OBJECT™ TECHNOLOGIES

Servlet II

What will be covered

OBJECT TECHNOLOGIES

- Handling request parameters
- Servlet with JDBC
- Config and Context parameters
- Request dispatching
- Request redirection

Handling request parameters

OBJECT TECHNOLOGIES

- There is always the need to pass some information from your browser to web server and ultimately to your backend program.
- The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.
- The GET method sends the encoded user information appended to the page request URL and separated by the ? (question mark) symbol.
- A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) in the URL it sends it as embdded in the request body.

Handling request parameters

OBJECT TECHNOLOGIES

- One of the nice features of Java servlets is that all of this form parsing is handled automatically.
- **getParameter()** method of the HttpServletRequest is used to get this data by supplying the parameter name as an argument.
- Note that parameter names are case sensitive.
- This method is irrespective of the request method .i.e. it works for get as well as post method.
- The return value is a String corresponding to the decoded value of the first occurrence of that parameter name.
- An empty String is returned if the parameter exists but has no value, and null is returned if there was no such parameter.
- If the parameter could potentially have more than one value, as in the example above, you should call **getParameterValues()** instead of getParameter(). This returns an array of strings.
- Finally, although in real applications your servlets probably have a specific set of parameter names they are looking for, for debugging purposes it is sometimes useful to get a full list. Use **getParameterNames()** for this, which returns an Enumeration, each entry of which can be cast to a String and used in a getParameter call.

Handling request parameters

OBJECT TECHNOLOGIES

```
<form action="http://localhost:8080/FirstWebApp/logincheck"
  method="post">
    Enter uid : <input type="text" name="uid" />
    <br/>
    Enter pwd : <input type="password" name="pwd" />
    <br/>
    <input type="submit" value="LOGIN" />
</form>
```

- Input field names are very important for sending the data to server side. In this case data will be sent like

  uid=<value entered by user>&pwd=<value entered by user>

Handling request parameters

OBJECT TECHNOLOGIES

```
protected void doPost(HttpServletRequest request,
        HttpServletResponse response)
                throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    String uid = request.getParameter("uid");
    String pwd = request.getParameter("pwd");
    if(uid.equals("bakul") && pwd.equals("bakul123"))
        out.println("<h2> Login successful </h2>");
    else
        out.println("<h3> Login failed </h3>");
}
```

- Please note : Parameter name passed to getParameter() method exactly matched with input field name in the HTML form.

Servlet with JDBC

OBJECT TECHNOLOGIES

- Whenever servlet needs to have commnication with database, servlet can use JDBC API.
- All the steps of JDBC will be executed in the servlet to fetch or update data in database.
- As we are aware first 2 steps of JDBC are performance heavy operation, instead of executing them in service method, we can get connection established in init() method of servlet.
- As init() method gets executed only once in life cycle of servlet, it is used for getting JDBC connection object created.
- doGet() and doPost() methods can create the statements and execute the statements for fetching and saving data.

Servlet with JDBC

OBJECT TECHNOLOGIES

- Servlet's init() method for creating connection

```
Connection con;
//overridden method
public void init(ServletConfig config) throws
                    ServletException {
    super.init(config);
    try //establish database connection
    {
        Class.forName("com.mysql.cj.jdbc.Driver");
        String jdbcurl = "jdbc:mysql://localhost:3306/shoppingdb"
                + "?serverTimezone=UTC";
        con = DriverManager.getConnection(jdbcurl, "root", "root");
        //gets displayed on server's console
        System.out.println("connection established");
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
```

Servlet with JDBC

OBJECT TECHNOLOGIES

Service method of servlet for retrieving data

```
protected void doPost(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
    // TODO Auto-generated method stub
    //primary logic
    PrintWriter out = response.getWriter();
    String uid = request.getParameter("uid");
    String pwd = request.getParameter("pwd");
    PreparedStatement ps = null;
    ResultSet rs = null;
    try
    {
        ps = con.prepareStatement("select * from users where u_id = ?"
                + " and password = ?");
        ps.setString(1, uid);
        ps.setString(2, pwd);
        rs = ps.executeQuery();
```

Servlet with JDBC

OBJECT TECHNOLOGIES

Resultset is iterated for finding the data accepted from user

```
boolean flag = false;
while(rs.next())
{
    if(rs.getString(1).equals(uid)&& rs.getString(2).equals(pwd)
    {
        flag = true;
        break;
    }
}
if(flag)
    out.print("<h2> Login successful </h2>");
else
    out.println("<h3> Login failed </h3>");
```

Config and Context parameters

OBJECT TECHNOLOGIES

- On top of request parameters sent by the client, servlet can retrieve information form other sources like config and context paramaters.
- These parameters are also in the form of key-value pairs.
- Config parameters are specifice for one servlets and can be accessed only by the same servlet whereas context parameters are shared between all servlets.
- If any specific content is modified from time to time, you can manage the Web application easily without modifing servlet through editing the value in for config paramters. Servlet need not be modified when data changes.
- If all servlets need access to a perticular information, this data can be configured as context parameter.

