Fpt University

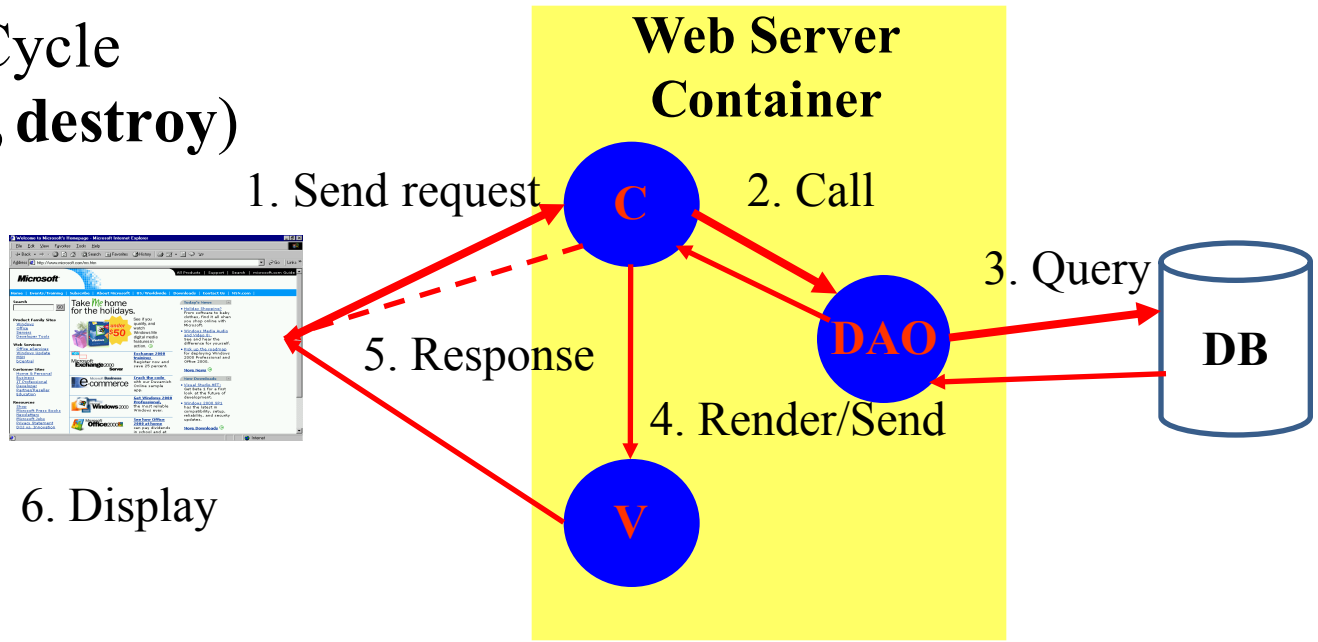
Web Applications & Web Containers

Web Applications
The Web Container Model

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

Review

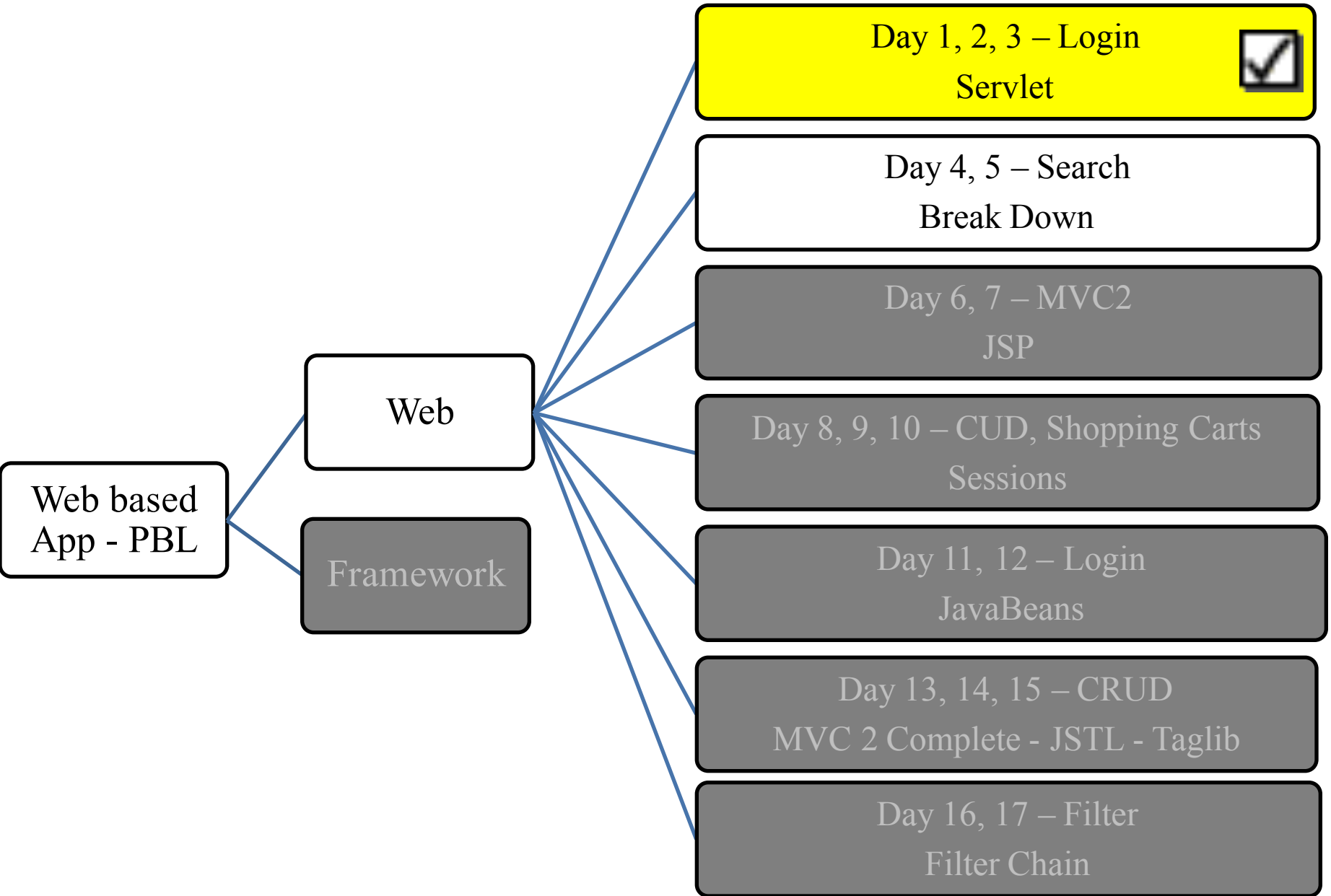
- **How to build the simple web site using html and servlet?**
  - Break down structure component in building web application
- **Some concepts**
  - Servlet vs. Java class, Parameter vs. Variable
  - Form Parameters
  - Http Protocol
  - HTTP Methods: **GET**, **POST**, …
  - Servlet Life Cycle (**init,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



Objectives

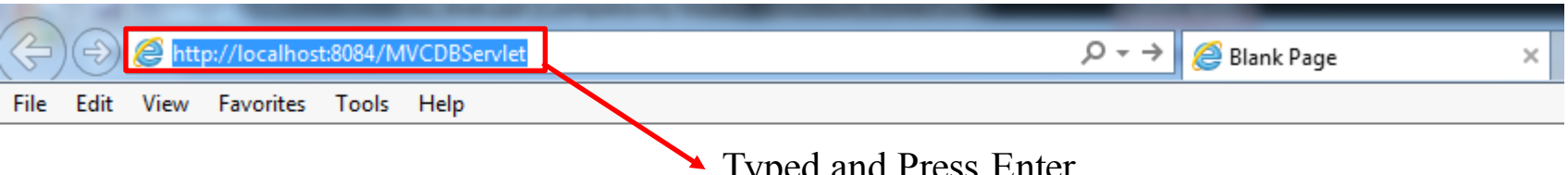
- **How to deploy the Web Application to Web Server without using Netbeans/ Eclipse tools?**
  - Web applications Structure
  - Request Parameters vs. Context Parameters vs. Config/Servlet Parameters
  - Application Segments vs. Scope
- **How to transfer from resources to others with/without data/objects?**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

Objectives

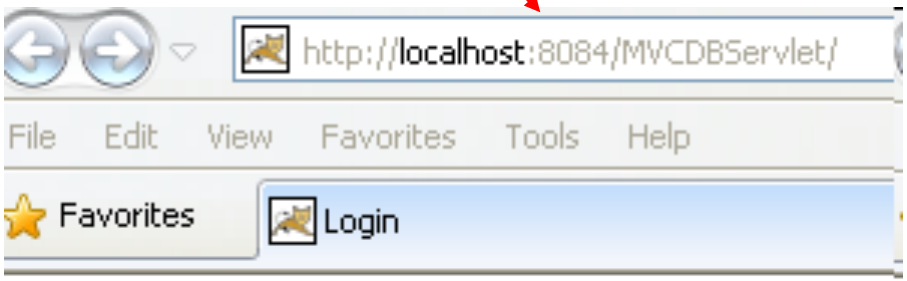

Web based App - PBL
Web
Framework
Day 1, 2, 3 – Login
Servlet
Day 4, 5 – Search
Break Down
Day 6, 7 – MVC2
JSP
Day 8, 9, 10 – CUD, Shopping Carts
Sessions
Day 11, 12 – Login
JavaBeans
Day 13, 14, 15 – CRUD
MVC 2 Complete - JSTL - Taglib
Day 16, 17 – Filter
Filter Chain

