

JAVA SERVLET TUTORIAL

THE ULTIMATE GUIDE



JAVA™

KAUSHIK PAL



Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

Java Servlet Tutorial

Contents

1	Introduction	1
1.1	Servlet Process	1
1.2	Merits	1
2	Life Cycle	3
3	Container	5
3.1	Services	5
3.2	Servlet Container Configurations	5
4	Demo: To start with	6
5	Filter	12
5.1	Interface	12
5.2	Example	13
6	Session	19
6.1	Session Handling	19
6.2	Mechanisms of Session Handling	19
6.3	Example	20
7	Exception Handling	23
7.1	Error Code Configuration	23
7.2	Exception-Type Configuration	23
8	Debugging	24
8.1	Message Logging	24
8.2	Java Debugger	24
8.3	Headers	24
8.4	Refresh	24
9	Internationalization	26
9.1	Methods	26
9.2	Example	26

10 Reference	29
11 Conclusion	30
12 Download	31

Copyright (c) Exelixis Media P.C., 2015

All rights reserved. Without limiting the rights under copyright reserved above, no part of this publication may be reproduced, stored or introduced into a retrieval system, or transmitted, in any form or by any means (electronic, mechanical, photocopying, recording or otherwise), without the prior written permission of the copyright owner.

Preface

Java Servlets is a Java based web technology. Java Servlet technology provides Web developers with a simple, consistent mechanism for extending the functionality of a Web server and for accessing existing business systems.

A servlet can almost be thought of as an applet that runs on the server side-without a face. Java servlets make many Web applications possible.

Java Servlets comprise a fundamental part of the Java Enterprise Edition (Java EE). Please note that Java Servlets have to be executed inside a Servlet compatible “Servlet Container” (e.g. web server) in order to work.

This tutorial works as a comprehensive, kick-start guide for your Java Servlet based code.

About the Author

Kaushik has 16 years of experience as a technical architect and software consultant in enterprise application and product development. He has interest in new technology and innovation area along with technical writing. His main focus is on web architecture, web technologies, java/j2ee, Open source, big data and semantic technologies.

He has demonstrated his expertise in requirement analysis, architecture design & implementation, technical use case preparation, and software development. His experience has spanned in different domains like Insurance, banking, airlines, shipping, document management etc.

Kaushik worked with a wide variety of technologies starting from Mainframe (IBM S/390), midrange (AS/400), web technologies and open source area. He has worked with clients like IBM, Lexmark, United Airlines and many more.

Chapter 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 Servlet Process

The process of a servlet is shown below:

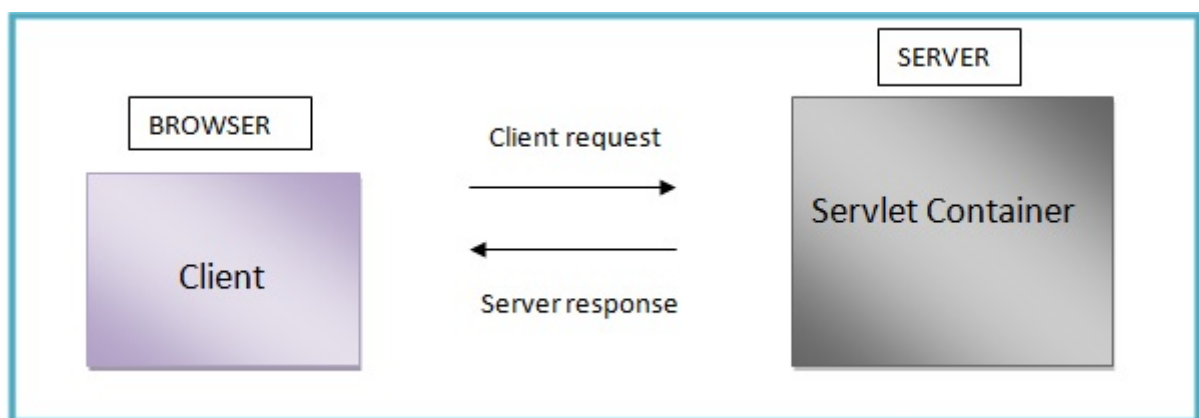


Figure 1.1: servlet processing of user requests

- 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.

Chapter 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.

```
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()`:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {  
    // code  
}
```

- `doPost()`:

```
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.

