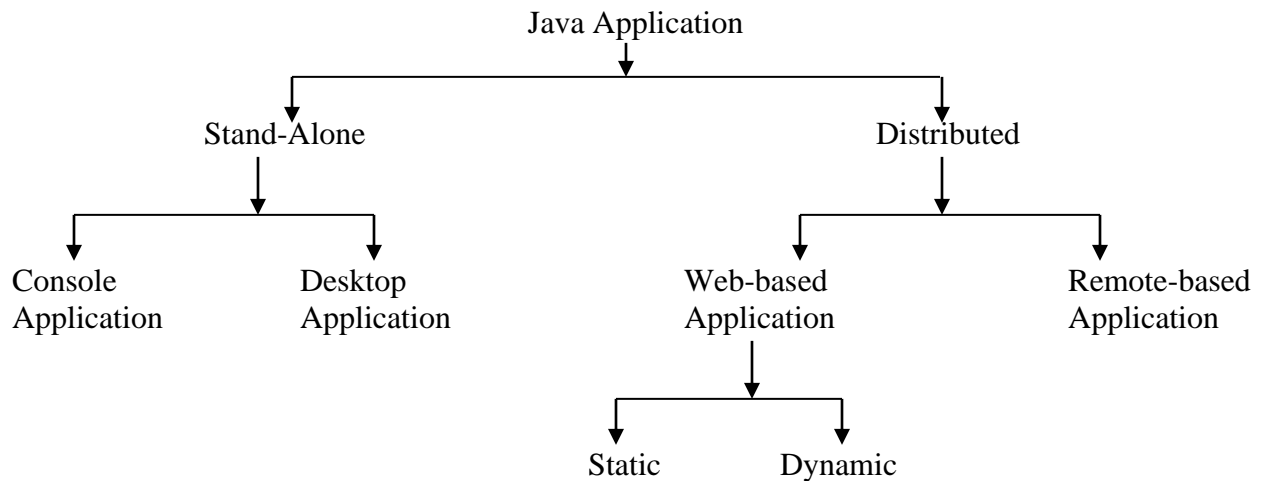


# [SERVLET]

Thursday, June 19, 2014

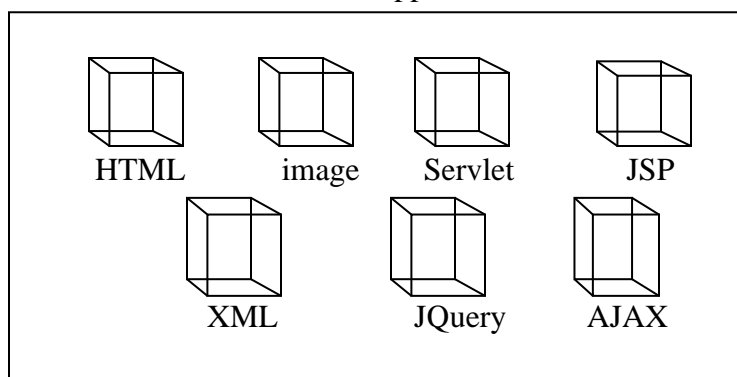


- Stand-alone applications of java are for providing service to one customer at a time.
- A stand-alone application is a client side application.
- We call stand-alone application as network-disabled applications because they are not accessible through network.
- A stand-alone application with CUI is as console application and a stand-alone application with GUI is a desktop application.
- Web applications are introduced to provide an application service to multiple clients across internet.

A Web application is a server side application; it runs on a server and provides its services to many clients across the internet.

- We call a web application as a group of web resources. Here a web resource can be a static or dynamic file.
- A web application is a group of passive and active web resources (passive means static and active means dynamic).

## Web Applications



- Web applications are divided into 2 types
  - (1) Presentation-Oriented or Static Web application.
  - (2) Service-Oriented or Dynamic Web application.
- Static web applications are created using HTML, images, java script files etc.
- Static applications with only concentrate in presentation.
- Dynamic web applications are created using all HTML, images, jsp, servlet, php etc.
- Dynamic web applications will concentrate on providing some service to the client.
- For example, mostly many online tutorial websites are static web applications
- For example, websites like gmail.com, yahoo.com are dynamic application because they offer mailing and chatting services to end-users.

**Friday, June 20, 2014**

Web applications are created using both client side and server side technology.

- Client side technologies are going to run on a web browser and server side technologies are run a web server.
- For Example- HTML pages, java script files are going to run on browser. So they are client side technologies.
- For example- Servlets and jsps are going to run on a server. So, they are a server side technologies.

A Web application which is running on server can be accessed in two ways.

1. Directly by sending request from browser.
2. Constructing a java program

Web browser is a **thin** client and a java program is a **thick** client.

If we create a java program to call a web application then we need java software in client system to run the client application. But, in case of browser there is no need of java software at client systems so , we always prefer a browser.

Web application directory structure

Each java application should follow a standard directory structure given by SUN.

- The important directories of a web applications are:
  1. Root directory
  2. Root/WEB-INF directory
  3. Root/WEB-INF/classes directory
  4. Root/WEB-INF/lib directory

Root directory name can be user defined , but remaining directory name are fixed.

> SUN Microsystems given a directory structure for web applications in order to make a web application as server independent.

> Each java web application follows a principle Write Once Deploy Anywhere(WODA)

Diagram

Root

WEB-INF

Classes(\*.classes)

Lib(\*.jar)

web.xml (Deployment Descriptor)  
\*.html, js, jsp, gif

## Container

- Generally, a class which providing run time support for other classes is called a container.  
Example-1: In AWT, a frame is a class which provides runtime support for Button, so frame is a container.  
Example-2: we created two classes A without main and B with main then B classes provides runtime support for A class. So, B is a container.  
Example-3: we create an applet and we run in browser. so, browser is a container for an applet.
- In web applications, we create Servlets , nothing but java classes. There is a class at server to provide runtime support for our servlets. That class is called a web container.

Diagram:

Web server  
Web Container  
servlet

### **Saturday, June 21, 2014**

Apart from providing run-time environment for Servlets, a container also offers some additional services.

The important services are

- Life cycle management
- Multi-threading supports
- Security
- Chaining

Life cycle management means, a container takes care about instantiation, initialization services and destroy phrases of a Servlet.

When a request comes to a Servlet, then the container creates a thread and after its work, a container will destroy it. It means a container provides multi-threading support.

A container will verify the credentials of a client and based on credentials either it allow the request to Servlet or denies the request. It means container provides security. Sometime a request should be passed on one Servlet to another. A container will performs this chaining.

Types of web-container:

- Stand-alone container
- In-process container
- Out-process container

If a container directly installed without a server. It is a stand-alone container.

If a container runs within the process of a server then it is called an in-process container.

If a container runs in a separate process of a server then it is called an out-of-process container.