Config and Context parameters

OBJECT TECHNOLOGIES

- An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.
- Methods of ServletConfig interface
  - **public String getInitParameter(String name)**:Returns the parameter value for the specified parameter name.
  - **public Enumeration getInitParameterNames()**:Returns an enumeration of all the initialization parameter names.
  - **public String getServletName()**:Returns the name of the servlet.
  - **public ServletContext getServletContext()**:Returns an object of ServletContext.
- Config parameter can be written in web.xml file using <init-param> tag as a child of <servlet> tag or using @WebInitParam annotation

Config and Context parameters

```
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>DemoServlet</servlet-class>
<init-param>
<param-name>MyName</param-name>
<param-value>Chaitanya</param-value>
```

- Initializing servlets parameters is also possible with annotations which allows us to keep configuration and source code in the same place.

```
@WebInitParam(name="name", value="Not provided"),
@WebInitParam(name="email", value="Not provided")}))
public class UserServlet extends HttpServlet {
    // ...

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

Config and Context parameters

OBJECT

- getServletConfig() method of Servlet interface returns the object of ServletConfig which can be used to access config parameters

```
//config parameter - ServletConfig
ServletConfig sc = getServletConfig();
String mname = sc.getInitParameter("module");
```

- In the init() method of servlet Servletconfig object is readily available to access config parameters

```
public void init(ServletConfig servletConfig) throws ServletException{
    this.testParam = servletConfig.getInitParameter("test");
}
```

Config and Context parameters

OBJECT

- 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 which is a child tag of <web-app>

```
<context-param>
  <param-name>parametername</param-name>
  <param-value>parametervalue</param-value>
</context-param>
```

- getServletContext() method of GenericServlet class returns the object of ServletContext as well as getServletContext() method of ServletConfig interface returns the object of ServletContext.
- In the service method of servlet, ServletContext object can be created like :

Config and Context parameters

OBJECT

```
//creating ServletContext object
ServletContext context=getServletContext();
```

- In the init() method ServletContext object can be created like :

