Servlets

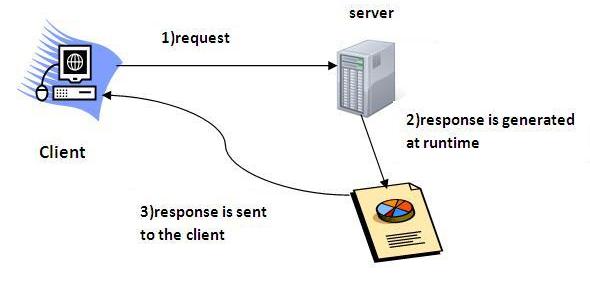
**Servlet** technology is used to create web application (resides at server side and generates dynamic web page).

**Servlet** technology is robust and scalable as it uses the java language.

## What is a Servlet?

Servlet can be described in many ways, depending on the context.

* **Servlet** is a technology i.e. used to create web application.
* **Servlet API** that provides many interfaces and classes including documentations.
* **Servlet** **interface** that must be implemented for creating any Servlet.
* **Servlet is a web component that is deployed on the server to create dynamic web page**.

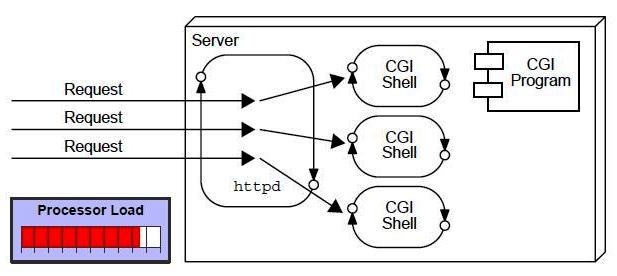


### What is web application?

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.

### CGI (Common Gateway Interface)

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.

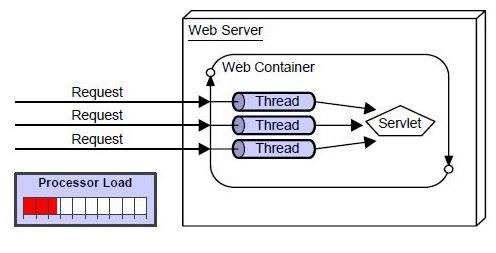


### Disadvantages of CGI

There are many problems in CGI technology:

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

### Advantage of Servlet



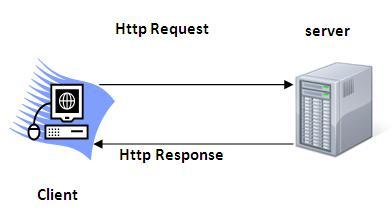
There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, light weight, cost of communication between the threads are low. The basic benefits of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request not process.
2. **Portability:** because it uses java language.
3. **Robust:** Servlets are managed by JVM so no need to worry about memory leak, garbage collection etc.
4. **Secure:** because it uses java language.

# Servlet Terminology

### HTTP (Hyper Text Transfer Protocol)

1. Http is the protocol that allows web servers and browsers to exchange data over the web.
2. It is a request response protocol.
3. Http uses reliable TCP connections by default on TCP port 80.
4. It is **stateless** means each request is considered as the new request. In other words, server doesn't recognize the user by default.



### Http Request Methods

Every request has a header that tells the status of the client. There are many request methods. Get and Post requests are mostly used.

The http request methods are:

* **GET (default method)**
* **POST**
* HEAD
* PUT
* DELETE
* OPTIONS
* TRACE

### What is the difference between Get and 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. |

### Anatomy of Get Request



### Anatomy of Post Request

In case of post request original data is sent in message body. Let's see how information

are passed to the server in case of post request.



### Container (Web Container)

It provides runtime environment for Java EE applications.

It performs many operations that are given below:

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

### Server

It is a running program or software that provides services.

There are two types of servers:

1. Web Server
2. Application Server

### Web Server

Web server contains only Web or Servlet container. It can be used for Servlet, JSP, struts, JSF etc. It can't be used for EJB.

Examples of Web Servers are: **Apache Tomcat** and **Resin**.

### Application Server

Application server contains Web and EJB containers. It can be used for Servlet, JSP, struts, JSF, EJB etc.

