**Servlet Container: Why required Servlet class only:**

Servlet is nothing but a java class which is capable to read data coming from browser. Now the question arises why these classes (Servlet) are only capable to read data. Because container will allow the servlet to read the data coming from browser. So to run server side servlet class (servlet) we required servlet container.

How to execute these classes i.e. what the servlet container exactly will do?

**Steps:**

1. Whenever we submit a from data from a browser state away it goes to server. This form data will be sent in the form of http protocol from the browser. Http protocol has two types of format header and body. We can send the data through header or body. If it header then we call it query string and if it is body then we call it payload. Now the question arises that who can understand this data. So the servlet container will understand this data.
2. Servlet container will accept or read this http protocol data and store this http protocol data into java bean object that is nothing but HttpServletRequest class object.
3. Now this HttpServletRequest object will be forward to servlet classes by servlet container. Now servlet will process the data and provide the response object in the form HttpServletResponse object.
4. Again this HttpServletResponse object will convert into Http protocol and the same response it will send to the browser finally.

Now we can conclude that the data which is sent to servlet is in the form of HttpServletRequest and response we get in the form of HttpServletResponse. That is why we need a servlet to process the data in the server side.

Now the question arises which servlet class it will go after converting the http data into HttpServletRequest as in the project we will have n(A,B,C) number of servlet classes i.e. which servlet class need to respond HttpServletRequest object? Is needed to execute all the servlet class?

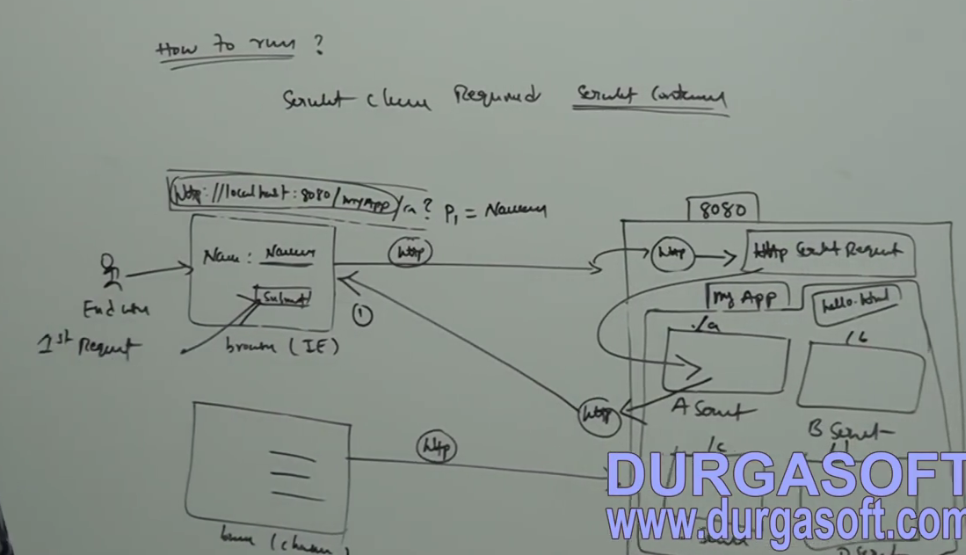
Answer: It depends on your URL. Let’s say if all the servlet classes are in one application (NXPApp) and inside this application we have A, B, C servlet classes and let’s say we have one Jsp page NXP.jsp.

Steps:

1. <http://localhost:8080/NXPApp/NXP.jsp> is submitted from the browser. JSP page will be displayed on browser.
2. Let’s say in the JSP page we have one text field and submit button. So when we click on submit button , the controller will search one URL pattern (./A or ./B) and now the http request will like as <http://localhost:8080/NXPApp/A> Now the Servlet A will be executed.
3. Now question is as Servlet class is a java class and java class is executed when we create an object of the class. So who will create the object: Container will create the object when the user made very first time request. Since the object created by container is a singleton object so from the next request on words the same object reference will be assigned to next request object.
4. After executing the servlet classes it generates response into HttpServletResponse which again will be converted into http protocol and then finally send to browser.
5. Here localhost can be your server IP or domain name.

