ICON OF JAVA

ADV. JAVA

Servlets

3.Servlets design

Mr. Nagoor Babu M.Tech

[ICON of JAVA]
(Sun certified & Realtime Expert)
Ex. HCL Employee
Trained Lakhs of Students
for last 14 years across INDIA

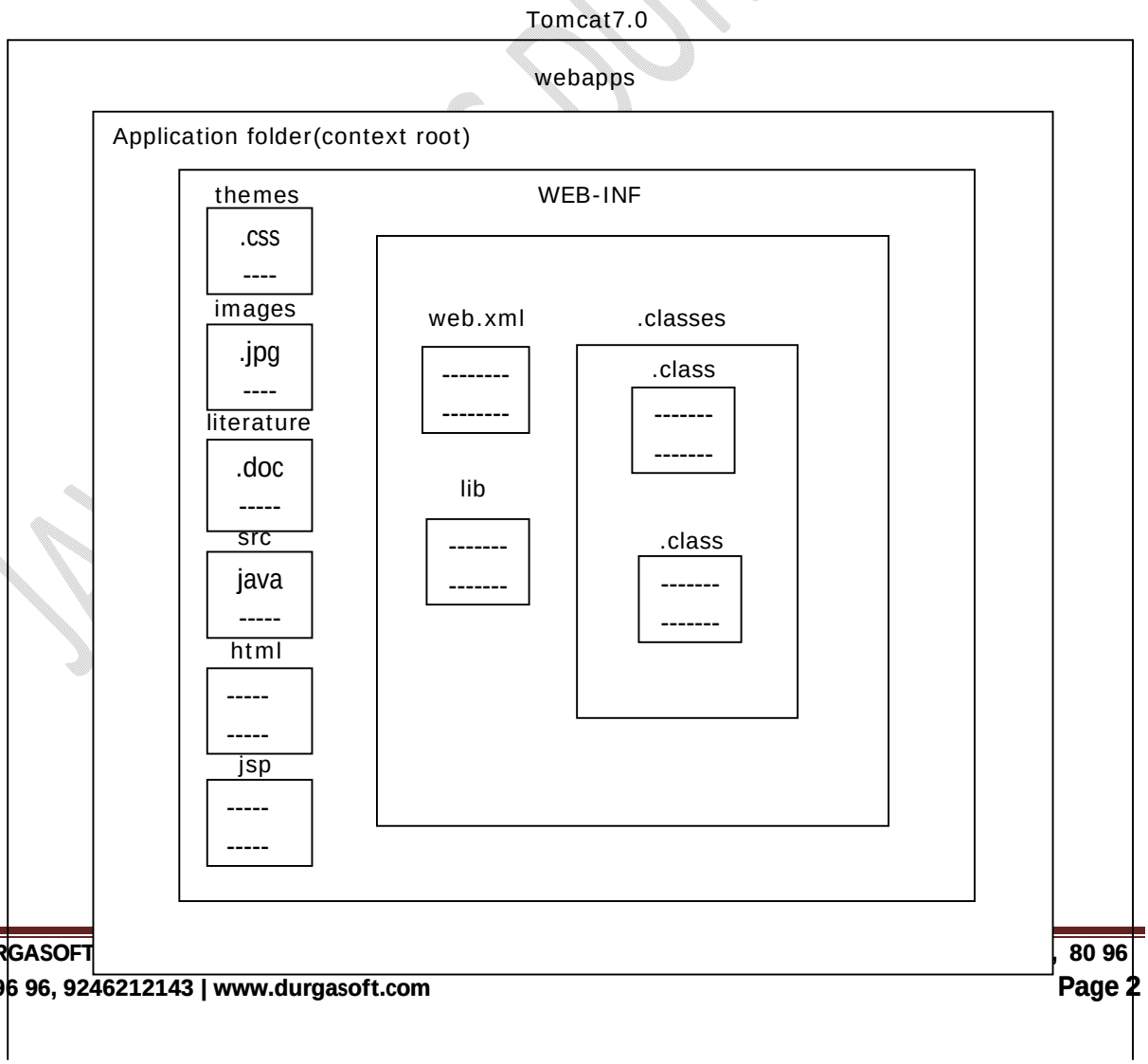
Steps to design first web application(Servletapplication)

To design first web application we have to use the following steps.

