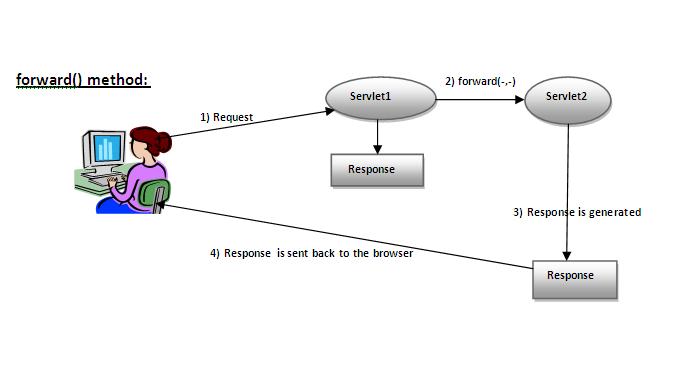
**RequestDispatcher in Servlet**

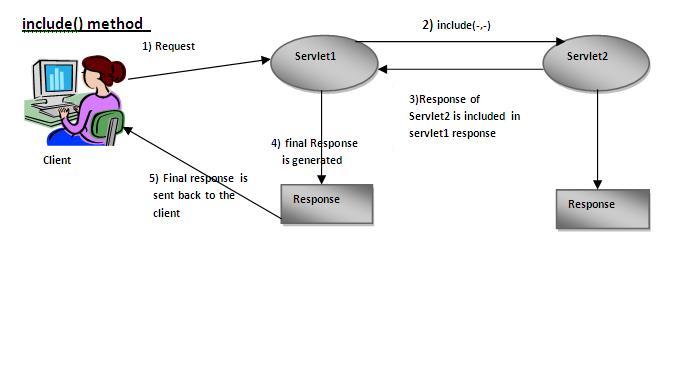
The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

The RequestDispatcher interface provides two methods. They are:

1. **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
2. **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

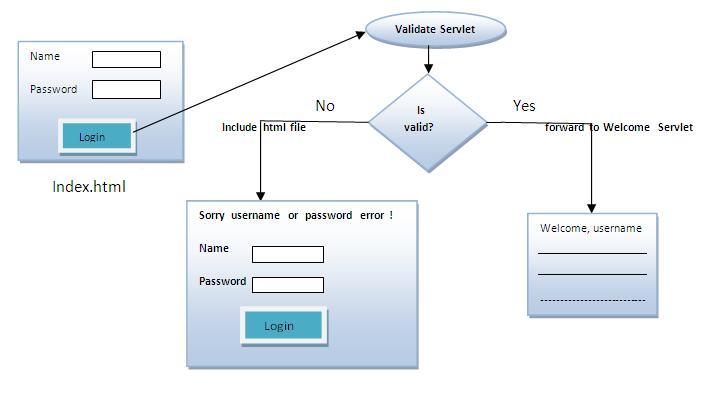




Example of RequestDispatcher interface

In this example, we are validating the password entered by the user. If password is servlet, it will forward the request to the WelcomeServlet, otherwise will show an error message: sorry username or password error!. In this program, we are cheking for hardcoded information. But you can check it to the database also that we will see in the development chapter. In this example, we have created following files:

* **index.html file:** for getting input from the user.
* **Login.java file:** a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.
* **WelcomeServlet.java file:** a servlet class for displaying the welcome message.
* **web.xml file:** a deployment descriptor file that contains the information about the servlet.



**index.html**

1. <form action="servlet1" method="post">
2. Name:<input type="text" name="userName"/><br/>
3. Password:<input type="password" name="userPass"/><br/>
4. <input type="submit" value="login"/>
5. </form>



**Login.java**

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;

6. **public** **class** Login **extends** HttpServlet {
8. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)
9. **throws** ServletException, IOException {
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. String n=request.getParameter("userName");
15. String p=request.getParameter("userPass");
17. **if**(p.equals("servlet"){
18. RequestDispatcher rd=request.getRequestDispatcher("servlet2");
19. rd.forward(request, response);
20. }
21. **else**{
22. out.print("Sorry UserName or Password Error!");
23. RequestDispatcher rd=request.getRequestDispatcher("/index.html");
24. rd.include(request, response);
26. }
27. }
29. }



**WelcomeServlet.java**

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
5. **public** **class** WelcomeServlet **extends** HttpServlet {
7. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)
8. **throws** ServletException, IOException {
10. response.setContentType("text/html");
11. PrintWriter out = response.getWriter();
13. String n=request.getParameter("userName");
14. out.print("Welcome "+n);
15. }
17. }



