**Chapter 6**

**Servlets**

**Servlet Introduction**

Servlets are small programs that execute on the server side of a Web connection. Just as applets dynamically extend the functionality of a Web browser, servlets dynamically extend the functionality of a Web server.

To understand the advantages of servlets, you must have a basic understanding of how Web browsers and servers cooperate to provide content to a user. Consider a request for a static Web page. A user enters a Uniform Resource Locator (URL) to a browser. The browser generates an HTTP request to the appropriate Web server. The Web server maps this request to a specific file. That file is returned in an HTTP response to the browser. The HTTP header in the response indicates the type of the content. The Multipurpose Internet Mail Extensions (MIME) are used for this purpose. For example, ordinary ASCII text has a MIME type of text/plain. The Hypertext Markup Language (HTML) source code of a Web page has a MIME type of text/html.Now consider dynamic content. Assume that an online bookstore uses a database to store information about its business, including book prices, availability, orders, and so forth. It wants to make this information accessible to customers via Web pages. The contents of those Web pages must be dynamically generated, to reflect the latest information in the database.In the early days of the Web, a server could dynamically construct a page by creating a separate process to handle each client request. The process would open onnections to one or more databases in order to obtain the necessary information. It communicated with the Web server via an interface known as the Common Gateway Interface (CGI).

CGI allowed the separate process to read data from the HTTP request and write data to

the HTTP response. A variety of different languages were used to build CGI programs,

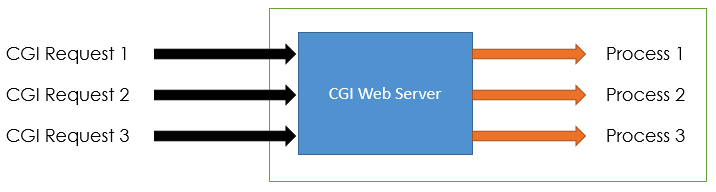
including C, C++, and Perl.

**Servlet Vs CGI**

## CGI (Commmon Gateway Interface) :

1. CGI (Common Gateway Interface) is used to provide dynamic content to the user.
2. CGI is used to execute a program that resides in the server to process data or access databases to produce the relevant dynamic content.
3. Programs that resides in server can be written in native operating system such as C++.

### Diagrammatic Representation :

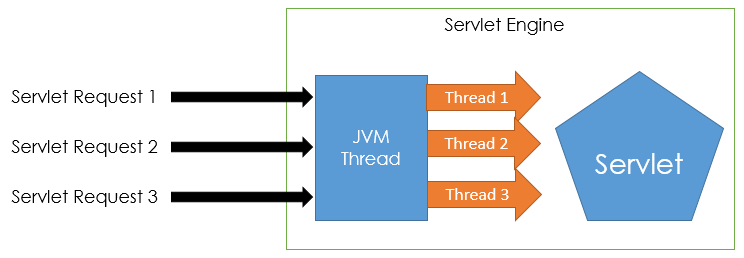


### Disadvantages of CGI :

1. For each request CGI Server receives, It creates new Operating System Process.
2. If the number of requests from the client increases then more time it will take to respond to the request.
3. As programs executed by CGI Script are written in the native languages such as C, C++, perl which are platform independent.

## Servlet :

CGI programs are used to execute programs written inside the native language. But in Servlet all the programs are compiled into the Java bytecode which is then run in the Java virtual machine.



In Servlet, All the requests coming from the Client are processed with the threads instead of the OS process.

## Servlet Vs CGI :

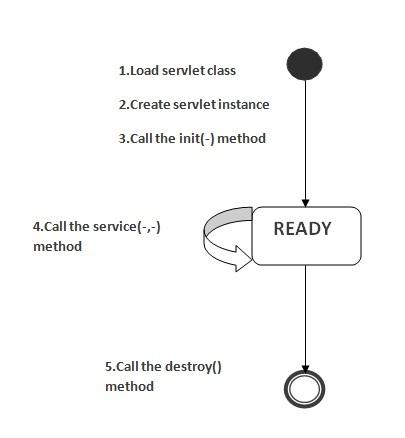
Let’s differentiate Servlet and CGI –

| **Servlet** | **CGI (Common Gateway Interface)** |
| --- | --- |
| Servlets are portable | CGI is not portable. |
| In Servlets each request is handled by lightweight Java Thread | IN CGI each request is handled by heavy weight OS process |
| In Servlets, Data sharing is possible | In CGI, data sharing is not available. |
| Servlets can link directly to the Web server | CGI cannot directly link to Web server. |
| Session tracking and caching of previous computations can be performed | Session tracking and caching of previous computations cannot be performed |
| Automatic parsing and decoding of HTML form data can be performed. | Automatic parsing and decoding of HTML form data cannot be performed. |
| Servlets can read and Set HTTP Headers | CGI cannot read and Set HTTP Headers |
| Servlets can handle cookies | CGI cannot handle cookies |
| Servlets can track sessions | CGI cannot track sessions |
| Servlets is inexpensive than CGI | CGI is more expensive than Servlets |

# 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.



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

### 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. Syntax of the init method is given below: |

1. public void init(ServletConfig config) throws ServletException

### 4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

1. public void service(ServletRequest request, ServletResponse response)
2. throws ServletException, IOException

### 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. The syntax of the destroy method of the Servlet interface is given below:

1. public void destroy()

**Using Tomcat for Servlet Developement**

To create servlets you need a servlet developement environment.The one recommended by sun is Tomcat.

Tomcat is an open source product maintained by Jakarta project of the Apache Software Foundation.

It Contains the class libraries, documentation & runtime support that you will need to create and test servlets.

When installed default location for Tomcat is

**C:\Program files\Apache Software Foundation\Tomcat7.0.**

Now go to

**C:\Program files\Apache Software Foundation\Tomcat7.0\common\lib**

And copy **servlet-api.jar** file and paste it in

**C:\program files\java\jre1.7\ext\lib**

Folder so that servlet package is available for JVM.

Then Update Environment variable path to the

**C:\program files\java\jre1.7\ext\lib\servlet-api.jar**

Now create following directory structure

**C:\Program files\Apache Software Foundation\Tomcat7.0\webapps**

Your own folder

WEB-INF .html files

classes web.xml

.class files

After compiling Servlet programs put .class files in the folder classes.classes folder of servlet applications is called as private folder which menas the client cannot directly access Servlet.