```
public void destroy()
```

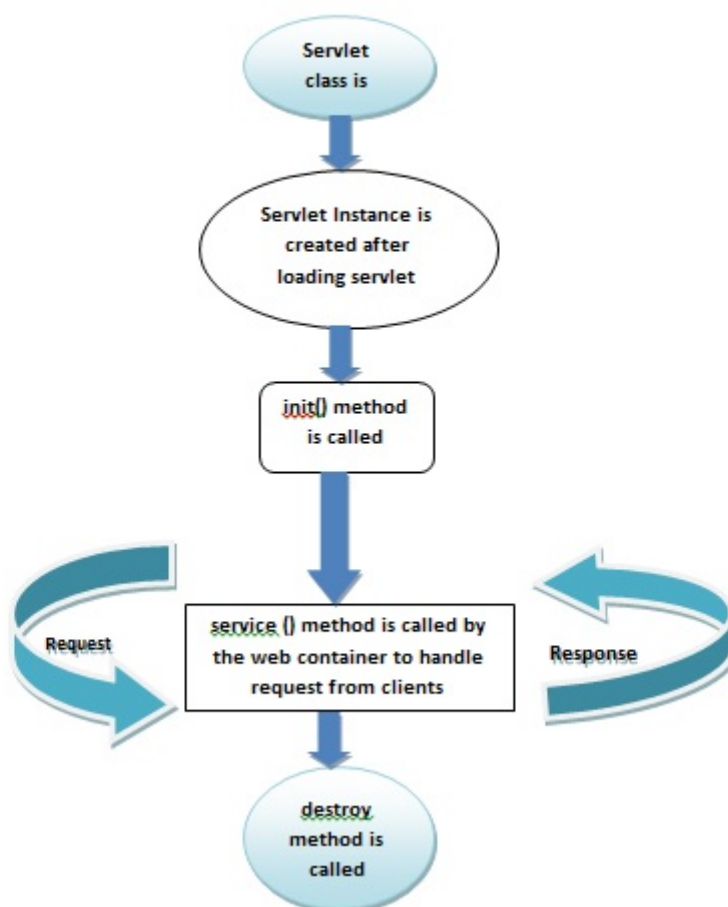


Figure 2.1: Servlet Life Cycle

Chapter 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.

Out-Process container: In this type there is a need to configure the servlet container to run outside the Web server process. It is used in some cases like if there is a need to run Servlets and Servlet container in different process/systems.

Chapter 4

Demo: To start with

Here is an example showing Demo Servlet. Follow these steps to start with your first Servlet Application in NetBeansIDE.

Step 1: Open *NetBeansIDE* → *File* → *New Project* → *WebApplication* → Set Project name as *WebApplicationServletDemo*

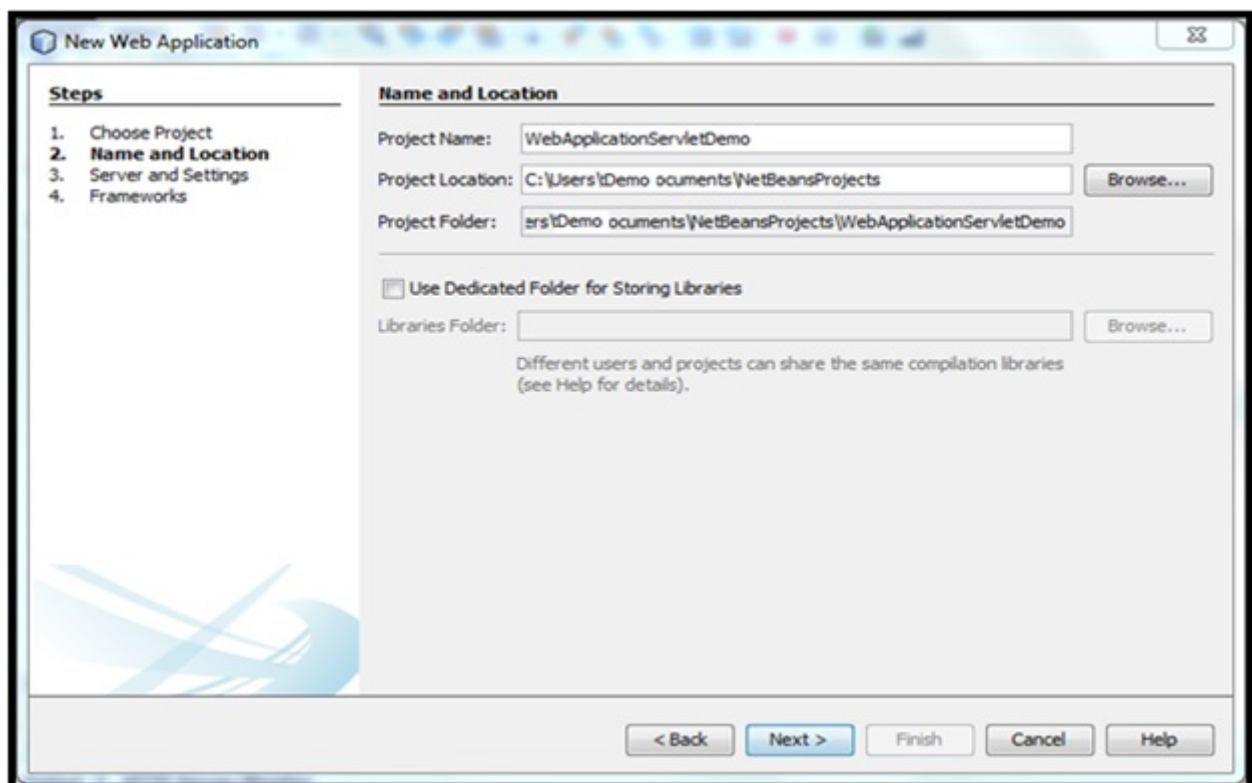


Figure 4.1: Create new WebApplication project in NetBeansIDE: WebApplicationServletDemo

Step 2: Now click on Next > as shown above. This will create new project with the following directory structure.

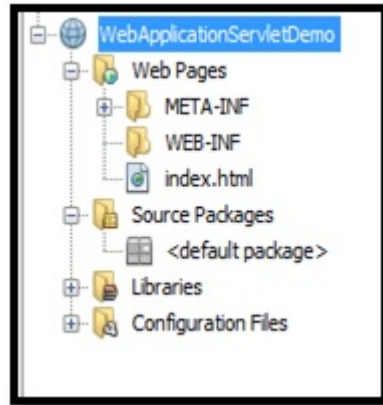


Figure 4.2: Project Directory after creating new project

