SERVLET API'S

(Exercise-II)

Table of Contents

Servlet Interface

In Servlet API's, Servlet is an interface that defines five methods, these all methods must be implemented by all Servlet

These methods are used to initialize a Servlet, to service requests, and to remove a Servlet from the server. These are known as life-cycle methods of Servlet.

Execution sequence of Servlet methods:

- ✓ First, The Servlet is constructed
- ✓ Second, The Servlet is initialized with the init method.
- ✓ Third, The Servlet is handled the requests and responses using service method.
- ✓ The Servlet is destroyed with the destroy method, then garbage collected and finalized.

Servlet Interface has mainly five methods these are as follows:

```
public interface Servlet {

   public void init(ServletConfig config) throws ServletException;

   public ServletConfig getServletConfig();

   public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException;

   public String getServletInfo();

   public void destroy();
}
```

Servlet interface Application

Required Files

1. **index.html**
2. **WelcomeServlet.java**
3. **web.xml**

Index.html

```
<h1>First Servlet Application;/h1>
<a href="welcome">Test Application</a>
```

WelcomeServlet.java

```
package com.kalibermind.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

 Public class WelcomeServlet implements Servlet {

   @Override
   public void init(ServletConfig config) throws ServletException {

   }

   @Override
   public void service(ServletRequest request, ServletResponse response)
         throws ServletException, IOException {

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<h1>Welcome to Servlet</h1>");
   }

   @Override
   public void destroy() {

   }

   @Override
   public String getServletInfo() {
      return "Welcome Servlet";
   }

   @Override
   public ServletConfig getServletConfig() {
      return null;
   }
}
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <display-name>http-servlet-app</display-name>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>WelcomeServlet</servlet-name>
    <servlet-class>com.kalibermind.servlet.WelcomeServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>WelcomeServlet</servlet-name>
    <url-pattern>/welcome</url-pattern>
  </servlet-mapping>
</web-app>
```

Types of Servlet

We can create Servlet by three ways

1. By **implementing Servlet** Interface
2. By **extends GenericServlet**
3. By **extends HttpServlet**

GenericServlet in Java

- ✓ **GenericServlet** defines a **protocol-independent** Servlet, that is not specific to any protocol.
- ✓ **GenericServlet** implements the **Servlet** and **ServletConfig** interfaces.
- ✓ **GenericServlet** may be directly extended by a Servlet.
- ✓ **GenericServlet** makes writing Servlet easier. It provides simple implementations of the lifecycle methods **init** and **destroy** and methods of the **ServletConfig** interface.
- ✓ To write a GenericServlet, you need only override the abstract **service** method.

Generic Servlet Application

Required Files

1. **index.html**
2. **WelcomeServlet.java**
3. **web.xml**

Index.html

```
      <h1>GenericServlet Application</h1>
      <a href="welcome">Test Application</a>
```

WelcomeServlet.java

```
package com.kalibermind.servlet;

 import java.io.IOException;
 import java.io.PrintWriter;

 import javax.servlet.GenericServlet;
 import javax.servlet.ServletException;
 import javax.servlet.ServletRequest;
 import javax.servlet.ServletResponse;

 public class WelcomeServlet extends GenericServlet{

   private static final long serialVersionUID = 1L;

   @Override
   public void service(ServletRequest request, ServletResponse response)
         throws ServletException, IOException {

      response.setContentType("text/html"); PrintWriter
      out = response.getWriter();
      out.println("<h1>Welcome to GenericServlet</h1>");

   }
}
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <display-name>http-servlet-app</display-name>

  <Welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>WelcomeServlet</servlet-name>
    <servlet-class>com.kalibermind.servlet.WelcomeServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>WelcomeServlet</servlet-name>
    <url-pattern>/welcome</url-pattern>
  </servlet-mapping>
</web-app>
```

HttpServlet

HttpServlet defines a protocol-dependent Servlet that is specific to the HTTP protocol. HttpServlet extends the GenericServlet.
HttpServlet must override at least one method, usually one of these:

1. **doGet**, if the servlet supports HTTP GET requests
2. **doPost**, for HTTP POST requests
3. **doPut, for** HTTP PUT requests
4. **doDelete**, for HTTP DELETE requests
5. **init** and **destroy**, to manage resources that are held for the life of the Servlet
6. **getServletInfo**, which the Servlet uses to provide information about itself.