So the user should map some URL pattern for every Servlet class in web.xml

web.xml file is called as deployment discriptor.

It is as follows

**<web-app>**

**<servlet>**

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

**<servlet-class>Servletclassfile</servlet-class>**

**</servlet>**

**<servlet-mapping>**

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

**<url-pattern>/url</url-pattern>**

**<s/ervlet-mapping>**

Now when the client wants to access Servletclassfile then he has to type in web browser’s address bar

[**http://localhost:8080/your**](http://localhost:8080/your) **own folder/url**

Some times we create .html files also.For creating .html the syntax is

**<html>**

**<form name=”any-name” method=”method used in servlet class(get/post)” action=”**[**http://localhost:8080/your**](http://localhost:8080/your) **own folder/url”>**

**<input type=typename value=val>**

**</html>**

When we create .html then to access Servletclassfile type in web browser’s address bar as

[**http://localhost:8080/your**](http://localhost:8080/your) **own folder/file.html**

# Servlet API

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

The **javax.servlet** package contains many interfaces and classes that are used by the servlet . These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Let's see what are the interfaces of javax.servlet package.

### Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletConfig
3. ServletContext
4. ServletRequest
5. ServletResponse

### Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletException
5. UnavailableException

# Servlet Interface

All servlets must implement the **Servlet** interface. It declares the **init( )**, **service( )**, and

**destroy( )** methods that are called by the server during the life cycle of a servlet. A

method is also provided that allows a servlet to obtain any initialization parameters.

The methods defined by **Servlet** are shown as

**Methods**

1. **void destroy( )** -Called when the servlet is unloaded.
2. **ServletConfig getServletConfig( )** -Returns a **ServletConfig** object that contains any initialization parameters.
3. **String getServletInfo( )** Returns a string describing the servlet.
4. **void init(ServletConfig *sc*)throws ServletException-** Called when the servlet is initialized.Initialization parameters for the servlet can be obtained from *sc*. An **UnavailableException** should be thrown if the servlet cannot be initialized.
5. **void service(ServletRequest *req*,ServletResponse *res*)throws ServletException,**

**IOException -**Called to process a request from a client. The request from the client can be read from *req*.The response to the client can be written to *res*. An exception is generated if a servlet or IO problem occurs.

# The ServletConfig Interface

**The ServletConfig interface is implemented by the server. It allows a servlet to obtain**

**configuration data when it is loaded.**

**Methods**

1. **ServletContext getServletContext( )** - Returns the context for this servlet.
2. **String getInitParameter(String *param*) -Returns the value of the initialization**

**parameter named *param*.**

1. **Enumeration getInitParameterNames( )**- Returns an enumeration of all initialization parameter names.
2. **String getServletName( )** Returns the name of the invoking servlet.

# The ServletContext Interface

The **ServletContext** interface is implemented by the server. It enables servlets to obtain

information about their environment.

**Methods**

1. **Object getAttribute(String *attr*)** -Returns the value of the server attribute

named *attr*.

1. **String getMimeType(String *file*)** Returns the MIME type of *file*.
2. **String getRealPath(String *vpath*)** Returns the real path that corresponds

to the virtual path *vpath*.

1. **String getServerInfo( )** Returns information about the server.
2. **void log(String *s*)** Writes *s* to the servlet log.
3. **void log(String *s*, Throwable *e*)** Write *s* and the stack trace for *e* to the

servlet log.

1. **void setAttribute(String *attr*, Object *val*)** Sets the attribute specified by *attr* to the

value passed in *val*.

**The ServletRequest Interface**

The **ServletRequest** interface is implemented by the server. It enables a servlet to

obtain information about a client request.

**Methods**

1. **Object getAttribute(String *attr*)** -Returns the value of the server attribute

named *attr*.

1. **String getCharacterEncoding( )** Returns the character encoding of the request.
2. **int getContentLength( )** Returns the size of the request. The value –1 is returned if the size is unavailable.
3. **String getContentType( )** Returns the type of the request. A **null** value is returned if the type cannot be determined.
4. **String getParameter(String *pname*)** Returns the value of the parameter named *pname*.
5. **Enumeration getParameterNames( )** Returns an enumeration of the parameter names for this request.
6. **String[ ] getParameterValues(String *name* )** Returns an array containing values

associated with the parameter specified by *name*.

1. **String getProtocol( )** Returns a description of the protocol.
2. **String getRemoteAddr( ) Returns** the string equivalent of the client IP address.
3. **String getRemoteHost( )** Returns the string equivalent of the client host name.
4. **String getScheme( )** Returns the transmission scheme of the URL used for the request (for example, “http”, “ftp”).
5. **String getServerName( )** Returns the name of the server.
6. **int getServerPort( )** Returns the port number.

**The ServletResponse Interface**

The **ServletResponse** interface is implemented by the server. It enables a servlet to

formulate a response for a client.

**Methods**

1. **String getCharacterEncoding( )** Returns the character encoding for the

response.

1. **ServletOutputStream getOutputStream( ) throws IOException-** Returns a **ServletOutputStream** that can be used to write binary data to the response.

An **IllegalStateException** is thrown if **getWriter( )** has already been invoked for

this request.

1. **PrintWriter getWriter( ) throws IOException-** Returns a **PrintWriter** that can be used to write character data to the response.An **IllegalStateException** is thrown if **getOutputStream( )** has already been invoked for this request.
2. **void setContentLength(int *size*)** Sets the content length for the response to *size*.
3. **void setContentType(String *type*)** Sets the content type for the response to *type*.

**The GenericServlet Class**

The **GenericServlet** class provides implementations of the basic life cycle methods for

a servlet and is typically subclassed by servlet developers. **GenericServlet** implements

the **Servlet** and **ServletConfig** interfaces. In addition, a method to append a string to

the server log file is available. The signatures of this method are shown here:

void log(String *s*)

void log(String *s*, Throwable *e*)

Here, *s* is the string to be appended to the log, and *e* is an exception that occurred.

**The ServletInputStream Class**

The **ServletInputStream** class extends **InputStream**. It is implemented by the server

and provides an input stream that a servlet developer can use to read the data from a

client request. It defines the default constructor. In addition, a method is provided to

read bytes from the stream. Its signature is shown here:

int readLine(byte[ ] *buffer*, int *offset*, int *size*) throws IOException

Here, *buffer* is the array into which *size* bytes are placed starting at *offset*. The method

returns the actual number of bytes read or –1 if an end-of-stream condition is encountered.