Step 3: Create new servlet application by Right Clicking on *Project Directory* → *New* → *Servlet*

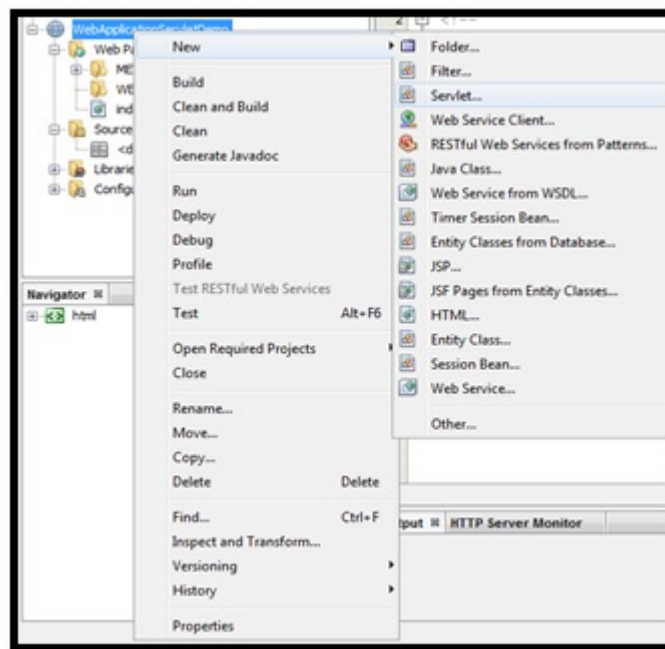


Figure 4.3: Adding Servlet file

Step 4: Add the Servlet Class Name as "ServletDemo" and click on *Next*.

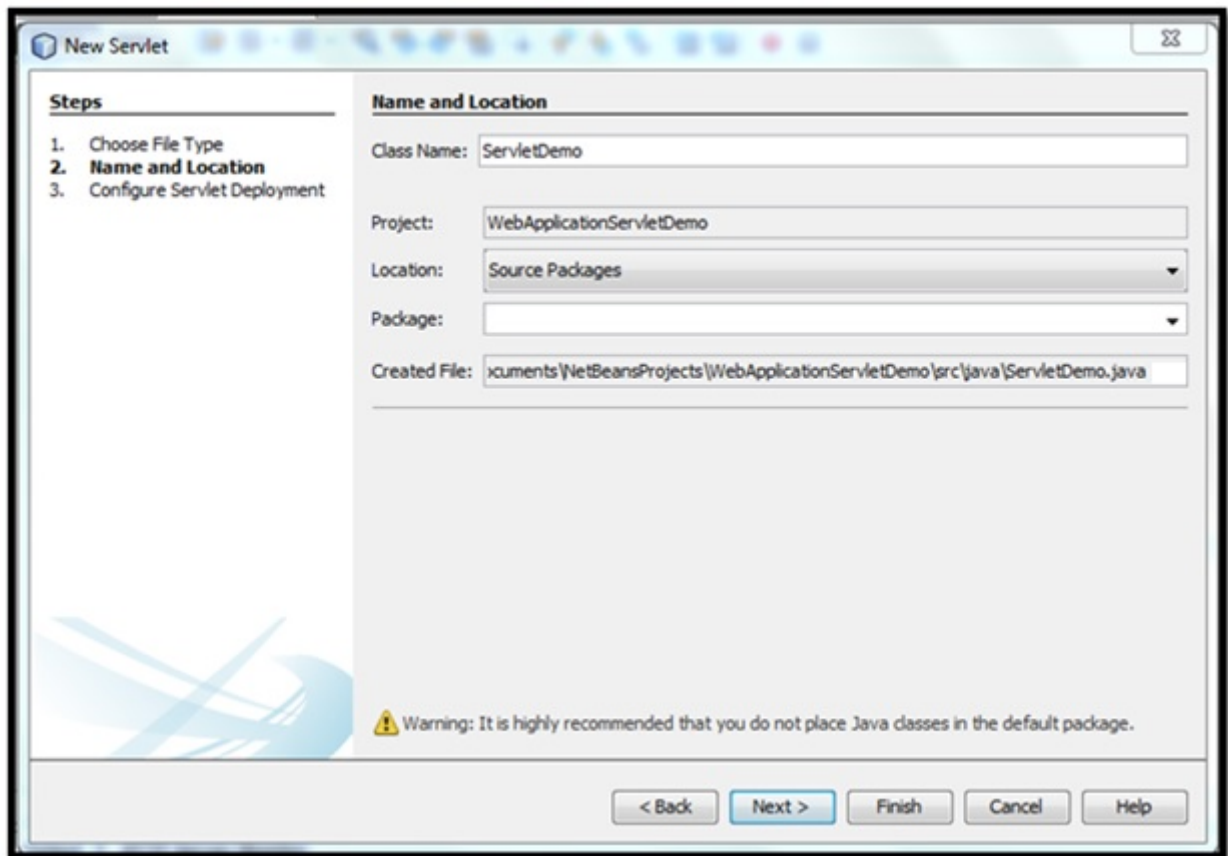


Figure 4.4: Adding Servlet Class Name

Step 5: Now, Configure Servlet Deployment by checking "Add information to deployment descriptor (web.xml)" and adding URL Pattern (the link visible) as *ServletDemo*. This step will generate *web.xml* file in *WEB-INF* folder.

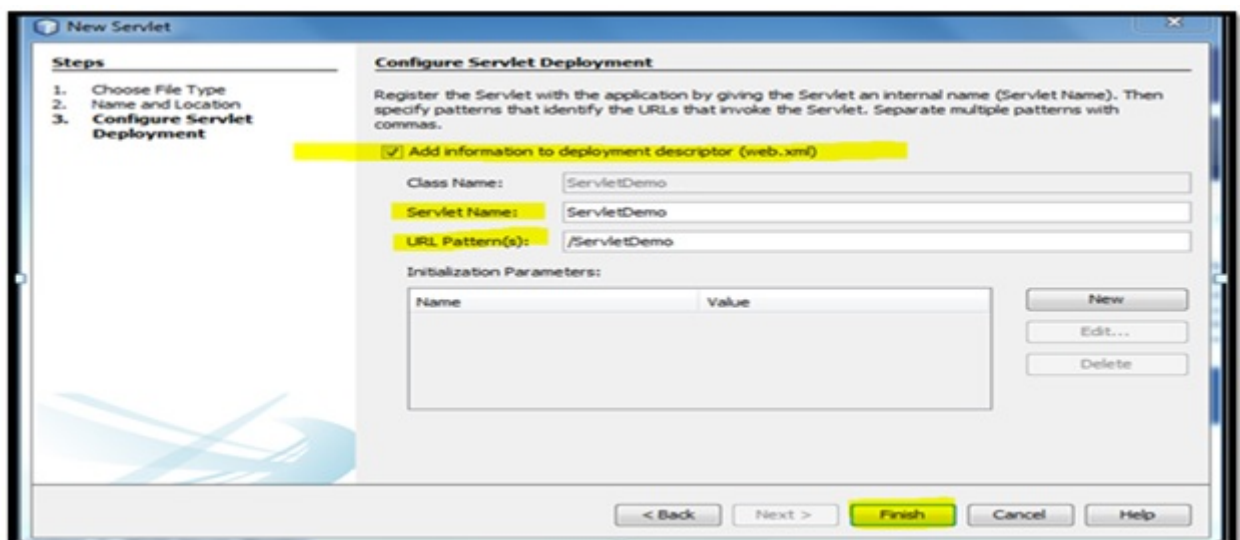


Figure 4.5: Configuring Servlet Deployment

Step 6: Click on Finish as shown above, this will add ServletDemo.java servlet under project directory. Check the changes under

Directory Structure:

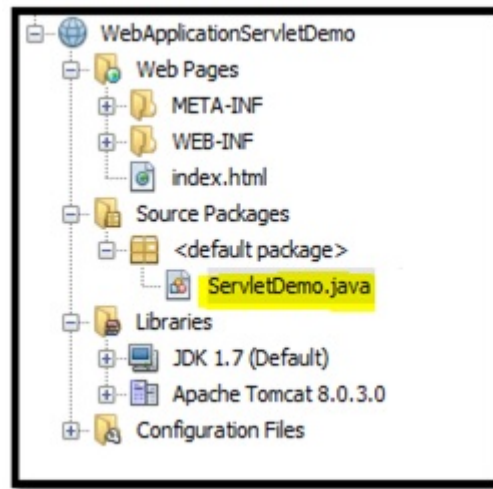


Figure 4.6: Changes under project directory after configuring

Here is the code for deployment descriptor (web.xml) with URL-pattern as */ServletDemo*:

Listing 1: web.xml

