Servlets

- Satya Kaveti

Small Codes

Programming Simplified

A SmlCodes.Com Small presentation

In Association with Idleposts.com

For more tutorials & Articles visit SmlCodes.com

Servlets by Satya Kaveti

If you discover any errors on our website or in this tutorial, please notify us at support@smlcodes.com or smlcodes@gmail.com

First published on 27th Oct 2016, Published by **SmlCodes.com**

Author Credits

Name	: **Satya Kaveti**
Email	: satyakaveti@gmail.com
Website	: smlcodes.com, satyajohnny.blogspot.com

Digital Partners

Small Codes Programming Simplified

Idle Posts

Black Tree Software Solutions

Table of Contents

.... 1

- Satya Kaveti 1

So called TITLE 1

Author Credits 2

Digital Partners 2

TABLE OF CONTENT 3

1. SERVLETS 4

3.1 Basics of Web Technologies 5

3.2 Servlet API 8

1. javax.servlet.Servlet (Interface) 8

2.javax.servlet.GenericServlet (absraect class) 8

3.javax.servlet.http.HttpServlet 9

3.3 Servlet Lifecycle 10

3.4 ServletRequest (interface)→ getParamaters() 13

3.5 ServletConfig (interface)→ getInitParamaters() 17

3.6 ServletContext (interface)→ getInitParamaters() 18

3.7 ServletChaining 21

3.8 Attribbutes 25

3.9 Session Tracking 28

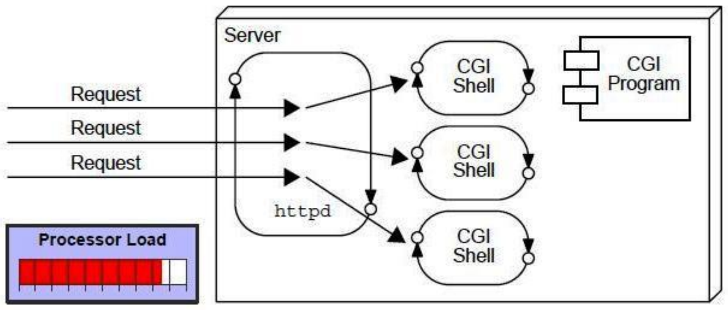
3.10 Filters 33

1. Servlets

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request.

1. CGI

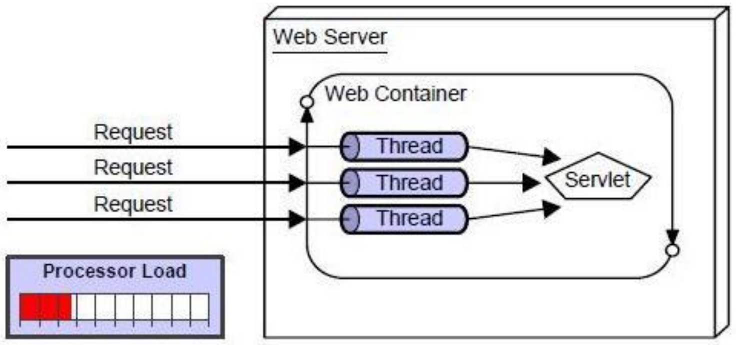
CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



Disadvantages of CGI

- If number of client’s increases, it takes more time for sending response.
- For each request, it starts a process and Web server is limited to start processes.
- It uses platform dependent language e.g. C, C++, perl.

2. Servlet



The web container creates threads for handling the multiple requests to the servlet.

- **Better performance:** because it creates a thread for each request not process.
- **Portability:** because it uses java language.
- **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
- **Secure:** because it uses java language..

3.1 Basics of Web Technologies

Static Website	Dynamic Website
Prebuilt content is same every time the page is loaded.	Content is generated quickly and changes regularly.
It uses the **HTML** code for developing a website.	It uses the server side languages such as **PHP,SERVLET, JSP, and ASP.NET** etc. for developing a website.
It sends exactly the same response for every request.	It may generate different HTML for each of the request.
The content is only changes when someone publishes and updates the file (sends it to the web server).	The page contains **"server-side"** code it allows the server to generate the unique content when the page is loaded.

1. HTTP (Hyper Text Transfer Protocol)

HTTP is TCP/IP based communication protocol, which is used to deliver the data like image files, query results, HTML files etc on the World Wide Web (WWW) with the default port is TCP 80. It provides the standardized way for computers to communicate with each other.

There are three fundamental features that make the HTTP a simple and powerful protocol used for communication:

- **HTTP is media independent:** It refers to any type of media content can be sent by HTTP as long as both the server and the client can handle the data content.
- **HTTP is connectionless:** It is a connectionless approach in which HTTP client i.e., a browser initiates the HTTP request and after the request is sends the client disconnects from server and waits for the response.
- **HTTP is stateless:** The client and server are aware of each other during a current request only. Afterwards, both of them forget each other. Due to the stateless nature of protocol, neither the client nor the server can retain the information about different request across the web pages.

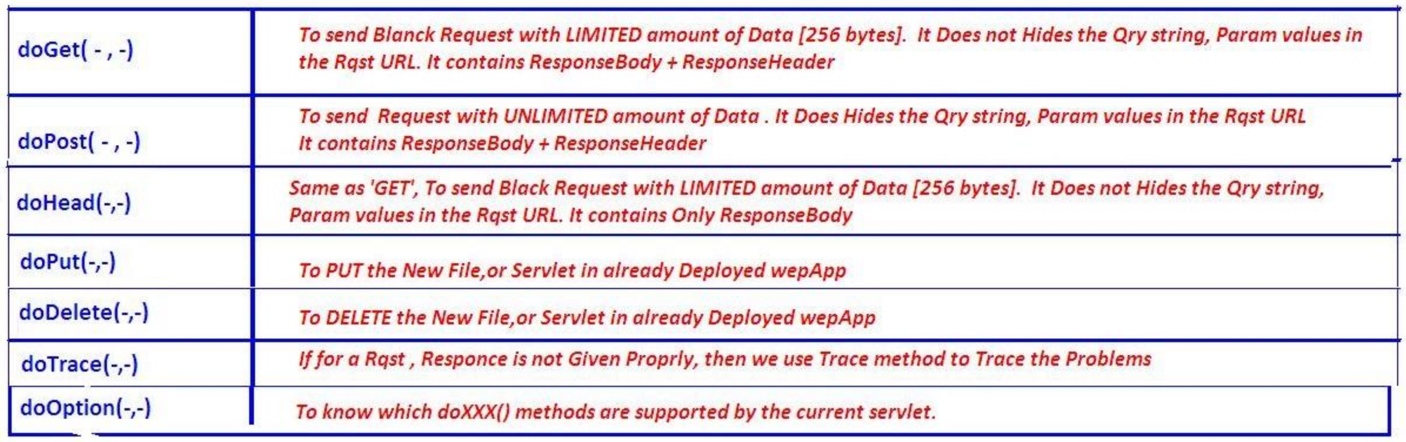
2. HTTP Requests

The request sends by the computer to a web server that contains all sorts of potentially interesting information is known as HTTP requests.

It will send following information to Server

- The analysis of source IP address, proxy and port
- The analysis of destination IP address, protocol, port and host
- The Requested URI (Uniform Resource Identifier)
- The Request method and Content
- The User-Agent header
- The Connection control header

We have following HTTP request methods:



doGet(- , -)	To send Blanck Request with LIMITED amount of Data [256 bytes]. It Does not Hides the Qry string, Param values in the Rqst URL. It contains ResponseBody + ResponseHeader
doPost(- , -)	To send Request with UNLIMITED amount of Data . It Does Hides the Qry string, Param values in the Rqst URL It contains ResponseBody + ResponseHeader
doHead(-,-)	Same as 'GET', To send Black Request with LIMITED amount of Data [256 bytes]. It Does not Hides the Qry string, Param values in the Rqst URL. It contains Only ResponseBody
doPut(-,-)	To PUT the New File,or Servlet in already Deployed wepApp
doDelete(-,-)	To DELETE the New File,or Servlet in already Deployed wepApp
doTrace(-,-)	If for a Rqst , Responce is not Given Proprly, then we use Trace method to Trace the Problems
doOption(-,-)	To know which doXXX() methods are supported by the current servlet.

3. GET vs POST

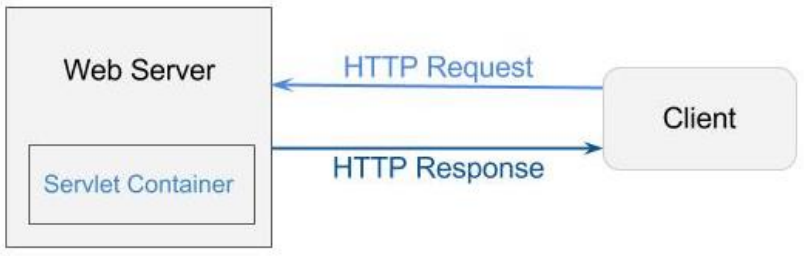
GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data**can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked.**	Post request **cannot be bookmarked.**
4) Get request is **idempotent.** It means second request will be ignored until response of first request is delivered	Post request is **non-idempotent.**
5) Get request is **more efficient** and used more than Post.	Post request is **less efficient** and used less than get.