Examples of Application Servers are:

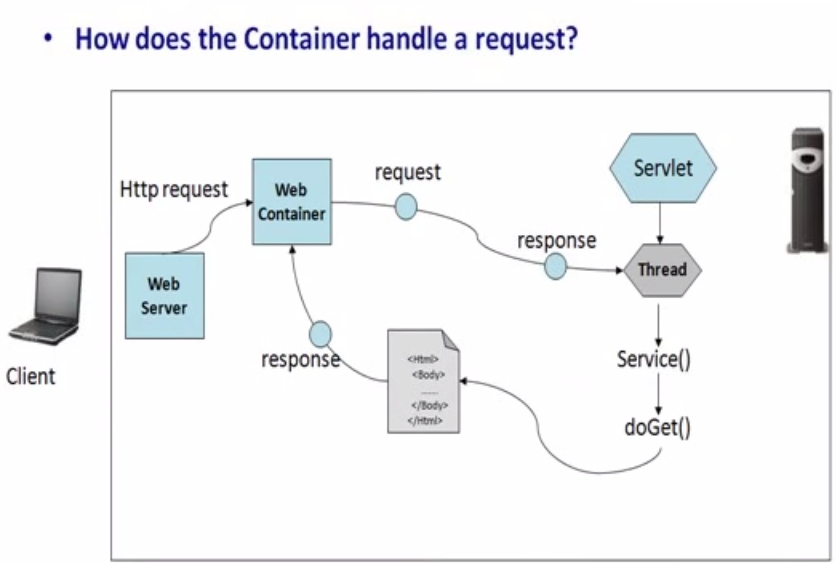
1. **JBoss** Open-source server from JBoss community.
2. **Glassfish** provided by Sun Microsystems. Now acquired by Oracle.
3. **Weblogic** provided by Oracle. It more secured.
4. **Websphere** provided by IBM.

### 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.

There are many content types:

* text/html
* text/plain
* image/jpeg



# Servlet API

### Servlet Interface

### Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of Servlet. These are invoked by the web container.

|  |  |
| --- | --- |
| 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 request,ServletResponse response)** | Provides response for the incoming request. It is invoked at each request by the web container. |
| **public void destroy()** | 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 |

### Servlet Example by implementing Servlet interface

**import** java.io.\*;

**import** javax.servlet.\*;

**public** **class** First **implements** Servlet{

ServletConfig config=**null**;

**public** **void** init(ServletConfig config){

**this**.config=config;

System.out.println("servlet is initialized");

}

**public** **void** service(ServletRequest req,ServletResponse res)

**throws** IOException,ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

out.print("<html><body>");

out.print("<b>hello simple servlet</b>");

out.print("</body></html>");

}

**public** **void** destroy(){

System.out.println("servlet is destroyed");

}

**public** ServletConfig getServletConfig(){

**return** config;

}

**public** String getServletInfo(){

**return** "copyright 2007-1010";

}

}

GenericServlet class

**GenericServlet** class implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

### Methods of GenericServlet class

There are many methods in GenericServlet class. They are as follows:

|  |  |
| --- | --- |
| Method | Description |
| **public void init(ServletConfig config)** | is used to initialize the servlet. |
| **public abstract void service(ServletRequest request, ServletResponse response)** | 1. provides service for the incoming request. It is invoked at each time when user requests for a servlet. |
| **public void destroy()** | 1. is invoked only once throughout the life cycle and indicates that servlet is being destroyed. |
| **public ServletConfig getServletConfig()** | returns the object of ServletConfig. |
| **public String getServletInfo()** | 1. returns information about servlet such as writer, copyright, version etc. |
| **public void init()** | 1. it is a convenient method for the servlet programmers, now there is no need to call super.init(config) |
| **public ServletContext getServletContext()** | 1. returns the object of ServletContext. |
| **public String getInitParameter(String name)** | 1. returns the parameter value for the given parameter name. |
| **public String getServletName()** | 1. returns the name of the servlet object. |

### Servlet Example by extending the GenericServlet class

**import** java.io.\*;

**import** javax.servlet.\*;

**public** **class** First **extends** GenericServlet{

**public** **void** service(ServletRequest req,ServletResponse res)

**throws** IOException,ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

out.print("<html><body>");

out.print("<b>hello generic servlet</b>");

out.print("</body></html>");

}

}

HttpServlet class

### Methods of HttpServlet class

There are many methods in HttpServlet class. They are as follows:

|  |  |
| --- | --- |
| **Method** | Description |
| ****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. |
| ****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. |
| ****protected void doGet(HttpServletRequest req, HttpServletResponse res)**** | handles the GET request. It is invoked by the web container. |
| ****protected void doPost(HttpServletRequest req, HttpServletResponse res)**** | handles the POST request. It is invoked by the web container. |

### Servlet Example by extending the HttpServlet class

**import** javax.servlet.http.\*;

**import** javax.servlet.\*;

**import** java.io.\*;

**public** **class** DemoServlet **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");//setting the content type

PrintWriter pw=res.getWriter();//get the stream to write the data

//writing html in the stream

pw.println("<html><body>");

pw.println("Welcome to servlet");

pw.println("</body></html>");

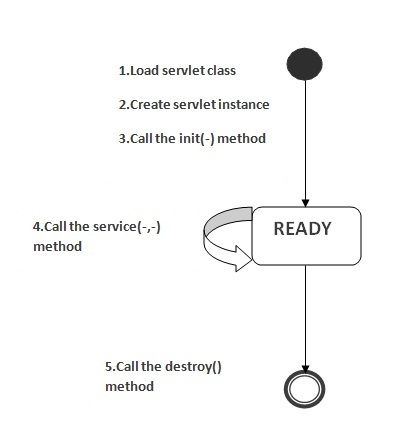
pw.close();//closing the stream

}

}

Life Cycle of a Servlet (Servlet Life Cycle)

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:



1. Servlet class is loaded.
2. Servlet instance is created.
3. Init() method is invoked.
4. service method is invoked.
5. destroy method is invoked.