```
<?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>ServletDemo</servlet-name>
    <servlet-class>ServletDemo</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServletDemo</servlet-name>
    <url-pattern>/ServletDemo</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
</web-app>
```

Here,

```
<servlet-name>: name given to Servlet
<servlet-class>: servlet class
<servlet-mapping>: maps internal name to URL
<url-pattern>: link displays when Servlet runs
```

The hyperlink *Next* is mentioned as *ServletDemo*. So, when the user will click on it, the page will redirect to *ServletDemo* servlet whose url-pattern is mentioned as *ServetDemo*:

Listing 2: index.html

```
<html>
<head>
<title>Welcome</title>
<meta charset="UTF-8">
```



```

<meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
<h2>Welcome</h2>
We're still under development stage. Stay Tuned for our website's new design and learning ↵
    content.
<a href="ServletDemo"><b>Next</b></a>
</body>
</html>

```

Listing 3: ServletDemo.java

```

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ServletDemo extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse ↵
        response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet ServletDemo</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet ServletDemo at " + request.getContextPath ↵
                () + "</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            /* TODO output your page here. You may use following sample code. ↵
             */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlets</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<br /><h2>First Demo Servlet application</h2><br /> ↵
                Here, the URL-pattern is ServletDemo in web.xml. So, the address ↵
                is <i>WebApplicationServletDemo/ServletDemo</i>.");
            out.println("<br /><br /><a href=\"index.html\">Previous Page</a>") ↵
                ;
            out.println("</body>");
            out.println("</html>");
        }
        finally

```

