ATHARAV INSTITUTE OF COMPUTER TRAINING AND EDUCATION

SantTukaramNagar,NearDr.D.Y.Patil college campus Pimpri
Mobile :: 7028851096 /7721818535
Phone :: 020-27459656

Copy
servlet-api .jar(for the execution of servlet program)
&
jsp-api.jar (for the execution of servlet program)
From
C:\Program Files\Apache Software Foundation\Tomcat 5.5\common\lib
and paste to
C:\Program Files\Java\jdk1.6.0_01\jre\lib\ext

Servlet

What is servlet

A servlet is a Server-side java program that is platform independent. Servlet runs within Web-Server.

Even the servlet are written in java, their client is not necessarily written in java. Servlet client can be written in any desired language.

Servlet are not tied to specific client-server protocol but they are most commonly used with HTTP. The word Servlet always used to represent HTTP Servlet.

What Can Servlet Do

- Servlets can be used as a plug-in, that features a search engine or semi-customized applications such as web-based banking or inventory systems.
- A servlet can handle multiple requests concurrently and these requests can be synchronized to support system allowing collaboration between people.
- Servlet can forward requests to another Servlets. This allows them to be used to balance load among several servers that mirror the same content.
- Servlet can maintain and track the session.(either by URL rewriting or cookies)
- Servlets can pass data between themselves.

Container :

Servlet do not have a main() method. They are under the control of another java application called container.

Tomcat is example of container.

When web server gets requests for a servlet, the server hands the requests not to the servlet itself, but to the container that gives the HTTP requests and response, and container calls the serve's methods like doget() or dopost()

What does the container give you?

- **Communication Support**
  The container provides an easy way for your servlet to talk to your web server.
  You do not have to build a ServerSocket, listens protocol between the web server and itself.

- **Lifecycle Management**
  Container controls the lifecycle and death of your servlets.

- **Multithreading Support**
  Container automatically creates a new java thread for every servlet request it receives.

- **Declarative Security**
  With a Container, you get to use an XML deployment descriptor to configure (and modify) security without having to hard-code it into your Servlet or any other class code.

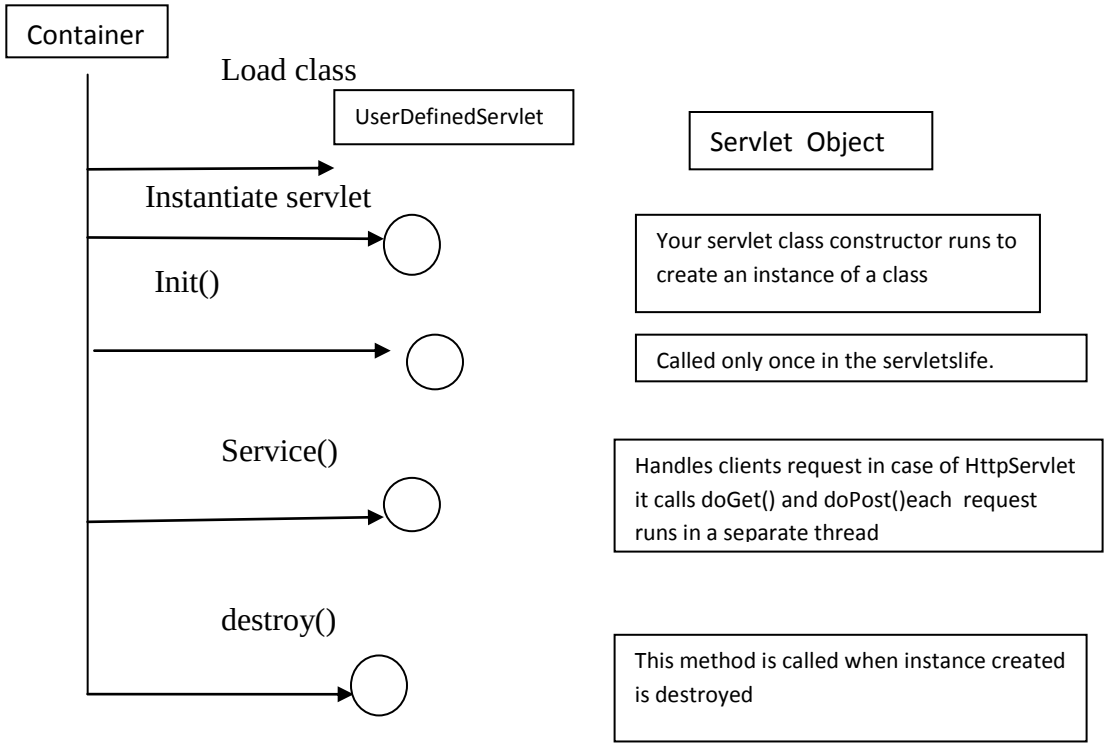
New features added to Servlet 2.5
Following are the changes introduced in Servlet 2.5:

- A new dependency on J2SE 5.0
- Support for annotations
- Loading the class
- Several web.xml conveniences
- A handful of removed restrictions
- Some edge case clarifications

What are the advantages of Servlet over CGI?
Servlets have several advantages over CGI:

- A Servlet does not run in a separate process. This removes the overhead of creating a new process for each request.
- A Servlet stays in memory between requests. A CGI program (and probably also an extensive runtime system or interpreter) needs to be loaded and started for each CGI request.
- There is only a single instance which answers all requests concurrently. This saves memory and allows a Servlet to easily manage persistent data.
- Several web.xml conveniences
- A handful of removed restrictions
- Some edge case clarifications

Life cycle of servlet:



What are the phases of the servlet life cycle?