4. Servlet Container: is the place where Servlet programs are executed

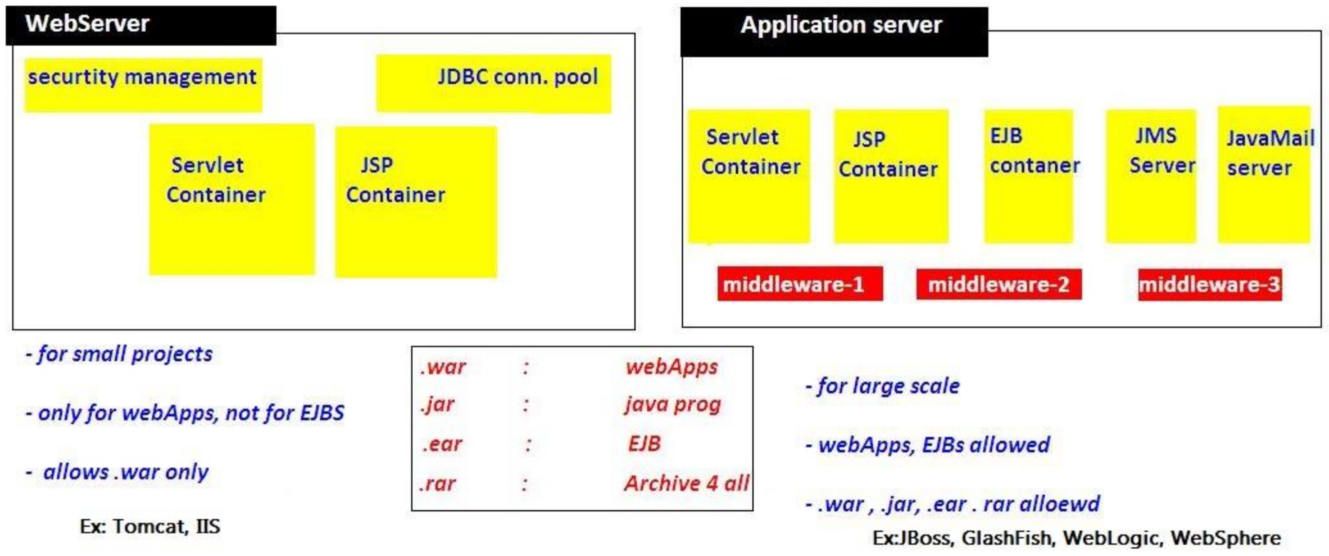
The servlet container is used in java for dynamically generate the web pages on the server side. Therefore the servlet container is the part of a web server that interacts with the servlet for handling the dynamic web pages from the client.

The Servlet Container performs many operations that are given below:

1. **Life Cycle Management**
2. **Multithreaded support**
3. **Object Pooling**
4. **Security etc.**



5. Web Server VS Application Server



- for small projects
- only for webApps, not for EJBS
- allows .war only

Ex: Tomcat, IIS

.war	:	webApps
.jar	:	java prog
.ear	:	EJB
.rar	:	Archive 4 all

- for large scale
- webApps, EJBs allowed
- .war , .jar, .ear . rar alloewd

Ex:JBoss, GlashFish, WebLogic, WebSphere

6. Content Type

Content Type is also known as MIME (Multipurpose internet Mail Extension) Type. It is a HTTP header that provides the description about what are you sending to the browser.

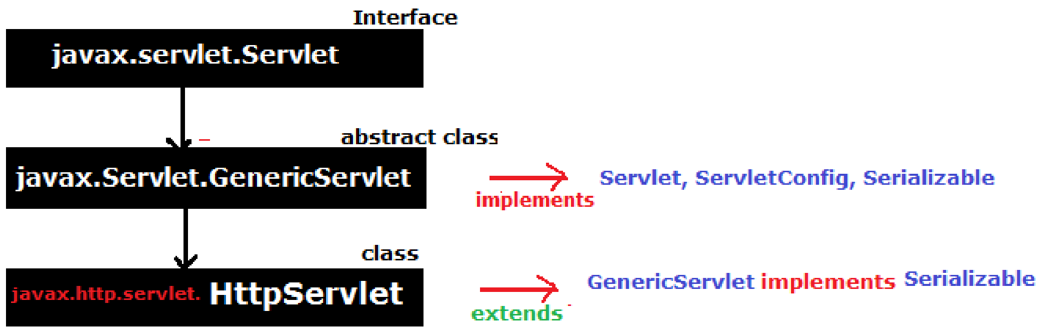
- **It supports the non-ASCII characters**
- **It supports the multiple attachments in a single message**
- **It supports the attachment contains audio, images and video files etc.**
- **It supports the unlimited message length.**

Commonly used content types are given below:

- **text/html**
- **text/plain**
- **application/msword**
- **application/vnd.ms-excel**
- **application/jar**
- **application/pdf**
- **application/octet-stream**
- **application/x-zip**
- **images/jpeg**
- **images/png**
- **images/gif**
- **audio/mp3**
- **video/mp4**
- **Video/quicktime etc.**

3.2 Servlet API

we can create any servlet program by using below 3 ways


Interface
javax.servlet.Servlet
abstract class
javax.Servlet.GenericServlet
implements
Servlet, ServletConfig, Serializable
class
javax.http.servlet. HttpServlet
extends
GenericServlet implements Serializable

1. javax.servlet.Servlet (Interface)

Servlet interface is the ROOT interface of Servlet API. It provides common behaviour to all the servlets.

Method	Description
public void init(ServletConfig config)	Initializes the servlet. It is the life cycle method of servlet and invoked by the web container **only once.**
public void service(ServletRequest req,ServletResponse response)	Provides response for the incoming request. **It is invoked at each request by the web container.**
public void destroy()	Is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	Returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

Steps to implement Servlet program using Servlet Interface

1. Create a Class which **implements Servlet Interface**
2. **Implement** all **5** abstract **methods**
3. Write Request Processing logic in **service(req,res) method**

1. javax.servlet.GenericServlet (absraect class)

- GenericServlet class implements Servlet, ServletConfig and Serializable interfaces.
- It provides implementation for all methods of Servlet interface **except the service().**
- **it is protocol-independent**, so it can handle any request of any protocal
- Create servlet by providing the implementation of the service() method.

Init(),destroy(),getServletConfig(),getservletInfo() are inherted and implemented

1. public abstract void service(ServletRequest req, ServletResponse res)

2. public void init()
it is a convenient method for the servlet programmers, now there is no need to call super.init(config)

3. public ServletContext getServletContext()

4. public String getInitParameter(String name)

5. public Enumeration getInitParameterNames()

6. public String getServletName()

Steps to write Servlet Program using GenericServlet

1. Create a Class which **extends GenericServlet Interface**
2. Implement & Write Request Processing logic in **service(req,res) method**

3.javax.servlet.http.HttpServlet

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

We have 2 service methods

1. **Public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req,HttpServletResponse res)**

   Receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.

7 proreddoXXX (HttpServletRequest, HttpServletResponce) service methods

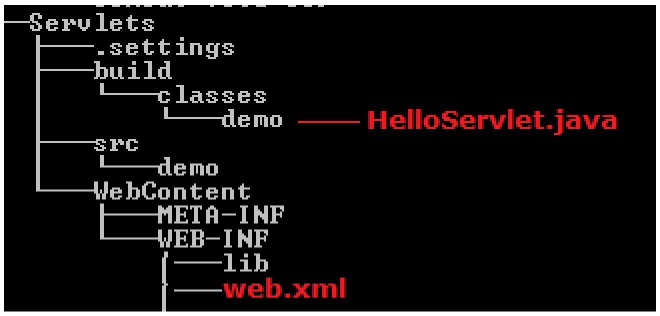
1. **protected void doGet(Htt**pServletRequest req, HttpServletResponse res)
2. **protected void doPost(Ht**tpServletRequest req, HttpServletResponse res)
3. **protected void doHead(H**ttpServletRequest req, HttpServletResponse res)
4. **protected void doOptions**(HttpServletRequest req, HttpServletResponse res)
5. **protected void doPut(Htt**pServletRequest req, HttpServletResponse res)
6. **protected void doTrace(H**ttpServletRequest req, HttpServletResponse res)
7. **protected void doDelete(**HttpServletRequest req, HttpServletResponse res)

Steps to write Servlet Program using GenericServlet

1. Create a Class which **extends HttpServlet Interface**
2. Write Request Processing logic in **service(req,res)** **OR → Not Recommended**
3. Write Request Processing logic in **doXXX(req,res)** **→ doGet,doPost Recommended**

1.3 Servlet Lifecycle

First we see the example, then we can understand the LifeCycle. For every Servlet program contains following strcuture



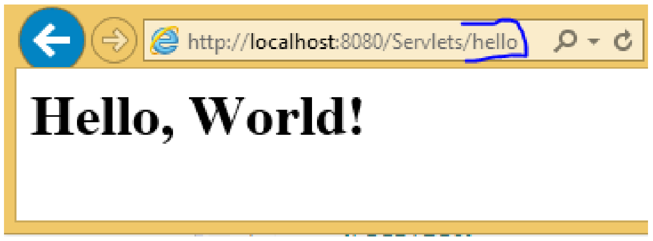
```
─Servlets
  ├───.settings
  ├───build
  │   └───classes
  │       └───demo  ──── HelloServlet.java
  ├───src
  │   └───demo
  └───WebContent
      ├───META-INF
      └───WEB-INF
          ├───lib
          └───web.xml
```

Here **Servlets** is Application name

Example 1: Using Servlet Interface

```
public class HelloServlet implements Servlet{
	ServletConfig config = null;
	@Override
	public void init(ServletConfig config) throws ServletException {
		this.config = config;
		System.out.println("1.Init...");
	}
	@Override
	public void service(ServletRequest req, ServletResponse res) throws
ServletException, IOException {
		System.out.println("2.Service ...");
	PrintWriter pw =  res.getWriter();
	pw.write("<h1>Hello, World!</h1>");
	}
	@Override
	public void destroy() {
		System.out.println("3.Destroy ..");
	}
	@Override
	public ServletConfig getServletConfig() {
		System.out.println("4.getServletConfig ..");
		return config;
	}
	@Override
	public String getServletInfo() {
		return "getServletInfo";
	}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
      <servlet>
            <servlet-name>hello</servlet-name>
            <servlet-class>demo.HelloServlet</servlet-class>
      </servlet>

      <servlet-mapping>
            <servlet-name>hello</servlet-name>
            <url-pattern>/hello</url-pattern>
      </servlet-mapping>

      <welcome-file-list>
            <welcome-file>index.jsp</welcome-file>
      </welcome-file-list>
</web-app>
```



Hello, World!

Flow of Excecution

1. When ever we deploys the application, container loads the application & creates **ServletContext** Object & waits for the Request

2. if we give **<load-on-startup>**1**</load-on-startup>** container will creates ServletConfig Object when the time of Deploying application

3. when we give the url : **http://localhost:8080/Servlets/hello** **,** request goes to container, and it searches for **/hello** url pattern in web.xml

4. web.xml searches for **/hello** , in `<servlet-mapping>` and gets **Servelt-name**

5. container loads `demo.HelloServlet` class and creates creates **ServletConfig** Object and calls inti() method

6. for every request it will calls **service(req,res)** method, for 100 requests it will execute 100 times

7. **destroy()** method will be called before servlet is removed from the container, and finally it will be garbage collected as usual.

