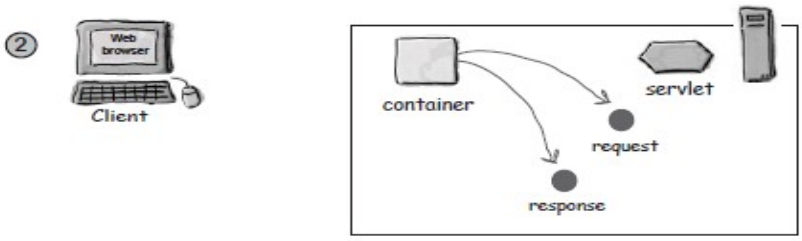
- Communication support
- Lifecycle management
- Multithreading support
- Declarative security
- JSP Support

## How an container handle dynamic request




User clicks a link that has a URL to a servlet instead of a static page.

---

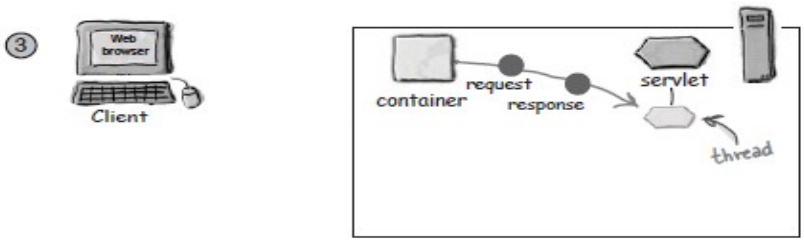



The container "sees" that the request is for a servlet, so the container creates two objects:

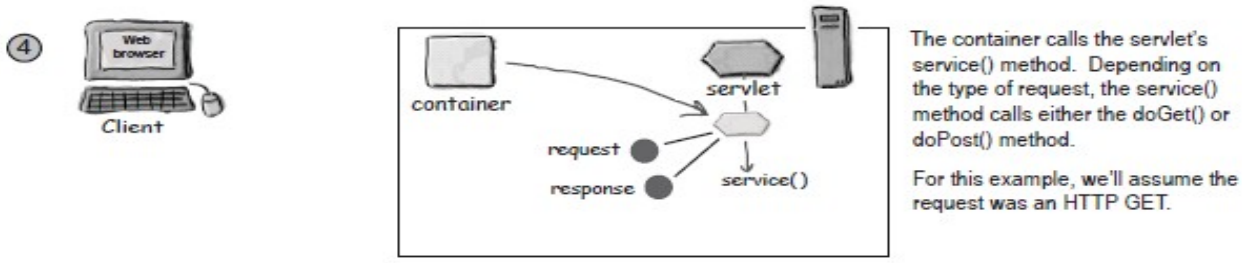
1) HttpServletResponse

2) HttpServletRequest

---



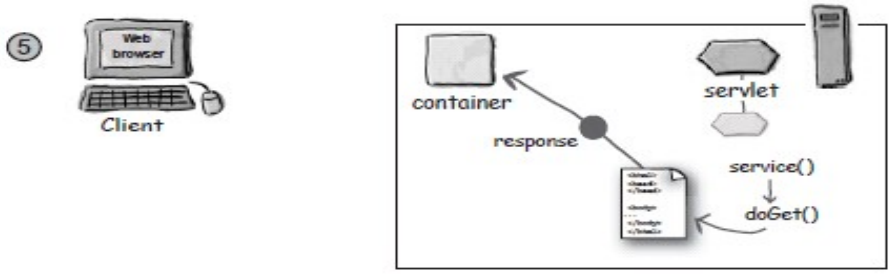

The container finds the correct servlet based on the URL in the request, creates or allocates a thread for that request, and passes the request and response objects to the servlet thread.




The container calls the servlet's service() method. Depending on the type of request, the service() method calls either the doGet() or doPost() method.

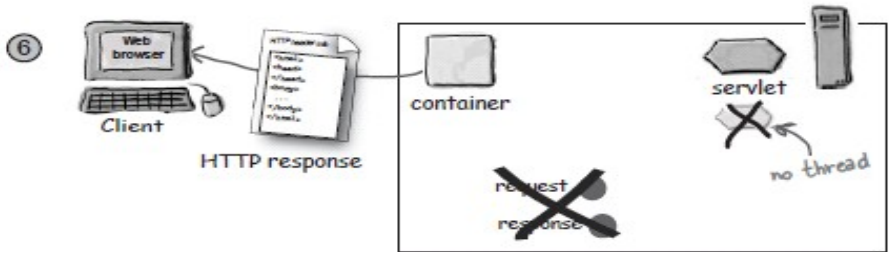
For this example, we'll assume the request was an HTTP GET.

---




The doGet() method generates the dynamic page and stuffs the page into the response object. Remember, the container still has a reference to the response object!

---




The thread completes, the container converts the response object into an HTTP response, sends it back to the client, then deletes the request and response objects.

**Servlet API**

Specification from sun microsystem that provide the way so that server can handle dynamic request.

Implementation provided by vendor; We are going to use tomcat 6.x/7.x that provide implemention of Servlet/JSP/JNDI API

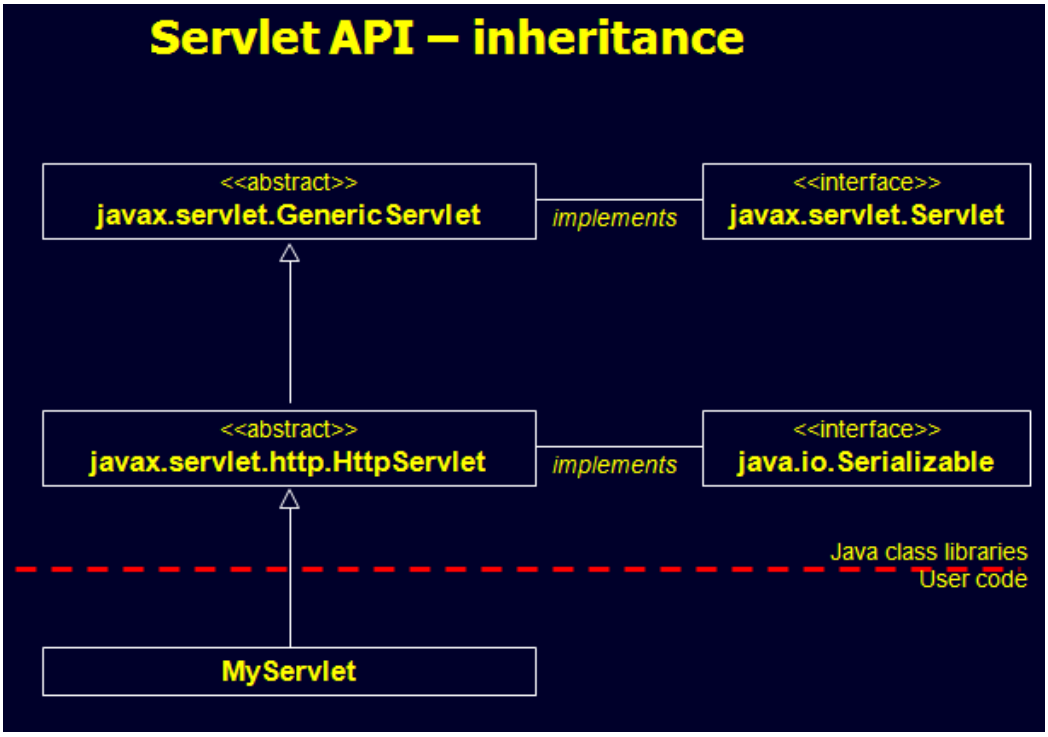
**Servlet API packages**

**Package *javax.servlet***

- Generic servlet handling
- Not specific to HTTP
  - Servlet architecture designed to cope with any kind of request/response protocol

