Servlet Collaboration

Table of Contents

Servlet Collaboration

- The Servlet collaboration is all about sharing information between the servlets.
- Collaborating servlets is to pass the common information that is to be shared directly by one servlet to another through invoking some methods.

Servlets can communicate with one another:

- Using **RequestDispatcher's include**() and **forward**() method;
- Using **HttpServletResponse**.sendRedirect() method;

Redirect page using sendRedirect() method

- **SendRedirect ()** is a method that is belong to HttpServletResponse Interface.
- It redirects the response to the client using the specified URL.
- **SendRedirect ()** works at client side.
- This method can accept relative URL; the Servlet container converts that URL to an absolute URL before sending the response to the client.
- If the location is relative without '/' the container interprets it as relative to the current request URI.
- If the response has already been committed, this method throws an IllegalStateException.

Note: **sendRedirect ()** works at client side, it will show the redirected URL in the web browser.

Servlet collaboration using senRedirect() Application

Required Files:

index.html

welcome.html
FirstServlet.java

Index.html

```
<h3>Redirect Servlet using sendRedirect</h3>
<a href="FirstServlet">Click Here</a>
```

Welcome.html

```
<h3>Welcome to KaliberMind</h3>
```

FirstServlet.java

```
package com.kalibermind.servlet;

import java.io.IOException;
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("/FirstServlet")
public class FirstServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

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

      response.setContentType("text/html");
      response.sendRedirect("welcome.html");
   }
}
```

RequestDispatcher

- RequestDispatcher is an interface that defines an object that receives requests from the client and sends them to any resource (such as a Servlet, HTML file, or JSP file) on the server.
- The Servlet container creates the object of RequestDispatcher, which is used as a wrapper around a server resource located at a particular path or given by a particular name.\
- The RequestDispatcher works on the server only.

Methods of RequestDispatcher

Sn.	Method	Description
1	**public void forward(ServletRequest request, ServletResponse response)**	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)**	Includes the content of a resource (servlet, JSP page, HTML file) in the response.

✓ The forward method should be called before the response has been committed to the client (before response body output has been flushed).

Remember

The forward method should be used to give another resource responsibility for replying to the user. If you have already accessed a ServletOutputStream or PrintWriter object within the Servlet, you cannot use this method. It will throw an IllegalStateException.

Get Object of RequestDispatcher

To get the RequestDispatcher object, we call **getRequestDispatcher ()** method of ServletRequest interface

```
 RequestDispatcher dispatcher = request.getRequestDispatcher("resourceName");
dispatcher.forward (request, response);
 or
dispatcher.include (request, response);
```

Servlet collaboration using RequestDispatcher -Application

Required Files:

index.html   web.xml FirstServlet.java   SecondServlet.java

index.html

```
<h3>Servlet Redirect Using
RequestDispatcher</h3> <form
action="FirstServlet"> <table>
    <tr>
        <td>Name:</td>
        <td><input type="text" name="name"
    required></td> </tr>
    <tr>
        <td>Email:</td>
        <td><input type="email" name="email"
    required></td> </tr>
    <tr>
        <td>Password:</td>
        <td><input type="password" name="password" required></td>
    </tr>
    <tr>
        <td></td>
        <td><input type="submit"
    value="Submit"></td> </tr>
</table>
</form>
```

FirstServlet.java

```
package com.kalibermind.servlet;

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

public class FirstServlet extends HttpServlet
{
   private static final long serialVersionUID = 1L;
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out=response.getWriter();
      String password=request.getParameter("password");

      if(password.equals("admin"))
      {
         RequestDispatcher rd=request.getRequestDispatcher("SecondServlet");
         rd.forward(request,response);
      }
      else
      {
         out.println("<h3 style="color:red;">Sorry your password is wrong</h3>");
         RequestDispatcher rd=request.getRequestDispatcher("/index.jsp");
         rd.include(request, response);
      }
   }
}
```

SecondServlet.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 SecondServlet extends HttpServlet
{
   private static final long serialVersionUID = 1L;

   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out=response.getWriter();
      String name=request.getParameter("name");
      String email=request.getParameter("email");
      out.println("<h2>This is KaliberMind</h2>");
      out.println("<h3>Welcome Mr. "+name+"</h3>");
      out.println("<h3>Email Id: "+email+"</h3>");
   }
}
```

web.xml

```
<web-app >
  <display-name>servlet-collaboration2</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>FirstServlet</servlet-name>
    <servlet-class>com.kalibermind.servlet.FirstServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>FirstServlet</servlet-name>
    <url-pattern>/FirstServlet</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>SecondServlet</servlet-name>
    <servlet-class>com.kalibermind.servlet.SecondServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SecondServlet</servlet-name>
    <url-pattern>/SecondServlet</url-pattern>
  </servlet-mapping>
```

sendRedirect() Vs forward()

RequestDispatcher - forward () method

- When we use forward method, request is transfer to other resource within the same server for further processing.
- In case of forward, web container handle all process internally and client or browser is not involved.
- When forward is called on requestdispatcher object we pass request and response objects so our old request object is present on new resource which is going to process our request.
- Visually we are not able to see the forwarded address, it is transparent.
- Using forward () method is faster then send redirect.
- When we redirect using forward and we want to use same data in new resource we can use request.setAttribute () as we have request object available.

HttpServletResponse - sendRedirect () method

- In case of sendRedirect, request is transfer to another resource to different domain or different server for further processing.
- When you use sendRedirect, container transfers the request to client or browser so URL given inside the sendRedirect method is visible as a new request to the client.
- In case of sendRedirect call, old request and response objects are lost because it's treated as new request by the browser.
- In address bar, we are able to see the new redirected address. It's not transparent.
- sendRedirect() is slower because one extra round trip is required, because completely new request is created and old request object is lost. Two browser request required.
- But in sendRedirect, if we want to use we have to store the data in session or pass along with the URL.

Which one is good?

- Its depends upon the scenario that which method is more useful.
- If you want control is transfer to new server or context and it is treated as completely new task then we go for Send Redirect. Generally, forward should be used if the operation can be safely repeated upon a browser reload of the web page will not affect the result.