Deploy Application

Expectation



Typed and Press Enter



Login Page

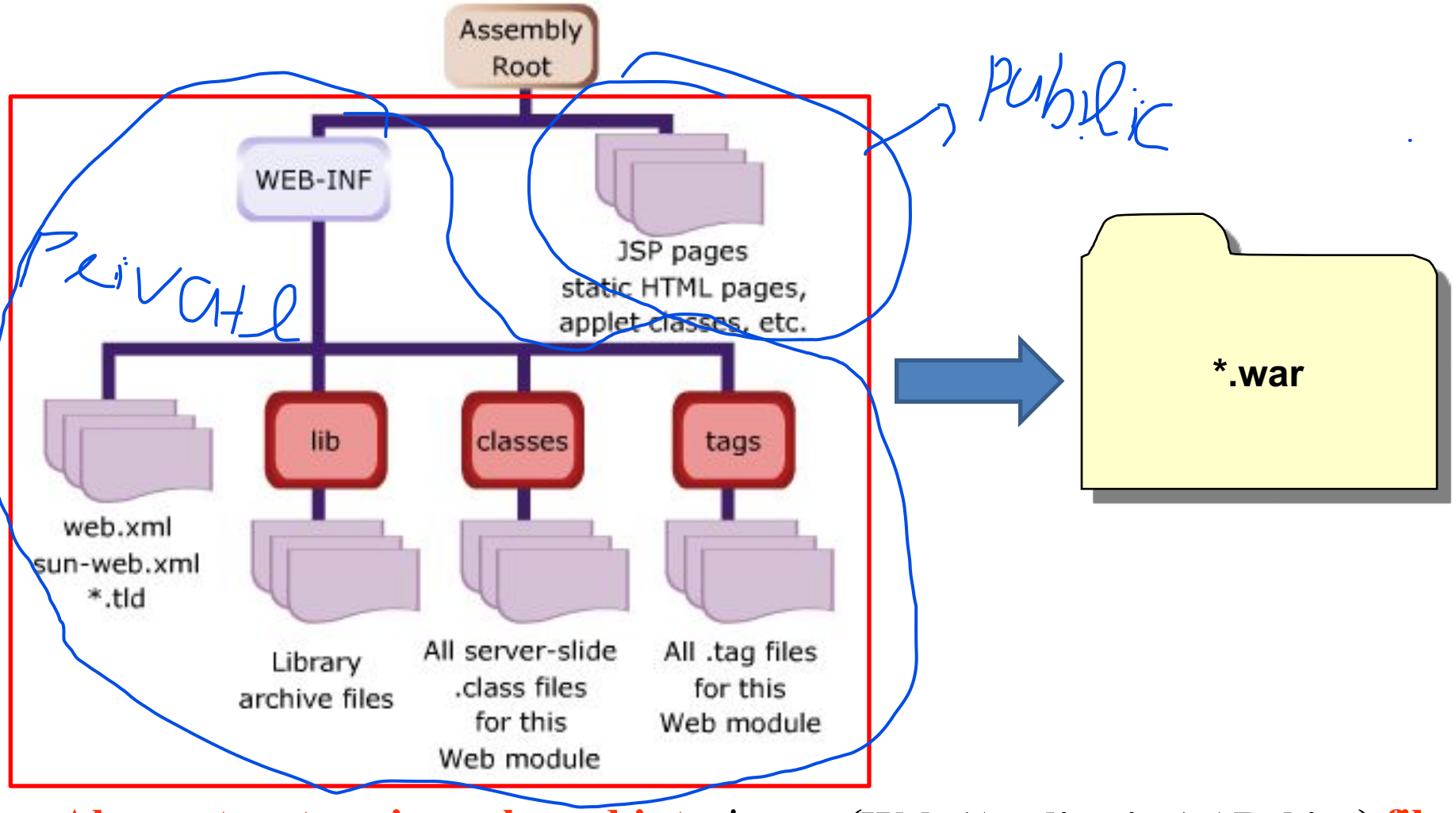
Username

Password

Login Reset

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

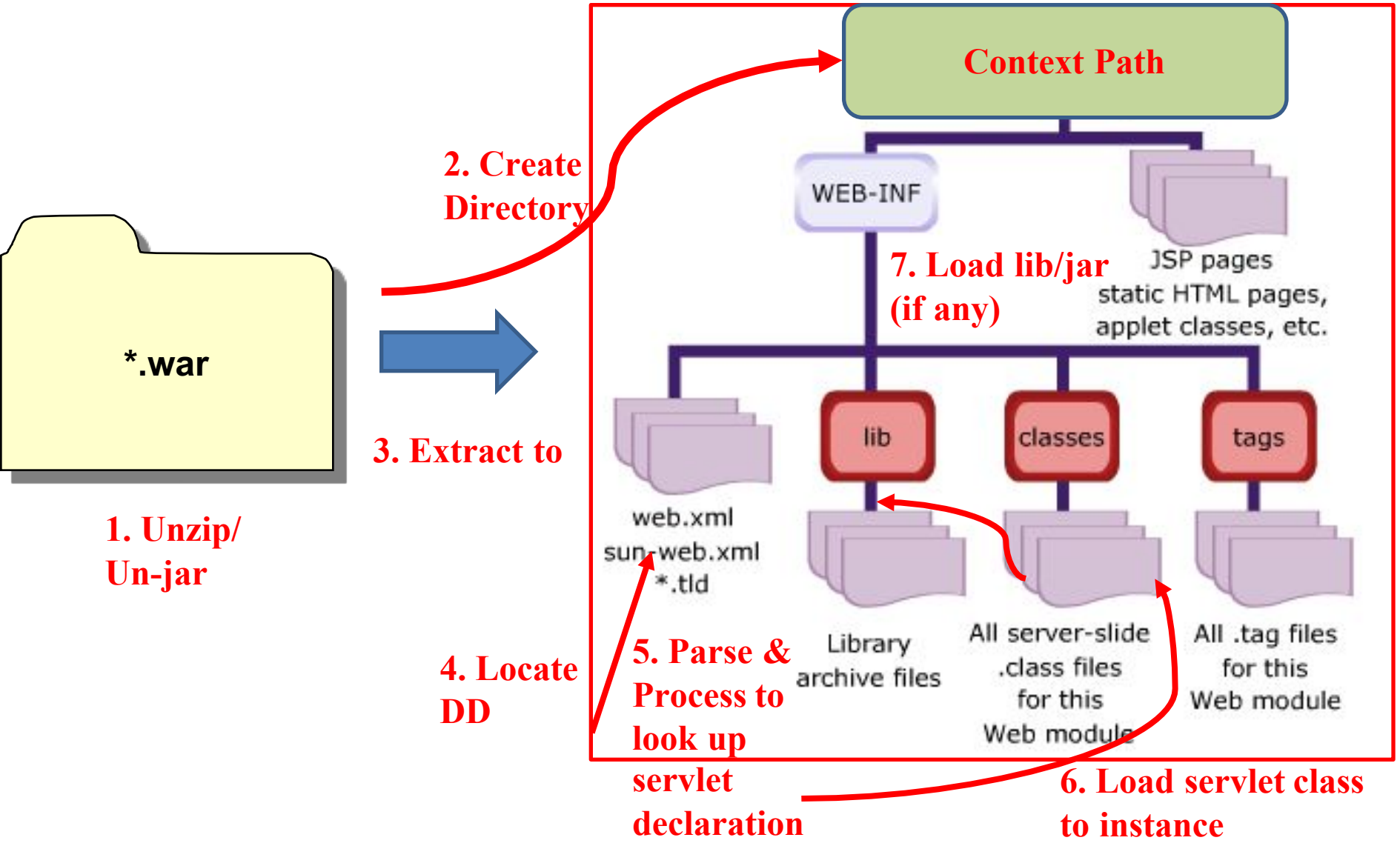
Fpt University

Web Applications

File and Directory Structure

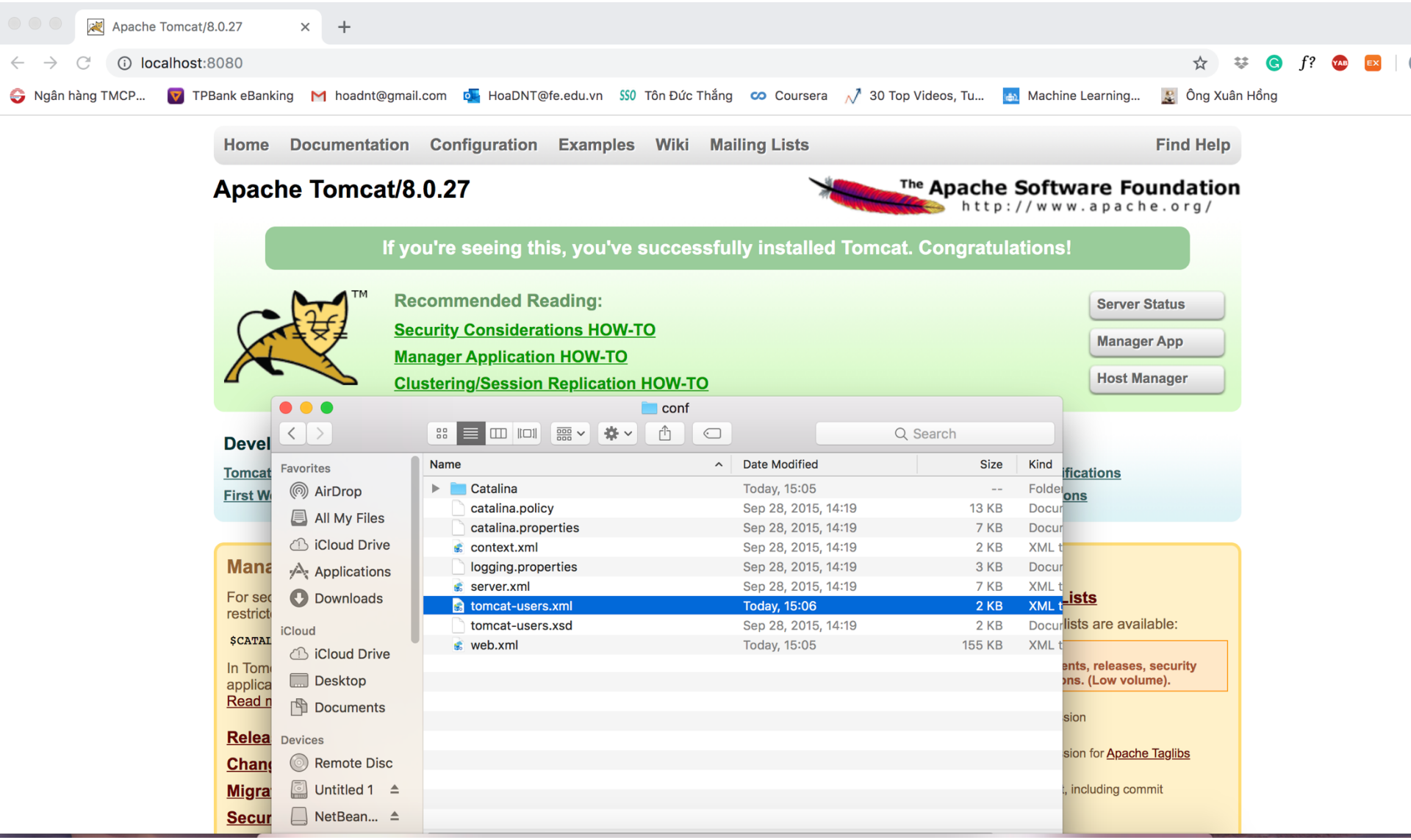
- A Place for Everything and Everything in Its Place.
  - On Tomcat Server, it locates at **CATALINA_HOME/webapps**
  - **Execute**: http://host:port/webappcontext/resourceIneed
- Construct the file and directory structure of a Web **Application** that may **contain**:
  - Static content,
  - JSP pages,
  - 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

Deploy Mechanism



Web Applications

Deploy Mechanism

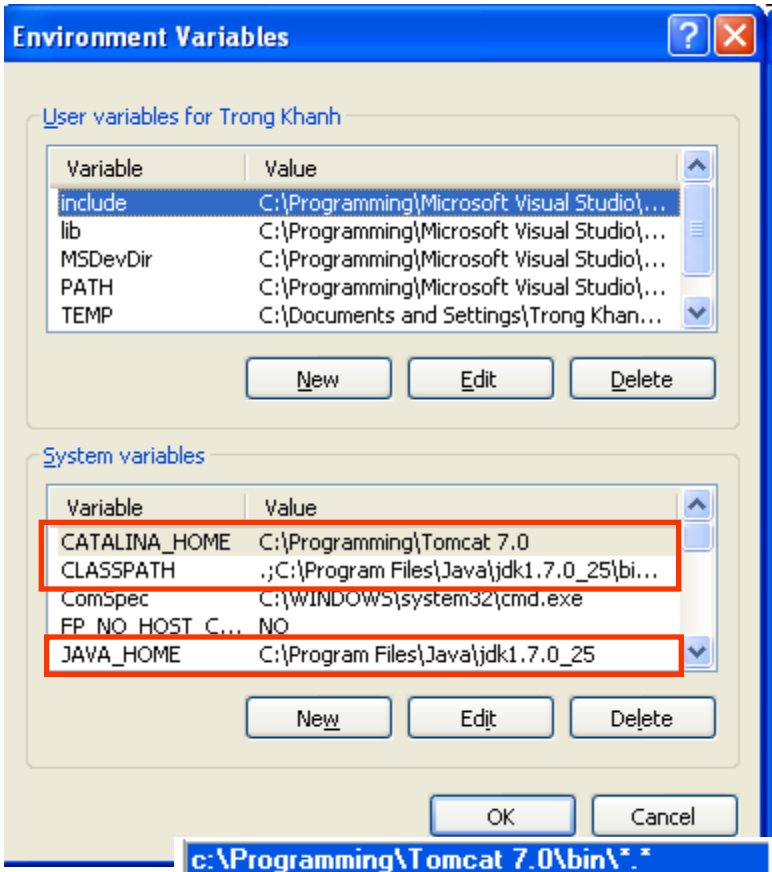
tomcat-users.xml

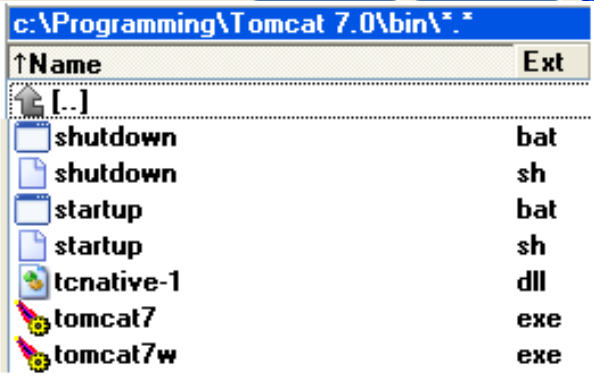
Applications > NetBeans > apache-tomcat-8.0.27 > conf > tomcat-users.xml

```
20    NOTE:  By default, no user is included in the "manager-gui" role required
21    to operate the "/manager/html" web application.  If you wish to use this app,
22    you must define such a user - the username and password are arbitrary.
23  -->
24      <!--
25    NOTE:  The sample user and role entries below are wrapped in a comment
26    and thus are ignored when reading this file. Do not forget to remove
27    <!.. ..> that surrounds them.
28  -->
29      <!--
30    <role rolename="tomcat"/>
31    <role rolename="role1"/>
32    <user username="tomcat" password="tomcat" roles="tomcat"/>
33    <user username="both" password="tomcat" roles="tomcat,role1"/>
34    <user username="role1" password="tomcat" roles="role1"/>
35  -->
36      <user password="1" roles="manager-script,admin,manager-gui" username="admin"/>
37  </tomcat-users>
38
```