### 1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

### 2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

### 3) init method is invoked

|  |
| --- |
| The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. |

### 4) service method is invoked

The web container calls the service method each time when request for the servlet is received.

### 5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc.

How Servlet works?

The server checks if the servlet is requested **for the first time**.

**If yes,** web container does the following tasks:

* loads the servlet class.
* instantiates the servlet class.
* calls the init method passing the ServletConfig object

**else**

* calls the service method passing request and response objects

The web container calls the destroy method when it needs to remove the servlet such as at time of stopping server or undeploying the project.

### How web container handles the servlet request?

The web container is responsible to handle the request. Let's see how it handles the request.

**http://localhost:8080/ServletDemo/hello**

* maps the request with the servlet in the web.xml file.
* creates request and response objects for the client request
* calls the service method **on the thread**
* The public service method internally calls the protected service method
* The protected service method calls the doXXX method depending on the type of request. (the default request method is GET and hence doGet() is called)
* The doXXX() method generates the response and it is passed to the client.
* **After sending the response, the web container deletes the request and response objects.**

welcome-file-list in web.xml

The **welcome-file-list** element of **web-app**, is used to define a list of welcome files. Its sub element is **welcome-file** that is used to define the welcome file.

A **welcome file** is the file that is invoked automatically by the server, if you don't specify any file name.

By default server looks for the welcome file in following order:

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

If none of these files are found, server renders 404 error.

# load on startup in web.xml

The **load-on-startup** element of **web-app** loads the servlet at the time of deployment or server start if value is positive. It is also known as **pre initialization of servlet**.

You can pass positive and negative value for the servlet.

#### Advantage of load-on-startup element

As you know well, servlet is loaded at first request. That means it consumes more time at first request. If you specify the load-on-startup in web.xml, servlet will be loaded at project deployment time or server start. So, it will take **less time** for responding to first request.

*web.xml*

**<web-app>**

 ....

**<servlet>**

**<servlet-name>**servlet1**</servlet-name>**

**<servlet-class>**com.javatpoint.FirstServlet**</servlet-class>**

**<load-on-startup>**0**</load-on-startup>**

**</servlet>**

**<servlet>**

**<servlet-name>**servlet2**</servlet-name>**

**<servlet-class>**com.javatpoint.SecondServlet**</servlet-class>**

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

**</servlet>**

 ...

**</web-app>**

There are defined 2 servlets, both servlets will be loaded at the time of project deployment or server start. But, servlet1 will be loaded first then servlet2.

#### Passing negative value

If you pass the negative value, servlet will be loaded at request time, at first request.

ServletRequest Interface

An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.

### Methods of ServletRequest interface

|  |  |
| --- | --- |
| **Method** | **Description** |
| **public String getParameter(String name)** | is used to obtain the value of a parameter by name. |
| **public String[] getParameterValues(String name)** | returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box. |
| **java.util.Enumeration getParameterNames()** | returns an enumeration of all of the request parameter names. |
| **public int getContentLength()** | Returns the size of the request entity data, or -1 if not known. |
| **public String getCharacterEncoding()** | Returns the character set encoding for the input of this request. |
| **public String getContentType()** | Returns the Internet Media Type of the request entity data, or null if not known. |
| **public ServletInputStream getInputStream() throws IOException** | Returns an input stream for reading binary data in the request body. |
| **public abstract String getServerName()** | Returns the host name of the server that received the request. |
| **public int getServerPort()** | Returns the port number on which this request was received. |

### Example of ServletRequest to display the name of the user

**welcome.html**

<form action="welcome" method="get">

Enter your name<input type="text" name="name"><br>

<input type="submit" value="login">

</form>

**welcome.java**

**public** **class** DemoServ **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();

String name=request.getParameter("name");//will return value

out.println("Welcome "+name);

out.close();

}

}

### Example of displaying all the header information in the servlet

|  |
| --- |
| The getHeaderNames() of ServletRequest interface returns an Enumeration object, containing all the header names. The getHeader() method of ServletRequest interface returns the header value for the given header name. In this example, we are displaying all the header information of a request in the servlet page. |

#### Syntax of getHeaderNames() method

**public** Enumeration getHeaderNames()

#### Syntax of getHeader() method

**public** String getHeader(String headerName)

**index.html**

<a href="shiowheaders"> Show Header Information </a>

**ShowHeader.java**

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("HTTP headers sent by your client:<br>");

Enumeration enums = request.getHeaderNames();