In above **<load-on-startup>**1**</load-on-startup>** we may give (1,2..10). based up on priority order it will creates the ServletConfig Object

<welcome-file-list>

- If we want to make any page/servlet as Homepage we have to specify in this tag
- If it contains more then 1 file, it will give priority by the Order

Example 2: Using GenericServlet

```
public class HelloServlet extends GenericServlet  {
      public void service(ServletRequest req, ServletResponse res) throws 
ServletException, IOException {
            res.setContentType("text/html");
            PrintWriter pw = res.getWriter();
            pw.write("Hello, Generic Servlet");
      }
}
```

Example 3: Using HttpServlet

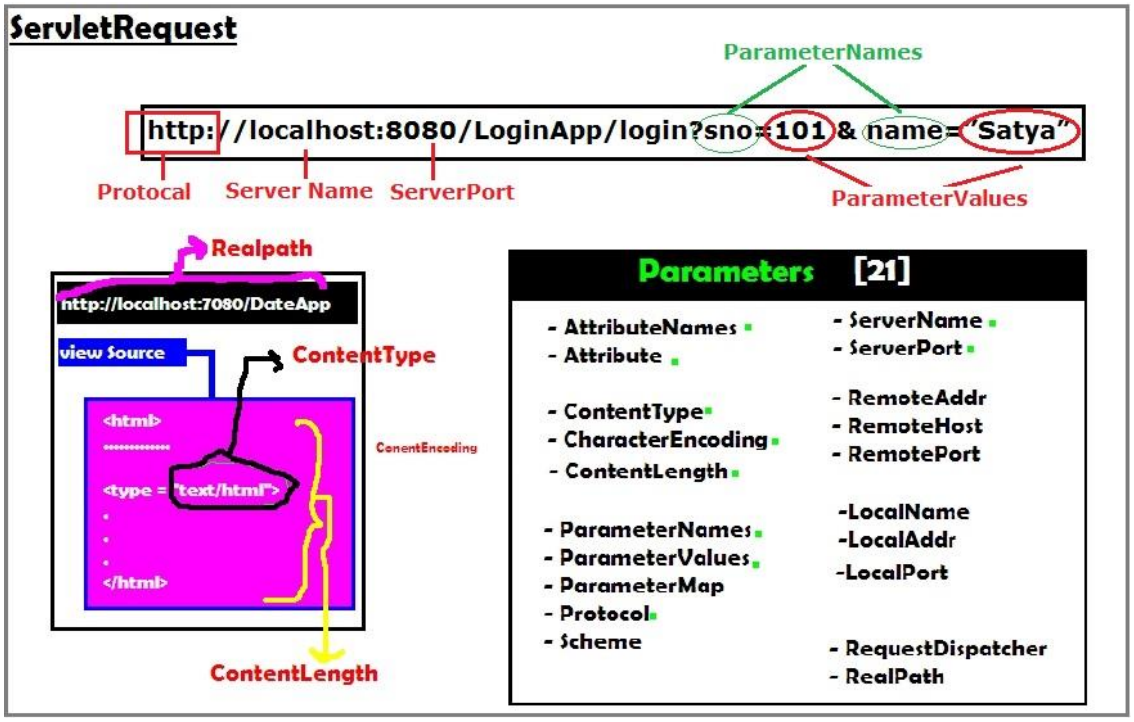
```
public class HelloServlet extends HttpServlet  {
      @Override
      public void service(ServletRequest req, ServletResponse res) throws 
ServletException, IOException {
            System.out.println("Public Service ........");
      }

      @Override
      protected void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            System.out.println("Protecd Service ........");
      }

      @Override
      protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
            System.out.println("doGet()  ....");
      }

      @Override
      protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
            // TODO Auto-generated method stub
            System.out.println("doPost()  ....");
      }
}
```

```
INFO: Reloading Context with name [/Servlets] is completed
Public Service........
```

- Container first calls **public Service(req,res)** method
- Public Service() method internally calls **protected Service(req,res)** method
- Protected Service() method will internally calling **doGet() or doPost() or doXXX()** depends on the type of http method used by the client
- If the client is **not specifying the type of Http** method then Http protocol by **default consider GET method**,
- so **finally** the client request is processed at **doGet() method**

3.4 ServletRequest (interface) → getParamaters()

ServletRequest is send to Server to process particular request. It can send following details to servlet by submitting FORM or by URL.we can get these details at server side



Example to getRequest details

```java
public class ServletReq extends HttpServlet {
        @Override
        protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
                res.setContentType("text/html");
                PrintWriter pw = res.getWriter();
                pw.println("<br> getProtocol   \t:" + req.getProtocol());
                pw.println("<br> getServerName \t:" + req.getServerName());
                pw.println("<br> getServerPort \t:" + req.getServerPort());
                pw.println("<br> getRemotePort \t:" + req.getRemotePort());
                pw.println("<br> getLocalPort  \t:" + req.getLocalPort());

                pw.println("<br> getContentType   \t:" + req.getContentType());
                pw.println("<br> getContentLength \t:" + req.getContentLength());
        pw.println("<br> CharacterEncoding\t:" + req.getCharacterEncoding());
                pw.println("<br> req.getScheme    \t:" + req.getScheme());
        }
}
```

```
getProtocol   :HTTP/1.1
getServerName :localhost
getServerPort :8080
getRemotePort :63205
getLocalPort  :8080
getContentType :null
getContentLength :-1
CharacterEncoding :null
req.getScheme   :http
```

We mainly use ServletRequest **Object to retrieve data from FORM Submission or URL**

We can get the paramaters by using following methods

1. public String	**getParameter("paramname");**
2. public Enumeration	**getParameterNames();**
3. public String[]	**getParamterValues("paramname");**
4. public Map	**getParameterMap();**

Example: getParamater ()

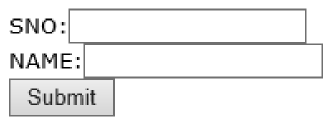
Index.html

```
      <form action="login" method="get">
            SNO:<input type="text" name="sno"><br>
            NAME:<input type="text" name="name"><br>
         <input type="submit" value="Submit">
      </form>
```

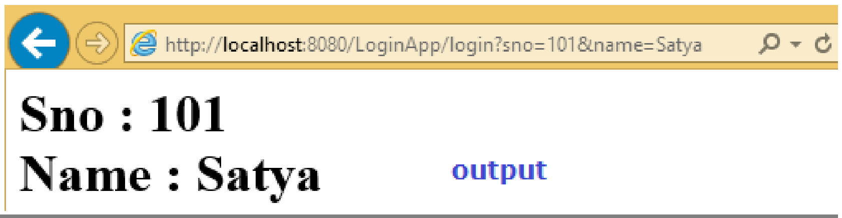
LoginServlet.java

```
public class LoginServlet extends HttpServlet {
      protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            PrintWriter pw = res.getWriter();
            res.setContentType("text/html");
            String sno = req.getParameter("sno");
            String name = req.getParameter("name");
            pw.println("<h1>Sno : " + sno);
            pw.println("<br>Name : " + name);
            pw.close();
      }
}
```

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


SNO:
NAME:
Submit

index.html


http://localhost:8080/LoginApp/login?sno=101&name=Satya

Sno : 101
Name : Satya

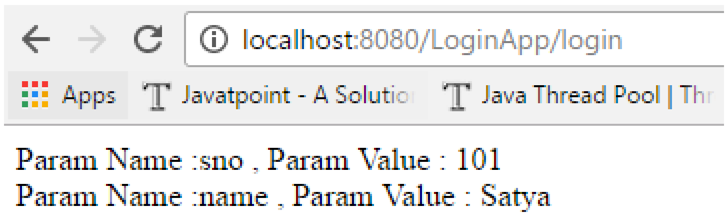
output

Make sure if we use GET method in form we must use doGet (Req, Res) & for POST we have to use doPost(req,res). Otherwise it throws Get/Post not supported error.in this type of case write logic doGet() & call doGet() in doPost()

public Enumeration getParameterNames();

Sometimes we don’t know the request parameter names, in this case we use getParameterNames ();. See the same UI for this only Servlet code is changed

```
public class LoginServlet extends HttpServlet {
	protected void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
			PrintWriter pw = res.getWriter();
			res.setContentType("text/html");
			Enumeration e = req.getParameterNames();
			while (e.hasMoreElements()) {
				String s = (String) e.nextElement();
pw.write("Param Name :" +s + ",Param Value : " + req.getParameter(s)+"<br>");
			}
			pw.close();
	}
}
```


localhost:8080/LoginApp/login

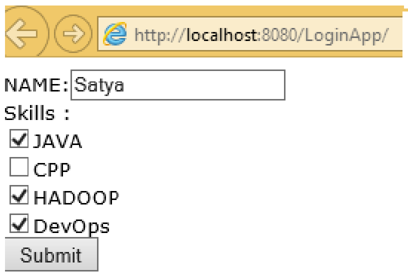
Param Name :sno , Param Value : 101
Param Name :name , Param Value : Satya

getParamterValues(“paramname”), getParameterMap(); are used in the case of Single parameter can having multiple values, like checkboxes. See below example