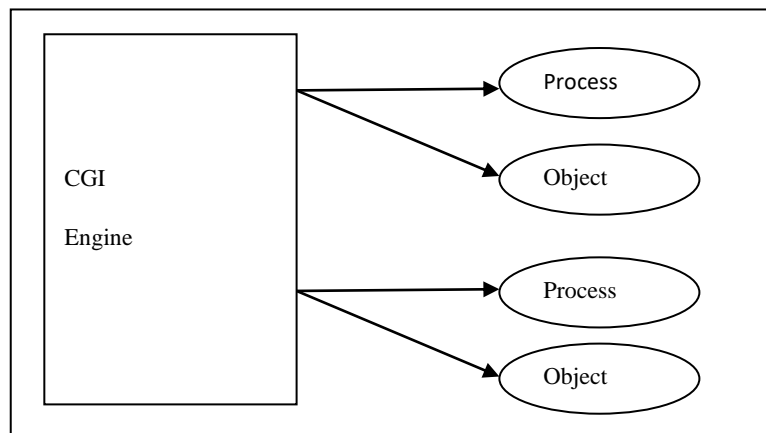
For example: if we download and install Apache Tomcat then Tomcat acts as a stand-alone container.

If we install GlassFish or JBoss servers, then along with these servers Tomcat also starts within the same process. So in this case, Tomcat is acting as an in-process container.

**Monday, June 23, 2014**

Servlet technology reduces the burden on the server when compared with CGI because Servlet technology follows a thread-model, but CGI technology follows a process-model.

- Threads are lightweight objects compared to processes. So, Servlet is a lightweight technology compared to CGI.



Q )What is a Servlet?

Def 1: A Servlet is a Java class, it runs on a server and provides services to multiple clients across the internet.

Def 2: A Servlet is a platform-independent Java class, which extends the functionality of a web server.

Q ) why a Servlet class is created?

- In a web application, if there is a requirement to generate dynamic content onto the browser (client) then a Servlet class is created.

Servlet API:

Servlet API contains three packages.

- javax.servlet
- javax.servlet.http
- javax.servlet.annotation (added in Servlet 3.x)

Each package consists of a group of classes and interfaces to develop a Servlet class.

Q) How a Servlet class is created?

> Every Servlet class created by the developers will implements Servlet interface of Servlet api.

> A Servlet class indirectly implements Servlet interface by extending GenericServlet or by extending HttpServlet class.

For example:

Public class MyServlet extends javax.servlet.Servlet

```
{  
}
```

Or

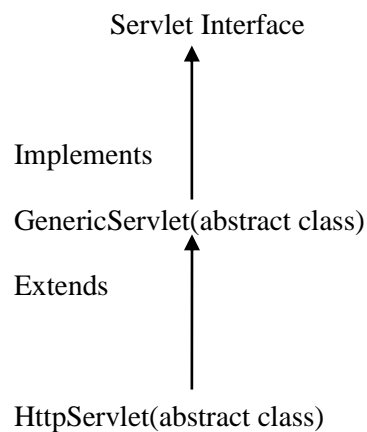
Public class MyServlet extends javax.servlet.GenericServlet

```
{  
}
```

Or

Public class MyServlet extends java.servlet.http.HttpServlet

```
{  
}
```



- Generic Servlet is an abstract class, because it contains one abstract methods called service.
- HttpServlet is also an abstract class, but it doesn't contain any abstract methods.
- In java an abstract class may or mayn't contain abstract methods.

Q) What is a difference between an applet and a Servlet?

Applet:

- Applet runs on a web browser.
- Applet increases the functionality of a browser.
- Applet can provide service to one client at time.
- Applets are visible, because of width and height.

Servlet:

- Servlet runs on a web server.
- Servlet increases the functionality of a server.
- Servlet can provide service to multiple clients at time.
- Servlet are invisible.

**Tuesday, June 24, 2014**

**Life cycle stages of servlet:**

- Does not exist
- Instantiated
- Initialized
- Service
- Destroy

-----diagram-----

- The first stage in the life cycle of a servlet is doesn't exist. It means, an object for a servlet is not yet created by a web container.
- When a first request is arrived for a servlet or when a servlet is deployed in a server then web container creates an object of a servlet. It means a servlet is instantiated.
- If a servlet is instantiated immediately whenever it is deployed in a server then it is called "early loading".
- If a servlet is instantiated when a first request is arrived for it then it is called "lazy loading".
- After a servlet object is created a web container initializes it by calling constructor then followed by init();
- After a servlet object is initialized then it will provide service to all the request made by the client.
- A web container invokes a service() to provide the service to a client.
- After a servlet has provided service to client request, whenever a servlet object is no longer need then container will destroy that object.
- To destroy the container called destroy().
- Finally, once a servlet object is destroyed then a servlet will reach to doesn't

**Life cycle methods:**

A servlet has three life cycle method and these are given javax.servlet.Servlet interface.

The syntax of life cycle methods are,

1. public void init(ServletConfig config) throws ServletException
  2. public void service(ServletRequest request, ServletResponse response) throws ServletExceptionIOException
  3. public void destroy()
- > init () and destroy() life cycle methods are called for once.
  - > service () is called for each request
  - > diagram>>>

### **Request Flow in a servlet:**

.....diagram.....

Step-1: when a client made a request from browser to a servlet then it is first stopped at server.

- A server will delegate the request to container.
  - A container prepares a request, response and thread objects for a given request.
  - A container will assign request, response object to thread.
  - Now thread calls service () by passing request and response objects as parameters.
  - The logic defined in service () will process the given request and generates the response data.
  - A container collects the response of a Servlet.
  - A container prepared a dynamic web page for the response of a Servlet.
  - A container will send the dynamic web page to a server.
- Note: A container will remove request, response and thread objects automatically whenever a web page is sent to a server.
- Finally, a server will transfer a webpage as a response to the browser.

**Wednesday, June 24, 2014**

### **Servlet Configuration:**

Servlets in a server are maintained by a web container.

- A web container is pre-defined program that comes along with a server.
- By default a container doesn't know anything about Servlets that created by developers.
- In order to tell the container about servlet, we configure that servlet in **deployment descriptor file**.
- A descriptor file name should be "web.xml".
- This web.xml file is a mediator file from a program to a container, in order to talk with the container.

Q) What is configuration?

Ans- If the programmer writing something in the xml file then it is called configuration.