```
//We can get the ServletContext object from ServletConfig object
ServletContext application=getServletConfig().getServletContext();
```

- To access context parameters

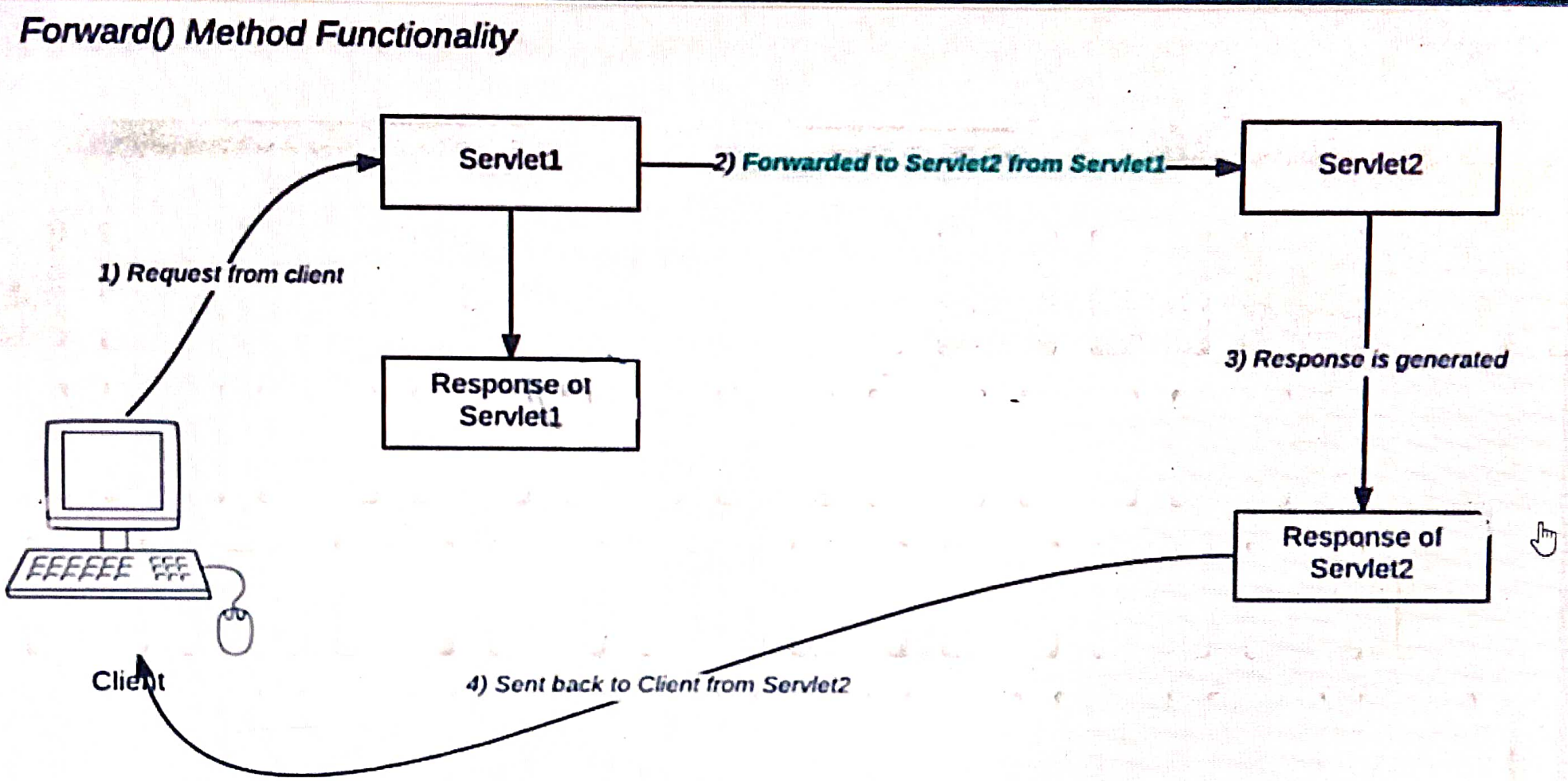
```
//context parameters - SevletContext
ServletContext ctx = getServletContext();
String center = ctx.getInitParameter("center");
```

Request dispatching

OBJECT

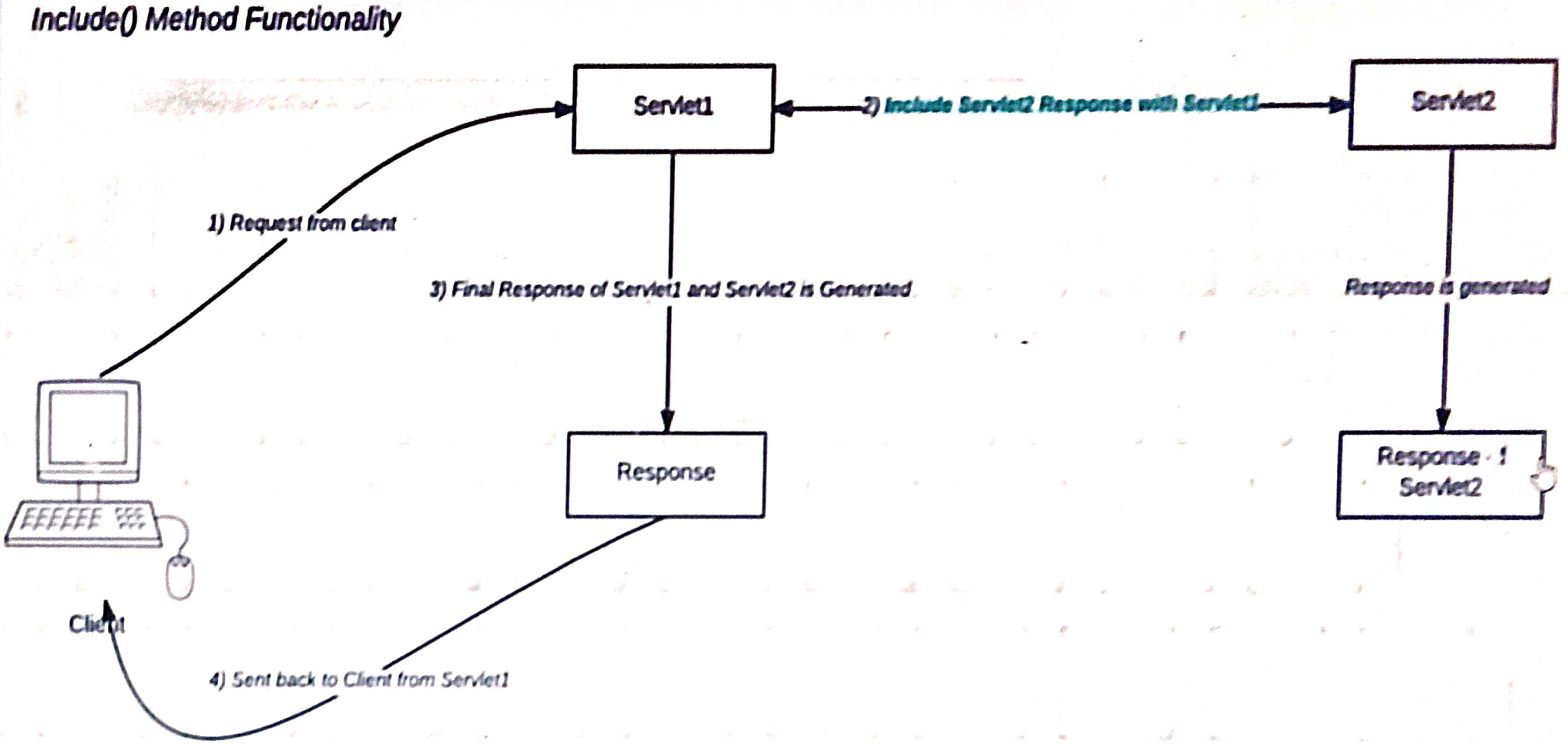
- The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also.
- It is one of the way of servlet collaboration.
- This helps when the request processing or the response generation has to be shared between multiple servlets.
- With the RequestDispatcher object we can -
  - Forward the current request from one servlet to another servlet.
  - Include the response from another servlet in the current servlet.
- These methods will accept an object of the Servlet request and response interface.
- The main difference is that when a programmer uses forward, the control is transferred to the next Servlet or JSP the application is calling while in the case of include, the control retains with the current Servlet and it just includes the processing done by the calling of Servlet or the JSP.

Request dispatching - forward



- Note : Once the request is forwarded from one servlet to another, response is generated from forwarded servlet and sent back to client. Client is not aware about which resource has actually generated the response because for client request URL is not modified. Client makes request for servlet1 and assumes response generated from servlet1

Request dispatching - include



- Note : Servlet1 includes the response generated by servlet2 and final response of both together is sent back to the client. Client is unaware as there is no change in request URL. Client makes request to servlet1 and servlet1 internally takes help from servlet2

Request dispatching

OBJECT

- To create an object of RequestDispatcher
  - By calling the getRequestDispatcher() method of ServletRequest.
  - By calling the getRequestDispatcher() method of ServletContext.
- Using forward :