**Package *javax.servlet.http***

- Specific for web servlets
- For:
  - format of HTTP request/responses
  - Support for HTTP sessions
  - Support for HTTP cookies




**What is an servlet?**

For creating a basic HTTP Java servlet, you must extends HttpServlet class and provide implementation to service() method

```
class MyServlet extends HttpServlet{
        public void service(……,……..)……
}
```

**<<Servlet>> interface**

**The Servlet Interface**

- specifes the contract between the web container and a servlet
- containers use this interface to reference servlets
- implement this indirectly by extending
    - javax.servlet.GenericServlet or javax.servlet.http.HTTPServlet

- Methods
- **void init(ServletConfig config)**
    - called with ServletConfig parameter by the container
    - container calls this before any request - guaranteed
    - allows a servlet to load any initialization of parameters
    - done once only / not per request

- **void service(ServletRequest req, ServletResponse res)**
    - entry point for executing logic
- **void destroy()**
    - container calls this before removing a servlet
    - deallocate resources (specially non Java resources)
- **ServletConfig getServletConfig()**
    - return the ServletConfig that was passed to init()
- **String getServletInfo()**
    - return a String containing servlet information

**javax.servlet.http.HttpServlet**

provides a default implementation of the service() method
- casts request / response to HTTP request / response
- calls its protected service() method
- service() uses get*Method*() to call do*XXX*() method
-

- It makes calls to other methods, which **YOU** implement. The main two are:
  - **doGet**() – HTTP GET requests
  - **doPost**() – HTTP POST requests

Better not to call service() method ourself; give chance to container …

**Requests and responses**

When the container calls service() it passes two objects as parameters, of types:
- HttpServletRequest (extends ServletRequest)
- HttpServletResponse (extends ServletResponse)

***ServletRequest common methods***

- **int getContentLength()**
  - number of bytes in the body of a request
- **String getParameter(String name)**
  - value correspond to name or null
- **Enumeration getParameterNames()**
  - names of parameters parsed out of the request
- **String[] getParameterValues(String name)**
  - for more than value associated with this name
  - null if none associated

**Cookie[] getCookies()**
- array of Cookie objects stored on the client

- **String getQueryString()**
  - query string present in the URL of GET request
- **HttpSession getSession()**
  - HTTPSession associated with this session
- **String getHeader(String name)**
  - value associated with the name; could be null
- **Enumeration getHeaderNames()**
  - an Enumeration for header names

**ServletResponse methods**

- **void setContentType(String MIMEtype)**
  - eg: text/html
- **PrintWriter getWriter()**

- **ServletOutputStream getOutputStream()**

- Difference between Writer and OutputStream??
  - Readers & Writers are for TEXT data
  - InputStreams & OutputStreams are for BINARY data
- Thus use:
  - getWriter() for Content-Type text/*  (e.g. text/html)
  - getOutputStream() for binary (e.g. image/gif)

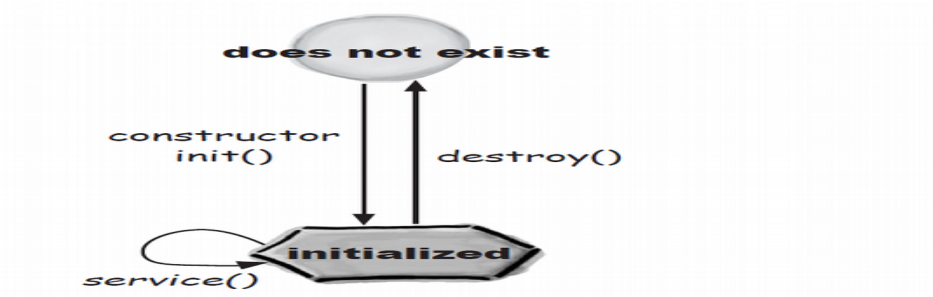
- **void setHeader(String name, String value)**
  - modify a value for name in the response
  - **void sendRedirect(String location)**
  - redirects the user's browser

- **String encodeURL(String url)**
  - see Session

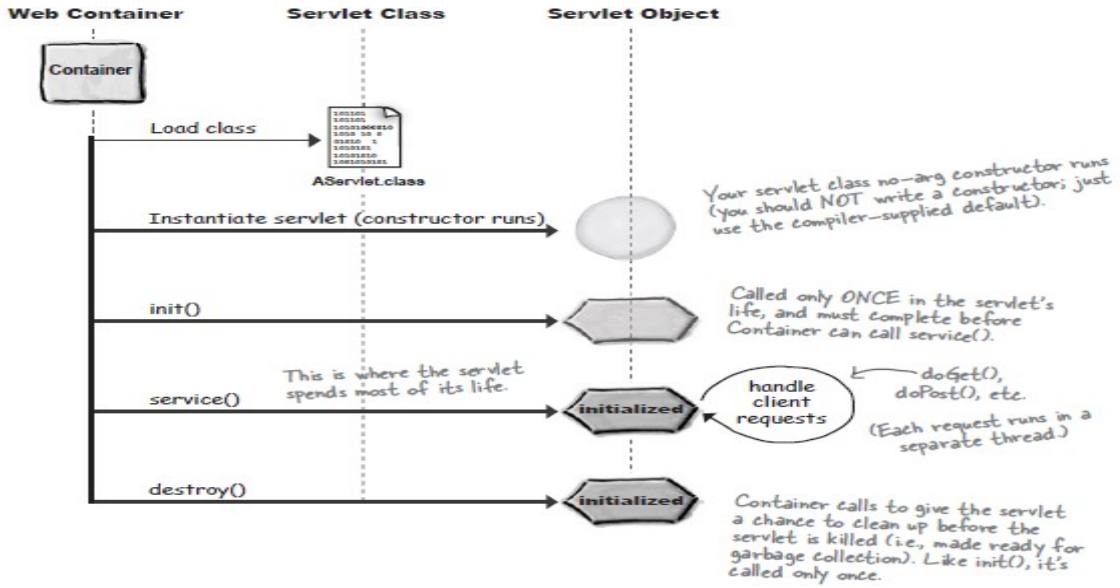
**void setError(int sc)**
  - sends default error page indicating the staus code (dreaded 404)

**void addCookie(Cookie cookie)**
  - adds a Cookie to the response

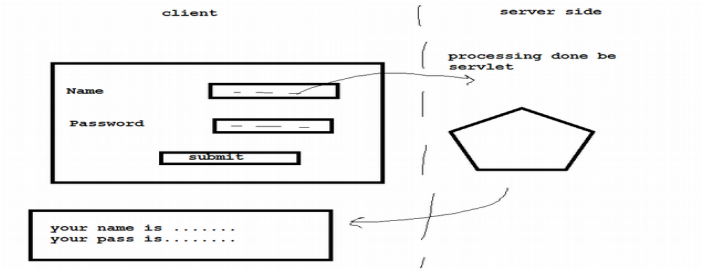
**Servlet Life Cycle**




**In detail**



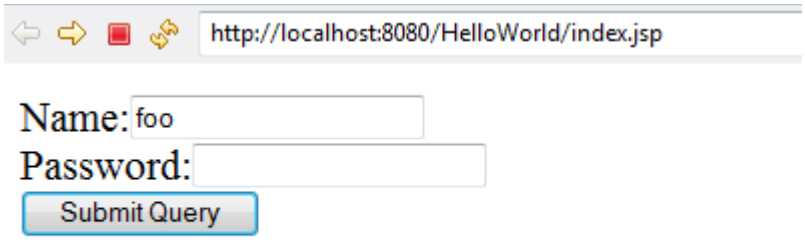

**Hello World**




User enter his name and password servlet process the data and show result to the user…….