- While configuring a servlet in web.xml file, we configure three names to a servlet

1. Alias name
  2. Fully qualified class name
  3. url pattern
- One web application can contain multiple Servlets. A web container decides whether a request is made by a client to which servlet, container uses url pattern given in the request.
  - A container decides a servlet to handle a given request, based on its url pattern.
  - Alias name is used by the container, in order to forward a request to a particular servlet class.
  - We say that alias name is a mediator in connecting a container and a servlet class.
  - In order to configure the above three names of a servlet we use two entries(tags) in the deployment descriptor file(web.xml).
    1. <servlet>
    2. <servlet-mapping>

web.xml

```
<web-app>
  <servlet>
    <servlet-name>alias name</servlet-name>
    <servlet-class>Fully qualified class name</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>alias name</servlet-name>
    <url-pattern>/some pattern</url-pattern>
  </servlet-mapping>
</web-app>
```

**Example:**

web.xml

=====

```
<web-app>
  <servlet>
    <servlet-name>sone</servlet-name>
    <servlet-class>pack1.Servlet1</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>sone</servlet-name>
    <url-pattern>/srv1</url-pattern>
  </servlet-mapping>
</web-app>
```

Q) How a request reached to a servlet through url?

Ans:

- .....Diagram.....



**[Tuesday, June 26, 2014]**

### **Installing Tomcat Server:**

- Visit [tomcat.apache.org](http://tomcat.apache.org) and download tomcat7 service installer.
- By default tomcat server is installed at c:\Program File\Apache Software Foundation
- By default Tomcat server is installed on port 8080. But oracle Database Server comes with Oracle HTTP server on port 8080. So a port number clash occurs.
- To change the port number of Tomcat, to the following  
Open Tomcat\conf\server.xml  
Change port number of connector tag from 8080 to 1414.  
Note: Port no ranges from 1-65536
- Tomcat server can be started in the following ways
  1. Go to tomcat7\bin folder and double on tomcat7
  2. Go to tomcat7\bin and double click on tomcat7w
  3. Go to system tray of icons, right click on tomcat icon and start service.

### **Steps to create a web application with a Servlet:**

Step-1: create directory structure like the following

Step-2: create WelcomeServlet.java

```
//WelcomeServlet.java
import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
public class WelcomeServlet extends GenericServlet
{
    public void service(ServletRequest request, ServletResponse response) throws
    ServletException, IOException
    {
        PrintWriter out=response.getWriter();
        out.println("Welcome to Servlet");
        Date date=new Date();
        out.println("<br>");
        out.println("Today date is "+date);
        out.close();
    }
}
```

Step-3: configure Servlet in web.xml

```
<!--web.xml-->
<web-app>
    <servlet>
        <servlet-name>welcomes</servlet-name>
        <servlet-class>WelcomeServlet</servlet-class>
    </servlet>
```

```

<servlet-mapping>
  <servlet-name>welcomes</servlet-name>
  <url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>

```

Step -4: Servlet-api is a part of java EE(JEE), so this api is unknown to java compiler.

- The api's of Java EE comes along with servers in the form jar files.
- So, we need to set a server provided jar file in class path to compile our Servlet.
- In case of tomcat, the jar file is Servlet-api.jar and it exists tomcat\lib folder. So we need to set classpath like the following.

- D:\WelcomeServlet\WEB-INF\classes>set classpath= C:\Program Files\Apache Software Foundation\Tomcat 7.0\lib\servlet-api.jar;.
- Javac WelcomeServlet.java

Step-5: Deploy the web application into tomcat server.

- Copy WelcomeApp folder into tomcat\webapps folder.

Step-6: Start tomcat server

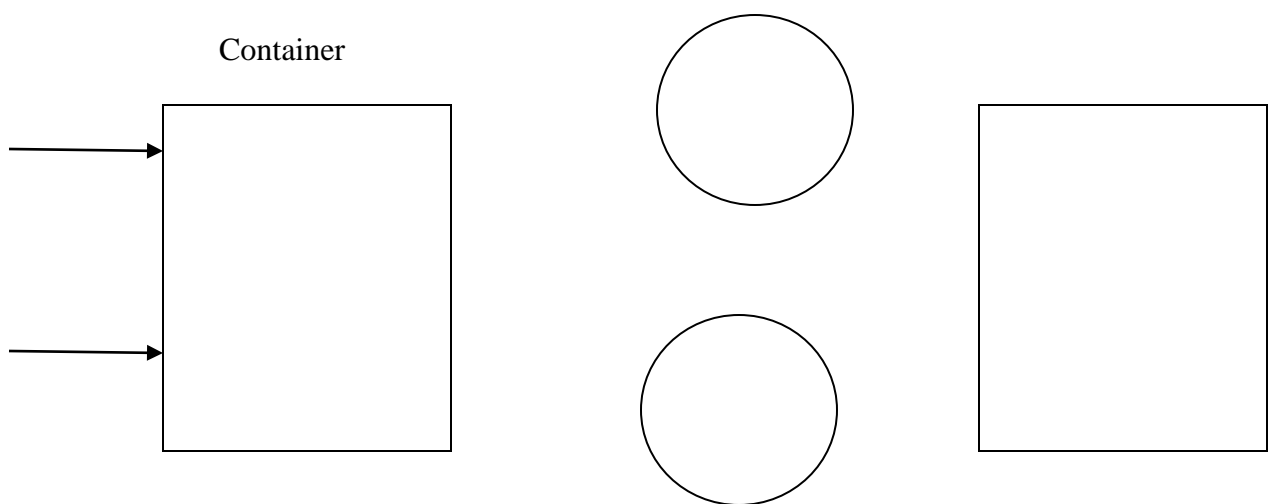
Step-7: Open the browser and type the following request.

**[Friday, June 27, 2014]**

**Q) What is the use of request and response object in a Servlet?**

Ans :-

- If any input values are send with request to a Servlet then container stores the input in request object. From that request object a Servlet will get the input.
- If a Servlet wants to write any output to the browser then Servlet will put output into response object. Later container collects output from response object and then it will be send to a browser.



**Q) what is the need of PrintWriter object?**

Ans-

- If a Servlet wants to write output into a response object then PrintWriter object is required.

- The PrintWriter object for a Servlet will be supplied by response object only. so, to get PrintWriter object from response object we call getWriter() method.

```
PrintWriter out=response.getWriter();
```

It is object name and it can be any name.

**Q) Can we create a PrintWriter() object explicitly in a Servlet class or not?**

Ans:-

- If we directly create PrintWriter class object then it will write output on console of server, not in response object.

```
PrintWriter out=new PrintWriter(System.out);
out.println("Welcome to Servlet");
```

**Q) Why every Servlet class must be a public class?**

Ans:-

- An object of Servlet class is created by a container.
- A Servlet class becomes visible to a container, if that Servlet class is public.

**Q) What happens if a Servlet class is not public?**

Ans:-

- If a Servlet class is not a public class then it can be compiled and it can be deployed in a server. But, when a request is given from browser then container throws the following exception

1. javax.servlet.ServletException: Error instantiating Servlet class

**Q) Sending input values to a Servlet?**

Ans:-

- While sending a request to a Servlet input values can be send along with request, in two ways
1. By adding input values to the url in address bar
  2. By designing html page with form tags.
- If we add input values to url of address bar then those values are called **query string**.
  - The url and query string are separated with **?** symbol.
  - The query string contains one or more parameters, where a parameter is nothing but key=value pair.
  - One parameter and another parameter in a query string are separated with **&** symbol.

