**Format this file with verdana 9, consolas 10 (3 space) from left, header rules, and other rules. No spacing template. Remove the dashe3s “-“ from the beginning of paragraphs.**

**-Cache**

-Shift + F5 = refresh while ignoring cache

-Using incognito mode to ignore cache.

* **Architecture**

Step 13 in <https://www.studytonight.com/servlet/creating-servlet-in-netbeans.php> work (the way xml works, etc.) because of this,



<https://www.javatpoint.com/server-web-vs-application>

-How to access web pages:Your root directory is Web Pages (it says “Web Pages” on Netbeans projects window. But in the files window and in windows file explorer, it is named “web”. So you don’t have to type “web” in the URL. Servlets are carried to “web” root directory at deployment. HTML and JSPs are already in “web” folder. If a file is in a subfolder in web, then you need to type that in the URL. If login.jsp is under web\public\login.jsp, then we access it via

“http://localhost:8080/WebApplication1/public/login.jsp”.

* **Deployment descriptor:** The deployment descriptor is an xml file, from which Web Container gets the information about the servlet to be invoked. The web container uses the Parser to get the information from the web.xml file.

If we don’t use the @WebServlet("/Servlet") annotation in servlet then we should declare the servlets in web.xml file using the code below. But with xml you can do much more than just declaring the servlet.

@WebServlet(name = "Servlet", urlPatterns = {"/ServletURL"})

public class NewServlet extends HttpServlet {

…

}

<servlet>

<servlet-name>Servlet</servlet-name>

<servlet-class>NewServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Servlet</servlet-name>

<url-pattern>/ServletURL</url-pattern>

</servlet-mapping>

When you create a servlet its information is added to web.xml if you choose the add to xml file

option. If you copy paste a servlet, add the appropriate metadata to web.xml and other places.

Servlet tag creates a servlet with the name Servlet and the class NewServlet.

<servlet>

<servlet-name>ServletName</servlet-name>

<servlet-class>ServletClass</servlet-class>

</servlet>

Servlet mapping tag allows us to choose which URL will access the servlet. With this mapping

you can use hyperlinks to access your servlets instead of being dependent on buttons.

<servlet-mapping>

<servlet-name>ServletName</servlet-name>

<url-pattern>/ServletURL</url-pattern>

</servlet-mapping>

To invokes the servlet for all files under foo,

<servlet-mapping>

<servlet-name>ServletName</servlet-name>

<url-pattern>/foo/\*</url-pattern>

</servlet-mapping>

Accessing servlets on a server.

http://hostname:portno/contextroot/urlpatternofservlet

http://localhost:9999/demo/welcome

The welcome-file-list tag allows us to set welcome files. (tag: homepage)

By default, server looks for the welcome file in the following order. If none of these files are found, the server renders a 404 error.

1. welcome-file-list in web.xml
2. index.html
3. index.htm
4. index.jsp

In the xml file below, we made the index.html file the welcome file.

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

For servlets, use ServletURL without slash.

<welcome-file-list>

<welcome-file>ServletURL</welcome-file>

</welcome-file-list>

The load-on-startup tag shows the order of servlet deployment. We use this to make sure some of them start working as soon as possible. Also, we want some servlets to be running before some other servlets. You change their order in web.xml file. First order is 0, not 1. If you don't load a servlet on startup, it will be loaded at request time, which will execute slower for the first time because of the loading.

<servlet>

<servlet-name>NewServlet</servlet-name>

<servlet-class>NewServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet>  
 <servlet-name>NewServlet2</servlet-name>

<servlet-class>NewServlet2</servlet-class>

<load-on-startup>2</load-on-startup>

</servlet>

* **API:** Servlet, GenericServlet, HttpServlet have their own implementations. They do different things. Check all 3 of them from the left menu. <https://www.javatpoint.com/GenericServlet-class>

You can create a servlet using three different ways.

You have to implement 5 methods of Servlet interface yourself(implements Servlet).