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859
<title>Insert title here</title>
</head>
<body>
<form action="Hello" method="get">
    Name:<input type="text" name="name"/><br/>
    Password:<input type="password" name="pass"/><br/>
    <input type="submit"/>

</form>
</body>
</html>
```




```java
public class MyServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;


    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        // TODO Auto-generated method stub

        response.setContentType("text/html");
        String name=request.getParameter("name");
        String pass=request.getParameter("pass");
        PrintWriter out=response.getWriter();
        out.print("name:"+name);
        out.print("password:"+pass);
        System.out.println("i will be printed on console of server not to the client browser......");

    }
```

Most importantly an mapping file that glue it all togather
Web.xml
Also called Deployment descriptor

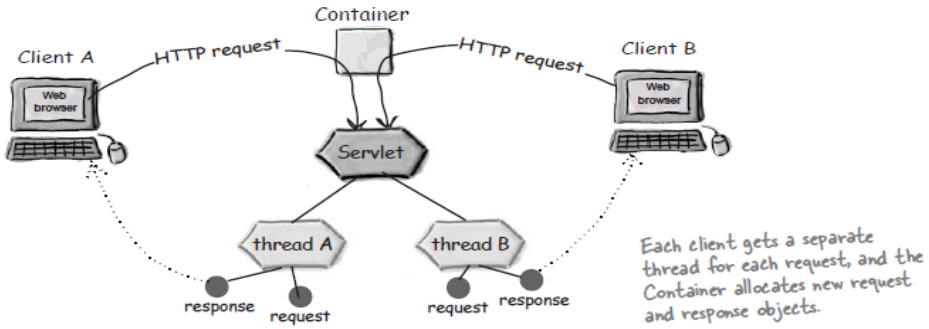
```xml
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  "http://java.sun.com/xml/ns/javaee" xmlns:web="http:/
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  id="WebApp_ID" version="2.5">
    <display-name>HelloWorld</display-name>
    <welcome-file-list>
      <welcome-file>index.html</welcome-file>
      <welcome-file>index.htm</welcome-file>
      <welcome-file>index.jsp</welcome-file>
      <welcome-file>default.html</welcome-file>
      <welcome-file>default.htm</welcome-file>
      <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
    <servlet>
      <description></description>
      <display-name>MyServlet</display-name>
      <servlet-name>MyServlet</servlet-name>
      <servlet-class>com.MyServlet</servlet-class>
    </servlet>
    <servlet-mapping>
      <servlet-name>MyServlet</servlet-name>
      <url-pattern>/Hello</url-pattern>
    </servlet-mapping>
</web-app>
```

Container refer web.xml when user submit index.jsp as it contain url pattern “Hello”
It find in web.xml and container map it to servlet class ie MyServlet.class in com package
…

Then servlet life cycle starts….

**Single thread model**

Only one instance of servlet is going to serves n request ….




Each request runs in a separate thread!

If you want that each user request must create an new servlet instance….then you implement an marker inteface SinglethreadModel……

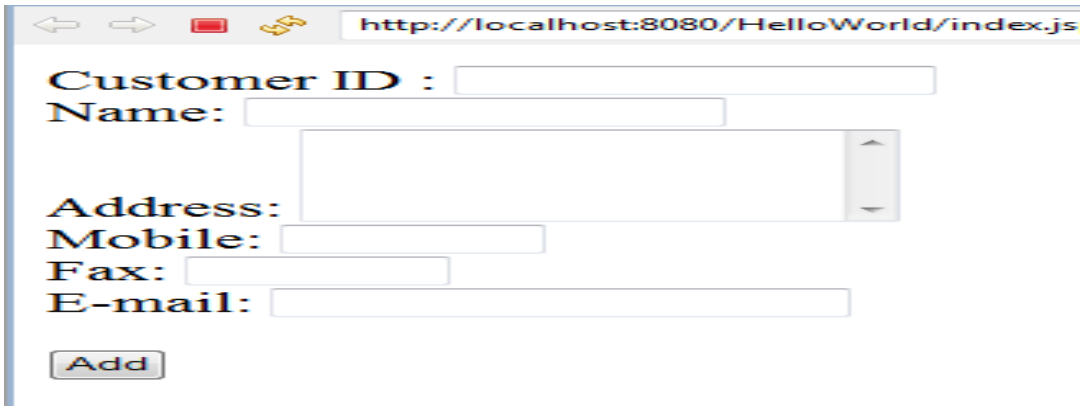
```
public class MyServlet extends HttpServlet implements SingleThreadModel {
    private static final long serialVersionUID = 1L;
```

But don't use it deprecated……

Technically there is no advantage of SingleThreadModel

**customer form processing ex**

```
<form action="addCustomer.do" method="POST">
        Customer ID : <input name="id" /> <br>
        Name: <input name="name" /> <br>
        Address: <textarea name="addr" rows="4" cols="20"></textarea> <br>
        Mobile: <input name="mobile" size="9" /> <br>
        Fax: <input name="fax" size="9" /> <br>
        E-mail: <input name="email" size="25" /> <br><br>
<input type="submit" value="Add" />
</form>
```



servlet 3.x Lab
---

Agenda:
_____

0. Hello World, port problem

1. Print client information

2. Demostration of servlet life cycle

3. configure username / password for tomcat

4. Simple form

5. ServletContext vs. ServletConfig

6. Servlet 3.0 annotations

7. Asynch processing

8. Simple login application: Dispatching VS redirecting

9. Demo attributes: application, request and session

10. MVC

11.CRUD application Storing book info in database

12.Session management

13. Listners: ServletContextListner, HttpSessionListner

14. Servlet Filter

15. Case Study CRUD application

16. Servlet security

0. hello world
---
With xml
With annotation

port problem
---

netstat -a -o -n
taskkill /F /PID 4036

1. Print client information
---

```
        // Get client's IP address
    String ipAddress = request.getRemoteAddr(); // ip

    // Get client's hostname
    String hostname = request.getRemoteHost(); // hostname
```

2. Demostration of servlet life cycle

override init() and destroy()

---

3. configure username / password for tomcat

---

create an customer record processing application:

1. accept name, address, mobile,email and favourite programming language
2. add record to db
3. use all discussed design pattern and follow coding practice

    =>MVC

    => putting jsp in webcontent X, should make them sec

    => connection object should be put in lister(servletcontext lister)

    => factory, singleton for connection factory

    => dao and dto

    => exception wrapping and rethowing

4. take it as a exam...

    marker will be offered and will tell u why ur marks are deduced

4. Simple form

---

creating customer form and Printing customer information

step 1:

create form:

---

```
<form action="addCustomer.do" method="POST">
Customer ID : <input name="id" /> <br>
Name: <input name="name" /> <br>
Address: <textarea name="addr" rows="4" cols="20"></textarea> <br>
Mobile: <input name="mobile" size="9" /> <br>
Fax: <input name="fax" size="9" /> <br>
E-mail: <input name="email" size="25" /> <br><br>
```

```
<input type="submit" value="Add" />
</form>
```

create servlet to process form:
---

```
.....
......