Example:

<http://localhost:2014/App/serv1 ? username=sathya&password=java>

url

query string

- In the above query string there are two parameters and key are username and password.
- A container will put the input values in request object
- A Servlet class will read the input values by calling getParameter().

Example:

```
String s1=request.getParameter ("username");
```

```
String s2=request.getParameter ("password");
```

**[Tuesday, July 01, 2014]**

**[Wednesday, July 02, 2014]**


- If a web application contain index.html page and in web.xml if login.html is configured as a welcome file then login.html is considered as welcome page.

- If a request comes for an html page for a server from a browser then the request is handled by server.
- If a request comes for a **static** resource then **server** will handle the request and if a request is came for a **dynamic** resource then **container** will handle the request.

#### Steps to develop a login application in Eclipse IDE:

- Eclipse is an open source java/Java EE ide from eclipse foundation.
- Myeclipse is not open source ide but on top of eclipse.
- The difference eclipse and myeclipse is, myeclipse had build in support for developing framework applications. But, in eclipse we need to add xml's ,jar's etc explicitly to develop framework applications.
- Eclipse can be downloaded from eclipse from [www.eclipse.org/downloads](http://www.eclipse.org/downloads).

Step-1:

Start eclipse IDE and enter some workspace(c:\ServletsWorkspace) 

- Click on file → New → dynamic web project → Project name(LoginProject) →  
Select dynamic web module version(2.5) → Finish

Step-3:

Right click on WebContent → New → HTML file → enter file name(Login.html) → finish

```
//login.html
<center>
<h2>Login Credentials</h2>
<hr width="50%">
<form action="loginsrv">
Username:<input type="text" name="uname"/>
Password:<input type="password" name="pwd"/>
<input type="submit" value="SUBMIT"/>
</form>
</center>
```

Step-4: Right click on project name → Build path → Configure Build path → libraries → Add external jars Button → Servlet-api.jar → Ok

Step-6:

Add the following code in service method of LoginServlet

```
Publicvoid service(ServletRequest request,ServletResponse response)throws
ServletException,IOException
{
    PrintWriter out=response.getWriter();
    String str1=request.getParamater("uname");
    String str2=request.getParamater("pwd");
    if(str1.equals("sathya") && (str2.equals("java")))
    {
        out.println("<h1> Login Success</h1>");
    }
    else
    {
        out.println("<h1> Login Failed</h1>");
    }
}
```

Note:

The IDE automatically configures our Servlet in web.xml

Step-7:

Click on window menu → show view → servers → right click on servers view → New → server → expand apache → select tomcat 7.0 → Next → click on browser and select tomcat 7 → next → finish.

Step-8: Right click on project name → Run as → Run on server → select tomcat → next → finish

**[Thursday, July 03, 2014]**

Q) can we define more than one <url-pattern> for a Servlet?

Ans- yes, we can repeat <url-pattern> for multiple times in <Servlet-mapping> tags of web.xml.

```
<Servlet>
  <Servlet-name>login</Servlet-name>
  <Servlet-class>LoginServlet</Servlet-class>
</Servlet>
<Servlet-mapping>
  <Servlet-name>login</Servlet-name>
  <url-pattern>/loginsrv</url-pattern>
  <url-pattern>/login</url-pattern>
  <url-pattern>/srv1</url-pattern>
</Servlet-mapping>
```

### **Early loading of a Servlet:**

By default, a web container loads a Servlet lazily. It means a Servlet object is created by container when first request is arrived for that Servlet.

- While developing MVC(Model-View-Controller) projects using struts framework or spring MVC framework etc, a Servlet object is needed immediately whenever an application is deployed in a server.
- If a Servlet object is created before a first request is arrived then it is called early loading.
- In order to inform a container that load a Servlet early, we configure a tag <load-on-startup> under <Servlet> tag.

### **<load-on-startup>:**

Allows a positive integer as a value. <load-on-startup> works like the following

1. If the value is -ve then container ignores load on startup and creates a Servlet object when first request is arrived. But container doesn't throw any exception.
2. If more than one Servlet contains load on startup then the objects are created based on priority order. The order is decided based on load on startup value.
3. Low value get high priority and high value gets low priority.
4. If more than one Servlet contains equal load on startup then the order will be decided by a container.

```
<Servlet>
  <Servlet-name>sone</Servlet-name>
  <Servlet-class>Servlet1</Servlet-name>
  <load-on-startup>10</load-on-startup>
</Servlet>
```

### **Request parameter methods:**

```
→getParameter()
→getParameterValues()
→getParameterNames()
→getParameterMap()
```

- If a request parameter has a single value then we call getParameter().
- If a parameter has multiple values then we call getParameterValues().

Username	<input type="text"/>
Hobbies	<input type="checkbox"/> Watching TV <input type="checkbox"/> Sleeping <input type="checkbox"/> Music <input type="checkbox"/> Art
	<input type="submit" value="SUBMIT"/>

HTML PAGE

```
String username=request.getParameter("uname");
String[] hobbies=request.getParameterValues("hobby");
```

- If a Servlet developer don't know parameters names of a request then all parameters names can be collected at once by calling `getParameterNames()`.
- `getParameterNames()` stores all parameters in collection object of type `Enumeration` and returns a reference to the `Enumeration` objects.
- If a Servlet we can get one by one name from `Enumeration` object and then we can read the value of the parameter.

```
Enumeration e=request.getParameterNames();
```

→


```
While(e.hasMoreElements())
{
String name=(String)e.nextElement();
String value[]=request.getParameterNames(name);
}
getParameterMap():
```

- It returns a `Map` object which contains `parameterNames` as "keys" and `parameterValues` as values.
- If a parameter contain more than one value then its values are stored in the form of `String` array.

- This `getParameterMap()` is a very rarely used method.  
`Map map=request.getParameterMap();`

Key	Value

**[Friday, July 04, 2014]**

Q) Can we make a Servlet class as final class or not?

Ans- yes, we can make a java class as final when we want to restrict the class to participate in inheritance.

Q) Can we define an user defined method in a Servlet or not?

Ans- yes, but a container doesn't call the user defined method, it is not a life cycle method.

- We can explicitly call user-defined method from one of the life cycle method.

Example-

```
public final class LoginServlet extends GenericServlet
{
    public void service(ServletRequest request, ServletResponse response) throws
    ServletException, IOException
    {
        m1();
    }
    public void m1()
    {
    }
}
```

Q) Can we define `main()` in a Servlet class or not?

Ans- Yes, `main()` is a also not a life cycle method. So, it is not called by container.

