1. **What are Servlets?**
2. A servlet is a **Java programming language class** used to extend the capabilities of servers that host applications accessed by means of a **request-response programming model**. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines **HTTP-specific servlet classes**.

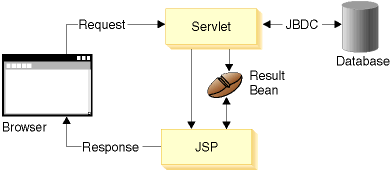
**The javax.servlet and javax.servlet.http** packages provide **interfaces** and classes for writing servlets. All servlets must implement the Servlet interface, which defines **lifecycle methods**. When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.

1. **What are the advantages of servlets over CGI?**
2. Servlets offer several advantages in comparison with the CGI.

* Performance is significantly better.
* Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
* Servlets are platform-independent because they are written in Java.
* Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
* The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

1. **What are the major tasks of servlets?**
2. Servlets perform the following major tasks:

* Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
* Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
* Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
* Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
* Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

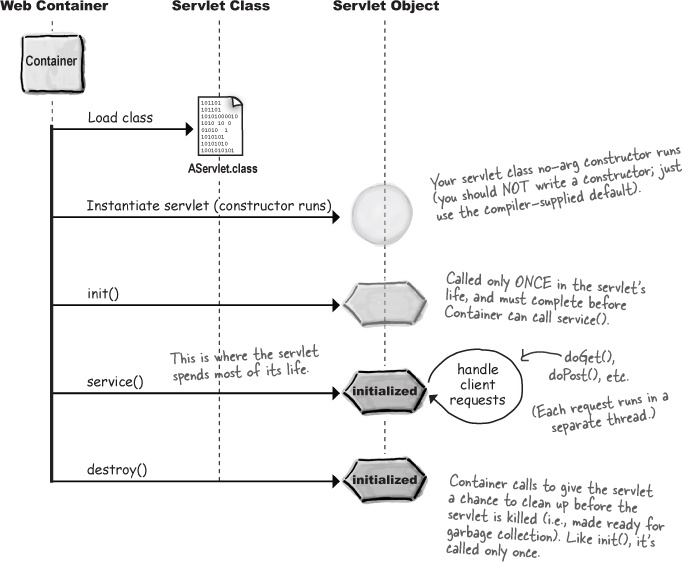


1. **Explain servlet life cycle.**
2. A servlet life cycle can be defined as the entire process from its creation till the destruction. The sequence in which the Web container calls the life cycle methods of a servlet is:
3. The Web container loads the servlet class.
4. Creates an instance of the servlet class.
5. The Web container invokes init() method of the servlet instance during initialization of the servlet**.**

**Note:** The init() method is invoked only once in the servlet life cycle.

1. The Web container invokes the service() method to allow a servlet to process a client request.
2. The service() method processes the request and returns the response back to the Web container.
3. The servlet then waits to receive and process subsequent requests as explained in steps 4 and 5.
4. The Web container calls the destroy() method before removing the servlet instance from the service.

**Note:** The destroy() method is also invoked only once in a servlet life cycle.



1. **When init() method of servlet gets called?**
   1. The init() method is called during initialization phase of the servlet life cycle.

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request.

The init() method simply creates or loads some data that will be used throughout the life of the servlet. To customize this process, one can override the init method of the Servlet interface.

The Web container first maps the requested URL to the corresponding servlet available in the Web container and then instantiates the servlet.

The Web container then creates an object of the ServletConfig interface, which contains the start-up configuration information, such as persistent configuration data, initialize resources, and perform any other one-time activities.

The Web container then calls the init() method of the servlet and passes the ServletConfig object to it.

The init() method throws a ServletException if the Web container cannot initialize the servlet resources. A servlet that cannot complete its initialization process should throw UnavailableException.

The servlet initialization completes before any client requests are accepted.

