

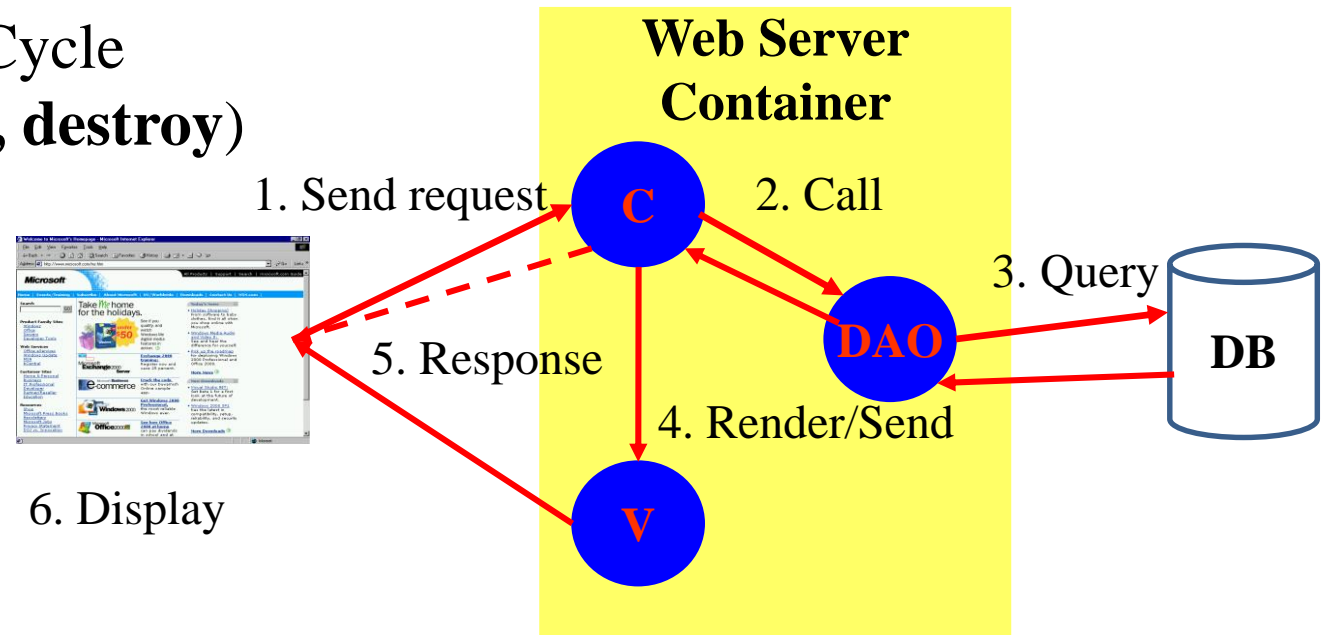
Web Applications & Web Containers

Web Applications **The Web Container Model**

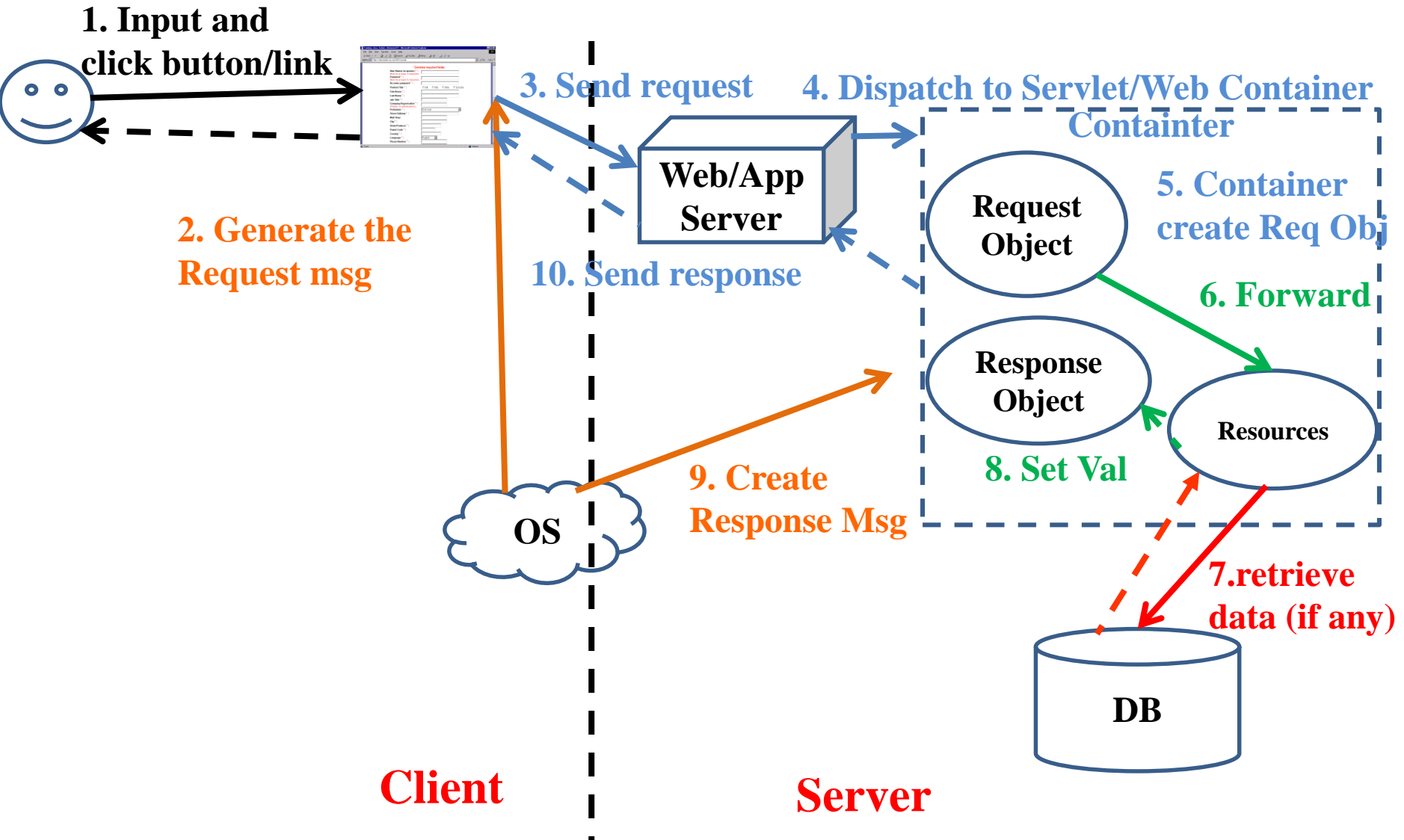
#Servlet #Tomcat #Deploy
#Dispatcher #Scope #Video

Review

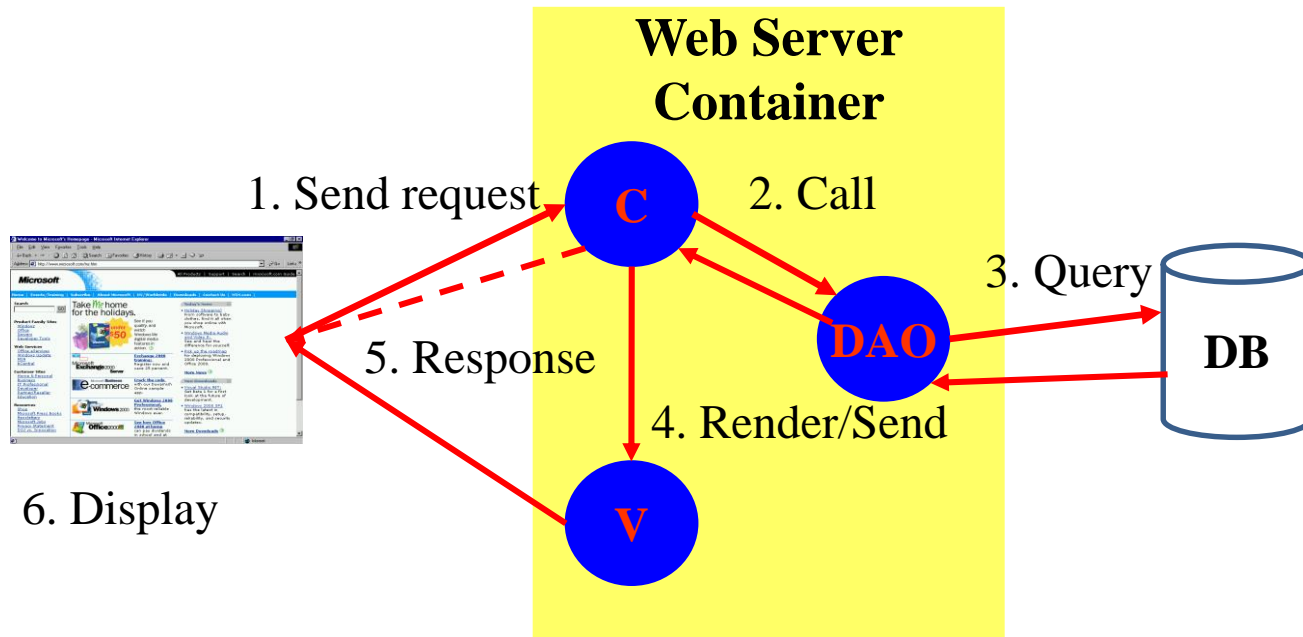
- **How to build the simple web site using html and servlet?**
 - Break down structure component in building web application
- **Some concepts**
 - Servlet vs. Java class, Parameter vs. Variable
 - Form Parameters
 - Http Protocol
 - HTTP Methods: **GET, POST, ...**
 - Servlet Life Cycle (**init, service, destroy**)
 - JDBC



Review



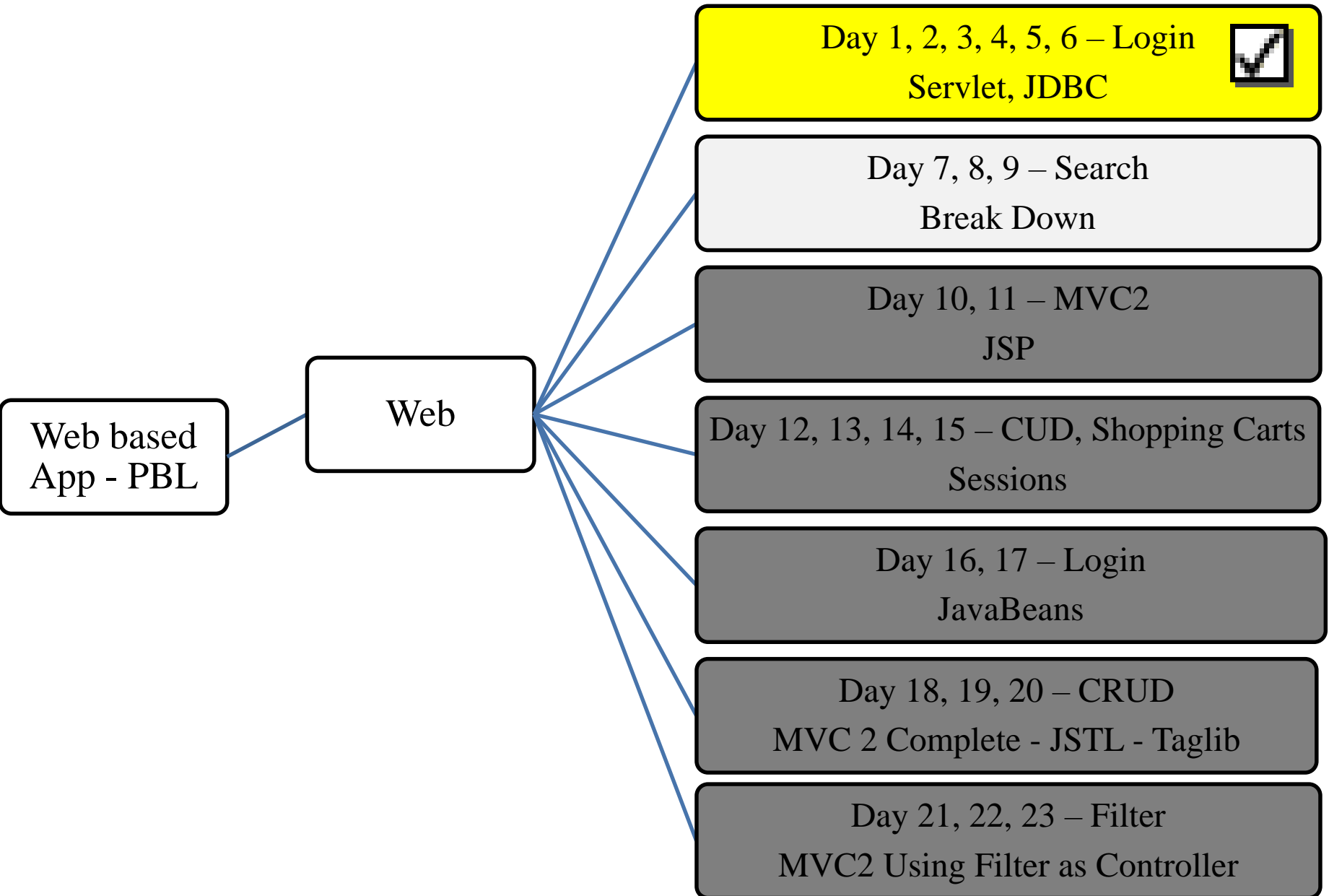
Review



Objectives

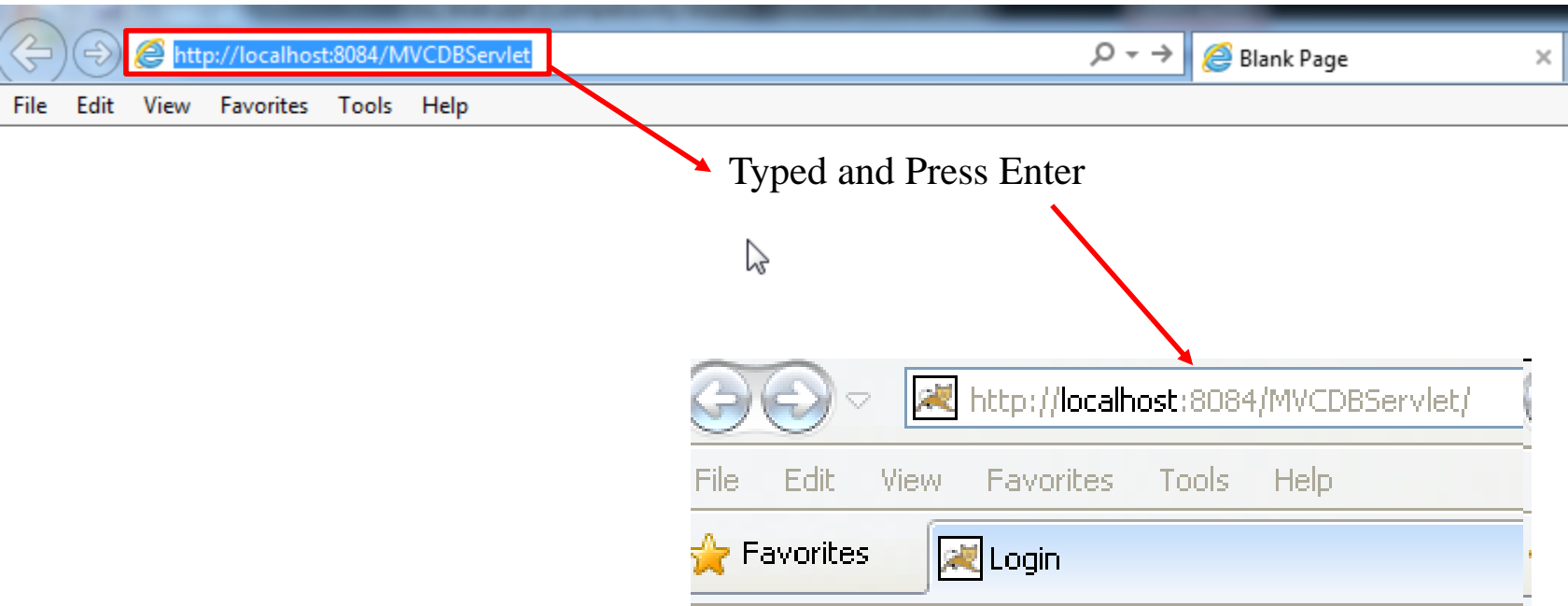
- **How to deploy the Web Application to Web Server without using Netbeans/ Eclipse tools?**
 - Web applications Structure
 - Request Parameters vs. Context Parameters vs. Config/Servlet Parameters
 - Application Segments vs. Scope
- **How to transfer from resources to others with/without data/objects?**
 - Attributes vs. Parameters vs. Variables
 - Redirect vs. RequestDispatcher