```
RequestDispatcher rd = request.getRequestDispatcher("/home");
rd.forward(request, response);
```

- Using include :

```
RequestDispatcher rd = request.getRequestDispatcher("/login.jsp");
rd.forward(request, response);
```

- Note here request object is used for getting RequestDispatcher object.

Request redirection

OBJECT

- The sendRedirect() method of HttpServletResponse interface can be used to redirect response to another resource, it may be servlet, jsp or html file.
- It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.
- In Java web development, to redirect the users to another page, you can call the following method on the HttpServletResponse object response:

```
response.sendRedirect(String location)
```

- Technically, the server sends a HTTP status code 302 (Moved Temporarily) to the client. Then the client performs URL redirection to the specified location. The location in the sendRedirect() method can be a relative path or a completely different URL in absolute path.

Request redirection

OBJECT

response.sendRedirect("login.jsp");

- This statement redirects the client to the login.jsp page relative to the application's context path.

response.sendRedirect("/login.jsp");

- But this redirects to a page relative to the server's context root:

response.sendRedirect("https://www.yoursite.com");

- And the above statement redirects the client to a different website.
- this method throws IllegalStateException if it is invoked after the response has already been committed. For example:

```
PrintWriter writer = response.getWriter();
writer.println("One more thing...");
writer.close();