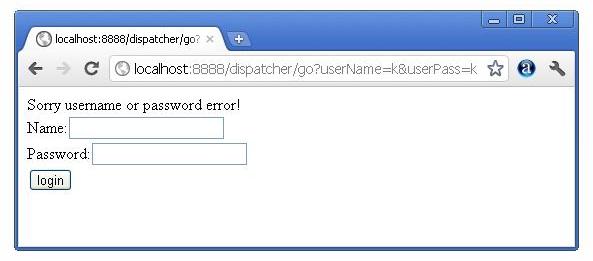
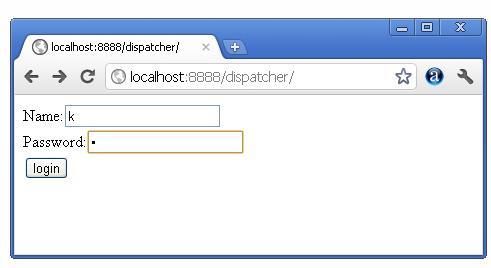
**web.xml**

1. <web-app>
2. <servlet>
3. <servlet-name>Login</servlet-name>
4. <servlet-**class**>Login</servlet-**class**>
5. </servlet>
6. <servlet>
7. <servlet-name>WelcomeServlet</servlet-name>
8. <servlet-**class**>WelcomeServlet</servlet-**class**>
9. </servlet>

12. <servlet-mapping>
13. <servlet-name>Login</servlet-name>
14. <url-pattern>/servlet1</url-pattern>
15. </servlet-mapping>
16. <servlet-mapping>
17. <servlet-name>WelcomeServlet</servlet-name>
18. <url-pattern>/servlet2</url-pattern>
19. </servlet-mapping>
21. <welcome-file-list>
22. <welcome-file>index.html</welcome-file>
23. </welcome-file-list>
24. </web-app>



[download this example](http://www.javatpoint.com/src/servlet/requestdispatcher.zip)  
[download this example (developed in Myeclipse IDE)](http://www.javatpoint.com/src/servlet/requestdispatcherm.zip)  
[download this example (developed in eclipse IDE)](http://www.javatpoint.com/src/servlet/eclipse/requestdispatcher.zip)  
[download this example (developed in netbeans IDE)](http://www.javatpoint.com/src/servlet/netbeans/requestdispatcher.zip)



**SendRedirect in servlet**

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

## Difference between forward() and sendRedirect() method

There are many differences between the forward() method of RequestDispatcher and sendRedirect() method of HttpServletResponse interface. They are given below:

|  |  |
| --- | --- |
| **forward() method** | **sendRedirect() method** |
| The forward() method works at server side. | The sendRedirect() method works at client side. |
| It sends the same request and response objects to another servlet. | It always sends a new request. |
| It can work within the server only. | It can be used within and outside the server. |
| Example: request.getRequestDispacher("servlet2").forward(request,response); | Example: response.sendRedirect("servlet2"); |

### Full example of sendRedirect method in servlet

|  |
| --- |
| In this example, we are redirecting the request to the google server. Notice that sendRedirect method works at client side, that is why we can our request to anywhere. We can send our request within and outside the server. |

*DemoServlet.java*

1. import java.io.\*;
2. import javax.servlet.\*;
3. import javax.servlet.http.\*;
5. public class DemoServlet extends HttpServlet{
6. public void doGet(HttpServletRequest req,HttpServletResponse res)
7. throws ServletException,IOException
8. {
9. res.setContentType("text/html");
10. PrintWriter pw=res.getWriter();
12. response.sendRedirect("http://www.google.com");
14. pw.close();
15. }}



### Creating custom google search using sendRedirect

In this example, we are using sendRedirect method to send request to google server with the request data.

*index.html*

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset="ISO-8859-1">
5. <title>sendRedirect example</title>
6. </head>
7. <body>

10. <form action="MySearcher">
11. <input type="text" name="name">
12. <input type="submit" value="Google Search">
13. </form>
15. </body>
16. </html>



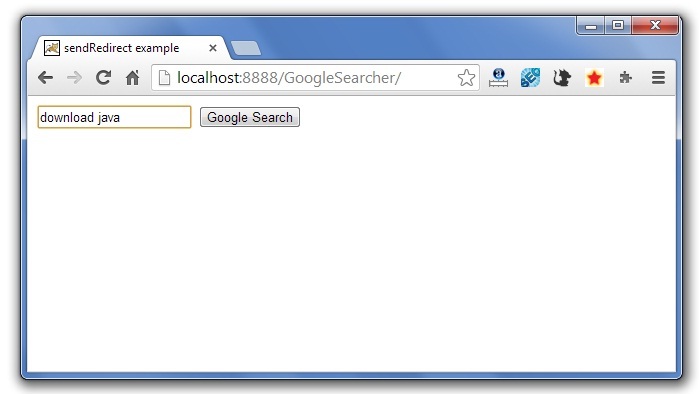
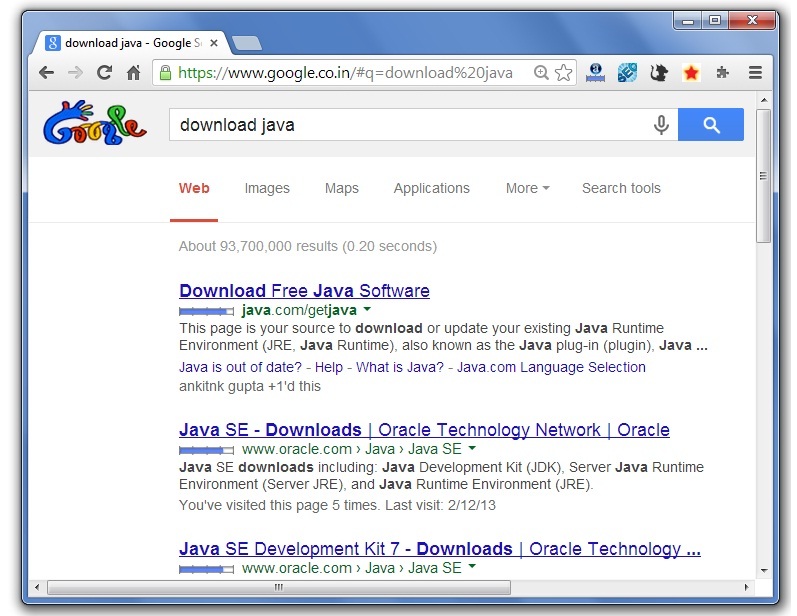
*MySearcher.java*