HttpServlet Application

Same required files, we have to change in **WelcomeServlet.java** file .

Index.html

```
<h1>HttpServlet Application</h1>

<form action="welcome">
   Name: <input type="text" name="userName" />
   <input type="submit" value="Submit" />
</form>
```

WelcomeServlet.java

```
package com.kalibermind.servlet;

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 WelcomeServlet extends HttpServlet {

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

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<h1>Welcome to HttpServlet</h1>");

      String userName = request.getParameter("userName");
      out.print("Hello "+ userName);
   }
}
```

Note: Web.xml is same as per the above application

ServletRequest

- ✓ **ServletRequest** Defines an object to provide client request information to a Servlet.
- ✓ Servlet container creates a **ServletRequest** object and passes it as an argument to the servlet's **service** method.
- ✓ A **ServletRequest** object provides data including parameter **name** and **values**, **attributes**, and an **input stream**.
- ✓ Returns the value of the named attribute as an **Object**, or **null** if no attribute of the given name exists.

Methods of ServletRequest

SNo.	Methods	Description
1	**public Object getAttribute(String name);**	Returns an Object containing the value of the attribute, or null if the attribute does not exist
2	**public Enumeration getAttributeNames();**	Returns an Enumeration of strings containing the names of the request's attributes
3	**public String getParameter(String name);**	Returns String that represent the single value of the parameter
4	**public RequestDispatcher getRequestDispatcher(String path);**	Return a RequestDispatcher object that acts as a wrapper for the resource at the specified path, or null if the Servlet container cannot return a RequestDispatcher.
5	**public String getCharacterEncoding();**	Return a String containing the name of the character encoding, or null if the request does not specify a character encoding.
6	**public int getContentLength();**	return an integer containing the length of the request body or -1 if the length is not known
7	**public String getContentType();**	return a String containing the name of the MIME type of the request, or null if the type is not known
8	**public ServletInputStream getInputStream() throws IOException;**	return a ServletInputStream object containing the body of the request
9	**public Enumeration getParameterNames();**	return an Enumeration of String objects, each String containing the name of a request parameter; or an empty Enumeration if the request has no parameters
10	**public String[] getParameterValues(String name);**	return an array of String objects containing the parameter's values
11	**public Map getParameterMap();**	Return an immutable **java.util.Map** containing parameter names as keys and parameter values as map values. The keys in the parametermap are of type String. The values in the parameter map are of type String array.
12	**public String getProtocol();**	return a String containing the protocol name and version number
13	**public String getScheme();**	return a String containing the name of the scheme used to make this request
14	**public String getServerName();**	return a String containing the name of the server
15	**public int getServerPort();**	Returns the port number to which the request was sent.
16	**public BufferedReader getReader() throws IOException**	return a BufferedReader containing the body of the request
17	**public String getRemoteAddr();**	return a String containing the IP address of the client that sent the request
18	**public String getRemoteHost();**	return a String containing the fully qualified name of the client
19	**public void setAttribute(String name, Object o);**	Stores an attribute in this request. Attributes are reset between requests. This method is most often used in conjunction with RequestDispatcher.
20	**public void removeAttribute(String name);**	Removes an attribute from this request. This method is not generally needed as attributes only persist as long as the request is being handled.
21	**public Locale getLocale();**	return the preferred Locale for the client
22	**public Enumeration getLocales();**	return an Enumeration of preferred Locale objects for the client
23	**public boolean isSecure();**	Return a boolean indicating if the request was made using a secure channel
24	**public String getRealPath(String path);**	deprecated As of Version 2.1 of the Java Servlet API
25	**public int getRemotePort();**	Returns the Internet Protocol (IP) source port of the client or last proxy that sent

		the request.
26	**public String getLocalName();**	Return a String containing the host name of the IP on which the request was received.
27	**public String getLocalAddr();**	Return a String containing the IP address on which the request was received.
28	**public int getLocalPort();**	Returns the Internet Protocol (IP) port number of the interface on which the request was received.