//retrive info

String id = request.getParameter("id");
String name = request.getParameter("name");
String addr = request.getParameter("addr");
String mobile = request.getParameter("mobile");
String fax = request.getParameter("fax");
String email = request.getParameter("email");

//Display customer informations

out.println("<h1> Customer Information </h1>");
out.println("<b>ID: </b>" + id + "<BR>");
out.println("<b>Name: </b>" + name + "<BR>");
out.println("<b>Address: </b>" + addr + "<BR>");
out.println("<b>Mobile: </b>" + mobile + "<BR>");
out.println("<b>Fax: </b>" + fax + "<BR>");
out.println("<b>E-mail: </b>" + email + "<BR>");
```

Form processing: Careful!
---

Accepting i/p form checkboxes /multiple select boxes
---

==> Use HttpServletRequest.getParameterValues() method

Ex;

```
<td><input name="options" type="checkbox" value="option1" />
<td><input name="options" type="checkbox" value="option2" />
<td><input name="options" type="checkbox" value="option3" />
```

In servlet:

```
String[] selectedOptions = request.getParameterValues("options");

if (selectedOptions != null)
{
        for (String option : selectedOptions)
                {
                  printWriter.print(option+"<br/>");
```

```
                }
        }
```

5. ServletContext vs. ServletConfig
---

ServletContext:
per applications

ServletConfig:
per servlet

8. Simple login application: Dispatching VS redirecting
---

9. Demo attributes: application, request and session
---

10. MVC
---

Application 1: Calculator application

form:

```
<form action="Cal.do" method="post">
        Enter first No : <input type="text" name="numberA"/><br/>
        Enter second No:<input type="text" name="numberB"/><br/><br/>
        <input type="submit"/>
</form>
```

Application 2: Book Advice Application

Step 1:

create view:
---

create form:

```
        <html><body>
        <h1 align="center">Book Selection Page</h1>
        <form action="SelectBook" method="post">
        Select book <p>

        Book:
        <select name="topic" size="1">
        <option value="Java">Java</option>
```

```
<option value="Servlet">Servlet</option>
<option value="Struts">Struts</option>

</select>
<br><br>
<center>
<input type="submit">
</center>
</form>
</body>
</html>
```

Create controller
---

```
String topic=request.getParameter("topic");
List<String>choices=BookAdviser.bookAdviser(topic);
request.setAttribute("booklist", choices);
RequestDispatcher rd=request.getRequestDispatcher("show2.jsp");
rd.forward(request, response);
```

create model
---

```
public class BookAdviser {

public static List<String> bookAdviser(String topic){
	List<String>list=new ArrayList<String>();

	if(topic.equalsIgnoreCase("Java")){
		list.add("head first");
		list.add("thinking in java");
	}else if(topic.equalsIgnoreCase("Servlet")){
		list.add("head first servlet jsp");
		list.add("core servlet.com");
	}else if(topic.equalsIgnoreCase("Struts")){
		list.add("struts2 in action");
		list.add("black book");
	}else
		list.add("no book");

	return list;
}
}
```

create display.jsp
---

```
<%
List<String>list=(List<String>)request.getAttribute("key");
Iterator it=list.iterator();
while(it.hasNext()){
```

```
		out.print(it.next()+"</br>");
	}

	%>
```

better view
---

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<c:forEach var="book" items="${booklist}">
		<b> ${book} </b><br/>
</c:forEach>
```

11. CRUD application Storing book info in database
---

Step 1: Add jdbc driver jar to project

Step 2: create mysql table book

step 1:
---

```
create table books(id int not null auto_increment, isbn varchar(20) not null, title varchar(40) not null, author varchar(80) not null,
pubDate date not null, price double not null, primary key (id), unique key (isbn));

populate some records
insert into books(isbn, title, author, pubDate, price ) values ('12PZ', 'C basics','ekta','1011-12-22',345);
```

Step 3:
Add jar to tomcat lib folder

Step 4:
create an form

```
<form action="bookController" method="post">
<input type="text" name="id" ><br/>
Enter  isbn : <input type="text" name="isbn" /><br/>
Enter title : <input type="text" name="title" /><br/>
Enter author: <input type="text" name="author" /><br/>
Enter pubDate: <input type="text" name="pubDate" /><br/>
Enter price:   <input type="text" name="price" /><br/>
<input type="submit"/>


</form>
```

Step 5:

create processing servlet:

i) load the driver in init() method and create an con object

```
con=DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/raj","root","root");
```

ii) put insertion code inside doPost()

```
String isbn = request.getParameter("isbn");
String author = request.getParameter("author");
String title = request.getParameter("title");
String priceStr = request.getParameter("price");

pstmt=con.prepareStatement("insert into books(isbn, author, title, price)values    (?,?,?,?)");

pstmt.setString(1, isbn);
pstmt.setString(2, author);
pstmt.setString(3, title);
pstmt.setFloat(4, price);
pstmt.executeUpdate();
```

9.1: DB best practices:
---

=> reading through property file
=> Connection factory

db.properties
---
```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/springexp
jdbc.username=root
jdbc.password=root
```

reading property file:
---
```
Properties prop = new Properties();
InputStream inputStream = DbUtil.class.getClassLoader().
getResourceAsStream("/db.properties");
prop.load(inputStream);
String driver = prop.getProperty("driver");
String url = prop.getProperty("url");
```

```
String user = prop.getProperty("user");
String password = prop.getProperty("password");
Class.forName(driver);
connection = DriverManager.getConnection(url, user, password);
```

what we did today?

==========

life cycle again
context vs config ( delhi => ITO)
ReqeustDis vs Redirect
mvc design pattern: bookadv app
servlet chaining

Job of controller
Dao dto with jdbc property file

MVC, Singleton, factory, DAO,DTO, GPP to keep jsp secure

bookapp

9.2: DB best practices:tomcat connection pooling

---

Step 1;

mapping in context.xml

```
<Resource
name="jdbc/test"
auth="Container"
driverClassName="com.mysql.jdbc.Driver"
type="javax.sql.DataSource"
url="jdbc:mysql://localhost:3306/exp121"
username="root"
password="root" >
</Resource>
```

Step 2:

mapping in web.xml

---

```
<resource-ref>
  <description>Test Database</description>
 <res-ref-name>jdbc/test</res-ref-name>
 <res-type>javax.sql.DataSource</res-type>
 <res-auth>Container</res-auth>
  </resource-ref>
```

Step 3: pull vs push

---

```
@Resource(name="jdbc/test")
private DataSource ds;
private Connection conn;


conn = ds.getConnection();
```

then use connection object as usual;

Optional : if u want to have a try :)

---

```
Context initContext = new InitialContext();
Context envContext  = (Context)initContext.lookup("java:/comp/env");
DataSource ds = (DataSource)envContext.lookup("jdbc/myoracle");
Connection conn = ds.getConnection();
```

12.Session management

---

cookies, HttpSession, Url-rewriting

HttpSession object

---

simple program to check session...

```
HttpSession session=request.getSession();
        String heading="hello";

        synchronized (session) {
            Integer count=(Integer)session.getAttribute("sessioncount");
            if (count==null)
            {
                count=0;
                session.setAttribute("sessioncount", count);
                heading="welcome back first time user";

            }
            else
            {
                count=count+1;
                session.setAttribute("sessioncount", count);
                heading="welcome back "+count;
            }
        }
        PrintWriter out=response.getWriter();
        out.print(heading);
```

login/ logout

---

using httpsession

```
<form action="login.do" method="post">
        Enter name : <input type="text" name="name"/><br/>
        Enter password : <input type="password" name="password"/><br/>
        <input type="submit"/>
</form>
```

Session Mgt using Cookies object

---

set the cookies

---

```
Cookie cookie= new Cookie("key", name);
cookie.setMaxAge(30*60);
response.addCookie(cookie);
```

get the cookie

---

```
Cookie[]cookies=request.getCookies();

if(cookies!=null){
    for(int i=0;i<cookies.length;i++){
        Cookie cookie=cookies[i];
        if(cookie.getName().equals("key"))
        {
            String uname=cookie.getValue();
            out.print(uname);
            break;
        }
    }
```

Url rewriting

---

```
// Encodes the specified URL by including the session ID in it,
// or, if encoding is not needed, returns the URL unchanged

String newURL = response.encodeURL("GetSession");

// Return a <a> tag with the new url

writer.println("Click <a href=\"" + newURL + "\">here</a> for another servlet");
```