The life cycle of a servlet consists of the following phases:

- **Servlet class loading :** For each servlet defined in the deployment descriptor of the Web application, the servlet container locates and loads a class of the type of the servlet. This can happen when the servlet engine itself is started, or later when a client request is actually delegated to the servlet.
- **Servlet instantiation :** After loading, it instantiates one or more object instances of the servlet class to service the client requests.
- **Initialization (call the init method) :** After instantiation, the container initializes a servlet before it is ready to handle client requests. The container initializes the servlet by invoking its init() method, passing an object implementing the ServletConfig interface. In the init() method, the servlet can read configuration parameters from the deployment descriptor or perform any other one-time activities, so the init() method is invoked once and only once by the servlet container.

- **Request handling (call the service method) :** After the servlet is initialized, the container may keep it ready for handling client requests. When client requests arrive, they are delegated to the servlet through the service() method, passing the request and response objects as parameters. In the case of HTTP requests, the request and response objects are implementations of HttpServletRequest and HttpServletResponse respectively. In the HttpServlet class, the service() method invokes a different handler method for each type of HTTP request, doGet() method for GET requests, doPost() method for POST requests, and so on.
- **Removal from service (call the destroy method) :** A servlet container may decide to remove a servlet from service for various reasons, such as to conserve memory resources. To do this, the servlet container calls the destroy() method on the servlet. Once the destroy() method has been called, the servlet may not service any more client requests. Now the servlet instance is eligible for garbage collection

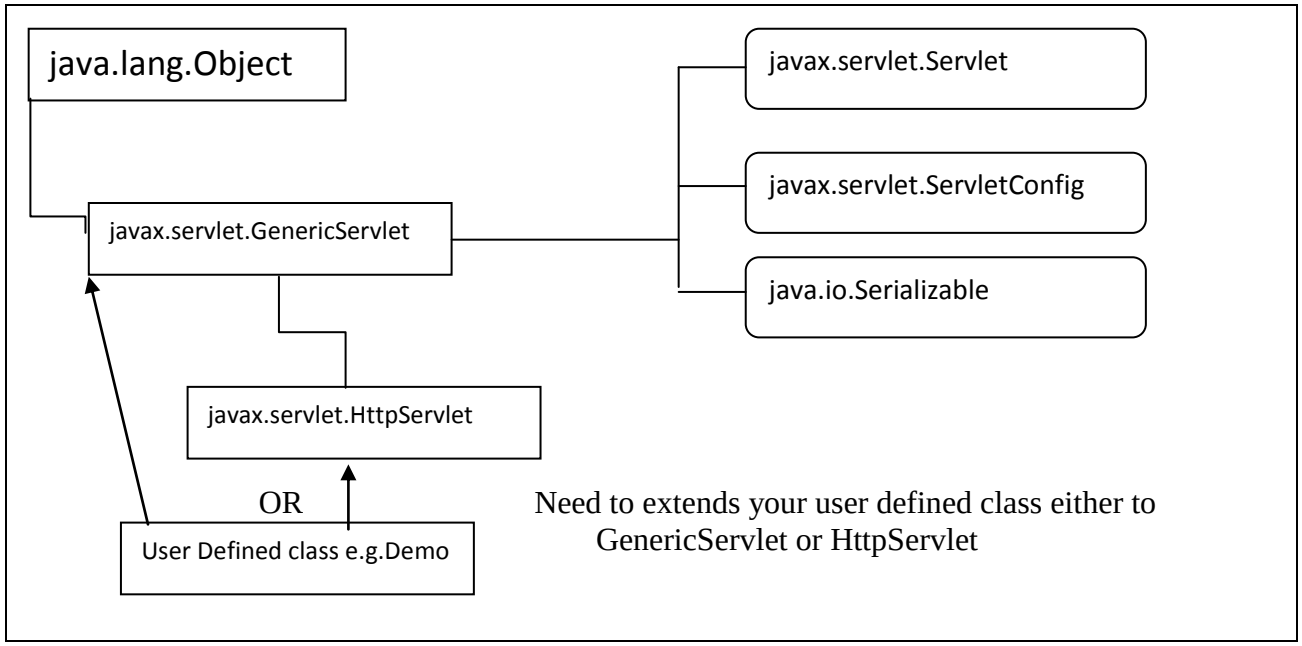
  The life cycle of a servlet is controlled by the container in which the servlet has been deployed.