Now the question arises that how the container will execute the servlet class. So answer is we need to fallow some directory structure as we deploy the application in the form of war or jar file. So this directory structure will understand by servlet container

|  |  |
| --- | --- |
|  | ApplicationName folder [  View pages(html ,jsp ,css)  Web-INF folder[  Classes folder  Lib folder  ]  ] |



**How the container will execute the servlet class present inside the application:**

Steps: When we start the container

1. It will try to read web.xml file present inside the application.
2. In the web.xml file it will read all the mappings (Url pattern and servlet class )
3. Now for the servlet class it will search under classes’ folder if not found it will show class not found exceptions?
4. Now container will create the object of servlet class (either from first user call or while starting up the server load-on-startup)

**Servlet Life Cycle:**

The life cycle of servlet start when we deploy the Servlet into Servlet container. Now the question arises, what the container exactly do when we deploy application in container. Usually here we are using tomcat container as a servlet container.

|  |  |
| --- | --- |
|  | Let say if we have one Servlet tomcat container. So when we start the tomcat container then container will read the entire war file (A.war, B.war, and C.war which contains the entire application contents) deployed in the server and it will extract entire war file (i.e. remove .war and convert in the form of A,B,C) and store the extracted application in web-app server i.e. inside the container. As we know that when we create a project then we have to create a well defined structure i.e. first of all we will have project folder, inside this we will have web-INF and inside the Web-INF we will have classes and web.xml. Here HelloWorld.War will be extracted into HelloWorld having the folder structure shown on LHS. |

So after extraction of the war file the container will go inside of web-inf and read the web.xml file placed inside the web-inf.

After reading web.xml file the container will create two types of objects

1- Servlet Context objects. (2) Servlet config object.

**Servlet Context:** The servlet context object will be public and will be shared by all the servlet present inside the application i.e. servlet context object scope will application level scope which will be available throughout the application. Suppose if any common value has to be shared by all then this value must be public and will be defined in servlet-context.

For each application, servlet container will create one–one servlet context object which will be public and common and can be shared by all the servlet present inside the application. We can read the servlet-context object from any servlet as well as we can store or modify the servlet-context object from any servlet present inside the application as it is public and is available to the application.

Servlet context is defined inside of <context-param> and outside of <servlet> tag so that it could be available for all the servlet defined in the web.xml or application

**Servlet Config:** For each application and for each servlet, the container will create servlet-config object which will be available for the only servlet for which it has created. In other word servlet-config object will be private and will be available to the servlet for which it is created. In other word the servlet-config object scope will be servlet level scope and it will not be available for other servlet.

Servlet config object is written inside the <servlet> tag so that it could be available for that particular of servlet only.

So servlet-context object is public for all the servlet and servlet-config object is private for each servlet.

**Servlet Config Declaration in web.xml.**

|  |
| --- |
| <web-app>  <display-name>Login Servlet Application</display-name>  **<servlet>**  **<servlet-name>FirstServlet</servlet-name>**  **<servlet-class>com.FirstServlet</servlet-class>**  **<init-param>**  **<param-name>a</param-name>**  **<param-value>123</param-value>**  **</init-param>**  **</servlet>**  **<servlet>**  **<servlet-name>SecondServlet</servlet-name>**  **<servlet-class>com.SecondServlet</servlet-class>**  **<init-param>**  **<param-name>b</param-name>**  **<param-value>456</param-value>**  **</init-param>**  **</servlet>**  </web-app> |
| **Servlet config object will be written inside the <init-param> of inside <servlet> tag so that it will be available for that particular of servlet. Here param-value 123 and 456 cannot be shared between any servlet. This value can be accessed by one the servlet under which it is declared.** |

**ServletContext Declaration in web.xml**

|  |
| --- |
| **<web-app>**  **<display-name>Login Servlet Application</display-name>**  <servlet>  <servlet-name>FirstServlet</servlet-name>  <servlet-class>com.FirstServlet</servlet-class>  <init-param>  <param-name>a</param-name>  <param-value>123</param-value>  </init-param>  </servlet>  <servlet>  <servlet-name>SecondServlet</servlet-name>  <servlet-class>com.SecondServlet</servlet-class>  <init-param>  <param-name>b</param-name>  <param-value>456</param-value>  </init-param>  </servlet>  **<context-param>**  **<param-name>name</param-name>**  **<param-name>Arun</param-name>**  **</context-param>**  **</web-app>** |
| **Servlet context is written inside of <context-param> and outside of <servlet> tag so that it will be available for all the servlet (FirstServlet and SecondServlet)** |

So by reading the servlet-context and servlet-config value present inside the web.xml it will assign this value to servlet-context object and servlet-config object respectively.