13. Listners: ServletContextListner, HttpSessionListner

---

14. Servlet Filter

---

```
public class LoginFilter implements Filter {

public void init(FilterConfig fConfig) throws ServletException {
	// TODO Auto-generated method stub
}

public void doFilter(ServletRequest request, ServletResponse response,
		FilterChain chain) throws IOException, ServletException
{

	HttpServletRequest req = (HttpServletRequest) request;
	HttpSession session = req.getSession();

	Boolean flag = (Boolean) session.getAttribute("loginFlag");

	boolean loginFlag;
	if (flag == null)
	{
		loginFlag = false;
	}
	else
	{
		loginFlag = flag;
	}

	if (!loginFlag)
	{
		HttpServletResponse res = (HttpServletResponse) response;
		res.sendError(HttpServletResponse.SC_UNAUTHORIZED,"login failed!!!");
		return;
	}

	chain.doFilter(request, response);
}



public void destroy() {
	// TODO Auto-generated method stub
}

}
```

Extracting part of uri

---

```
	String uri = request.getRequestURI();
	int lastIndex = uri.lastIndexOf("/");
	String action = uri.substring(lastIndex + 1);
```

```
                    <dispatcher>REQUEST</dispatcher>
                    <dispatcher>FORWARD</dispatcher>
                    <dispatcher>INCLUDE</dispatcher>
                    <dispatcher>ERROR</dispatcher>
```

15. Case Study CRUD application
---

---
tomcat basic security
---

mentions users in tomcat-users.xml

```
<role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="employee"/>
  <user username="concretepage" password="concretepage" roles="employee"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
```

Define in web.xml of project:
---

```
<!--Defines Security Constraint -->
   <security-constraint>
      <display-name>JSP Demo Constraint</display-name>
      <web-resource-collection>
         <web-resource-name>cp</web-resource-name>
         <description/>
         <url-pattern>/*</url-pattern>
      </web-resource-collection>
      <auth-constraint>
         <description/>
         <role-name>employee</role-name>
      </auth-constraint>
   </security-constraint>
```

```
<!--Defines Login Config -->
   <login-config>
      <auth-method>FORM</auth-method>
      <realm-name>file</realm-name>
      <form-login-config>
         <form-login-page>/login.jsp</form-login-page>
         <form-error-page>/error.jsp</form-error-page>
```

```
        </form-login-config>
    </login-config>



<!--Defines Security Role -->
    <security-role>
        <description/>
        <role-name>employee</role-name>
    </security-role>
```

login.jsp

```
<form name="loginForm" method="POST" action="j_security_check">
            <p>User name: <input type="text" name="j_username" size="20"/></p>
            <p>Password: <input type="password" size="20" name="j_password"/></p>
            <p>  <input type="submit" value="Submit"/></p>
        </form>
```

logout.jsp

```
<%
session.invalidate();
response.sendRedirect("index.jsp");
%>
```

error.jsp

```
 <h3>Login Error</h3>
       <a href="index.jsp">Click to Login Again</a>
```

index.jsp

```
<h1>You have successfully logged-in</h1>
<a href="logout.jsp" >Click to Logout </a>
```

maven dependencies:

```
----------------------
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
```

6. New features introduced in Servlet 3.0
---

==> Optional web.xml

==> Annotations supports

==> Configuring web applications programmatically

==> Asynchronous processing

@WebServlet annotation
==========

Ex:

```
@WebServlet(urlPatterns = {"/simpleservlet"})
public class SimpleServlet extends HttpServlet
{
        .....
        .....
}
```

```
@WebServlet(urlPatterns = {"/simpleservlet", "*.foo"})
```

Passing initialization parameters to a servlet via annotations

```
@WebServlet(name = "InitParamsServlet", urlPatterns = {
        "/InitParamsServlet"}, initParams = {
        @WebInitParam(name = "param1", value = "value1"),
        @WebInitParam(name = "param2", value = "value2")})
```

@WebFilter annotation
==========

```
@WebFilter(filterName = "SimpleFilter", initParams = {
@WebInitParam(name = "filterparam1", value = "filtervalue1")},
urlPatterns = {"/InitParamsServlet"})

public class SimpleFilter implements Filter
{
        .....
```

```
	.....
}
```

@WebListener annotation
==========

```
@WebListener()
public class HttpRequestListener implements ServletRequestListener
{
	.....
	.....
}
```

Asynchronous processing
==========

Why it required?
---

==> Ajax has the side effect of generating a lot more HTTP
requests than traditional web applications.

==> If some of these threads block for a long
time waiting for a resource to be ready or are doing anything
that takes a long time to process, it is possible our application
may suffer from thread starvation.

==> Servlet 3.0 introduced asynchronous processing

==> Using this new capability, we are no longer limited to a single
thread per request. We can now spawn a separate thread and return the
original thread back to the pool to be reused by other clients

```
@WebServlet(name = "AsynchronousServlet", urlPatterns = {"/AsynchronousServlet"}, asyncSupported = true)

public class AsynchronousServlet extends HttpServlet
{
	@Override
	protected void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
	{

		final AsyncContext ac = request.startAsync();


		ac.start(new Runnable(){
				@Override
				public void run(){
					Sysout("inside thread");

					try
```

```
                                                {
                                                        //simulate a long running process.
                                                        Thread.sleep(10000);
                                                }
                                                catch (InterruptedException ex)
                                                {
                                                        Sysout("erroer!");
                                                }

                                        try
                                         {
                                                ac.getResponse().getWriter().println("You should see this after a
                                                brief wait");

                                                ac.complete();
                                        }
                                                catch (IOException ex)
                                                {

                                                }
                                        }
                                });
                }

        }
```

how i can persist session to db?
---
https://www.youtube.com/watch?v=kN_DVkwBxyY

get vs post in details
https://www.diffen.com/difference/GET-vs-POST-HTTP-Requests
http://www.pearsonitcertification.com/articles/article.aspx?p=30082&seqNum=3

---

https://danielniko.wordpress.com/2012/12/03/simple-crud-using-java-hibernate-and-mysql/

Java server pages 2.3
xxxxxxxxxxxxxxxxx

JSP (Its all about how to
display data in best way!!!)
==========

=>Nut and bolts of JSP
-> Directive,Scriptlet,Expression

=>JSP Life Cycle

=>Understanding JSP via servlet

=>Implicit Objects

=>Actions

=>Database Interaction

=>Calling a JavaBean from a JSP page

=>Expression Langauage

=>JSP Standard Tag Library

=>Custom Tags

JSP first look:
---
directive
---
<%@ page import="java.util.*" %>

declaration
---
<%! int y = 3; %>

EL Expression
---
email: ${applicationScope.mail}

scriptlet
---
<% Float one = new Float(42.5); %>

expression
---
<%= pageContext.getAttribute(foo") %>

action
---
<jsp:include page="foo.html" />

=>Nut and bolts of JSP
-> Directive,Scriptlet,Expression
_____

diective
---
decide behaviour of whole page

<%@ ..... %>

3 kind of directive: page, include, taglib direction
_____

<%@ page import="java.util.*, java.sql.*" %>

for page directive 13 kind

Scriptlet :its pure java code! u can write html code inside it
---
whatever i want to write in doGet/doPost of a servlet

```
<%

		sop("hi");

		i++;
		// out predefine object
		out.println(i);

%>
```

expresssion
---

```
	<%= i  %>
```

	out.prinln(i);

	expression is an replacement
	(Shortcut of out.println() )

decleration
---

```
	<%!	%>
```

JSP Comments
---

	HTML Comment		<!-- HTML Comment -->

	JSP Comment	<%-- JSP Comment --%>

decleration tag!
---

```
<%!
```

```
		int i=0;// instance
		static int counter =0;

		public void foo(){

		}
	%>
```

=> Understanding JSP via servlet

---

how to learn jsp easily ?

u should think jsp form sevlet prospective!

```
	import java.sql.*;
	import java.util.*;

	class MySevlet extends HttpServlet
	{
		int i=0;// instance
		static int counter =0;

		public void foo(){

		}

		public void doGet(.....,.....)
		{
			sop("hi");

			i++;
			PrintWriter out=res.get...();
			out.println(i);
		}


	}
```

```
	<%!int counter=0;%>
	<%=counter++%>
```

=> JSP directive in details

---

```
	<%@ .... %>
```

page directive
---

defines page-specific properties such as character encoding,
the content type for this pages response and whether this
page should have the implicit session object.