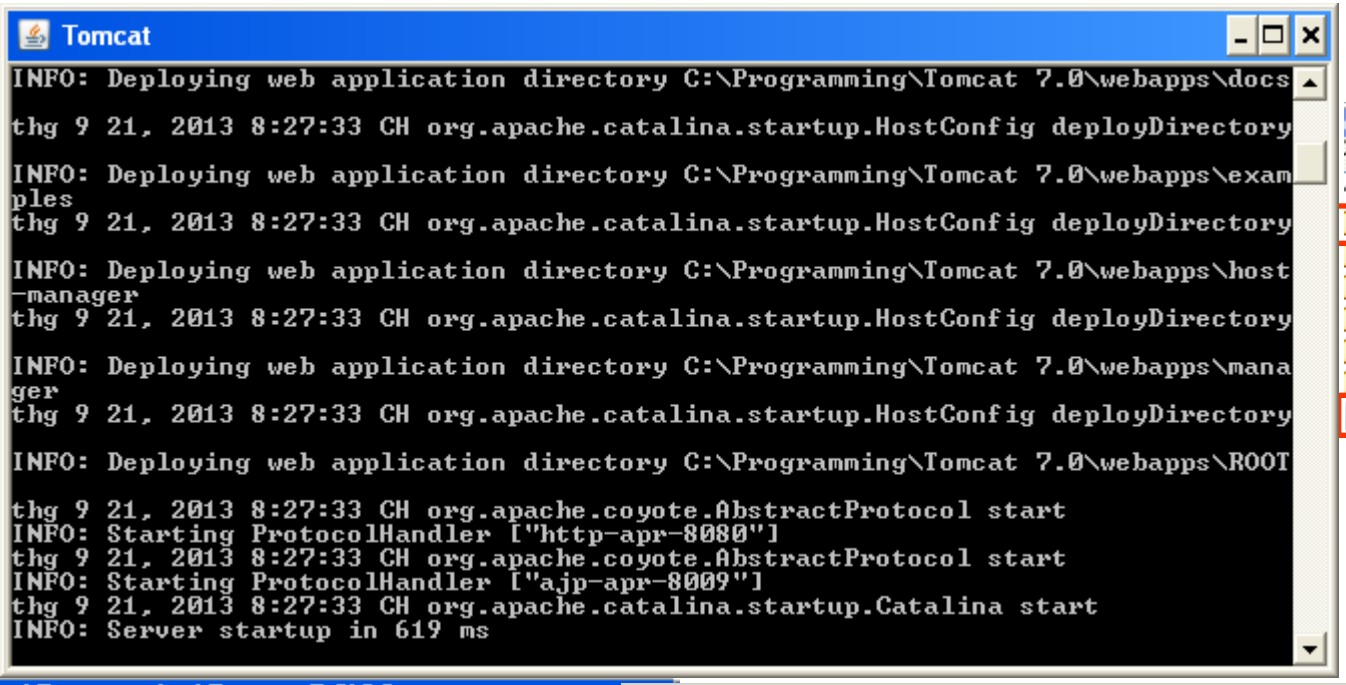
Web Applications
Manual Deploying

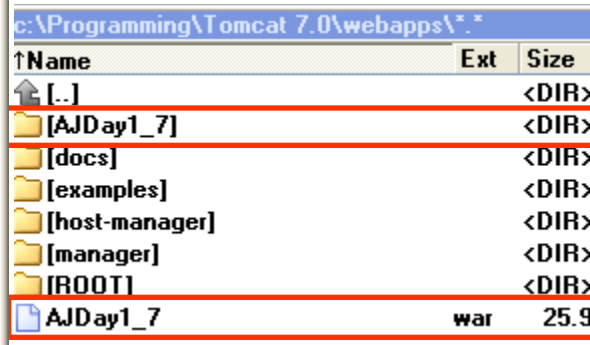
- Setup the environment for JAVA and TOMCAT
  - **Win XP**: click Properties of "My Computer", Choose Advanced, Click "Environment Variables", to set following environment variables
  - **Win Vista and Win 7**: click Properties of Computer, choose "Advanced System Setting", choose Advanced, Click "Environment Variables", to set following environment variables
- Go to the **Installed_Tomcat\bin** directory, click **startup.bat** or **tomcat7w.exe**

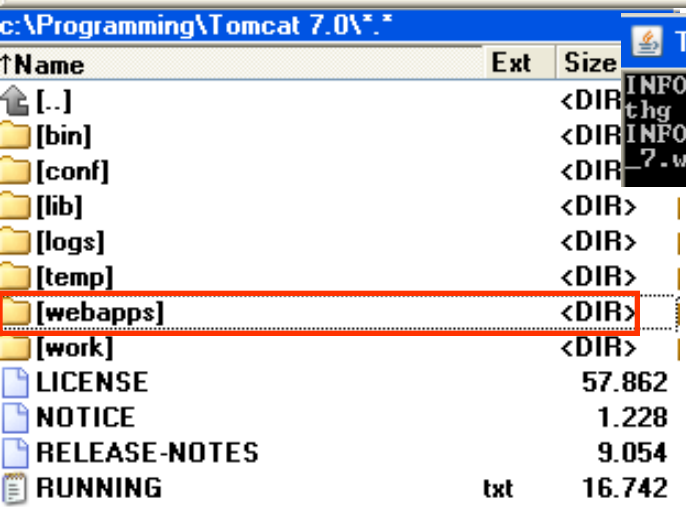


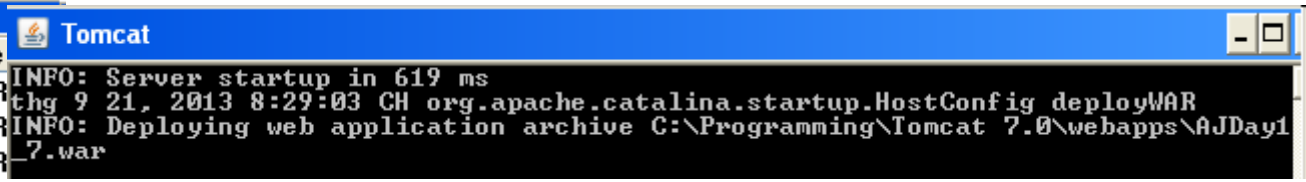


Web Applications
Manual Deploying









- 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

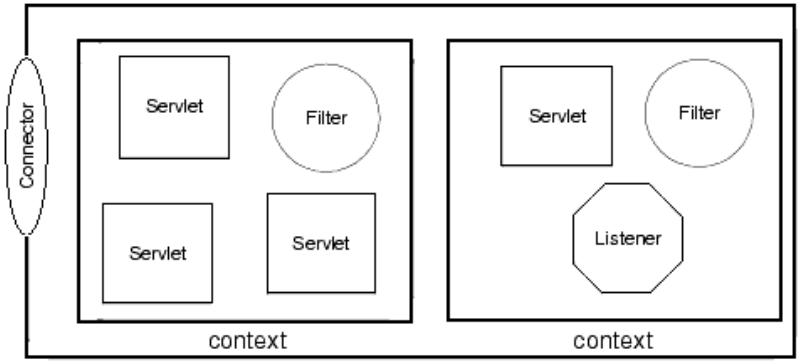
The Servlet Container

- Fortunately(lucky),
  - 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

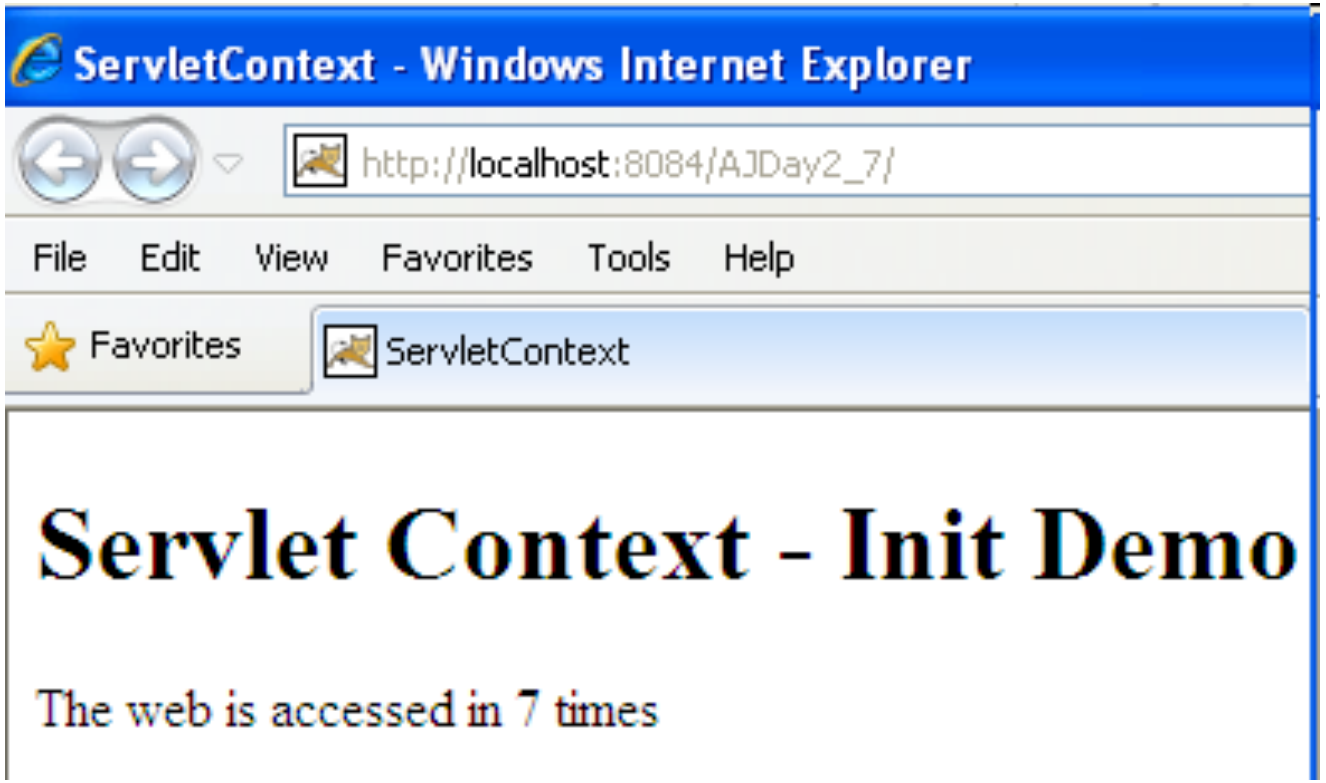
- Providing some fundamental information available to all the dynamic resources (servlets, JSP) within the web application is allowed by
  - **Using** servlet **initialization parameters in** the **deployment descriptor** with the **getInitParameter(String parName)** method to provide **initialization 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