The following code snippet shows the init() method:

public class ServletLifeCycle extends HttpServlet

{

static int count;

public void init(ServletConfig config) throws ServletException

{

count=0;

}

}

**Note:** init() is a convenience method which can be overridden so that there's no need to call super.init(config).

Instead of overriding [init(ServletConfig)](https://docs.oracle.com/javaee/6/api/javax/servlet/GenericServlet.html#init(javax.servlet.ServletConfig)), simply override init() method and it will be called by GenericServlet.init(ServletConfig config). The ServletConfig object can still be retrieved via [getServletConfig()](https://docs.oracle.com/javaee/6/api/javax/servlet/GenericServlet.html#getServletConfig()).

init() could have been abstract, but then each servlet would be forced to implement it. This way, by default, nothing happens on init(), and each servlet can override this behaviour.

<https://docs.oracle.com/javaee/6/api/javax/servlet/GenericServlet.html#init()>

<http://beginnersbook.com/2013/05/servlet-life-cycle/>

<http://docs.oracle.com/javaee/5/tutorial/doc/bnafu.html>

<http://stackoverflow.com/questions/5140416/what-is-the-servlets-init-method-used-for>

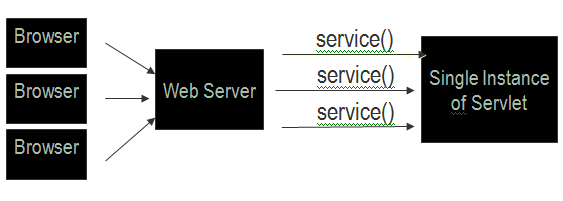
1. **When service() method of servlet gets called?**
2. The service() method processes the client requests. Each time the Web container receives a client request, it invokes the service() method. The service() method is invoked only after the initialization of the servlet is complete. When the Web container calls the service() method, it passes an object of the ServletRequest interface and an object of the ServletResponse interface. The ServletRequest object contains information about the service request made by the client. The ServletResponse object contains the information returned by the servlet to the client. The following code snippet shows the service() method:

public void service(ServletRequest req, ServletResponse res)

throws ServletException, IOException

The service() method throws ServletException exception when an exception occurs that interferes with the normal operation of the servlet. The service() method throws IOException when an input or output exception occurs.

The service() method dispatches a client request to one of the request handler methods of the HttpServlet interface, such as the doGet(), doPost(), doHead() or doPut(). The request handler methods accept the objects of the HttpServletRequest and HttpServletResponse as parameters from the service() method.



**Note:** This method is declared abstract so subclasses, such as HttpServlet, must override it.

1. **When doGet() method of servlet to be called?**
2. The doGet() method is called by the server (via the service method) to allow a servlet to handle a GET request, which is sent by the client, using the HTTP GET method.

GET is a type of HTTP request method that is commonly used to retrieve static resources.

To handle client requests that are received using GET method, we need to override the doGet() method in the servlet class. In the doGet() method, we can retrieve the client information of the HttpServletRequest object.

We can use the HttpServletResponse object to send the response back to the client. The following code snippet shows the doGet() method:

protected void **doGet**([HttpServletRequest](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html) req,

[HttpServletResponse](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html) resp)

throws [ServletException](https://docs.oracle.com/javaee/6/api/javax/servlet/ServletException.html),

java.io.IOException

The GET method should be safe, that is, without any side effects for which users are held responsible. For example, most form queries have no side effects. If a client request is intended to change stored data, the request should use some other HTTP method.

The GET method should also be idempotent, meaning that it can be safely repeated. Sometimes making a method safe also makes it idempotent. For example, repeating queries is both safe and idempotent, but buying a product online or modifying data is neither safe nor idempotent.

If the request is incorrectly formatted, doGet returns an HTTP "Bad Request" message.

1. **When doPost() method of servlet to be called?**
2. The doPost() method is called by the server (via the service method) to allow a servlet to handle a POST request, which is sent by the client, using the HTTP POST method.

The HTTP POST method allows the client to send data of unlimited length to the Web server a single time and is useful when posting information such as credit card numbers.

For example, if a client is entering registration data in an HTML form, the data can be sent using the POST method. Unlike the GET method, the POST request sends the data as part of the HTTP request body. As a result, the data sent does not appear as a part of URL.