<%@ page import="foo.*" session="false" %>

<%@ page language="java" import="java.util.Date(),java.util.Dateformate()" iserrorpage="false"%>

A page directive can use up to thirteen different attributes *(At the end)

include directive
---

Defines text and code that gets added
into the current page at translation time

<%@ include file="hi.html" %>

web page

header

footer

include action vs include directive?
---

taglib directive * how to define customer tag?
---

Defines tag libraries available to the JSP

<%@ taglib tagdir="/WEB-INF/tags/cool" prefix="cool" %>

what is the syntax of include directive
---

a.jsp
---

```
<h1>file 1</h2>
<%@ include file="b.jsp" %>
```

b.jsp
---

```
<h1>file 2</h2>
<h1>india  is shining?</h1>
<%=new Date()%>
```

in theory
---
we should not use include directive if content of b.jsp
is changing with time

```
<%@ page %>
```

demo isError page and error page in JSP
---

a.jsp
---

```
<%@ page errorPage="b.jsp" isErrorPage="false"%>
<%

Dog d=null;

d.toString();

%>
```

b.jsp
---

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

This is the Error page.The following error occurs:- <br>
<%= exception.toString() %>
```

=> Implicit Object
---

| | |
|---|---|
| JspWriter | out |
| HttpServletRequest | request |

```
request.setAttribute("key","foo");
String temp=request.getAttribute("key");
```

| | |
|---|---|
| HttpServletResponse | response |
| HttpSession | session |
| | session.setAttribute("key","foo");<br>String temp=session.getAttribute("key"); |
| ServletContext | application |
| | application.setAttribute("key","foo");<br>String temp=application.getAttribute("key"); |
| ServletConfig | config |
| Throwable | exception |
| PageContext | pageContext (not in servlet)<br>is an handly way to access any type of scoped variable |
| Object | page (not in servlet) |

=> Scope in JSP

---

application
session
request
page

=> Standard Actions

---

Tags that affect runtime behavior of JSP and
response send back to client

Std action types:

---

<jsp:useBean>
<jsp:setProperty>
<jsp:getProperty>

<jsp:forward/>
<<jsp:include/>
....
.....

```
RequestDispacher rd=request.getRequestDispacher("show.jsp");
                rd.forward(req,res);
```

rd.include(req,res);

Equ code in JSP:

<jsp:include>
<jsp:forward>
(How to pass parameters in include and forward)

Simple login app with jsp only (bad code)
---

```
<form action ="myLogin.jsp".
        <input type="text" name="name"/>
        <input type="password" name="pass"/>
        <input type="submit"/>
</form>
```

```
<%
        if((request.getParameter("un").equals("raj")) &&(request.getParameter("pw").equals("java")))
        {
%>
        <jsp:forward page="forward2.jsp"/>
<%
}
        else
{
%>
        <%@include file="index.jsp"%>
<%

        }
%>
```

passing parameter with dispaching
---

```
   <jsp:include page="/foo2.jsp" >
                <jsp:param name="sessionID" value="<%= session.getId()      %>" />
   </jsp:include>
```

Sepration of concern
---
no business logic should be done in jsp at any cost

<jsp:useBean>
<jsp:setProperty>
<jsp:getProperty>

Example:
---

```
<form action ="LoginServlet" method="get">
        ID:<input type="text" name="id"/></br>
        Name:<input type="text" name="name"/></br>
        Pass:<input type="password" name="pass"/></br>
        <input type="submit"/>
</form>
```

automatic type conversion

processing of bean

API for the Generated Servlet
==========

jspInit()
---

This method is called from the
init() method and it can be overridden

jspDestroy()
---

This method is called from the servlets destroy()
method and it too can be overridden

_jspService()

This method is called from the servlets service()
method which means its runs
in a separate thread for each request,
the container passes the request and response
object to this method.

You cannot override this method.

Initializing your JSP
---

put this in web.xml
---

```
<web-app ...>

  <servlet>
        <servlet-name>foo</servlet-name>
        <jsp-file>/index.jsp</jsp-file>

 <init-param>
    <param-name>email</param-name>
    <param-value>rgupta.mtech@gmail.com</param-value>
  </init-param>

 </servlet>

 <servlet-mapping>
        <servlet-name>foo</servlet-name>
        <url-pattern>/index.jsp</url-pattern>
 </servlet-mapping>

</web-app>
```

now getting them in init method
==========

```
<%!
 public void jspInit()
 {

   ServletConfig sConfig = getServletConfig();
   String emailAddr = sConfig.getInitParameter("email");
   ServletContext ctx = getServletContext();
   ctx.setAttribute("mail", emailAddr);
 }

%>
```

now get attributes in service method
==========

```
<%= "Mail Attribute is: " + application.getAttribute("mail") %>


<%= "Mail Attribute is: " + pageContext.findAttribute("mail") %>


<%
  ServletConfig sConfig = getServletConfig();
  String emailAddr = sConfig.getInitParameter("email");
  out.println("<br><br>Another way to get web.xml attributes: " + emailAddr );
%>




<%
```

```
   out.println("<br><br>Yet another way to get web.xml attributes: " + getServletConfig().getInitParameter("email") );
%>
```

Setting scoped attributes in JSP
==========

Application
---

in servlet
---
getServletContext().setAttribute("foo",barObj);

in jsp
---
application.setAttribute("foo",barObj);

Request
---

in servlet
---
request.setAttribute("foo",barObj);

in jsp
---
request.setAttribute("foo",barObj);

Session
---

in servlet
---
request.getSession().setAttribute("foo",barObj);

in jsp
---
session.setAttribute("foo",barObj);

Page
---

in servlet
---
do not apply

in jsp
---
pageContext.setAttribute("foo",barObj);

Note
==========

Using the pageContext to get a session-scoped attribute
---

<%= pageContext.getAttribute("foo", PageContext.SESSION_SCOPE ) %>

Using the pageContext to get an application-scoped attribute
---
<%= pageContext.getAttribut("mail",PageContext.APPLICATION_SCOPE) %>

Using the pageContext to find an attribute
when you don't know the scope
---
<%= pageContext.findAttribute("mail") %>

=> Java bean in jsp
_____
GPP: dont use scriplet!
if i am using scriplet ie i am adding my logic jsp X
EL
JSTL

=> EL: Expression language
_____

Putting java code in jsp is bad habbit

Scriplet in ur project u may be gone!

then what to do?
---

Use EL
JSP 2.0 spec. EL offers a simpler way to
invoke Java code but code itself belongs somewhere else

Although EL looks like Java it behaves differently,
so do not try and map the same Java with EL.

EL are always within curly braces
and prefixed with the dollar sign.

The first named variable in the expression is either an
implicit object or an attribute

how EL make my life easy....
---

EL example
---
old way don't do now
---

Please contact: <%= application.getAttribute("mail") %>

EL way
---

please contact: ${applicationScope.mail}

stop JSP from using scripting elements
---