**while** (enums.hasMoreElements()) {

String headerName = (String) enums.nextElement();

String headerValue = request.getHeader(headerName);

out.print("<b>"+headerName + "</b>: ");

out.println(headerValue + "<br>");

}

}

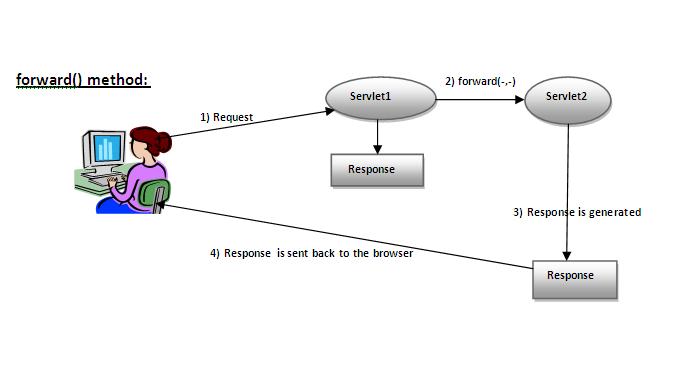
RequestDispatcher in Servlet

The RequestDispacher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp.

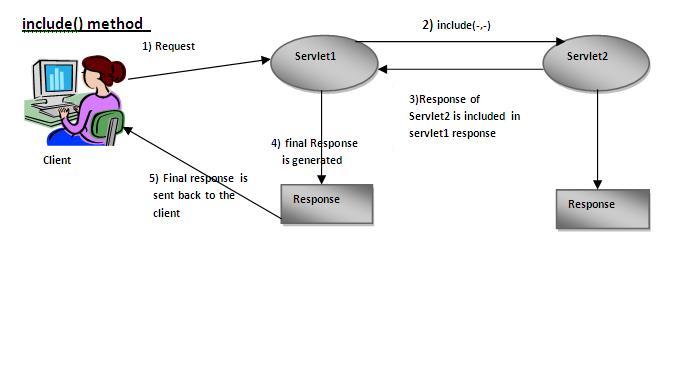
### Methods of RequestDispatcher interface

The RequestDispatcher interface provides two methods. They are:

1. **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
2. **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Includes the content of a resource (servlet, JSP page, or HTML file) in the response.



As you see in the above figure, response of second servlet is sent to the client. Response of the first servlet is not displayed to the user.



As you can see in the above figure, response of second servlet is included in the response of the first servlet that is being sent to the client.

### How to get the object of RequestDispatcher

The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher. Syntax:

#### Syntax of getRequestDispatcher method

**public** RequestDispatcher getRequestDispatcher(String resource);

#### Example of using getRequestDispatcher method

RequestDispatcher rd=request.getRequestDispatcher("servlet2");

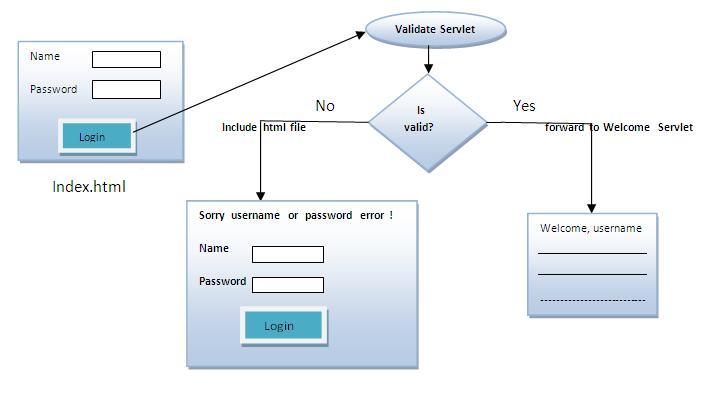
//servlet2 is the url-pattern of the second servlet

rd.forward(request, response);//method may be include or forward

### Example of RequestDispatcher interface

In this example, we are validating the password entered by the user. If password is servlet, it will forward the request to the WelcomeServlet, otherwise will show an error message: sorry username or password error!. In this program, we are cheking for hardcoded information. But you can check it to the database also that we will see in the development chapter. In this example, we have created following files:

* **index.html file:** for getting input from the user.
* **Login.java file:** a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.
* **WelcomeServlet.java file:** a servlet class for displaying the welcome message.
* **web.xml file:** a deployment descriptor file that contains the information about the servlet.