Store your source file in WEB-INF - > classes folder

Servlet API

Contains classes and interfaces in the packages.

- javax.servlet
- javax.servlet.http



Servlet Interfaces and Classes

	Important classes and interfaces	
Packages	javax.servlet	javax.servlet.http
Interfaces	Servlet ServletRequest ServletResponse ServletConfig ServletContext	HttpServletRequest HttpServletResponse HttpSession HttpSessionContext
Classes	GenericServlet	Cookie HttpServlet

GenericServlet	**HttpServlet**
The GenericServlet is an abstract class that is extended by HttpServlet to provide HTTP protocol-specific methods.	An abstract class that simplifies writing HTTP servlets. It extends the GenericServlet base class and provides an framework for handling the HTTP protocol.
The GenericServlet does not include protocol-specific methods for handling request parameters, cookies, sessions and setting response headers.	The HttpServlet subclass passes generic service method requests to the relevant doGet() or doPost() method.
GenericServlet is not specific to any protocol.	HttpServlet only supports HTTP and HTTPS protocol.

	doGet()	**doPost()**
1	In doGet() the parameters are appended to the URL and sent along with header information.	In doPost(), on the other hand will (typically) send the information through a socket back to the webserver and it won't show up in the URL bar.
2	The amount of information you can send back using a GET is restricted as URLs can only be 1024 characters.	You can send much more information to the server this way - and it's not restricted to textual data either. It is possible to send files and even binary data such as serialized Java objects!
3	doGet() is a request for information; it does not (or should not) change anything on the server. (doGet() should be idempotent)	doPost() provides information (such as placing an order for merchandise) that the server is expected to remember
4	Parameters are not encrypted	Parameters are encrypted
5	doGet() is faster if we set the response content length since the same connection is used. Thus increasing the performance	doPost() is generally used to update or post some information to the server.doPost is slower compared to doGet since doPost does not write the content length
6	doGet() should be idempotent. i.e. doget should be able to be repeated safely many times	This method does not need to be idempotent. Operations requested through POST can have side effects for which the user can be held accountable.
7	doGet() should be safe without any side effects for which user is held responsible	This method does not need to be either safe
8	It allows bookmarks.	It disallows bookmarks.

javax.servlet.Servlet interface

- The Servlet Life cycle

1) public void init(ServletConfigconfig) throws ServletException

It is called only once when the Servlet is loaded in a memory for the first time.
The init () method is called immediately after the server construct the Servlet instance.
Depending upon the server and the configuration, this can be any of the following cases:

- When the Server starts
- When the Servlet is first requested, just before service() method is invoked.
- At the request of the server administrator.

When a Servlet is set up for first time, the configuration data such as information about the parameters and references to the ServletContext is stored in ServletConfig object.

ServletException

If an exception has occurred that interferes with Servlet's normal operation.

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

Called by the container to allow the Servlet to repond to a request. This method is called after Servlet's init() method completed successfully.

For, every request to the Servlet, its service() method is called. Two objects ServletRequest and ServletResponse are passed as parameters to the service() method.

- ServletRequest interface

Defines an object to provide client request information to a Servlet. The Servlet container creates a ServletRequest object and passes it as argument to the Servlet's service() method.

Methods:

i) String getParameter(String)

Returns the value of a request parameter as a String or null if the parameter does not exists.

- ServletResponse

Defines object to assist a Servlet in sending a response to the client. The Servlet container creates ServletResponse object and passes it as an argument to the service() method.

i) pubic void setContentType(String type)

Sets the content type of the response being sent to the client.

MIME: - Multipurpose Internet Mail

ii) PrintWritergetWriter()
PrintWriter class has following methods
a) print() to display output on html page
b) println() to display output on html page
c) close()

Other two service methods.

- public void doGet(HttpServletRequest request, Http ServletResponse response) throws ServletException, IOException

- public void doPost(HttpServletRequest request, Http ServletResponse response) throws ServletException, IOException

- HttpServletRequest interface(extends ServletRequest)
Methods :: Inherited methods from ServletRequest

i) Cookie [] getCookies()
Returns the array of objects of cookies otherwise null.

ii) HttpSessiongetSession()
Gets the new session object

iii) HttpSessiongetSession(boolean)

- HttpServletResponse interface (extends ServletResponse)
Methods :: Inherited methods from ServletResponse

i) public void addCookie(Cookie object)
Will add cookies on client hard disk

3) public void destroy()

When instructed to unload the Servlet, perhaps by the server administrator or programmatically by the Servlet itself, Servlet engine calls the Servlet's destroy() method. The Servlet is the eligible for garbage collection.
All resources which were allocated in init() method should be released in the destroy() method. The method is run once.