```
<web-app ...>
...
  <jsp-config>
    <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <scripting-invalid>true</scripting-invalid>
    </jsp-property-group>
  </jsp-config>
...
</web-app>
```

JSTL  is to EL

as PL/SQL to SQL

stop using EL
---
<%@ page isELIgnored="true" %>

Note: this takes priority over the DD tag above

JSTL

JSTL :jsp std tag lib

As PL/SQL is to SQL
JSTL is to EL

Scriptless JSP
==========
why we should not use scriplet?

EL provide better way to accept DTO send from controller to view

Some Ex:
======

Consiser Servlet (controller) code
---

```
Person p = new Person();

p.setName("Paul");

request.setAttribute("person", p);

RequestDispatcher view = request.getRequestDispatcher("result.jsp");

view.forward(request, response);
```

JSP (view) code
---

```
Person is: <%= request.getAttribute("person") %>

Does it work?
```

Correct way?
---
<% Person p = (Person) request.getAttribute("person");

Person is: <%= p.getName() %>

or

Person is: <%= ((Person) request.getAttribute("person")).getName() %>

Correct Way
---

<jsp:useBean id="person" class="foo.Person" scope="request" />

Person is: <jsp:getProperty name="person" property="name" />

```
class Person
{
	private String name;
	....
	...
	...
}

public class Dog
{
	private String dogName;
	.....
	.....
}

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

	.....
	.....
}
```

Person has A dog
---

```
Dog dog=new Dog();

dog.setDogName("myDog");

Person p=new Person();

p.setPersonName("foo");

p.setDog(dog);

request.setAttribute("person", p);
```

Expression Language More examples
---

consider controller code
---

adding persons dog in request attributes in an servlet
---

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

p.setName("Paul");

foo.Dog dog = new foo.Dog();

dog.setName("Spike");

p.setDog(dog);

request.setAttribute("person", p);
```

getting same in jsp
---

using tags

```
<%= ((Person) request.getAttribute("person")).getDog().getName() %>

Dog name: ${person.dog.dogName}
```

Some more examples
---

in servlet
---

String[] footballTeams = { "Liverpool", "Manchester Utd", "Arsenal", "Chelsea" }

request.setAttribute("footballList", footballTeams);

in jsp
---

Favorite Team: ${footballList[0]}

Worst Team: ${footballList["1"]}

Note ["one"] would not work but ["10"] would

<%-- using the arraylist toString()
---

All the teams: ${footballList}

Another Example:EL
---

java.util.Map foodMap = new java.util.HashMap();

foodMap.put("Fruit", "Banana");
foodMap.put("TakeAway", "Indian");
foodMap.put("Drink", "Larger");
foodMap.put("Dessert", "IceCream");
foodMap.put("HotDrink", "Coffee");

String[] foodTypes = {"Fruit", "TakeAway", "Drink", "Dessert", "HotDrink"}

request.setAttribute("foodMap", foodMap);
request.setAttribute("foodTypes",foodTypes);

JSP code
---

Favorite Hot Drink is: ${foodMap.HotDrink}

Favorite Take-Away is: ${foodMap["TakeAway"]}

Favorite Dessert is: ${foodMap[foodTypes[3]]}

SQL => PL/SQL

EL => JSTL

=> JSTL (JSP std tag library)

---

The JSTL is hugh, version 1.2 has five libraries,
four with custom tags and one with a bunch of functions for String manipulation

JSTL Core - core  c

JSTL fmt - formatting  fmt

JSTL xml- xml  xmt

JSTL sql - sql

JSTL function - string manipulation

standard.jar, jstl.jar

Hello world jstl

---

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html><body>
...
 <c:forEach var="i" begin="1" end="10" >
            <c:out value="${i}" />
 </c:forEach>
 ...
</body></html>
```

Example with map
---

```
Map<Integer,String>map=new HashMap<Integer, String>();
map.put(22, "foo");
map.put(44, "bar");
map.put(55, "jar");
map.put(88, "war");
request.setAttribute("map", map);
RequestDispatcher rd=request.getRequestDispatcher("show2.jsp");
rd.forward(request,response);
```

Now in JSP
---

```
<c:forEach var="i" items="${map}">
    key: ${i.key }-value: ${i.value}<br>
</c:forEach>
```

Step 1:

create view:
---

create form:

```
<html><body>
<h1 align="center">Book Selection Page</h1>
<form action="SelectBook" method="post">
Select book <p>

Book:
<select name="topic" size="1">
<option value="Java">Java</option>
<option value="Servlet">Servlet</option>
<option value="Struts">Struts</option>

</select>
<br><br>
<center>
<input type="submit">
</center>
</form>
</body>
</html>
```

Create controller
---

```
String topic=request.getParameter("topic");
```

```
List<String>choices=BookAdviser.bookAdviser(topic);
request.setAttribute("booklist", choices);
RequestDispatcher rd=request.getRequestDispatcher("show2.jsp");
rd.forward(request, response);
```

create model

---

```
public class BookAdviser {

public static List<String> bookAdviser(String topic){
	List<String>list=new ArrayList<String>();

	if(topic.equalsIgnoreCase("Java")){
		list.add("head first");
		list.add("thinking in java");
	}else if(topic.equalsIgnoreCase("Servlet")){
		list.add("head first servlet jsp");
		list.add("core servlet.com");
	}else if(topic.equalsIgnoreCase("Struts")){
		list.add("struts2 in action");
		list.add("black book");
	}else
		list.add("no book");

	return list;
}
```

view

---

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<c:forEach var="book" items="${booklist}">
	<b> ${book} </b><br/>
</c:forEach>
```

Now consider book:

---

id
title
categery
author
price

```
<body>
  <table border=1>
    <thead>
```

```
            <tr>Book Id</th>
             <th>title</th>
             <th>Last Name</th>
             <th>categery</th>
             <th>author</th>
                      <th>price</th>
          </tr>
       </thead>
       <tbody>
          <c:forEach items="${books}" var="book">
             <tr>
                <td><c:out value="${book.id}" /></td>
                <td><c:out value="${book.title}" /></td>
                         <td><c:out value="${book.categery}" /></td>
                <td><c:out value="${book.author}" /></td>
                          <td><c:out value="${book.price}" /></td>

             </tr>
          </c:forEach>
       </tbody>
    </table>
</body>
```

---

import

Defines the Java import statements that'll
be added to the generated servlet class

isThreadSafe

Defines whether the generated servlet needs to
implement the Single ThreadModel which as you know
know is a bad thing

contentType

Defines the MIME type for the JSP response (default is "text/html")

isELIgnored*

Defines whether EL expressions are ignored when this page is translated

EL*
--

Expression language

isErrorPage

Defines whether the current page
represents another JSPs error page

errorPage

Defines a URL to the resource to which
uncaught Throwables should be sent

language

Defines the scripting language used in the scriptlets,
expressions and declarations,

only java is available at the moment

extends
Defines the superclass of the class this JSP will become

session
Defines whether the page will have an implicit session object

buffer
Defines how buffering is handled by the implicit out object (reference to the jspWriter)

autoFlush
Defines whether the buffered output is flushed automatically

info
Defines a String that gets put into the translated page,
just so that you can get it using the generated servlets inherited getServletInfo() method

pageEncoding
Defines the character encoding for the JSP

---
custom tag library
---

custom tag that simply print "hi"

user defind tag!