**The ServletOutputStream Class**

The **ServletOutputStream** class extends **OutputStream**. It is implemented by the

server and provides an output stream that a servlet developer can use to write data

to a client response. A default constructor is defined. It also defines the **print( )** and

**println( )** methods, which output data to the stream.

**The Servlet Exception Classes**

**javax.servlet** defines two exceptions. The first is **ServletException**, which indicates that

a servlet problem has occurred. The second is **UnavailableException**, which extends

**ServletException**. It indicates that a servlet is unavailable.

**Working with parameters of Servlet**

Parameters may come into our application **from the client request, or may be configured through deployment descriptor (web.xml)**elements or their corresponding annotations. When you submit a form, form values are sent as request parameters to a web application.

Servlet init parameters and context init parameters are set through the deployment descriptor (web.xml). All parameters are **read-only**from the application code. We have methods in the Servlet API to retrieve various parameters.

Parameters are String objects.

The API methods to retrieve the**request parameters**are:

* ServletRequest.**getParameterValues**(String parmName)
  + returns a String array with all values present, or null if no value exists for the parameter name.
* ServletRequest.**getParameter**(String parmName)
  + returns the first value for the given parameter.
* ServletRequest.**getParameterNames**()
  + returns an Enumeration of String objects representing the names of all the parameters in the request. If there are no parameters Enumeration will be empty.

The API methods to retrieve the**ServletContext initialization parameters**from a ServletContext object are:

* ServletContext.**getInitParameterNames**()
  + will always return an enumeration of names.
* ServletContext.**getInitParameter**(String paramName)
  + will return a String or null.

The API methods to retrieve the **ServletConfig initialization parameters**from a ServletConfig object are:

* ServletConfig.**getInitParameterNames**()
  + returns an enumeration of all the parameter names available to the servlet.
* ServletConfig.**getInitParameter**(String paramName)
  + return a parameter value.

Both the methods are implemented in the GenericServlet abstract class.

**Example of reading request parameters of servlet**

**ChekPassword.java**

import java.io.\*;

import javax.servlet.\*;

public class CheckPassword extends GenericServlet

{

public void service(ServletRequest req,ServletResponse res)throws IOException,ServletException

{

res.setContentType("text/html");

PrintWriter w=res.getWriter();

String u=req.getParameter("username");

String p=req.getParameter("pass");

if(p.length()>6)

{

w.println("password should not be more than 6 char");

}

else

w.println("password ok");

w.close();

}

}

**web.xml**

<web-app>

<servlet>

<servlet-name>pass</servlet-name>

<servlet-class>CheckPassword</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>pass</servlet-name>

<url-pattern>/checkpass</url-pattern>

</servlet-mapping>

</web-app>

**Password.html**

<html>

<form name="pass" method="post" action="/servletpgm/checkpass">

<input type=textbox name="username" value=" ">

<input type=password name="pass" value=" ">

<input type=submit value="ok">

</html>

**Example of handling initialization parameters of servlet**

**InitServlet.java**

import java.io.\*;

import javax.servlet.\*;

public class InitServlet extends GenericServlet {

public void service(ServletRequest request,

ServletResponse response)

throws ServletException, IOException {

// Get ServletConfig object

ServletConfig sc = getServletConfig();

// Display two initialization arguments

response.setContentType("text/html");

PrintWriter pw = response.getWriter();

pw.println("<B>Country: " +

sc.getInitParameter("country"));

pw.println("<br>City: " +

sc.getInitParameter("city"));

pw.close();

}

}

**web.xml**

<web-app>

<servlet>

<servlet-name>demo</servlet-name>

<servlet-class>InitServlet</servlet-class>

<init-param>

<param-name>country</param-name>

<param-value>India</param-value>

</init-param>

<init-param>

<param-name>city</param-name>

<param-value>Solapur</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>demo</servlet-name>

<url-pattern>/initser</url-pattern>

</servlet-mapping>

</web-app>

**The javax.servlet.http Package**

The javax.servlet.http package contains several interfaces and classes that are commonly used by servlet developers. Its functionality makes it easy to build servlets that work with HTTP requests and responses.

The following table summarizes the interfaces that are provided in this package:

**Interface Description**

HttpServletRequest Enables servlets to read data from an HTTP request.

HttpServletResponse Enables servlets to write data to an HTTP response.

HttpSession Allows session data to be read and written.

HttpSessionBindingListener Informs an object that it is bound to or unbound from a

session.

The following table summarizes the classes that are provided in this package. The most

important of these is HttpServlet Servlet developers typically extend this class in order

to process HTTP requests.

**Class Description**

Cookie Allows state information to be stored on a client machine.

HttpServlet Provides methods to handle HTTP requests and responses.

HttpSessionBindingEvent Indicates when a listener is bound to or unbound from a

session value.

**The HttpServletRequest Interface**

The HttpServletRequest interface is implemented by the server. It enables a servlet to

obtain information about a client request.

Method Description

String getAuthType( ) Returns authentication scheme.

Cookie[ ] getCookies( ) Returns an array of the cookies in this request.

long getDateHeader(String field) Returns the value of the date header field named field

String getHeader(String field) Returns the value of the header field named field

Enumeration getHeaderNames( ) Returns an enumeration of the header names.

int getIntHeader(String field) Returns the int equivalent of the header field named field.

String getMethod( ) Returns the HTTP method for this request.

String getPathInfo( ) Returns any path information that is located after the servlet path and before a query string of the URI.

String getPathTranslated( ) Returns any path information that is located after the servlet path and before a query string of the URI, after translating it to a real path.

String getQueryString( ) Returns any query string in the URI.

String getRemoteUser( ) Returns the name of the user who issued this request.

String getRequestedSessionId( ) Returns the ID of the session.

String getRequestURI( ) Returns that part of the URI to the left of any query string.

String getServletPath( ) Returns the part of the URI that identifies the servlet.