To handle requests in a servlet that is sent using the POST method, we need to override the doPost() method. In the doPost() method, we can process the request and send the response back to the client. The following code snippet shows the doPost() method:

protected void **doPost**([HttpServletRequest](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html) req,

[HttpServletResponse](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html) resp)

throws [ServletException](https://docs.oracle.com/javaee/6/api/javax/servlet/ServletException.html),

java.io.IOException

This method does not need to be either safe or idempotent. Operations requested through POST can have side effects for which the user can be held accountable, for example, updating stored data or buying items online.

If the HTTP POST request is incorrectly formatted, doPost returns an HTTP "Bad Request" message.

1. **When destroy() method of servlet gets called?**
2. The destroy() method marks the end of the life cycle of a servlet. The Web container calls the destroy() method before removing a servlet instance from the service. The Web container calls the destroy() method when

* The time period specified for the servlet has elapsed. The time period of a servlet is the period for which the servlet is kept in the active state by the Web container to service the client request.
* The Web container needs to release servlet instances to conserve memory.
* The Web container is about to shut down.

In the destroy() method we can write the code to release the resources occupied by the servlet. The destroy() method is also used to save any persistent information before the servlet instance is removed from the service. The following code snippet shows the destroy() method:

public void destroy();

This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

1. **How to read form data in servlet?**
2. Servlets handles form data parsing automatically using the following methods depending on the situation:

**getParameter():**

You call request.getParameter() method to get the value of a form parameter.

java.lang.String **getParameter**(java.lang.String name)

It returns the value of a request parameter as a String (or null if the parameter does not exist). Request parameters are extra information sent with the request. For HTTP servlets, parameters are contained in the query string or posted form data.

You should only use this method when you are sure the parameter has only one value. If the parameter might have more than one value, use getParameterValues( java.lang.String).

If you use this method with a multivalued parameter, the value returned is equal to the first value in the array returned by getParameterValues.

If the parameter data was sent in the request body, such as occurs with an HTTP POST request, then reading the body directly via getInputStream() or getReader() can interfere with the execution of this method.

**getParameterValues():**

Call this method if the parameter appears more than once and returns multiple values, for example checkbox.

java.lang.String[] **getParameterValues**(java.lang.String name)

Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.

If the parameter has a single value, the array has a length of 1.

**getParameterNames():**

getParameterNames() method of HttpServletRequest returns complete list of all parameters in the current request.

java.util.Enumeration<java.lang.String> **getParameterNames**()

Returns an Enumeration of String objects containing the names of the parameters contained in this request. If the request has no parameters, the method returns an empty Enumeration.

Once we have an Enumeration, we can loop down the Enumeration in the standard manner, using hasMoreElements() method to determine when to stop and using nextElement() method to get each parameter name.

1. **How to read http header information in servlet?**
   1. Returns an enumeration of all the header names this request contains. If the request has no headers, this method returns an empty enumeration.

java.util.Enumeration<java.lang.String> getHeaderNames()

Some servlet containers do not allow servlets to access headers using this method, in which case this method returns null

1. **What is HTTPServletRequest class?**
   1. When a browser requests for a web page, it sends lot of information to the web server which cannot be read directly because this information travel as a part of header of HTTP request. HTTPServletRequest represents this HTTP Request.

public interface **HttpServletRequest**

extends [ServletRequest](http://docs.oracle.com/javaee/6/api/javax/servlet/ServletRequest.html)

HTTPServletRequest extends the [ServletRequest](http://docs.oracle.com/javaee/6/api/javax/servlet/ServletRequest.html) interface to provide request information for HTTP servlets.

The servlet container creates an HttpServletRequest object and passes it as an argument to the servlet's service methods (doGet, doPost, etc).

1. **What is HTTPServletResponse class?**
2. When a Web server responds to a HTTP request to the browser, the response typically consists of a status line, some response headers, a blank line, and the document. HTTPServletResponse represents this HTTP Response.

public interface **HttpServletResponse**

extends [ServletResponse](http://docs.oracle.com/javaee/6/api/javax/servlet/ServletResponse.html)

**HttpServletResponse** Extends the ServletResponse interface to provide HTTP-specific functionality in sending a response. For example, it has methods to access HTTP headers and cookies.

The servlet container creates an HttpServletResponse object and passes it as an argument to the servlet's service methods (doGet, doPost, etc).