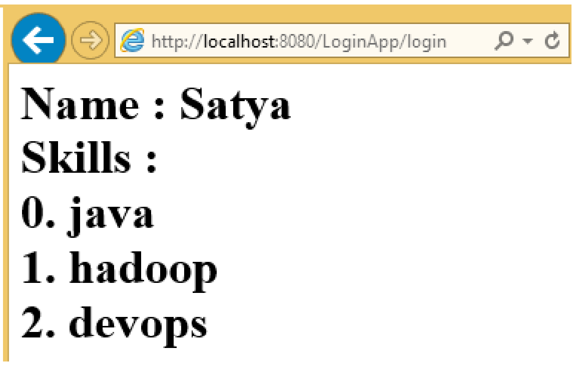
getParamterValues(“paramname”) Example

```
<form action="login" method="post">
		NAME:<input type="text" name="name"><br>
		Skills : <br>
		  <input type="checkbox" name="skill" value="java">JAVA<br>
		  <input type="checkbox" name="skill" value="cpp">CPP<br>
		  <input type="checkbox" name="skill" value="hadoop">HADOOP<br>
		  <input type="checkbox" name="skill" value="devops">DevOps<br>
		<input type="submit" value="Submit">
	</form>
```

```
public class LoginServlet extends HttpServlet {
	protected void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
			PrintWriter pw = res.getWriter();
			res.setContentType("text/html");
			pw.write("<h1> Name : " + req.getParameter("name"));
			pw.write("<br> Skills : <br> ");
			String[] skills = req.getParameterValues("skill");
			for (int i = 0; i < skills.length; i++) {
				pw.write(i + ". " + skills[i] + "<br>");
			}
			pw.close();
	}
}
```


http://localhost:8080/LoginApp/

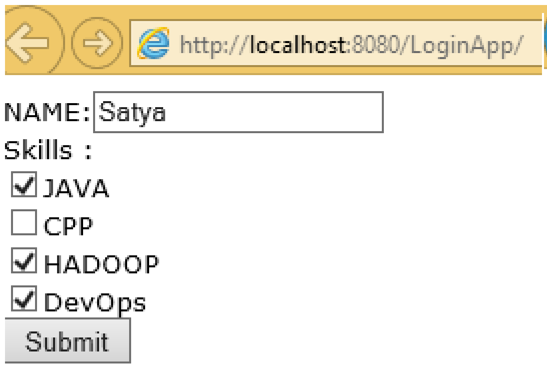
NAME: Satya
Skills :
[✓] JAVA
[ ] CPP
[✓] HADOOP
[✓] DevOps


http://localhost:8080/LoginApp/login

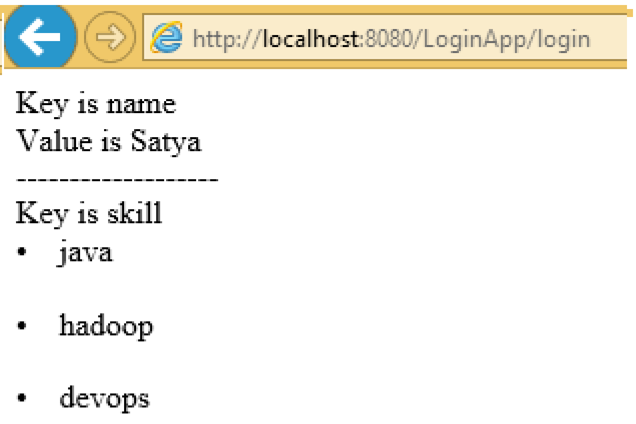
```
Name : Satya
Skills :
0. java
1. hadoop
2. devops
```

getParameterMap(); Example

```java
public class LoginServlet extends HttpServlet {
      protected void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            PrintWriter pw = res.getWriter();
            res.setContentType("text/html");

            Map m = req.getParameterMap();
            Set s = m.entrySet();
            Iterator it = s.iterator();

            while (it.hasNext()) {
            Map.Entry  entry =  it.next();

            String key = entry.getKey();
            String[] value = entry.getValue();
            pw.println("Key is " + key + "<br>");

      if (value.length > 1) {
        for (int i = 0; i < value.length; i++) {
            pw.println("<li>" + value[i].toString() + "</li><br>");
            }
      } else
         pw.println("Value is " + value[0].toString() + "<br>");
      pw.println("-------------------<br>");
            }
            pw.close();
      }
}
```


http://localhost:8080/LoginApp/

NAME: Satya
Skills :
[✓] JAVA
[ ] CPP
[✓] HADOOP
[✓] DevOps


http://localhost:8080/LoginApp/login

```
Key is name
Value is Satya
-------------------
Key is skill
•  java

•  hadoop

•  devops
```

3.5 ServletConfig (interface) → getInitParamaters()

- ServletConfig is one of the **pre-defined interface**.
- ServletConfig object exists **one per servlet program**.
- An object of ServletConfig created by the container **during its initialization phase**.
- An object of ServletConfig is available to the servlet during its execution, once the servlet execution is completed, automatically ServletConfig interface object will be removed by the container.
- **It contains <init-param> details at web.xml, of a particular servlet.**
- The moment when we are using an object of ServletConfig, **we need to configure the web.xml by writing <init-param> tag under <servlet> tag of web.xml**.

1. How to get ServletConfig Object

We can ServletConfig object in 2 ways

1. By calling getServletConfig() on current servlet

ServletConfig conf = getServletConfig();

Above method is available in Servlet interface, inherited in to GenericServlet & HttpServlet

2. ServletConfig object will be available in init() method of the servlet.

```
  public void init(ServletConfig config)
{
// ………………….
}
```

2. How to place <init-param> in web.xml

We have to place **<init-param>** in between **<servlet>** tags

```
<web-app>
  <servlet>
    <servlet-name>login</servlet-name>
    <servlet-class>demo.LoginServlet</servlet-class>
     <init-param>
          <param-name>s1</param-name>
          <param-value> 100 </param-value>
      </init-param>

      <init-param>
          <param-name>s2</param-name>
          <param-value>200</param-value>
      </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>login</servlet-name>
    <url-pattern>/login</url-pattern>
  </servlet-mapping>
</web-app>
```

3.how to get Initparameters in Servlet Programe

We can retrieve <init-param> values by using following methods

- public String **getInitParameter("param name");**
- public Enumeration **getInitParameterNames();**

```
public class LoginServlet extends HttpServlet {
      protected void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            PrintWriter pw = res.getWriter();
            res.setContentType("text/html");

            ServletConfig cfg = getServletConfig();
            pw.write("<h3> 1. Using getInitParameter()");
            pw.write("<br> s1 : " + cfg.getInitParameter("s1"));
            pw.write("<br> s2 :" + cfg.getInitParameter("s2"));

            pw.write("<br><br>  2. Using getInitParameterNames()");
            Enumeration e = cfg.getInitParameterNames();
            while (e.hasMoreElements()) {
                  String s = (String) e.nextElement();
                  pw.write("<br>" +s + "\t : " + cfg.getInitParameter(s));
            }
      }
}
```

http://localhost:8080/LoginApp/login

```
1. Using getInitParameter()
s1 : 100
s2 :200

2. Using getInitParameterNames()
s1 : 100
s2 : 200
```

3.6 ServletContext (interface) → getInitParamaters()

- Object of ServletContext interface is available **one per web application.**
- ServletContext object is automatically created by the container **when the web application is deployed**.
- <context-param> is placed between <web-app> tags. Because the paramaters can be accessed by all the servlets in the Web Application

1. how to get ServletContext Object

We have 3 ways

1. Using ServletConfig Object

```
ServletConfig conf        = getServletConfig();
ServletContext context    = conf.getServletContext();
```

2. By calling getServletContext() on GenericServlet

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

getServletContext () method is defined in GenericServlet

3. By calling getServletContext() on HttpServlet

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

getServletContext () method is defined in GenericServlet inherited to HttpServlet

2. How to place <context-param> in web.xml

<context-param> is placed between <web-app> tags. Because the paramaters can be accessed by all the servlets in the Web Application

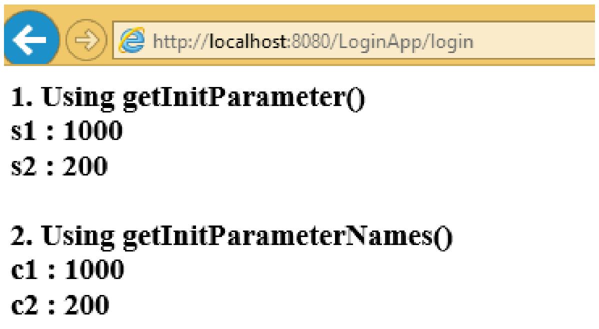
```
<web-app>
      <context-param>
            <param-name>c1 </param-name>
            <param-value>1000</param-value>
      </context-param>

      <context-param>
            <param-name>c2 </param-name>
            <param-value>200</param-value>
      </context-param>

      <servlet>
            <servlet-name>login</servlet-name>
            <servlet-class>demo.LoginServlet</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>login</servlet-name>
            <url-pattern>/login</url-pattern>
      </servlet-mapping>
      <welcome-file-list>
            <welcome-file>index.html</welcome-file>
      </welcome-file-list>
</web-app>
```

3. how to context-params in Servlet Programe

We have two methods

- **public String**   **getInitParameter("param name");**
- **public Enumeration**   **getInitParameterNames();**

