SERVLET

1. Introduction

Servlet is a Java programming language class, part of Java Enterprise Edition (Java EE). Sun Microsystems developed its first version 1.0 in the year 1997. Its current Version is Servlet 3.1.

Servlets are used for creating dynamic web applications in java by extending the capability of a server. It can run on any web server integrated with a Servlet container.

### **1.1 Java Servlet Process**

The process of a servlet is shown below:

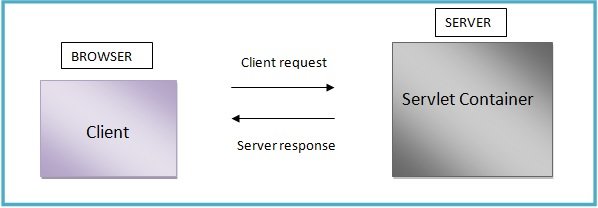


Figure 1: servlet processing of user requests

* A Request is sent by a client to a servlet container. The container acts as a Web server.
* The Web server searches for the servlet and initiates it.
* The client request is processed by the servlet and it sends the response back to the server.
* The Server response is then forwarded to the client.

### **1.2 Merits**

* Servlets are platform independent as they can run on any platform.
* The Servlet API inherits all the features of the Java platform.
* It builds and modifies the security logic for server-side extensions.
* Servlets inherit the security provided by the Web Server.
* In Servlet, only a single instance of the requests runs concurrently. It does not run in a separate process. So, it saves the memory by removing the overhead of creating a new process for each request.

## 2. Life Cycle

Servlet lifecycle describes how the servlet container manages the servlet object.

* Load Servlet Class
* Servlet Instance is created by the web container when the servlet class is loaded
* init():This is called only once when the servlet is created. There is no need to call it again and again for multiple requests.

|  |  |
| --- | --- |
| 1  2  3 | public void init() throws ServletException {     } |

* service(): It is called by the web container to handle request from clients. Here the actual functioning of the code is done. The web container calls this method each time when request for the servlet is received.

It calls doGet(), doPost(), doTrace(), doPut(), doDelete() and other methods

* doGet():

|  |  |
| --- | --- |
| 1  2  3  4 | public void doGet(HttpServletRequest request,HttpServletResponse response)      throws ServletException, IOException {  // code  } |

* doPost():

|  |  |
| --- | --- |
| 1  2  3  4 | public void doPost(HttpServletRequest request, HttpServletResponse response)      throws ServletException, IOException {      // code  } |

* destroy(): It is used to clean resources and called before removing the servlet instance.

|  |  |
| --- | --- |
| 1 | public void destroy() |

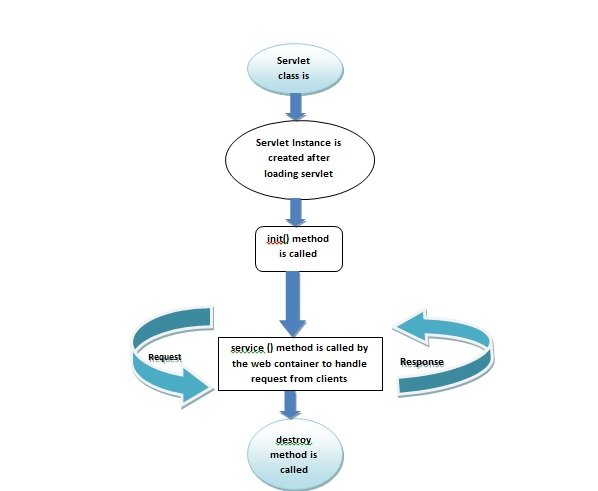


Figure 2: Servlet Life Cycle

## 3. Container

It is known as servlet engine which manages Java Servlet components on top of a web server to the request send by the client.

### **3.1 Services**

Servlet Container provides the following services:

* It manages the servlet life cycle.
* The resources like servlets, JSP pages and HTML files are managed by servlet container.
* It appends session ID to the URL path to maintain session.
* Provides security service.
* It loads a servlet class from network services, file systems like remote file system and local file system.

### **3.2 Servlet Container Configurations**

The servlet container can be configured with the web server to manage servlets in three ways listed below:

* Standalone container
* In-process container
* Out-process container

**Standalone container**: In this type the Web Server functionality is taken by the Servlet container. Here, the container is strongly coupled with the Web server.

**In-Process container**: In this the container runs within the Web server process.

## 5. Filter

Filters transform the content of requests, responses, and header information from one format to another. These are reusable codes.