**Servlet Mapping:** For each servlet present inside the application we will have mapping in the web.xml. We will write that mapping inside <servlet-mapping> tag and inside this tag we will have <servlet-name> and <url-patter> tag.

|  |
| --- |
| **<web-app>**  **<!—Servlet Mapping -->**  **<servlet-mapping>**  **<servlet-name>FirstServlet</servlet-name>**  **<url-pattern>/first</url-pattern>**  **</servlet-mapping>**  **<servlet-mapping>**  **<servlet-name>SecondServlet</servlet-name>**  **<url-pattern>/second</url-pattern>**  **</servlet-mapping>**  **<!—Servlet Config -->**  <servlet>  <servlet-name>FirstServlet</servlet-name>  <servlet-class>com.FirstServlet</servlet-class>  <init-param>  <param-name>a</param-name>  <param-value>123</param-value>  </init-param>  </servlet>    <servlet>  <servlet-name>SecondServlet</servlet-name>  <servlet-class>com.SecondServlet</servlet-class>  <init-param>  <param-name>b</param-name>  <param-value>456</param-value>  </init-param>  </servlet>  **<!—Servlet Context -->**  <context-param>  <param-name>name</param-name>  <param-name>Arun</param-name>  </context-param>  **</web-app>** |
| **Here inside the <url-pattern> we have defined one pattern /first and / second which will be exactly same with the form action tag defined in the jsp and this url pattern will be unique for each servlet because with this url pattern the container will execute the particular servlet after matching with http request url pattern**  **(http://** **localhost:8080/myservlet/first) or (http://** **localhost:8080/myservlet/second)** |

Q: How the object of servlet (FirstServlet and SecondServlet) is created?

The object of servlet can be created by two ways.

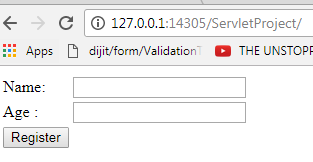
1. During the servlet container startup using tag <load-on-startup>1</ load-on-startup>
2. Second way is when the user makes the first request for that particular of servlet using the URL pattern [**http://localhost:8080/myservlet/first**](http://localhost:8080/myservlet/first) **or** [**http://localhost:8080/myservlet/second**](http://localhost:8080/myservlet/second)**.**
3. Now if same user or any other user make the same request again then this time it not create any object, instead of creating new object it will assign the same object for repeated request .i.e. it will create one singleton object for each servlet and same object will be assigned for every request. And this single object will be created by servlet container only. So servlet object is singleton but servlet class is not singleton.

**Servlet Life Cycle: Creating MyServelet using Servlet Interface:**

|  |
| --- |
| package com.servlet.interfac.demo;  import java.io.IOException;  import javax.servlet.Servlet;  import javax.servlet.ServletConfig;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.annotation.WebServlet;  public class MyServlet implements Servlet {  public MyServlet() {  System.out.println("My default constructor gets called");  }  public void init(ServletConfig config) throws ServletException {  String servletInfor= config.getInitParameter(getServletInfo());  String configVal = config.getInitParameter("a");  System.out.println("Servlet Info: "+servletInfor );  System.out.println("Config value :"+ configVal);  }  public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {  System.out.println("Service Method is getting called");  String name =req.getParameter("name");  String age = req.getParameter("age");  }  public void destroy() {  System.out.println("Destroy method is getting called");  }  public ServletConfig getServletConfig() {  // TODO Auto-generated method stub  return null;  }  public String getServletInfo() {  // TODO Auto-generated method stub  return null;  }  } |

|  |
| --- |
| <html>  <head></head>  <body>  <form action=*"./MyServlet"* method=*"get"*>  <table>  <tr>  <td>Name:</td>  <td><input type=*"text"* value=*""* id=*"name"*></td>  </tr>  <tr>  <td>Age :</td>  <td><input type=*"text"* value=*""* id=*"age"*></td>  </tr>  <tr>  <td><input type=*"Submit"* value=*"Register"* id=*"Register"*></td>  </tr>  </table>  </form>  </body>  </html> |