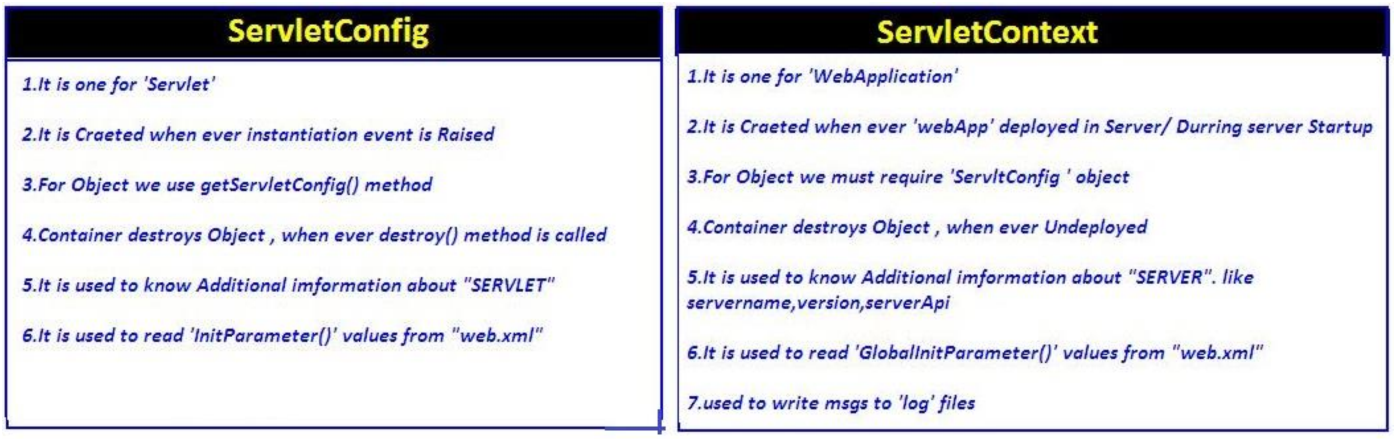
```java
public class LoginServlet extends HttpServlet {
      protected void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            PrintWriter pw = res.getWriter();
            res.setContentType("text/html");

            ServletConfig cfg = getServletConfig();
            ServletContext context = cfg.getServletContext();

            pw.write("<h3> 1. Using getInitParameter()");
            pw.write("<br> s1 : " + context.getInitParameter("c1"));
            pw.write("<br> s2 : " + context.getInitParameter("c2"));

            pw.write("<br><br>  2. Using getInitParameterNames()");
            Enumeration e = context.getInitParameterNames();
            while (e.hasMoreElements()) {
                  String s = (String) e.nextElement();
            pw.write("<br>" + s + "\t : " + context.getInitParameter(s));
            }
      }
}
```



http://localhost:8080/LoginApp/login

1. Using getInitParameter()
s1 : 1000
s2 : 200

2. Using getInitParameterNames()
c1 : 1000
c2 : 200



ServletConfig	ServletContext
1.It is one for 'Servlet'	*1.It is one for 'WebApplication'*
2.It is Craeted when ever instantiation event is Raised	*2.It is Craeted when ever 'webApp' deployed in Server/ Durring server Startup*
3.For Object we use getServletConfig() method	*3.For Object we must require 'ServltConfig ' object*
4.Container destroys Object , when ever destroy() method is called	*4.Container destroys Object , when ever Undeployed*
5.It is used to know Additional information about "SERVLET"	*5.It is used to know Additional information about "SERVER". like servername,version,serverApi*
6.It is used to read 'InitParameter()' values from "web.xml"	*6.It is used to read 'GlobalInitParameter()' values from "web.xml"*
	7.used to write msgs to 'log' files

we never Creates our ServletClassObject, ServletConfig, ServletContext, Objects!

Bcoz ServletContainer Takes care about these things

3.7 ServletChaining

Servlet chaining is used to achive Communication beween servlets. To perform this we have to use RequestDisptcher inferface .following are the possible ways to achive ServletChaining

1. **rd. forward(req, res)**
2. **rd. include (req, res)**
3. **res.sendRedirect(/url)**

RequestDisptcher Interface

The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. We have 2 main methods in this RequestDisptcher

1. **public void forward(ServletRequest req,ServletResponse res)**
2. **public void include (ServletRequest req,ServletResponse res)**

1. How to get RequestDisptcher Object

We have 3 ways to get RequestDisptcher object

1) using Request object

```
RequestDispatcher rd  =  request.getRequestDispatcher("/url or servletname");
      rd.forward(req, res);
      rd.include(req, res);
```

If we use **request** object, the webresource programs **are must be** in **same web application**

2) using ServletContext object with getRequestDispatcher("url") method

```
RequestDispatcher rd  =  context.getRequestDispatcher("/url or servletname");
      rd.forward(req, res);
      rd.include(req, res);
```

If we use **Context** object, the webresource programs **are may in same/different web applictions**

3) using ServletContext object with getNamedDispatcher("servletname") method

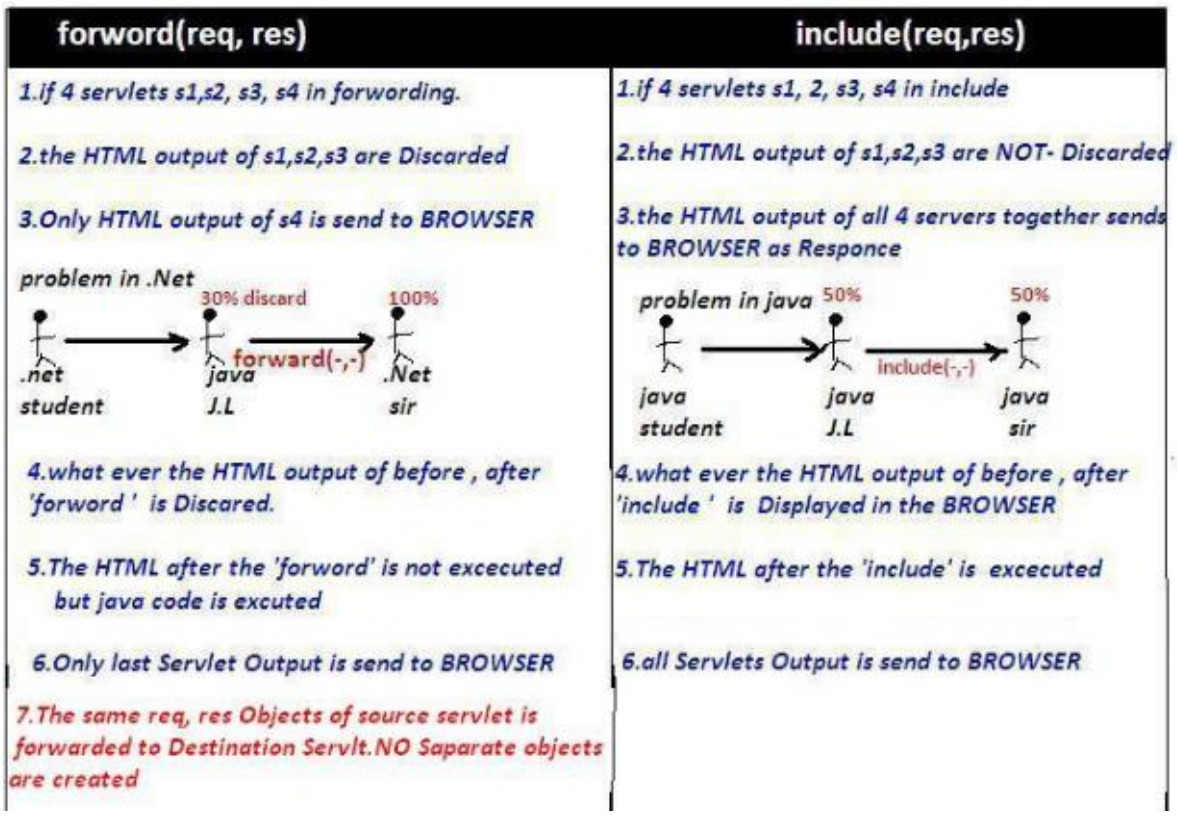
```
RequestDispatcher rd  =  context. getNamedDispatcher("serv1");
      rd.forward(req, res);
      rd.include(req, res);
```

- ✓ **/URL** → if we are placing .html, .jsp type of files we have to add **'/'** in path
- ✓ **Servltname** → if we are using logical names of servlet/jsp like serv1, serv2 etc, then **we must not to** add **'/'** in path

RequestDispatcher	NamedDispatcher
Invokable on both request & context Objects	Invokable only on context Object
Exptects servlet url-pattren logical name or filenames of .html, .jsp files as argument	Exptects only servlet url-pattren logical name as argument
RequestDisptacher Object can points destination servlets,jsps & .html pages	RequestDisptacher Object can points only destination servlets,jsps but not .html pages

Servlet chaining in Same Server

We can use forward(), include() methods to perform chaining between two servlets which are resides in same web application or different web applications of same server



forword(req, res)	include(req,res)
1.if 4 servlets s1,s2, s3, s4 in forwording.	1.if 4 servlets s1, 2, s3, s4 in include
2.the HTML output of s1,s2,s3 are Discarded	2.the HTML output of s1,s2,s3 are NOT- Discarded
3.Only HTML output of s4 is send to BROWSER	3.the HTML output of all 4 servers together sends to BROWSER as Responce
problem in .Net 30% discard 100% .net student → java J.L forward(-,-) → .Net sir	problem in java 50% 50% java student → java J.L include(-,-) → java sir
4.what ever the HTML output of before , after 'forword ' is Discared.	4.what ever the HTML output of before , after 'include ' is Displayed in the BROWSER
5.The HTML after the 'forword' is not excecuted but java code is excuted	5.The HTML after the 'include' is excecuted
6.Only last Servlet Output is send to BROWSER	6.all Servlets Output is send to BROWSER
7.The same req, res Objects of source servlet is forwarded to Destination Servlt.NO Saparate objects are created	

Forword() example

Input.html

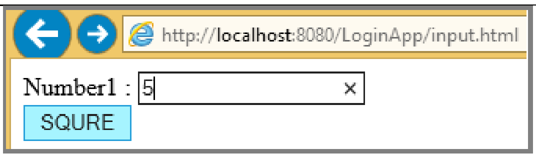
```
        <form action="s1" method="GET">
                Number1 : <input type="text" name="n1"><br>
                <input type="submit"  value="SQURE">
        </form>
```

Web.xml

```
<web-app>
        <servlet>
                <servlet-name>s1</servlet-name>
                <servlet-class>demo.srv1</servlet-class>
        </servlet>
        <servlet>
                <servlet-name>s2</servlet-name>
                <servlet-class>demo.srv2</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>s1</servlet-name>
                <url-pattern>/s1</url-pattern>
        </servlet-mapping>
        <servlet-mapping>
                <servlet-name>s2</servlet-name>
                <url-pattern>/s2</url-pattern>
        </servlet-mapping>
</web-app>
```