- We can call the `main()` explicitly from one of the life cycle methods of a Servlet.

Q) Is a Servlet object is serialized object or not?

Ans- Yes, our Servlet class extends `GenericServlet` and `GenericServlet` interface implements `Serializable` interface.

- In java if super class implements `Serializable` then its subclasses also implements `Serializable` indirectly. So, subclass object also becomes a serialized object.

**Life cycle and non-life cycle `init()` methods of Servlet:**

- In `GenericServlet` class there are two `init()` methods.
  1. Lifecycle

## 2. Non life-cycle

- Lifecycle init() method is came from Servlet interface and non-lifecycle init() method is created by GenericServlet class.
- We can differentiate life cycle and non life cycle init() methods with the help of parameter. Life cycle init() contains ServletConfig object as a parameter and non life cycle method doesn't contain any parameters.  
Public void init(ServletConfig config) throws ServletException → life cycle

Public void init() throws ServletException → non-life cycle

- In GenericServlet class life cycle init() method contain code and non-life cycle init() is empty.
- When we are creating our Servlet class, we have two options to define initialization logic of our Servlet.
  1. We can directly override life cycle init() method
  2. We can override non-life cycle init() method
- It is recommended to override non-life cycle init(), because in super class it doesn't contain any logic.
- If we override life cycle init() method then our Servlet class loose the logic defined by super class in the life cycle init(). So, it is always recommended to override non-life cycle init() in a Servlet class.

Example :

```
Public class MyServlet extends GenericServlet
{
    @Override
    Public void init( ) throws ServletException
    {

    }

    @Override
    Public void service(ServletRequest request, ServletResponse response) throws
    ServletException, IOException
    {

    }
}
```

**[Saturday, July 05, 2014]**

**Extending HttpServlet class:**



**[Sunday, July 06, 2014]**

**Q) When doGet() and doPost() methods are Overridden in a Servlet?**

Ans- There are two cases, where we override both doGet() and doPost() methods in a Servlet.

1. When we want to define separate logics for get request and post request.
2. When a Servlet programmer doesn't know a request from browser comes with get or post.

```
public class MyServlet extends HttpServlet
{
    Public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        //Login-1
    }
    Public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        //Login-2
    }
}
```

➤ If we don't know whether a request comes from with get() or post() from browser and if we want to define same logic for both request then again we have two options.

1. Define the logic in one method and call it from another method.
2. Create an user-**defined** method for the logic and call it from doGet() and doPost() methods.

**Option-1:**

```
public class MyServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        doPost(request,response);
    }
    Public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        //Login-2
    }
}
```

**Option-2:**

```
public class MyServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        processRequest(request,response);
    }
}
```

```

    }
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        processRequest(request,response);
    }
    public void processRequest(HttpServletRequest request,HttpServletResponse response) throws
ServletException, IOException
    {
        //login
    }
}

```

**Q) What happens if a Servlet doesn't contain doGet() for a GET request and doPost() for POST() request?**

Ans- If you don't override doGet() for a get request and doPost() for a post request then super class(HttpServlet) methods are called.

➤ The super class methods send Http Status 405 error to the browser.

**Q) How many init() and service() methods are present in GenericServlet?**

Ans- Two init() and one service().

**Q) How many init() and service() methods are present in HttpServlet?**

Ans- Two int() and two service().

Note:

Non-life cycle service method and seven doXxx() methods are protected methods in HttpServlet. So, while overriding these methods we can use either protected or public modifier.

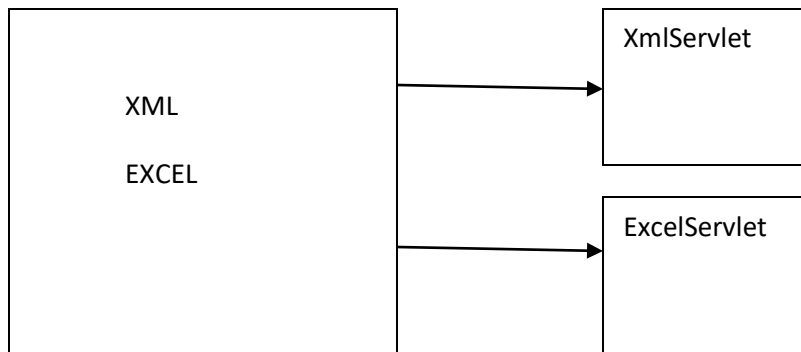
➤ While overriding a super class method in a subclass, either we can use same level of access modifier or we can replace low level to high level but, we can't replace high level to low level.

## **MIME type/Content type:**

MIME-(Multipurpose Internet Mail Extensions)

- response of a Servlet will be shown to the end users by a web browser.
- A Servlet can generate response in different text format like html , xml, pdf , xml etc.
- If a response of a Servlet is a non html format then a Servlet has to inform a browser about the format of the response.
- A Servlet can inform the browser by setting contentType to the response. This can be done by calling setContentType().
- If the response format is html then it is optional to set the content type.
- If a content type set from a response then the browser will be ready and takes necessary actions to show the response of a Servlet to a given format.
- The predefined MIME type
  - text/html→defalut
  - text/plain
  - text/xml
  - application/pdf
  - application/vnd.ms-excel
  - application/vnd.ms-powerpoint
  - image/gif
  - image/jpeg
  - audio/mp4
  - video/mpeg

The following example contains two Servlets to send xml and Excel type of response to the browser.



Eclipse folder structure:

Diagram-1

```
//index.html
<center>
  <a href="xml">XML</a>
  <a href="excel">EXCEL</a>
</center>

//XMLServlet.java
//XMLServlet
public class XMLServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
  IOException
  {
    response.setContentType("text/xml");
    PrintWriter out=response.getWriter();
    out.println("<book>");
    out.println("<bookid>101</bookid>");
    out.println("<price>400</price>");
    out.println("</book>");
    out.close();
  }
}

//ExcelServlet.java
//ExcelServlet
public class ExcelServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
  IOException
  {
    response.setContentType("application/vnd.ms-excel");
    PrintWriter out=response.getWriter();
    out.println("This is of excel type content");
  }
}
```

```
out.close();
}
}
```

**[Monday, July 07, 2014]**

## **Differences between Get and Post Method:-**

### **Get**

- Get is used to get a file from server.
- With get method, we can send approx. maximum of 1 kb of data only.
- Get is not a secured method. Because it will display a form data in address bar.
- File upload operation will not be supported by get method.
- Get is an idempotent method.