```
    {  
        out.close();  
    }  
}
```



Figure 4.7: Output showing index.html welcome page

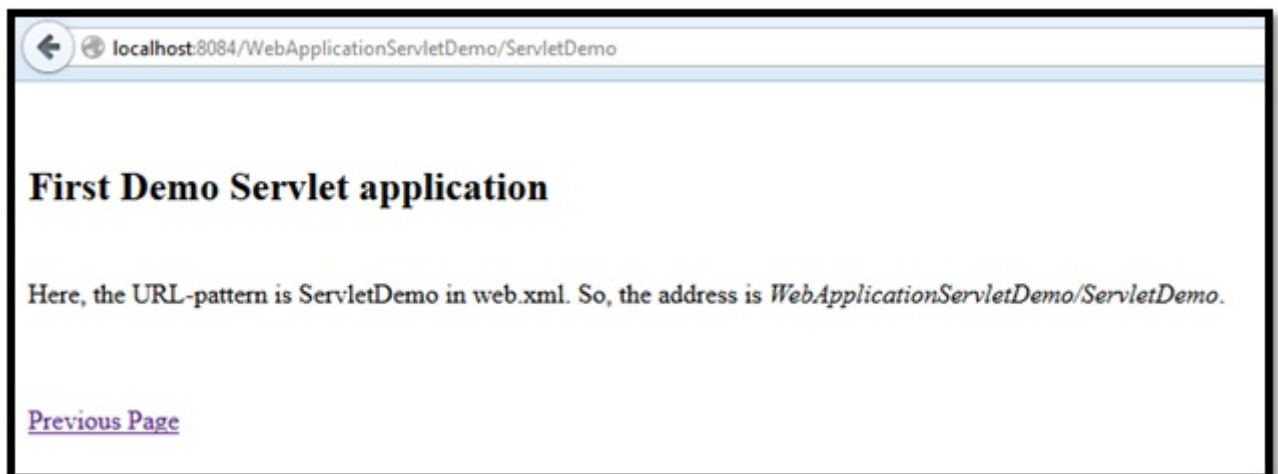


Figure 4.8: Output showing redirection to ServletDemo.java

Chapter 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:

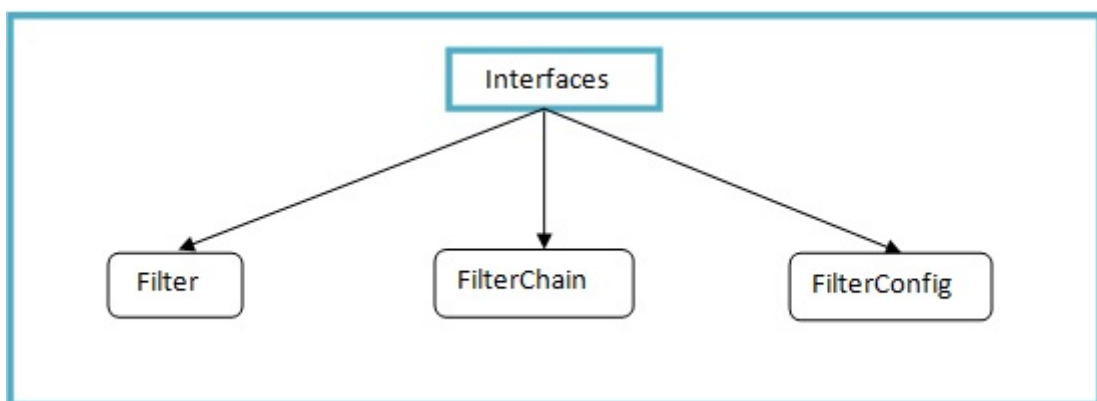


Figure 5.1: Filter API Interfaces

Filter

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

Methods	Description
<code>init(FilterConfig)</code>	This method initializes a filter
<code>doFilter(ServletRequest, ServletResponse, FilterChain)</code>	This method encapsulates the service logic on <code>ServletRequest</code> to generate <code>ServletResponse</code> . <code>FilterChain</code> is to forward request/response pair to the next filter.
<code>destroy()</code>	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
<code>getFilterName()</code>	It returns the name of filter in <code>web.xml</code>
<code>getInitParameter(String)</code>	It returns specified initialization parameter's value from <code>web.xml</code>
<code>getInitParameterNames()</code>	It returns enumeration of all initialization parameters of filter.
<code>getServletContext()</code>	It returns <code>ServletContext</code> 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