String location = "https://www.codejava.net";
response.sendRedirect(location);
```

Servlet II

Now we are aware about creating simple servlet and it's life cycle. Now we will go in more details about how to access request parameters, how to use more than one servlets for response generation, servlet redirection etc.

Handling Request Parameters

Each access to a servlet can have any number of request parameters associated with it. These parameters are typically name/value pairs that tell the servlet any extra information it needs to handle the request.

From the request object we can read the parameter submitted by the user?s browser either through an HTTP GET or POST method.

1. public String ServletRequest.getParameter(String name)

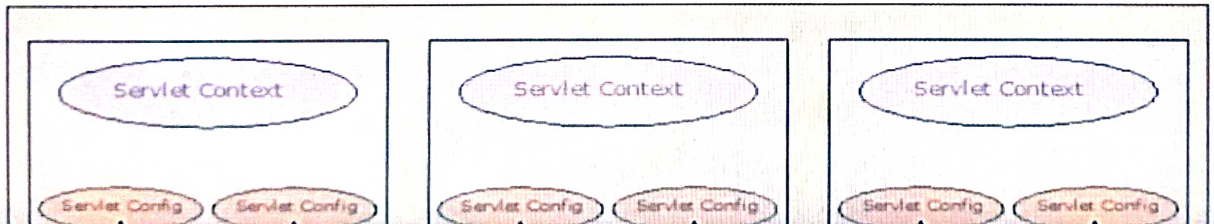
2. public String[] ServletRequest.getParameterValues(String name)

getParameter() returns the value of the named parameter as a String or null if the parameter was not specified. The value is guaranteed to be in its normal, decoded form. If the parameter has multiple values, the value returned is server-dependent. If there is any chance a parameter could have more than one value, you should use the getParameterValues() method instead. This method returns all the values of the named parameter as an array of String objects or null if the parameter was not specified. A single value is returned in an array of length 1.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    String uid = request.getParameter("uid");
    String pwd = request.getParameter("pwd");
}
```

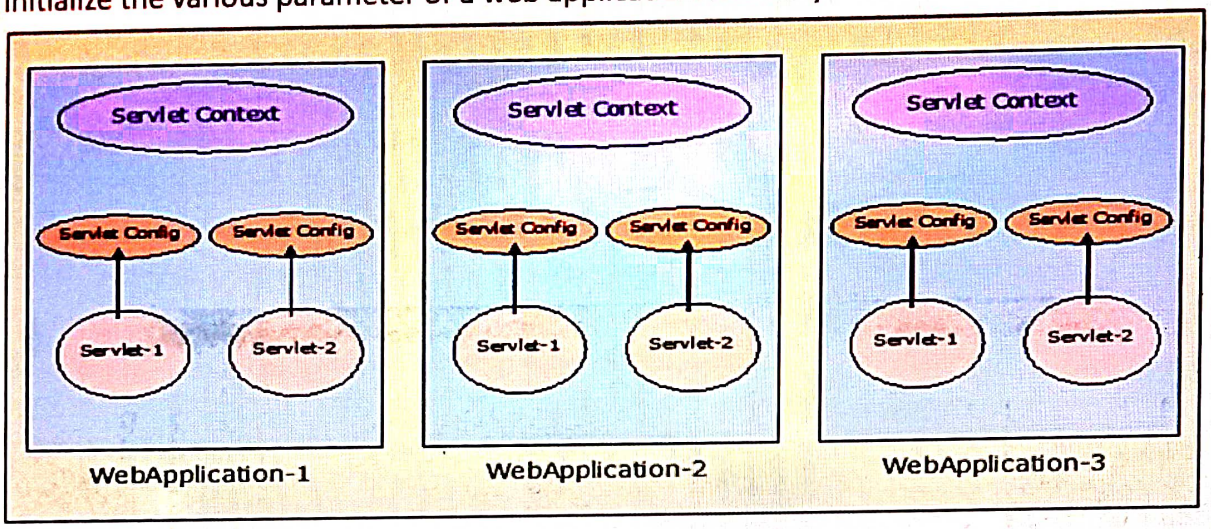
Config and Context Parameters

Both ServletContext and ServletConfig are basically the configuration objects which are used by the servlet container to initialize the various parameter of a web application. But they have some difference in terms of scope and availability.



Config and Context Parameters

Both ServletContext and ServletConfig are basically the configuration objects which are used by the servlet container to initialize the various parameter of a web application. But they have some difference in terms of scope and availability.



Web Container

What is ServletConfig?