* Filter class is declared in the deployment descriptor.
* It is used to write reusable components.
* The request is process before it is called using filters.
* It can be used under a web application for some tasks like:
  + Validation
  + Compression
  + Verification
  + Internationalization

### **5.1 Interface**

It consists of these 3 filters:

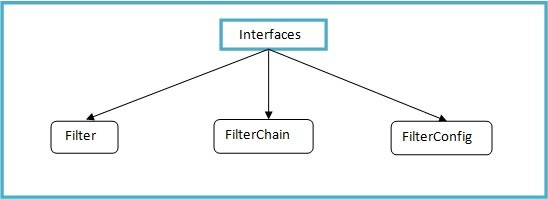


Figure 11: Filter API Interfaces

**Filter**

This is the initial and basic interface which all filter class should implement. Java.servlet.Filter interface has the following methods:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| init(FilterConfig) | This method initializes a filter |
| doFilter(ServletRequest, ServletResponse, FilterChain) | This method encapsulates the service logic on ServletRequest to generate ServletResponse. FilterChain is to forward request/response pair to the next filter. |
| destroy() | It destroys the instance of the filter class. |

**FilterConfig**

Its object is used when the filters are initialized. Deployment descriptor (web.xml) consists of configuration information. The object of FilterConfig interface is used to fetch configuration information about filter specified in web.xml. Its methods are mentioned below:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| getFilterName() | It returns the name of filter in web.xml |
| getInitParameter(String) | It returns specified initialization parameter’s value from web.xml |
| getInitParameterNames() | It returns enumeration of all initialization parameters of filter. |
| getServletContext() | It returns ServletContext object. |

**FilterChain**

It stores information about more than 1 filter (chain). All filters in this chain should be applied on request before processing of a request.

### **5.2 Example**

Listing 4: web.xml

The Filter NewFilter can be applied to every servlet as /\* is specified here for URL-pattern.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20 | <?xml version="1.0" encoding="UTF-8"?>  <web-app version="3.1" xmlns="<http://xmlns.jcp.org/xml/ns/javaee>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:schemaLocation="<http://xmlns.jcp.org/xml/ns/javaee> <http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd>">  <filter>  <filter-name>NewFilter</filter-name>  <filter-class>NewFilter</filter-class>    <init-param>        <param-name>newParam</param-name>        <param-value>valueOne</param-value>    </init-param>  </filter>  <filter-mapping>       <filter-name>NewFilter</filter-name>       <url-pattern>/\*</url-pattern>  </filter-mapping>  <session-config>      <session-timeout>              30     </session-timeout>  </session-config>  </web-app> |

Listing 5: NewFilter.java

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | import java.io.\*;  import javax.servlet.\*;  import javax.servlet.http.\*;  import java.util.\*;    public class NewFilter implements Filter {        public void init(FilterConfigfilterConfig) {          // init parameter          String value = filterConfig.getInitParameter("newParam");            // displaying init parameter value          System.out.println("The Parameter value: " + value);      }        public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)              throws IOException, ServletException {            // IP address of the client machine.          String remoteAddress = request.getRemoteAddr();            // Returns the remote address          System.out.println("Remote Internet Protocl Address: "+ remoteAddress);            chain.doFilter(request,response);      }        public void destroy( ){        }  } |

Figure 17: Showing console output

## 6. Session

It is a collection of HTTP requests between client and server. The session is destroyed when it expires and its resources are back to the servlet engine.

### **6.1 Session Handling**

It is a means to keep track of session data. This represents the data transferred in a session. It is used when session data from one session may be required by a web server for completing tasks in same or different sessions. Session handling is also known assession tracking.

### **6.2 Mechanisms of Session Handling**

There are four mechanisms for session handling:

**URL rewriting**: The session data required in the next request is appended to the URL path used by the client to make the next request.

·        Query String: A string appended after the requested URI is query string. The string is appended with separator as ‘?’ character.

Example 1): http://localhost:8080/newproject/login?user=test&passwd=abcde

·        Path Info: It is the part of the request URI. Session data can be added to the path info part of the request URI.

Example 2): http://localhost:8080/newproject/myweb/login;user=test&passwd=abcde

**Hidden form field**:  A type of HTML form field which remains hidden in the view. Some other form fields are: textbox, password etc. This approach can be used with form-based requests. It is just used for hiding user data from other different types of users.

Example 3: <input type="hidden" username="name" value="nameOne"/>

**Cookies**: It is a file containing the information that is sent to a client by a server. Cookies are saved at the client side after being transmitted to clients (from server)through the HTTP response header.

Cookies are considered best when we want to reduce the network traffic. Its attributes are name, value, domain, version number, path, and comment. The package javax.servlet.http consists of a class names Cookie.