1. import java.io.IOException;
2. import javax.servlet.ServletException;
3. import javax.servlet.http.HttpServlet;
4. import javax.servlet.http.HttpServletRequest;
5. import javax.servlet.http.HttpServletResponse;
7. public class MySearcher extends HttpServlet {
8. protected void doGet(HttpServletRequest request, HttpServletResponse response)
9. throws ServletException, IOException {
11. String name=request.getParameter("name");
12. response.sendRedirect("https://www.google.co.in/#q="+name);
13. }
14. }



[download this example (developed in Eclipse)](http://www.javatpoint.com/src/servlet/eclipse/GoogleSearcher.zip)

#### Output

**ServletContext Interface**

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

### Advantage of ServletContext

**Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

Usage of ServletContext Interface

There can be a lot of usage of ServletContext object. Some of them are as follows:

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.

Syntax to provide the initialization parameter in Context scope

|  |
| --- |
| The **context-param** element, subelement of web-app, is used to define the initialization parameter in the application scope. The param-name and param-value are the sub-elements of the context-param. The param-name element defines parameter name and and param-value defines its value. |

1. <web-app>
2. ......
4. <context-param>
5. <param-name>parametername</param-name>
6. <param-value>parametervalue</param-value>
7. </context-param>
8. ......
9. </web-app>



Example of ServletContext to get the initialization parameter

|  |
| --- |
| In this example, we are getting the initialization parameter from the web.xml file and printing the value of the initialization parameter. Notice that the object of ServletContext represents the application scope. So if we change the value of the parameter from the web.xml file, all the servlet classes will get the changed value. So we don't need to modify the servlet. So it is better to have the common information for most of the servlets in the web.xml file by context-param element. Let's see the simple example: |

**DemoServlet.java**

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;

6. **public** **class** DemoServlet **extends** HttpServlet{
7. **public** **void** doGet(HttpServletRequest req,HttpServletResponse res)
8. **throws** ServletException,IOException
9. {
10. res.setContentType("text/html");
11. PrintWriter pw=res.getWriter();
13. //creating ServletContext object
14. ServletContext context=getServletContext();
16. //Getting the value of the initialization parameter and printing it
17. String driverName=context.getInitParameter("dname");
18. pw.println("driver name is="+driverName);
20. pw.close();
22. }}



**web.xml**

1. <web-app>
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <servlet-**class**>DemoServlet</servlet-**class**>
6. </servlet>
8. <context-param>
9. <param-name>dname</param-name>
10. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
11. </context-param>
13. <servlet-mapping>
14. <servlet-name>sonoojaiswal</servlet-name>
15. <url-pattern>/context</url-pattern>
16. </servlet-mapping>
18. </web-app>



Example of ServletContext to get all the initialization parameters

|  |
| --- |
| In this example, we are getting all the initialization parameter from the web.xml file. For getting all the parameters, we have used the getInitParameterNames() method in the servlet class. |

**DemoServlet.java**

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;

6. **public** **class** DemoServlet **extends** HttpServlet{
7. **public** **void** doGet(HttpServletRequest req,HttpServletResponse res)
8. **throws** ServletException,IOException
9. {
10. res.setContentType("text/html");
11. PrintWriter out=res.getWriter();
13. ServletContext context=getServletContext();
14. Enumeration<String> e=context.getInitParameterNames();
16. String str="";
17. **while**(e.hasMoreElements()){
18. str=e.nextElement();
19. out.print("<br> "+context.getInitParameter(str));
20. }
21. }}



**web.xml**

1. <web-app>
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <servlet-**class**>DemoServlet</servlet-**class**>
6. </servlet>
8. <context-param>
9. <param-name>dname</param-name>
10. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
11. </context-param>
13. <context-param>
14. <param-name>username</param-name>
15. <param-value>system</param-value>
16. </context-param>
18. <context-param>
19. <param-name>password</param-name>
20. <param-value>oracle</param-value>
21. </context-param>
23. <servlet-mapping>
24. <servlet-name>sonoojaiswal</servlet-name>
25. <url-pattern>/context</url-pattern>
26. </servlet-mapping>
28. </web-app>