ServletConfig is an interface in the Servlet API and is used to initialize a single servlet in a web application by the servlet container. Inside deployment descriptor known as web.xml, developers define the servlet initialization parameter related to that servlet. The particulars are declared in the <init-param /> tag in a name-value pair. Following is the signature of the ServletConfig interface:

public interface ServletConfig

1. For each Servlet under execution, a ServletConfig object is created by the servlet container and is used by the programmer to read the servlet specific data declared in the web.xml.
2. ServletConfig is an interface from the javax.servlet package.
3. The getServletConfig() method of the GenericServlet class returns an object of the ServletConfig.
4. The ServletConfig object created by the container for a specific servlet cannot read the other servlet's config parameters
5. The ServletConfig object is specific for a Servlet. Another way to say is that, if 100 servlet objects are being created in the container, 100 ServletConfig objects are also created implicitly for communication with the Servlet.
6. When the container destroys the Servlet object, the corresponding ServletConfig object is also destroyed.
7. Config parameters can be written in either web.xml or in the form of annotation @WenInitParam

3. The getServletConfig() method of the GenericServlet class returns an object of the ServletConfig.
4. The ServletConfig object created by the container for a specific servlet cannot read the other servlet's config parameters
5. The ServletConfig object is specific for a Servlet. Another way to say is that, if 100 servlet objects are being created in the container, 100 ServletConfig objects are also created implicitly for communication with the Servlet.
6. When the container destroys the Servlet object, the corresponding ServletConfig object is also destroyed.
7. Config parameters can be written in either web.xml or in the form of annotation @WenInitParam

What is ServletContext?

ServletContext is even more important than the ServletConfig and its one per web application. This object is common for all the servlet and developers use this object to get the detail of the whole web application or the execution environment. An important thing to remember is that the ServletContext represents a web application in a single JVM. Following is the signature of the ServletContext interface:

```
public interface ServletContext
```

ServletContext is an interface from the javax.servlet package and its object can be obtained from the getServletContext() method of the ServletContext i.e.

```
ServletConfig configObj = getServletConfig();
ServletContext contextObj = configObj.getServletContext();
```

The ServletContext object comes into existence when the container is started and is destroyed when the container stops its execution. The ServletContext object is common to all servlets and other components ServletContext parameters are defined outside the Servlet tag but, inside the <context-param /> tag

```
<context-param>
   <param-name>globalVariable</param-name>
   <param-value>javacodegeek.com</param-value>
</context-param>
```

In servlet, we get this parameter value by the getInitParameter("param-name") method i.e.

```
String value = getServletContext().getInitParameter("globalVariable");
```

Difference between Servletconfig and ServletContext

ServletConfig	ServletContext

Difference between Servletconfig and ServletContext

ServletConfig	ServletContext
ServletConfig object represent a single servlet	ServletContext object represent the whole application running on a particular JVM and is common for all the servlets.
It is like a local parameter associated with the particular servlet	It is like a global parameter associated with the whole application and is shareable by all servlets
It is a name-value pair defined inside the servlet sectionof the web.xml file and it has a servlet wide scope	ServletContext has an application-wide scope and is defined outside of the servlet tag
getServletConfig() method is used to get the ServletConfig object	getServletContext() method is used to get the ServletContext object
For e.g.: Shopping cart of a user is a specific to a particular user and developers can use the getServletConfig() object	To get the MIME type of a file or an application, the session related information is stored using the ServletContext object
Each Servlet comes with a separate ServletConfig object	There will be only one ServletContext object available and accessible by all servlets

Request Dispatching

A RequestDispatcher is a technique that allows for 'including' content in a request/response or 'forwarding' a request/response to a resource. As a typical example, a servlet can use a RequestDispatcher to include or forward a request/response to a JSP. In Model-View-Controller programming in Java, a servlet typically serves as the 'controller'. The controller basically contains or references code to perform particular actions, and it decides which 'view' to send the user to. In Java, the view is typically a JSP. Thus, a RequestDispatcher performs a very important role in Java MVC architecture since it can serve as the mechanism for the 'controller' (servlet) to pass the user to the 'view' (JSP).

The RequestDispatcher interface defines an object that receives the request from client and dispatches it to the resource(such as servlet, JSP, HTML file). This interface has following two methods:

1. public void forward(ServletRequest request, ServletResponse response):

   It forwards the request from one servlet to another resource (such as servlet, JSP, HTML file).