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app id=*"WebApp\_ID"* version=*"3.0"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*>  <display-name>ServletProject</display-name>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>    <servlet>  <servlet-name>MyServlet</servlet-name>  <servlet-class>com.servlet.interfac.demo.MyServlet</servlet-class>  <init-param>  <param-name>a</param-name>  <param-value>123</param-value>  </init-param>  </servlet>    <servlet-mapping>  <servlet-name>MyServlet</servlet-name>  <url-pattern>/MyServlet</url-pattern>  </servlet-mapping>    <context-param>  <param-name>name</param-name>  <param-value>Arun</param-value>  </context-param>  </web-app> |



When we run the above application

1. First it reads web.xml file
2. When we click on button , container will go to <servlet-mapping> section of web.xml and it will generate one action url (Mentioned in the **<url-pattern>/MyServlet</url-pattern>)**
3. After that it will go to <Servlet-class> and create an object of the servlet class mentioned as

**<servlet-class>com.servlet.interfac.demo.MyServlet</servlet-class>**

1. As soon as object gets created, it will execute init method. Next time onwards it will execute service () method for every call.
2. Very first time user request it will execute Constructor, init and service method. But after this for every user call it will execute the service method only

**Note**: If we mention <load-on-startup> 1 </ load-on-startup> in web.xml for servlet class then during the project deployment it will create the object of the servlet for which it is mentioned as <load-on-startup>. But if we have not mentioned <load-on-startup> then object of servlet will be created on user first request and same object will be assigned for the every request.

|  |
| --- |
| <servlet>  <servlet-name>MyServlet</servlet-name>  <servlet-class>com.servlet.interfac.demo.MyServlet</servlet-class>  <load-on-startup>**1**</ load-on-startup>  <init-param>  <param-name>a</param-name>  <param-value>123</param-value>  </init-param>  </servlet> |

Here we have mentioned load-on-startup, so container will not wait for user request for creating the object .Container will create the object on server startup itself. So in this case constructor and init method will be executed at the time server startup. After that for every user request it will execute service () method only .

1. Destroy () will be executed at the time of project deployment or server shutdown.

**GenericServlet**

We have already discussed about servlet (I) interface for creating our own servlet class. There is Servlet Class for creating our own servlet

1. Servlet (I)
2. GenericServlet
3. HttpServlet

While creating our server side servlet class by implementing Servlet interface, we need to implement all the abstract method (init, service, destroy, getServletConfig, getServletInfo).

1. Now if we want to write a server side java class (servlet class) without writing servlet life cycle methods (init, destroy, getServletConfig, getServletInfo) except service method. In this case we need to extend **GenericServlet** class. Actually GenericServlet is written by implementing Servlet (I) interface only and in the GenericServlet class they have implemented all the abstract class (init, destroy, getServletConfig, getServletInfo) with empty body. But they did not implement service () method.
2. So while creating our own servlet class by extending GenericServlet, we need to implement only one method Service (). If required then we can override init and remaining method but it is not compulsory.

|  |
| --- |
| **package** com.servlet.interfac.demo;  **import** java.io.IOException;  **import** javax.servlet.Servlet;  **import** javax.servlet.ServletConfig;  **import** javax.servlet.ServletException;  **import** javax.servlet.ServletRequest;  **import** javax.servlet.ServletResponse;  **import** javax.servlet.annotation.WebServlet;  **abstract public** **void**  GenericServlet **implements** Servlet {    **public** **void** init(ServletConfig config) **throws** ServletException {  // Empty Body  }  **abstract** **public** **void service(ServletRequest req, ServletResponse res);**  // Left implementing service method only declare the prototype.  **public** **void** destroy() {  // Empty Body  }  **public** ServletConfig getServletConfig() {  // Empty Body  **return** **null**;  }  **public** String getServletInfo() {  // Empty Body  **return** **null**;  }  } |

As we can see that they have left implementing only one abstract method service that developer need to implement while using GenericServlet class.