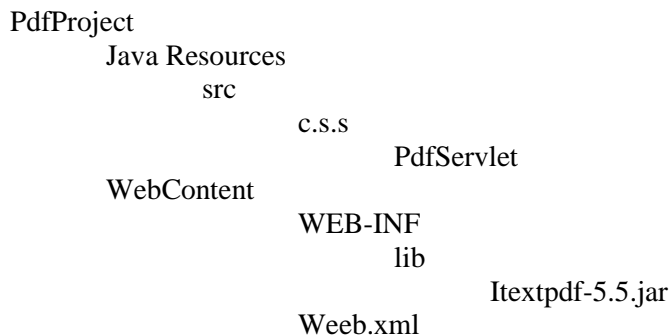
### **Post**

- Post is used to post data to the server.
- Using post method, we can send any amount of data to the server.
- Post is a secured method. Because post method does not display a form data in address bar.
- File upload method is supported by post method.
- Post is a non-idempotent method.

**Note: - An idempotent method will be used for collecting a file from server and non-idempotent method will be used sending some data to a file in a server.**

### **Creating a PDF type of response of a servlet:-**

- To generate a pdf format of response from a servlet, servlet api has not given pre-defined classes or interfaces for generating pdf response.
- So we take the help of third party api for generating the response in pdf from a servlet.
- iText is a third party vendor provides software for creating pdf response. So first we need to download itextpdf-5.5.1.jar



```
<center>
    <a href="pdf"> PDF </a>
</center>
```

```
package com.satya.servlet;
```

```
import java.awt.Font;
import java.io.IOException;
```

```
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
```

```

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

/**
 * Servlet implementation class PDFServlet
 */
public class PDFServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * Default constructor.
     */
    public PDFServlet() {
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("application/pdf");
        ServletOutputStream sos=response.getOutputStream();
        Document doc=new Document();
        try
        {
            PdfWriter.getInstance(doc,sos);
            Font
font=newFontFamily.TIMES_ROMAN,25,Font.BOLD,BaseColor.RED);
            doc.open();
            doc.add(new Paragraph("This is a sample PDF message, font"));

        }
        catch(Exception e)
        {
            System.out.println(e);
        }
        doc.close();
    }
}

```

- download the required jar from the internet and add the jars into the buildpath and also in lib folder and then run the above program.

**[Tuesday, July 08, 2014]**

Q. Can we added checked exceptions to the throws statement, while overriding a super class method in a sub-class?

Ans. No, We can add additional unchecked exceptions to the throws statement but we cannot add checked exceptions.

**protected void** service(HttpServletRequest request, HttpServletResponse response)  
**throws** ServletException, IOException, SQLException ----invalid, it is checked exception

**protected void** service(HttpServletRequest request, HttpServletResponse response)  
**throws** ServletException, IOException, NullPointerException ---Valid it is unchecked exception

### Exception Handling in Servlet:-

- While executing the business implemented in a servlet, if any exception is occurred at runtime then that exception stack trace message will be printed on the browser of end-user.
- End-users are unknown about technologies, so an end-user cannot understand the actual problem occurred at server.
- In order to display a meaningful message to the end-user, instead of exception stack trace, we need to handle the exceptions in a servlet.
- Exception handling in web application can be done in two ways.
  1. In programmatic approach
  2. In declarative approach

#### Programmatic Approach:-

- In this approach we need to put servlet logic under try and catch blocks.
- If an exception is occurred in servlet logic then control will be jumped to catch block and from there we can redirect control to another html, jsp or servlet to generate the response on to the browser.

For example,

```
public class MyServlet extends HttpServlet
{
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        try{
            .....
        }
        catch(Exception e)
        {
            response.sendRedirect("error.html");
        }
    }
}
```

- With this programmatic approach, there are few drawbacks.
  1. A programmer has to insert try and catch blocks in each servlet
  2. If any changes are needed in exception handling then we need to recompile, reload, and sometimes restart the server.

#### Declarative Approach:-

- In this approach, we can configure tags in web.xml instead of putting try and catch block in servlet.
  - We can commonly handle exceptions of all servlets in a project, at web.xml
  - If any modifications are done in web.xml then there is no need of recompile or reload or restart of server.
- <error page>

```

<exception-type>java.lang.Exception</exception-type>
<location>/error.html</location>
</error-page>

```

- In web.xml, <error-page> tag can be repeated for any no of times.
- If an exception occurred in a servlet then container will first search for exception of particular type, if not then super class error page will be executed.

```

<error page>
<exception-type>java.lang.Exception</exception-type>
<location>/error.html</location>
</error-page>
<error page>
<exception-type>java.lang.ArithmeticException</exception-type>
<location>/sorry.jsp</location>
</error-page>

```

Wednesday, July 09, 2014

- In the following example, our servlet class takes account number and amount as input and throws InvalidAmountException if amount<=0 and throws AmountOverflowException if amount>20000.
- In this example, we are applying declarative exception handling mechanism.

```

ExceptionProject
  JAvA Resources
    src
      com.satya.servlet
        FundTranserServlet.java
        ErrorServlet.Java
      com.satya.exception
        InvalidAmountException.java
        AmountOverflowException.java
  WebContent
    WEB-INF
      web.xml
      bank.html
      invalid.html

```

- Open web.xml and add the following error page tags

```

<error-page>
<exception-type>com.sathya.exception.InvalidAmountException</exception-type>
<location>/invalid.html</location>
</error-page>
<error-page>
<exception-type>com.sathya.exception.AmountOverflowException</exception-type>
<location>/errorsrv</location>
</error-page>

```

```

package com.sathya.servlet;

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;

/**
 * Servlet implementation class ErrorServlet
 */
public class ErrorServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ErrorServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        PrintWriter out=response.getWriter();
        out.println("<h2>");
        out.println("Amount is too large, Amount should be <=20000");
        out.println("</h2>");
    }

}

package com.sathya.servlet;

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;

import com.sathya.exception.AmountOverflowException;
import com.sathya.exception.InvalidAmountException;

```



```

/**
 * Servlet implementation class FundTransfer
 */
public class FundTransfer extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public FundTransfer() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
     * response)
     */
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        processMyRequest(request, response);
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
     * response)
     */
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        processMyRequest(request, response);
    }

    public void processMyRequest(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException,
        InvalidAmountException, AmountOverflowException {
        // TODO Auto-generated method stub
        Integer accno = Integer.parseInt(request.getParameter("accno").trim());
        Integer amount = Integer
            .parseInt(request.getParameter("amount").trim());
        PrintWriter out = response.getWriter();
        if (amount <= 0)
            throw new InvalidAmountException();
        if (amount > 20000)
            throw new AmountOverflowException();
        out.println("<h1>");
        out.println("Amount transfered successfully.....");
        out.println("</h1>");
        out.close();
    }
}

```

```

    }
    package com.sathya.exception;

    public class AmountOverflowException extends RuntimeException {

    }
    package com.sathya.exception;

    public class InvalidAmountException extends RuntimeException {

    }

