Day - 11
forward( ) v/s sendRedirect( )

Previously we have seen that from a Servlet or a JSP file we can take control of resources like an HTML file, Servlet or a JSP file and we can achieve this by using the RequestDispatcher object.
RequestDispatcher object has two methods namely -

- ➔ **forward( )**
- ➔ **include( )**

We have another method called sendRedirect( ) which can perform all the operations which the forward( ) does.
sendRedirect( ) is present inside the Response object and we will be discussing this method in this session.
Let us assume that in our JEE project we have 3 servlets, **Servlet1.java**, **Servlet2.java** and **Servlet3.java** and whenever a user requests for http://localhost:9090/J2EE/servlet1, then request will be sent to Servlet1 and that request will be forwarded to Servlet2 and Servlet2 will forward the request to Servlet3. Now Servlet3 will process the request and send the response back to the client.
Let us implement this in our code-
Create 3 servlets- **Servlet1.java**, **Servlet2.java,** and **Servlet3.java** in a package ***com.tap.exmp*** and also map the servlets in the web.xml

```
<servlet>
		<servlet-name>Servlet1</servlet-name>
		<servlet-class>com.tap.exmp.Servlet1</servlet-class>
	</servlet>

	<servlet-mapping>
		<servlet-name>Servlet1</servlet-name>
		<url-pattern>/servlet1</url-pattern>
	</servlet-mapping>

	<servlet>
		<servlet-name>Servlet2</servlet-name>
		<servlet-class>com.tap.exmp.Servlet2</servlet-class>
	</servlet>

	<servlet-mapping>
		<servlet-name>Servlet2</servlet-name>
		<url-pattern>/servlet2</url-pattern>
	</servlet-mapping>

	<servlet>
		<servlet-name>Servlet3</servlet-name>
		<servlet-class>com.tap.exmp.Servlet3</servlet-class>
	</servlet>

	<servlet-mapping>
		<servlet-name>Servlet3</servlet-name>
		<url-pattern>/servlet3</url-pattern>
```

```
	</servlet-mapping>
```

Type in the following-

Servlet1.java

```
package com.tap.exmp;

public class Servlet1 extends HttpServlet {

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {

		req.getRequestDispatcher("/servlet2").forward(req, resp);
	}
}
```

Servlet2.java

```
package com.tap.exmp;

public class Servlet2 extends HttpServlet {

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {

		req.getRequestDispatcher("/servlet3").forward(req, resp);
	}
}
```

Servlet3.java

```
package com.tap.exmp;

public class Servlet3 extends HttpServlet {

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {

		resp.setContentType("text/html");

		PrintWriter writer = resp.getWriter();
		writer.println("<h3>Response of Servlet-3</h3>");
	}
}
```

And upon execution, we will get the following output-

Now let us try to understand how sendRedirect( ) works

Note:

- ➔ sendRedirect( ) belongs to HttpServletResponse
- ➔ Never use "/" when mentioning the URL in sendRedirect( )

Let us assume that, the client sends a request to Servlet1, Now Servlet1 will forward the request to Servlet2 but since we are using sendRedirect( ) which is belongs to HttpServletResponse, it will send a response back to the client
Now in the URL bar, the servlet1 will be replaced by servlet2 as we are redirecting to servlet2 from servlet1. Since the URL is changed, now another request is sent from the client to the server to Servlet2.java.
We have again used sendRedirect( ) in the Servlet2.java file, so the Servlet2.java file will send 'servlet3' as the response and servlet2 will be replaced by servlet3 in the URL bar.
Now a request will be sent from the client to the server and the response of Servlet3 will be displayed in the browser.

Servlet1.java

```
public class Servlet1 extends HttpServlet {

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {

		//req.getRequestDispatcher("/servlet2").forward(req, resp);

		resp.sendRedirect("servlet2");
	}
}
```

Servlet2.java

```
public class Servlet2 extends HttpServlet {

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {

		//req.getRequestDispatcher("/servlet3").forward(req, resp);

		resp.sendRedirect("servlet3");
	}

}
```

Servlet3.java

```
public class Servlet3 extends HttpServlet {

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {

		resp.setContentType("text/html");

		PrintWriter writer = resp.getWriter();
		writer.println("<h3>Response of Servlet-3</h3>");
	}

}
```

Upon execution, we will get the following output-

And as we can see, the URL has changed from servlet1 to servlet3 as every time chaining happened, a new request was sent to the server

Difference between forward( ) and sendRedirect( )

	forward()	sendRedirect()
1	forward( ) will perform internal handling within the server	sendRedirect( ) will perform handling inside the client
2	The URL does not change	The URL inside the client will change
3	forward( ) mechanism is faster	sendRedirect( ) is slower
4	Whenever we use forward( ), the same request and response will be forwarded	A new request object will be created for each time the request is redirected

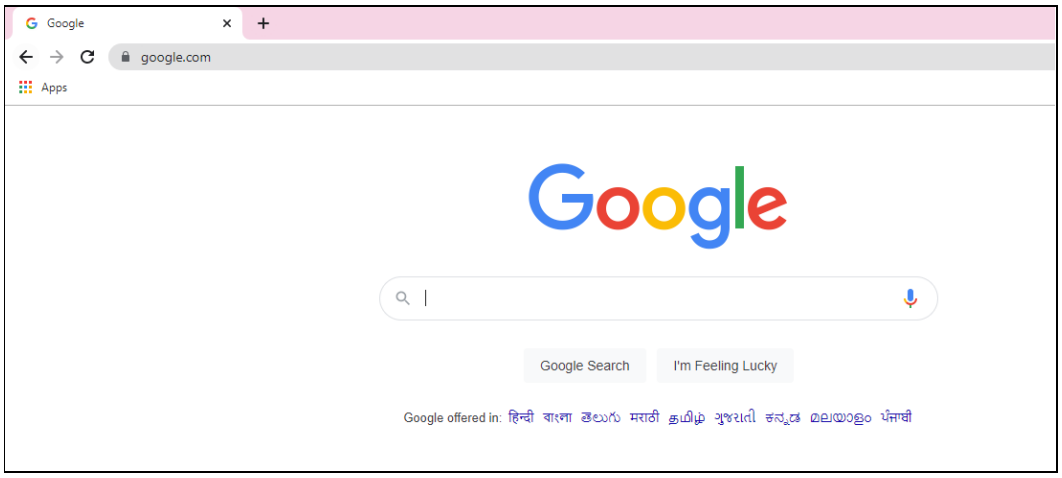
So when to use forward( ) and sendRedirect( )?

We can use forward( ) whenever redirection is within the same server but when we have to redirect from one server to another, we have to use sendRedirect( ) as with forward( ) we will not be able to achieve it.

For eg:

If we want to redirect the request from our server to the Google server, then with forward( ) we will get the following error-

But if we use sendRedirect( ), we will the following output-



As we can clearly see, the request was forwarded from our server to Google's server.

Now the next question would be, When to use forward( ) and include( )?

We can use forward( ) whenever the response of the previous servlet is not important or not required but when the previous responses of the servlets are important or required, we always have to use include( ). The include( ) will also capture the pending responses which were mentioned in the servlets while chaining back.