Objectives



Deploy Application

Expectation



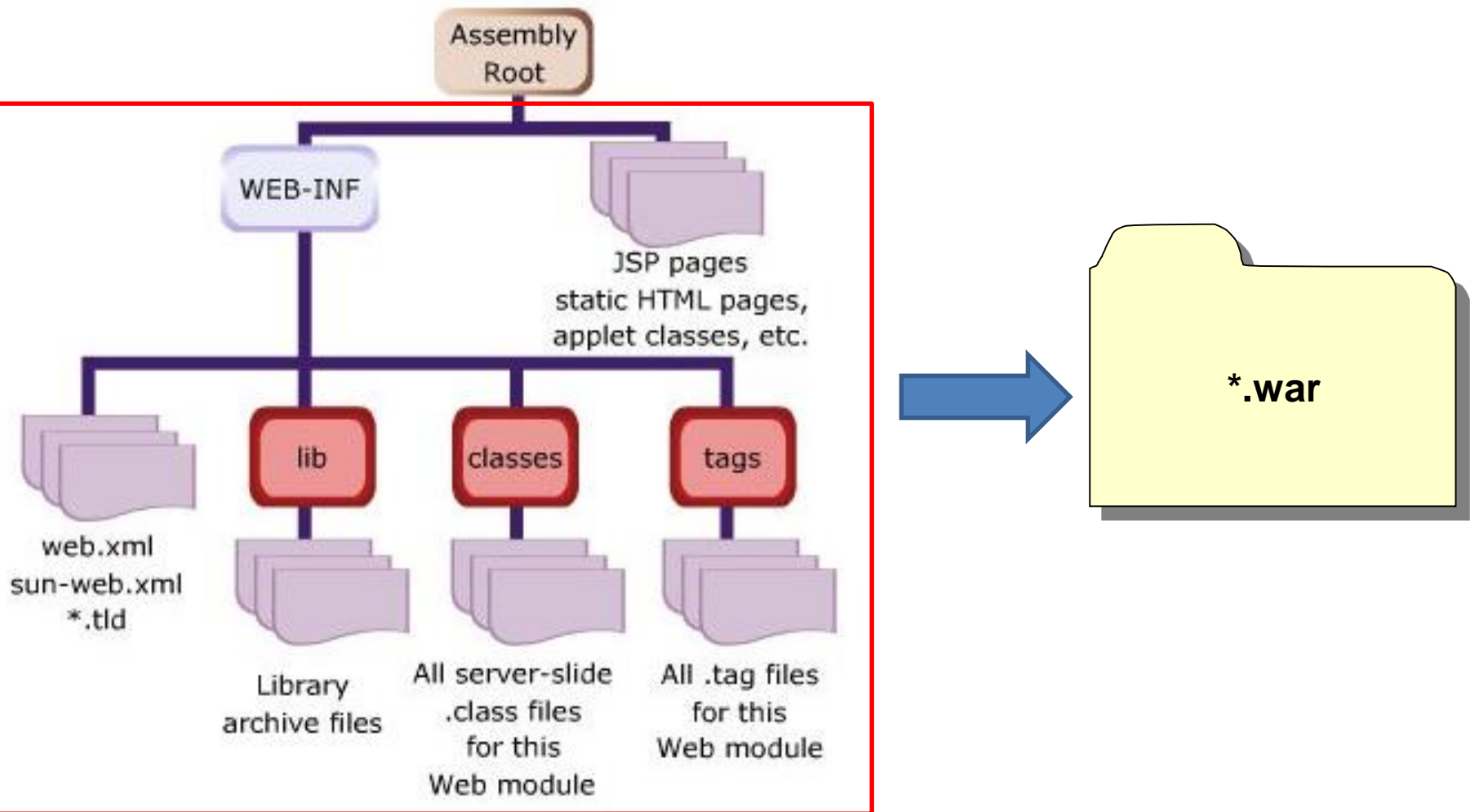
Login Page

Username

Password

Web Applications

File and Directory Structure



Above structure is packaged into *.war (Web (Application) ARchive) file to deploy on Web Server

Web Applications

File and Directory Structure

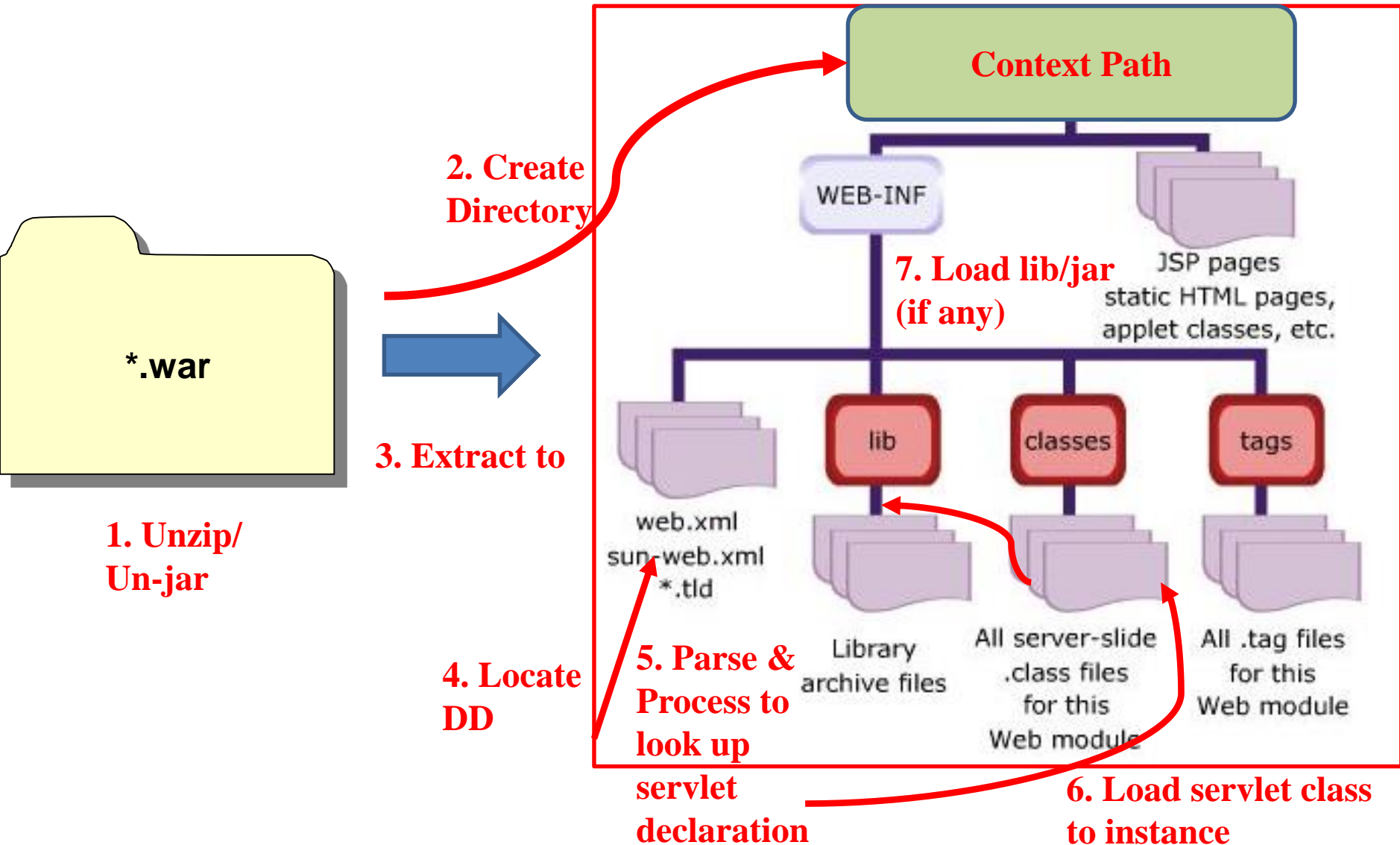
- **/WEB-INF/classes** – for **classes that exist** as separate Java classes (*not* packaged within JAR files). These might be servlets or other support classes.
- **/WEB-INF/lib** – for JAR file. These can contain anything at all – the main servlets for your application, supporting classes that connect to databases – whatever.
- **/WEB-INF** itself is the home for an absolutely crucial file called **web.xml**, the **web deployment descriptor** file.
- **2 special rules** apply to files within the **/WEB-INF** directory
 - Direct client access should be disallowed with an HTTP 404 code
 - The **order** of class **loading** the java classes in the **/WEB-INF/classes** directory should be **loaded before** classes resident in **jar files** in the **/WEB-INF/lib** directory

Web Applications

File and Directory Structure

- A Place for Everything and Everything in Its Place.
 - On Tomcat Server, it locates at **CATALINA_HOME/webapps**
 - **Execute:** **<http://host:port/webappcontext/resourceIneed>**
- Construct the file and directory structure of a Web **Application** that may **contain**:
 - Static content,
 - JSP pages,
 - Servlet classes,
 - The deployment descriptor,
 - Tag libraries,
 - JAR files and Java class files;
 - and describe how to protect resource file from HTTP access.

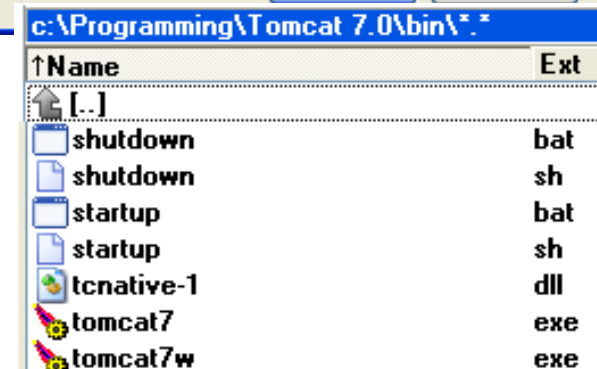
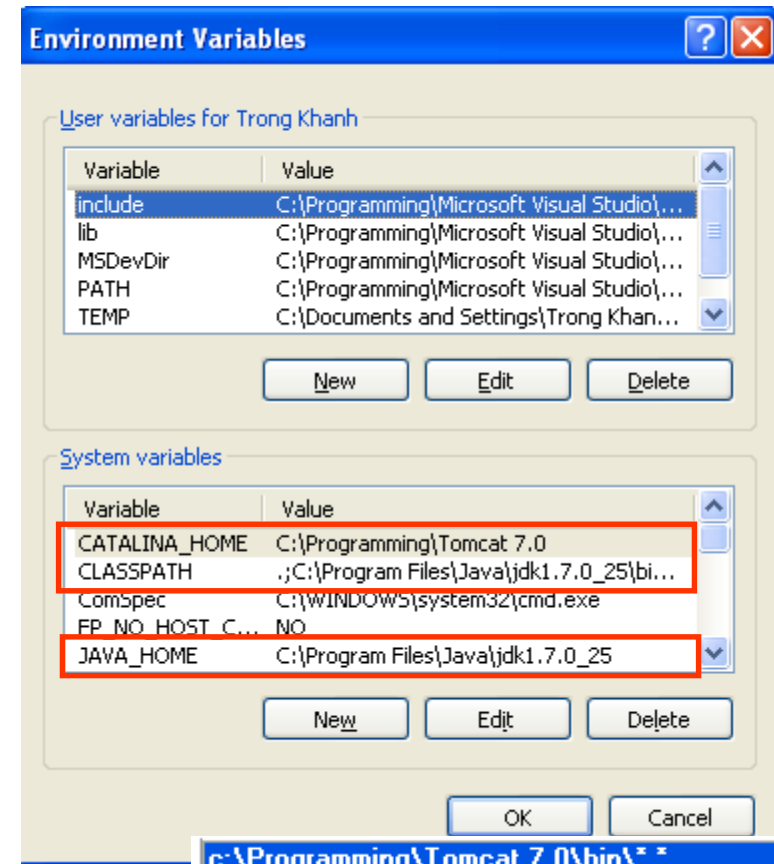
Web Applications Deploy Mechanism



Web Applications

Manual Deploying

- Setup the environment for JAVA and TOMCAT
 - **In Windows OS:** click Properties of Computer, choose “Advanced System Setting”, choose Advanced, Click “Environment Variables”, to set following environment variables
- Go to the **Installed_Tomcat\bin** directory, click **startup.bat** or **tomcat7w.exe**



Web Applications

Manual Deploying

```

Tomcat
INFO: Deploying web application directory C:\Programming\Tomcat 7.0\webapps\docs
thg 9 21, 2013 8:27:33 CH org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory C:\Programming\Tomcat 7.0\webapps\exam
ples
thg 9 21, 2013 8:27:33 CH org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory C:\Programming\Tomcat 7.0\webapps\host
-manager
thg 9 21, 2013 8:27:33 CH org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory C:\Programming\Tomcat 7.0\webapps\mana
ger
thg 9 21, 2013 8:27:33 CH org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory C:\Programming\Tomcat 7.0\webapps\ROOT
thg 9 21, 2013 8:27:33 CH org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-apr-8080"]
thg 9 21, 2013 8:27:33 CH org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-apr-8009"]
thg 9 21, 2013 8:27:33 CH org.apache.catalina.startup.Catalina start
INFO: Server startup in 619 ms
  
```

c:\Programming\Tomcat 7.0\webapps*. *		
↑Name	Ext	Size
↑[.]		<DIR>
↑[AJDay1_7]		<DIR>
↑[docs]		<DIR>
↑[examples]		<DIR>
↑[host-manager]		<DIR>
↑[manager]		<DIR>
↑[ROOT]		<DIR>
↑AJDay1_7	war	25.9

c:\Programming\Tomcat 7.0*. *		
↑Name	Ext	Size
↑[.]		<DIR>
↑[bin]		<DIR>
↑[conf]		<DIR>
↑[lib]		<DIR>
↑[logs]		<DIR>
↑[temp]		<DIR>
↑[webapps]		<DIR>
↑[work]		<DIR>
↑LICENSE		57.862
↑NOTICE		1.228
↑RELEASE-NOTES		9.054
↑RUNNING	txt	16.742

- Testing on web browser
- Delete the war file and the directory to undeploy application
- Press Ctrl + C to stop server

The Web Container Model

The Servlet Container

- Is a **compiler**, executable program.
- Is the **intermediary** between the Web server and the servlets in the container.
- **Loads, initializes, and executes** the servlets.
 - When a **request arrives**, the container **maps the request to a servlet**, **translates the request**, and then **passes the request** to the servlet.
 - The servlet **processes the request** and **produces a response**.
 - The container **translates the response** into the **network format**, then **sends the response back** to the Web server.
- Is designed to perform well while **serving large** numbers of **requests**.
- Can hold any number of active servlets, filters, and listeners.
- Both the container and the objects in the container are **multithreaded**.
 - The container creates and manages threads as necessary to handle incoming requests.
 - The container handles multiple requests concurrently, and more than one thread may enter an object at a time.
 - Therefore, each object within a container must be threadsafe.

The Web Container Model

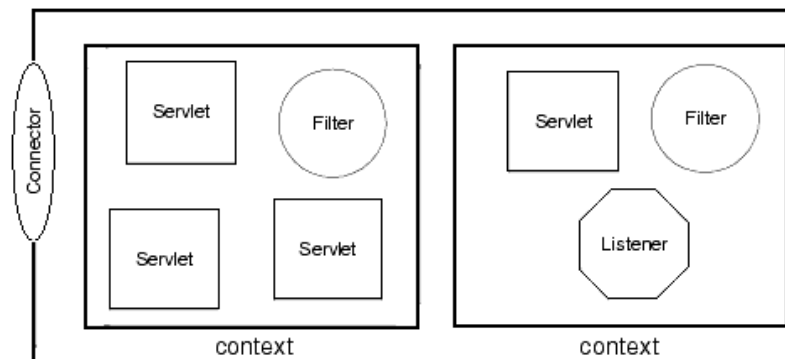
The Servlet Container

- Fortunately,
 - We are a web *component* developer, not a *web container* developer.
 - So we can take for granted much of what is built into the web container.
- We are a **consumer** of what the web container provides, and
- We have to understand the infrastructure only insofar as it affects our own business applications

The Web Container Model

The ServletContext

- Is considered as a **memory segment** that
 - Collects all methods that are used for particular Web application in server side
 - Support to interact with Servlet container
 - Stores some object in server side that all web's component can access
 - Exists from the application has been deployed to undeployed (or server is crashed)
- The container uses a *context* to
 - **Group related** components.
 - **Share** data in easily.
 - **Provide** a set of **services** for the web application to work with the container
- **Each context** usually corresponds to a **distinct** Web application.