```

Bank.html

```

<center>
    <form action="/fund">
        Account No:<input type="text" name="accno">
        Amount:<input type="text" name="amount">
        <input type="submit" value="TRANSFER">
    </form>
</center>
Invalid.html
<font color="red" size="10">
<h1>Amount is invalid and amount should be >=0</h1>
</font>

```

Web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
    <display-name>ExceptionProject</display-name>
    <welcome-file-list>
        <welcome-file>bank.html</welcome-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>FundTransfer</servlet-name>
        <servlet-class>com.sathya.servlet.FundTransfer</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>FundTransfer</servlet-name>
        <url-pattern>/fund</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>ErrorServlet</servlet-name>
        <servlet-class>com.sathya.servlet.ErrorServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ErrorServlet</servlet-name>
        <url-pattern>/errorsrv</url-pattern>
    </servlet-mapping>
    <error-page>

```

```

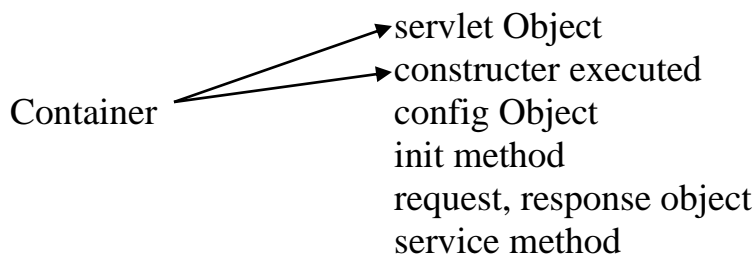
<exception-type>com.sathya.exception.InvalidAmountException</exception-type>
<location>/invalid.html</location>
</error-page>
<error-page>
<exception-type>com.sathya.exception.AmountOverflowException</exception-type>
<location>/errorsrv</location>
</error-page>
</web-app>

```

Thursday, July 10, 2014

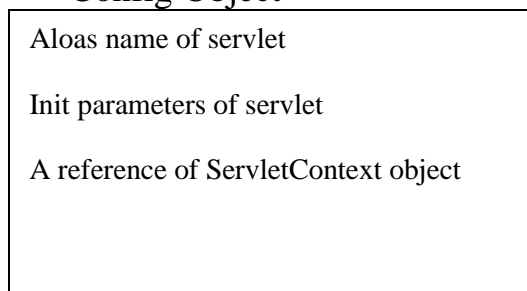
### **ServletConfig Interface:-**

- It is an interface of javax.servlet package.
- ServletConfig an interface and its implementation class is given by server vendor.
- A ServletConfig object is nothing but it is an object of its implementation class.
- A container creates config object just before calling life cycle init() method of a servlet class object.
- A container will follow the below order to call init and service methods of a servlet.



- A container stores following three types of information in a config object, whenever it is created.

#### **Config Object**



- A web container gathers alias names and init parameters from web.xml file.
- Alias name is used in a servlet chaining mechanism.

The need of init parameters:-

- While creating logic in a servlet class, sometimes it needs technical values.
- For example, while creating a servlet to connect with servlet we need technical values like driver, url, username and password.

- If technical values are directly hard-coded in servlet then there is a drawback. The drawback is, if any changes are done on technical values then we need to recompile, reload and restart the server.
- To overcome the above problem we have two options,
  1. We can use a properties file
  2. We can use init parameter in web.xml
- If properties file is created then to read the content of properties file, some huge code is required in a servlet. So second option is better than first option.
- If any changes are made in web.xml, then there is **no need of** recompile or reload or restart.
- Finally, init parameters are used to pass technical values to a servlet from web.xml file.

### How to configure init parameters:-

- To configure, we use <init-param> tag under <servlet> tag of web.xml.
  - We can configure any no of init parameters under a <servlet> tag.
  - Every init parameters is a combination of key / value pair.
- ```
<init-param>
<param-name>key</param-name>
<param-value>value</param-value>
```

Friday, July 11, 2014

### Q. How to obtained config object into service method?

- Actual, a container will pass config object to init() method of servlet. But we develop a logic in service method so we need config object in service.
- We have two option to get config object into service method.

Option1:-

If we override life cycle init in our servlet then we can store the local config of init() method in a instance config object of servlet.

We can access the instance config object in service method.

```
public class MyServlet extends GenericServlet
{
    private ServletConfig servletConfig
    public void init(ServletConfig config){
        servletConfig=config;
    }
    public void service(request, response) throws SE, IOE{
        //we can use sc object
    }
}
```

Option2:-

If we don't override life cycle service method in our servlet, then container call supere class init() method by passing config object.

In order to get config() object from super class into our class service method, we can call getServletConfig() method.

```
public class MyServlet extends GenericServlet{
```

```
    public void service(request, response) throws SE, IOE{
```

```
        ServletConfig config = getServletConfig();
```

```
        //we can use sc object
```

```
    }
```

```
}
```

Methods of config object:-

1. getServletName()

2. getInitParameter()

3. getInitParameterName()

4. getServletContext()

➤ To read alias names of a servlet we call getServletName()

```
String aliasName=config.getServletName();
```

➤ To read the value of a particular init parameter then we call getInitParameter(). It returns value in string format and we need to apply wrapping if required.

```
String value = config.getInitParameter(key);
```

➤ If there are many init parameters exists in config object then we can get all the parameter keys into Enumeration object by calling getInitParameterNames() method.

```
Enumeration e = config.getInitParameterNames();
```

➤ To read reference of servlet context object, we call getServletContext() method.