**Login.html**

<body>

<form action=*"servlet1"* method=*"post"*>

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

Password:<input type=*"password"* name=*"userPass"*/><br/>

<input type=*"submit"* value=*"login"*/>

</form>

</body>

**Login.java**

public class Login extends HttpServlet {

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String n=request.getParameter("userName");

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

if(p.equals("servlet")){

RequestDispatcher rd=request.getRequestDispatcher("servlet2");

rd.forward(request, response);

}

else{

out.print("Sorry UserName or Password Error!");

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

rd.include(request, response);

}

}

}

**Welcome.java**

public class WelcomeServlet extends HttpServlet {

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String n=request.getParameter("userName");

out.print("Welcome "+n);

}

}

**Web.xml**

<servlet>

<description></description>

<display-name>Login</display-name>

<servlet-name>Login</servlet-name>

<servlet-class>servlets.Login</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Login</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

<servlet>

<description></description>

<display-name>WelcomeServlet</display-name>

<servlet-name>WelcomeServlet</servlet-name>

<servlet-class>servlets.WelcomeServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>WelcomeServlet</servlet-name>

<url-pattern>/servlet2</url-pattern>

</servlet-mapping>

SendRedirect in servlet

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

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

## Difference between forward() and sendRedirect() method

|  |  |
| --- | --- |
| 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. | It can be used within and outside the server. |
| Example: request.getRequestDispacher("servlet2").forward(request,response); | Example: response.sendRedirect("servlet2"); |

### example of sendRedirect method in servlet

**invoking google through java servlet:**

**SendRedirectDemo.java**

public class SendRedirectDemo extends HttpServlet {

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

response.setContentType("text/html");

PrintWriter pw=response.getWriter();

response.sendRedirect("http://www.google.com");

pw.close();

}

}

**Custom Google Search:**

**Googlesearch.html**

<body>

<form action="MySearcher">

<input type="text" name="name">

<input type="submit" value="Google Search">

</form> </body>

**MySearcher.java**

public class MySearcher extends HttpServlet {

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

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

response.sendRedirect("https://www.google.co.in/#q="+name);

}

}

ServletConfig Interface

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

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

### Methods of ServletConfig interface

**public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.

**public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.

**public String getServletName():**Returns the name of the servlet.

**public ServletContext getServletContext():**Returns an object of ServletContext.

### Example of ServletConfig to get initialization parameter

**ServletConfigDemo.java**

public class ServletConfigDemo extends HttpServlet {

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

ServletConfig config=getServletConfig();

String company=config.getInitParameter("company");

out.print("My Company Name is: <h2>"+company + "</h2>");

out.close();

}

}

**Web.xml**

<servlet>

<display-name>ServletConfigDemo</display-name>

<servlet-name>ServletConfigDemo</servlet-name>

<servlet-class>servlets.ServletConfigDemo</servlet-class>

<init-param>

**<param-name>company</param-name>**

**<param-value>Impulse Software</param-value>**

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>ServletConfigDemo</servlet-name>

<url-pattern>/ServletConfigDemo</url-pattern>

</servlet-mapping>

### Example of ServletConfig to get all the initialization parameters

public class ServletAllConfigParametersDemo extends HttpServlet {

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

ServletConfig config=getServletConfig();

Enumeration<String> e=config.getInitParameterNames();

out.println("<h2 style='color:red'>Config Parameters Information</h2><hr /><p></p>");

out.println("<html><body style='background-color:lightgray'><table border=2><th>Parameter Name</th><th>Parameter Value</th>");

String str="";

while(e.hasMoreElements()){

str=e.nextElement();

out.println("<tr><td>" + str + "</td><td>" + config.getInitParameter(str) + "</td></tr>");

}

out.println("</table></body></html>");

out.close();

}

}

**Web.xml**

<servlet>

<description></description>

<display-name>ServletAllConfigParametersDemo</display-name>

<servlet-name>ServletAllConfigParametersDemo</servlet-name>

<servlet-class>servlets.ServletAllConfigParametersDemo</servlet-class>

**<init-param>**

**<description></description>**

**<param-name>customername</param-name>**

**<param-value>srinivas</param-value>**

**</init-param>**

**<init-param>**

**<description></description>**

**<param-name>email</param-name>**

**<param-value>dachepallisrinivas@gmail.com</param-value>**

**</init-param>**

**<init-param>**

**<description></description>**

**<param-name>mobile</param-name>**

**<param-value>091-09247175823</param-value>**

**</init-param>**

**<init-param>**

**<description></description>**

**<param-name>location</param-name>**

**<param-value>vizag</param-value>**

**</init-param>**

</servlet>

<servlet-mapping>

<servlet-name>ServletAllConfigParametersDemo</servlet-name>

<url-pattern>/ServletAllConfigParametersDemo</url-pattern>

</servlet-mapping>

ServletContext Interface

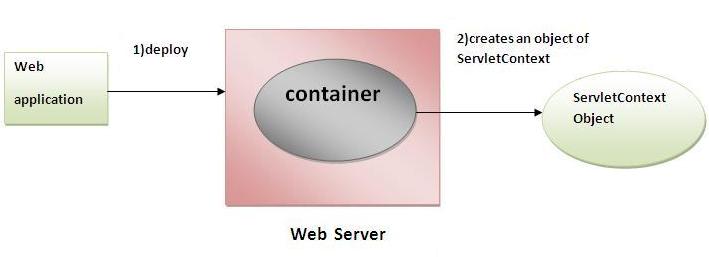
An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlets, it is better to provide it from the web.xml file using the **<context-param>** element.