This is an example showing filters application in NetBeansIDE. Create a WebApplication project *WebApplicationFilterDemo* in the same ways as shown under *Demo* section. New Filter can be added in the web application by Right Clicking on *Project Directory* → *New* → *Filter*

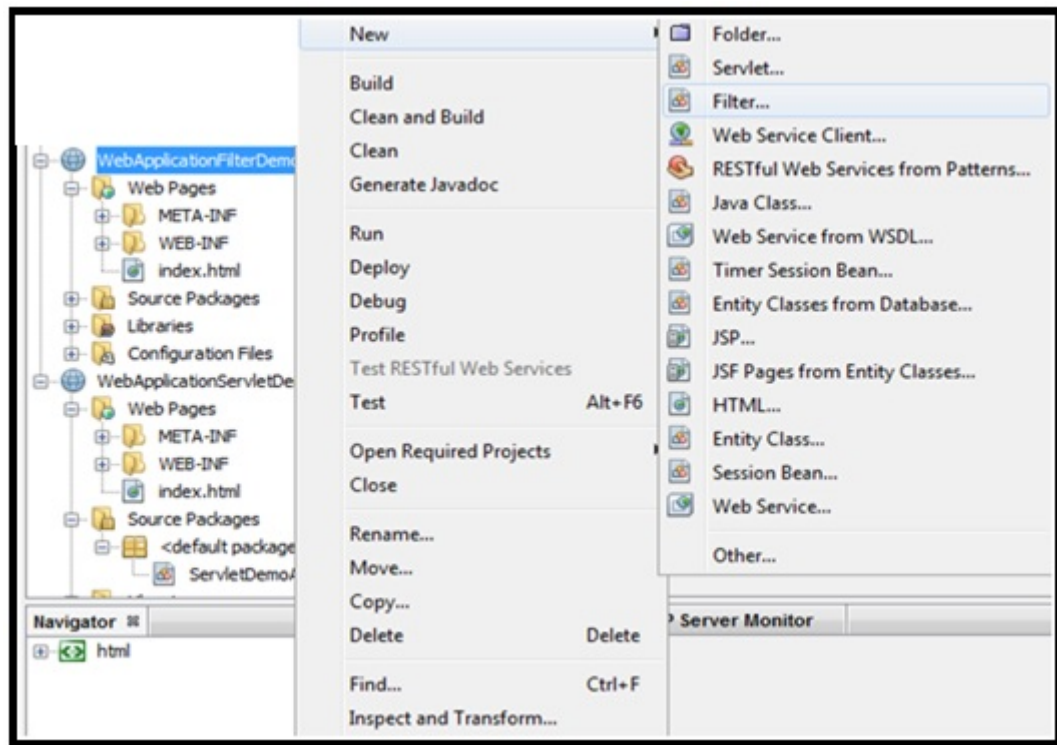


Figure 5.2: Add new Filter to web application

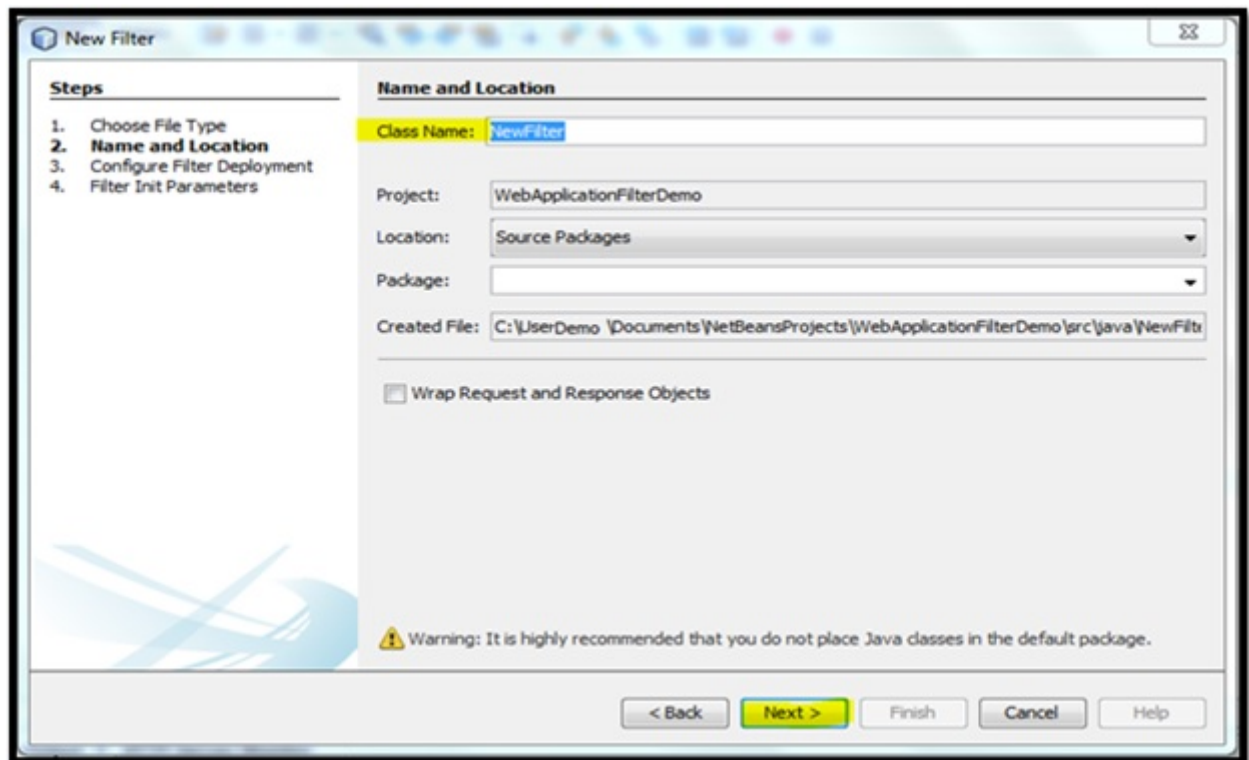


Figure 5.3: Add Class Name as NewFilter and click on Next

Configure Filter Deployment by checking "Add information to deployment descriptor (web.xml)". Now, the *Next* button is disabled here due to an error highlighted in Figure 13. The error "Enter at least one URL pattern" can be solved by clicking on "New".

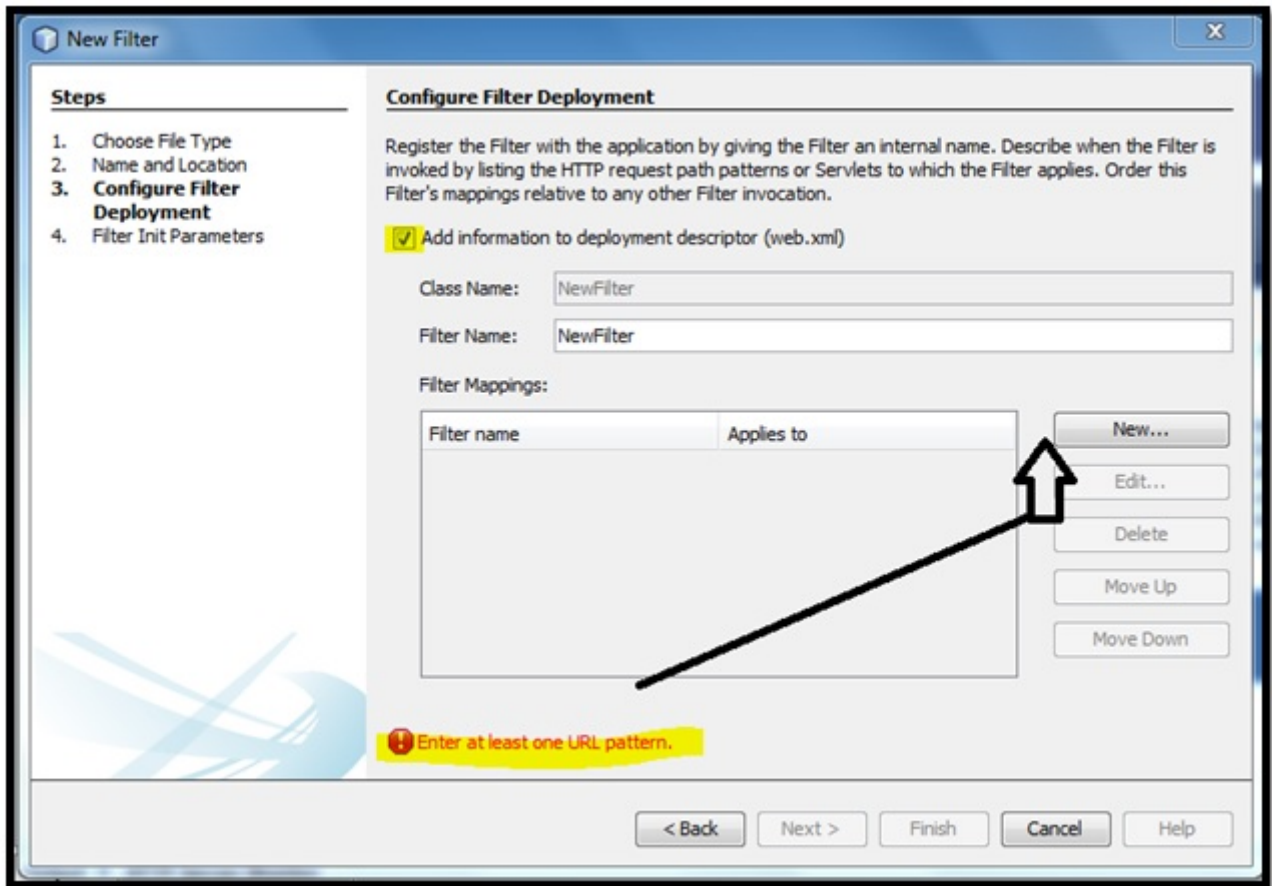


Figure 5.4: Configure Filter Deployment by checking "Add information to deployment descriptor(web.xml)"

Now, filter is mapped by adding URL-pattern as shown in Figure 15.

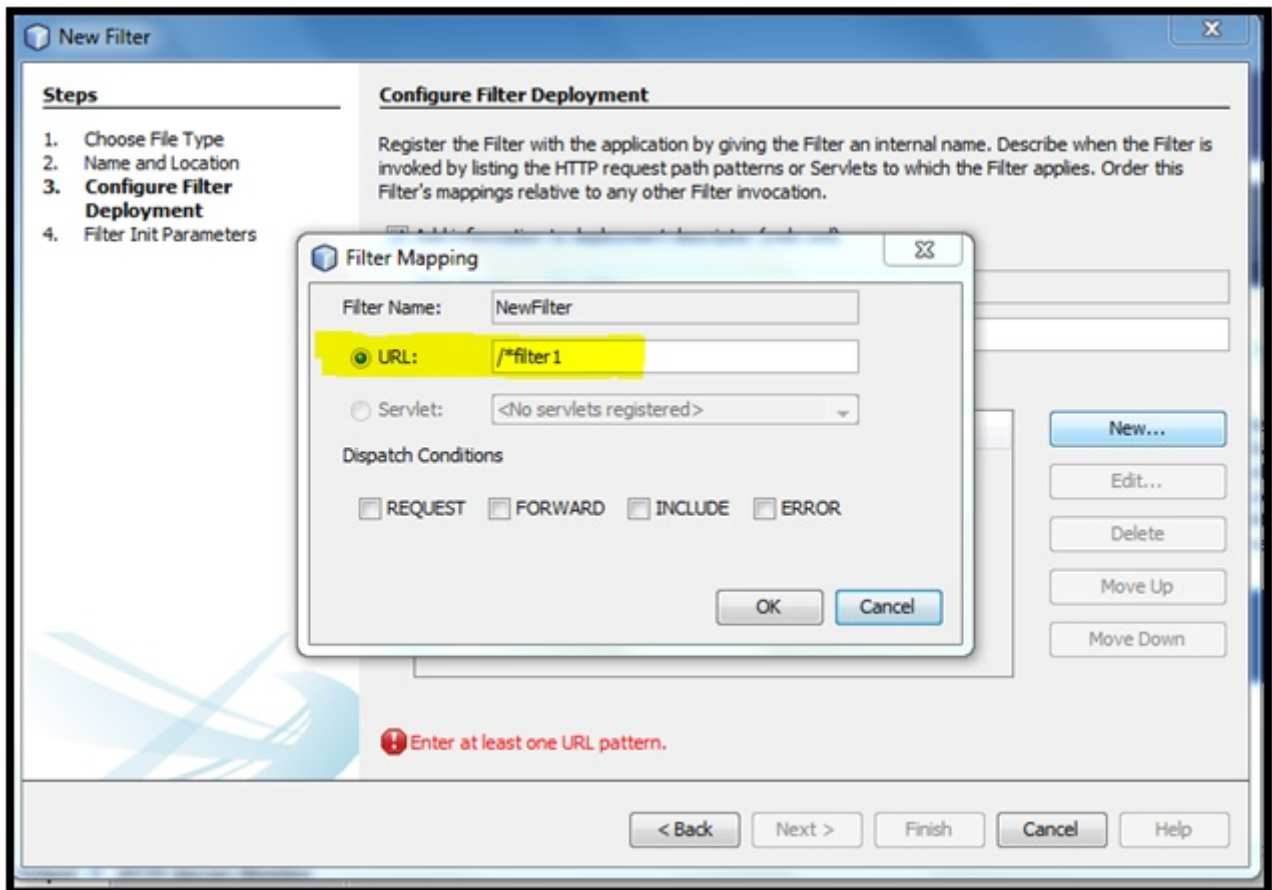


Figure 5.5: Filter mapping

After adding new filter and clicking on OK, the error will get resolved. Now, add *init-parameter* with name and value. Then click *Finish*.

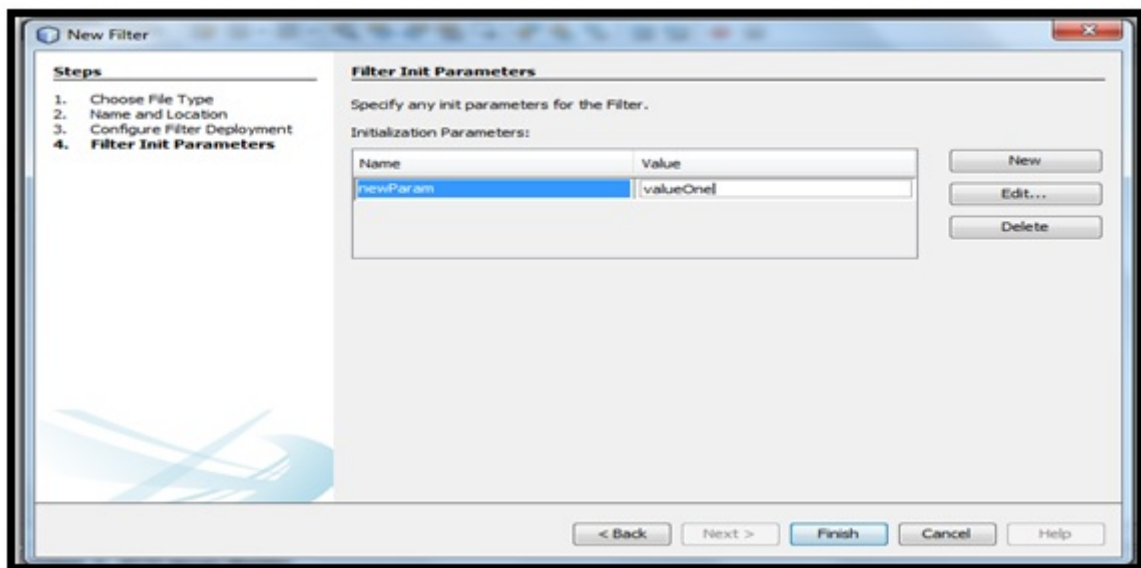


Figure 5.6: Adding init-parameter

Listing 4: web.xml

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