1. **How to write html contents using servlets?**
2. Get the object of PrintWriter using request.

PrintWriter out = response.getWriter();

Now print html.

out.println("Hello World");

// Import required java libraries

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

// Extend HttpServlet class

public class HelloForm extends HttpServlet {

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

{

// Set response content type

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String title = "Using GET Method to Read Form Data";

String docType =

"<!doctype html public \"-//w3c//dtd html 4.0 " +

"transitional//en\">\n";

out.println(docType +

"<html>\n" +

"<head><title>" + title + "</title></head>\n" +

"<body bgcolor=\"#f0f0f0\">\n" +

"<h1 align=\"center\">" + title + "</h1>\n" +

"<ul>\n" +

" <li><b>First Name</b>: "

+ request.getParameter("first\_name") + "\n" +

" <li><b>Last Name</b>: "

+ request.getParameter("last\_name") + "\n" +

"</ul>\n" +

"</body></html>");

}

}

1. **How to send an authentication error from a servlet?**
2. We can use setStatus(statuscode) method of HttpServletResponse to send an authentication error.

// Set error code and reason.

response.sendError(407, "Need authentication!!!" );

1. **How to redirect a request from a servlet to another servlet?**
2. Page redirection is generally used when a document moves to a new location and we need to send the client to this new location or may be because of load balancing, or for simple randomization. The simplest way of redirecting a request to another page is using method sendRedirect() of response object.

void **sendRedirect**(java.lang.String location)

throws java.io.IOException

This method Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer. The buffer will be replaced with the data set by this method. Calling this method sets the status code to SC\_FOUND 302 (Found). This method can accept relative URLs; the servlet container must convert the relative URL to an absolute URL before sending the response to the client. If the location is relative without a leading '/' the container interprets it as relative to the current request URI. If the location is relative with a leading '/' the container interprets it as relative to the servlet container root.

If the response has already been committed, this method throws an IllegalStateException. After using this method, the response should be considered to be committed and should not be written to.

1. **How sendError method works?**
2. This method sends a status code (usually 404) along with a short message that is automatically formatted inside an HTML document and sent to the client.

void **sendError**(int sc,

java.lang.String msg)

throws java.io.IOException

Sends an error response to the client using the specified status and clears the buffer. The server defaults to creating the response to look like an HTML-formatted server error page containing the specified message, setting the content type to "text/html". The server will preserve cookies and may clear or update any headers needed to serve the error page as a valid response. If an error-page declaration has been made for the web application corresponding to the status code passed in, it will be served back in preference to the suggested msg parameter and the msg parameter will be ignored.

If the response has already been committed, this method throws an IllegalStateException. After using this method, the response should be considered to be committed and should not be written to.

**Parameters:**

sc - the error status code

msg - the descriptive message

**Throws:**

java.io.IOException - If an input or output exception occurs

IllegalStateException - If the response was committed

1. **What are servlets filters?**
2. Servlet Filters are Java classes that can be used in Servlet Programming for the following purposes:

* To intercept requests from a client before they access a resource at back end.
* To manipulate responses from server before they are sent back to the client.

Filters perform filtering in the doFilter method. Every Filter has access to a FilterConfig object from which it can obtain its initialization parameters, and a reference to the ServletContext which it can use, for example, to load resources needed for filtering tasks.

Filters are configured in the deployment descriptor of a web application.

Examples that have been identified for this design are:

1. Authentication Filters
2. Logging and Auditing Filters
3. Image conversion Filters
4. Data compression Filters
5. Encryption Filters
6. Tokenizing Filters
7. Filters that trigger resource access events
8. XSL/T filters
9. Mime-type chain Filter
10. **How to do servlet filter mapping?**
11. Filters are deployed in the deployment descriptor file web.xml and then map to either servlet names or URL patterns in your application's deployment descriptor.

<filter>

<filter-name>LogFilter</filter-name>

<filter-class>LogFilter</filter-class>

<init-param>

<param-name>test-param</param-name>

<param-value>Initialization Paramter</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>LogFilter</filter-name>

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

</filter-mapping>

The above filter would apply to all the servlets because we specified /\* in our configuration. You can specify a particular servlet path if you want to apply filter on few servlets only.