4) **getServletConfig()**
5) **getServletInfo()**

Deployment Descriptor
When you deploy your servlet into your web Container, you'll create a fairly simple XML document called the Deployment Descriptor (DD) to tell the Container how to run yourservlets and JSPs

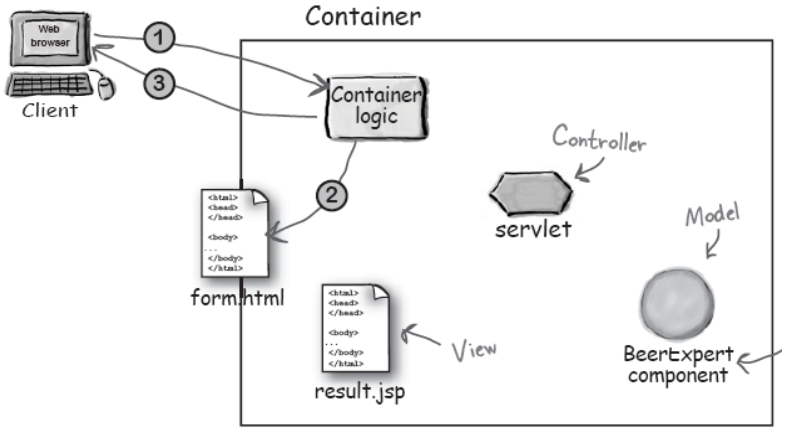
The DD elements for URL mapping:
<servlet>maps internal name to fully-qualified class name <servlet-mapping>maps internal name to public URL name

```
<web-app ...>
        <servlet>
                <servlet-name>Internal name 1</servlet-name>
                <servlet-class> Servlet class</servlet-class>
        </servlet>

        <servlet-mapping>
                <servlet-name>Internal name 1</servlet-name>
                <url-pattern>/Public1</url-pattern>
        </servlet-mapping>

</web-app>
```

Internal name 1
The <servlet-name> element is used to tie a <servlet> element to a specific <servlet-mapping> element. The end-user NEVER sees this name; it's used only in other parts of the DD.
Servlet class
You put in the fully-qualified name of the class (but you don't add the ".class" extension).
<servlet-mapping>
Think of the <servlet-mapping> element as what the Container uses at runtime when a request comes in, to ask, "which servlet should I invoke for this requested URL?".



1 - The client makes a request for the *form.html* page.

2 - The Container retrieves the *form.html* page.

3 - The Container returns the page to the browser, where the user answers the questions on the form and...

4 - The browser sends the request data to the container.

5 - The Container finds the correct servlet based on the URL, and passes the request to the servlet.

6 - The servlet calls the BeerExpert for help.

6 - The servlet calls the BeerExpert for help.

7 - The expert class returns an answer, which the servlet adds to the request object.

8 - The servlet forwards the request to the JSP.

9 - The JSP gets the answer from the request object.

10. The JSP Generate page for the container.
11. The container returns the page to the user.

Container runs multiple threads to process multiple requests.

Every client request generates a new pair of request and response objects.

Container allocates new request and response object.

Each request runs in a separate thread.

ServletConfig Object

ServletConfig interface have following methods.

getServletConfig() returns the object of ServletConfig interface.

1) getInitParameter(String)
2) getServletContext()

- One ServletConfig object per servlet.
- Use it to pass deploy time information to the servlet(e.g. database name)
- Use it to access ServletContext.
- Parameter are configured in deployment descriptor.

ServletContext Object

ServletContext interface have following methods.

getServletContext() returns the object of ServletContext interface.

1) getInitParameter(String)
2) getInitParameterNames()
3) String getAttribute()
4) getAttributeNames()
5) setAttribute(String,object)
6) removeAttribute(String)
7) getRequstDispatcher(String)

- One ServletContext per web app.(They should have AppContext)
- Use it to access web app parameter(also configured in the deployment Descriptor).
- Use it as a kind of application bulletin-board, where you can put up message (Called attributes) that other parts of the application can access.

- Use it to get sever info, including the name and version of the container and the version of the API that's supported.

==========

Servlet init parameters

Deployment Descriptor

```
<servlet>
    <servlet-name>  Demo</servlet-name>
    <servlet-class>  Demo</servlet-class>
    <init-param>
        <param-name>ranjit</param-name>
        <param-value>ranjit.bhosle2013@gmail.com</param-value>
    </init-param>
```

</servlet>
In the servlet code:
Out.println(getServletConfig().getInitParameter("ranjit"));
Context init parameters
Deployment Descriptor
<web-apps>
Other servlet declarations

<context-param>
<param-name>ranjit</param-name>
<param-value> ranjit.bhosle2013@gmail.com</param-value>
</ context-param>
</web-apps>
In the servlet code:
Out.println(getServletContext().getInitParameter("ranjit"));

Availability
Sevletinit parameters are available to only the servlet for which <init-param> was Configured.
Context init parameters are available to any servlets and jsps that are parts of these Web app.
==========

Navigation from one servlet to other:
We can call one servlet from other or can navigate to one servlet from other by using following three methods.