The Web Container Model

The ServletContext – Example

- The directory structure below describes **two contexts**, one named **day1** and one named **day2**. The day2 context contains a static HTML page, intro.html.

webapps

\day1

\WEB-INF

web.xml

\day2

intro.html

\WEB-INF

web.xml

The Web Container Model

The ServletContext – Initialization Parameters

- Providing some fundamental information available to all the dynamic resources (servlets, JSP) within the web application is allowed by
 - Using servlet **initialization parameters** in the **deployment descriptor** with the **getInitParameter(String parName)** method to provide **initialization information for servlets**
 - The servlet initialization parameters is accessible only from its containing servlet
- Setting up the Deployment Descriptor

```
<web-app>
  <context-param>
    <param-name>parName</param-name>
    <param-value>parValue</param-value>
  </context-param>
  ...
</web-app>
```

The Web Container Model

The ServletContext – Initialization Parameters

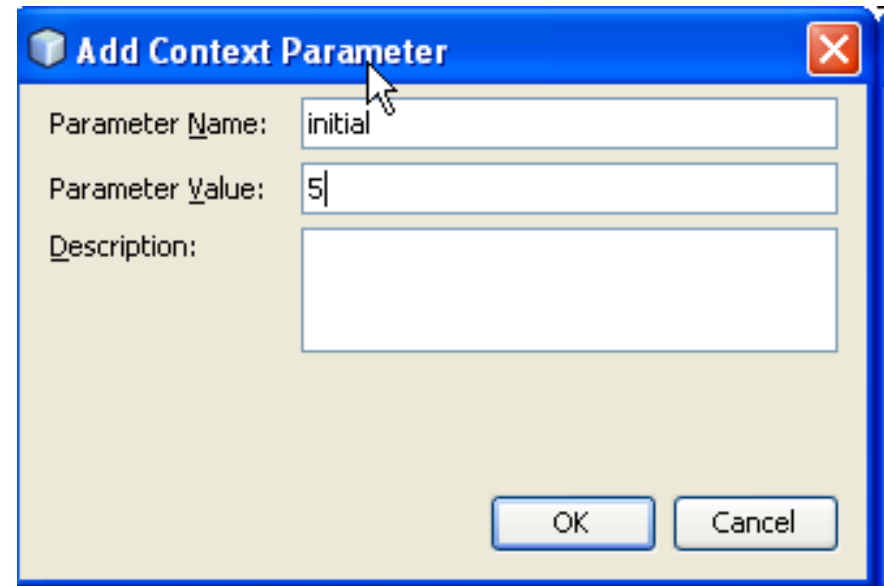
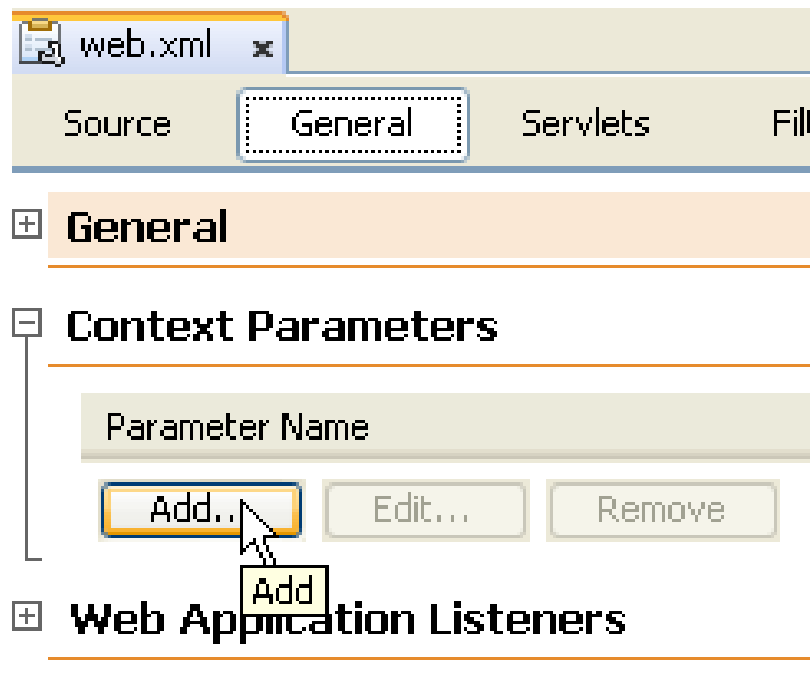
- Example
 - Building the web application have the counter function that allows the web site can account the number of accessed users
 - The application's GUI should be same as



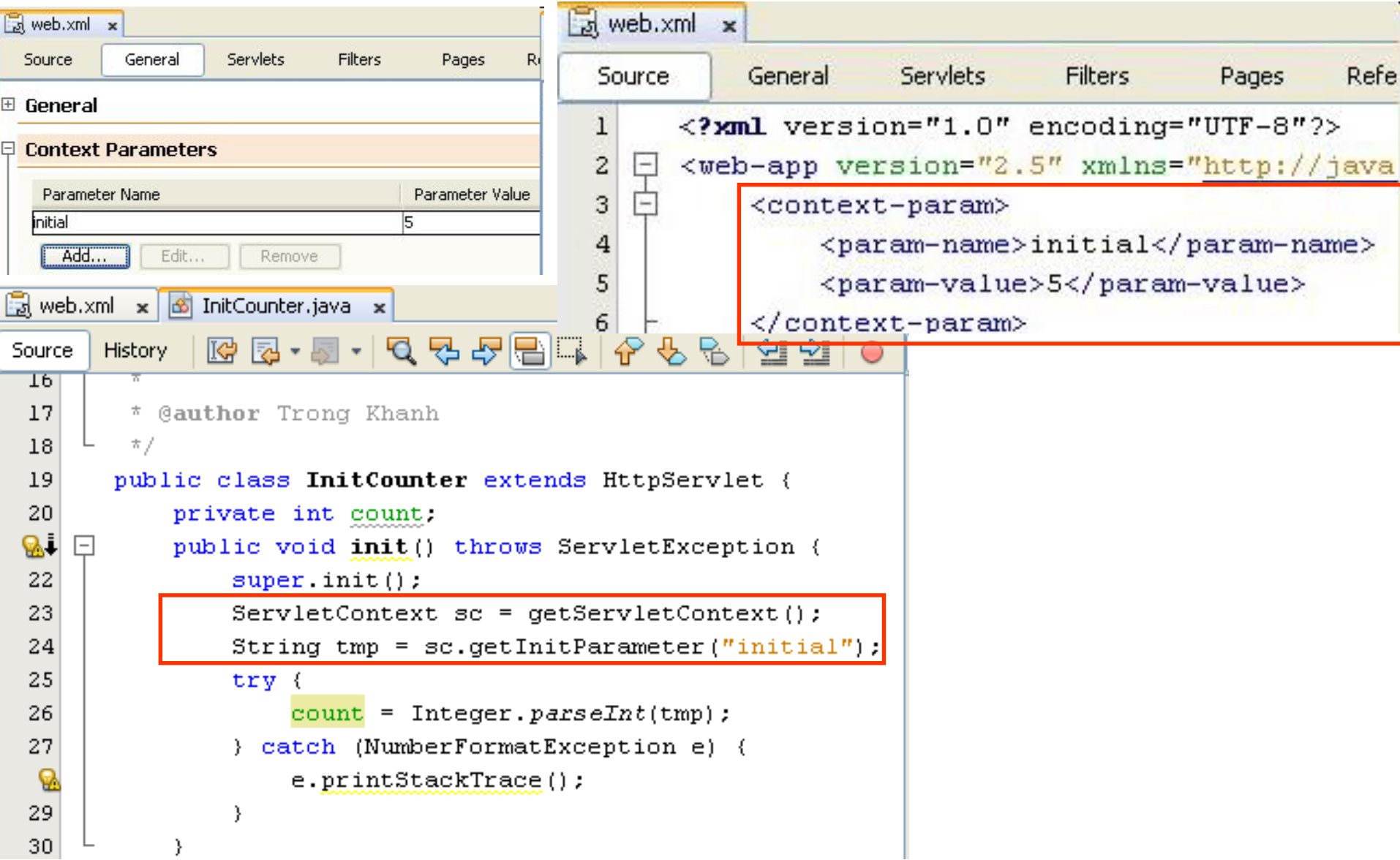
The ServletContext – Initialization Parameters

- Writing Code to Retrieve ServletContext Initialization Parameters

```
ServletContext sc = getServletContext();
String var = sc.getInitParameter("parName");
```



The ServletContext – Initialization Parameters



The image shows an IDE with two windows. The top window displays the `web.xml` file in the 'Source' tab. The bottom window displays the `InitCounter.java` file in the 'Source' tab.

web.xml Configuration:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app version="2.5" xmlns="http://java
3  <context-param>
4      <param-name>initial</param-name>
5      <param-value>5</param-value>
6  </context-param>
  
```

InitCounter.java Code:

```

16  *
17  * @author Trong Khanh
18  */
19  public class InitCounter extends HttpServlet {
20      private int count;
21      public void init() throws ServletException {
22          super.init();
23          ServletContext sc = getServletContext();
24          String tmp = sc.getInitParameter("initial");
25          try {
26              count = Integer.parseInt(tmp);
27          } catch (NumberFormatException e) {
28              e.printStackTrace();
29          }
30      }
  
```

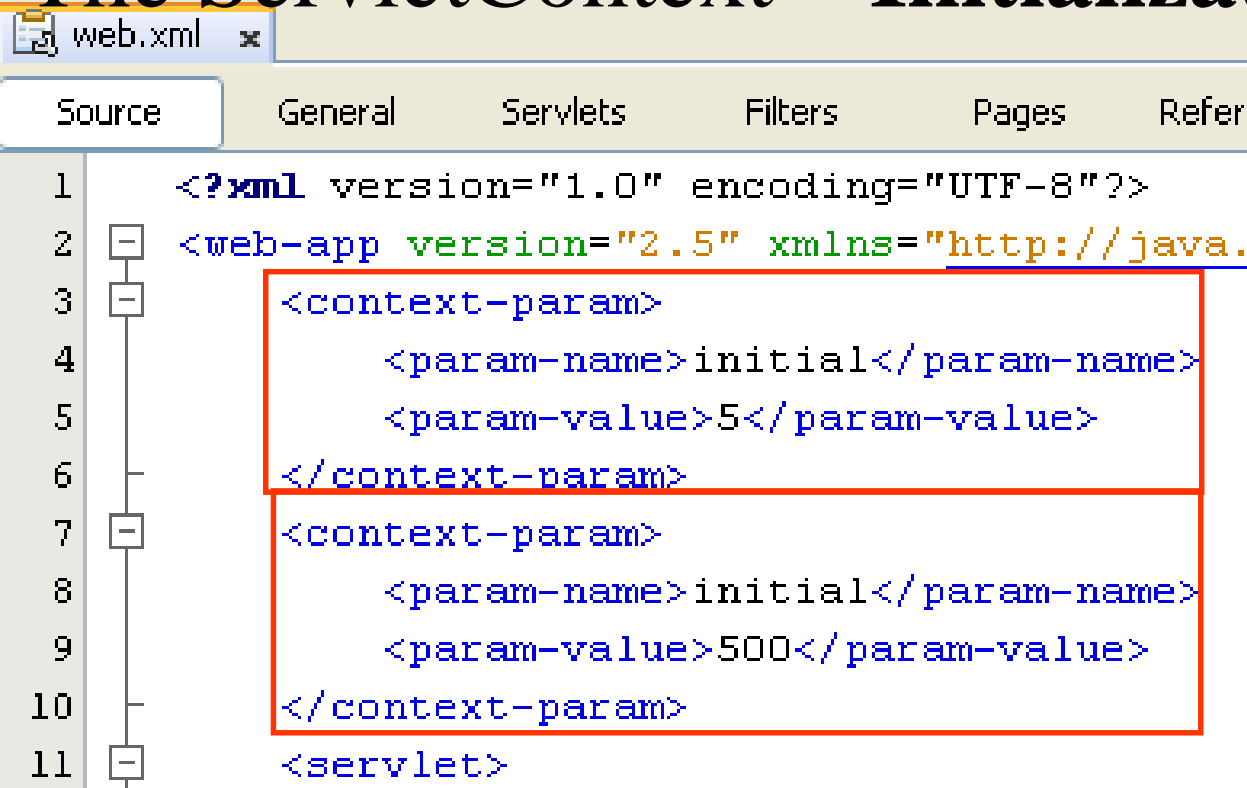
The `init` method in `InitCounter.java` is highlighted with a red box, showing how it retrieves the `initial` parameter from the `ServletContext` using `getInitParameter`.

The ServletContext – Initialization Parameters

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        ...
        out.println("<body>");
        out.println("<h1>The ServletContext-Init Demo</h1>");
        count++;
        out.println("The web is accessed in " + count + "times");
        ...
        out.println("</body>");
        out.println("</html>");
    } finally {
        out.close();
    }
}
```



The ServletContext – Initialization Parameters

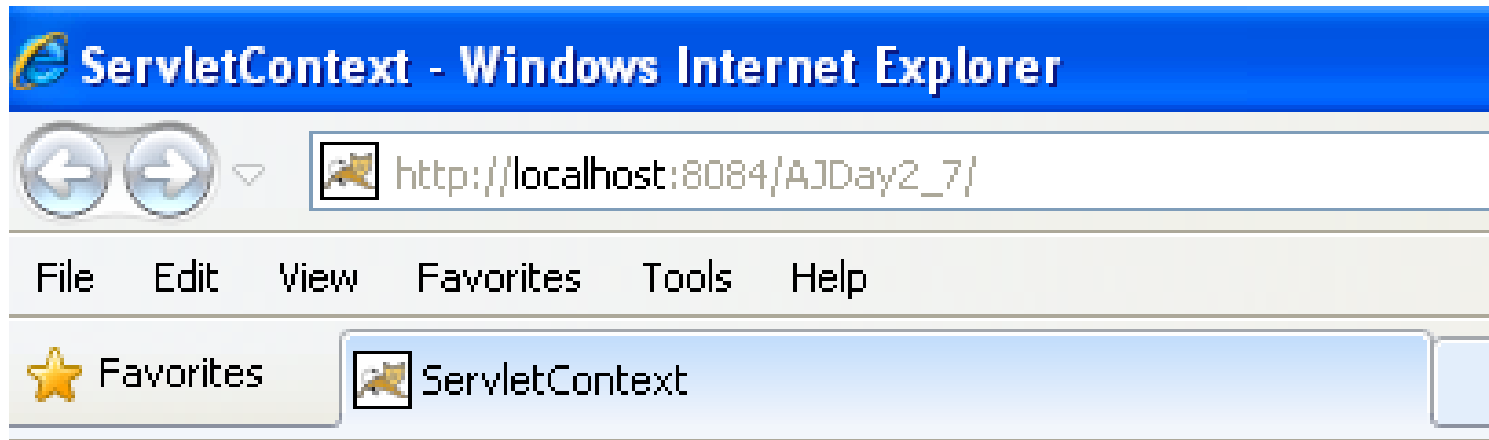


```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app version="2.5" xmlns="http://java.
3      <context-param>
4          <param-name>initial</param-name>
5          <param-value>5</param-value>
6      </context-param>
7      <context-param>
8          <param-name>initial</param-name>
9          <param-value>500</param-value>
10     </context-param>
11     <servlet>
  
```

The Web Container Model

The ServletContext – Initialization Parameters



Servlet Context - Init Demo

The web is accessed in 501 times

The Web Container Model

The ServletConfig interface

- To **pass as an argument** during initialization, the servlet container uses an object of ServletConfig interface
- **Configuring a servlet before processing** requested data
- Retrieve servlet initialization parameters

Methods	Descriptions
getServletName	<ul style="list-style-type: none"> - public String getServletName() - Searches the configuration information and retrieves name of the servlet instance - String servletName = getServletName();
getInitParameter	<ul style="list-style-type: none"> - public String getInitParameter (String name) - Retrieves the value of the initialisation parameter - Returns null if the specified parameter does not exist - String password = getInitParameter("password");
getServletContext	<ul style="list-style-type: none"> - public ServletContext getServletContext() - returns a ServletContext object used by the servlet to interact with its container. - ServletContext ctx = getServletContext();

The Web Container Model

The ServletConfig – Initialization Parameters

- Setting up the Deployment Descriptor

```
<servlet>
  <servlet-name>servletName</servlet-name>
  <servlet-class>servletClass</servlet-class>
  <init-param>
    <param-name>parName</param-name>
    <param-value>parValue</param-value>
  </init-param>
</servlet>
```

- Writing Code to Retrieve ServletConfig Initialization Parameters

```
ServletConfig sc = getServletConfig();
String name = sc.getInitParameter("parName");
```

