Servlets

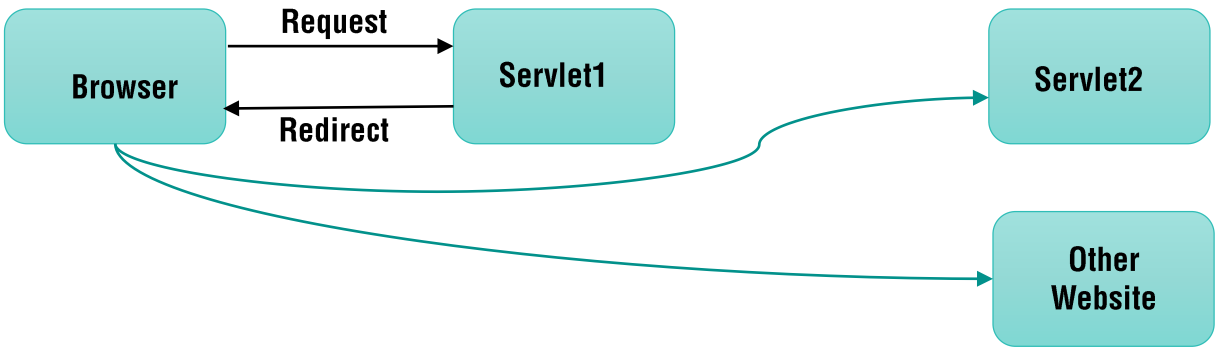
LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

○ Redirection
○ Attributes
○ Session Management

Redirection

- Redirection used to redirect response to another resource which may be servlet, jsp or html file.
- Control is passed to the browser, which redirects the request to the url specified.
- Request can be passed within the same web application or to other web applications on different servers.
- Done using **sendRedirect(String url)** of HttpServletResponse



```
resp.sendRedirect("http://www.manipalglobal.com");
resp.sendRedirect("servlet2");
```

- Redirect Method sets status code to 302
- It sets the url specified in the response header named ‘location’

Servlet Chaining using RequestDispatcher

- Servlet chaining means communication between two servlets

- RequestDispatcher is used to FORWARD or INCLUDE a request from one servlet to another Servlet/JSP **within the same web application**

- RequestDispatcher object can be obtained in following two ways

  - request.getRequestDispatcher(String path)
    - path can be absolute or relative from current path

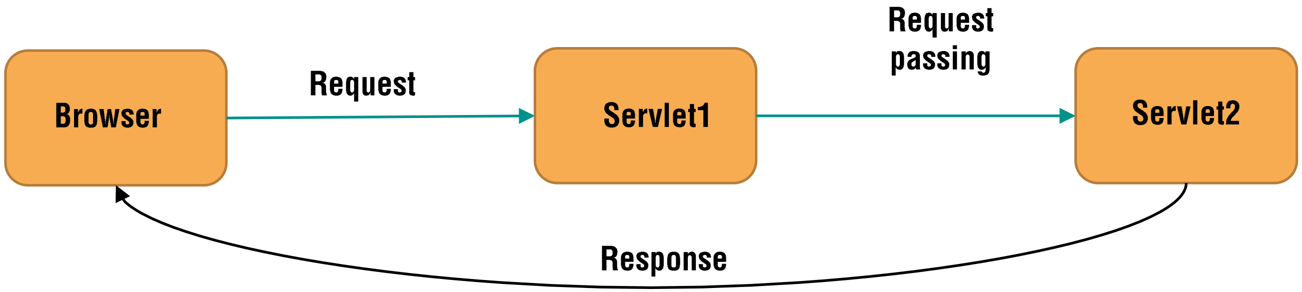
  - context.getRequestDispatcher(String path)
    - path can be only absolute path from context root

- RequestDispatcher interface provides two methods.
  - forward(request,response)
  - include(request,response)

Forward method

- Allows a servlet to forward the request to another servlet/JSP/html
- Request is forwarded to other resource permanently
- Most commonly used way of transferring requests
- Can be called only, if the response is not committed else an IllegalStateException is thrown

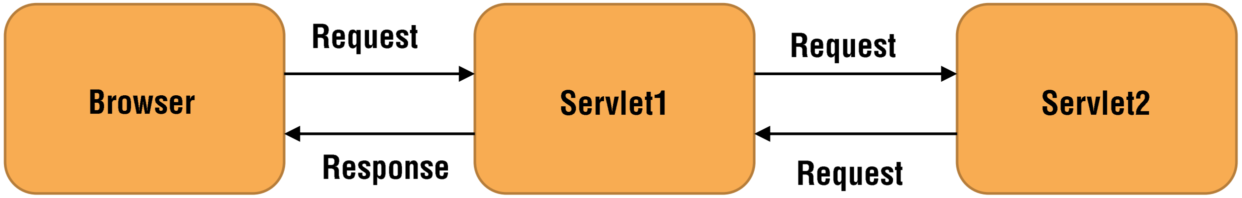
```
RequestDispatcher dispatch = request.getRequestDispatcher("Servlet2");
dispatch.forward(request,response);
```



Include Method

- Request is transferred to other resource(Servlet/JSP/html) temporarily
- Other component processes the request and returns back the control

```
ServletContext cntx=getServletContext();
RequestDispatcher dispatch=cntx.getRequestDispatcher("/Servlet2");
dispatch.include(request,response);
```



Attributes

- An attribute is an object bound to one of three objects given below
  - ServletContext
  - HttpServletRequest
  - HttpSession
    - Session is an object used to maintain conversational state with a client
    - The session persists *across multiple requests from the same client*