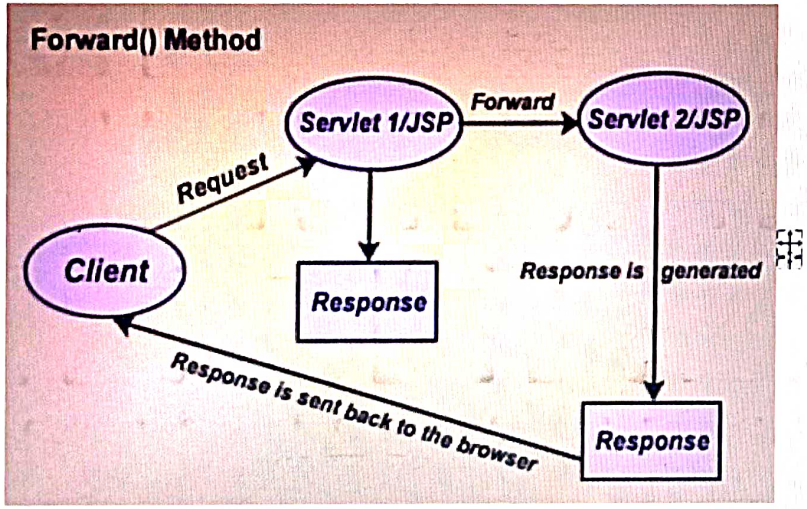
Request Dispatching

A RequestDispatcher is a technique that allows for 'including' content in a request/response or 'forwarding' a request/response to a resource. As a typical example, a servlet can use a RequestDispatcher to include or forward a request/response to a JSP. In Model-View-Controller programming in Java, a servlet typically serves as the 'controller'. The controller basically contains or references code to perform particular actions, and it decides which 'view' to send the user to. In Java, the view is typically a JSP. Thus, a RequestDispatcher performs a very important role in Java MVC architecture since it can serve as the mechanism for the 'controller' (servlet) to pass the user to the 'view' (JSP).

The RequestDispatcher interface defines an object that receives the request from client and dispatches it to the resource(such as servlet, JSP, HTML file). This interface has following two methods:

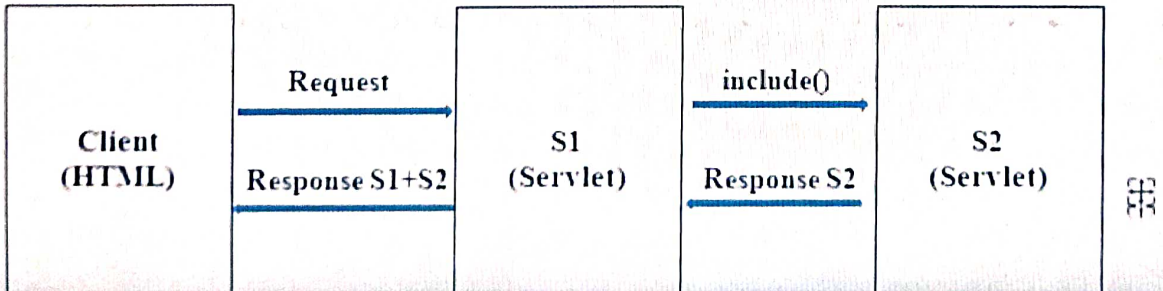
1. public void forward(ServletRequest request, ServletResponse response):

   It forwards the request from one servlet to another resource (such as servlet, JSP, HTML file).



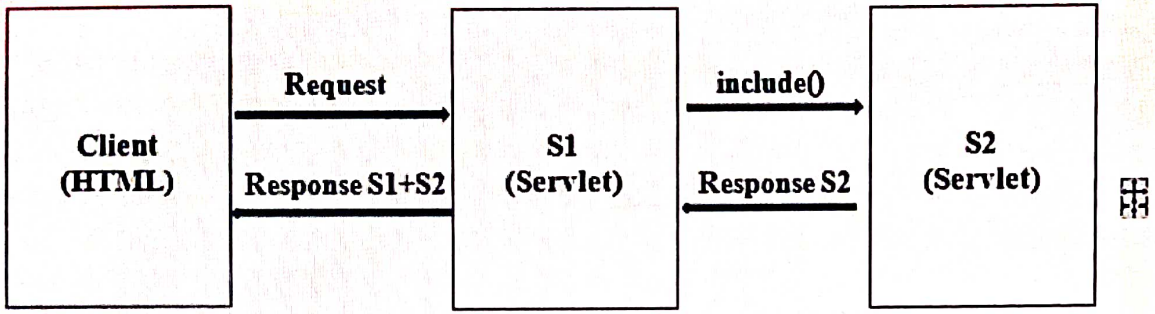
2. public void include(ServletRequest request, ServletResponse response):

   It includes the content of the resource(such as servlet, JSP, HTML file) in the response.