HttpSession getSession(boolean new) If newis true, creates and returns a session for this

request. Otherwise, returns the existing session for this request.

boolean isRequestedSessionIdFromCookie() Returns true if a cookie contains the session ID. Otherwise, returns false.

boolean isRequestedSessionIdFromUrl( ) Returns true if the URL contains the session ID. Otherwise, returns false.

boolean isRequestedSessionIdValid( ) Returns true if the requested session ID is valid in the current session context

**The HttpServletResponse Interface**

The HttpServletResponse interface is implemented by the server. It enables a servlet to

formulate an HTTP response to a client. Several dozen constants are defined, which

correspond to the different status codes that can be assigned to an HTTP response. For

example, SC\_OK indicates that the HTTP request succeeded, and SC\_NOT\_FOUND

indicates that the requested resource is not available.

**Method Description**

void addCookie(Cookie cookie) Adds cookie to the HTTP response.

boolean containsHeader(String field) Returns true if the HTTP response header contains a field named field.

String encodeURL(String url) Determines whether the session ID must be encoded in the URL identified as url. If so, returns

the modified version of url. Otherwise, returns url.

All URLs generated by a servlet should be processed by this method.

String encodeRedirectUrl(String url) Determines whether the session ID must be encoded in the URL identified as url. If so, returns

the modified version of url. Otherwise, returns url.

All URLs passed to sendRedirect( ) should be processed by this method.

void sendError(int c) throws IOException Sends the error code c to the client.

void sendError(int c, String s) throws IOException Sends the error code c and message s to the client.

void sendRedirect(String url) throws IOException Redirects the client to url.

void setDateHeader(String field, long msec) Adds field to the header with date value equal to msec (milliseconds since midnight, January 1, 1970, GMT).

void setHeader(String field, String value) Adds field to the header with value equal to

value.

void setIntHeader(String field, int value) Adds field to the header with value equal to

value.

void setStatus(int code) Sets the status code for this response to code.

void setStatus(int code, String s) Sets the status code and message for this response to code and s.

**The HttpSession Interface**

The HttpSession interface is implemented by the server. It enables a servlet to read and

write the state information that is associated with an HTTP session.

All methods throw an IllegalStateException if the session has already been invalidated.

**Method Description**

long getCreationTime( ) Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when this session was created.

String getId( ) Returns the session ID.

long getLastAccessedTime( ) Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the client last made a request for this session.

HttpSessionContext getSessionContext( ) Returns the context associated with this

session.

Object getValue(String name) Returns the object bound to name. Returns null

if no such binding exists.

String[ ] getValueNames( ) Returns the names of all objects that are bound

in the session.

void invalidate( ) Invalidates this session and removes it from the context.

boolean isNew( ) Returns true if the server created the session

and it has not yet been accessed by the client.

void putValue(String name, Object obj) Binds obj to name in this session.

void removeValue(String name) Removes the object bound to name from the

session.

**The HttpSessionBindingListener Interface**

The HttpSessionBindingListener interface is implemented by objects that need to be

notified when they are bound to or unbound from an HTTP session. The methods that are

invoked when an object is bound or unbound are the following:

void valueBound(HttpSessionBindingEvent e)

void valueUnbound(HttpSessionBindingEvent e)

Here, e is the event object that describes the binding.

**Cookie class**

The Cookie class encapsulates a cookie. A cookie is stored on a client and contains state information. Cookies are valuable for tracking user activities. For example, assume that a user visits an online bookstore. A cookie can save the user's name, address, and other information. The user does not need to enter this data each time he or she visits the

bookstore.A servlet can write a cookie to a user's machine via the addCookie( )

method of the HttpServletResponse interface. The data for that cookie is then included in the header of the HTTP response that is sent to the browser.The names and values of cookies are stored on the user's machine. Some of the information that is saved includes the cookie's:

name

value

expiration date

domain and path

The expiration date determines when this cookie is deleted from the user's machine. If an

expiration date is not explicitly assigned to a cookie, it is deleted when the current browser session ends. Otherwise, the cookie is saved in a file on the user's machine.

The domain and path of the cookie determine when it is included in the header of an

HTTP request. If the user enters a URL whose domain and path match these values, the

cookie is then supplied to the Web server. Otherwise, it is not.

The only constructor for Cookie is

Cookie(String name, String value)

Here, the name and value of the cookie are supplied as arguments to the constructor.

**Method Description**

Object clone( ) Returns a copy of this object.

String getComment( ) Returns the comment.

String getDomain( ) Returns the domain.

int getMaxAge( ) Returns the age (in seconds).

String getName( ) eturns the name.

String getPath( ) Returns the path.

boolean getSecure( ) Returns true if the cookie is secure. Otherwise, returns

false.

String getValue( ) Returns the value.

int getVersion( ) Returns the version.

void setComment(String c) Sets the comment to c.

void setDomain(String d) Sets the domain to d.

void setMaxAge(int secs) Sets the maximum age of the cookie to secs. This is the

number of seconds after which the cookie is deleted.

void setPath(String p) Sets the path to p.

void setSecure(boolean secure) Sets the security flag to secure.

void setValue(String v) Sets the value to v.

void setVersion(int v) Sets the version to v

**Example on Cookies**

**AddCook.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class AddCook extends HttpServlet

{

public void service(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException

{

res.setContentType("text/html");

PrintWriter w=res.getWriter();

String data=req.getParameter("data");

Cookie c=new Cookie("SampleCookie",data);

res.addCookie(c);

w.println("<br>"+"Cookie has been set to"+data);

}

}

**GetCook.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class GetCook extends HttpServlet

{

public void service(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException

{

res.setContentType("text/html");

PrintWriter w=res.getWriter();

Cookie c[]=req.getCookies();

for(int i=0;i<c.length;i++)

{

String name=c[i].getName();

String val=c[i].getValue();

c[i].setComment("This is a demo of cookie");

int j=c[i].getMaxAge();

w.println("<br>"+"Name"+name);

w.println("<br>"+"Value"+val);

w.println("<br>"+"Age is"+j);

}

}

}

**web.xml**

<web-app>

<servlet>

<servlet-name>pass</servlet-name>

<servlet-class>CheckPassword</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>pass</servlet-name>

<url-pattern>/checkpass</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>data</servlet-name>

<servlet-class>ReadData</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>data</servlet-name>

<url-pattern>/data</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>addcook</servlet-name>

<servlet-class>AddCook</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>addcook</servlet-name>

<url-pattern>/addcookie</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>getcook</servlet-name>

<servlet-class>GetCook</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>getcook</servlet-name>