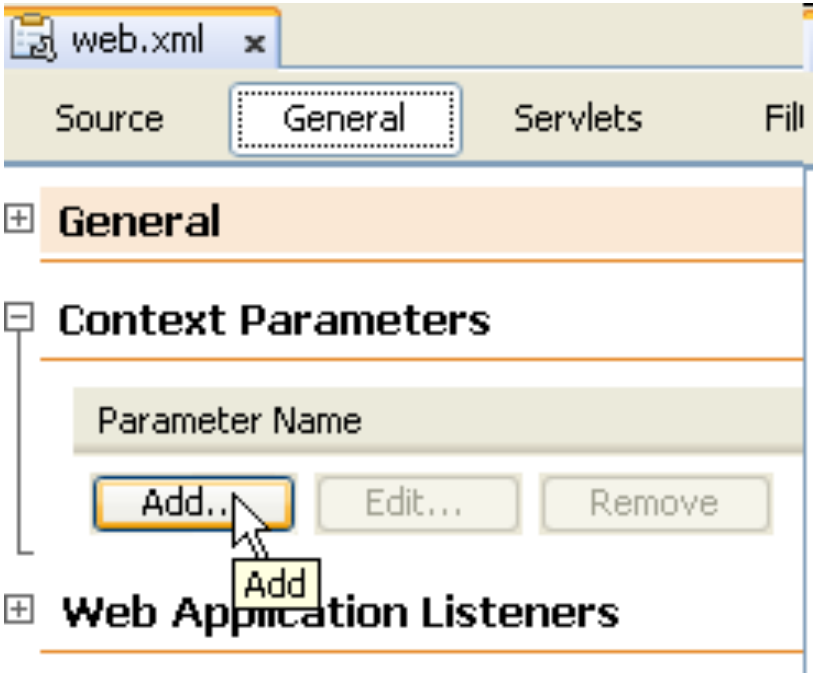


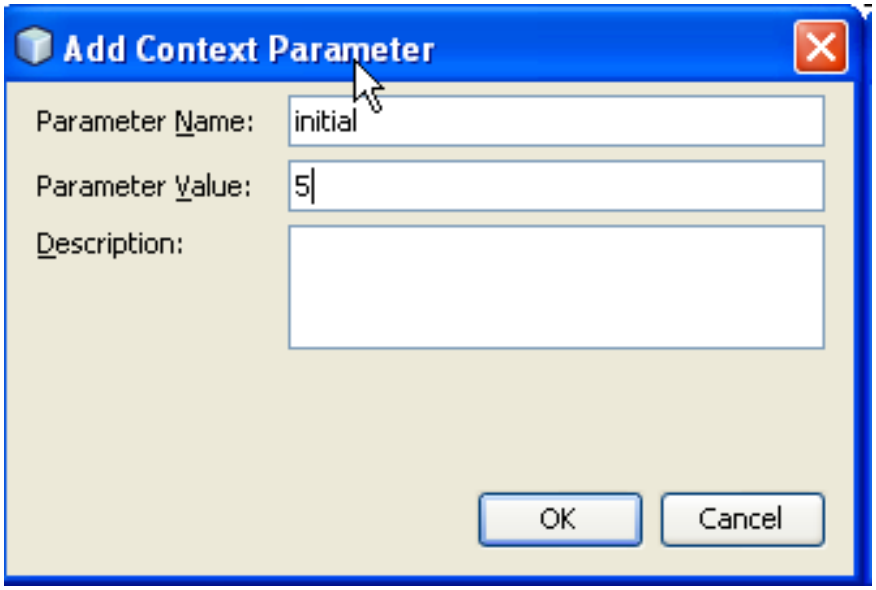
The Web Container Model

The ServletContext – Initialization Parameters

- Writing Code to Retrieve ServletContext Initialization Parameters

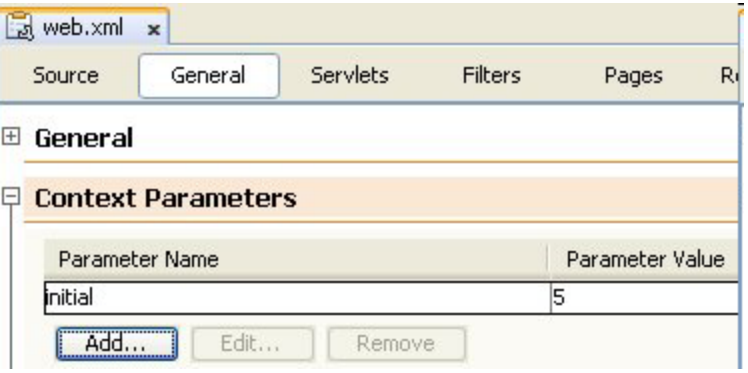
```
ServletContext sc = getServletContext();
String var = sc.getInitParameter("parName");
```