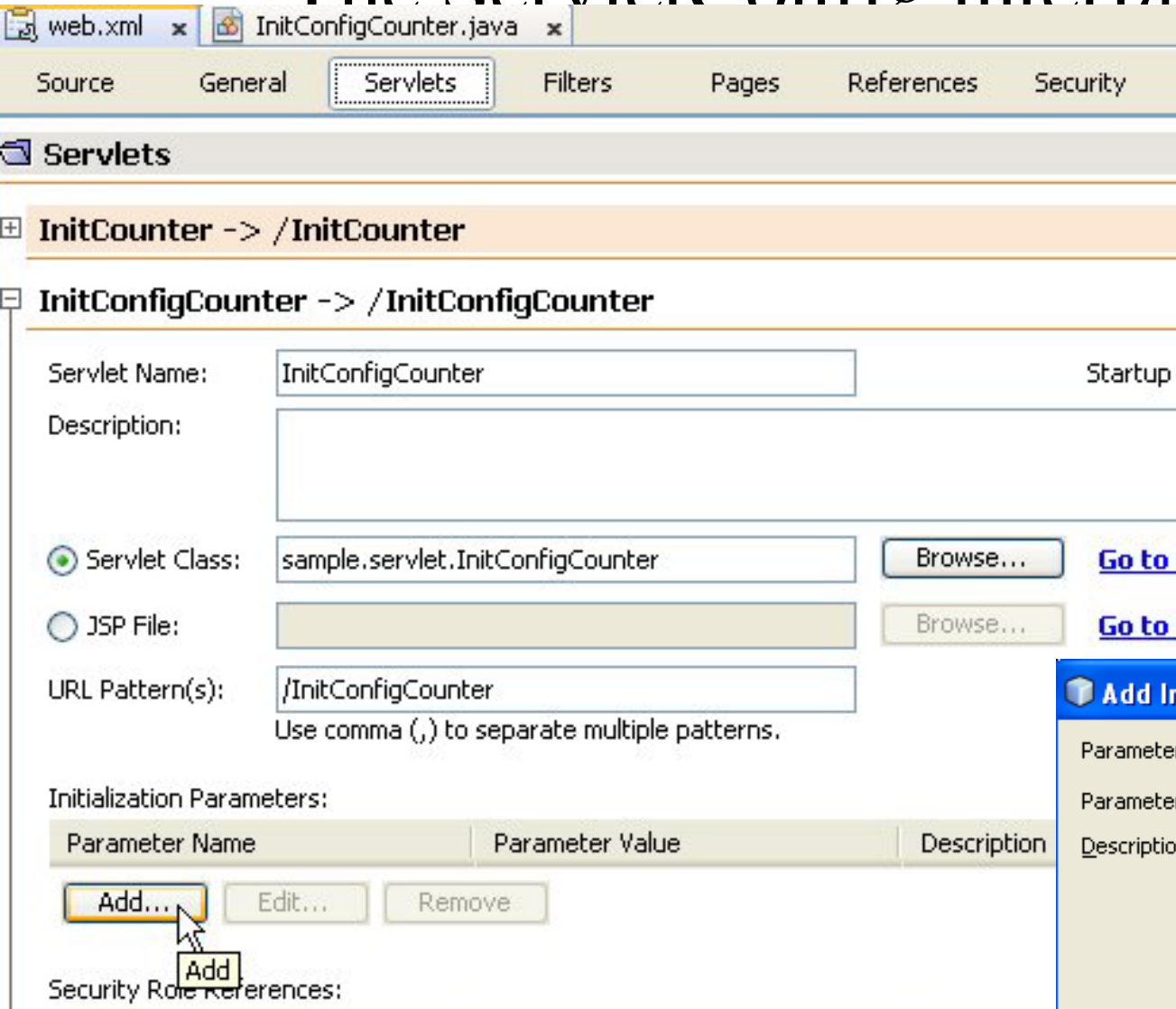
The Web Container Model

The ServletConfig interface – Example



The Web Container Model

The ServletConfig interface – Example



The screenshot shows the Eclipse IDE with the 'Servlets' tab selected. The configuration is for the 'InitConfigCounter' servlet, which is mapped to the URL pattern '/InitConfigCounter'. The servlet class is set to 'sample.servlet.InitConfigCounter'. The 'Add...' button in the 'Initialization Parameters' section is highlighted with a mouse cursor, and a tooltip labeled 'Add' is visible.

web.xml x InitConfigCounter.java x

Source General **Servlets** Filters Pages References Security

Servlets

InitCounter -> /InitCounter

InitConfigCounter -> /InitConfigCounter

Servlet Name: InitConfigCounter Startup

Description:

☒ Servlet Class: sample.servlet.InitConfigCounter Browse... Go to...

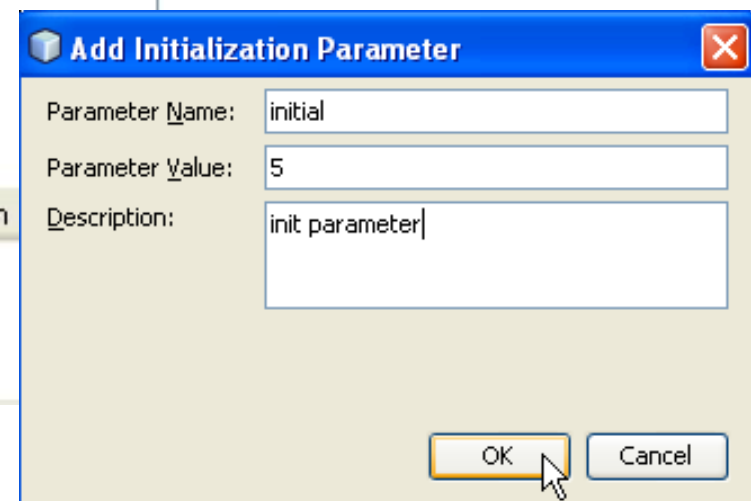
☐ JSP File: Browse... Go to...

URL Pattern(s): /InitConfigCounter
Use comma (,) to separate multiple patterns.

Initialization Parameters:

Parameter Name	Parameter Value	Description
<input type="button" value="Add..."/> <input type="button" value="Edit..."/> <input type="button" value="Remove"/>		

Security Role References:



The dialog box is titled 'Add Initialization Parameter'. It contains three input fields: 'Parameter Name' with the value 'initial', 'Parameter Value' with the value '5', and 'Description' with the value 'init parameter'. The 'OK' button is highlighted with a mouse cursor.

Add Initialization Parameter

Parameter Name: initial

Parameter Value: 5

Description: init parameter

The Web Container Model

The ServletConfig interface – Example

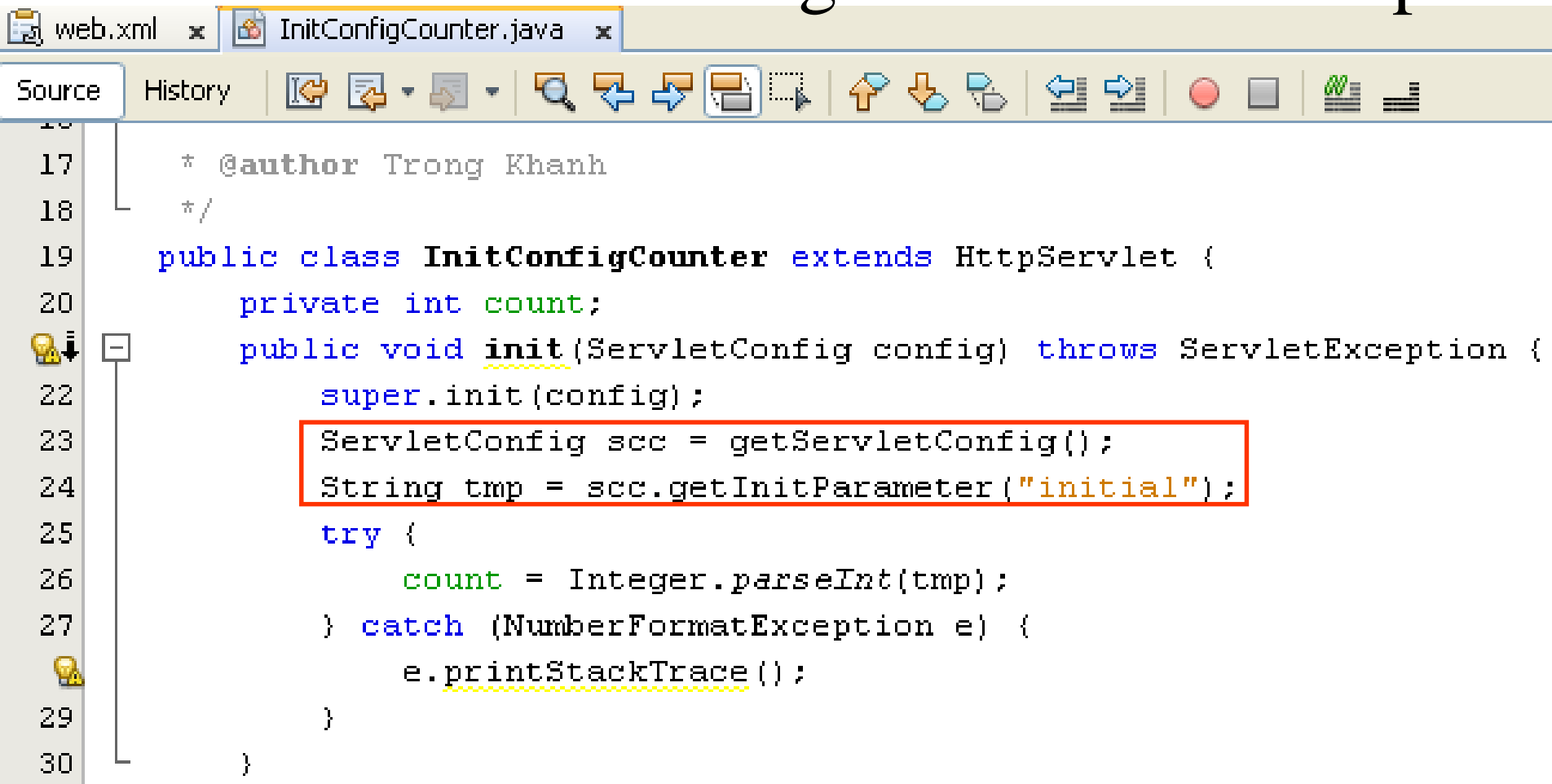


The screenshot shows an IDE window with two tabs: 'web.xml' and 'InitConfigCounter.java'. The 'web.xml' tab is active, displaying XML code. The code defines a web application with version '2.5' and a namespace. It includes two context parameters, two servlets, and a servlet mapping. The second servlet, 'InitConfigCounter', is configured with a class 'sample.servlet.InitConfigCounter' and an initialization parameter 'initial' with a value of '5'. This configuration block is highlighted with a red rectangle.

```

2  <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:
3      <context-param>
7      <context-param>
11     <servlet>
15     <servlet>
16         <servlet-name>InitConfigCounter</servlet-name>
17         <servlet-class>sample.servlet.InitConfigCounter</servlet-class>
18         <init-param>
19             <description>init parameter</description>
20             <param-name>initial</param-name>
21             <param-value>5</param-value>
22         </init-param>
23     </servlet>
24     <servlet-mapping>
  
```

The ServletConfig interface – Example



```

17  * @author Trong Khanh
18  */
19  public class InitConfigCounter extends HttpServlet {
20      private int count;
21      public void init(ServletConfig config) throws ServletException {
22          super.init(config);
23          ServletConfig scc = getServletConfig();
24          String tmp = scc.getInitParameter("initial");
25          try {
26              count = Integer.parseInt(tmp);
27          } catch (NumberFormatException e) {
28              e.printStackTrace();
29          }
30      }
  
```

The ServletConfig interface – Example

```

42  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
43      throws ServletException, IOException {
44      response.setContentType("text/html;charset=UTF-8");
45      PrintWriter out = response.getWriter();
46      try {
47          /* TODO output your page here. You may use following sample */
48          out.println("<!DOCTYPE html>");
49          out.println("<html>");
50          out.println("<head>");
51          out.println("<title>ServletConfig</title>");
52          out.println("</head>");
53          out.println("<body>");
54          out.println("<h1>Servlet Config - Init Counter Demo</h1>");
55
56          count++;
57          out.println("The web is accessed in " + count + "times");
58          out.println("</body>");
59          out.println("</html>");
60      } finally {
61          out.close();
62      }
63  }

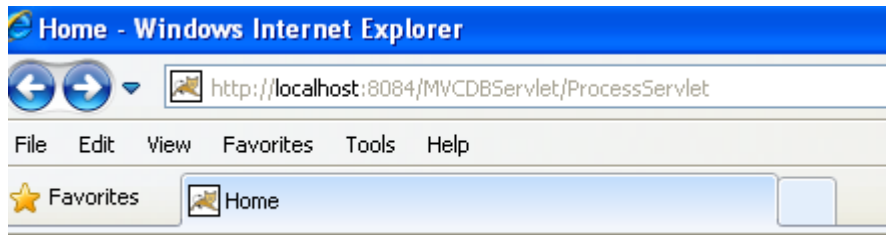
```



How To Transfer Requirements

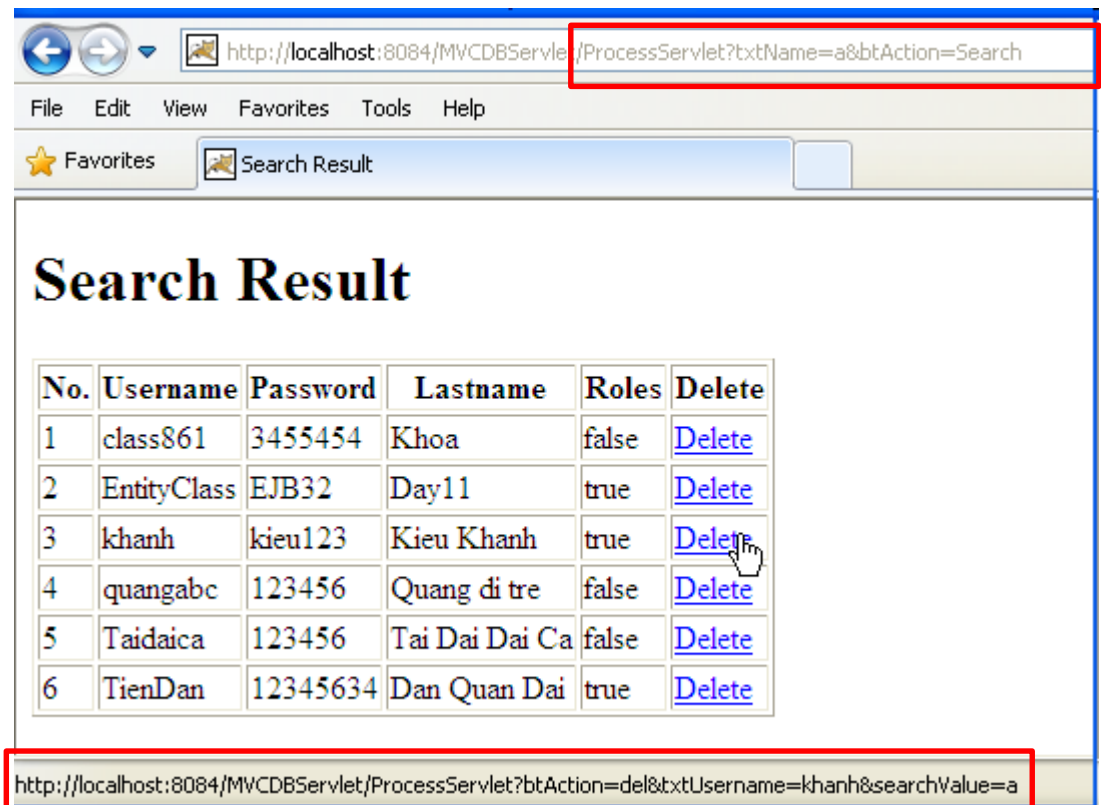
- After built the web application in the first topic
 - **The search page** allows user **search appropriate the last name of users**
 - **The result** of searching is **shown in the data grid**. In each row, the **information about ordinary number, username, password, last name and roles** is shown
- The GUI of web application is present as following