<url-pattern>/getcookie</url-pattern>

</servlet-mapping>

</web-app>

**addcook.html**

<html>

<form name="cookie" method="post" action="/servletpgm/addcookie">

Enter Value for MyCookie

<input type=textbox name="data">

</form>

</html>

**The HttpServlet Class**

**The HttpServlet class extends GenericServlet. It is commonly used when developing**

**servlets that receive and process HTTP requests.**

**Method Description**

void doDelete(HttpServletRequest req,

HttpServletResponse res) throws IOException,

ServletException Performs an HTTP DELETE.

void doGet(HttpServletRequest req,

HttpServletResponse res) throws IOException,

ServletException Performs an HTTP GET.

void doOptions(HttpServletRequest req,

HttpServletResponse res) throws IOException,

ServletException Performs an HTTP OPTIONS.

void doPost(HttpServletRequest req,

HttpServletResponse res) throws IOException,

ServletException Performs an HTTP POST.

void doPut(HttpServletRequest req,

HttpServletResponse res) throws IOException,

ServletException Performs an HTTP PUT.

void doTrace(HttpServletRequest req,

HttpServletResponse res) throws IOException,

ServletException Performs an HTTP TRACE.

long getLastModified(HttpServletRequest req) Returns the time (in milliseconds since

midnight, January 1, 1970, UTC) when the requested resource was last modified.

void service(HttpServletRequest req,

HttpServletResponse res) throws IOException,

ServletException Called by the server when an HTTP request arrives for this servlet. The arguments provide access to the HTTP request and response, respectively.

**Handling HTTP Requests and Responses**

The HttpServlet class provides specialized methods that handle the various types of

HTTP requests. A servlet developer typically overrides one of these methods. These

methods are doDelete( ), doGet( ), doOptions( ),doPost( ),doPut( ), and doTrace( ).

The GET and POST methods are commonly used when handling form input.

**Handling HTTP GET and POST Requests**

There are many differences between the Get and Post request. Let's see these differences:

|  |  |
| --- | --- |
| **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. |

GET and POST

Two common methods for the request-response between a server and client are:

* **GET**- It requests the data from a specified resource
* **POST**- It submits the processed data to a specified resource

The query string (name/value pairs) is sent inside the URL of a GET request:

**GET /RegisterDao.jsp?name1=value1&name2=value2**

**Example on GET request**

<html>

<body>

<center>

<form name="Form1"

action="http://localhost:8080/servlet/ColorGetServlet">

<B>Color:</B>

<select name="color" size="1">

<option value="Red">Red</option>

<option value="Green">Green</option>

<option value="Blue">Blue</option>

</select>

<br><br>

<input type=submit value="Submit">

</form>

</body>

</html>

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class ColorGetServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException

{

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

response.setContentType("text/html");

PrintWriter pw = response.getWriter();

pw.println("<B>The selected color is: ");

pw.println(color);

pw.close();

}

}

**Example on POST request**

<html>

<body>

<center>

<form name="Form1"

method="post"

action="http://localhost:8080/servlet/ColorPostServlet">

<B>Color:</B>

<select name="color" size="1">

<option value="Red">Red</option>

<option value="Green">Green</option>

<option value="Blue">Blue</option>

</select>

<br><br>

<input type=submit value="Submit">

</form>

</body>

</html>

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class ColorPostServlet extends HttpServlet {

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

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

response.setContentType("text/html");

PrintWriter pw = response.getWriter();

pw.println("<B>The selected color is: ");

pw.println(color);

pw.close();

}

}

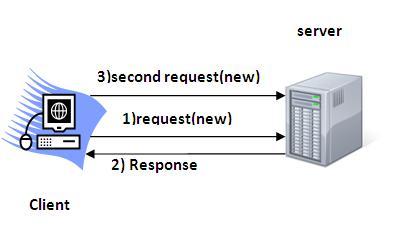
# Session Tracking in Servlets

**Session** simply means a collection of requests over 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.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



### Why use Session Tracking?

**To recognize the user** It is used to recognize the particular user.

### Session Tracking Techniques

There are four techniques used in Session tracking:

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

* **Using Cookies**

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, we recognize the user as the old user.

**Types of Cookie**

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

### Non-persistent 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 session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

### 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.

* **Hidden Form Field**

In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.

In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Let's see the code to store value in hidden field.

<input type="hidden" name="uname" value="Vimal Jaiswal">

Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.

### Real application of hidden form field

It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

### Advantage of Hidden Form Field

1. It will always work whether cookie is disabled or not.

### Disadvantage of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each pages.
3. Only textual information can be used.

* **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&??

A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.

**Advantage of URL Rewriting**

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

### Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

* **HttpSession**

HTTP is a stateless protocol, which means that each request is independent of the

previous one. However, in some applications, it is necessary to save state information so that information can be collected from several interactions between a browser and a

server. Sessions provide such a mechanism.

A session can be created via the **getSession( )**method of **HttpServletRequest**

. An **HttpSession** object is returned. This object can store a set of bindings that associate names with objects. The putValue( ), getValue( ), getValueNames( ), and

removeValue( )methods of HttpSessionmanage these bindings. It is important to note

that session state is shared among all the servlets that are associated with a particular

client.

The following servlet illustrates how to use session state. The getSession( )method gets

the current session. A new session is created if one does not already exist. The getValue( ) method is called to obtain the object that is bound to the name "date". That

object is a Date object that encapsulates the date and time when this page was last accessed. (Of course, there is no such binding when the page is first accessed.) A

Date object encapsulating the current date and time is then created. The putValue( )

method is called to bind the name "date" to this object.

import java.io.\*;

import java.util.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class DateServlet extends HttpServlet {

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

// Get the HttpSession object

HttpSession hs = request.getSession(true);

// Get writer

response.setContentType("text/html");

PrintWriter pw = response.getWriter();

pw.print("<B>");

// Display date/time of last access

Date date = (Date)hs.getValue("date");

if(date != null) {

pw.print("Last access: " + date + "<br>");

}

// Display current date/time

date = new Date();

hs.putValue("date", date);

pw.println("Current date: " + date);

}

}

When you first request this servlet, the browser displays one line with the current date and time information. On subsequent invocations, two lines are displayed. The first line shows the date and time when the servlet was last accessed. The second line shows the current date and time.