1. **For what purpose init() method of a filter is used?**
2. This method is called by the web container to indicate to a filter that it is being placed into service.

void **init**([FilterConfig](https://docs.oracle.com/javaee/6/api/javax/servlet/FilterConfig.html" \o "interface in javax.servlet) filterConfig)

throws [ServletException](https://docs.oracle.com/javaee/6/api/javax/servlet/ServletException.html)

This method is called by the web container to indicate to a filter that it is being placed into service.

The servlet container calls the init method exactly once after instantiating the filter. The init method must complete successfully before the filter is asked to do any filtering work.

The web container cannot place the filter into service if the init method either

1. Throws a ServletException
2. Does not return within a time period defined by the web container
3. **For what purpose doFilter() method of a filter is used?**
4. This method is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.

void **doFilter**([ServletRequest](https://docs.oracle.com/javaee/6/api/javax/servlet/ServletRequest.html" \o "interface in javax.servlet) request,

[ServletResponse](https://docs.oracle.com/javaee/6/api/javax/servlet/ServletResponse.html) response,

[FilterChain](https://docs.oracle.com/javaee/6/api/javax/servlet/FilterChain.html) chain)

throws java.io.IOException,

[ServletException](https://docs.oracle.com/javaee/6/api/javax/servlet/ServletException.html)

The doFilter method of the Filter is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain. The FilterChain passed in to this method allows the Filter to pass on the request and response to the next entity in the chain.

A typical implementation of this method would follow the following pattern:

1. Examine the request
2. Optionally wrap the request object with a custom implementation to filter content or headers for input filtering
3. Optionally wrap the response object with a custom implementation to filter content or headers for output filtering
   * **Either** invoke the next entity in the chain using the FilterChain object (chain.doFilter()),
   * **or** not pass on the request/response pair to the next entity in the filter chain to block the request processing
4. Directly set headers on the response after invocation of the next entity in the filter chain.
5. **For what purpose destroy() method of a filter is used?**
   * 1. This method is called by the web container to indicate to a filter that it is being taken out of service.

void **destroy**()

This method is called by the web container to indicate to a filter that it is being taken out of service.

This method is only called once all threads within the filter's doFilter method have exited or after a timeout period has passed. After the web container calls this method, it will not call the doFilter method again on this instance of the filter.

This method gives the filter an opportunity to clean up any resources that are being held (for example, memory, file handles, threads) and make sure that any persistent state is synchronized with the filter's current state in memory.

1. **Can multiple filters be configured?**
2. Yes.

<filter>

<filter-name>LogFilter</filter-name>

<filter-class>LogFilter</filter-class>

<init-param>

<param-name>test-param</param-name>

<param-value>Initialization Paramter</param-value>

</init-param>

</filter>

<filter>

<filter-name>AuthenFilter</filter-name>

<filter-class>AuthenFilter</filter-class>

<init-param>

<param-name>test-param</param-name>

<param-value>Initialization Paramter</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>LogFilter</filter-name>

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

</filter-mapping>

<filter-mapping>

<filter-name>AuthenFilter</filter-name>

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

</filter-mapping>

1. **Can filtering be done in an ordered way? If so then how to achieve it?**
2. Yes. The order of filter-mapping elements in web.xml determines the order in which the web container applies the filter to the servlet. To reverse the order of the filter, you just need to reverse the filter-mapping elements in the web.xml file.

For example, above example would apply LogFilter first and then it would apply AuthenFilter to any servlet but the following example would reverse the order:

<filter-mapping>

<filter-name>AuthenFilter</filter-name>

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

</filter-mapping>

<filter-mapping>

<filter-name>LogFilter</filter-name>

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

</filter-mapping>

1. **How to configure a central error handling page in servlets?**
2. Use the error-page element in web.xml to specify the invocation of servlets in response to certain exceptions or HTTP status codes.