1. Public void sendRedirect(string path)this method declared in HttpServletResponse interface.
2. Using following methods declared in RequestDispatcher interface.
3.

Void	Forward (servletRequest request, servletResponse response) Forward a request from a servlet to another resource(servlet, JSP file,or HTML file)on the server.
void	Include(servletRequest request, servletResponse response) Include the content of a resource(servlet, JSP page, HTML file)in the response.

1. **sendRedirect()** : this method directly takes the flow to the called servlet. And new request and response objects are created for that servlet.
   sendRdirected can redirect to any page.E.g. html, servlet, jsp
2. **forword()** : this method also takes the control directly to the called servlet but servlet uses request and response objects which are passed as a parameter in forward method We can pass the current request and response object in it.
   This method can redirect only to servlet and jsp pages.
3. **include()**:: this method works similar to forward but after executing the called servlet control comes back to calling servlet and output of both servlet is combined and displayed

This method can redirect only to servlet and jsp pages

ServletConfig	**ServletContext**
The ServletConfig interface is implemented by the servlet container in order to pass configuration information to a servlet. The server passes an object that implements the ServletConfig interface to the servlet's init() method.	A ServletContext defines a set of methods that a servlet uses to communicate with its servlet container.
There is one ServletConfig parameter per servlet.	There is one ServletContext for the entire webapp and all the servlets in a webapp share it.
The param-value pairs for ServletConfig object are specified in the <init-param> within the <servlet> tags in the web.xml file	The param-value pairs for ServletContext object are specified in the <context-param> tags in the web.xml file.

forward()	**sendRedirect()**
A forward is performed internally by the servlet.	A redirect is a two step process, where the web application instructs the browser to fetch a second URL, which differs from the original.
The browser is completely unaware that it has taken place, so its original URL remains intact.	The browser, in this case, is doing the work and knows that it's making a new request.
Any browser reload of the resulting page will simple repeat the original request, with the original URL	A browser reloads of the second URL ,will not repeat the original request, but will rather fetch the second URL.
Both resources must be part of the same context (Some containers make provisions for cross-context communication but this tends not to be very portable)	This method can be used to redirect users to resources that are not part of the current context, or even in the same domain.

Since both resources are part of same context, the original request context is retained	Because this involves a new request, the previous request scope objects, with all of its parameters and attributes are no longer available after a redirect. (Variables will need to be passed by via the session object).
Forward is marginally faster than redirect.	redirect is marginally slower than a forward, since it requires two browser requests, not one.

include()	**forward()**
The RequestDispatcher include() method inserts the the contents of the specified resource directly in the flow of the servlet response, as if it were part of the calling servlet.	The RequestDispatcher forward() method is used to show a different resource in place of the servlet that was originally called.
If you include a servlet or JSP document, the included resource must not attempt to change the response status code or HTTP headers, any such request will be ignored.	The forwarded resource may be another servlet, JSP or static HTML document, but the response is issued under the same URL that was originally requested. In other words, it is not the same as a redirection.
The include() method is often used to include common "boilerplate" text or template markup that may be included by many servlets.	The forward() method is often used where a servlet is taking a controller role; processing some input and deciding the outcome by returning a particular response page.

ServletRequest.getRequestDispatcher(String path)	**ServletContext.getRequestDispatcher(String path)**
The getRequestDispatcher(String path) method ofjavax.servlet.ServletRequest interface accepts parameter the path to the resource to be included or forwarded to, which can be relative to the request of the calling servlet. If the path begins with a "/" it is interpreted as relative to the current context root.	The getRequestDispatcher(String path) method ofjavax.servlet.ServletContext interface cannot accept relative paths. All path must start with a "/" and are interpreted as relative to current context root.

Cookies

The browser generates request. The request is forwarded to the web-server via internet. The Web-Server immediately respond to this request and forget about the Browser and its request. This cause a problem for the web sites attempting to do business on the internet.
E.g. Web server find it difficult to create a single bill for multiple purchase being made by a user.
This is because the Web server treats each request unique one, different from any of its earlier requests.
To overcome this issue, somehow the web server must be able to register that the request for the multiple product purchase are originated from the very same Browser only then can a single bill be
Created and delivered to the user who have purchased multiple products form a website.
Cookies are small text files that are created by a Servlet program and stored on user's computer(Hard disk)
Cookies contains:-

1. The name of the cookie
2. The value of the cookie.
3. The expiration date of Cookie
4. The IP of the web server form where the cookie originated.
5. A unique ID number.
6. A date and time stamp

Where Cookies are used :-

1. Shopping cart applications
2. Online Banking
3. Generation of a Visitor's profile
4. Fee based services
5. Website Tracking

Working with Cookies
Package:-javax.servlet.http.cookie
Cookie class
Constructor
Public Cookies(String name, String value)
Name:- Cookies name(this is mandatory). The name cannot be changed after creation of the cookie.
Value: value of the cookie(e.g username)

Methods:

1. public void setMaxAge(int expiry)
   Sets the maximum age in seconds for this Cookie.