### Usage of ServletContext Interface

There can be a lot of usage of ServletContext object. Some of them are as follows:

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.



### Commonly used methods of ServletContext interface

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| There is given some commonly used methods of ServletContext interface.   1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name. 2. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters. 3. **public void setAttribute(String name,Object object):**sets the given object in the application scope. 4. **public Object getAttribute(String name):**Returns the attribute for the specified name. 5. **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context.  |  |  | | --- | --- | | Context parameters | Config parameters | | Available to any servlets and JSPs that are part of web | Available to only servlet for which it was configured | | Context parameters is initialized within the < web-app> not within a specific < servlet> element | Initialized within the < servlet> for each specific servlet | | ServletContext object is used to get Context parameters | ServletConfig object is used to get Config parameters | | Only one ServletContext object for entire web app | Each servlet has its own ServletConfig object |   **ServletContextDemo.java**  public class ServletContextDemo extends HttpServlet {  private static final long serialVersionUID = 1L;  /\*\*  \* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  response.setContentType("text/html");  PrintWriter pw=response.getWriter();    //creating ServletContext object  ServletContext context=getServletContext();    //Getting the value of the initialization parameter and printing it  String course=context.getInitParameter("course");  pw.println("course name is="+course);    pw.close();  }  } Example of ServletContext to get all the initialization parameters **ServletAllContextParametersDemo.java** |

public class ServletAllContextParametersDemo extends HttpServlet {

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

response.setContentType("text/html");

PrintWriter out=response.getWriter();

ServletContext context=getServletContext();

Enumeration<String> e=context.getInitParameterNames();

String str="";

while(e.hasMoreElements()){

str=e.nextElement();

out.print("<br> "+str + " : " + context.getInitParameter(str));

}

}

}

**Web.xml**

<context-param>

<param-name>course</param-name>

<param-value>Java EE</param-value>

</context-param>

<context-param>

<param-name>duration</param-name>

<param-value>6 Months</param-value>

</context-param>

<context-param>

<param-name>fee</param-name>

<param-value>5000</param-value>

</context-param>

Attribute in Servlet

An **attribute in servlet** is an object that can be set, get or removed from one of the following scopes:

1. request scope
2. session scope
3. application scope

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.

### Attribute specific methods of ServletRequest, HttpSession and ServletContext interface:

**public void setAttribute(String name,Object object):**sets the given object in the application scope.

**public Object getAttribute(String name):**Returns the attribute for the specified name.

**public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.

**public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context.

### Example of ServletContext to set and get attribute

**ContextAttributeSetServlet.java**

public class ContextAttributeSetServlet extends HttpServlet {

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

try{

response.setContentType("text/html");

PrintWriter out=response.getWriter();

ServletContext context=getServletContext();

context.setAttribute("company","Impulse Software");

out.println("Welcome to first servlet");

out.println("<a href='ContextAttributeGetServlet'>visit</a>");

out.close();

}catch(Exception e){

System.out.println(e.getMessage());

}

}

}

**ContextAttributeGetServlet.java**

public class ContextAttributeGetServlet extends HttpServlet {

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

try{

response.setContentType("text/html");

PrintWriter out=response.getWriter();

ServletContext context=getServletContext();

String n=(String)context.getAttribute("company");

out.println("Welcome to "+n);

out.close();

}catch(Exception e){

System.out.println(e.getMessage());

}

}

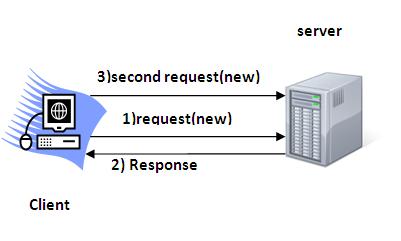
}

Session Tracking in Servlets

**Session** simply means a particular interval of time.

**Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.



### Why use Session Tracking?

**To recognize the user**

### Session Tracking Techniques

There are four techniques used in Session tracking:

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

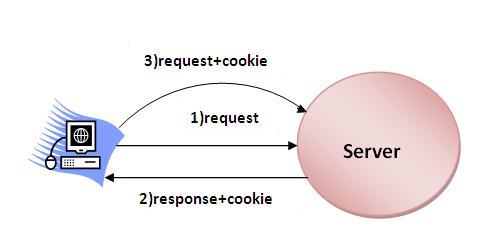
Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

### How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, server recognizes the user as the old user.



### Types of Cookie

There are 2 types of cookies in servlets.

1. Browser Based cookie
2. Persistent cookie Or Durable Cookie

### Browser Based cookie

It is **valid for single session** only. It is removed each time when user closes the browser.

### Persistent cookie

It is **valid for multiple sessions**. It is not removed each time when user closes the browser. It is removed only if user logout or sign out.