So while extending GenericServlet class then we must implement service method because service method is and abstract method that is compulsory to implement in child class (our own servlet class)

Creating MyServlet by extending GenericServlet class and implementing abstract service method present in Generic Servlet.

|  |
| --- |
| **package** com.servlet.interfac.demo;  **import** java.io.IOException;  **import** javax.servlet.GenericServlet;  **import** javax.servlet.Servlet;  **import** javax.servlet.ServletConfig;  **import** javax.servlet.ServletException;  **import** javax.servlet.ServletRequest;  **import** javax.servlet.ServletResponse;  **import** javax.servlet.annotation.WebServlet;  **public** **class** MyServlet **extends** GenericServlet {    **public** **void** service(ServletRequest req, ServletResponse res) **throws** ServletException, IOException {  // do your operations  System.*out*.println ("Service Method is getting called .............");  String name =req.getParameter("name");  String age = req.getParameter("age");  System.*out*.println("Name:"+name);  System.*out*.println("Age: "+age);  }    } |

**Note:** Here while implementing service method we need to pass plane (ServletRequest, ServletResponse) parameters. This ServletRequest and ServletResponse do not have any session management capabilities.

So get session capability we must extends HttpServlet class as while extending HttpServlet class we have HttpServletRequest and HttpServletResponse which has the session management capability.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | |

Now we will get servlet class GenericServletDemo extending GenericServlet Class with service method implementation.

|  |
| --- |
| **package** com.servlet.interfac.demo;  **import** java.io.IOException;  **import** javax.servlet.GenericServlet;  **import** javax.servlet.ServletException;  **import** javax.servlet.ServletRequest;  **import** javax.servlet.ServletResponse;  **import** javax.servlet.annotation.WebServlet;  **public** **class** GenericServletDemo **extends** GenericServlet {  **private** **static** **final** **long** *serialVersionUID* = 1L;  **public** GenericServletDemo() {  **super**();  // **TODO** Auto-generated constructor stub  }  **public** **void** service(ServletRequest request, ServletResponse response) **throws** ServletException, IOException {  // **TODO** Auto-generated method stub  }  } |

|  |
| --- |
| <html>  <head></head>  <body>  **<form action=*"./GenericServletDemo"* method=*"post"*>**  <table>  <tr>  <td>Name:</td>  <td><input type=*"text"* id=*"name"* name =*"name"* ></td>  </tr>  <tr>  <td>Age :</td>  <td><input type=*"text"* id=*"age"* name =*"age"*></td>  </tr>  <tr>  <td>Email :</td>  <td><input type=*"text"* id=*"email"* name =*"email"*></td>  </tr>  <tr>  <td>Address :</td>  <td><textarea rows=*"5"* cols=*"12"* id=*"address"* name=*"address"/*>  </td>  </tr>  <tr>  <td><input type=*"Submit"* value=*"Register"* id=*"Register"*></td>  </tr>  </table>  </form>  </body>  </html> |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app id=*"WebApp\_ID"* version=*"3.0"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*>  <display-name>ServletProject</display-name>  <welcome-file-list>  <welcome-file>registration.jsp</welcome-file>  <welcome-file>index.html</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  </welcome-file-list>    <!-- First Servlet configuration -->  <servlet>  <servlet-name>MyServlet</servlet-name>  <servlet-class>com.servlet.interfac.demo.MyServlet</servlet-class>  <init-param>  <param-name>a</param-name>  <param-value>123</param-value>  </init-param>  </servlet>  <servlet-mapping>  <servlet-name>MyServlet</servlet-name>  <url-pattern>/MyServlet</url-pattern>  </servlet-mapping>    <!-- Second Servlet configuration -->  **<servlet>**  **<servlet-name>GenericServletDemo</servlet-name>**  **<servlet-class>com.servlet.interfac.demo.GenericServletDemo</servlet-class>**  **<init-param>**  **<param-name>a</param-name>**  **<param-value>123</param-value>**  **</init-param>**  **</servlet>**  **<servlet-mapping>**  **<servlet-name>GenericServletDemo</servlet-name>**  **<url-pattern>/GenericServletDemo</url-pattern>**  **</servlet-mapping>**    <context-param>  <param-name>name</param-name>  <param-value>Arun</param-value>  </context-param>  </web-app> |
| **public** **class** GenericServletDemo **extends** GenericServlet {  **private** **static** **final** **long** *serialVersionUID* = 1L;    **public** GenericServletDemo() {  **super**();  // **TODO** Auto-generated constructor stub  }  **public** **void** service(ServletRequest req, ServletResponse res) **throws** ServletException, IOException {  // **TODO** Auto-generated method stub  res.setContentType("text/html");  PrintWriter pw = res.getWriter();  String name = req.getParameter("name");  String age = req.getParameter("age");  String email = req.getParameter("email");  String address = req.getParameter("address");  System.*out*.println("Name :"+name+"Age :"+age+"Email :"+email);  System.*out*.println("Address : "+address);  **try** {  Class.*forName*("Put driver Name");  Connection con = DriverManager.*getConnection*("Put url name");  PreparedStatement pst = con.prepareStatement("Inser query");  pst.setString(1, name);  pst.setString(2, age);  pst.setString(3, email);  pst.setString(4, address);  **int** i = pst.executeUpdate(); // will return integer  **if**(i!=0){  pw.println("Registration Success");  }**else**{  pw.println("Registration failed");  }  } **catch** (Exception e) {  // **TODO**: handle exception  }  }  } |