But if you extend GenericServlet, it provides an implementation for all methods of Servlet, ServletConfig, and Serializable except the service method.

The HttpServlet class extends the GenericServlet class and implements the Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace, etc.

System.out writes to console(glassfish server console). “PrintWriter out = response.getWriter();” outputs to response.

**-destroy:** You can call destroy explicitly but it will be called again when undeploying. Which will cause an exception. If you use destroy explicitly make sure to handle this exception.

* **request, response:** When we use parameters of a request as server, we dont get it from client. The client already sent the request with the parameters. So when we say request.getParameter, we get it from server memory. The request parameters were saved in server memory when request happened.

**-get, post:** get request is the default request (for example with forms).

**-getParameter:** We can get parameters that were sent with http protocol. One way to send parameters is, after file name in URL, put the parameter or parameters with the syntax below,

<http://localhost:8080/WebApplication1/newjsp.jsp?name=testname&amount=testamount>

String name = request.getParameter("name");

int amount = request.getParameter("amount")

This is one way to do a get request. You can do a get request with a form too. If you dont choose post as method of the form, when you click the button you will automatically send a get request such as

<http://localhost:8080/HttpServlet2/ServletURL?name=michael>

Or you can send the parameters with a post request.

**-Calling servlet from html:** We can create these hypertext or form links from html files or from inside servlet(using embedded html code). If you are going to create a hypertext link inside servlet, use \” instead of “ for values of attributes such as href of a. Or use single quotation marks ‘. Or dont use them at all since they are only mandatory in xhtml. But the standart i chose is using single quotes.

-In the html code below, we used "ServletURL" as an intraserver link.

<p> Click here to go to <a href="ServletURL">Servlet</a> </p>

So you dont have to use forms(action = ServletURL) to call servlets. You can just link to them,

<a href="ServletURL">View My Guestbook</a>

Here is how we do this in servlet. Used single quotes for attributes.

out.print(“<a href=’ServletURL’>View My Guestbook</a>”);

You can even send parameters while calling a servlet with a link,

<a href="ServletURL?operation=view">View My Guestbook</a>

Here is how we do this in servlet. Used single quotes for attributes.

out.print("<a href=’ServletURL?operation=view’>View My Guestbook</a>”);

And concatenated strings.

String operation = getParameter(“operation”);

out.print("<a href=’ServletURL?operation=’” + operation + ">View My Guestbook</a>”);

Or you can use javascript. First, create a form with hidden input,

<form name=form method=post action=ServletURL>

<input type=hidden value=xxx name=getit>

</form>

then, add a hyperlink with a submit function

<a href='javascript:document.form[0].submit()'>send it!</a>

-You can also use a form(button).

<form action="ServletURL">

Name:<input type="text" name="name" /><br/><br/>

Tel number:<input type="text" name="tel\_number" /><br/><br/>

<input type="hidden" value="insert" name="operation">

<input type="submit" value="Insert to database"/>

</form>

Here is how we do this in servlet. Used single quotes for attributes.

out.print("<form action='ServletURL’>");

out.print("Name:<input type='text' name='name' /><br/><br/>");

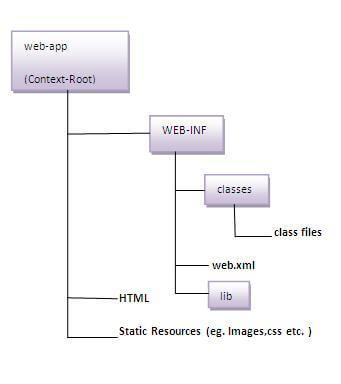
out.print("Tel number:<input type='text' name='tel\_number' /><br/><br/>");

out.print("<input type='hidden' value='insert' name='operation'>");

out.print("<input type='submit' value='Insert to database'>");

out.print("</form>");

-We are able to refer to the servlet just with its name because when the application is deployed, all web pages and servlets are held under Web Pages folder. You can see this if you look at the “build” folder. That is the version of the program that actually runs on server.