1. Prepare web application directory structure
2. Prepare deployment descriptor
3. Prepare the required web resources like html, servlets, Jsp's and so on
4. Start the server
5. Access the web application

Step 1: Web Application Directory Structure:

If we want to design any web application then we have to prepare the following directory structure at server machine.



When we install Tomcat server automatically Tomcat7.0 folder will be created. In Tomcat7.0 folder, the main purpose of webapps folder is to accommodate all the web applications provided by the developer.

To represent our own web applications in server we have to prepare a separate folder under webapps folder i.e. application folder or context root. Inside the application folder,

1. Themes: To store .css files(cascade style sheet) to generate reports.

2. Images: To store logos of the organizations, background sceneries and so on in the form of .jpg, .jpeg, .gif files.

3. Literature: To store documentations in the form of .doc, .docex and so on.

4. src(Source code): It can be used to store all the source files like Java files.

5. Along with all these folders it is possible to provide some static resources directly like .html files and dynamic resources like .jsp files.

6. WEB-INF folder will include

1. web.xml: This xml file can be used to store the metadata about the web application required by the container to perform a particular action.

2. lib: It is able to manage jar files which are required by the server side components.

3. classes: It will include .class files of servlets, filters, listeners and so on.

The above web application directory structure was divided into the following

1. Private area:

In web application directory structure, WEB-INF folder and its internal is treated as private area. If we deploy any resource under private area then client is unable to access that resource by using its name directly.

Note: In general we will keep servlet .class files under classes folder i.e. private area so that we are unable to access that servlet by using its name directly from client.

To access any servlet we have to define an URL pattern in web.xml file w.r.t this servlet, with the defined URL pattern only client is able to access the respective servlets.

2. Public area:

The area which is in outside of WEB-INF folder and inside the application folder is treated as public area. If we deploy any resource under public area then client is able to access that resource by using its name directly.

Note: In general in web applications, we are able to keep html files under application folder i.e. public area so that we are able to access that resources by using its name directly from client.

Step 2: Deployment Descriptor or web.xml file:

Deployment Descriptor is web.xml file, it can be used to provide the metadata about the present web application required by the container in order to perform a particular server side action.

In web applications, web.xml file include the following configuration details w.r.t the web application

1. Welcome Files Configuration
2. Display Name Configuration
3. Servlets Configuration
4. Filters Configuration
5. Listeners Configuration
6. Context Parameters Configuration
7. Initialization Parameters Configuration
8. Session Time Out Configuration
9. Load On Startup Configuration
10. Error Page Configuration
11. Tag Library Configuration
12. Security Configuration

In general in web applications, we will deploy the servlets .class files under classes folder of the web application directory structure i.e. private area.

If we deploy any resource under private area then client is unable to access that resource through its name, client is able to access that resource through alias names or locators. In case of servlets, client is able to access servlet classes through the locators called as **URL Patterns.**

If we provide multiple number of servlets under classes folder and we provide a particular request to a particular servlet available under classes folder with an URL pattern then container should require mapping details between URL patterns and servlets class names as metadata in order to identify w.r.t servlet.

In the above context, to provide the required metadata to the container we have to provide servlet configuration in web.xml file. To provide servlet configuration in web.xml file we have to use the following xml tags.

```
<web-app>
    --------
<servlet>
<servlet-name>logical_name</servlet-name>
<servlet-class>fully qualified name of servlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>logical_name</servlet-name>
<url-pattern>urlpattern_name</url-pattern>
</servlet-mapping>
    --------
</web-app>
```

Note: In the above servlets configuration, <servlet-name> tag value under <servlet> tag and <servlet-mapping> tag must be same.