Use Case: Suppose we have three form (Name.jsp, Education.jsp and Address.jsp)

Name.jsp : First Name, Middle Name, Last Name

Education.jsp: Qualification Name, Percentage, Passing Year.

Address.jsp : House No, City, District, Country, Pin code.

We have to fill all the information in the database.

Now if we Extend GenericServlet class then for reading each form data we need to create one-one servlet and as GenericServlet can handle only one form data i.e. multiple form data cannot be handled by GenericServlet as it cannot store form data in the session i.e. when we submit first form data, then first from data it needs to remember or store along with second form data when we submit second form. Finally we submit last form data then it needs to remember all three form data and need to store in the database. But that kind of capability we don’t have in GenericServlet. Suppose if we are using one-one servlet for each form, then there may be inconsistency of stored data, we cannot match each record with its name, education and address as each form data is submitting separately.

So GenericServlet class is not at all suitable when multiple forms need to submit at one time for data consistency. In this case HttpServlet Class can be used to maintain data consistence when we have multiple form (Where all the from data needs to submit together)

When we use HttpServlet class we don’t need to submit data into database until we finishes last data reading.

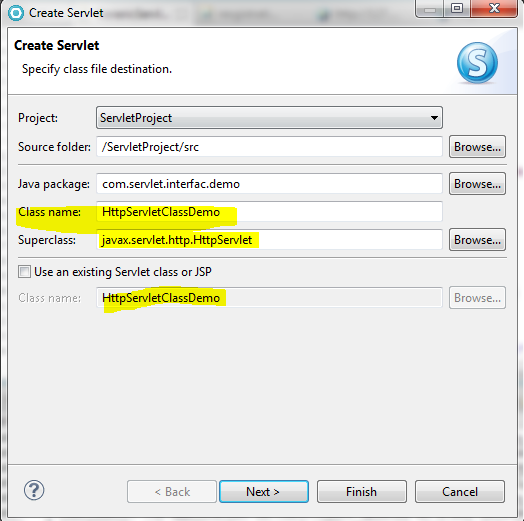
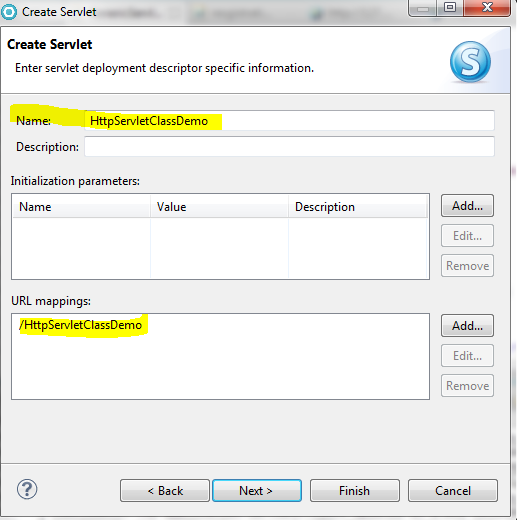
And we can write only one single servlet for all three forms.