Creating war file has an advantage that moving the project from one location to another takes less time. To create the war file you need to use **-c** switch of jar. Go to project directory and use

jar cvf ProjectName.war \*

Here, -c is used to create file, -v to generate the verbose output and -f to specify the arhive file name.

The \* (asterisk) symbol signifies that all the files of this directory (including sub directory).

To extract the war file, you need to use -x switch of jar tool of JDK.

jar -xvf ProjectName.war

* **Servlet Collaboration**

**-RequestDispatcher:**

**-forward:** Forward request from a servlet to another resource on the server.

**-include:** Include the response of the resource(the one called with include) to the caller servlet. For example if someone enters a wrong password, you include the response of the login screen and you add “Username or password error” to that.

String p = request.getParameter("userPass");

if (p.equals("servlet"))

{

RequestDispatcher rd = request.getRequestDispatcher("WelcomeServletURL");

rd.forward(request, response);

}

else

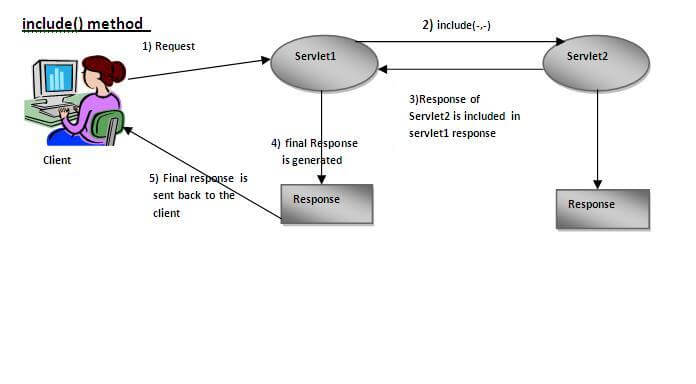
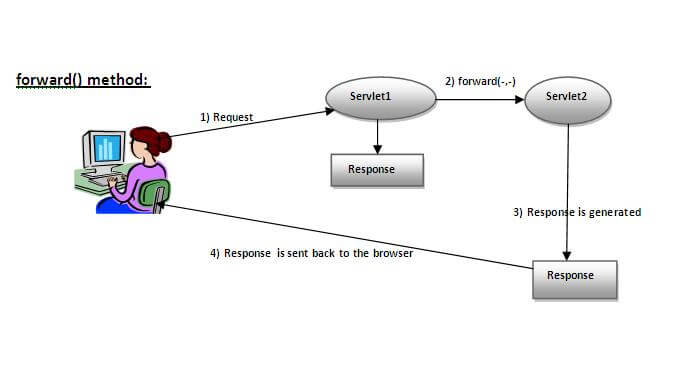
{

RequestDispatcher rd = request.getRequestDispatcher("login.html");

rd.include(request, response);

out.print("Username or password error.");

}

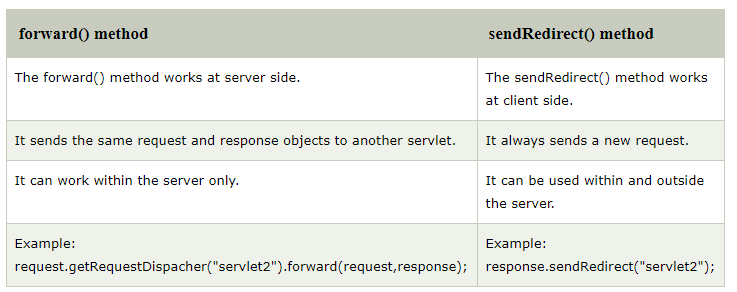


**-SendRedirect:** Redirects response to another resource, it may be servlet, jsp or html file.

It accepts relative as well as absolute URL.