ServletResponse

- ✓ ServletResponse defines an object to help a Servlet in sending a response to the client.
- ✓ The Servlet container creates a ServletResponse object and passes it as an argument to the servlet's service method.
- ✓ To send binary data in a MIME body response, use ServletOutputStream returned by getOutputStream().
- ✓ To send character data, use the PrintWriter object returned by getWriter().
- ✓ To send mix binary and text data,for example, to create a multipart response, use a ServletOutputStream and manage the character sections manually.

Methods of ServletResponse

SNo.	Methods	Description
1	**public String getCharacterEncoding();**	return a String specifying the name of the character encoding, for example, UTF-8
2	**public String getContentType();**	return a String specifying the content type, for example, text/html; charset=UTF-8, or null
3	**public ServletOutputStream getOutputStream() throws IOException;**	return a ServletOutputStream for writing binary data
4	**public PrintWriter getWriter() throws IOException;**	return a PrintWriter object that can return character data to the client
5	**public void setCharacterEncoding(String charset);**	Sets the character encoding (MIME charset) of the response being sent to the client, for example, to UTF-8.
6	**public void setContentLength(int len);**	Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header.
7	**public void setContentType(String type);**	Sets the content type of the response being sent tothe client, if the response has not been committed yet.
8	**public void setBufferSize(int size);**	Sets the preferred buffer size for the body of the response. The servlet container will use a buffer at least as large as the size requested.
9	**public int getBufferSize();**	Returns the actual buffer size used for the response. If no buffering is used, this method returns 0.
10	**public void flushBuffer() throws IOException;**	Forces any content in the buffer to be written to the client.
11	**public void resetBuffer();**	Clears the content of the underlying buffer in the response without clearing headers or status code. If the response has been committed, this method throws an IllegalStateException
12	**public boolean isCommitted();**	return a boolean indicating if the response has been committed

13	**public void reset();**	Clears any data that exists in the buffer as well as the status code and headers.
14	**public void setLocale(Locale loc);**	Sets the locale of the response, if the response has not been committed yet.
15	**public Locale getLocale();**	Returns the locale specified for this response using the setLocale method. Calls made to setLocale after the response is committed have no effect. If no locale has been specified, the container's default locale is returned.

Get vs POST

Get	Post
Data is sent in the form of URL	Data is sent separately
The URL with client data can be bookmarked	It can't be bookmarked
Limited data can be sent	Unlimited data can be sent
Only ASCII data can be sent	Any type of data can be sent
Prefer when data sent is not secret. Do not use for passwords **etc.**	Prefer for critical and sensitive data like passwords etc
If not mentioned, GET is assumed as default	Should be mentioned explicitly
Get request is **more efficient** and used more than Post.	Post request is **less efficient** and used less than get.
Get request is **idempotent.**	Post request is **non-idempotent.**

ServletConfig Interface

- ✓ A **ServletConfig** object is used by a Servlet container to pass information to a Servlet during initialization.
- ✓ Before Servlet **initialization**, web container **creates** a ServletConfig object for every servlet, then pass it to the **init ()** method to initialize the Servlet.
- ✓ Programmer can pass own parameters and values through **web.xml** file.

Methods of ServletConfig

SNo.	Methods	Description
1	**public String getServletName();**	return the name of the servlet instance
2	**public ServletContext getServletContext();**	Returns a reference to the ServletContext in which the caller is executing.
3	**public String getInitParameter(String name);**	Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.
4	**public Enumeration getInitParameterNames();**	return an Enumeration of String objects containing the names of the servlet's initialization parameters.