Ex:

```
<web-app>
<servlet>
<servlet-name>loginservlet</servlet-name>
<servlet-class>com.dss.login.LoginServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>loginservlet</servlet-name>
        <url-pattern>/login</url-pattern>
```

```
</servlet-mapping>

</web-app>
```

If we want to access the above servlet then we have to provide the following URL at client browser.

http://localhost:8080/loginapp/login

In servlet configuration, there are 3 ways to define URL patterns.

1. Exact Match Method
2. Directory Match Method
3. Extension Match Method

1. Exact Match Method:

In Exact Match Method, we have to define an URL pattern in web.xml file, it must be prefixed with forward slash("/") and pattern name may be anything.

Ex: <url-pattern>/abc/xyz</url-pattern>

If we define any URL pattern with exact match method then to access the respective resource we have to provide an URL pattern at client address bar along with URL, it must be matched with the URL pattern which we defined in web.xml file.

Ex: http://localhost:8080/app1/abc/xyz Valid

http://localhost:8080/app1/xyz/abc Invalid

http://localhost:8080/app1/xyz Invalid

http://localhost:8080/app1/abc Invalid

Note: In general in web applications, we will prefer to use exact match method to define an URL pattern for a particular servlet when we have a requirement to access respective servlet independently.

2. Directory Match Method:

In Directory Match Method, we have to define an URL pattern in web.xml file, it must be prefixed with forward slash("/") and it must be terminated with "*".

Ex: <url-pattern>/abc/*</url-pattern>

If we define any URL pattern with this method then to access the respective resource from client we have to specify an URL pattern at client address bar it should match its prefix value with the prefix value of the URL pattern defined in web.xml file.

Ex: http://localhost:8080/app1/abc/xyz Valid

http://localhost:8080/app1/xyz/abc Invalid

http://localhost:8080/app1/abc Valid

http://localhost:8080/app1/abc/abc Valid

Note 1: In general in web applications, we will prefer to use directory match method to define an URL pattern when we have a requirement to pass multiple number of requests to a particular server side resource.

Note 2: In web applications we will use Filters to provide preprocessing and post processing to one or more number of servlets. In this context, when we send a request to respective servlet then container will bypass all the requests to the respective Filter.

To achieve this type of requirement we have to use directory match method to define an URL pattern for the respective Filter.

3. Extension Match Method:

In Extension Match Method, we have to define an URL pattern in web.xml file, it must be prefixed with "*" and it must be terminated with a particular extension.

Ex: <url-pattern>*.do</url-pattern>

If we define an URL pattern with this method then to access the respective server side resource from client we have to specify an URL pattern at client address bar, it may start with anything, but must be terminated with an extension which was specified in web.xml file.

Ex: http://localhost:8080/app1/login.do Valid

http://localhost:8080/app1/reg.do Valid

http://localhost:8080/app1/add.xyz Invalid

http://localhost:8080/app1/search.doo Invalid

Note 1: In general in web applications, we will prefer to use extension match method to define an URL pattern when we have a requirement to trap all the requests to a particular server side resource and to perform the respective server side action on the basis of the URL pattern name if we provided at client browser.

Note 2: If we design any web application on the basis of MVC then we have to use a servlet as controller, where the controller servlet has to trap all the requests and it has to perform a particular action on the basis of URL pattern name. Here to define URL pattern for the controller servlet we have to use extension method.

In web applications, web.xml file is mandatory or optional is completely depending on the server which we used.

In Apache Tomcat Server, web.xml file is optional when we have not used servlets, filters and so on. In Weblogic Server, web.xml file is mandatory irrespective of using servlets, filters and so on.