```
<?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

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class NewFilter implements Filter {

    public void init(FilterConfig filterConfig) {
        // 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 Protocol Address: " + remoteAddress);

        chain.doFilter(request, response);
    }

    public void destroy() {
    }
}
```



```
[13-Nov-2014 14:43:16.441 INFO [http-nio-8084-exec-48] org.apache.catalina.core.StandardC  
ontext.reload Reloading Context with name [/WebApplicationFilterDemo] is completed  
The Parameter value: valueOne  
Remote Internet Protocol Address: 0:0:0:0:0:0:0:1
```

Figure 5.7: Showing console output

Chapter 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 as session 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 named `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 a session management service implemented through `HttpSession` object.

Some `HttpSession` object methods are listed here; this is referred from the official [Oracle Documentation](#):

Method	Description
<code>public Object getAttribute (String name)</code>	It returns the object bound with the specified name in this session or null if no object is bound under the name.
<code>public Enumeration getAttributeNames ()</code>	It returns Enumeration of String objects containing the names of all the objects bound to this session.
<code>public String getId ()</code>	It returns a string containing the unique identifier assigned to this session.
<code>public long getCreationTime ()</code>	It returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
<code>public long getLastAccessedTime ()</code>	It returns the last time the client sent a request associated with this session.
<code>public int getMaxInactiveInterval ()</code>	It returns the maximum time interval, in seconds that the servlet container will keep this session open between client accesses.
<code>public void invalidate ()</code>	It Invalidates this session then unbinds any objects bound to it.
<code>public boolean isNew ()</code>	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`

```
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)
```

```

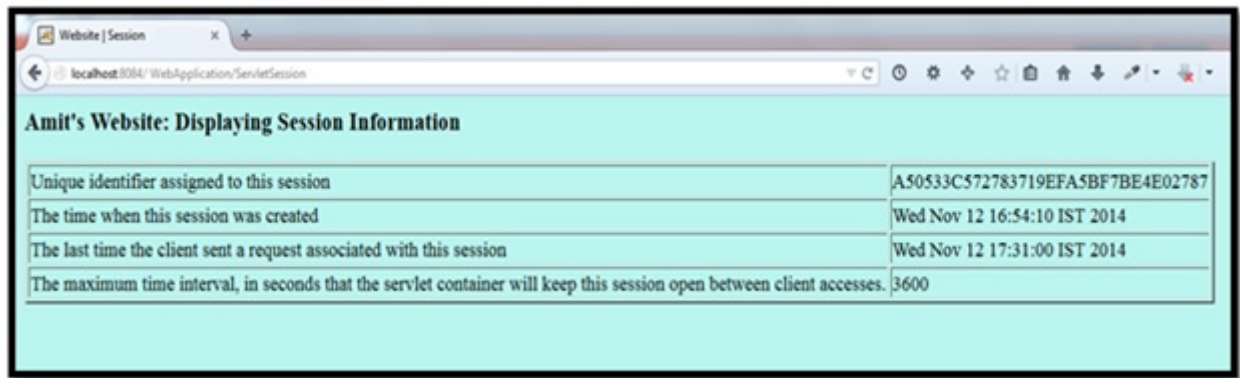
        throws ServletException, IOException {
    // session object creation
    HttpSession newSession = request.getSession(true);
    // Session creation time.
    Date cTime = new Date(newSession.getCreationTime());
    // The last time the client sent a request.
    Date lTime = new Date(newSession.getLastAccessedTime());

