**SERVLETS**

# Life Cycle of a Servlet

 The Servlet Container is responsible for maintaining the life cycle of a **[Servlet](https://www.javainterviewpoint.com/category/servlets/)**. The life cycle has the below following phases.

1. Load Servlet Class
2. Servlet instance is created
3. **init()** method is invoked
4. **service()** method is invoked
5. **destroy()** method is invoked

## 

## ****Load Servlet Class****

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

## Servlet instance is created

Once the Servlet class is loaded the Web Container creates the instance of it. Servlet instance will be created only once in the life cycle.

## init() method is invoked

The Web container calls the **init()** method after creating the servlet instance, the init() method is used to initialize the servlet. Below is the signature of the init() method

 public void init(ServletConfig config) throws ServletException

## service() method is invoked

The Web Container calls the **service()**method each time when the request for the servlet is received. when a request is received the server creates a new thread and calls the service() method. The **service()** method checks the HTTP request type (**GET, POST, PUT, DELETE**) and calls the appropriate **doGet()**, **doPost()**, **doPut()**, **doDelete()**methods. The Signature of the service() method is given below.

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException

## destroy() method is invoked

The destroy() method is called only once at the end of the life cycle of the servlet. This method gives the servlet an opportunity to do clean up of the resources such as closing the database, thread, etc. The signature of the destroy() method is given below

public void destroy()

**SERVLET API**

1. **servlet API**

Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :

* **javax.servlet**
* **javax.servlet.http**

## Package javax.servlet Description

|  |  |
| --- | --- |
| INTERFACES | CLASSES |
| Servlet | ServletInputStream |
| ServletContext | ServletOutputStream |
| ServletConfig | ServletRequestWrapper |
| ServletRequest | ServletResponseWrapper |
| ServletResponse | ServletRequestEvent |
| ServletContextListener | ServletContextEvent |
| RequestDispatcher | ServletRequestAttributeEvent |
| SingleThreadModel | ServletContextAttributeEvent |
| Filter | ServletException |
| FilterConfig | UnavailableException |
| FilterChain | GenericServlet |
| ServletRequestListener |  |

The javax.servlet package contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container.

Interfaces

|  |  |
| --- | --- |
| [**RequestDispatcher**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/RequestDispatcher.html) | Defines an object that receives requests from the client and sends them to any resource (such as a servlet, HTML file, or JSP file) on the server. |
| [**Servlet**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/Servlet.html) | Defines methods that all servlets must implement. |
| [**ServletConfig**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletConfig.html) | A servlet configuration object used by a servlet container to pass information to a servlet during initialization. |
| [**ServletContext**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletContext.html) | Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file. |
| [**ServletContextAttributeListener**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletContextAttributeListener.html) | Implementations of this interface receive notifications of changes to the attribute list on the servlet context of a web application. |
| [**ServletContextListener**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletContextListener.html) | Implementations of this interface receive notifications about changes to the servlet context of the web application they are part of. |
| [**ServletRequest**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletRequest.html) | Defines an object to provide client request information to a servlet. |
| [**ServletRequestAttributeListener**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletRequestAttributeListener.html) | A ServletRequestAttributeListener can be implemented by the developer interested in being notified of request attribute changes. |
| [**ServletRequestListener**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletRequestListener.html) | A ServletRequestListener can be implemented by the developer interested in being notified of requests coming in and out of scope in a web component. |
| [**ServletResponse**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletResponse.html) | Defines an object to assist a servlet in sending a response to the client. |

**Classes**

|  |  |
| --- | --- |
| [**GenericServlet**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/GenericServlet.html) | Defines a generic, protocol-independent servlet. |
| [**ServletContextAttributeEvent**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletContextAttributeEvent.html) | This is the event class for notifications about changes to the attributes of the servlet context of a web application. |
| [**ServletContextEvent**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletContextEvent.html) | This is the event class for notifications about changes to the servlet context of a web application. |
| [**ServletInputStream**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletInputStream.html) | Provides an input stream for reading binary data from a client request, including an efficient readLine method for reading data one line at a time. |
| [**ServletOutputStream**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletOutputStream.html) | Provides an output stream for sending binary data to the client. |

GenericServlet is an abstract class that provides implementation of most of the basic servlet methods. This is a very important class.