How To Transfer Expectation

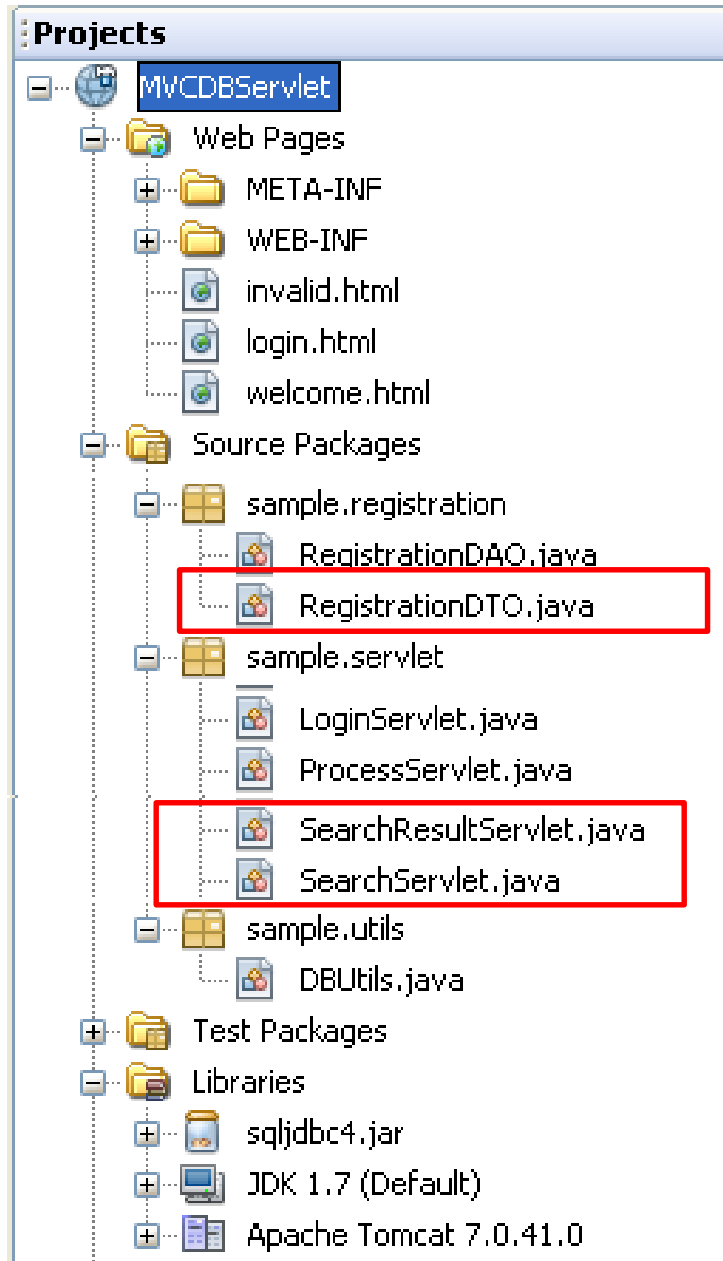


Welcome to DB Servlet

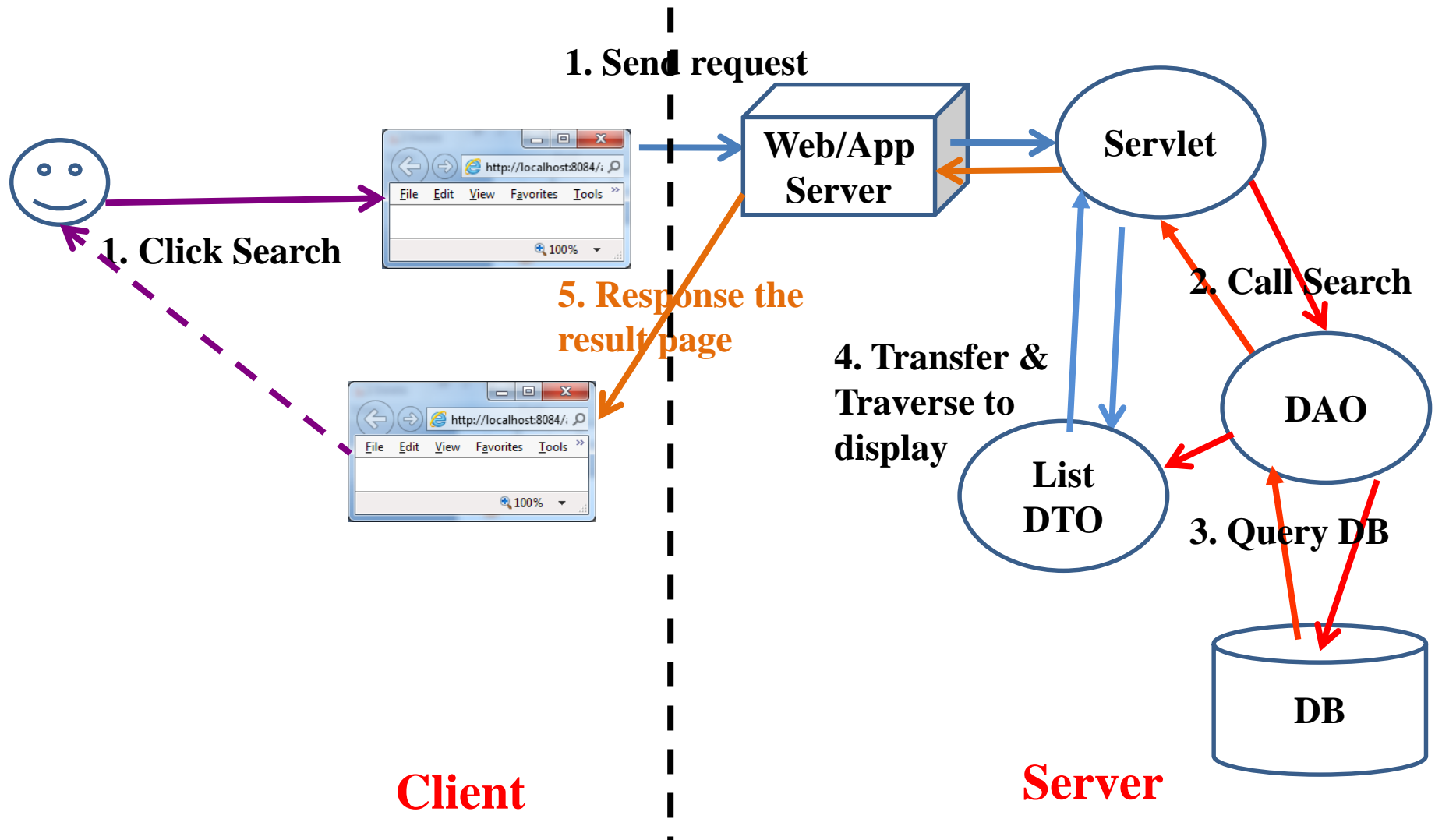
Name



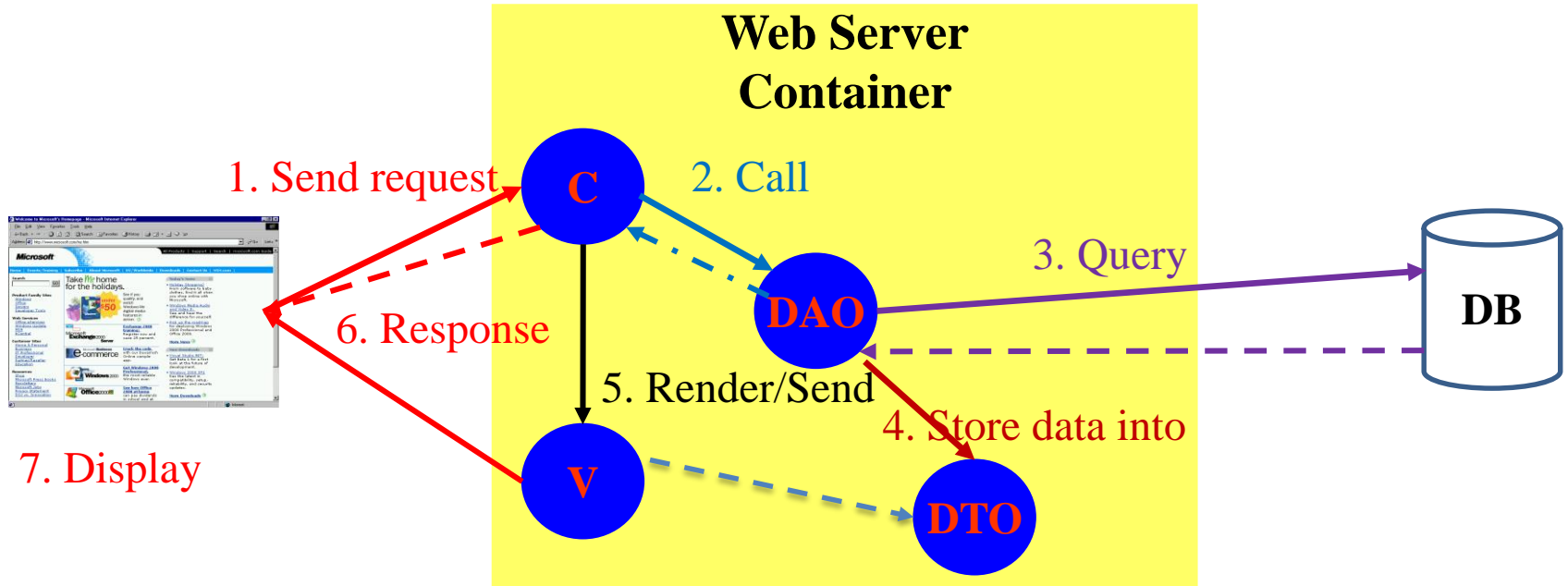
How To Transfer Expectation



How To Transfer Interactive Server Model



How To Transfer Abstraction



The Web Container Model

Need for using attributes

- **Problems:**

- **How to remember an user that has already logged into the particular website?**
- **How to store a collection of selected products online** when the user has **already chosen** while the HTTP is a stateless protocol?
Besides, they can search and choose other products

- **Solutions:**

- **Store data or object as long as user still browses the web site**
- **Attributes is a qualified candidate: Attributes are a collection of <attribute-name, value> pairs that is stored in a scope (segment) in server**
- **Life cycle of them is long as its defined scope.**

The Web Container Model

Attributes, **Scope**, and Multithreading

- Defines **how long** a reserved **memory segment** is **available in the context on the server**.
- There are **3 scopes**
 - **Request Scope**
 - **Lasts** from HTTP request hits a **web container** to the servlet **delivers** the HTTP response.
 - `javax.servlet.HttpServletRequest`
 - **Session Scope**
 - A **browser window** **establishes** up to the point where that **browser window** is **closed**
 - **Open session** up to the point where that **session** is **closed**, **session** is **time out**, **server** is **crashed**.
 - `javax.servlet.http.HttpSession`
 - **Context (Application) Scope**
 - Is the **longest-lived** of the three scopes available to you.
 - Exists **until** the **web container** is **stopped**.
 - `javax.servlet.ServletContext`

The Web Container Model

Attributes, **Scope**, and Multithreading

- **Choosing Scopes**

- **Request Scope:** attributes are required for a one-off web page and aren't part of a longer transaction
- **Session Scope:** attributes are part of a longer transaction, or are spanned **several request** but they are information **unique to particular client**
 - **Ex:** username or account
- **Context Scope:** attributes can allow **any web resource to access** (e.g. public variables in application)

The Web Container Model

Attributes, Scope, and Multithreading

- **Parameters vs. Attributes**
 - **Parameters** allow information to flow into a web application (**passed** to web application **via form or query string**). They **exist** in **request scope**
 - **Attributes** are more of a means of handling information *within* the web application. They can be **shared or accessed within** their **defined scope**
 - **Data types of Parameter is String but the Attribute is Object**
- The web container uses attributes as a place to
 - **Provide information to interested code:** the way supplement the standard APIs that yield information about the web container
 - **Hang on to information that your application, session, or even request requires later.**
- The developer can access the attribute value with **attribute's name**

The Web Container Model

Attributes, Scope, and Multithreading

Methods	Descriptions
getAttribute	<ul style="list-style-type: none">- public Object getAttribute(String name)- returns the value of the name attribute as Object- Ex: String user = (String)servletContext.getAttribute("USER");
setAttribute	<ul style="list-style-type: none">- public void setAttribute(String name, Object obj)- Binds an object to a given attribute name in the scope- Replace the attribute with new attribute, if the name specified is already used- servletContext.setAttribute("USER", "Aptech");
removeAttribute	<ul style="list-style-type: none">- public void removeAttribute(String name)- Removes the name attributes- Ex: servletContext.removeAttribute("USER");
getAttributeNames	<ul style="list-style-type: none">- public Enumeration getAttributeNames()- Returns an Enumeration containing the name of available attributes. Returns an empty if no attributes exist.

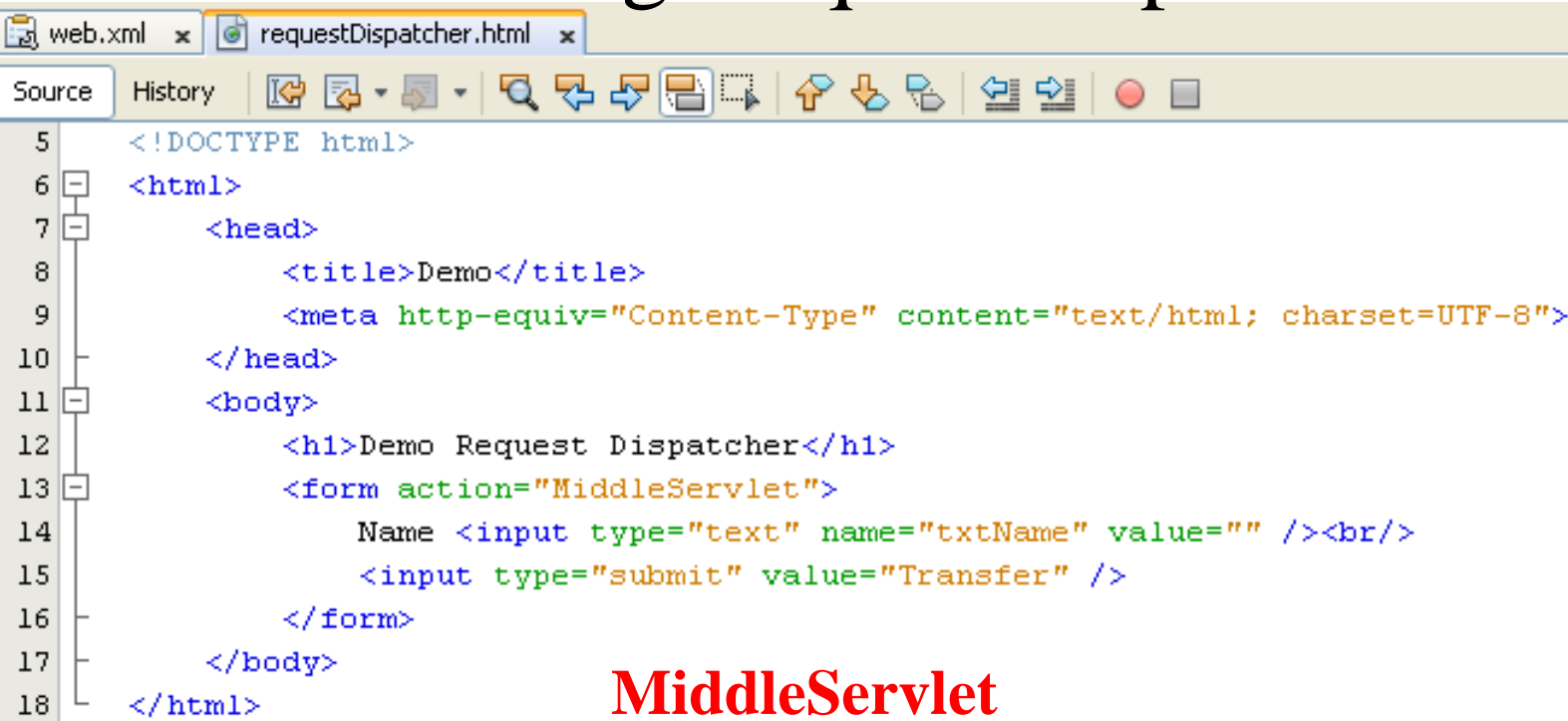
The Web Container Model

Attributes, Scope, and Multithreading

- **Multithreading and Request Attributes**
 - request attributes are thread safe (*because everything will only ever be accessed by one thread and one thread alone*)
- **Multithreading and Session Attributes**
 - session attributes are *officially* not thread safe.
- **Multithreading and Context Attributes**
 - context attributes are not thread safe
 - You have **two approaches** to solve the multithreading dilemma:
 - **Set up servlet context attributes** in the **init() method** of a servlet that loads on the startup of the server, and at no other time. Thereafter, treat these **attributes** as “read only”.
 - If there are **context attributes** where you have no option but to update them later, surround the updates with synchronization blocks.