Some methods in javax.servlet.http.Cookie class are listed below:

* setValue (String)
* getValue()
* getName()
* setComment(String)
* getComment()
* setVersion(String)
* getVersion()
* setDomain(String)
* setPath(String)
* getPath()
* setSecure(boolean)
* getSecure(boolean)

**HTTP session**: It provides asession management service implemented through HttpSession object.

Some HttpSession object methods are listed here; this is referred from the official [Oracle Documentation](https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSession.html):

|  |  |
| --- | --- |
| **Method** | **Description** |
| public Object getAttribute(String name) | It returns the object bound with the specified name in this session or null if no object is bound under the name. |
| public Enumeration getAttributeNames() | It returns Enumeration of String objects containing the names of all the objects bound to this session. |
| public String getId() | It returns a string containing the unique identifier assigned to this session. |
| public long getCreationTime() | It returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT. |
| public long getLastAccessedTime() | It returns the last time the client sent a request associated with this session. |
| public int getMaxInactiveInterval() | It returns the maximum time interval, in seconds that the servlet container will keep this session open between client accesses. |
| public void invalidate() | It Invalidates this session then unbinds any objects bound to it. |
| public boolean isNew() | It returns true if the client does not yet know about the session or if the client chooses not to join the session. |

### **6.3 Example**

Session Information like session id, session creation time, last accessed time and others is printed under this example.

Listing 6: ServletSession.java

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | import java.io.IOException;  import java.io.PrintWriter;  import java.util.Date;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import javax.servlet.http.HttpSession;    public class ServletSession extends HttpServlet {    @Override   protected void doGet(HttpServletRequest request, HttpServletResponse response)              throws ServletException, IOException {             // session object creation             HttpSessionnewSession = request.getSession(true);        // Session creation time.        Date cTime = new Date(newSession.getCreationTime());        // The last time the client sent a request.        Date lTime = new Date( newSession.getLastAccessedTime());          /\* sets the time, in seconds, between client requests before the servlet container        invalidates this session \*/        newSession.setMaxInactiveInterval(1 \* 60 \* 60);        String str = "Website | Session";          response.setContentType("text/html");        PrintWriter out = response.getWriter();          String document =        "<!doctype html public \"-//w3c//dtd html 4.0 " +        "transitional//en\">\n";        out.println(document +                  "<html>\n" +                  "<head><title>" + str + "</title></head>\n" +                  "<body bgcolor=\"#bbf5f0\">\n" +                   "<h2>Website: Displaying Session Information</h2>\n" +                  "<table border=\"2\">\n" +                  "<tr>\n" +                  "  <td>Unique identifier assigned to this session</td>\n" +                  "  <td>" + newSession.getId() + "</td>"                + "</tr>\n" +                  "<tr>\n" +                  "  <td>The time when this session was created</td>\n" +                  "  <td>" + cTime +                  "  </td>"                + "</tr>\n" +                  "<tr>\n" +                  "  <td>The last time the client sent a request associated with this session</td>\n"                + "  <td>" + lTime +                  "  </td>"                + "</tr>\n" +                  "</tr>\n" +                  "<tr>\n" +                  "  <td> the maximum time interval, in seconds that the servlet container will keep this session open between client accesses.</td>\n" +                  "  <td>" + newSession.getMaxInactiveInterval() +                  "  </td>"                + "</tr>\n" +                       "</table>\n" +                  "</body></html>");    }  } |

Figure 18: Displaying output

## 7. Exception Handling

Exceptions are used to handle errors. It is a reaction to unbearable conditions. Here comes the role of web.xml i.e. deployment description which is used to run JSP and servlet pages. The container searches the configurations in web.xml for a match. So, in web.xml use these exception-type elements for match with the thrown exception type when a servlet throws an exception.

### **7.1 Error Code Configuration**

The /HandlerClass servlet gets called when an error with status code 403 occurs as shown below:

Listing 7: For Error code 403

|  |  |
| --- | --- |
| 1  2  3  4 | <error-page>    <error-code>403</error-code>    <location>/HandlerClass</location>  </error-page> |

### **7.2 Exception-Type Configuration**

If the application throws IOException, then /HandlerClass servlet gets called by the container:

Listing 8: For Exception Type IOException

|  |  |
| --- | --- |
| 1  2  3  4 | <error-page>    <exception-type>java.io.IOException</exception-type >    <location>/HandlerClass</location>  </error-page> |

If you want to avoid the overhead of adding separate elements, then use java.lang.Throwable as exception-type:

Listing 9: For all exceptions mention java.lang.Throwable:

|  |  |
| --- | --- |
| 1  2  3  4 | <error-page>    <exception-type>java.lang.Throwable</exception-type >    <location>/HandlerClass</location>  </error-page> |

## 8. Debugging

Client-server interactions are in large number in Servlets. This makes errors difficult to locate. Different ways can be followed for location warnings and errors.