   A positive value indicates that the cookie will expire after that many seconds have passed. Note that the value is the *maximum* age when the cookie will expire, not the cookie's current age.

A negative value means that the cookie is not stored persistently and will be deleted when the Web browser exits. A zero value causes the cookie to be deleted.

Parameters:

expiry - an integer specifying the maximum age of the cookie in seconds; if negative, means the cookie is not stored; if zero, deletes the cookie

publicintgetMaxAge()

Gets the maximum age in seconds of this Cookie.

By default, -1 is returned, which indicates that the cookie will persist until browser shutdown.

Returns:

2. an integer specifying the maximum age of the cookie in seconds; if negative, means the cookie persists until browser shutdown.
3. public intgetMaxAge()
   Gets the maximum age in seconds of this Cookie.

   By default, -1 is returned, which indicates that the cookie will persist until browser shutdown.
   Returns:
   an integer specifying the maximum age of the cookie in seconds; if negative, means the cookie persists until browser shutdown.
4. public java.lang.StringgetValue()
   Gets the current value of this Cookie.

5. public java.lang.StringgetName()
   Gets the current Name of this Cookie.

- **Setting a Cookie**
  To set the call the constructor of the cookie.
  Cookie c1=new Cookie(“username”,”Atharav1”)
- **Sending a Cookie the client**
  Consider res is a object of HttpServletRespones interface
  Res.addCookies(c1)

No cookie is send to the browser, if above line of code spec is not included after creating a cookie.

- **Retrieving Cookies**

  HttpServletRequset interface
  Cookie[] getCookies()
  The getCookies() method returns an array of Cookie objects
  e.g.
  considerreq is a object of HttpServletRequest interface
  cookiearr[]=req.getCookies();

- **Destroying a cookie**

  Following are the steps that destroy an already existing cookie.

  1. Set the Cookie's value of null
  2. Set the maximum age of cookie to zero

  Example:
  Cookie c1= new Cookie("MyCookie","");
  C1.setMaxAge(0);
  Res.addCookie(c1);

Session

Http is a stateless protocol . A stateless protocol means that the protocol cannot remember prior connections and thus distinguish one visitor's request for cannot distinguish form that of another. FTP is not stateless protocol. Connections are not opened and closed with every request. After initial login, the FTP server maintains the visitor's credentials throughout the tracked.
Because of HTTPs stateless nature, a visitor's who navigates through a Web site cannot be tracked.
State is extremely necessary for secure sites hat:

1. Require visitors log in
2. Provide a virtual shopping cart[The E-commerce sites].

Disadvantages of stateless Nature

1. Increased overhead required creating a new connection with each request.
2. Inability to track a single visitor as the visitor traverse a web site.

Sessions are great of storing temporary data about the visitors. Sessions are used when the Web site does not want this data to be accessible external to the web server.
Session data is stored on the web server and deleted when browser is closed.

A session works even when a client has disabled cookies in their browser.

A session refers to all the requests that a single client might make to a server in the course of viewing any pages associated with a given application. Sessions are specific to both the individual user and the application. As a result, every user of an application has a separate session and has access to a separate set of session variables.

Difference

Cookie	Session
1. Cookie is stored on the clients hard disk.	1. Session is stored on the Web server.
2. Cookie are not secure than session Because they are stored on clients hard disk.	2 Session are more secure than cookie.

The Httpsession interface
Package :- The javax.servlet .http
This HttpSession interface is implemented by the server. It enables the server to read and write the state information that is associated with an HTTP session.
Methods:-

1. Long getCreationTime()
   Returns the time(in millisecond since midnight January 1,1970.) when this session was created.
2. String getId()
   Returns the session ID.
3. Long getlastAccessedTime()
   Returns the time when the client lost mods.
   A request for this session
4. void invalidate()
   Invalidates this session and removes it form the context
5. Boolean isNew()
   Returns true if the server created the session
6. void removeAttribute(string attr)
   Remove the attribute specified by the attr form the session
7. void setAttribute(string attr, Object val)
   Association the value passed in val with the attribute name passed to it,
8. String getAttribute()

What is Session Tracking?

Session tracking is a mechanism that servlets use to maintain state about a series of requests from the same user (that is, requests originating from the same browser) across some period of time.

What is the need of Session Tracking in web application?

HTTP is a stateless protocol i.e., every request is treated as new request. For web applications to be more realistic they have to retain information across multiple requests. Such information which is part of the application is reffered as "state". To keep track of this state we need session tracking.

Typical example: Putting things one at a time into a shopping cart, then checking out--each page request must somehow be associated with previous requests.

.What are the types of Session Tracking ?