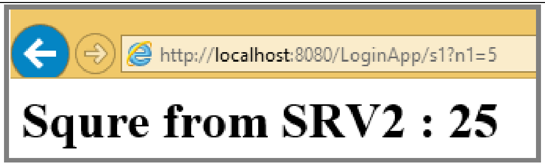
```java
public class srv1 extends HttpServlet {
      public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            ServletConfig cg = getServletConfig();
            ServletContext sc = cg.getServletContext();
            res.setContentType("text/html");
            PrintWriter pw = res.getWriter();

            String s1 = req.getParameter("n1");
            int a = Integer.parseInt(s1);
            int b = a * a;

            pw.println("<h1>Before forword            :     " + b + "</h1>");
            RequestDispatcher rd = sc.getRequestDispatcher("/s2");
            rd.forward(req, res);     //→ (1)
            pw.println("<h1> After forword</h1>");
      }

      public void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            doGet(req, res);
      }
}
```

```java
public class srv2 extends HttpServlet {
      public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            res.setContentType("text/html");
            PrintWriter pw = res.getWriter();
            String s1 = req.getParameter("n1");
            int a = Integer.parseInt(s1);
            int b = a * a;
            pw.println("<h1>Squre from SRV2            :     " + b + "</h1>");
      }
      public void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            doGet(req, res);
      }
}
```



http://localhost:8080/LoginApp/input.html

Number1 : 5

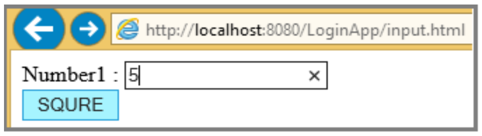


http://localhost:8080/LoginApp/s1?n1=5

Squre from SRV2 : 25

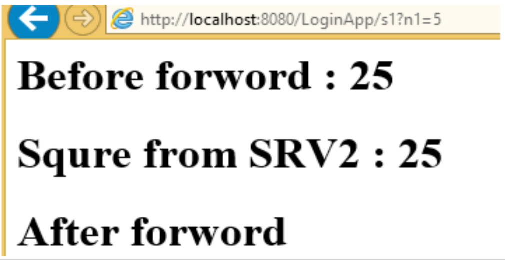
In above (1), if we just replace with **include(req, res)** as below it include serv1 result also

```java
RequestDispatcher rd = sc.getRequestDispatcher("/s2");
                  rd.include(req, res);
```



http://localhost:8080/LoginApp/input.html

Number1 : 5



http://localhost:8080/LoginApp/s1?n1=5

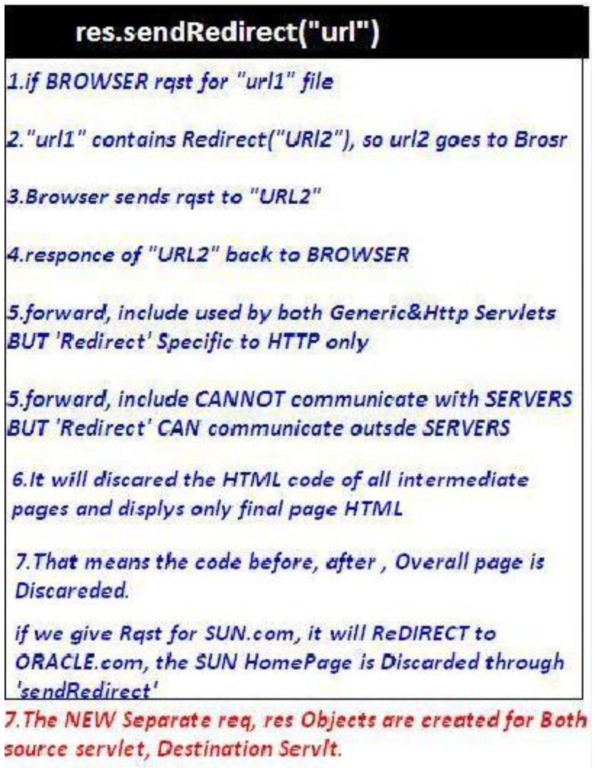
Before forword : 25

Squre from SRV2 : 25

After forword

Servlet chaining in Different Server 314 page in natraj

We can use res.sendRedirect(url) method to perform chaining between two servlets which are running on different servers



res.sendRedirect("url")

1.if BROWSER rqst for "url1" file

2."url1" contains Redirect("URl2"), so url2 goes to Brosr

3.Browser sends rqst to "URL2"

4.responce of "URL2" back to BROWSER

5.forward, include used by both Generic&Http Servlets BUT 'Redirect' Specific to HTTP only

5.forward, include CANNOT communicate with SERVERS BUT 'Redirect' CAN communicate outsde SERVERS

6.It will discared the HTML code of all intermediate pages and displys only final page HTML

7.That means the code before, after , Overall page is Discareded.

if we give Rqst for SUN.com, it will ReDIRECT to ORACLE.com, the SUN HomePage is Discarded through 'sendRedirect'

7.The NEW Separate req, res Objects are created for Both source servlet, Destination Servlt.

```
public class srv3 extends HttpServlet {
      public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            res.setContentType("text/html");
            PrintWriter pw = res.getWriter();
            pw.println("<h1>Before sendReditrect</h1>");
            res.sendRedirect("http://www.google.com");
            pw.println("<h1>After sendReditrect</h1>");

      }
      public void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            doGet(req, res);
      }
}
```

forward() method	sendRedirect() method
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only. **Example:** request.getRequestDispacher("servlet2").forward(request,response);	It can be used within and outside the server. **Example:** response.sendRedirect("servlet2");

3.8 Attribbutes

The servlet programmer can pass informations from one servlet to another using attributes. It is just like passing object from one class to another so that we can reuse the same object again and again.

We have 3 types of scopes for attribues

1) **Request scope**
2) **Session scope**
3) **Application scope (ServletContext Scope)**

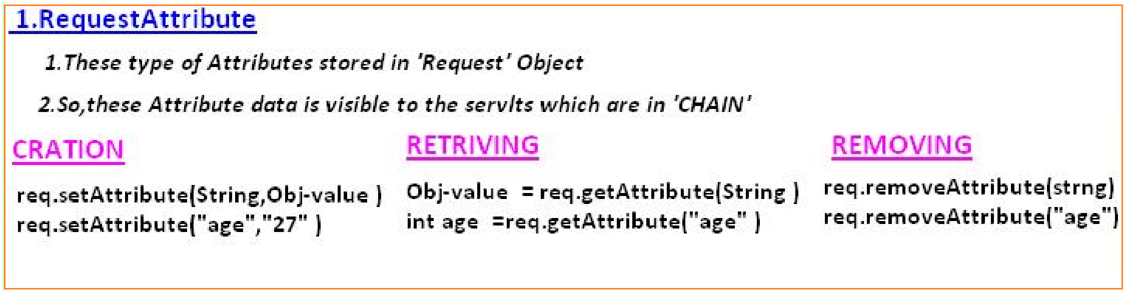
Attributes can be apply

- if Both Source& Destination Servlets are in same/difffrent webapplication
- if Both Source& Destination Servlets are in same Server
- It is **not** applicable if Both Source& Destination **Servlets are in different Server**

We have 3 methods to deal with attribues

1) **public void setAttribute(String name,Object object**
2) **public Object getAttribute(String name)**
3) **public void removeAttribute(String name)**

1. Request Attribute: It applicable only if both servlets must be in **CHAIN**

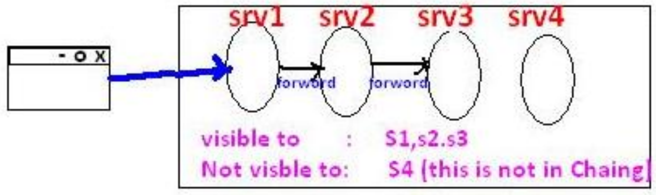


1.RequestAttribute

1.These type of Attributes stored in 'Request' Object

2.So,these Attribute data is visible to the servlts which are in 'CHAIN'

CRATION	RETRIVING	REMOVING
req.setAttribute(String,Obj-value)	Obj-value = req.getAttribute(String)	req.removeAttribute(strng)
req.setAttribute("age","27")	int age =req.getAttribute("age")	req.removeAttribute("age")

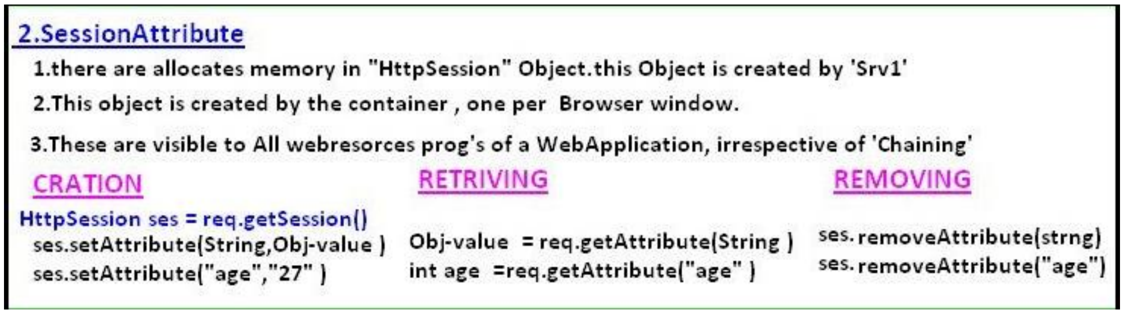


```
public class srv1 extends HttpServlet {
        public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
                req.setAttribute("uname", "ADMIN");
                req.setAttribute("pwd", "123abc$");
                RequestDispatcher rd = req.getRequestDispatcher("/s2");
                rd.forward(req, res);
        }
}
```

```
public class srv2 extends HttpServlet {
      public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            res.setContentType("text/html");
            PrintWriter pw = res.getWriter();
            pw.write("Username : "+req.getAttribute("uname"));
            pw.write("Password : "+req.getAttribute("pwd"));
      }
}
```

Output: `Username: ADMIN  Password : 123abc$`

2. Session Attribute: It applicable per **one browser window** at a time. I.e Session is maintain in single window



2.SessionAttribute

1.there are allocates memory in "HttpSession" Object.this Object is created by 'Srv1'

2.This object is created by the container , one per Browser window.