2. public void include(ServletRequest request, ServletResponse response):

It includes the content of the resource(such as servlet, JSP, HTML file) in the response.



RequestDispatcher – include() method

Note : The key difference between the two is the fact that the forward method will close the output stream after it has been invoked, whereas the include method leaves the output stream open.

Example code for forwarding request to /home after successful login :

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String uid = request.getParameter("uid");
    String pwd = request.getParameter("pwd");
     //create and execute query
    PreparedStatement ps = null;
    ResultSet rs= null;
    try  {
      ps = con.prepareStatement("select * from users where u_id=? and password = ?");
      ps.setString(1, uid);
      ps.setString(2, pwd);
      rs = ps.executeQuery();   //record pointer - zeroth record
      //login is successful
      if(rs.next())  { // record pointer - first record
        //generate response using home servlet
        RequestDispatcher rd = request.getRequestDispatcher("/home");
        rd.forward(request, response);
      }
      else {
        response.sendRedirect("/FirstWeb/login.html");
```

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String uid = request.getParameter("uid");
    String pwd = request.getParameter("pwd");
     //create and execute query
    PreparedStatement ps = null;
    ResultSet rs= null;
    try  {
       ps = con.prepareStatement("select * from users where u_id=? and password = ?");
       ps.setString(1, uid);
       ps.setString(2, pwd);
       rs = ps.executeQuery();   //record pointer - zeroth record
       //login is successful
       if(rs.next())  { // record pointer - first record
          //generate response using home servlet
          RequestDispatcher rd = request.getRequestDispatcher("/home");
          rd.forward(request, response);
       }
       else {
           response.sendRedirect("/FirstWeb/login.html");
       }
    }
    catch(Exception e) {
       e.printStackTrace();
    }
    finally {
       try {
          rs.close();
          ps.close();
       } catch (SQLException e) {
           e.printStackTrace();
       }
    }
}
```

Example for include method where /home includes header and footer

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    PrintWriter out = response.getWriter();
```

Example for include method where /home includes header and footer

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    RequestDispatcher rd = request.getRequestDispatcher("/header");
    rd.include(request, response);
    out.println("<hr/> <br/> Welcome to my web app <br/> <hr/>");
    RequestDispatcher rd1 = request.getRequestDispatcher("/footer");
    rd1.include(request, response);
  }
```

Request Redirection

Request redirection is a technique which redirects the user from one URL to another. HTTP has a special kind of response, called a HTTP redirect, for this operation.

Request redirection is useful when:

- Temporary redirects during site maintenance or downtime
- Permanent redirects to preserve existing links/bookmarks after changing the site's URLs, progress pages when uploading a file, etc.

The sendRedirect() method

This method of HttpServletResponse interface can be used to redirect the response to another resource i.e. it may be a Servlet, JSP or HTML file. It works on the client side because it uses the address bar of the browser to make another request. Hence, it can work inside and outside the server.

Method signature is as follows :

```
public void sendRedirect(String url) throws IOException
```

To do this, we use the sendRedirect method belonging to the HttpServletResponse interface:

```
response.sendRedirect("https://www.google.com");
```

The drawback is that we lose the original request along with its parameters and attributes.

Which One to Use?

Use forward when:

1. We want to pass control to a resource in the same web app

Which One to Use?

Use forward when:

1. We want to pass control to a resource in the same web app
2. We want to preserve the data attributes in the original request

Use sendRedirect when:

1. If we want to transfer control to another domain, then we'd use sendRedirect.
2. Our request is changing server state (e.g. insert/update/delete to database)

Assignments

1. Accept login information from the user, write a servlet to check whether login is successful or failed and generate dynamic message.
   a. by comparing with constant strings
   b. by checking in database
2. Accept information about the user from the html page like userid, password, first name, last name, email id, contact no. Write a servlet which reads all request parameters and insert it as record in database table.
3. Create a web page displaying hyperlinks about different DAC modules. Clicking on each hyperlink should display the content of respective module. Use seperate servlet of each module (Use init parameter as subject and context parameter as institution_name for servlets)
4. Display search link which makes request to google search home page
5. Modify login validation by forwarding the request to succes servlet or error servlet depending on login check
6. Modify assignment no 5 to add common header and footer
7. After successful login, success servlet should show all available product categories to the user in the form of hyerlinks.