The Web Container Model

Need for using RequestDispatcher – Redirect



```

5 <!DOCTYPE html>
6 <html>
7   <head>
8     <title>Demo</title>
9     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
10  </head>
11  <body>
12    <h1>Demo Request Dispatcher</h1>
13    <form action="MiddleServlet">
14      Name <input type="text" name="txtName" value="" /><br/>
15      <input type="submit" value="Transfer" />
16    </form>
17  </body>
18 </html>
  
```

MiddleServlet

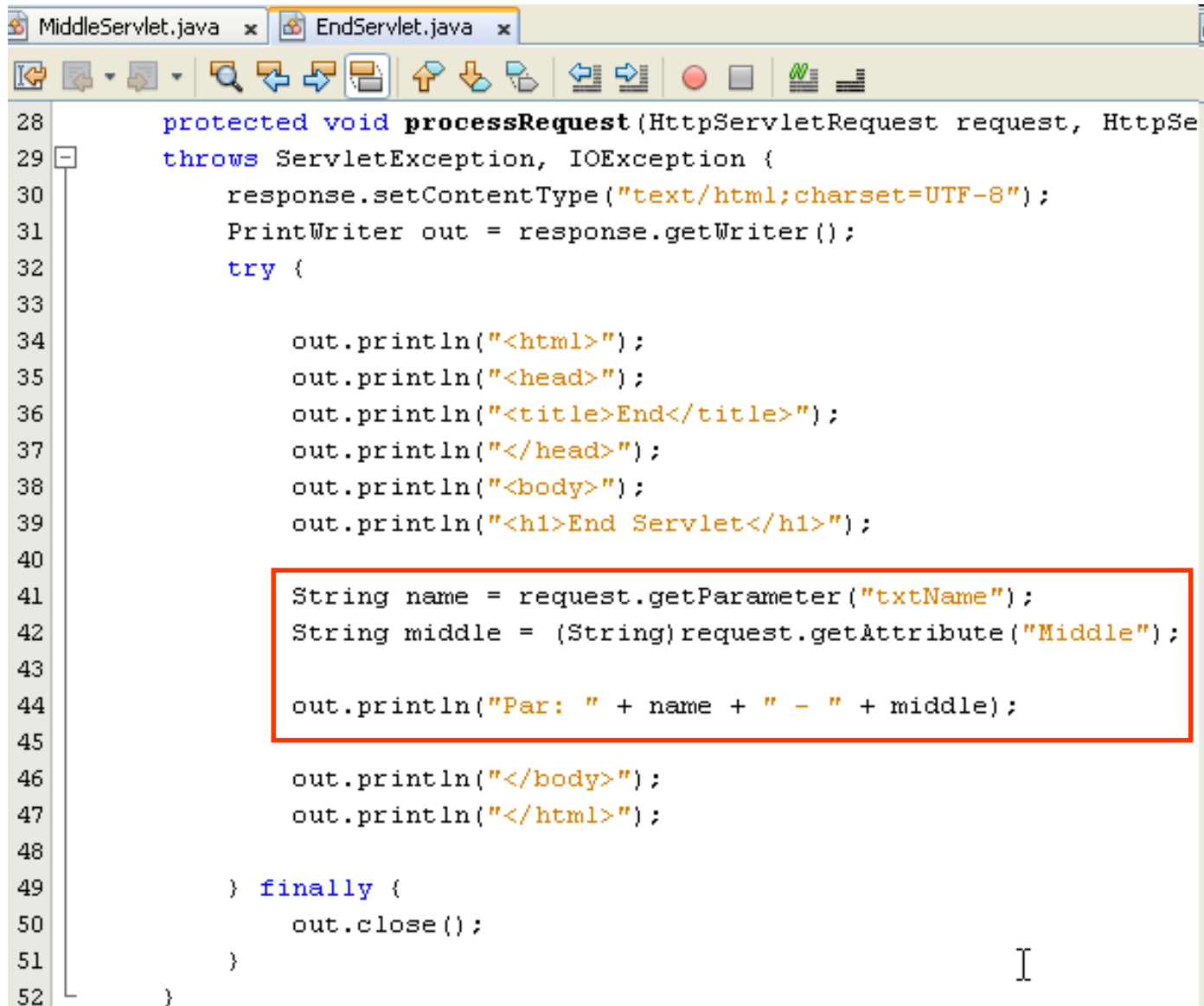
```
out.println("<h1>Middle Servlet</h1>");
```

```
request.setAttribute("Middle", "Middle Information");
response.sendRedirect("EndServlet");
```

```
out.println("</body>");
```

```
out.println("</html>");
```

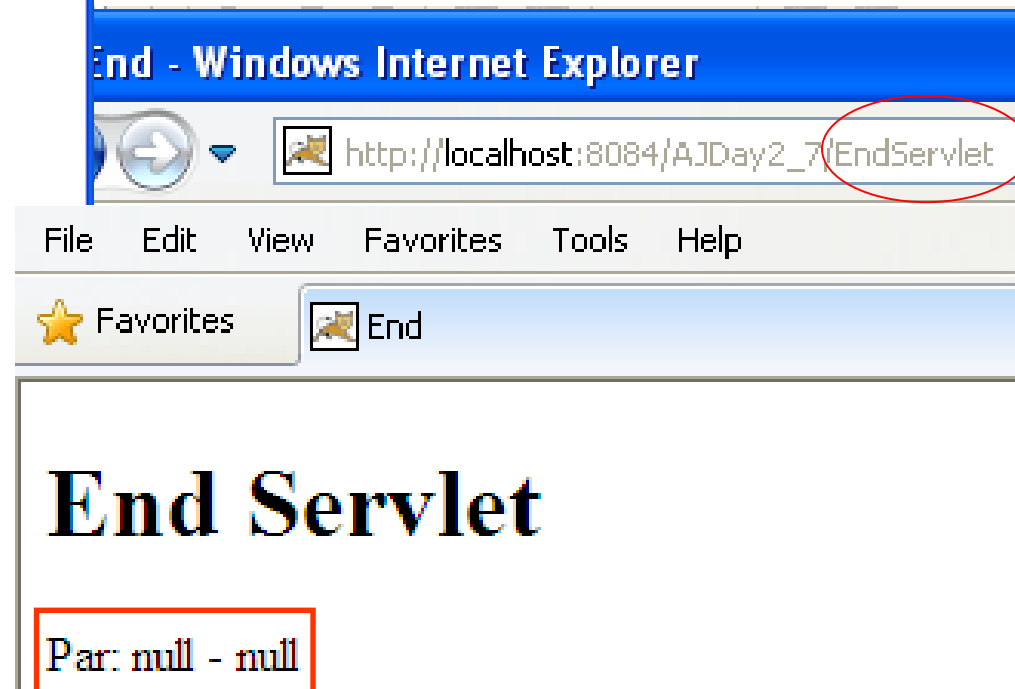
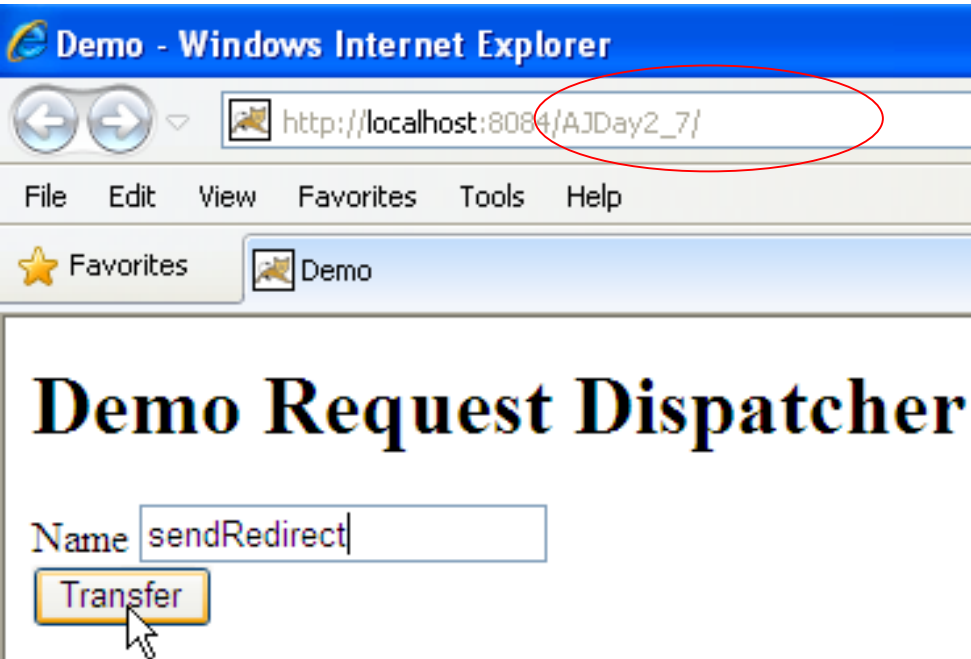
Need for using RequestDispatcher – Redirect



```
28     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
29     throws ServletException, IOException {
30         response.setContentType("text/html;charset=UTF-8");
31         PrintWriter out = response.getWriter();
32         try {
33
34             out.println("<html>");
35             out.println("<head>");
36             out.println("<title>End</title>");
37             out.println("</head>");
38             out.println("<body>");
39             out.println("<h1>End Servlet</h1>");
40
41             String name = request.getParameter("txtName");
42             String middle = (String)request.getAttribute("Middle");
43
44             out.println("Par: " + name + " - " + middle);
45
46             out.println("</body>");
47             out.println("</html>");
48
49         } finally {
50             out.close();
51         }
52     }
```

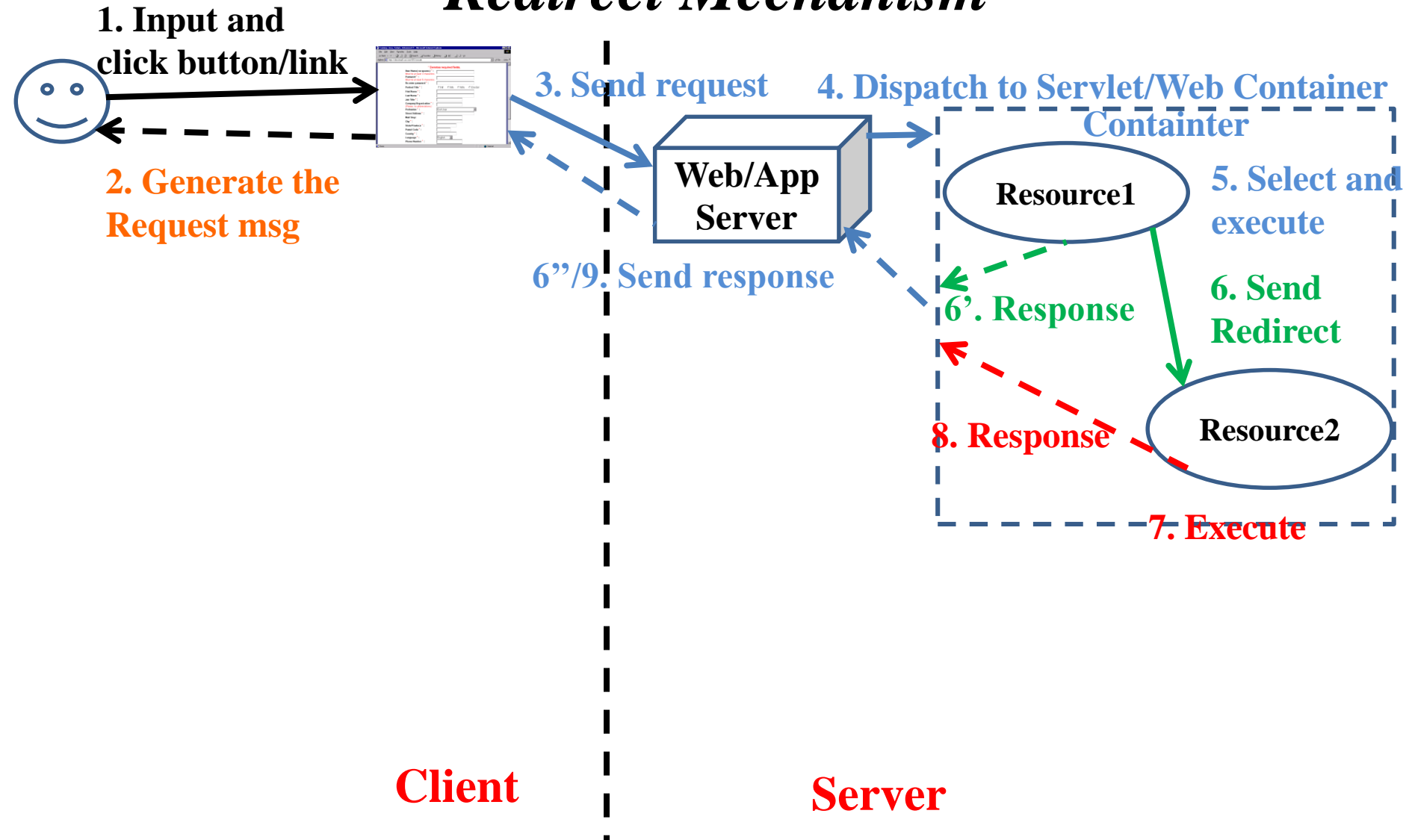
The Web Container Model

Need for using RequestDispatcher – Redirect



The Web Container Model

Need for using RequestDispatcher *Redirect Mechanism*



The Web Container Model

Request Dispatching

- Is a mechanism for controlling the flow of control within the web resources in the web application
- The `ServletRequest` and `ServletContext` support the **`getRequestDispatcher(String path)` method**
 - Returns `RequestDispatcher` instance
 - The path parameter can be a full path beginning at the context root (“/”) – **requirement with `ServletContext`**
 - The `ServletContext` offers the **`getNameDispatcher(String name)`** method that requires providing the resource’s name to want to execute (e.g. the name must match one of the `<servlet-name>`)
- A **`RequestDispatcher` object**
 - Is created by the servlet container
 - Redirect the client request to a particular Web page

The Web Container Model

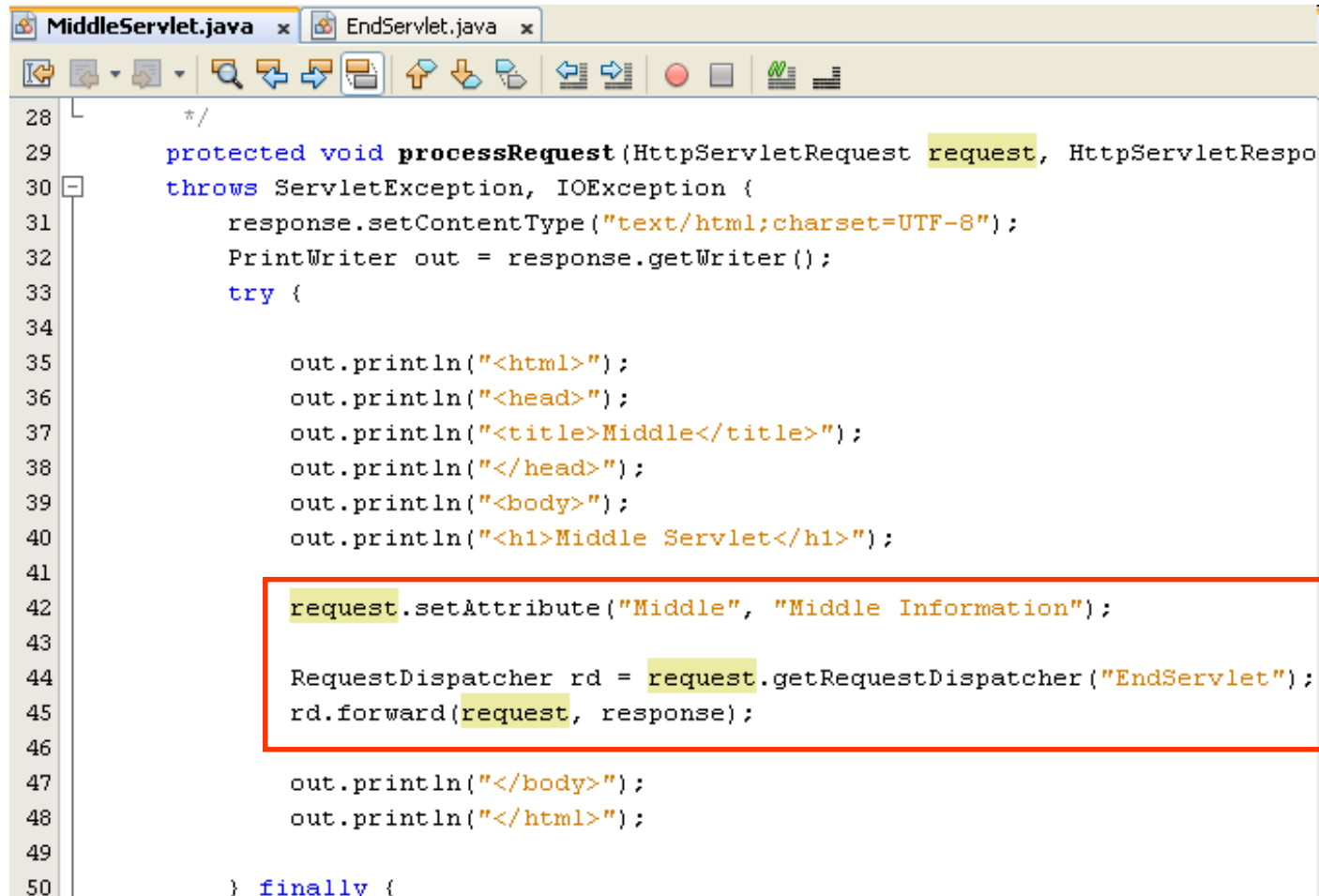
Using RequestDispatcher

Methods	Descriptions
forward	<ul style="list-style-type: none">- Redirect the output to another servlet- Forward the request to another Servlet to process the client request.- Ex: <code>RequestDispatcher rd = request.getRequestDispatcher("home.jsp");</code> <code>rd.forward(request, response);</code>
include	<ul style="list-style-type: none">- Include the content of another servlet into the current output stream- Include the output of another Servlet to process the client request- Ex <code>RequestDispatcher rd = request.getRequestDispatcher("home.jsp");</code> <code>rd.include (request, response);</code>