3.These are visible to All webresorces prog's of a WebApplication, irrespective of 'Chaining'

CRATION

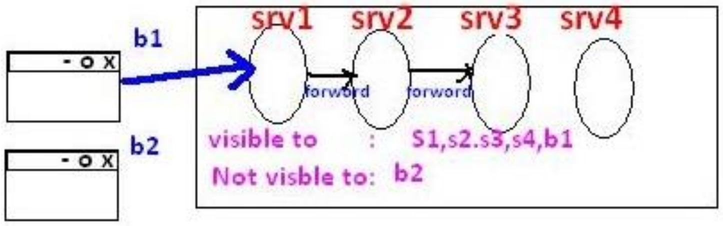
HttpSession ses = req.getSession()
ses.setAttribute(String,Obj-value)
ses.setAttribute("age","27")

RETRIVING

Obj-value = req.getAttribute(String)
int age =req.getAttribute("age")

REMOVING

ses.removeAttribute(strng)
ses.removeAttribute("age")

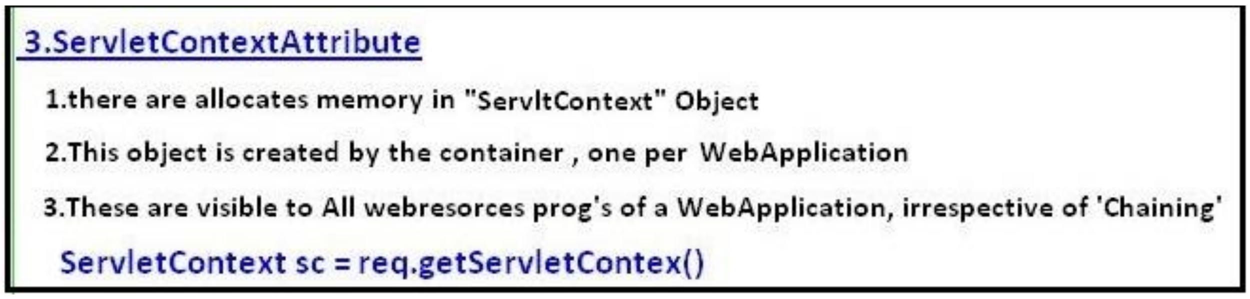


```
public class srv1 extends HttpServlet {
      public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            HttpSession sess = req.getSession();
            sess.setAttribute("id", "10001");
            sess.setAttribute("name", "Satya");
      }
}

public class srv2 extends HttpServlet {
      public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            res.setContentType("text/html");
            PrintWriter pw = res.getWriter();
            HttpSession sess = req.getSession();
            pw.write("ID : "+sess.getAttribute("id"));
            pw.write("<br>Name : "+sess.getAttribute("name"));
      }
}
```

Output → `ID : 10001   Name : Satya`

<u>3. Application /Context Attribute</u>

- It can applicable both servelts **must be in Single Server**
- No need of Servlet Chaining & Session because it is per web application



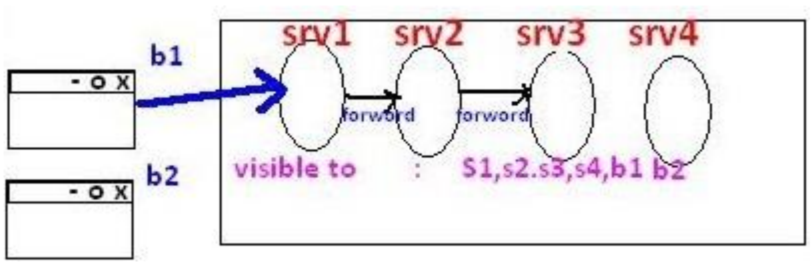
3.ServletContextAttribute

1.there are allocates memory in "ServltContext" Object

2.This object is created by the container , one per WebApplication

3.These are visible to All webresorces prog's of a WebApplication, irrespective of 'Chaining'

ServletContext sc = req.getServletContex()



```
public class srv1 extends HttpServlet {
        public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
                ServletContext cxt= req.getServletContext();
                cxt.setAttribute("name", "Johnny");
                cxt.setAttribute("age", "26");
        }
}
```

```
public class srv2 extends HttpServlet {
        public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
                res.setContentType("text/html");
                PrintWriter pw = res.getWriter();
                ServletContext cxt= req.getServletContext();
                pw.write("Name: "+cxt.getAttribute("name"));
                pw.write("Age : "+cxt.getAttribute("age"));
        }
}
```

Output: `Name: Johnny Age : 26`

<u>Stateless Behaviour:</u> is nothing but while processing current request in any web resource program is cannot use previous request data is nothing but Statteless here.

HTTP is a Stateless protocol that means each request is considered as the new request

To make our HttpServlet as a Statefull resource program we use Session Tracking

3.9 Session Tracking

Session Tracking is a way of remembering clinet data across the multiple requests during a session.

There are 4 techniques used in Session tracking:

1. **Hidden Form Field**
2. **Cookies**
3. **HttpSession**
4. **URL Rewriting**

1. Hidden Form Field

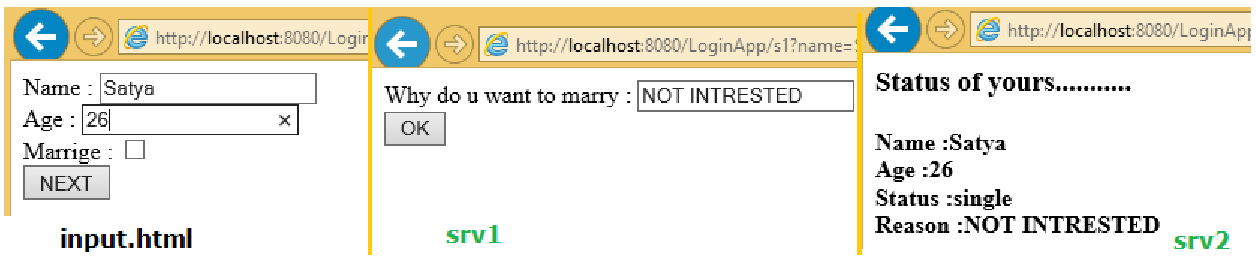
- We store the information in the hidden field and get it from another servlet
- <input type="hidden" name="uname" value="Satya">
- It easy to write

Disadvantages

- Used only on TextBoxes
- If we see the view-source of html page, the hidden values can visible
- Not secure

```
<form action = "s1"  method = "get">
      Name    :    <input type = "text" name = "name"><br>
      Age     :    <input type = "text" name = "age"><br>
      Marrige  :  <input type = "checkbox" name = "mrg" value = "yes"><br>
<input type = "submit" name = "btn" value = "NEXT"> <br>
</form>

public class srv1 extends HttpServlet {
      public void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            res.setContentType("text/html");
            PrintWriter pw = res.getWriter();

            String name = req.getParameter("name");
            String age = req.getParameter("age");
            String mrg = req.getParameter("mrg");

if (mrg == null) {
 mrg = "single";
 pw.println("<form action = 's2'>");
 pw.println("Why do u want to marry :<input type='text' name = 'why'><br>");
 pw.println("<input type = 'hidden' name = 'name' value = " + name + ">");
 pw.println("<input type = 'hidden' name = 'age' value = " + age + ">");
 pw.println("<input type = 'hidden' name = 'mrg' value = " + mrg + ">");
 pw.println("<input type = 'submit' name = 'btn' value = 'OK'><br>");
 pw.println("</form>");
}

else {
 mrg = "married";
 pw.println("<form action = 's2'>");
 pw.println("How Many Childrens:<input type = 'text' name = 'child'><br>");
 pw.println("<input type = 'hidden' name = 'name' value = " + name + ">");
 pw.println("<input type = 'hidden' name = 'age' value = " + age + ">");
 pw.println("<input type = 'hidden' name = 'mrg' value = " + mrg + ">");
 pw.println("<input type = 'submit' name = 'btn' value = 'OK'><br>");
 pw.println("</form>");
 }
}
}
```

```
public class srv2 extends HttpServlet {
      public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            res.setContentType("text/html");
            PrintWriter pw = res.getWriter();
            pw.println("<h3>  Status of yours..........</h1>");
            pw.println("<h4> Name :" + req.getParameter("name"));
            pw.println("<br> Age  :" + req.getParameter("age"));
            pw.println("<br> Status  :" + req.getParameter("mrg"));

            if (req.getParameter("mrg").equals("single")) {
                  pw.println("<br> Reason  :" + req.getParameter("why"));
            }else{
            pw.println("<br> No.of Childrents :"+ req.getParameter("child"));
            }
      }
}
```



http://localhost:8080/Login

Name : Satya
Age : 26
Marrige : ☐
NEXT

http://localhost:8080/LoginApp/s1?name=

Why do u want to marry : NOT INTRESTED
OK

http://localhost:8080/LoginApp

Status of yours..........

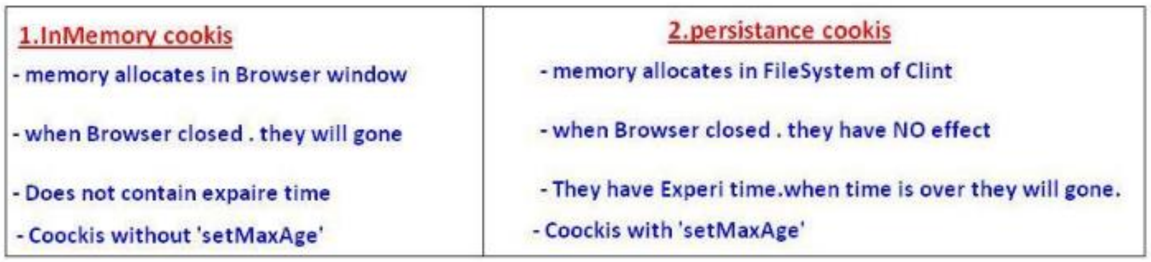
Name :Satya
Age :26
Status :single
Reason :NOT INTRESTED

2. Cookies

A cookie is a small piece of information saved in the browser between the multiple client requests.

There are 2 types of cookies in servlets.