Sessions need to work with all web browsers and take into account the users security preferences. Therefore there are a variety of ways to send and receive the identifier:

- **URL rewriting :** URL rewriting is a method of session tracking in which some extra data (session ID) is appended at the end of each URL. This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session. This method is used with browsers that do not support cookies or where the user has disabled the cookies.
- **Hidden Form Fields :** Similar to URL rewriting. The server embeds new hidden fields in every dynamically generated form page for the client. When the client submits the form to the server the hidden fields identify the client.
- **Cookies :** Cookie is a small amount of information sent by a servlet to a Web browser. Saved by the browser, and later sent back to the server in subsequent requests. A cookie has a name, a single value, and optional attributes. A cookie's value can uniquely identify a client.
- **Secure Socket Layer (SSL) Sessions :** Web browsers that support Secure Socket Layer communication can use SSL's support via HTTPS for generating a unique session key as part of the encrypted conversation.

What are some advantages of storing session state in cookies?

- Cookies are usually persistent, so for low-security sites, user data that needs to be stored long-term (such as a user ID, historical information, etc.) can be maintained easily with no server interaction.
- For small- and medium-sized session data, the entire session data (instead of just the session ID) can be kept in the cookie.

What are some disadvantages of storing session state in cookies?

- Cookies are controlled by programming a low-level API, which is more difficult to implement than some other approaches.
- All data for a session are kept on the client. Corruption, expiration or purging of cookie files can all result in incomplete, inconsistent, or missing information.
- Cookies may not be available for many reasons: the user may have disabled them, the browser version may not support them, the browser may be behind a firewall that filters cookies, and so on. Servlets and JSP pages that rely exclusively on cookies for client-side session state will not operate properly for all clients. Using cookies, and then switching to an alternate client-side session state strategy in cases where cookies aren't available, complicates development and maintenance.
- Browser instances share cookies, so users cannot have multiple simultaneous sessions.
- Cookie-based solutions work only for HTTP clients. This is because cookies are a feature of the HTTP protocol. Notice that the while package javax.servlet.http supports session management (via class HttpSession), package javax.servlet has no such support.

What is URL rewriting?

URL rewriting is a method of session tracking in which some extra data is appended at the end of each URL. This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session.

Every URL on the page must be encoded using method HttpServletResponse.encodeURL(). Each time a URL is output, the servlet passes the URL to encodeURL(), which encodes session ID in the URL if the browser isn't accepting cookies, or if the session tracking is turned off. E.g., http://abc/path/index.jsp;jsessionid=123465hfhs

Advantages

- URL rewriting works just about everywhere, especially when cookies are turned off.
- Multiple simultaneous sessions are possible for a single user. Session information is local to each browser instance, since it's stored in URLs in each page being displayed. This scheme isn't foolproof, though, since users can start a new browser instance using a URL for an active session, and confuse the server by interacting with the same session through two instances.
- Entirely static pages cannot be used with URL rewriting, since every link must be dynamically written with the session state. It is possible to combine static and dynamic content, using (for example) templating or server-side includes. This limitation is also a barrier to integrating legacy web pages with newer, servlet-based pages.

DisAdvantages

- Every URL on a page which needs the session information must be rewritten each time a page is served. Not only is this expensive computationally, but it can greatly increase communication overhead.
- URL rewriting limits the client's interaction with the server to HTTP GETs, which can result in awkward restrictions on the page.
- URL rewriting does not work well with JSP technology.
- If a client workstation crashes, all of the URLs (and therefore all of the data for that session) are lost.

How can an existing session be invalidated?

An existing session can be invalidated in the following two ways:

- Setting timeout in the deployment descriptor: This can be done by specifying timeout between the<session-timeout>tags as follows:

```
<session-config>
    <session-timeout>10</session-timeout>
</session-config>
```

This will set the time for session timeout to be ten minutes.

- Setting timeout programmatically: This will set the timeout for a specific session. The syntax for setting the timeout programmatically is as follows:

```
public void setMaxInactiveInterval(int interval)
```

The setMaxInactiveInterval() method sets the maximum time in seconds before a session becomes invalid.
Note :Setting the inactive period a*s negative(-1), makes the container stop tracking session, i.e, session never expires.*

How can the session in Servlet can be destroyed?

An existing session can be destroyed in the following two ways:

- Programatically : Using session.invalidate() method, which makes the container abonden the session on which the method is called.

What is servlet lazy loading?

- A container doesnot initialize the servlets ass soon as it starts up, it initializes a servlet when it receives a request for that servlet first time. This is called lazy loading.
- The servlet specification defines the <load-on-startup> element, which can be specified in the deployment descriptor to make the servlet container load and initialize the servlet as soon as it starts up.
- The process of loading a servlet before any request comes in is called preloading or preinitializing a servlet.

What is Servlet Chaining?

Servlet Chaining is a method where the output of one servlet is piped into a second servlet. The output of the second servlet could be piped into a third servlet, and so on. The last servlet in the chain returns the output to the Web browser.