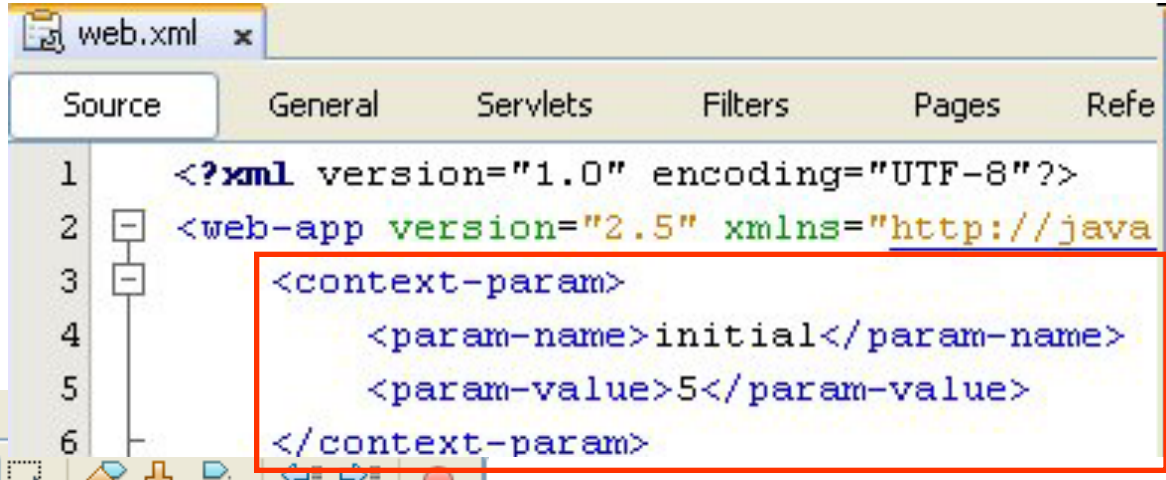




The Web Container Model

The ServletContext – Initialization Parameters





web.xml | InitCounter.java

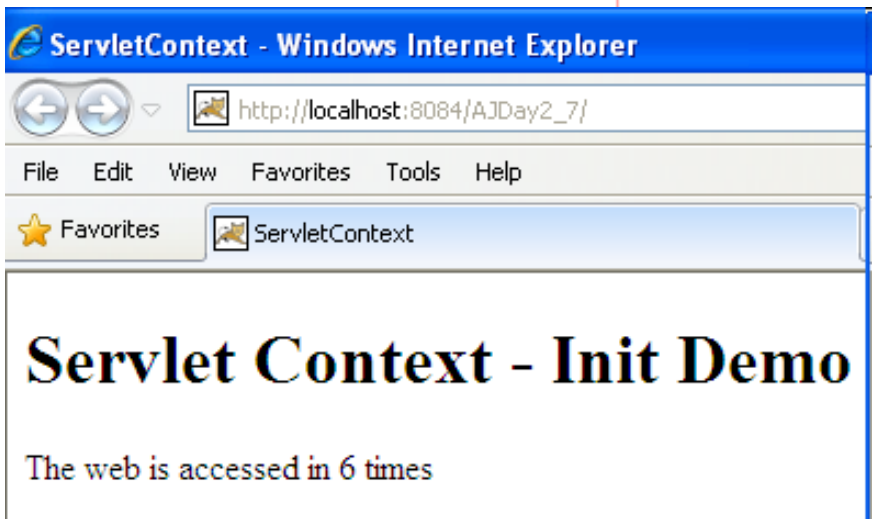
Source | History

```
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 Web Container Model

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 Web Container Model

The ServletContext – **Initialization Parameters**

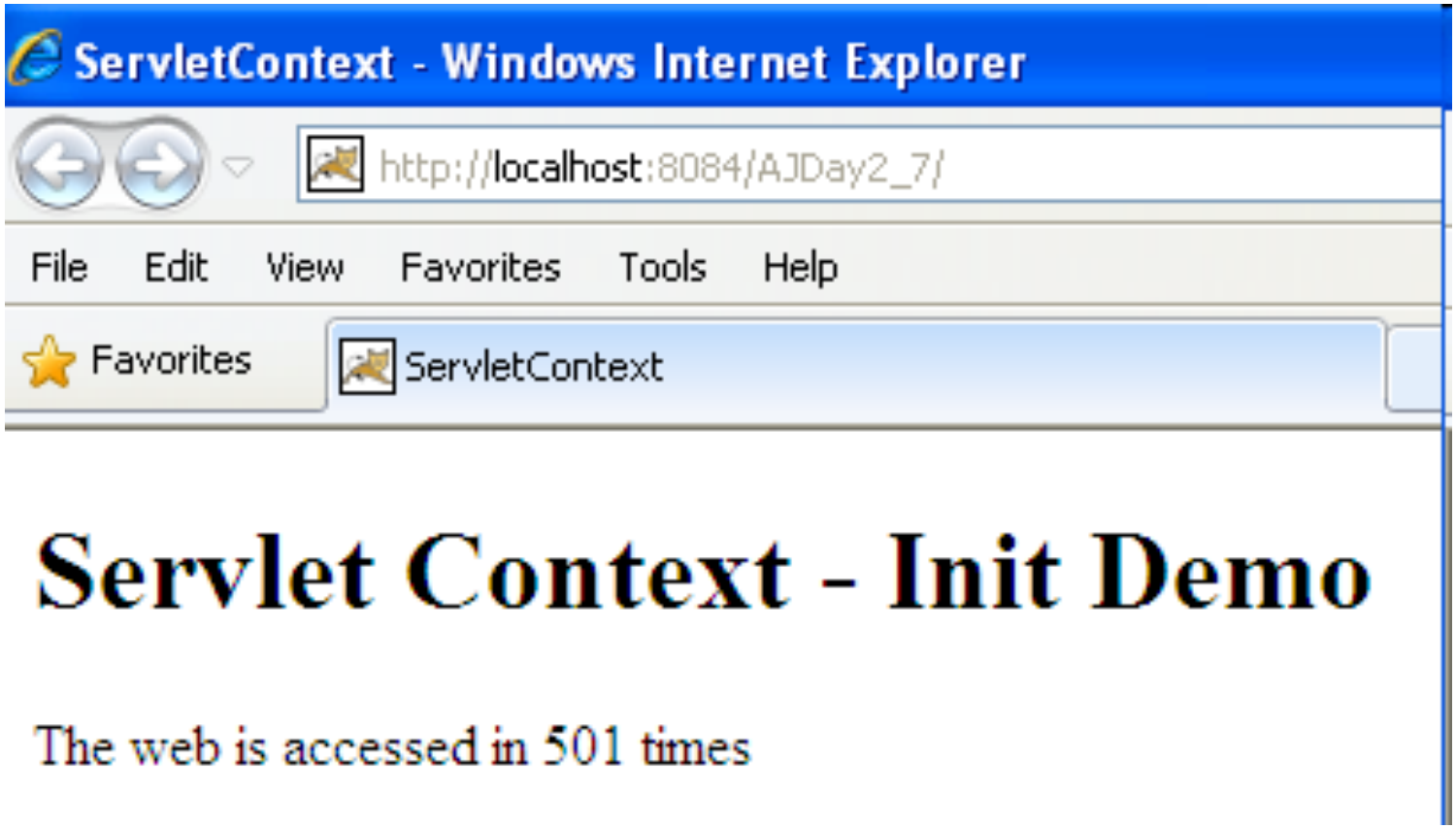
web.xml

Source | General | Servlets | Filters | Pages | Refer

```
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**



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	- **public String getServletName()** - Searches the configuration information and retrieves name of the servlet instance - String servletName = getServletName();
getInitParameter	- **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	- **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

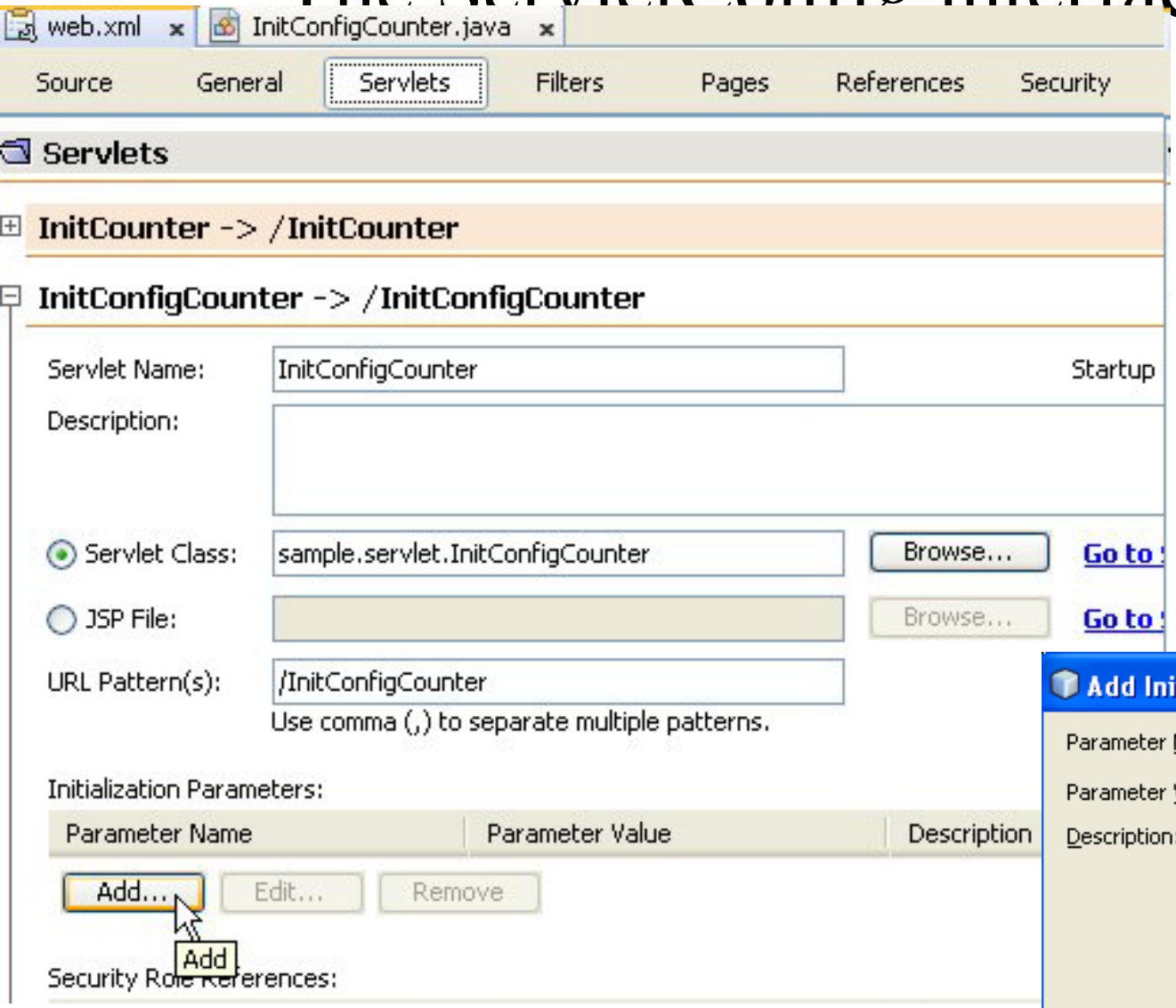
```
ServletConfig sc = getServletConfig();
String name = sc.getInitParameter(“parName");
```