1. **Non-persistent cookie**
2. **Persistent cookie**



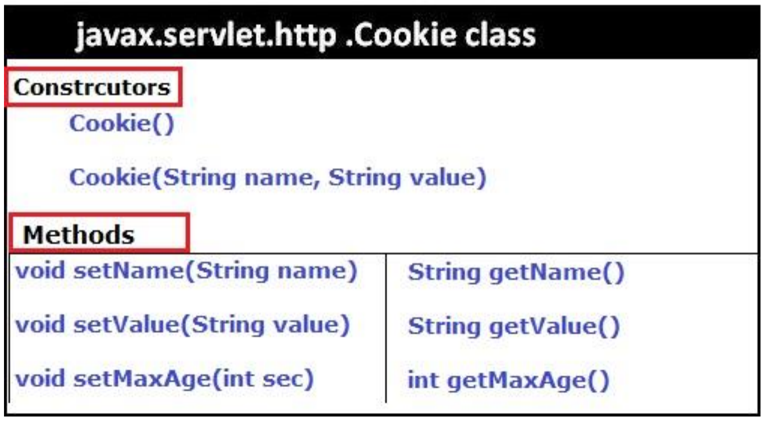
1.InMemory cookis	2.persistance cookis
- memory allocates in Browser window	- memory allocates in FileSystem of Clint
- when Browser closed . they will gone	- when Browser closed . they have NO effect
- Does not contain expaire time	- They have Experi time.when time is over they will gone.
- Coockis without 'setMaxAge'	- Coockis with 'setMaxAge'

Advantage of Cookies

- Simplest technique of maintaining the state.
- Cookies are maintained at client side.

Disadvantage of Cookies

- It will not work if cookie is disabled from the browser.
- Only textual information can be set in Cookie object.



javax.servlet.http .Cookie class

Constrcutors

Cookie()

Cookie(String name, String value)

Methods

void setName(String name)	String getName()
void setValue(String value)	String getValue()
void setMaxAge(int sec)	int getMaxAge()

adding Cookie

public void addCookie(Cookie c)

```
res.addCookie(c1);
```

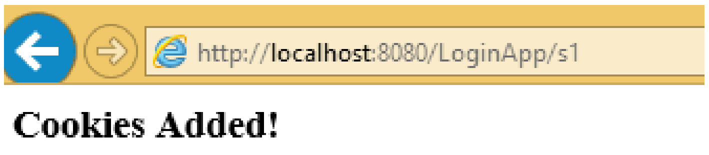
Retriving Cookie

public Cookie[] getCookies()

```
Cookie c[] = req.getCookies();
```

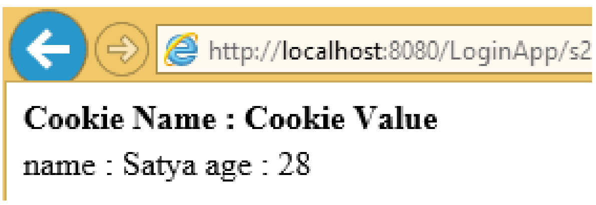
```
public class srv1 extends HttpServlet {
	public void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
		res.setContentType("text/html");
		PrintWriter pw = res.getWriter();
		Cookie c1 = new Cookie("name", "Satya");
		Cookie c2 = new Cookie("age", "28");
		c1.setMaxAge(5000); //max 5 sec alive
		res.addCookie(c1);
		res.addCookie(c2);
		pw.write("<h3>Cookies Added!");
	}
}
```

```
public class srv2 extends HttpServlet {
	public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
		res.setContentType("text/html");
		PrintWriter pw = res.getWriter();
		Cookie c[] = req.getCookies();
		pw.println("<h4> Cookie Name  : Cookie Value   </h4>");
		for (int i = 0; i < c.length; i++) {
			pw.println(c[i].getName() + " : " + c[i].getValue());
		}
	}
}
```



http://localhost:8080/LoginApp/s1

Cookies Added!



http://localhost:8080/LoginApp/s2

Cookie Name : Cookie Value

name : Satya age : 28

3. Http Session

- HttpSession Object memory **allocates in Server**
- it remembers the client data across the multiple requests in the form of **Session Attribute values**
- Every Session object contains **SessionID, & stored in browser.**
- Session of a browser can be identified by SessionID.

Constrcurors

1. **public HttpSession getSession():**
    - Returns the current session associated with this request
    - If the request does not have a session, creates one.

2. **public HttpSession getSession(boolean create**
    - Returns the current session associated with this request
    - **True** → request does not have a session, creates new session.
    - **False** → request does not have a session, it wont create new session

Methods

- **public String getId()** :Returns a string containing the unique identifier value.
- **public long getCreationTime()** :Returns the time when this session was created
- public long **getLastAccessedTime():**Returns the last time the client sent a
- **public void invalidate():** Invalidates this session then unbinds any objects bound to it.

```
public class srv1 extends HttpServlet {
      public void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            res.setContentType("text/html");
            PrintWriter pw = res.getWriter();
            HttpSession ses = req.getSession();
            ses.setAttribute("name", "Ravi");
            ses.setAttribute("city", "HYD");
            pw.write("<h3>Session Added!");
      }
}
```

```
public class srv2 extends HttpServlet {
      public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
            res.setContentType("text/html");
            PrintWriter pw = res.getWriter();
            HttpSession sess =      req.getSession();
            pw.write("Name : "+sess.getAttribute("name"));
            pw.write("City : "+sess.getAttribute("city"));
      }
}
```

Output : Name : Ravi , City : HYD

4. URL Rewriting

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource.

We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

Advantage of URL Rewriting

- It will always work whether cookie is disabled or not (browser independent).
- Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

- It will work only with links.
- It can send only textual information.
- Not Secure, user can read the information what we are sending

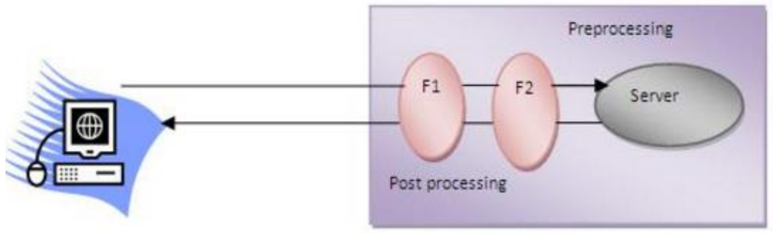
```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

```
public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String n=request.getParameter("userName");
        out.print("Welcome "+n);

        //appending the username in the query string
        out.print("<a href='servlet2?uname="+n+"'>visit</a>");

        out.close();
    }
}
```

```
public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        //getting value from the query string
        String n=request.getParameter("uname");
        out.print("Hello "+n);
        out.close();
    }
}
```

3.10 Filters

A filter is invoked at the **preprocessing and postprocessing of a request.**



Filter is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet. So it will be easier to maintain the web application.

Usage of Filter

- recording all incoming requests
- logs the IP addresses of the computers from which the requests originate
- conversion
- data compression
- encryption and decryption
- Input validation etc.

Advantage of Filter

- Filter is pluggable.
- One filter don't have dependency onto another resource.
- Less Maintenance

Filter API

Like servlet filter have its own API.**The javax.servlet** package contains the **3 interfaces**

1. **Filter**
2. **FilterChain**
3. **FilterConfig**

1) Filter interface

For creating any filter, you must implement the Filter interface.Filter interface provides the life cycle methods for a filter.

Method	**Description**
public void init(FilterConfig config)	init() method is invoked only once. It is used to initialize the filter.
Public void doFilter(HttpServletRequest req,HttpServletResponse res, FilterChain chain)	doFilter() method is invoked every time when user request to any resource, to which the filter is mapped.It is used to perform filtering tasks.
public void destroy()	This is invoked only once when filter is taken out of the service.

2) FilterChain interface

The object of FilterChain is responsible to invoke the next filter or resource in the chain.This object is passed in the doFilter method of Filter interface.The FilterChain interface contains only one method:

public void doFilter(HttpServletRequest, HttpServletResponse):
It passes the control to the next filter or resource.

Example

index.html

```
<a href="servlet1">click here</a>
```

```
public class MyFilter implements Filter{

public void init(FilterConfig arg0) throws ServletException {}

public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws IOException, Serv
letException {

    PrintWriter out=resp.getWriter();
    out.print("filter is invoked before");

    chain.doFilter(req, resp);//sends request to next resource

    out.print("filter is invoked after");
    }
    public void destroy() {}
}
```

```
public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.print("<br>welcome to servlet<br>");
    }
}
```

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>HelloServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<filter>
<filter-name>f1</filter-name>
<filter-class>MyFilter</filter-class>
</filter>

<filter-mapping>
<filter-name>f1</filter-name>
<url-pattern>/servlet1</url-pattern>
</filter-mapping>

</web-app>
```

FilterConfig

An object of FilterConfig is created by the web container. This object can be used to get the configuration information from the web.xml file.

1. **public void init(FilterConfig config):** init() method is invoked only once it is used to initialize the filter.
2. **public String getInitParameter(String parameterName):** Returns the parameter value for the specified parameter name.
3. **public java.util.Enumeration getInitParameterNames():** Returns an enumeration containing all the parameter names.
4. **public ServletContext getServletContext():** Returns the ServletContext object

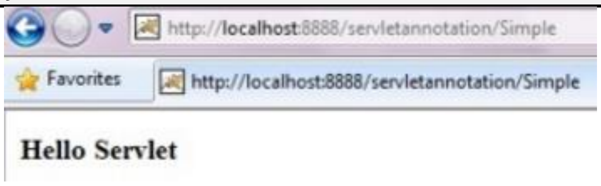
```
<filter>
  <filter-name>f1</filter-name>
  <filter-class>MyFilter</filter-class>
        <init-param>
                <param-name>age</param-name>
                <param-value>27</param-value>
        </init-param>
  </filter>

  <filter-mapping>
  <filter-name>f1</filter-name>
  <url-pattern>/servlet1</url-pattern>
  </filter-mapping>
```

Servlet with Annotation (feature of servlet3):

- Annotation represents the metadata.
- If you use annotation, deployment descriptor (web.xml file) is not required.
- But you should have tomcat7 as it will not run in the previous versions of tomcat.

@WebServlet("/url") annotation is used to map the servlet with the specified name.

```
@WebServlet("/Simple")
public class Simple extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
                            throws ServletException, IOException {


        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

        out.print("<html><body>");
        out.print("<h3>Hello Servlet</h3>");
        out.print("</body></html>");
    }
}
```


http://localhost:8888/servletannotation/Simple
Favorites
http://localhost:8888/servletannotation/Simple

Hello Servlet