It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

**-Difference between forward() and sendRedirect() methods**

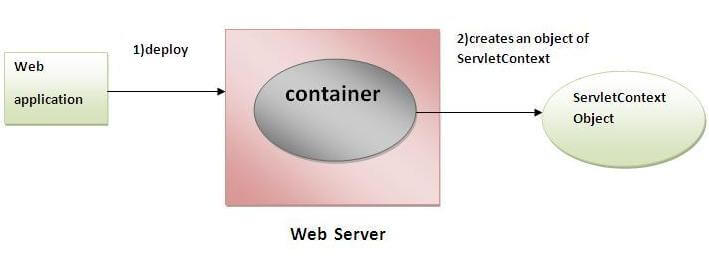


* **ServletConfig, ServletContext, Servlet Attribute:** Since we use a loosly coupled approach(access to this data from servlet) we can just change the information and dont need to modify the servlet.

**-ServletConfig:** All servlets have their own data. Web Container creates a ServletConfig object for all servlets.

-More info on javatpoint.

**-ServletContext:** There is only one ServletContext object per web application. All servlets in the web application can use it. You can get, set, remove attributes from web.xml. Can be used to provide inter-application communication.



More info on javatpoint.

**-ServletAttribute:** Instead of keeping the data on the xml file, we keep the data in a ServletContext object. Instead of giving the data to ServletContext object from xml, we give the data to the ServletContext object from code.

-More info on javatpoint.

* **Session Tracking**

**-Cookies**

-Cookies have their own expire time.

-You can have multiple cookies per visitor. Number of cookies a single website can place to clients computer is limited. It is around 20. And total number of cookies can be stored to clients computer is limited around 300.

**-**Cookies cant have white space characters in their names or values.

-Why are cookies held in client? Because servers dont hold information with ip, they use accounts so a person without an account doesnt have any persistant information in server.

-Cookies that were set by a website are sent to the website with every request. You can look at the ShowHeaders program (program 8) of my 2. Internshıp to look at HTTP headers that come with HTTP request. One of them is cookie header. This is why we can get the cookie list using

Cookie ck[] = request.getCookies();

-More info on javatpoint. Creating, getting, deleting cookies.

**-Hidden Form Field**

-You can have hidden form fields both client side and server side.

-Difference from cookies is that it is not dependant on browsers (whether cookies are disabled or not).

-More info on javatpoint.

**-URL Rewriting**

**-**Difference from cookies is that it is not dependant on browsers (whether cookies are disabled or not).

-More info on javatpoint.

**-Sessions**

-Sessions are deleted from client side when you close the browser. They are still in server side but browser gets new ones since it doesnt have any in client side. Other than that you can delete sessions using invalidate or we can set a time out just like a cookie.

-You have only one session per visitor.

* **Event and Listener**
* **Filters:** Lets say we have an AdminServlet which is an admin menu. We login with a username and password from index.html. This form sends the data to AdminServlet with action=”AdminServlet”. But before we let the user to the admin menu page, we need to check if he is really admin or not. We can use a filter here. That filter will intercept every request and response coming in and out of its target. We set filter’s target in web.xml in filter’s url-pattern line.

<servlet>

<servlet-name>AdminServlet</servlet-name>

<servlet-class>AdminServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>AdminServlet</servlet-name>

<url-pattern>/AdminServlet</url-pattern>

</servlet-mapping>

<filter>

<filter-name>MyFilter</filter-name>

<filter-class>MyFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>MyFilter</filter-name>

<url-pattern>/AdminServlet</url-pattern>

</filter-mapping>

In the code above we define a servlet and a filter that has the servlet as its target.

Flush

public void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException

{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.write("test");

out.write("test2");

out.write("test3");

out.println("<br>");

out.flush(); // Normally the complete response is formed at the end of the

// method and sent to the client. But we can use the flush method to send

// the responses that were added to the complete response so far. The

// responses added after flush method will be included at the end of

// previous response.

try {

Thread.sleep(5000);

} catch (InterruptedException ex) {

Logger.getLogger(Flush\_Wait.class.getName()).log(Level.SEVERE, null, ex);

}

out.write("test4");

out.write("test5");

out.write("test6");

}

Are there other things to add from first internship.