<nu:hello/>

hello world
---
step 1:
create an simple tag handler SimpleHelloTag in com.tags package

```
public class SimpleHelloTag extends SimpleTagSupport {

  @Override
  public void doTag() throws JspException, IOException {
    JspWriter out = getJspContext().getOut();

    out.print("hi");
  }
}
```

step 2:
create hello taglib tld file
---

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

```
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
  version="2.0">
  <tlib-version>1.0</tlib-version>
  <short-name>hello-taglib</short-name>

  <tag>
    <description>hello world exmple</description>
    <name>hello</name>
    <tag-class>coreservlets.tags.SimpleHelloTag</tag-class>
    <body-content>empty</body-content>
  </tag>

</taglib>
```

step 3:
---

invoke from jsp;

```
<%@ taglib uri="/WEB-INF/tlds/hello-taglib.tld" prefix="hello" %>
```

tag

```
<hello:hello/>
```

custom tag library
---

hello world with attributes
---

step 1:
add following to tag defination

```
<attribute>
        <name>length</name>
```

```
        <required>false</required>
 </attribute>
```

step 2:

provide getter setter in corrosponding tag handler class

```
public class SimpleHelloTag extends SimpleTagSupport {
        private String length;

        public String getLength() {
                return length;
        }

        public void setLength(String length) {
                this.length = length;
        }
}
```

step 3:

then use it:

```
<hello:hello name="raj"/>
```

Ex 3:
including the tag body
---

rather then
```
<hello:hello length="raj"/>
```

Now want to have:

```
<hello:hello length="raj">
        this is an body example! (Scriptless jsp contents)
</hello:hello>
```

Step 1;

call getJsbBody().invoke(null) method from doTag()

```
	public void doTag() throws JspException, IOException {
		JspWriter out = getJspContext().getOut();

		out.print(length);

		getJspBody().invoke(null);

		out.print("hi");
	}
```

Step 2;

change body contents form TLD from empty to scriptless

Example:
---
custom tag with body:

step 1:

create an tag class

com.jsp.customtags packages

```
public class SubstrTagHandler extends TagSupport {
   private String input;
   private int start;
   private int end;

   @Override
   public int doStartTag() throws JspException {

      try {
         //Get the writer object for output.
         JspWriter out = pageContext.getOut();

         //Perform substr operation on string.
         out.println(input.substring(start, end));

      } catch (IOException e) {
         e.printStackTrace();
      }
```

```
        return SKIP_BODY;
    }
    public String getInput() {
        return input;
    }
    public void setInput(String input) {
        this.input = input;
    }
    public int getStart() {
        return start;
    }
    public void setStart(int start) {
        this.start = start;
    }
    public int getEnd() {
        return end;
    }
    public void setEnd(int end) {
        this.end = end;
    }
}
```

step 2:
create an tld file:

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib>
            <tlibversion>1.0</tlibversion>
            <jspversion>1.1</jspversion>
            <shortname>substr</shortname>
            <info>Sample taglib for Substr operation</info>
            <uri>http://rajsupport.net/blogs/jsp/taglib/substr</uri>
<tag>
    <name>substring</name>
    <tagclass>com.jsp.customtags.SubstrTagHandler</tagclass>
    <info>Substring function.</info>

    <attribute>
      <name>input</name>
      <required>true</required>
  </attribute>

    <attribute>
      <name>start</name>
      <required>true</required>
  </attribute>

    <attribute>
      <name>end</name>
      <required>true</required>
  </attribute>
```

```
</tag>
</taglib>
```

step 3:
use it in jsp

```
<%@taglib prefix="test" uri="/WEB-INF/SubstrDescriptor.tld"%>

   <test:substring input="GOODMORNING" start="1" end="6"/>
```

Dynamic attribute and Looping tags
---

tag example that support dynamic Attribute values:

```
<rtexprvalue>true</rtexprvalue>
```

```
public class ForTag extends SimpleTagSupport {
  private int count;

  public void setCount(int count) {
    this.count = count;
  }

  @Override
  public void doTag() throws JspException, IOException {
    for(int i=0; i<count; i++) {
      getJspBody().invoke(null);
    }
  }
}
```

tld
---

```
  <tag>
    <description>
      Loops specified number of times.
    </description>
    <name>for</name>
    <tag-class>com.tags.ForTag</tag-class>
```

```
    <body-content>scriptless</body-content>
    <attribute>
      <description>
        Number of times to repeat body.
      </description>
      <name>count</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>
```

Coin bean

```
public class CoinBean {
  public String getFlip() {
    if (Math.random() < 0.5) {
      return("Heads");
    } else {
      return("Tails");
    }
  }
}
```

Servlet

```
....
 CoinBean coin = new CoinBean();
    request.setAttribute("coin", coin);

    RequestDispatcher dispatcher =
      request.getRequestDispatcher(test-loop.jsp);
    dispatcher.forward(request, response);
```

testing

```
  <c:for count="<%=(int)(Math.random()*10)%>">
    <LI>Blah
  </c:for>
</UL>
<H2>Some Coin Flips</H2>
<UL>
  <c:for count="<%=(int)(Math.random()*10)%>">
    <LI>${coin.flip}
  </c:for>
```

Complex Dynamic attribute and Looping tags
---

What if u want type other then String or primitive type for an tag attribute value?

Attribute should accept an collection.

tag
---

```
public class ForEachTag extends SimpleTagSupport {
  private Object[] items;
  private String attributeName;

  public void setItems(Object[] items) {
    this.items = items;
  }

  public void setVar(String attributeName) {
    this.attributeName = attributeName;
  }

  @Override
  public void doTag() throws JspException, IOException {
    for(Object item: items) {
      getJspContext().setAttribute(attributeName, item);
      getJspBody().invoke(null);
    }
  }
}
```

```
<tag>
   <description>
     Loops down each element in an array
   </description>
   <name>forEach</name>
```

```
    <tag-class>com.tags.ForEachTag</tag-class>
    <body-content>scriptless</body-content>
    <attribute>
      <description>
        The array of elements.
      </description>
      <name>items</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
      <description>
        The name of the local variable that
        each entry will be assigned to.
      </description>
      <name>var</name>
      <required>true</required>
    </attribute>
  </tag>
```

Servlet
---

```
 String[] servers =
      {"Tomcat", "Resin", "Jetty", "WebLogic",
       "WebSphere", "JBoss", "Glassfish" };
    request.setAttribute("servers", servers);


    RequestDispatcher dispatcher =
      request.getRequestDispatcher(loop-test.jsp);
    dispatcher.forward(request, response);
```

using it
---

```
  <c:forEach items="${servers}" var="server">
    <LI>${server}
  </c:forEach>
```

maven dependencies:
---

```
    <dependencies>
```

```
		<dependency>
			<groupId>javax</groupId>
			<artifactId>javaee-web-api</artifactId>
			<version>6.0</version>
			<scope>provided</scope>
		</dependency>
	</dependencies>

	<build>
		<pluginManagement>
			<plugins>
				<plugin>
					<groupId>org.apache.maven.plugins</groupId>
					<artifactId>maven-compiler-plugin</artifactId>
					<version>3.2</version>
					<configuration>
						<verbose>true</verbose>
						<source>1.7</source>
						<target>1.7</target>
						<showWarnings>true</showWarnings>
					</configuration>
				</plugin>
				<plugin>
					<groupId>org.apache.tomcat.maven</groupId>
					<artifactId>tomcat7-maven-plugin</artifactId>
					<version>2.2</version>
					<configuration>
						<path>/</path>
						<contextReloadable>true</contextReloadable>
					</configuration>
				</plugin>
			</plugins>
		</pluginManagement>
	</build>


<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>jstl</artifactId>
   <version>1.2</version>
</dependency>
```

http://www.sitepoint.com/jsp-2-simple-tags/

http://www.coderanch.com/t/174748/java-Web-Component-SCWCD/certification/Difference-EVAL-BODY-INCLUDE-EVAL

https://www.mail-archive.com/jsp-interest@java.sun.com/msg15923.html