Filters

Filters are Java components that are used to intercept an incoming request to a Web resource and a response sent back from the resource. It is used to abstract any useful information contained in the request or response. Some of the important functions performed by filters are as follows:

- Security checks
- Modifying the request or response
- Data compression
- Logging and auditing
- Response compression

Filters are configured in the deployment descriptor of a Web application. Hence, a user is not required to recompile anything to change the input or output of the Web application.

Some common tasks that we can do with servlet filters are:

- Logging request parameters to log files.
- Authentication and autherization of request for resources.
- Formatting of request body or header before sending it to servlet.
- Compressing the response data sent to the client.
- Alter response by adding some cookies, header information etc.

Servlet Filter interface

Servlet Filter interface is similar to Servlet interface and we need to implement it to create our own servlet filter. Servlet Filter interface contains lifecycle methods of a Filter and it's managed by servlet container.

Servlet Filter interface lifecycle methods are:

A. **void init(FilterConfig config) –**
When container initializes the Filter, this is the method that gets invoked. This method is called only once in the lifecycle of filter and we should initialize any resources in this method. FilterConfig is used by container to provide init parameters and servlet context object to the Filter. We can throw ServletException in this method.

B. **doFilter(ServletRequest paramServletRequest, ServletResponse paramServletResponse, FilterChain paramFilterChain) –**
This is the method invoked every time by container when it has to apply filter to a resource. Container provides request and response object references to filter as argument. FilterChain is used to invoke the next filter in the chain. This is a great example of Chain of Responsibility Pattern.

C. **void destroy()** – When container offloads the Filter instance, it invokes the destroy() method. This is the method where we can close any resources opened by filter. This method is called only once in the lifetime of filter.

Filter chaining

FilterChain interface:

Multiple filters can be written and applied to the same URL pattern . The order of executing is determined by ordering in the deployment descriptor.

A FilterChain is an object provided by the servlet container to the developer giving a view into the invocation chain of a filtered request for a resource. Filters use the FilterChain to invoke the next filter in the chain, or if the calling filter is the last filter in the chain, to invoke the resource at the end of the chain.

void	**doFilter**(ServletRequest request, ServletResponse response) Causes the next filter in the chain to be invoked, or if the calling filter is the last filter in the chain, causes the resource at the end of the chain to be invoked.

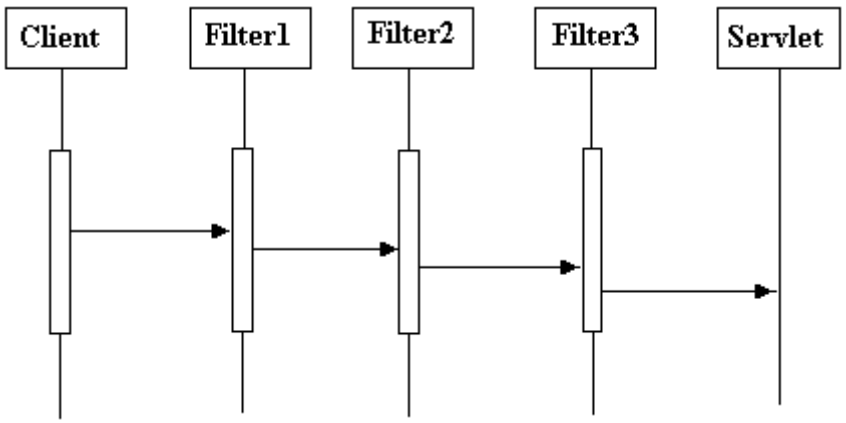
Servlet Filter configuration in web.xml

```
<filter>
   <filter-name>RequestLoggingFilter</filter-name> <!-- mandatory -->
   <filter-class> servlet.filters.RequestLoggingFilter</filter-class> <!-- mandatory -->
   <init-param> <!-- optional -->
   <param-name>test</param-name>
   <param-value>testValue</param-value>
   </init-param>
 </filter>
```

What are the functions of an intercepting filter?

The functions of an intercepting filter are as follows:

- It intercepts the request from a client before it reaches the servlet and modifies the request if required.
- It intercepts the response from the servlet back to the client and modifies the request if required.
- There can be many filters forming a chain, in which case the output of one filter becomes an input to the next filter. Hence, various modifications can be performed on a single request and response.



What are the functions of the Servlet container?

The functions of the Servlet container are as follows:

- **Lifecycle management** : It manages the life and death of a servlet, such as class loading, instantiation, initialization, service, and making servlet instances eligible for garbage collection.
- **Communication support** : It handles the communication between the servlet and the Web server.
- **Multithreading support** : It automatically creates a new thread for every servlet request received. When the Servlet service() method completes, the thread dies.
- **Declarative security** : It manages the security inside the XML deployment descriptor file.
- **JSP support** : The container is responsible for converting JSPs to servlets and for maintaining them

==========