The Web Container Model

Using RequestDispatcher – Example

```
<body>
  <h1>Demo Request Dispatcher</h1>
  <form action="MiddleServlet">
    Name <input type="text" name="txtName" value="" /><br/>
    <input type="submit" value="Transfer" />
  </form>
</body>
```



```
MiddleServlet.java x EndServlet.java x
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Middle</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Middle Servlet</h1>");

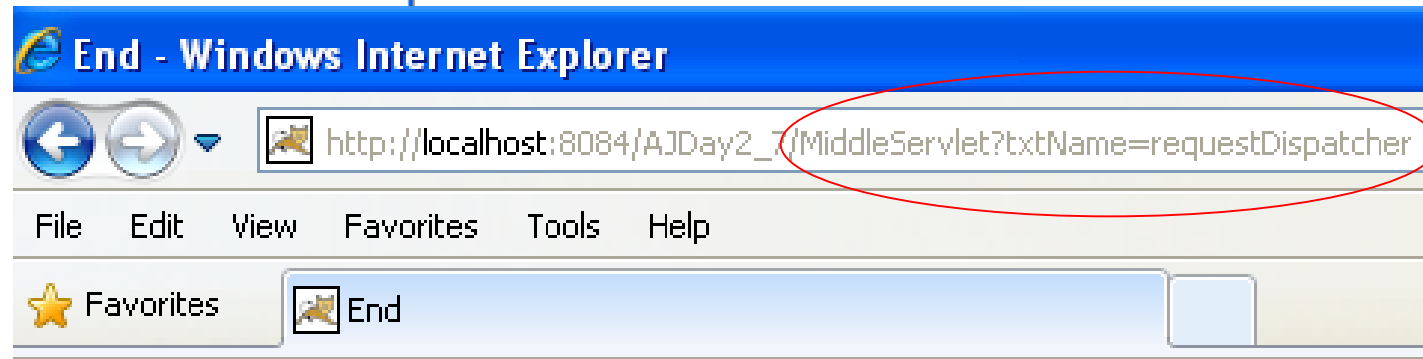
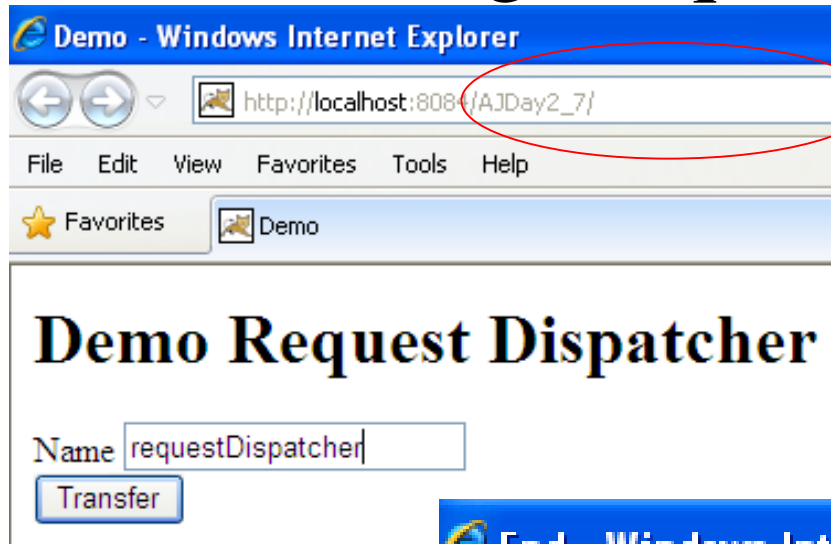
        request.setAttribute("Middle", "Middle Information");

        RequestDispatcher rd = request.getRequestDispatcher("EndServlet");
        rd.forward(request, response);

        out.println("</body>");
        out.println("</html>");
    } finally {
```

The Web Container Model

Using RequestDispatcher – Example

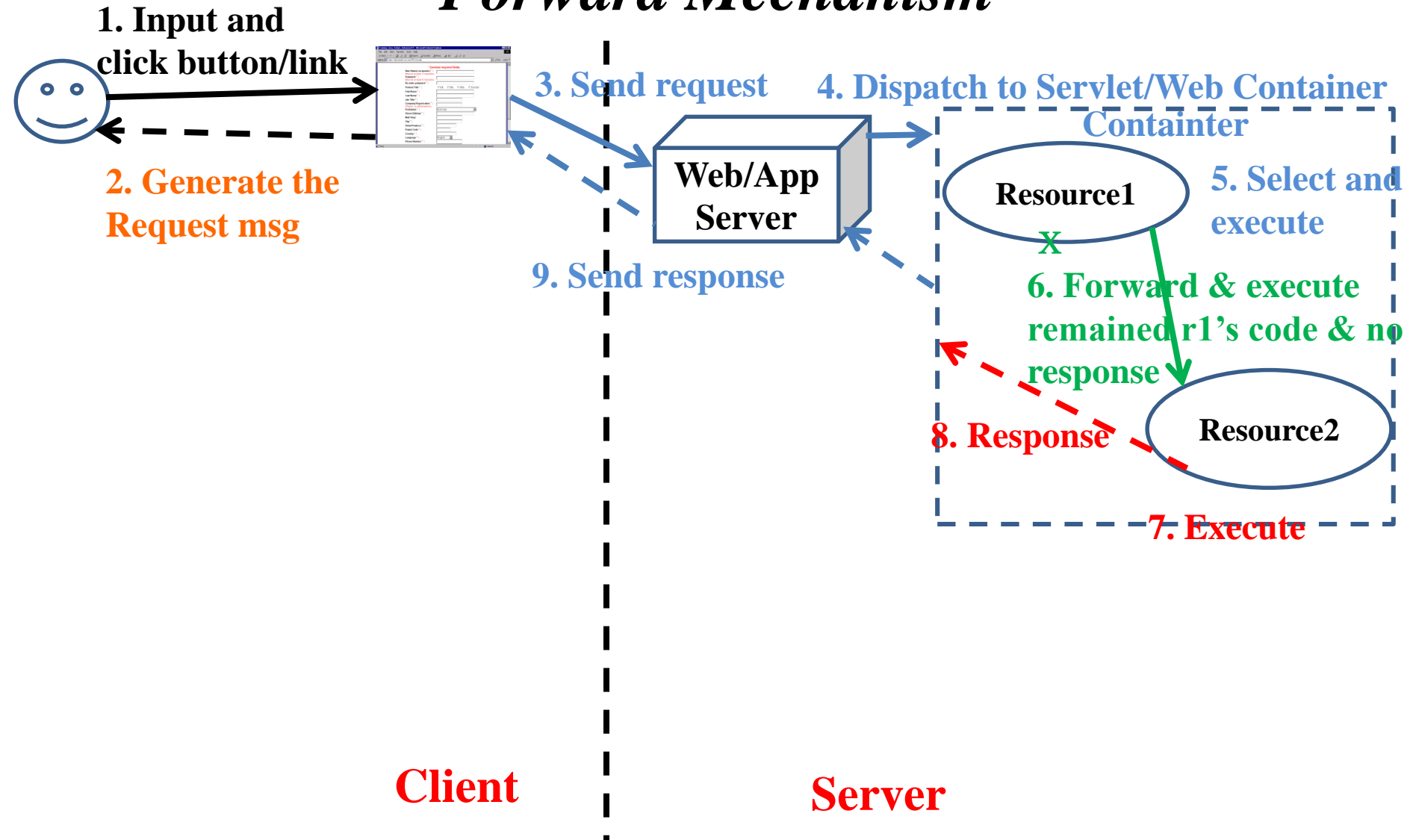


End Servlet

Par: requestDispatcher - Middle Information

The Web Container Model

Need for using RequestDispatcher *Forward Mechanism*



The Web Container Model

Using RequestDispatcher – Example

Change the RequestDispatch – forward method to include method

Demo - Windows Internet Explorer

http://localhost:8084/AJDay2_7/

File Edit View Favorites Tools Help

★ Favorites Demo

Demo Request Dispatcher

Name

Middle - Windows Internet Explorer

http://localhost:8084/AJDay2_7/MiddleServlet?txtName=includeDispatcher

File Edit View Favorites Tools Help

★ Favorites Middle

Middle Servlet

End Servlet

Par: includeDispatcher - Middle Information

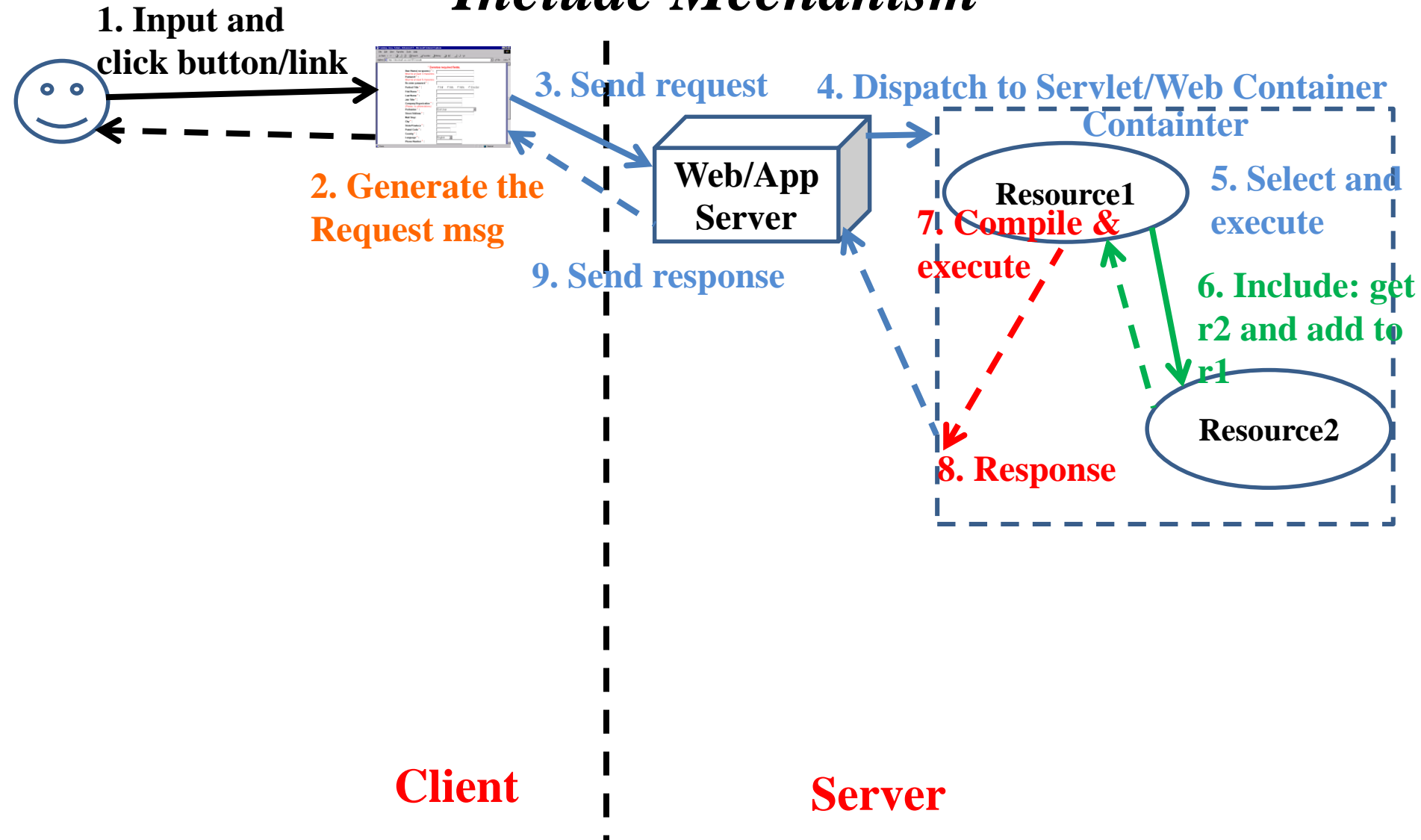
```
request.setAttribute("Middle", "Middle Information");

RequestDispatcher rd = request.getRequestDispatcher("EndServlet");

rd.include(request, response);
out.println("</body>");
out.println("</html>");
```

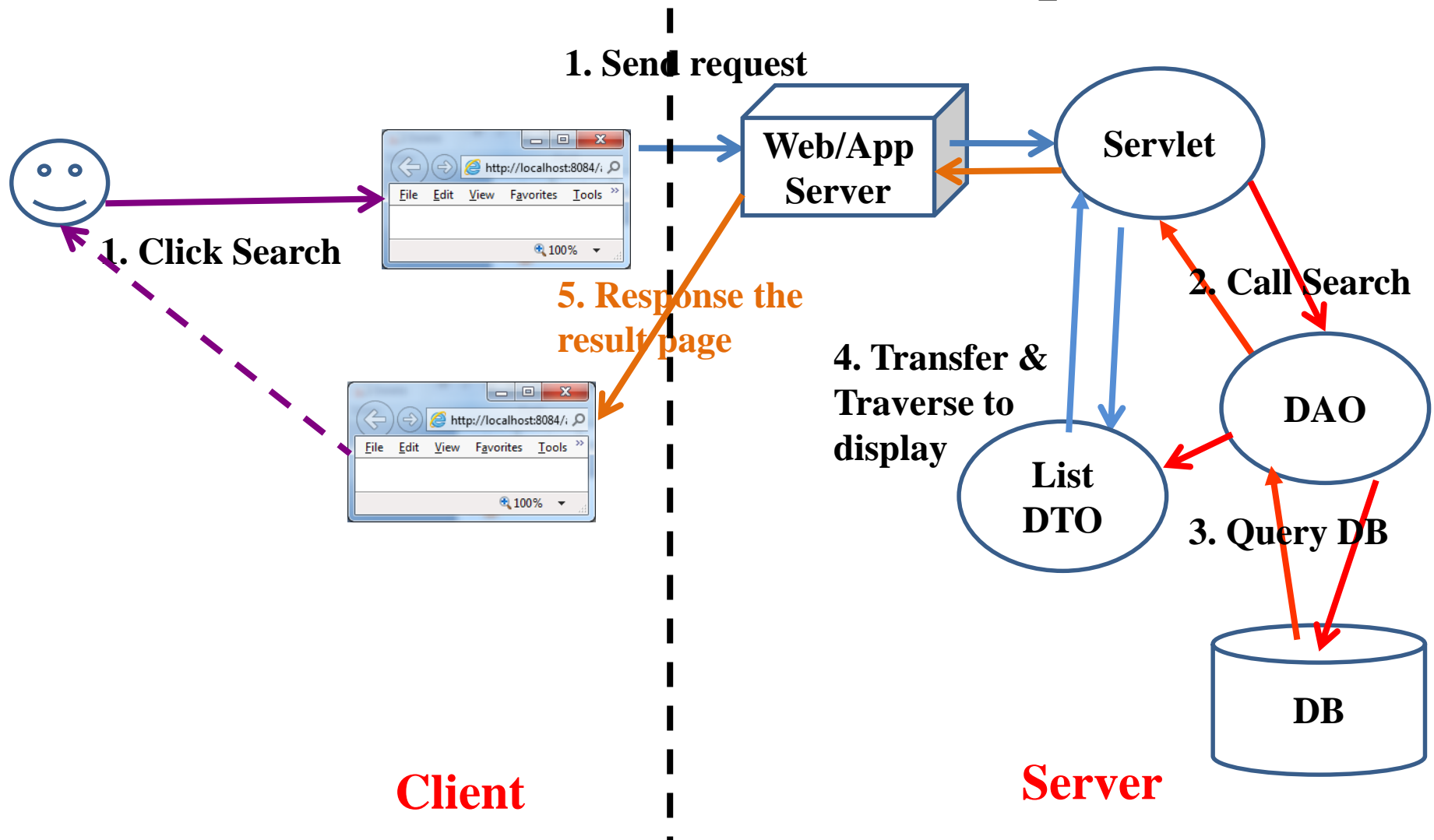
The Web Container Model

Need for using RequestDispatcher *Include Mechanism*



How To Transfer

Interactive Server Model – Implementation

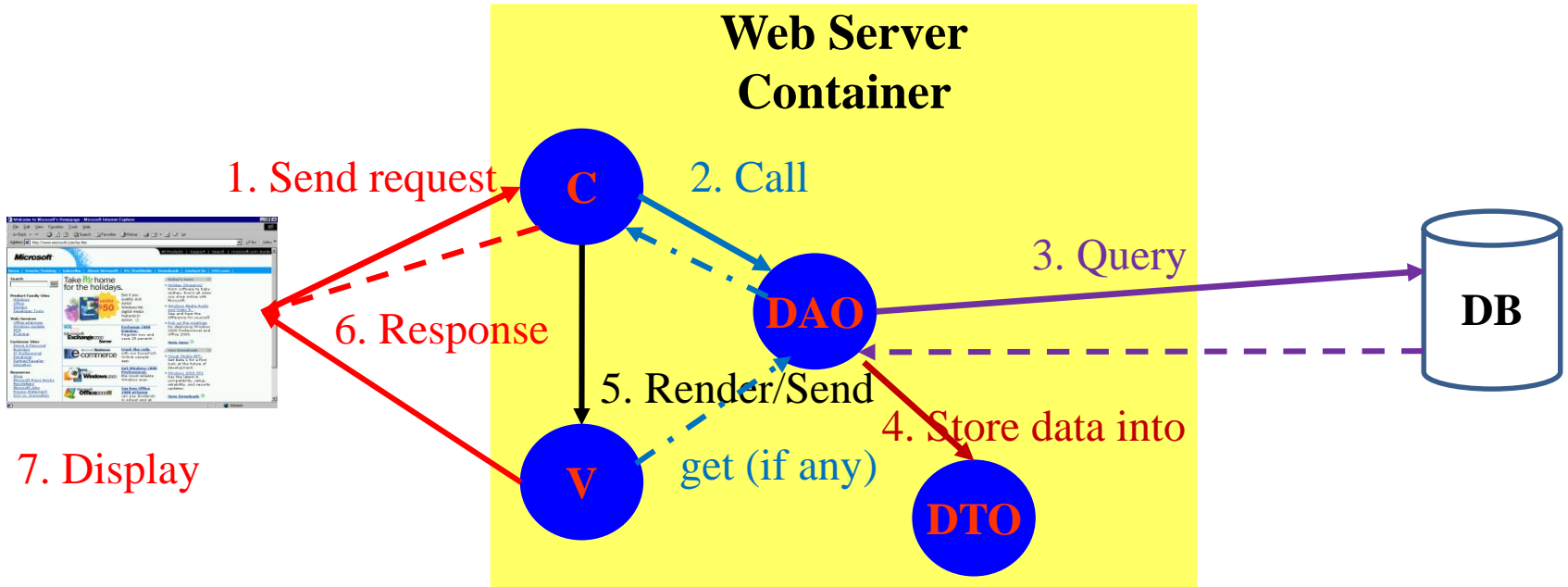


Summary

- **How to deploy the Web Application to Web Server?**
 - Web applications Structure
 - Request Parameters vs. Context Parameters vs. Config/Servlet Parameters
 - Application Segments vs. Scope
- **How to transfer from resources to others with/without data/objects?**
 - Attributes vs. Parameters vs. Variables
 - Redirect vs. RequestDispatcher