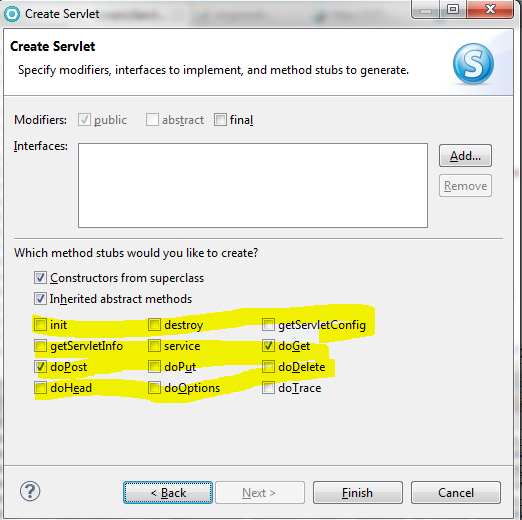
Steps: How we handle all three form data submission using HttpServlet class.

1. Create three jsp as given above.
2. Create one hidden field in each from with different id or name.
3. Map the servlet in web.xml
4. Create one servlet class and extend HttpServlet class.
5. Read the first form data and hidden field data of each form.
6. Based on hidden field data i.e. put first form hidden field data into if conditions and store all read data into Http session.
7. Same thing we will do for from 2.Now we have stored all the data into session.
8. In the first if condition after reading and storing the data into session we will use (response.sendRedirect (“form2 Location”) so that when we click on Next button of first form, it take us to next second form and store the form data into session.
9. Same stuff we will do for second from if condition as we do in step 8.
10. In the form 3 we will read all data for first and second form from session and will store into variables. Third from data we don’t need to store in session because this is the last step and here we are submitting the data into database.
11. Create a database connection.
12. Pass the insert query statement into PreparedStatement.
13. Set each from data into pst.setString(index , read variable)
14. Int a = Pst.executeUpdate ().
15. Based on a will show the success or failure message.

**HttpServlet Class**

This class can be used for data consistence when we have multiple form and all the from data we need to insert into data base in single click.



|  |
| --- |
| **package** com.servlet.interfac.demo;  **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.annotation.WebServlet;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  @WebServlet("/HttpServletClassDemo")  **public** **class** HttpServletClassDemo **extends** HttpServlet {  **private** **static** **final** **long** *serialVersionUID* = 1L;  **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  // **TODO** Auto-generated method stub  }  **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  // **TODO** Auto-generated method stub  }  } |

As we can see in the above step during http servlet creation, it has already implemented all the servlet life cycle methods (init, service, destroy, getServletConfig, getServletInfo) along with some other method (doGet, doPost, doPut, doDelete, doHead, doOptions, doTrace) which is nothing but similar to service () method

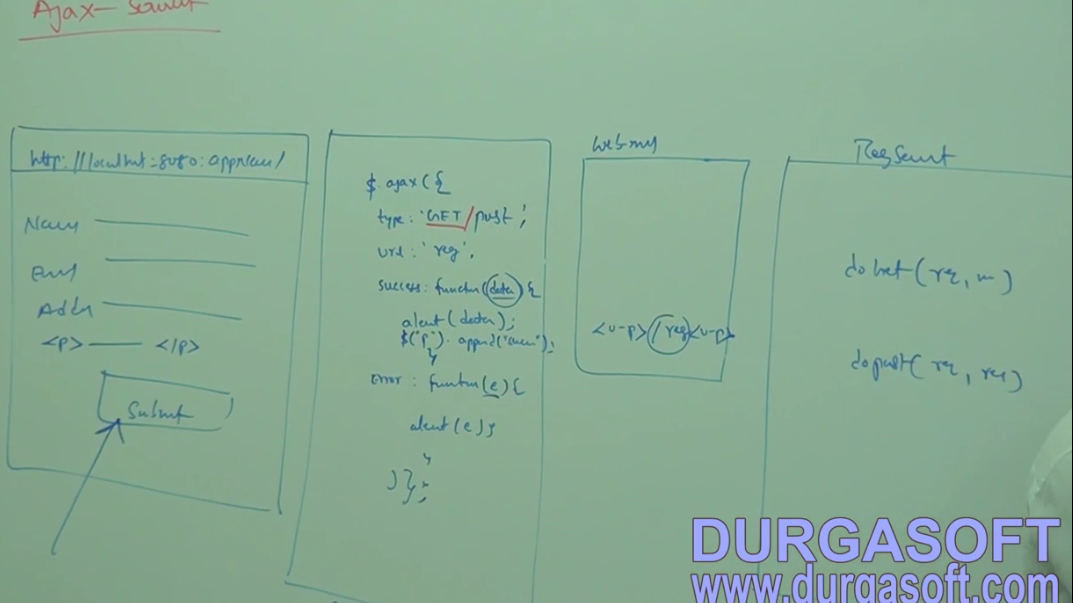
So in HttpServlet class we have fallowing methods