- Name/value pair where the name is a String and the value is an Object

- Method to add attributes

  setAttribute(String name, Object value)

- Method to get attribute

  getAttribute(String name) : returns Object

- Method to remove attributes

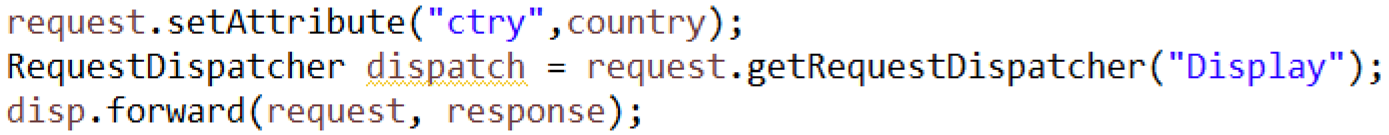
  removeAttribute(String name)

Scope of attributes

- Context/Application attributes
  - All Servlets/JSP’s in the application have access to this attribute
  - Any servlet/JSP can bind an object to context as a context attribute
  - Any servlet/JSP can remove an attribute bound to context
- Session attributes
  - Accessible to web components which participate in a specific HttpSession
- Request attributes
  - Accessible to Servlet’s and JSP’s processing the request
  - When a servlet transfer’s request to other Servlet/JSP, They will have acccess to the request attribute

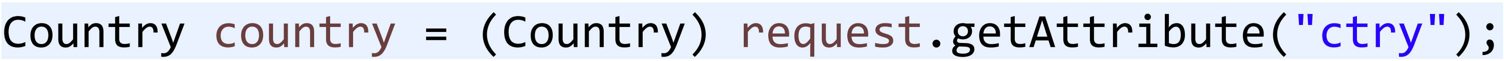
Scope of attributes

Servlet1 sets an attribute and forwards the request to Servlet 2



```
request.setAttribute("ctry",country);
RequestDispatcher dispatch = request.getRequestDispatcher("Display");
disp.forward(request, response);
```

Servlet2 gets the attribute



```
Country country = (Country) request.getAttribute("ctry");
```

Session Management

- Conversation consists of series of continuous request and response between client & server
- Session is a conversation between the server and a client
- For Applications like online shopping, Railway Booking,Banking etc, state information has to be saved between multiple requests, to maintain a conversation
- **Http is a stateless protocol** and hence conversational State is not maintained
- Session Management is a mechanism used to save the state of conversation between multiple client requests
- Following are ways used for managing sessions
  - Hidden form field
  - Cookies
  - HttpSession
  - URLRewriting

Ways of Managing Session

- Hidden field
  - a hidden textfield in the form is used for maintaining the state

```
<input type="hidden" name=“userId" value=“1000">
<input type="hidden" name=“role" value=“admin">
```

- Cookies
  - **cookie** is a small piece of information that is persisted between the multiple client requests
  - Cookie is data stored as name/value pair
  - Cookies are maintained at client side
  - Limitations
    - Session Management fails if cookies is disabled by the browser
    - Only textual information can be set in Cookie

Session Management with HttpSession

- HttpSession object is used to hold the conversational state across multiple requests(from the same client)
- HttpSession Object is created and stored on the Server. Every Session is assigned a unique identifier called a Session ID
- The state of the conversation can be stored by binding attributes to the session object
- Session id is sent to browser in a cookie which is stored on the users computer
- Browser sends the cookie back to the server every time a page is requested
- The Container will match the session ID from the cookie to a session object and used it for session management

HttpSession Methods

- request.getSession() / request.getSession(true)
  - Used for creating a session object
  - It will always return a session object
  - If there is no session object, it will create a new session
  - If a Session Object already exists, it will return the same.

- request.getSession(false)
  - Used for an existing session object
  - It returns only an existing session object
  - In case it does not exist, it will return null.

HttpSession Methods

- session.setAttribute(String, object) : Used for setting an attribute to a session
- session.getAttribute(String) : Used for reading an attribute
- session.invalidate() : used for discarding entire session
- session.setMaxInactiveInterval(int interval)
  - specifies the time, in seconds, before the servlet container will invalidate this session.
  - An interval of zero or negative value indicates that the session should never timeout

Session tracking - URL Rewriting

- URL rewriting appends the session ID to the URL for every page that's requested
- Server uses this session id to associate with a session
- All pages must be dynamically written
- Must rewrite every URL by using encodeURL() method of HTTPResponse
- The string returned by the methods will have the session ID appended
- Works even if cookies are disabled by the browser. Server determines if URL rewriting is required or not

```
HttpSession session = request.getSession();
out.println(“<html><body>”);
out.println(“<a href=\”” + response.encodeURL(“/TestServlet”) + “\”>click me</a>”);
                                    out.println(“</body></html>”);
```

```
<session-config>
  <session-timeout>300</session-timeout>    300 MINUTES
</session-config>

<login-config>
   <auth-method>FORM</auth-method>
   <form-login-config>
      <form-login-page>/login.jsp</form-login-page>
      <form-error-page>/error.jsp</form-error-page>
   </form-login-config>
</login-config>
```

<welcome-file-list>
Used to define welcome files

<listener>
Used to define listener

<context-param>
Used to define Context parameters

<servlet>
Used to define servlet name and servlet class

<servlet-mapping>
Used to map the url to a servlet

<session-config>
Used to define session configuration

<login-config>
Used to define the type of authentication used in the web application

SUMMARY

Servlets

SUMMARY

In this lesson, you've learned to:

- Redirection
- Attributes
- Session Management