```
ServletContext ctx = config.getServletContext();
```

**Q. In what object init parameters are stored internally by config object?**

Ans:- Map Object

**Q. What is the difference between constructor and init() method of a servlet?**

Ans:- A constructor cannot access config object, but init() method can access it. Because config object is created by container after a constructor is executed.

**Q. How many config object are created in web application?**

Ans:- one-per-servlet. One config object for one servlet object. The ratio is 1:1.

**Q. Can we write a parameterized constructor in servlet or not?**

Ans:-

In every servlet class a default constructor is mandatory.

If we write only a parameterized constructor in a servlet, then we will get an exception at runtime.

So if we want to define a parameterized constructor in servlet then I should also include a default constructor.

**Q. Why a default constructor is mandatory in servlet?**

Ans:- A container calls newInstance() method to create a servlet object. newInstance() method can create object only if default constructor exist in that class.

So, every servlet must contain a default constructor.

**Q. When a servlet object is destroyed?**

Ans:-

1. When a server is shutdown
2. When an application is un-deployed from server.
3. When a servlet object is idle (not doing nothing) in server for a long time.
4. When a server is crashed

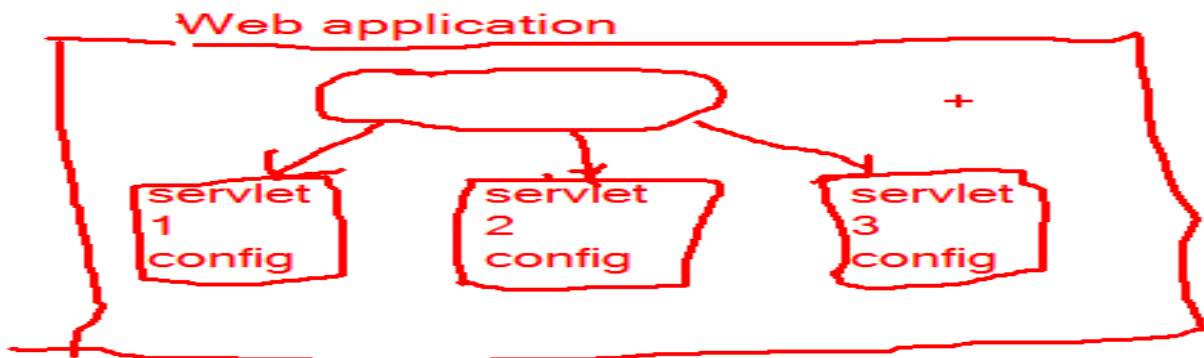
Q. Difference between server and container?

Ans:-

The following example is to insert a student data to the database using servlet with init() parameters.

Monday, July 14, 2014

ServletContext interface:-

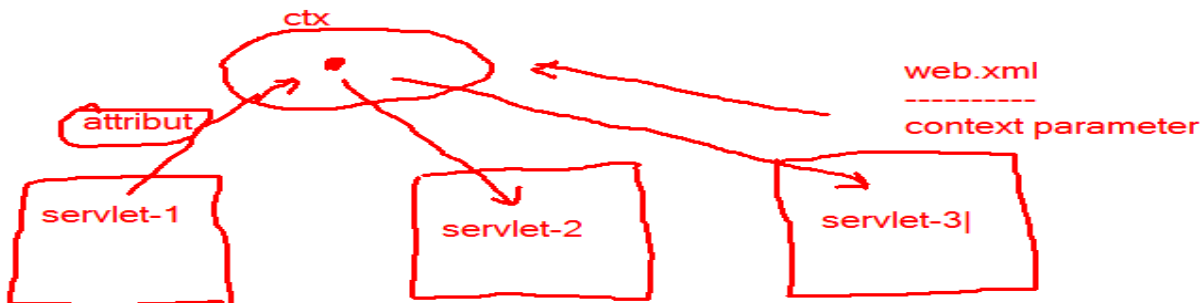


- ServletContext is a interface of javax.servlet package
- The implementation class of ServletContext interface will be provided by a server vender. So SContext object means object of implementation class
- Whene a web app is deployed in a server immediately a container is web.xml (descriptor file) and on successful deployment of a web app a container immediately create SCO.
- SCO is one per Web app per JVM
- The first object is created by a container for a web app is context object
- Context object of a web app is created

- Context object of a web app is created when an app is deployed and it will be destroyed when every app is undeployed.
- Context object is the first object of web app by container and is the last object destroyed by the container.

The order of creating object by container is :-

1. Context object
2. Servlet object
3. Config object



Tuesday, July 15, 2014

- Context object is a globally shareable object for all servlets of web applications.
- The data stored in a context object becomes visible to all servlets of the web application.
- In a context object, data can be stored only in a key/value pairs. Because context object internally maintains Map object to store the data.
- In to context object, data can be stored from two places
  1. From web.xml
  2. From servlet
- The data stored from web.xml are called parameters and the data stored from a servlet is called attributes.
- In web.xml we should configure context parameters to store the data in context object.
- If data stored from web.xml then it is called declarative approach and if data stored from servlet then it is called programmatic approach.
- To configure context parameters we use <context-param> tag under <web-app> tag.
 

```

<web-app>
  <context-param>
    <param-name>key</param-name>
    <param-value>value</param-value>
  </context-param>
</web-app>
      
```

*Q. What is the difference between init and context parameters?*

*Ans:*

1. *init parameters are local to a servlet. But context parameters are global to all servlets.*
2. *init parameters are configured inside servlet tag but context parameters are configured at outside servlet tag*
3. *init parameter goes to config object. Context parameter goes to context object.*

*Q. How many types of parameter of a servlet?*

*Ans: Three types*

1. *init parameter*
2. *context parameter*
3. *request parameter*

➤ Parameters and attributes of a context object having some differences

1. In a context object a parameter key and its value, both are stored strings. But in case of attributes key is stored as string and value is stored as an object.
2. In a servlet, we can only read the value of a parameter. But we can read and update the value of attribute. It means parameters are read-only and attributes are read and write.
3. We can get a reference of type ServletContext in a servlet class like the following.

`ServletContext ctx = config.getServletContext();`

`ServletContext ctx = getServletContext();`

*Q. How to read the context parameters?*

*Ans:- ServletConfig and ServletContext, both objects are having two common methods, for reading init parameters and context parameters.*

*String value = ctx.getInitParameter("key");*

*Enumeration e = ctx.getInitParameterNames();*

**Q. How to read/write attributes?**

**Ans:-** We have four methods in ServletContext interface to read or write attributes.

1. `setAttribute(key,value)` --- to write an attribute
2. `getAttribute(key)` --- to read an attribute
3. `getAttributeNames()` --- to read names of all attributes
4. `removeAttribute(key)` --- to remove or erase an attribute

**Q. \*\*\*\*\* Differences between Config and Context object?**

**ServletConfig:-**

1. Config object is local to a servlet. It is not sharable with other servlet.
2. A config object doesn't exist without a servlet object.
3. Config object only stores parameters (init). But it cannot store attributes.

**ServletContext:-**

1. Context object is global to a servlet. It means It is sharable for all servlets..
2. Context object can exist without a servlet object.
3. A Context object is can store both parameters and attributes.

**Wednesday, July 16, 2014**

➤ The following example is to perform insert and select operations on database using context parameters to open the connection.

Eclipse Directory Structure

ContextProject

Java Resources

src

com.sathya.servlet

InsertServlet.java

SelectServlet.java

WebContent

WEB-INF

Web.xml

lib



ojdbc14.jar  
index.html  
insert.html  
select.html

- Add the following four context parameters under <web-app> tag of web.xml

```
<context-param>  
    <param-name>driver</param-name>  
    <param-value>oracle.jdbc.driver.OracleDriver</param-value>  
</context-param>  
<context-param>  
    <param-name>url</param-name>  
    <param-value>jdbc:oracle:thin:@localhost:1521:XE</param-value>  
</context-param>  
<context-param>  
    <param-name>username</param-name>  
    <param-value>scott</param-value>  
</context-param>  
<context-param>  
    <param-name>password</param-name>  
    <param-value>tiger</param-value>  
</context-param>
```

Thursday, July 17, 2014