    /* 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" +
        "|===== \n" +
        "\n" +
        " |Unique identifier assigned to this session\n" +
        " |" + newSession.getId() + "
    + "\n" +
        "\n" +
        " |The time when this session was created\n" +
        " |" + cTime +
        " "
    + "\n" +
        "\n" +
        " |The last time the client sent a request associated with this session\n"
    + " |" + lTime +
        " "
    + "\n" +
        "</tr>\n" +
        "\n" +
        " | the maximum time interval, in seconds that the servlet container will ←
        keep this session open between client accesses.\n" +
        " |" + newSession.getMaxInactiveInterval() +
        " "
    + "\n" +
        "|===== \n" +
        "</body></html>");
    }
}

```

A screenshot of a web browser window. The title bar says "Website | Session". The address bar shows "localhost:8084/WebApplication/ServletSession". The page content has a light blue background and a title "Amit's Website: Displaying Session Information". Below the title is a table with four rows of session data.

Amit's Website: Displaying Session Information	
Unique identifier assigned to this session	A50533C572783719EFA5BF7BE4E02787
The time when this session was created	Wed Nov 12 16:54:10 IST 2014
The last time the client sent a request associated with this session	Wed Nov 12 17:31:00 IST 2014
The maximum time interval, in seconds that the servlet container will keep this session open between client accesses.	3600

Figure 6.1: Displaying output

Chapter 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

```
<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

```
<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:

```
<error-page>
  <exception-type>java.lang.Throwable</exception-type >
  <location>/HandlerClass</location>
</error-page>
```

Chapter 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.

```
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 {

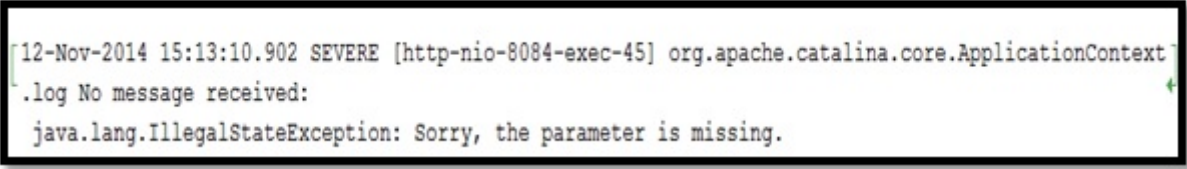
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // 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);

    }
}
```

A screenshot of a log window with a black border. The text inside is a log entry from Apache Tomcat. It starts with a timestamp and level, followed by the logger name and a message. The message indicates that no parameter was received and an exception was thrown.

```
12-Nov-2014 15:13:10.902 SEVERE [http-nio-8084-exec-45] org.apache.catalina.core.ApplicationContext
.log No message received:
java.lang.IllegalStateException: Sorry, the parameter is missing.
```

Figure 8.1: Output as visible in Apache Tomcat log

Chapter 9

Internationalization

For building a global website, some important points are considered which includes language related to user's nationality. Internationalization is enabling 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
<code>String getCountry()</code>	Returns the country code.
<code>String getDisplayCountry()</code>	Returns a name for the visitors' country.
<code>String getLanguage()</code>	Returns the language code.
<code>String getDisplayLanguage()</code>	Returns a name for the visitors' language.
<code>String getISO3Country()</code>	Returns a three-letter abbreviation for the visitors country.
<code>String getISO3Language()</code>	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:

```
Project Name: WebApplicationInternationalization
Project Location: C:\Users\Test\Documents\NetBeansProjects
Servlet: ServletLocale
URL Pattern: /ServletLocale
```

Listing 22: ServletLocale.java

```
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*

```

<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:
    <a href="ServletLocale"><b>Location</b></a>
</body>
</html>

```

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

```

<?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>

```



Figure 9.1: Displaying index.html

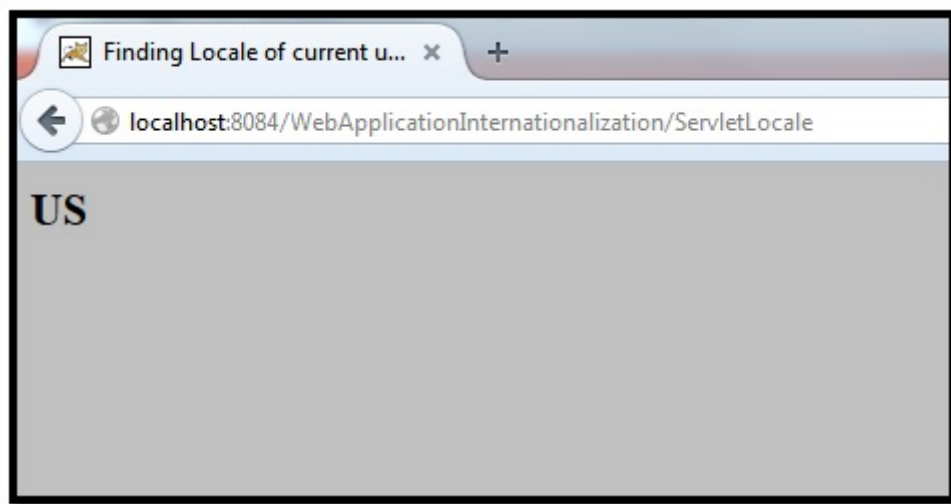


Figure 9.2: Displaying the locale as output

Chapter 10

Reference

Websites

- [Official Oracle Documentation](#)
- [Sun Developer Network](#)
- [Free NetBeans Download](#)
- [Free Apache Download](#)
- [Free Java Download](#)

Books

- Head First Servlets and JSP: Passing the Sun Certified Web Component Developer Exam, by Bryan Basham, Kathy Sierra , Bert Bates
 - Servlet and JSP (A Tutorial), by Budi Kurniawan
-

Chapter 11

Conclusion

Servlet is fast in performance and easy to use when compared with traditional Common Gateway Interfaces (CGI). Through this guide you can easily learn the concepts related to Java Servlets. The project codes are developed under NetBeansIDE, so you will get an idea about some of its amazing user-friendly features as well.

Chapter 12

Download

You can download the full source code of this tutorial here: [Servlet_Project_Code](#)