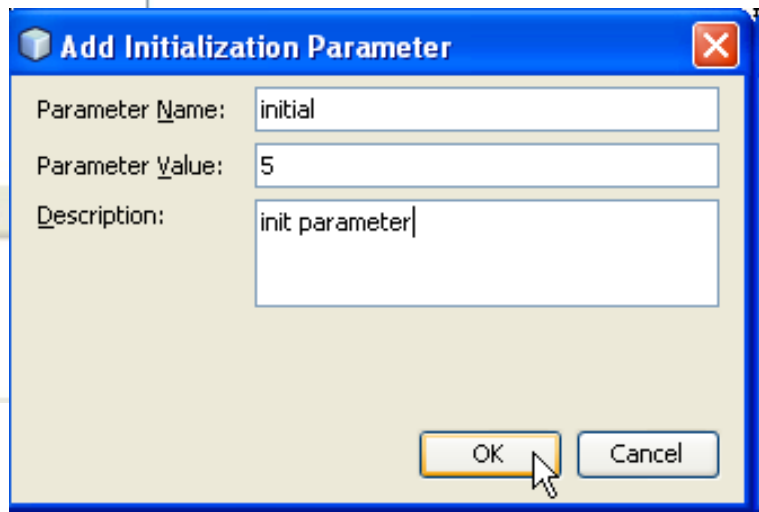
The Web Container Model

The ServletConfig interface – Example

The Web Container Model

The ServletConfig interface – Example





The Web Container Model

The ServletConfig interface – Example

web.xml | InitConfigCounter.java

Source | General | Servlets | Filters | Pages | References | Security | History

```
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 Web Container Model

The ServletConfig interface – Example

web.xml | InitConfigCounter.java

Source | History

```
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 Web Container Model

The ServletConfig interface – Example

```
42      protected void processRequest(HttpServletRequest request, HttpServle
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      }
```

The Web Container Model

The ServletConfig interface – Example

```
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          <init-param>
24              <description>init parameter</description>
25              <param-name>initial</param-name>
26              <param-value>5000</param-value>
27          </init-param>
28      </servlet>
29      <servlet-mapping>
```

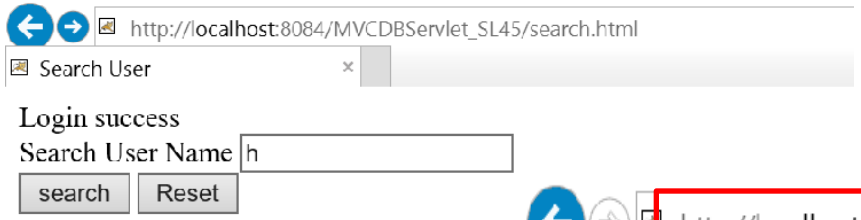
How To Transfer

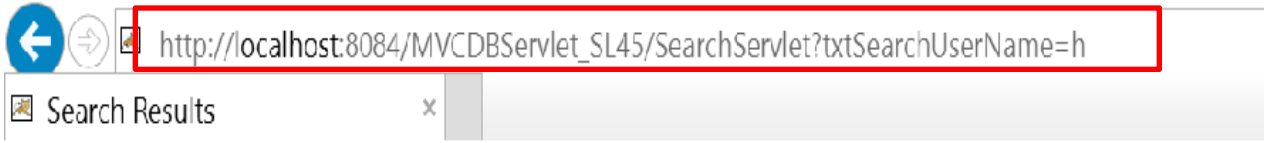
Requirements

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

How To Transfer

Expectation



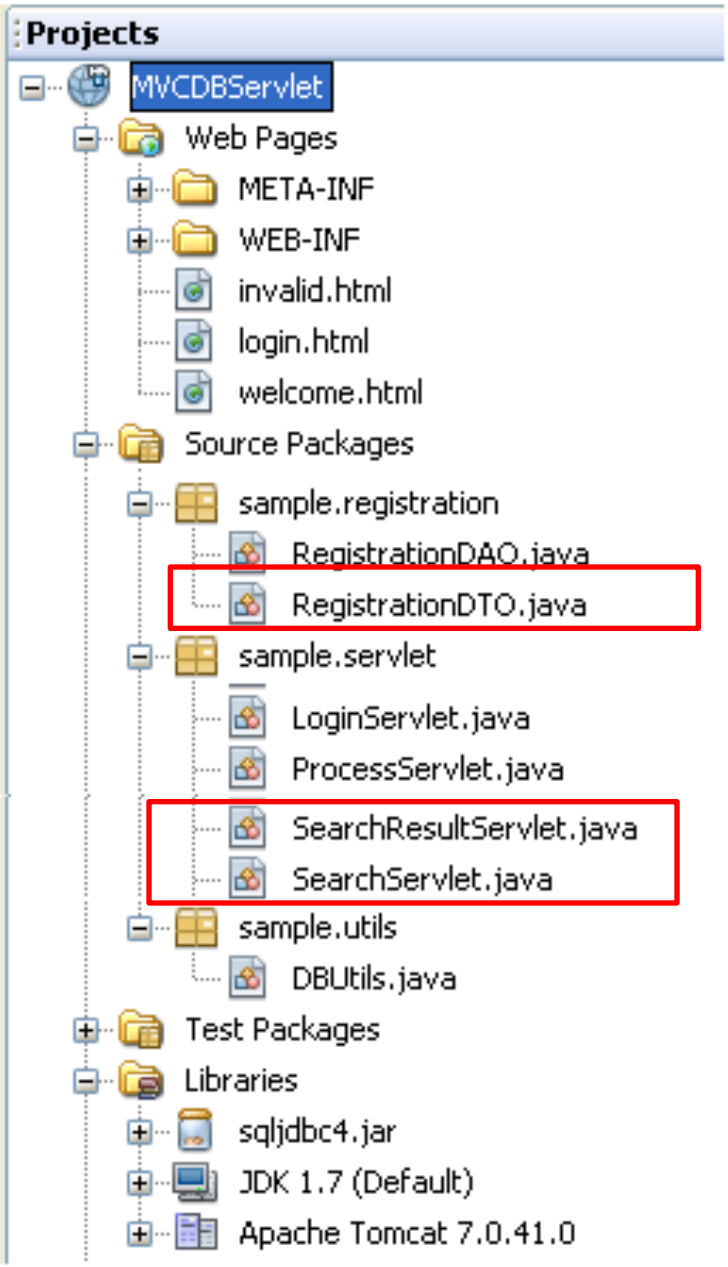


Search value: h

No.	UserID	Name	Password	Role
1	hoadnt	Hòa	123	AD
2	SE123	Hòa Đoàn	123	AD
3	Se124	Hồng Kim	123	ST