### Advantage of Cookies

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

### Disadvantage of Cookies

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

#### Gmail uses cookie technique for login. If you disable the cookie, gmail won't work.

### Cookie class

**javax.servlet.http.Cookie** class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

### Constructor of Cookie class

|  |  |
| --- | --- |
| Constructor | Description |
| Cookie() | constructs a cookie. |
| Cookie(String name, String value) | constructs a cookie with a specified name and value. |

### Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

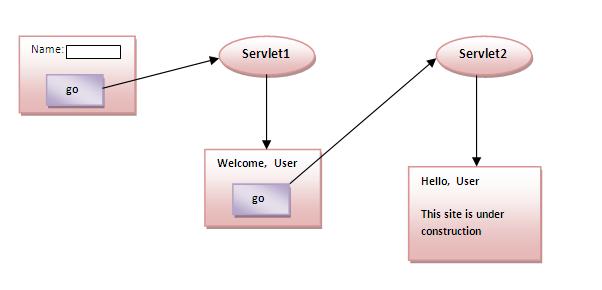
|  |  |
| --- | --- |
| Method | Description |
| public void setMaxAge(int expiry) | Sets the maximum age of the cookie in seconds. |
| public String getName() | Returns the name of the cookie. The name cannot be changed after creation. |
| public String getValue() | Returns the value of the cookie. |
| public void setName(String name) | changes the name of the cookie. |
| public void setValue(String value) | changes the value of the cookie. |

### Other methods required for using Cookies

|  |
| --- |
| For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:   1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object. 2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser. |

### Simple example of Servlet Cookies

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



**Cookie.html**

<body>

<form action="StoreCookieServlet" method="post">

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

<input type="submit" value="go"/>

</form>

</body>

**StoreCookieServlet.java**

public class StoreCookieServlet extends HttpServlet {

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

try{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String n=request.getParameter("userName");

out.print("Welcome "+n);

Cookie ck=new Cookie("uname",n);//creating cookie object

response.addCookie(ck);//adding cookie in the response

//creating submit button

out.print("<form action='GetCookieServlet' method='post'>");

out.print("<input type='submit' value='go'>");

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

out.close();

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

}

}

**GetCookieServlet.java**

public class GetCookieServlet extends HttpServlet {

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

try{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

Cookie ck[]=request.getCookies();

out.print("Hello "+ck[0].getValue());

out.close();

}catch(Exception e){

System.out.println(e);

}

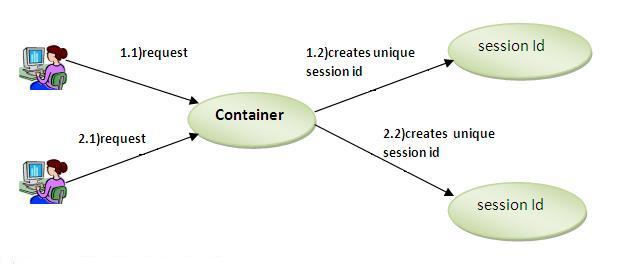
}

}

HttpSession interface

Web container creates a session id for each user.The container uses this id to identify the particular user. An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



### How to get the HttpSession object ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

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

### Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

### Example of using HttpSession

In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the setAttribute() method of HttpSession interface and to get the attribute, we have used the getAttribute method.

### index.html

1. <form action="servlet1">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>

### FirstServlet.java

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;

6. **public** **class** FirstServlet **extends** HttpServlet {
8. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response){
9. **try**{
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. String n=request.getParameter("userName");
15. out.print("Welcome "+n);
17. HttpSession session=request.getSession();
18. session.setAttribute("uname",n);
20. out.print("<a href='servlet2'>visit</a>");
22. out.close();
24. }**catch**(Exception e){System.out.println(e);}
25. }
27. }

### SecondServlet.java

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
5. **public** **class** SecondServlet **extends** HttpServlet {
7. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)
8. **try**{
10. response.setContentType("text/html");
11. PrintWriter out = response.getWriter();
13. HttpSession session=request.getSession(**false**);
14. String n=(String)session.getAttribute("uname");
15. out.print("Hello "+n);
17. out.close();
19. }**catch**(Exception e){System.out.println(e);}
20. }

23. }

### web.xml

1. <web-app>
3. <servlet>
4. <servlet-name>s1</servlet-name>
5. <servlet-**class**>FirstServlet</servlet-**class**>
6. </servlet>
8. <servlet-mapping>
9. <servlet-name>s1</servlet-name>
10. <url-pattern>/servlet1</url-pattern>
11. </servlet-mapping>
13. <servlet>
14. <servlet-name>s2</servlet-name>
15. <servlet-**class**>SecondServlet</servlet-**class**>
16. </servlet>
18. <servlet-mapping>
19. <servlet-name>s2</servlet-name>
20. <url-pattern>/servlet2</url-pattern>
21. </servlet-mapping>
23. </web-app>

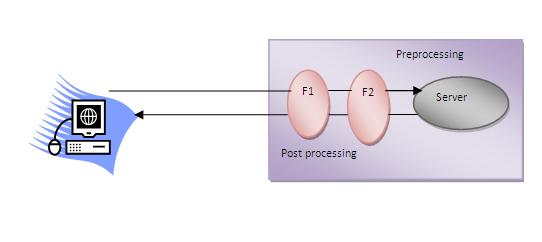
Servlet Filter

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

It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.