**Methods of GenericServlet class**

* public void init(ServletConfig)
* public abstract void service(ServletRequest request,ServletResposne response)
* public void destroy()
* public ServletConfig getServletConfig()
* public String getServletInfo()
* public ServletContext getServletContext()
* public String getInitParameter(String name)
* public Enumeration getInitParameterNames()
* public String getServletName()
* public void log(String msg)
* public void log(String msg, Throwable t)

## Package javax.servlet.http

The javax.servlet.http package contains a number of classes and interfaces that describe and define the contracts between a servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming servlet container.

|  |  |
| --- | --- |
| CLASSES and INTERFACES | |
| HttpServlet | HttpServletRequest |
| HttpServletResponse | HttpSessionAttributeListener |
| HttpSessionEvent | HttpSessionListener |
| Cookie | HttpSession |

|  |  |  |
| --- | --- | --- |
| **Interface Summary** | | |
| [**HttpServletRequest**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/http/HttpServletRequest.html) | | Extends the [ServletRequest](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletRequest.html" \o "interface in javax.servlet) interface to provide request information for HTTP servlets. |
| [**HttpServletResponse**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/http/HttpServletResponse.html) | | Extends the [ServletResponse](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/ServletResponse.html" \o "interface in javax.servlet) interface to provide HTTP-specific functionality in sending a response. |
| [**HttpSession**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/http/HttpSession.html) | | 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. |
| **Class Summary** | | |
| [**Cookie**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/http/Cookie.html) | Creates 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. | |
| [**HttpServlet**](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/http/HttpServlet.html) | Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. | |

**Methods of HttpServlet Class**

1. **public void service(ServletRequest req,ServletResponse res):** This method send the request to the protected service method by converting both the **req** and **res** to http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res):**This method receives the request from the aboveservice method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res):** This method handles GET request, doGet() method we can send specific amount of data. If we use doGet () method data is shown in address bar. We must override doGet () method depending on type of request.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res):**This method handles the POST request. Large amount of data can be sent using doPost() method. The data will not be visible in the address bar.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res):**This method request header part of the GET request without the GET response body.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res):**This method handles the OPTIONS request.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res):**This method is used to put files (i.e) uploading files on the server. If requests is incorrectly formated then it will return HTTP BAD\_REQUEST error.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res):**This method is used for logging and debugging purpose. It can be used for testing the requested message.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res):**This method handles the DELETE request. It deletes the files from the server.
10. **protected long getLastModified(HttpServletRequest req):**This method returns the time when HttpServletRequest was last modified.

**NOTE: refer Complete Reference JAVA 7th Edition**

1. **Difference Between ServletConfig and ServletContext?**

**ServletConfig**

* ServletConfig available in javax.servlet.\*; package
* ServletConfig object is one per servlet class
* Object of ServletConfig will be created during initialization process of the servlet
* This Config object is public to a particular servlet only
* *Scope*: As long as a servlet is executing, ServletConfig object will be available, it will be destroyed once the servlet execution is completed.
* We should give request explicitly, in order to create ServletConfig object for the first time
* In web.xml – <*init-param*> tag will be appear under <*servlet-class*> tag

<servlet>

<servlet-name>TestServlet</servlet-name>

<servlet-class>Login</servlet-class>

<init-param>

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

<param-value>xyz</param-value>

</init-param>

</servlet>

**ServletContext**

* ServletContext available in javax.servlet.\*; package
* ServletContext object is global to entire web application
* Object of ServletContext will be created at the time of web application deployment
* *Scope*: As long as web application is executing, ServletContext object will be available, and it will be destroyed once the application is removed from the server.
* ServletContext object will be available even before giving the first request
* getServletContext () method is used to get the context object.
* In web.xml – <*context-param*> tag will be appear under <*web-app*> tag

<context-param>

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

<param-value>xyz</param-value>

</context-param>

**Note:**

No. of web applications  =  That many number of ServletContext objects [ 1 per web application ]  
No. of servlet classes = That many number of ServletConfig objects

1. **Difference Between GenericServlet and HttpServlet?**

Generic Servlet:

* GenericServlet class is direct subclass of Servlet interface.
* Generic Servlet is protocol independent.It handles all types  of protocol  like http, smtp, ftp etc.
* GenericServlet is an abstract class which implements ***Servlet, ServletConfig and java.io.Serializable***interfaces. and GenericServlet belongs to ***javax.servlet*** package
* Generic Servlet only supports  service() method.It handles only simple request

public void service(ServletRequest req,ServletResponse res ).

* Generic Servlet only supports  service() method.
* GenericServlet implements **ServletConfig** interface and provides way to accept initialization parameter passed to Servlet from web.xml e.g. by using **getInitParamter()**.