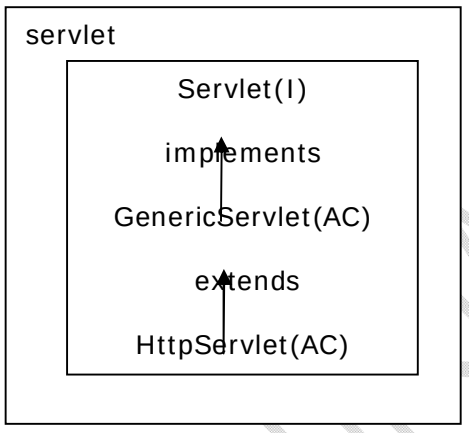
Up to servlets2.5 version[J2EE5.0] it is mandatory to provide web.xml file if we use servlets, listeners, filters and so on in our application. But in servlets3.0 version, there is a replacement for web.xml file i.e. Annotations, Annotations will make web.xml file is optional.

In web applications, it is not possible to change the name and location of the deployment descriptor because container will search for deployment descriptor with web.xml name under WEB-INF folder as per its predefined implementation.

Step 3: Prepare Web Resources:

If we want to prepare custom exceptions, threads and so on in Java applications then we have to use some predefined library provided by Java API.

Similarly if we want to prepare servlets in our web applications then we have to use some predefined library provided by Servlet API.



To design servlets Servlet API has provided the following predefined library as part of javax.servlet package and javax.servlet.http package.

Q: What is Servlet? and in how many ways we are able to prepare servlets?

Ans: Servlet is an object available at server machine which must implement either directly or indirectly Servlet interface.

As per the predefined library provided by Servlet API, there are 3 ways to prepare servlets.

1. Implementing Servlet interface:

In this approach, if we want to prepare servlet then we have to take an user defined class which must implement Servlet interface.

```
public class MyServlet implements Servlet
{ --------
  -------- }
```

2. Extending GenericServlet abstract class:

In this approach, if we want to prepare servlet then we have to take an user defined class as a subclass to GenericServlet abstract class.

```
public class MyServlet implements GenericServlet
{ --------
  -------- }
```

3. Extending HttpServlet abstract class:

In this approach, if we want to prepare servlet then we have to take an user defined class as a subclass to HttpServlet abstract class.

```
public class MyServlet implements HttpServlet
{ --------
  -------- }
```

Step 4: Start the Server:

There are 3 ways to start the server.

1. Execute either startup.bat file or Tomcat7 Service Runner available under bin folder of Tomcat server.
   C:\Tomcat 7.0\bin
2. Use system program monitor Tomcat
   Start ⟶ All Programs ⟶ Apache Tomcat 7.0 ⟶ Monitor Tomcat
3. Use Apache Tomcat System Service
   Start ⟶ Type services.MVC in search field ⟶ Select Apache Tomcat 7.0 ⟶ Select Start Service Icon.

Step 5: Access the Web Application:

There are 2 ways to access the web applications.

1. Open web browser and type the complete URL on address bar.
   http://localhost:8080/app1/servlet
2. Open web browser type the URL up to
   http://localhost:8080

If we do the above automatically the Tomcat Home Page will be opened, where we have to select Manager Applications, provide username and password in Security window and click on OK button.

If we do the above automatically list of applications will be opened, where we have to select the required application.

First Approach to Design Servlets (Implementing Servlet interface):

If we want to design a servlet with this approach then we have to take an user defined class it should be an implementation class to Servlet interface.

```
public interface Servlet {

    public void init(ServletConfig config)throws ServletException;

    public void service(ServletRequest req, ServletResponse res)throws ServletExcepption;

    public ServletConfig getServletConfig();

    public String getServletInfo();

    public void destroy();

}

public class MyServlet implements Servlet

{ -------

    ------- }
```

Where the purpose of init(_) method to provide servlets initialization.

Note: In servlets execution, to perform servlets initialization container has to access init(_) method. To access init(_) method container has to create ServletConfig object.