The **servlet filter is pluggable**, i.e. its entry 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 maintenance cost will be less.



### Filter API

Like servlet filter have its own API. The javax.servlet package contains the three interfaces of Filter API.

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 request,HttpServletResponse response, 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:

1. **public void doFilter(HttpServletRequest request, HttpServletResponse response):** it passes the control to the next filter or resource.

### How to define Filter

We can define filter same as servlet. Let's see the elements of filter and filter-mapping.

### index.html

1. <a href="servlet1">click here</a>

### MyFilter.java

1. **import** java.io.IOException;
2. **import** java.io.PrintWriter;
4. **import** javax.servlet.\*;
6. **public** **class** MyFilter **implements** Filter{
8. **public** **void** init(FilterConfig arg0) **throws** ServletException {}
10. **public** **void** doFilter(ServletRequest req, ServletResponse resp,
11. FilterChain chain) **throws** IOException, ServletException {
13. PrintWriter out=resp.getWriter();
14. out.print("filter is invoked before");
16. chain.doFilter(req, resp);//sends request to next resource
18. out.print("filter is invoked after");
19. }
20. **public** **void** destroy() {}
21. }

### HelloServlet.java

1. **import** java.io.IOException;
2. **import** java.io.PrintWriter;
4. **import** javax.servlet.ServletException;
5. **import** javax.servlet.http.\*;
7. **public** **class** HelloServlet **extends** HttpServlet {
8. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)
9. **throws** ServletException, IOException {
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. out.print("<br>welcome to servlet<br>");
16. }
18. }

**web.xml**

1. <web-app>
3. <servlet>
4. <servlet-name>s1</servlet-name>
5. <servlet-**class**>HelloServlet</servlet-**class**>
6. </servlet>
8. <servlet-mapping>
9. <servlet-name>s1</servlet-name>
10. <url-pattern>/servlet1</url-pattern>
11. </servlet-mapping>
13. <filter>
14. <filter-name>f1</filter-name>
15. <filter-**class**>MyFilter</filter-**class**>
16. </filter>
18. <filter-mapping>
19. <filter-name>f1</filter-name>
20. <url-pattern>/servlet1</url-pattern>
21. </filter-mapping>

24. </web-app>

Authentication Filter

**index.html**

1. <form action="servlet1">
2. Name:<input type="text" name="name"/><br/>
3. Password:<input type="password" name="password"/><br/>
5. <input type="submit" value="login">
7. </form>

**MyFilter.java**

1. **import** java.io.IOException;
2. **import** java.io.PrintWriter;
3. **import** javax.servlet.\*;
5. **public** **class** MyFilter **implements** Filter{
7. **public** **void** init(FilterConfig arg0) **throws** ServletException {}
9. **public** **void** doFilter(ServletRequest req, ServletResponse resp,
10. FilterChain chain) **throws** IOException, ServletException {
12. PrintWriter out=resp.getWriter();
14. String password=req.getParameter("password");
15. **if**(password.equals("admin")){
16. chain.doFilter(req, resp);//sends request to next resource
17. }
18. **else**{
19. out.print("username or password error!");
20. RequestDispatcher rd=req.getRequestDispatcher("index.html");
21. rd.include(req, resp);
22. }
24. }
25. **public** **void** destroy() {}
27. }

**AdminServlet.java**

1. **import** java.io.IOException;
2. **import** java.io.PrintWriter;
4. **import** javax.servlet.ServletException;
5. **import** javax.servlet.http.\*;
7. **public** **class** AdminServlet **extends** HttpServlet {
8. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)
9. **throws** ServletException, IOException {
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. out.print("welcome ADMIN");
15. out.close();
16. }
17. }

**web.xml**

1. <web-app>
2. <servlet>
3. <servlet-name>AdminServlet</servlet-name>
4. <servlet-**class**>AdminServlet</servlet-**class**>
5. </servlet>
7. <servlet-mapping>
8. <servlet-name>AdminServlet</servlet-name>
9. <url-pattern>/servlet1</url-pattern>
10. </servlet-mapping>
12. <filter>
13. <filter-name>f1</filter-name>
14. <filter-**class**>MyFilter</filter-**class**>
15. </filter>
16. <filter-mapping>
17. <filter-name>f1</filter-name>
18. <url-pattern>/servlet1</url-pattern>
19. </filter-mapping>
21. </web-app>