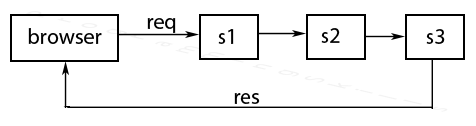
**Introduction to servlet chaining & filters**

If a client request is processed by group of servlets, then that servlets are known as servlet chaining or if the group of servlets process a single client request then those servlets are known as servlet chaining.

In order to process a client request by many number of servlets then we have two models, they are forward model and include model.

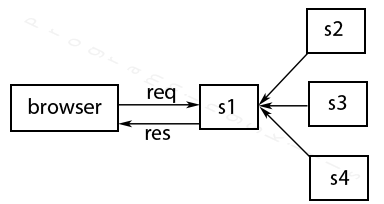
#### Forward model:

In this model when we forward a request to a group of servlets, finally we get the result of destination servlet as a response but not the result of intermediate servlets.



#### Include model:

If a single client request is passed to a servlet and that servlet makes use of other group of servlets to process a request by including the group of servlets into a single servlet.



In the above diagram client request goes to servlet s1 and s1 internally includes s2, s3 and s4 servlets and finally result of all these servlets given to the client by a source servlet s1.

**Note:** One servlet can include any number of servlets where as one servlet can forward to only one servlet at a time.

#### Steps for DEVELOPING Servlet Chaining:

1. Obtain an object of ServletContext by using in any of the following way:

ServletContext ctx1=getServletContext (); [GenericServlet method]

ServletContext ctx2=config.getServletContext (); [ServletConfig method]

ServletContext ctx3=req.getServletContext (); [HttpServletRequest method]

1. Obtain an object of RequestDispatcher. RequestDispatcher is an interface which is present in javax.servlet.\* package and it is used for forwarding the request and response objects of source servlet to destination servlet or for including the destination servlet into source servlet. To obtain an object of RequestDispatcher, the ServletContext contains the following method:



RequestDispatcher rd=ctx.getRequestDispatcher ("./s2");

1. Use forward or include model to send the request and response objects. RequestDispatcher contains the following methods for forwarding or including the request and response objects.

**For example:**

rd.forward (req, res) **throws** ServletException, IOException

rd.include (req, res) **throws** ServletException, IOException

#### web.xml:

<web-app>

<servlet>

<servlet-name>abc</servlet-name>

<servlet-class>Serv1</servlet-class>

</servlet>

<servlet>

<servlet-name>pqr</servlet-name>

<servlet-class>Serv2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>abc</servlet-name>

<url-pattern>/s1</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>pqr</servlet-name>

<url-pattern>/s2</url-pattern>

</servlet-mapping>

</web-app>

#### Serv1.java:

**import** **javax.servlet.\***;

**import** **javax.servlet.http.\***;

**import** **java.io.\***;

**public** **class** **Serv1** **extends** HttpServlet {

**public** **void** **doGet**(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

pw.println("<h2>I AM FROM Serv1 BEGINNING</h2>");

ServletContext ctx = getServletContext();

RequestDispatcher rd = ctx.getRequestDispatcher("/s2");

rd.include(req, res);

pw.println("<h2>I AM FROM Serv1 ENDING</h2>");

}

**public** **void** **doPost**(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

doGet(req, res);

}

}

#### Serv2.java:

**import** **javax.servlet.\***;

**import** **javax.servlet.http.\***;

**import** **java.io.\***;

**public** **class** **Serv2** **extends** HttpServlet {

**public** **void** **doGet**(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

pw.println("<h2>I AM FROM Serv2</h2>");

}

**public** **void** **doPost**(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

doGet(req, res);

}

}

# 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.

#### Note: Unlike Servlet, One filter doesn't have dependency on another filter.

### 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

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

### 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

# [Applet to Servlet Communication](http://ecomputernotes.com/servlet/intro/applet-to-servlet-communication)

HTML exhibits high performance by taking less time to load in the browser. However, when we use HTML page for important user details, by default, all the parameters that are passed appended in the URL. This compromises with the security. On the other hand, applet takes more time to load but there is no problem with Java security. This is an advantage of this technique.

Applet is a compiled Java class that can be sent over the network. Applets are an alternative to HTML form page for developing websites. HTML form gives good performance, takes less time to load but has poor security. Whereas, Applets give poor performance, take time to load but have good security.

There are two types of Applets:

1. Untrusted Applets: It cannot interact with files and file system, so writing malicious codes is not possible. Applets are untrusted.

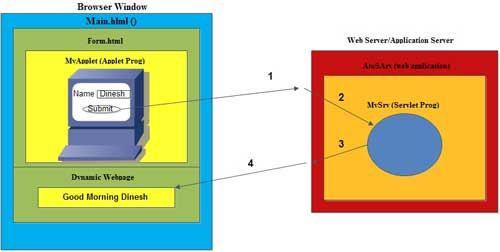
2. Trusted Applets: It can interact with files and file system so can write malicious codes.

**Difference between *applet*and**[***servlet***](http://ecomputernotes.com/servlet/intro/servlet)

|  |  |
| --- | --- |
| **APPLET** | **SERVLET** |
| Used to develop client side web-resource program to generate static web page. | Used to develop server side web-resource program to generate dynamic web-page. |
| Needs browser windw or appletviewer for execution. | Needs servlet container for execution. |
| Applet program comes to browser window from server for execution. | Servlet program reside and execute in web resource. |
| The life cycle methods are init(), start(), stop() and destroy(). | The life cycle methods are init(-), Service(-,-)  and destroy(). |

Similar to HTML-to-servlet communication we need to perform applet-to servlet communication. In HTML-to-servlet communication the browser window automatically forms request URL and automatically forms query string having form data, when submit button is clicked. In applet-to-servlet communication all these activities should be taken care of by the programmers manually or explicitly as event handling operation.

**Example on Application of Applet to Servlet Communication**

****

Frame is a logical partition of the web page. Frame with name is called Named ,

Frame.

Step 1: Prepare the deployment directory structure of web application.

Request url **http://localhost:8081/** AtoSApp/Main.html

Step 2: Develop the source code of above servlet program or web Application.