ServletConfig Application

Required Files

1. **index.html**
2. **WelcomeServlet.java**
3. **WelcomeServlet2.java**
4. **web.xml**

Index.html

```
<h1>ServletConfig Application</h1>
<a href="WelcomeServlet">Test Servlet-1 Here</a> <br>
<a href="WelcomeServlet2">Test Servlet-2 Here</a>
```

WelcomeServlet.java

```
package com.kalibermind.servlet;

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

public class WelcomeServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

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

      PrintWriter out = response.getWriter();

      ServletConfig config = getServletConfig();

      out.print("From Servlet-1: "+config.getInitParameter("servlet-name"));
      out.print("<br>From Servlet-2: "+config.getInitParameter("servlet-name2"));
   }
}
```

WelcomeServlet2.java

```
package com.kalibermind.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class WelcomeServlet2 extends HttpServlet {
   private static final long serialVersionUID = 1L;

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

      PrintWriter out = response.getWriter();

      ServletConfig config = getServletConfig();

      out.print("From Servlet-1: "+config.getInitParameter("servlet-name"));
      out.print("<br>From Servlet-2: "+config.getInitParameter("servlet-name2"));
   }
}
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <display-name>servlet-config-app</display-
  name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>WelcomeServlet</servlet-name>
    <servletclass>com.kalibermind.servlet.WelcomeServlet</servletclass>
    <init-param>
      <param-name>servlet-name</param-name> <param-
      value>This is Welcome Servlet.</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>WelcomeServlet</servlet-name>
    <url-pattern>/WelcomeServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>WelcomeServlet2</servlet-name>
    <servlet-class>com.kalibermind.servlet.WelcomeServlet2</servlet-
    class>
    <init-param>
      <param-name>servlet-name2</param-name>
      <param-value>This is Welcome Servlet2.</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>WelcomeServlet2</servlet-name>
    <url-pattern>/WelcomeServlet2</url-pattern>
  </servlet-mapping>
</web-app>
```

ServletContext Object in Servlet

1. ServletContext is an object that is created by servlet container when we deploy our application.
2. Servlet container creates one ServletContext object per web application.
3. We can share any parameter or information between all the servlet by using ServletContext Object.
4. ServletContext interface defines a set of methods that a servlet uses to communicate with it's servlet container.
5. Programmer can set, get and remove attributes from the ServletContext Object.

Mostly used Methods of ServletContext

Methods	Description
String getInitParameter(String name)	Returns a initialization parameter value, or null if the parameter does not exist.
void setAttribute(String name, Object obj)	Sets an object to a given attribute name in this servlet context.
Object getAttribute(String name)	Returns the servlet container attribute with the given name, or null if there is no attribute by that name.
void removeAttribute(String name)	Removes the attribute with the given name from the servlet context.

String getServletContextName()	Returns the name of this web application, specified in the web.xml by the **display-name** element.
String getMimeType()	Returns the MIME type of the specified file, or null if the MIME type is not known.

Passing Custom Parameters and values

- ✓ Programmer can pass own parameters and values in ServletContext object. To initialize these parameters, we need to define these parameters and values in **web.xml** file.
- ✓ **web.xml** file defines some elements to pass parameters and values.
- ✓ Programmer can pass any number of parameters that will be **shared** between all the Servlets.

```
<context-param>
    <param-name>username</param-name>
    <param-value>root</param-value>
</context-param>

<context-param>
    <param-name>password</param-name>
    <param-value>admin</param-value>
</context-param>
```

Here we have defined two parameters - **username** and **password**, when we get username it will return root as a value and password will return admin as a value.

- ✓ We can get ServletContext object by calling **getServletContext ()** method of ServletConfig Object.

ServletContext Application

Required Files

1. **index.html**
2. **WelcomeServlet.java**
3. **web.xml**

index.html

```
<h1>Servlet Context Application</h1>
<p>Test your application
   <a href="WelcomeServlet">here</a>
</p>
```

WelcomeServlet.java

```
package com.kalibermind.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/WelcomeServlet")
public class WelcomeServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

   protected void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      // TODO Auto-generated method stub
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      ServletContext context = getServletConfig().getServletContext();
      String username = context.getInitParameter("username");
      String password = context.getInitParameter("password");

      out.print("<h1>ServletContext Parameters</h1>");
      out.println("User Name: "+username+"<br>");
      out.println("Password: "+password);
   }
}
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
<display-name>servlet-context-app</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

  <context-param>
    <param-name>username</param-name>
    <param-value>root</param-value>
  </context-param>

  <context-param>
    <param-name>password</param-name>
    <param-value>admin</param-value>
  </context-param>
</web-app>
```