ServletConfig is an object, it will manage all the configuration details of a particular servlet like logical name of servlet, initialization parameters and so on.

In servlet, the main purpose of service(_,_) method is to accommodate the complete logic and to process the request by executing application logic.

Note: In servlet, service(_,_) method is almost all same as main(_) method in normal Java application.

When we send a request from client to server then container will access service(_,_) method automatically to process the request, where the purpose of getServletConfig() method is to get ServletConfig object at servlet initialization.

Where getServletInfo() method is used to return generalized description about the present servlet.

Where destroy() method can be used to perform servlet reinstantiation.

To prepare application logic in service(_,_) method we have to use the following conventions.

1. Specify response content type:

To specify response content type to the client we have to use the following method from ServletResponse.

public void setContentType(String MIME_TYPE)

where MIME_TYPE may be text/html, text/xml, image/jpeg, img/jpg and so on.

Ex: res.setContentType("text/html");

Note: The default content type in servlets is text/html.

When container encounters the above method then container will pick up the specified MIME_TYPE and container will set that value to content type response header in the response format.

When protocol dispatch the response format to client, before getting the response from response format body part first client will pick up content type response header value i.e. MiME_TYPE, client will prepare itself to hold the response as per the MIME_TYPE.

2. While executing the servlet we have to generate some dynamic response on response object, where to carry the dynamic response to the response object Servlet API has provide a predefined PrintWriter object internally.

To get the predefined PrintWriter object we have to use the following method from ServletResponse.

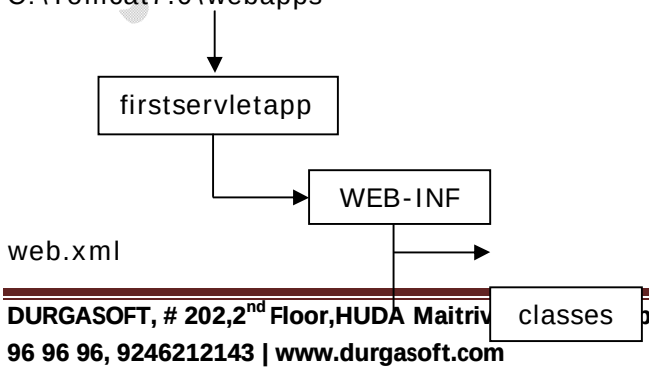
public PrintWriter getWriter()

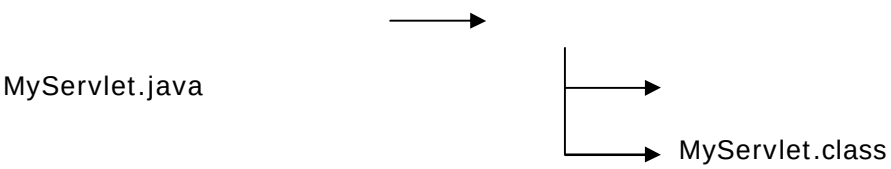
Ex: PrintWriter out=res.getWriter();

Steps to Design First Servlet Application:

1. Web application Directory Structure:

C:\Tomcat7.0\webapps





2. Prepare Deployment Descriptor(web.xml):

web.xml:-

```
<web-app>
	<servlet>
		<servlet-name>ms</servlet-name>
		<servlet-class>MyServlet</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>ms</servlet-name>
		<url-pattern>/ms</url-pattern>
	</servlet-mapping>
</web-app>
```

3. Prepare Servlet:

MyServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;


import javax.servlet.ServletResponse;

public class MyServlet  implements Servlet {

   public MyServlet() { }

public void init(ServletConfig config) throws ServletException {}

public void destroy() {}

public ServletConfig getServletConfig() {
	return null;
}
```

```
public String getServletInfo() {
        return null;
}

public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        out.println("<h1><center>hello</center></h1>");
}