***Source Code******MyApplet.[java](http://ecomputernotes.com/java/what-is-java/what-is-java-explain-basic-features-of-java-language)****import javax.servlet.\*;   
import javax.servlet.http.\*;   
import java.io.\*;   
import Java.util.\*;   
public class wishsrv extends HttpServlet   
{   
public void service(HttpServletRequest req,HttpServletResponse res) throws ServletException , IOException   
{   
//general settings   
PrintWriter.pw=res.getWriter{);   
setContentType("text/html") ;   
//read form data   
String name=req.getParameter("uname") i   
//generate wish message   
Calendar cl=Calendar.getlnstance();   
int h=cl.get(Calendar.HOUR\_OF\_DAY);   
if (h<=12)   
pw. println ("Good Morning :"+name) i   
elseif(h<=16}   
pw.println("Good Afternoon: "+name);   
elseif(h<=20}   
pw.println("Good Evening :"+name);   
else   
pw.println("Good Night :"+name);   
//close stream obj   
pw.close() i   
}//doGet   
} / /class   
<> javac MyServlet.java****web.xml****Configure MyServletprogram with /testurl url pattern and also configure Main.html as welcome file.   
<web-app>   
        <servlet>   
            <servlet-name>abc</servlet-name>   
            <servlet-class>MyServlet</servlet-class>   
        </servlet>   
          <servlet-mapping>   
             <servlet-name>abc</servlet-name>   
             <url-pattern>/testurl</url-pattren>   
          </servlet-mapping>   
          <welcome-file-list>   
              <welcome-file>Main.html</welcome-file>   
          </welcome-file-list>   
</web-app>****Main.html****<frameset rows = "40% , \*">   
<frame name = "f1" SYC = "Form.html">   
<frame =  "f2" />   
</frameset>****MyApplet.java****// MyApplet. Java   
import java.awt.\*;   
import java.applet.\*;   
import java.awt.event.\*;   
import java.net.\*;   
public class MyApplet extends Applet implements ActionListener   
{   
Button b;   
TextField tfl;   
Label l1;   
public void init ()   
{   
l1 = new Label ("User name :");   
add(l1) ;   
tfl = new TextField(10);   
add(tf1) ;   
b=new Button("Send");   
b.addActionListener(this) ;   
add(b) ;   
}   
public void actionPerformed(ActionEvent ae)   
{   
try{   
//read text value   
String name=tfl.getText().replace();   
//frame query String   
String qrystr=(“?uname=”+name) ;   
//frame request url having query String   
String url=(“[http://localhost:](http://localhost:8081/AtoSApp/testurl)*[*8081*](http://localhost:8081/AtoSApp/testurl)*[/AtoSApp/testurl](http://localhost:8081/AtoSApp/testurl)”+qrystr);   
//create URL class object   
URL requrl = new URL (ur1);   
//getAppletContext obj   
AppletContext apc=getAppletContext();   
}catch(Exception ee)   
{}   
{}*

***Form.html***

<applet code= "MyApplet.class" width= "500" height= "500">

</applet>

**Introduction to JSP**

JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.

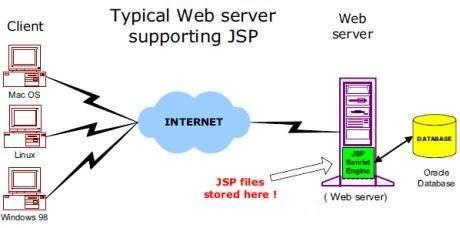
JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

# JSP – Architecture

The web server needs a JSP engine, i.e, a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development.

A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web application.

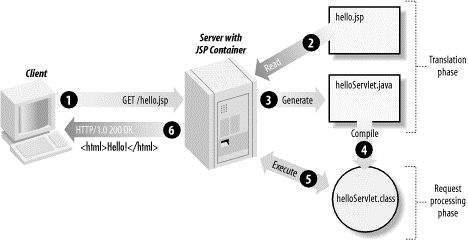


JSP Processing

The following steps explain how the web server creates the Webpage using JSP −

* As with a normal page, your browser sends an HTTP request to the web server.
* The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of **.html**.
* The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.
* The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
* A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is further passed on to the web server by the servlet engine inside an HTTP response.
* The web server forwards the HTTP response to your browser in terms of static HTML content.
* Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be seen in the following diagram −



Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with the other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

# JSP - Lifecycle

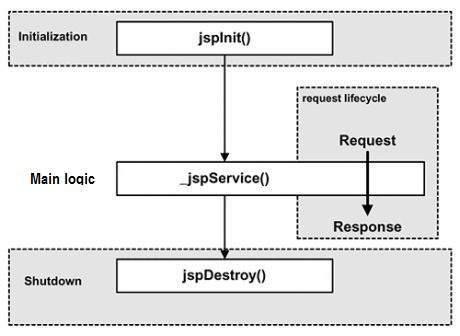
A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a H with an additional step which is required to compile a JSP into servlet.

Paths Followed By JSP

The following are the paths followed by a JSP −

* Compilation
* Initialization
* Execution
* Cleanup

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below −



JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps −

* Parsing the JSP.
* Turning the JSP into a servlet.
* Compiling the servlet.

JSP Initialization

When a container loads a JSP it invokes the **jspInit()** method before servicing any requests. If you need to perform JSP-specific initialization, override the **jspInit()** method −

public void jspInit(){

// Initialization code...

}

Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP Execution

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **\_jspService()**method in the JSP.

The \_jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows −

void \_jspService(HttpServletRequest request, HttpServletResponse response) {

// Service handling code...

}

The **\_jspService()** method of a JSP is invoked on request basis. This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, **GET, POST, DELETE**, etc.

JSP Cleanup

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

The jspDestroy() method has the following form −

public void jspDestroy() {

// Your cleanup code goes here.

}

### Advantage of JSP over Servlet

There are many advantages of JSP over servlet. They are as follows:

#### 1) Extension to Servlet

JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

#### 2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

#### 3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

#### 4) Less code than Servlet

In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

# JSP Scriptlet tag (Scripting elements)

In JSP, java code can be written inside the jsp page using the scriptlet tag.

**JSP Scripting elements**

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

* scriptlet tag
* expression tag
* declaration tag

### JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

1. <%  java source code %>

### Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

1. **<html>**
2. **<body>**
3. **<**% out.print("welcome to jsp"); %**>**
4. **</body>**
5. **</html>**

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

*File: index.html*

1. **<html>**
2. **<body>**
3. **<form** action="welcome.jsp"**>**
4. **<input** type="text" name="uname"**>**
5. **<input** type="submit" value="go"**><br/>**
6. **</form>**
7. **</body>**
8. **</html>**

*File: welcome.jsp*

1. <html>
2. <body>
3. <%
4. String name=request.getParameter("uname");
5. out.print("welcome "+name);
6. %>
7. </form>
8. </body>
9. </html>

# JSP expression tag

The code placed within **JSP expression tag** is *written to the output stream of the response*. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

### Syntax of JSP expression tag

1. **<**%=  statement %**>**

### Example of JSP expression tag

In this example of jsp expression tag, we are simply displaying a welcome message.