<!-- servlet definition -->

<servlet>

<servlet-name>ErrorHandler</servlet-name>

<servlet-class>ErrorHandler</servlet-class>

</servlet>

<!-- servlet mappings -->

<servlet-mapping>

<servlet-name>ErrorHandler</servlet-name>

<url-pattern>/ErrorHandler</url-pattern>

</servlet-mapping>

<!-- error-code related error pages -->

<error-page>

<error-code>404</error-code>

<location>/ErrorHandler</location>

</error-page>

<error-page>

<error-code>403</error-code>

<location>/ErrorHandler</location>

</error-page>

<!-- exception-type related error pages -->

<error-page>

<exception-type>

javax.servlet.ServletException

</exception-type >

<location>/ErrorHandler</location>

</error-page>

<error-page>

<exception-type>java.io.IOException</exception-type >

<location>/ErrorHandler</location>

</error-page>

1. **How to configure a central error handler in servlets?**
2. If you want to have a generic Error Handler for all the exceptions then you should define following error-page instead of defining separate error-page elements for every exception:

<error-page>

<exception-type>java.lang.Throwable</exception-type >

<location>/ErrorHandler</location>

</error-page>

1. **What are cookies?**
2. Cookies are text files stored on the client computer and they are kept for various information tracking purpose. Java Servlets transparently supports HTTP cookies.

public class Cookie

extends [Object](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html?is-external=true)

implements [Cloneable](http://docs.oracle.com/javase/7/docs/api/java/lang/Cloneable.html?is-external=true), [Serializable](http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html?is-external=true)

This class is used to create a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server. A cookie's value can uniquely identify a client, so cookies are commonly used for session management.

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. Some Web browsers have bugs in how they handle the optional attributes, so use them sparingly to improve the interoperability of your servlets.

The servlet sends cookies to the browser by using the [HttpServletResponse.addCookie( javax.servlet.http.Cookie)](https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletResponse.html" \l "addCookie-javax.servlet.http.Cookie-) method, which adds fields to HTTP response headers to send cookies to the browser, one at a time. The browser is expected to support 20 cookies for each Web server, 300 cookies total, and may limit cookie size to 4 KB each.

The browser returns cookies to the servlet by adding fields to HTTP request headers. Cookies can be retrieved from a request by using the [HttpServletRequest.getCookies()](https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html#getCookies--) method. Several cookies might have the same name but different path attributes.

Cookies affect the caching of the Web pages that use them. HTTP 1.0 does not cache pages that use cookies created with this class. This class does not support the cache control defined with HTTP 1.1.

This class supports both the Version 0 (by Netscape) and Version 1 (by RFC 2109) cookie specifications. By default, cookies are created using Version 0 to ensure the best interoperability.

1. **How to create a cookie using servlet?**

A. Setting cookies with servlet involves three steps:

1. Creating a Cookie object: You call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

Cookie cookie = new Cookie("key","value");

Keep in mind, neither the name nor the value should contain white space or any of the following characters: [ ] ( ) = , " / ? @ : ;

1. Setting the maximum age: You use setMaxAge to specify how long (in seconds) the cookie should be valid. Following would set up a cookie for 24 hours.

cookie.setMaxAge(60\*60\*24);

1. Sending the Cookie into the HTTP response headers: You use response.addCookie to add cookies in the HTTP response header as follows:

response.addCookie(cookie);

1. **How to read a cookie using servlet?**
2. To read cookies, you need to create an array of javax.servlet.http.Cookie objects by calling the getCookies( ) method of HttpServletRequest. Then cycle through the array, and use getName() and getValue() methods to access each cookie and associated value.
3. **How to delete a cookie using servlet?**
4. To delete cookies is very simple. If you want to delete a cookie then you simply need to follow up following three steps:

* Read an already exsiting cookie and store it in Cookie object.
* Set cookie age as zero using setMaxAge() method to delete an existing cookie.
* Add this cookie back into response header.