### **8.1 Message Logging**

Logs are provided for getting information about warning and error messages. For this a standard logging method is used. Servlet API can generate this information using log() method. Using Apache Tomcat, these logs can be found in TomcatDirectory/logs.

### **8.2 Java Debugger**

Servlets can be debugged using JDB Debugger i.e. Java Debugger. In this the program being debugged is sun.servlet.http.HttpServer. Set debugger’s class path for finding the following classes:

* servlet.http.HttpServer
* server\_root/servlets and server\_root/classes: Through this the debugger sets breakpoints in a servlet.

### **8.3 Headers**

Users should have some information related to structure of HTTP headers. Issues can be judged using them which can further locate some unknown errors. Information related to HTTP headers can help you in locating errors. Studying request and response can help in guessing what is not going well.

### **8.4 Refresh**

Refresh your browser’s web page to avoid it from caching previous request. At some stages, browser shows request performed previously. This is a known point but can be a problem for those who are working correctly but unable to display the result properly.

Listing 21: ServletDebugging.java

Here, Servlet Debugging is shown which displays the errors in Tomcat log.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | import java.io.IOException;  import java.io.PrintWriter;  import javax.servlet.ServletContext;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    public class ServletDebugging extends HttpServlet {        @Override      protected void doGet(HttpServletRequest request, HttpServletResponse response)              throws ServletException, IOException {            // parameter "name"          String strpm = request.getParameter("name");            ServletContext context = getServletContext( );            // checks if the parameter is set or not          if (strpm == null || strpm.equals(""))          context.log("No message received:", new IllegalStateException("Sorry, the          parameter is missing."));          else          context.log("Here is the visitor's message: " +strpm);        }  } |

Figure 19: Output as visible in Apache Tomcat log

## 9. Internationalization

For building a global website, some important points are considered which includes language related to user’s nationality. Internationalization isenabling a website for providing content translated in different languages according to user’s nationality.

### **9.1 Methods**

For finding visitors local region and language, these methods are used:

|  |  |
| --- | --- |
| **Method** | **Description** |
| String getCountry() | Returns the country code. |
| String getDisplayCountry() | Returns a name for the visitors’ country. |
| String getLanguage() | Returns the language code. |
| String getDisplayLanguage() | Returns a name for the visitors’ language. |
| String getISO3Country() | Returns a three-letter abbreviation for the visitors country. |
| String getISO3Language() | Returns a three-letter abbreviation for the visitors language. |

### **9.2 Example**

The example displays the current locale of a user. Following project is created in NetBeansIDE:

|  |  |
| --- | --- |
| 1  2  3  4 | Project Name: WebApplicationInternationalization  Project Location: C:\Users\Test\Documents\NetBeansProjects  Servlet: ServletLocale  URL Pattern: /ServletLocale |

Listing 22: ServletLocale.java

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | import java.io.IOException;  import java.io.PrintWriter;  import java.util.Locale;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    public class ServletLocale extends HttpServlet {    @Override      protected void doGet(HttpServletRequest request, HttpServletResponse response)              throws ServletException, IOException {             //Get the client's Locale        Locale newloc = request.getLocale();        String country = newloc.getCountry();          // Set response content type        response.setContentType("text/html");        PrintWriter out = response.getWriter();          // this sets the page title and body content        String title = "Finding Locale of current user";        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=\"#C0C0C0\">\n" + "<h3>" + country + "</h3>\n" +          "</body></html>");      }  } |

Listing23: index.html with location hyperlink as URL-pattern – ServletLocale

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11 | <html>  <head>  <title>User's Location</title>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  </head>  <body>      <p>Click on the following link for finding the locale of visitor:</p>      <a href="ServletLocale"><b>Location</b></a>  </body>  </html> |

Listing24: web.xml with URL-pattern as /ServletLocale

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | <?xml version="1.0" encoding="UTF-8"?>  <web-app version="3.1" xmlns="<http://xmlns.jcp.org/xml/ns/javaee>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:schemaLocation="<http://xmlns.jcp.org/xml/ns/javaee> <http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd>">  <servlet>      <servlet-name>ServletLocale</servlet-name>      <servlet-class>ServletLocale</servlet-class>  </servlet>  <servlet-mapping>       <servlet-name>ServletLocale</servlet-name>       <url-pattern>/ServletLocale</url-pattern>  </servlet-mapping>  <session-config>  <session-timeout>              30  </session-timeout>  </session-config>  </web-app> |