}
```

To compile the above servlet we have to set classpath environment variable to servlet-api.jar provided by Tomcat server at C:\Tomcat7.0\lib\servlet-api.jar.

Ex: set classpath=%classpath%;C:\Tomcat7.0\lib\servlet-api.jar;

To access the above application we have to use the following URL on client address bar.

http://localhost:2020/firstservletapp/ms

Servlets Flow of Execution:

When we start the server the main job of container is to recognize each and every web application and to prepare ServletContext object to each and every web application.

While recognizing web application container will recognize web.xml file under WEB-INF folder then perform loading, parsing and reading the content of web.xml file.

While reading the content of web.xml file, if container identifies any context data in web.xml file then container will store context data in ServletContext object at the time of creation.

After the server startup when we send a request from client to server protocol will pick up the request tehn perform the following actions.

1. Protocol will establish virtual socket connection between client and server as part of the server IP address and port number which we specified in the URL.
2. Protocol will prepare a request format having request header part and body part, where header part will maintain request headers and body part will maintain request parameters provided by the user.
3. After getting the request format protocol will carry request format to the main server.

Upon receiving the request from protocol main server will check whether the request data is in well-formed format or not, if it is in well-formed then the main server will bypass request to container.

Upon receiving the request from main server container will pick up application name and resource name from request and check whether the resource is for any html page or Jsp page or an URL pattern for a servlet.

If the resource name is any html page or Jsp page then container will pick up them application folder and send them as a response to client.

If the resource name is an URL pattern for a particular servlet available under classes folder then container will go to web.xml file identifies the respective servlet class name on the basis of the URL pattern.

After identifying the servlet class name from web,xml file container will recognize the respective servlet .class file under classes folder then perform the following actions.

Step 1: Servlet Loading:

Here container will load the respective servlet class byte code to the memory.

Step 2: Servlet Instantiation:

Here container will create a object for the loaded servlet.

Step 3: Servlet Initialization:

Here container will create ServletConfig object and access init(_) method by passing ServletConfig object reference.

Step 4: Creating request and response objects(Request Processing):

After the servlet initialization container will create a thread to access service(_,_) method, for this container has to create request and response objects.

Step 5: Generating Dynamic response:

By passing request and response objects references as parameters to the service(_,_) method then container will access service(_,_) method, executes application logic and generate the required response on response object.

Step 6: Dispatching Dynamic response to Client:

When container generated thread reaching to the ending point of service(_,_) method then container will keep the thread in dead state, with this container will dispatch the dynamic response to main server from response object, where main server will bypass the response to the protocol.

When protocol receives the response from main server then protocol will prepare response format with header part and body part, where header part will manage all the response headers and body part will manage the actual dynamic response.

After getting response format protocol will carry that response format to the client.

Step 7: Destroying request and response objects:

When the response is reached to the client protocol will terminate the virtual socket connection between client and server, with this container destroy the request and response objects.

Step 8: Servlet Deinstantiation:

When container destroy request and response objects then container will go to the waiting state depends on the container implementation, if container identifies no further request for the same resource then container will destroy servlet object.

Note: In servlet execution, container will destroy ServletConfig object just before destroying the servlet object.

Step 9: Servlet Unloading:

After servlet deinstantiation container will eliminate the loaded servlet byte code from operational memory.

Step 10: Destroying ServletContextobject:

In the above servlet life cycle, all the objects like request, response and ServletConfig are destroyed before servlet deinstantiation, but still Servlet object is available in memory.

In general ServletContext object will be destroyed at the time of server shut down.

Drawbacks of First Approach:

To design servlets if we use this approach we have to provide implementation for each and every method declared in Servlet interface irrespective of the actual application requirement.

The above approach will increase burden to the developers and it will increase unnecessary methods in web applications.

To overcome the above problem we have to use an alternative i.e. GenericServlet.

Second Approach to Design Servlets (Extendiging GenericServlet abstract class):

If we want to design servlets by using this approach then we have to take an user defined class which must be a subclass to GenericServlet abstract class.