HttpServlet:

1. HttpServlet class is the direct subclass of Generic Servlet.
2. HttpServlet is protocol dependent. It handles only http protocol.
3. HttpServlet is an abstract class which extends **GenericServlet** and implements ***java.io.Serializable*** interface
4. HttpServlet belongs to ***javax.servlet.http*** package
5. HttpServlet overrides **service()** method of Generic Servlet and provides callback on **doXXX(HttpServletRequest request, HttpServletResponse)**method whenever it receives HTTP request, it supports **doGet(), doPost(), doPut(), doDelete(), doHead(), doTrace(), doOptions()** methods.
6. HttpServlet  has  two service methods **public void service(ServletRequest req,ServletResponse res )** and **protected void service(HttpServletRequest req,HttpServletResponse res)**All the request first goes to the **public service()** method, which wrap into Http Objects and calls **protected service()** method
7. **use of Deployment Descriptor?**

# The Deployment Descriptor: web.xml

Java web applications use a deployment descriptor file to determine how URLs map to servlets, which URLs require authentication, and other information. This file is named web.xml, and resides in the app's WAR under the WEB-INF/ directory. web.xml is part of the servlet standard for web applications.

**Deployment descriptors**

A web application's deployment descriptor describes the classes, resources and configuration of the application and how the web server uses them to serve web requests. When the web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code that ought to handle the request.

The deployment descriptor is a file named web.xml. It resides in the app's WAR under the WEB-INF/ directory. The file is an XML file whose root element is <web-app>.

To map a URL to a servlet, you declare the servlet with the <servlet> element, then define a mapping from a URL path to a servlet declaration with the <servlet-mapping> element.

The <servlet> element declares the servlet, including a name used to refer to the servlet by other elements in the file, the class to use for the servlet, and initialization parameters. You can declare multiple servlets using the same class with different initialization parameters. The name for each servlet must be unique across the deployment descriptor.

The <servlet-mapping> element specifies a URL pattern and the name of a declared servlet to use for requests whose URL matches the pattern. The URL pattern can use an asterisk (\*) at the beginning or end of the pattern to indicate zero or more of any character.  The pattern matches the full path of the URL, starting with and including the forward slash (/) following the domain name

<web-app>

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

</servlet>

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

</web-app>

Reading Servlet Paramerers Refer Class Notes

Reading Initialization Paramerers Refer week5.2

Handling HttpRequest and responses refer Complete Reference JAVA and Class Notes

**Session Tracking**

# Session Tracking in Servlets

1. [Session Tracking](https://www.javatpoint.com/session-tracking-in-servlets#session1)
2. [Session Tracking Techniques](https://www.javatpoint.com/session-tracking-in-servlets#session1tech)

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

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

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

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**

# Cookies in Servlet

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

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

### How Cookie works

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

### Cookie class

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

### Constructor of Cookie class

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

### Useful Methods of Cookie class

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

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

### Other methods required for using Cookies

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

### How to create Cookie?

Let's see the simple code to create cookie.

1. Cookie ck=**new** Cookie("user","cseb");//creating cookie object
2. response.addCookie(ck);//adding cookie in the response

### How to delete Cookie?

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

1. Cookie ck=**new** Cookie("user","");//deleting value of cookie
2. ck.setMaxAge(0);//changing the maximum age to 0 seconds
3. response.addCookie(ck);//adding cookie in the response

### How to get Cookies?

Let's see the simple code to get all the cookies.

1. Cookie ck[]=request.getCookies();
2. **for**(**int** i=0;i<ck.length;i++){
3. out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
4. }

# 2) Hidden Form Field

1. [Hidden Form Field](https://www.javatpoint.com/hidden-form-field-in-session-tracking)
2. [Example of Hidden Form Field](https://www.javatpoint.com/hidden-form-field-in-session-tracking#session2ex)

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.

1. <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.

# 3)URL Rewriting

1. [URL Rewriting](https://www.javatpoint.com/url-rewriting-in-session-tracking)
2. [Advantage of URL Rewriting](https://www.javatpoint.com/url-rewriting-in-session-tracking#urladv)
3. [Disadvantage of URL Rewriting](https://www.javatpoint.com/url-rewriting-in-session-tracking#urldisadv)
4. [Example of URL Rewriting](https://www.javatpoint.com/url-rewriting-in-session-tracking#urlex)

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.

# 4) HttpSession interface

1. [HttpSession interface](https://www.javatpoint.com/http-session-in-session-tracking)
2. [How to get the HttpSession object](https://www.javatpoint.com/http-session-in-session-tracking#httpsessionhow)
3. [Commonly used methods of HttpSession interface](https://www.javatpoint.com/http-session-in-session-tracking#httpsessionmethod)
4. [Example of using HttpSession](https://www.javatpoint.com/http-session-in-session-tracking#httpsessionex)

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

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

### How to get the HttpSession object ?

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

1. **public HttpSession getSession():**

Returns the current session associated with this request, or if the request does not have a session, creates one.

1. **public HttpSession getSession(boolean create):**

Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

### Commonly used methods of HttpSession interface

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

**Next concept: Introduction to JDBC: refer JDBC PPT**