1. **What different scopes are available?**

|  |  |  |
| --- | --- | --- |
| **Scope** | **Annotation** | **Duration** |
| Request | @RequestScoped | A user’s interaction with a web application in a single HTTP request. |
| Session | @SessionScoped | A user’s interaction with a web application across multiple HTTP requests. |
| Application | @ApplicationScoped | Shared state across all users’ interactions with a web application. |
| Dependent | @Dependent | The default scope if none is specified; it means that an object exists to serve exactly one client (bean) and has the same lifecycle as that client (bean). |
| Conversation | @ConversationScoped | A user’s interaction with a JavaServer Faces application, within explicit developer-controlled boundaries that extend the scope across multiple invocations of the JavaServer Faces lifecycle. All long-running conversations are scoped to a particular HTTP servlet session and may not cross session boundaries. |

1. **What is session?**
2. Session provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user. The session persists for a specified time period, across more than one connection or page request from the user.
3. **What is URL rewriting?**
4. You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session. For example, with http://tutorialspoint.com/file.htm;sessionid=12345, the session identifier is attached as sessionid=12345 which can be accessed at the web server to identify the client.
5. **How to create a session in servlet?**
6. You would get HttpSession object by calling the public method getSession() of HttpServletRequest, as below:

// Create a session object if it is already not created.

HttpSession session = request.getSession();

1. **How to delete a session in servlet?**
2. When you are done with a user's session data, you have several options:

* **Remove a particular attribute:** You can call public void removeAttribute(String name) method to delete the value associated with a particular key.
* **Delete the whole session:** You can call public void invalidate() method to discard an entire session. Setting Session timeout: You can call public void setMaxInactiveInterval(int interval) method to set the timeout for a session individually.
* **Log the user out:** The servers that support servlets 2.4, you can call logout to log the client out of the Web server and invalidate all sessions belonging to all the users.

1. **How to update an attribute in session in servlet?**
2. setAttribute(String name, Object value) of HTTPSession object binds an object to this session, using the name specified and can be used to update an attribute in session.
3. **How getAttribute() and getParameter() are different?**
4. It is crucial to know that attributes are not parameters.

Additionally, there is no servlet specific attributes, and there are no session parameters.

|  |  |  |
| --- | --- | --- |
|  | **getAttribute()** | **getParameter()** |
| Declaration | java.lang.Object getAttribute(java.lang.String name) | java.lang.String getParameter(java.lang.String name) |
| Parameters | name - a String specifying the name of the attribute | name - a String specifying the name of the parameter |
| Returns | an Object containing the value of the attribute, or null if the attribute does not exist | a String representing the single value of the parameter |
| Usage | This is used for getting the parameters set previously in another or the same JSP or Servlet page. | Is used for getting the information you need from the Client's HTML page |
| When calling the getAttribute(String name) method, bear in mind that the attributes must be cast. | The return type for a parameter is a String so casting not required |
| Set an attribute in a servlet, and use a RequestDispatcher to forward a request to another resource (JSP / Servlet) | returns value of a parameter that was submitted by an HTML form or that was included in a query string of a HTTP request |

1. **How to set session timeout in servlet?**
2. setMaxInactiveInterval(int interval) of HTTPSession object specifies the time, in seconds, between client requests before the servlet container will invalidate this session.
3. **How to set auto page refresh in servlet?**
4. The simplest way of refreshing a web page is using method setIntHeader() of response object.
5. **What is internalization?**
6. This means enabling a web site to provide different versions of content translated into the visitor's language or nationality.
7. **What is localization?**
8. This means adding resources to a web site to adapt it to a particular geographical or cultural region for example Hindi translation to a web site.
9. **What is locale?**
   1. This is a particular cultural or geographical region. It is usually referred to as a language symbol followed by a country symbol which is separated by an underscore. For example "en\_US" represents english locale for US.
10. **How to detect locale in Servlets?**
    1. Following is the method of request object which returns Locale object.

java.util.Locale request.getLocale()

How to get country name in Servlets?

Following method returns a name for the locale's country that is appropriate for display to the user.

String getDisplayCountry()

1. **How to get country name in Servlets?**
2. Following method returns a name for the locale's country that is appropriate for display to the user.

String getDisplayCountry()

More questions:

<http://www.javatpoint.com/servletinterview>

Caching:

<http://www.javaworld.com/article/2072734/open-source-tools/j2ee-object-caching-frameworks.html?page=2>

XML:

<http://www.tutorialspoint.com/java_xml/java_xml_interview_questions.htm>

Security:

<https://docs.oracle.com/javaee/7/tutorial/security-webtier.htm#BNCAS>