1. **<html>**
2. **<body>**
3. **<**%= "welcome to jsp" %**>**
4. **</body>**
5. **</html>**

### Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

*File: index.jsp*

1. **<html>**
2. **<body>**
3. **<form** action="welcome.jsp"**>**
4. **<input** type="text" name="uname"**><br/>**
5. **<input** type="submit" value="go"**>**
6. **</form>**
7. **</body>**
8. **</html>**

*File: welcome.jsp*

1. **<html>**
2. **<body>**
3. **<**%= "Welcome "+request.getParameter("uname") %**>**
4. **</body>**
5. **</html>**

# JSP Declaration Tag

The **JSP declaration tag** is used *to declare fields and methods*.

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.

So it doesn't get memory at each request.

#### Syntax of JSP declaration tag

The syntax of the declaration tag is as follows:

1. **<**%!  field or method declaration %**>**

### Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

### index.jsp

1. **<html>**
2. **<body>**
3. **<**%! int data=50; %**>**
4. **<**%= "Value of the variable is:"+data %**>**
5. **</body>**
6. **</html>**

### Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

### index.jsp

1. **<html>**
2. **<body>**
3. **<**%!
4. int cube(int n){
5. return n\*n\*n\*;
6. }
7. %**>**
8. **<**%= "Cube of 3 is:"+cube(3) %**>**
9. **</body>**
10. **</html>**

# JSP Implicit Objects

There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.

The available implicit objects are out, request, config, session, application etc.

A list of the 9 implicit objects is given below:

|  |  |
| --- | --- |
| **Object** | **Type** |
| out | JspWriter |
| request | HttpServletRequest |
| response | HttpServletResponse |
| config | ServletConfig |
| application | ServletContext |
| session | HttpSession |
| pageContext | PageContext |
| page | Object |
| exception | Throwable |

### 1) JSP out implicit object

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

1. PrintWriter out=response.getWriter();

But in JSP, you don't need to write this code.

### Example of out implicit object

In this example we are simply displaying date and time.

### index.jsp

1. <html>
2. <body>
3. <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
4. </body>
5. </html>

|  |  |  |  |
| --- | --- | --- | --- |
| JSP directives  1. [JSP directives](https://www.javatpoint.com/jsp-page-directive)    1. [page directive](https://www.javatpoint.com/jsp-page-directive#page)    2. [Attributes of page directive](https://www.javatpoint.com/jsp-page-directive#pageattr)   The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.  There are three types of directives:   * page directive * include directive * taglib directive  Syntax of JSP Directive  1. <%@ directive attribute="value" %>  JSP page directive The page directive defines attributes that apply to an entire JSP page. Syntax of JSP page directive  1. <%@ page attribute="value" %>  Attributes of JSP page directive  * import * contentType * extends * info * buffer * language * isELIgnored * isThreadSafe * autoFlush * session * pageEncoding * errorPage * isErrorPage  1)import  |  | | --- | | The import attribute is used to import class,interface or all the members of a package.It is similar to import keyword in java class or interface. |  Example of import attribute  1. <html> 2. <body> 4. <%@ page **import**="java.util.Date" %> 5. Today is: <%= **new** Date() %> 7. </body> 8. </html>  2)contentType The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1". Example of contentType attribute  1. <html> 2. <body> 4. <%@ page contentType=application/msword %> 5. Today is: <%= **new** java.util.Date() %> 7. </body> 8. </html>  3)extends The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used. 4)info This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface. Example of info attribute  1. <html> 2. <body> 4. <%@ page info="composed by Sonoo Jaiswal" %> 5. Today is: <%= **new** java.util.Date() %> 7. </body> 8. </html>   The web container will create a method getServletInfo() in the resulting servlet.For example:   1. **public** String getServletInfo() { 2. **return** "composed by Sonoo Jaiswal"; 3. }  5)buffer The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb. Example of buffer attribute  1. <html> 2. <body> 4. <%@ page buffer="16kb" %> 5. Today is: <%= **new** java.util.Date() %> 7. </body> 8. </html>  6)language The language attribute specifies the scripting language used in the JSP page. The default value is "java". 7)isELIgnored  |  | | --- | | We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default. We see Expression Language later. |  1. <%@ page isELIgnored="true" %>//Now EL will be ignored  8)isThreadSafe  |  | | --- | | Servlet and JSP both are multithreaded.If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive.The value of isThreadSafe value is true.If you make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.If you make the value of isThreadSafe attribute like: |   <%@ page isThreadSafe="false" %>  The web container in such a case, will generate the servlet as:   1. **public** **class** SimplePage\_jsp **extends** HttpJspBase 2. **implements** SingleThreadModel{ 3. ....... 4. }  9)errorPage The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page. Example of errorPage attribute  1. //index.jsp 2. <html> 3. <body> 5. <%@ page errorPage="myerrorpage.jsp" %> 7. <%= 100/0 %> 8. NAHI SAMAJLA NA BHAI 10. </body> 11. </html>  10)isErrorPage The isErrorPage attribute is used to declare that the current page is the error page. Note: The exception object can only be used in the error page.Example of isErrorPage attribute  1. //myerrorpage.jsp 2. <html> 3. <body> 5. <%@ page isErrorPage="true" %> 7. Sorry an exception occured!<br/> 8. The exception is: <%= exception %> 10. </body> 11. </html> |

# Jsp Include Directive

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).

### Advantage of Include directive

Code Reusability

### Syntax of include directive

1. <%@ include file="resourceName" %>

### Example of include directive

In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

1. <html>
2. <body>
4. <%@ include file="header.html" %>
6. Today is: <%= java.util.Calendar.getInstance().getTime() %>
8. </body>
9. </html>

# JSP Taglib directive

The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags. In the custom tag section we will use this tag so it will be better to learn it in custom tag.

#### Syntax JSP Taglib directive

1. <%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>

### Example of JSP Taglib directive

In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

1. <html>
2. <body>
4. <%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>
6. <mytag:currentDate/>
8. </body>
9. </html>