|  |  |  |
| --- | --- | --- |
| **Servlet Life Cycle Methods** | **HttpServlet Class service methods** | **Recommended Methods** |
| init()  service()  destroy ()  getServletConfig()  getServletInfo() | doGet()  doPost()  doPut()  doDelete()  doHead()  doOptions()  doTrace() | While extending HttpServlet class we need to override only two methods doGet () and doPost () remaining methods are not recommended to use. All HttpServlet methods are equivalent to service () life cycle method. |

Example or use case implementation: Pending

**Ajax implantation in Servlet (Asynchronous java script to an xml)**

While using Ajax we don’t need to submit whole form data, only those filed data need to send which is required. In Ajax call form will not submitted or refresh. Link in non Ajax, we need to have any from action in the html or Jsp. Action we will write under java script / JQuery.



|  |
| --- |
| 1. Create one html or jsp form. 2. Add JQuery API script inside <script> tag of jsp for Ajax call. 3. In web.xml file will create servlet mapping and URL pattern. 4. Write one JQuery function on the bottom of jsp as shown in the next column.   $(document).ready(function() { } );   1. Create one servlet |
| $(document).ready(function() {  $(“#ButtonId”).click(function(event) {  // Read the form data  Var name = $(‘#name’).val();  Var email = $(‘# email).val();  // Write ajax function  $.ajax({  type : “POST”, // Method type Get or Post  URL : “reg” , // this will be url pattern (web.xml)  data: “name=” +name + “&email=” +email , // These the value that we are submitting to servlet or server  success : function(msg) {  alert(“success) ;// write whatever we want to write  return false ; },  error : function(e) {  alert(“fail) ;// write whatever we want to write  return false ; } // Finally we don’t need to put comma  })  });  }); |

1. In JQuery Ajax event is nothing but the from data that we have to pass here.
2. & is a separator between two values sending to servlet.

So when we click on submit button.

1. It calls the Ajax java script method.
2. Here we are mentioning the url patter under URL (as we are mentioning in the form action)
3. Under the data section we are submitting the data to the servlet or payload.
4. From the server side we will get either success or error that we have to handle under success or error section.

Example: Pending

**Servlet Filters**

As we know that every time always submit the form data to servlet. In the servlet we read the form data and pass it to model part for database operation. Again whatever response we get from model layer is received by servlet and again servlet pass the response to browser.

While submitting the form data to servlet, some time it may send valid data or invalid data and for doing validation we used to use java script. But in case java script execution fails (like in our local machine we disabled the java script then validation will not work) then invalid data will submit to servlet or server and we will get wrong output. So before submitting your data to servlet or server if we want do one more time filter kind of operations (like if we want to check field is empty, not null or valid input ect.) then we need to use filter.

So filter works as middle layer between form data and servlet. It filters the data coming from the form and send only valid data to the servlet. In the filter we can validate the from data whether it is empty or not, or whether it is not null or not or some other kind of validation.

If we are using filter between form data and servlet, it will execute two times (back and forth).

1. First time when we will submit form data to the servlet then filter (which will be available in the middle of form data and servlet) get execute like for validation or appending some other inputs.
2. Second time when we get response from the servlet like after getting response from servlet if we want append some other data in the response. So before sending the final response to the browser we can append some other data also (like some advertise data or adding some color)

If we try to implement filter, we need to implement three methods.

1. public void init(FilterConfig fConfig)
2. public void doFilter(ServletRequest request, ServletResponse response, FilterChain fc)
3. public void destroy()

To send the filtered data to servlet we need to fc.doFilter (req, res);. Once we call this method it will execute the servlet. After servlet execution again the response will come to the servlet. So filter will execute two times (before servlet execution and after servlet execution) while servlet execute only once.