```
public abstract class GenericServlet implements Servlet, ServletConfig, Serializable {

    private transient ServletConfig config;

    public void init(ServletConfig config)throws ServletException {
```

```
        this,config=config;
        init();
    }
    public void init()
    {   }
    public abstract void service(ServletRequest req, ServletResponse res)throws SE, IOE;
    public ServletConfig getServletConfig() {
      return config;
    }
    public String getServletInfo() {
        return null;
    }
    public void destroy()  {  }
}
public class MyServlet extends GenericServlet
{ -------
------- }
```

From the above predefined implementation of GenericServlet abstract class,

1. GenericServlet is an idea came from Adapter Design Pattern.
2. GenericServlet predefined abstract class has implemented Serializable interface so that all the GenericServlet objects(subclass objects of GenericServlet abstract class) are eligible for Serialization and Deserialization by default.
3. It is possible to serialize GenericServlet objects, but config predefined reference variable will not be participated in serialization and deserialization because config reference variable has declared as transient.
4. In GenericServlet abstract class, init(_) method is overloaded method.
5. In GenericServlet abstract class, still service(_,_) method is an abstract method.

Q: What is the requirement to override init(_,_) method in servlet applications?

Ans: In general as part of servlets design we will use service(_,_) method to provide application logic. In some cases, application logic may include the actual business logic and its prerequirement.

If we provide both prerequirement code and business logic within service(_,_) method then the performance of the application will be reduced because container will execute business logic and its prerequirement code for every request send by the client, but as per the convention it is sufficient to execute prerequirement code only one time.

In the above context, to improve the performance of servlet applicationwe have to provide the actual business logic in service(_,_) method and the prerequirement code in init(_) method because container will execute init(_) method only one time, but service(_,_) method for every request.

Note: In servlet applications, always it is suggestible to override init() method(second init() method) but if our prerequirement code may include any dat from ServletConfig object then it is suggestible to override init(ServletConfig config) method.

----------Application by using GenericServlet----------

Genericservletapp:-

web.xml:-

```
<web-app>
	<servlet>
		<servlet-name>GenericDemo</servlet-name>
		<servlet-class>GenericDemo</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>GenericDemo</servlet-name>
		<url-pattern>/gen</url-pattern>
	</servlet-mapping>
</web-app>
```

GenericDemo.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.GenericServlet;


import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class GenericDemo extends GenericServlet {

	public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out=response.getWriter();
		out.println("<h1><center>hello</center></h1>");
	}

}
```

The main difference between first approach and second approach of designing servlets in the point of flow of execution, in first approach container will execute only one init(_)

method in servlet initialization, but in second approach container will execute two init() methods in servlet initialization.

Third Approach to Design Servlets (Extendiging HttpServlet abstract class):

Q: Whar are the differences between GenericServlet and HttpServlet?

Ans: 1. GenericServlet is protocol independent, but HttpServlet is http protocol dependent.

2. In case of GenericServlet, container will execute only service(_,_) method for any type of protocol request provided by the client, but In case of HttpServlet, container will execute a separate method on the basis request type which will be specified by the client.

Ex: If we specify Get request type at client browser then HttpServlet is able to execute doGet(_,_) method, for POST request type HttpServlet will execute doGet(_,_) method.

3. GenericServlet is not very good compatible with protocols, but HttpServlet is very good compatible with http protocol.

4. GenericServlet will not implement any specific protocol at server side, but HttpServlet has implemented http protocol at server side.

5. GenericServlet will not give any option to the developers to specify different types of requests at client browser, but HttpServlet will provide flexibility to the developers to specify different types of requests at client browser.