Q&A

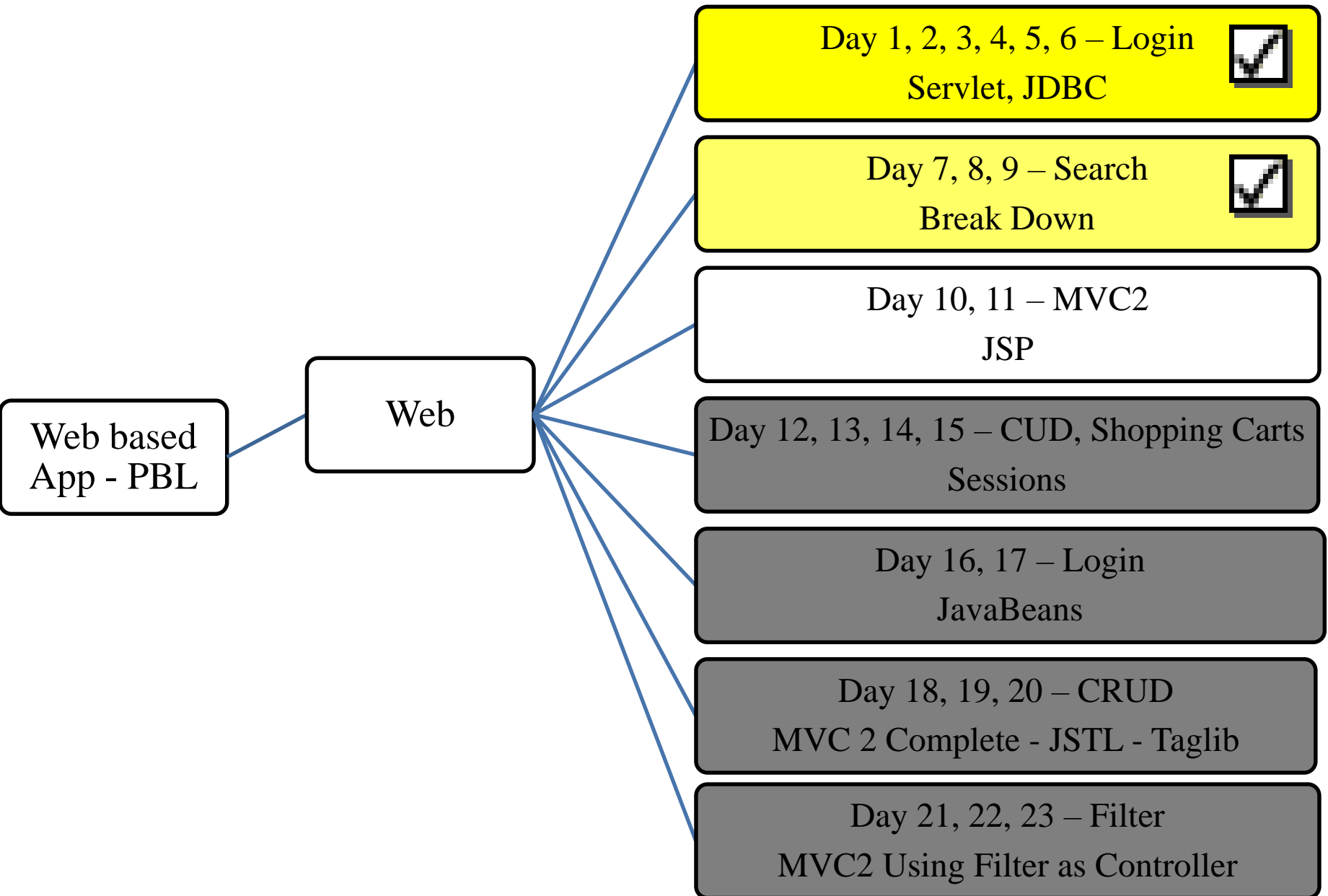
Summary



Next Lecture

- **How to upgrade Application in previous topics approach MVC Model**
 - **Using JSP to View**
 - **MVC Pattern Design**

Next Lecture



Appendix – How to Transfer DTO

RegistrationDTO.java

```

6 package sample.registration;
7 import ...
8 /**
9  *
10  * @author kieukhanh
11  */
12 public class RegistrationDTO implements Serializable {
13     private String username;
14     private String password;
15     private String lastname;
16     private boolean role;
17     public RegistrationDTO() {...2 lines }
18     public RegistrationDTO(String username, String password,
19         String lastname, boolean role) {...6 lines }
20     /**...3 lines */
21     public String getUsername() {...3 lines }
22     /**...3 lines */
23     public void setUsername(String username) {...3 lines }
24     /**...3 lines */
25     public String getPassword() {...3 lines }
26     /**...3 lines */
27     public void setPassword(String password) {...3 lines }
28     /**...3 lines */
29     public String getLastName() {...3 lines }
30     /**...3 lines */
31     public void setLastName(String lastname) {...3 lines }
32     /**...3 lines */
33     public boolean isRole() {...3 lines }
34     /**...3 lines */
35     public void setRole(boolean role) {...3 lines }
36 }
    
```

How to Transfer DAO

RegistrationDAO.java

Source History

```

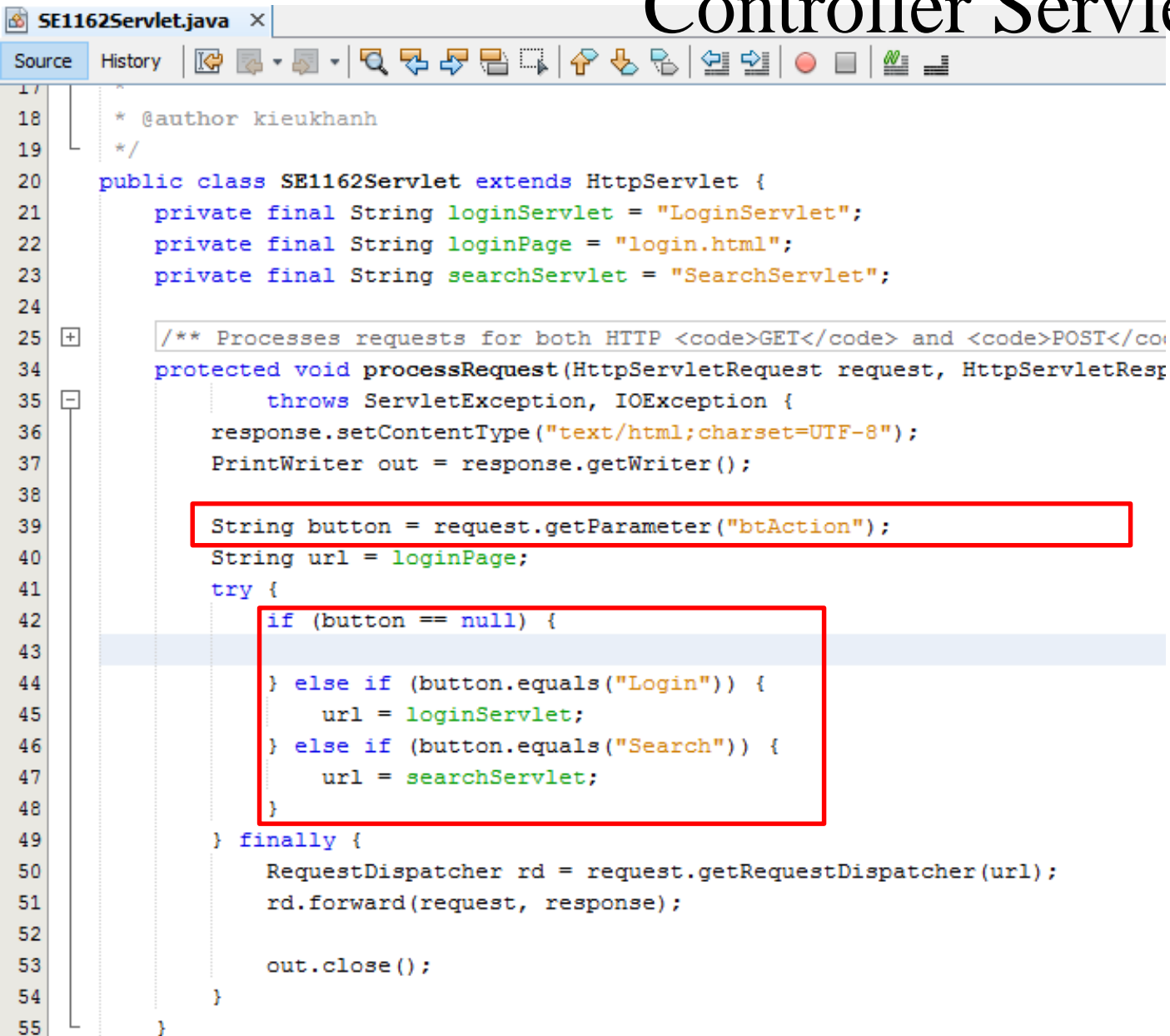
20  * @author kieukhanh
21  */
22  public class RegistrationDAO implements Serializable {
23      public boolean checkLogin(String username, String password)
24          throws SQLException, NamingException { ...35 lines ... }
59
60      private List<RegistrationDTO> listAccounts;
61
62      public List<RegistrationDTO> getListAccounts() {
63          return listAccounts;
64      }
65
66      public void searchLastname(String searchValue)
67          throws SQLException, NamingException {
68
69          Connection con = null;
70          PreparedStatement stm = null;
71          ResultSet rs = null;
72
73          try {
74              con = DBUtils.makeConnection();
75
76              if (con != null) {
77                  String sql = "Select * From Registration Where lastname Like ?";
  
```

How to Transfer DAO

```

79      stm = con.prepareStatement(sql);
80      stm.setString(1, "%" + searchValue + "%");
81
82      rs = stm.executeQuery();
83      while (rs.next()) {
84          String username = rs.getString("username");
85          String password = rs.getString("password");
86          String lastname = rs.getString("lastname");
87          boolean role = rs.getBoolean("isAdmin");
88
89          RegistrationDTO dto =
90              new RegistrationDTO(username, password, lastname, role);
91
92          if (this.listAccounts == null) {
93              this.listAccounts = new ArrayList<RegistrationDTO>();
94          }
95
96          this.listAccounts.add(dto);
97      }
98  }
99  } finally {
100      if (rs != null) {
101          rs.close();
102      }
103      if (stm != null) {
104          stm.close();
105      }
106      if (con != null) {
107          con.close();
108      }
  
```

How to Transfer Controller Servlet



```

17  *
18  * @author kieukhanh
19  */
20  public class SE1162Servlet extends HttpServlet {
21      private final String loginServlet = "LoginServlet";
22      private final String loginPage = "login.html";
23      private final String searchServlet = "SearchServlet";
24
25      /** Processes requests for both HTTP <code>GET</code> and <code>POST</code>
34  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
35      throws ServletException, IOException {
36      response.setContentType("text/html;charset=UTF-8");
37      PrintWriter out = response.getWriter();
38
39      String button = request.getParameter("btAction");
40      String url = loginPage;
41      try {
42          if (button == null) {
43
44          } else if (button.equals("Login")) {
45              url = loginServlet;
46          } else if (button.equals("Search")) {
47              url = searchServlet;
48          }
49      } finally {
50          RequestDispatcher rd = request.getRequestDispatcher(url);
51          rd.forward(request, response);
52
53          out.close();
54      }
55  }

```

How to Transfer Login Servlet

```

LoginServlet.java x
Source History
23  * @author kieukhanh
24  */
25  public class LoginServlet extends HttpServlet {
26
27      private final String searchPage = "search.html";
28      private final String invalidPage = "invalid.html";
29
30      /** Processes requests for both HTTP <code>GET</code> and <code>
39  protected void processRequest(HttpServletRequest request, HttpSer
40      throws ServletException, IOException {
41      response.setContentType("text/html;charset=UTF-8");
42      PrintWriter out = response.getWriter();
43      try {
44          String username = request.getParameter("txtUsername");
45          String password = request.getParameter("txtPassword");
46
47          RegistrationDAO dao = new RegistrationDAO();
48          boolean result = dao.checkLogin(username, password);
49
50          String url = invalidPage;
51
52          if (result) {
53              url = searchPage;
54          }
55          response.sendRedirect(url);
56      } catch (NamingException ex) {
57          ex.printStackTrace();
58      } catch (SQLException ex) {
59          ex.printStackTrace();
60      } finally {
  
```


How to Transfer Search Servlet

```

SearchServlet.java x
Source History
24 * @author kieuhanh
25 */
26 @WebServlet(name = "SearchServlet", urlPatterns = {"/SearchServlet"})
27 public class SearchServlet extends HttpServlet {
28     private final String searchPage = "search.html";
29     private final String showSearchResult = "ShowSearchResultServlet";
30     /** Processes requests for both HTTP <code>GET</code> and <code>POST
39     protected void processRequest(HttpServletRequest request, HttpServlet
40         throws ServletException, IOException {
41         response.setContentType("text/html;charset=UTF-8");
42         PrintWriter out = response.getWriter();
43         String url = searchPage;
44         String searchValue = request.getParameter("txtSearchValue");
45         try {
46             if (!searchValue.isEmpty()) {
47                 RegistrationDAO dao = new RegistrationDAO();
48                 dao.searchLastname(searchValue);
49                 List<RegistrationDTO> result = dao.getListAccounts();
50                 request.setAttribute("SEARCHRESULT", result);
51
52                 url = showSearchResult;
53             }
54         } catch (NamingException ex) {
55             ex.printStackTrace();
56         } catch (SQLException ex) {
57             ex.printStackTrace();
58         } finally {
59             RequestDispatcher rd = request.getRequestDispatcher(url);
60             rd.forward(request, response);
61             out.close();
62

```

How to Transfer Search Result Servlet

```

ShowSearchResultServlet.java x
Source History
19  *
20  * @author kieukhanh
21  */
22  @WebServlet(name = "ShowSearchResultServlet", urlPatterns = {"/ShowSearchResultServlet"})
23  public class ShowSearchResultServlet extends HttpServlet {
24
25      /** Processes requests for both HTTP GET and POST ..
34  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
35      throws ServletException, IOException {
36      response.setContentType("text/html;charset=UTF-8");
37      PrintWriter out = response.getWriter();
38      try {
39          /* TODO output your page here. You may use following sample code. */
40          out.println("<!DOCTYPE html>");
41          out.println("<html>");
42          out.println("<head>");
43          out.println("<title>Search Result</title>");
44          out.println("</head>");
45          out.println("<body>");
46          out.println("<h1>Search Result</h1>");
47
48          String searchValue = request.getParameter("txtSearchValue");
49
50          out.println("Your search value is " + searchValue);
51
52          List<RegistrationDTO> result =
53              (List<RegistrationDTO>) request.getAttribute("SEARCHRESULT");
54
  
```

How to Transfer Search Result Servlet



```

54
55     if (result != null) {
56         out.println("<table border='1'>");
57         out.println("<thead>");
58         out.println("<tr>");
59         out.println("<th>No.</th>");
60         out.println("<th>Username</th>");
61         out.println("<th>Password</th>");
62         out.println("<th>Lastname</th>");
63         out.println("<th>Role</th>");
64         out.println("</tr>");
65         out.println("</thead>");
66         out.println("<tbody>");
67         int count = 0;
68         for (RegistrationDTO dto : result) {
69             out.println("<tr>");
70             out.println("<td>");
71                 + ++count
72                 + ".</td>");
73             out.println("<td>");
74                 + dto.getUsername()
75                 + "</td>");
76             out.println("<td>");
77                 + dto.getPassword()
78                 + "</td>");
79             out.println("<td>");
80                 + dto.getLastname()
81                 + "</td>");
82             out.println("<td>");
83                 + dto.isRole()
84                 + "</td>");

```

```

85         out.println("</tr>");
86     }
87
88     out.println("</tbody>");
89     out.println("</table>");
90 } else {
91     out.println("<h2>No record is matched</h2>");
92 }
93
94 out.println("</body>");
95 out.println("</html>");
96 } finally {
97     out.close();
98 }
99

```