If we want to prepare servlets by using HttpServlet abstract class then we have to take an user defined class which must be a subclass to HttpServlet abstract class.

```
public abstract class HttpServlet extends GenericServlet {
   public void service(ServletRequest req, ServletResponse res)throws SE, IOException {
      HttpServletRequest hreq=(HttpServletRequest)req;
       HttpServletResponse hres=(HttpServletResponse)res;
       service(hreq, hres);
   }

   public void service(HttpServletRequest hreq, HttpServletResponse hres)throws SE, IOE {
```

```
        String method=hreq.getMethod();
        if(method.equals("GET")) {
             doGet(hreq, hres);
         }
         if(method.equals("POST")) {
              doPost(hreq, hres);
         }
    }
    public void doGet(HttpServletRequest hreq, HttpServletResponse hres)throws SE, IOE
    {  }
    public void doPost(HttpServletRequest hreq, HttpServletResponse hres)throws SE, IOE
    {  }
}
public class MyServlet extends HttpServlet
{ ------
  ------ }
```

Note: In case of HttpServlet, we have to override either doGet(_,_) method or doPodt(_,_) method and son on doXxx(_,_) method with our web application logic on the basis of request type which we provide at client browser.

----------Application by using HttpServlet----------

httpservletapp:-

web.xml:-

```
<web-app>
	<servlet>
		<servlet-name>HttpDemo</servlet-name>
		<servlet-class>HttpDemo</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>HttpDemo</servlet-name>
		<url-pattern>/http</url-pattern>
	</servlet-mapping>
</web-app>
```

HttpDemo.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

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

public class HttpDemo extends HttpServlet {

   public HttpDemo() {

   }

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out=response.getWriter();
		out.println("<h1>hello this is http</h1>");
	}

}
```

The main difference between GenericServlet flow of execution and HttpServlet flow of execution is in GenericServlet flow of execution container will execute only one service(_,_) method to process the request, but in case of HttpServlet container will execute

service(ServletRequest, Servletresponse), service(HttpServletRequest, HttpServletresponse), doXxx(HttpServletRequest, HttpServletresponse) on the basis of request type in order to process the request.

Q: Is it possible to override service(_,_) method in HttpServlet?

Ans: In HttpServlet, it is possible to override any of the service(_,_) methods, doXxx(_,_) methods, but always it is suggestible to override doXxx(_,_) methods on the basis of request types, it is not at all suggestible to override service_,_) methods.

If we override service(_,_) method in HttpServlet then container will execute user provided service(_,_) method, but container is unable to execute predefined service(_,_) method so that it is not possible to reach doGet(_,_) method or doPost(_,_) method and so on.

Q: Is it possible provide both constructor and init() method in a single servlet?

Ans: Yes, it is possible provide both constructor and init() method in a single servlet.

If we provide a static block, a constructor, init(_) method, doXxx(_,_) method and destroy()method within a single servlet then container will execute static block at the time of servlet loading, constructor at the time of servlet instantiation, init() method at the time of performing servlet initialization, doXxx(_,_) method at the time of request processing and destroy() method at the time of servlet deinstantiation.

Ex: public class MyServlet extends HttpServlet {

```
    static {
       System.out.println("Servlet Loading");
    }
    public MyServlet() {
        System.out.println("Servlet Instantiation");
    }
    public void init() {
        System.out.println("Servlet Initialization");
    }
```

public void doGet(HttpServletRequest hreq, HttpServletResponse hres)throws SE, IOE {

```
        System.out.println("Request Processing");
    }

    public void destroy() {
        System.out.println("Servlet Deinstantiation");
    }
}
```

If we send a request to above servlet then are able to see the following output on Server prompt

Servlet Loading

Servlet Instantiation

Servlet Initialization

Request Processing

Servlet Deinstantiation (when we close the server).

If we want to provide any constructor in servlet class then that constructor should be public and zero argument because container will search and execute public and zero argument constructor as part of servlet instantiation to create servlet object.

If we provide parameterized constructor without zero-argument constructor then container will raise an Exception like

javax.servlet.ServletException : Error-Instantiating Servlet class MyServlet with the root case java.lang.InstantiationException:MyServlet.