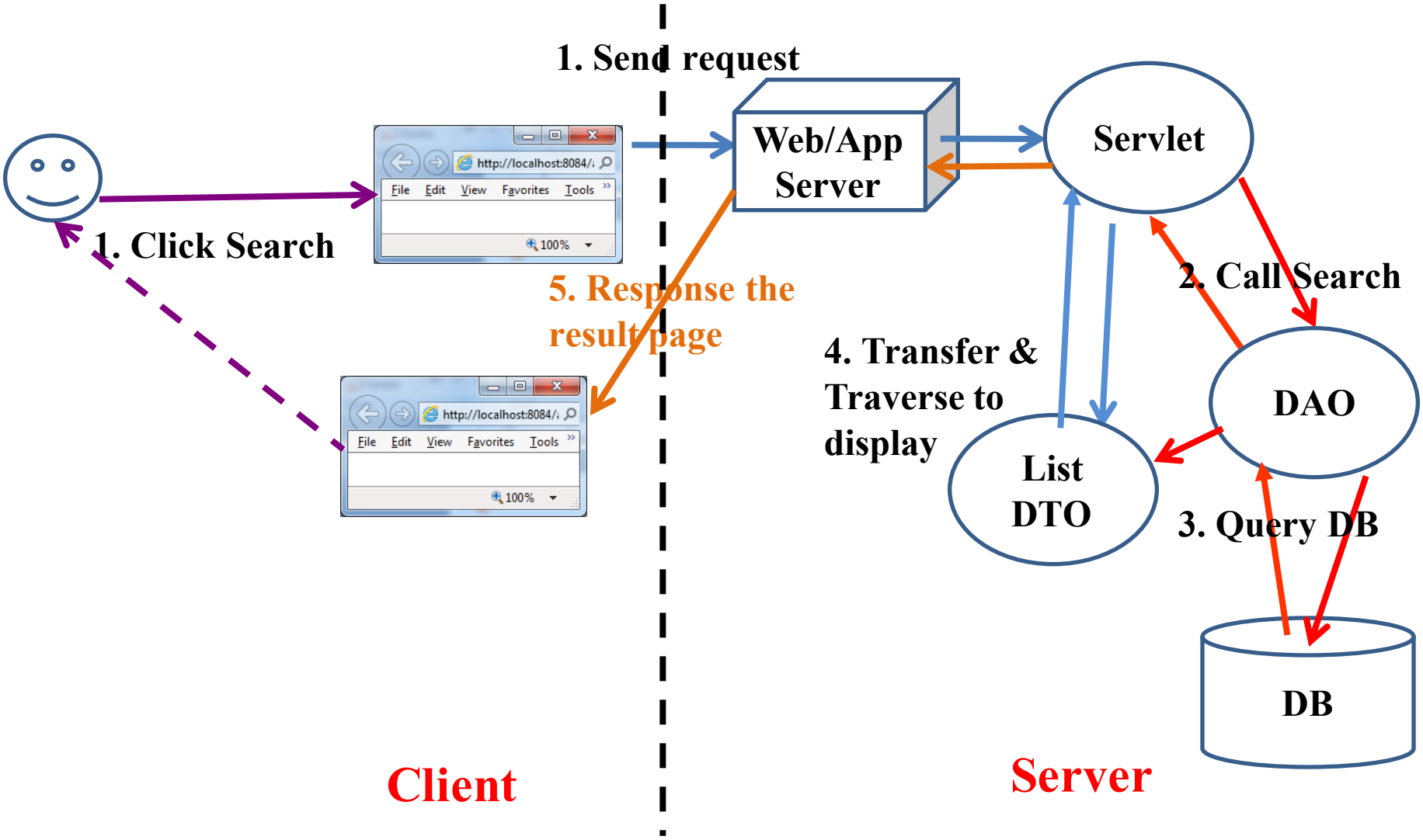
How To Transfer

Expectation



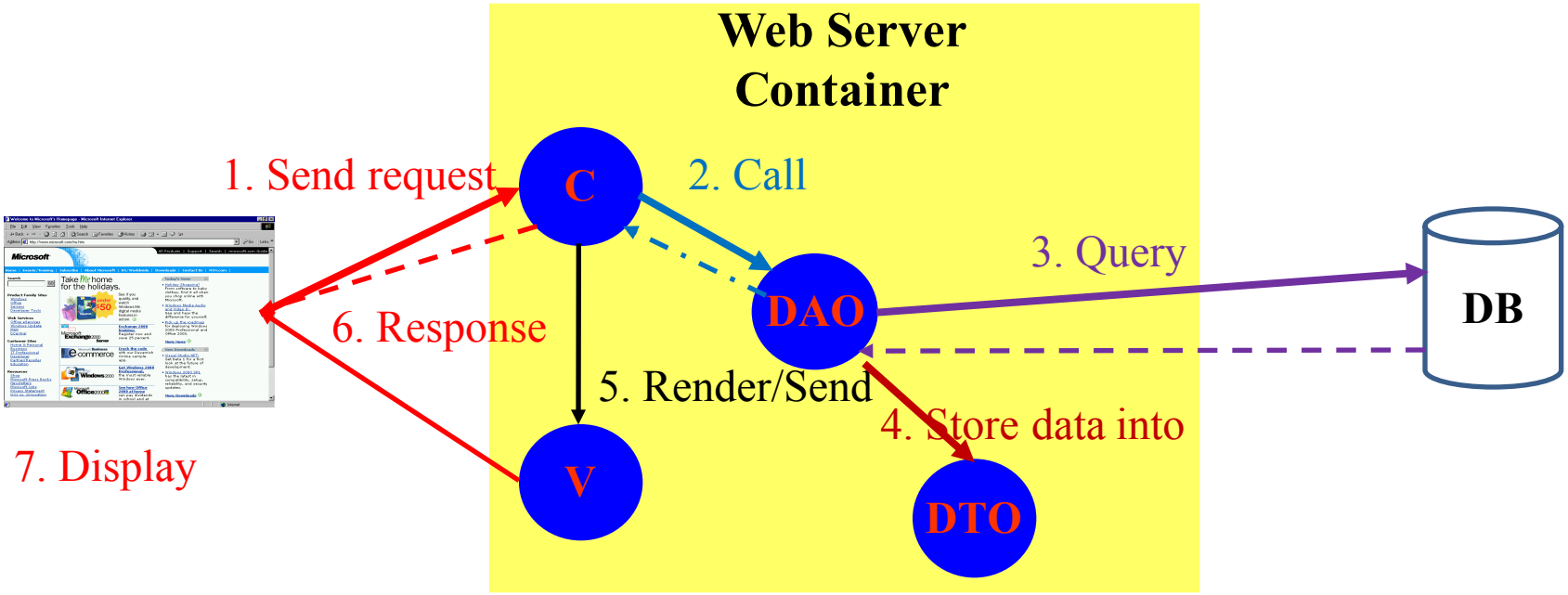
How To Transfer

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 **deliveres** the HTTP **response**.
    - **javax.servlet.ServletRequest**
  - **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**: (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	- **public Object getAttribute(String name)** - returns the value of the name attribute as Object - Ex: String user = (String)servletContext.getAttribute("USER");
setAttribute	- **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	- **public void removeAttribute(String name)** - Removes the name attributes - Ex: servletContext.removeAttribute("USER");
getAttributeNames	- **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

web.xml | requestDispatcher.html

Source | History

```
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>");
```

The Web Container Model

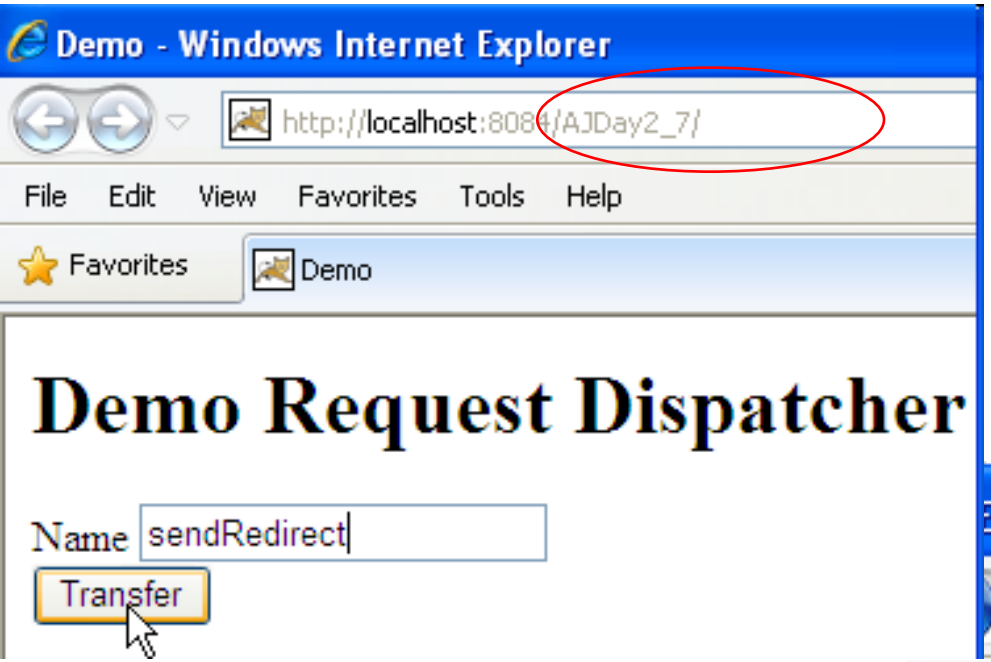
Need for using RequestDispatcher – Redirect

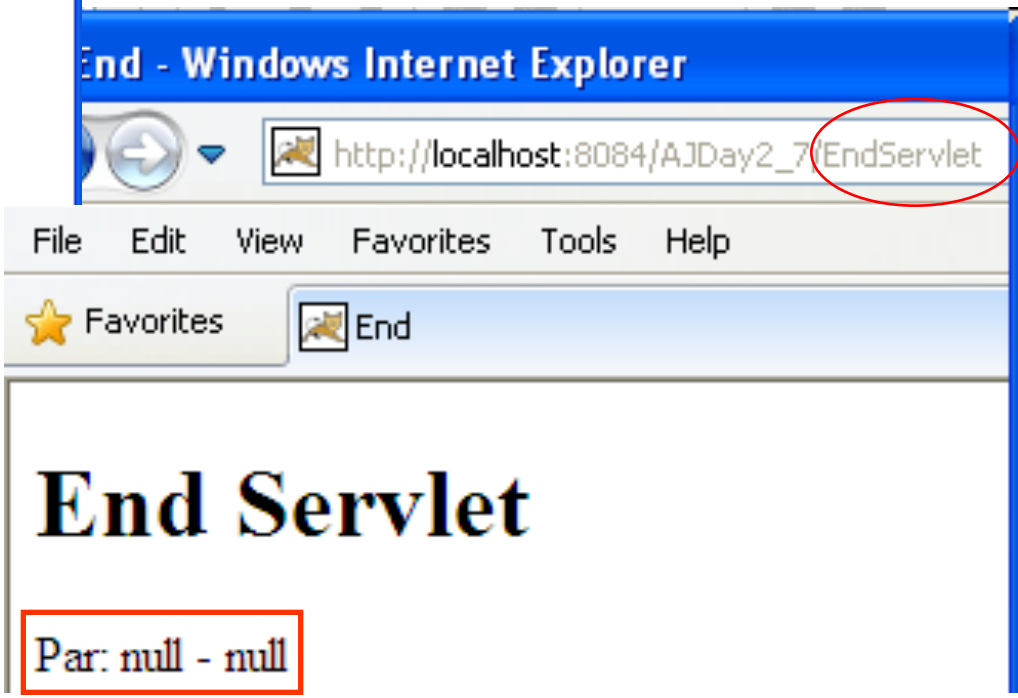
MiddleServlet.java | EndServlet.java

```
28      protected void processRequest(HttpServletRequest request, HttpSe
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

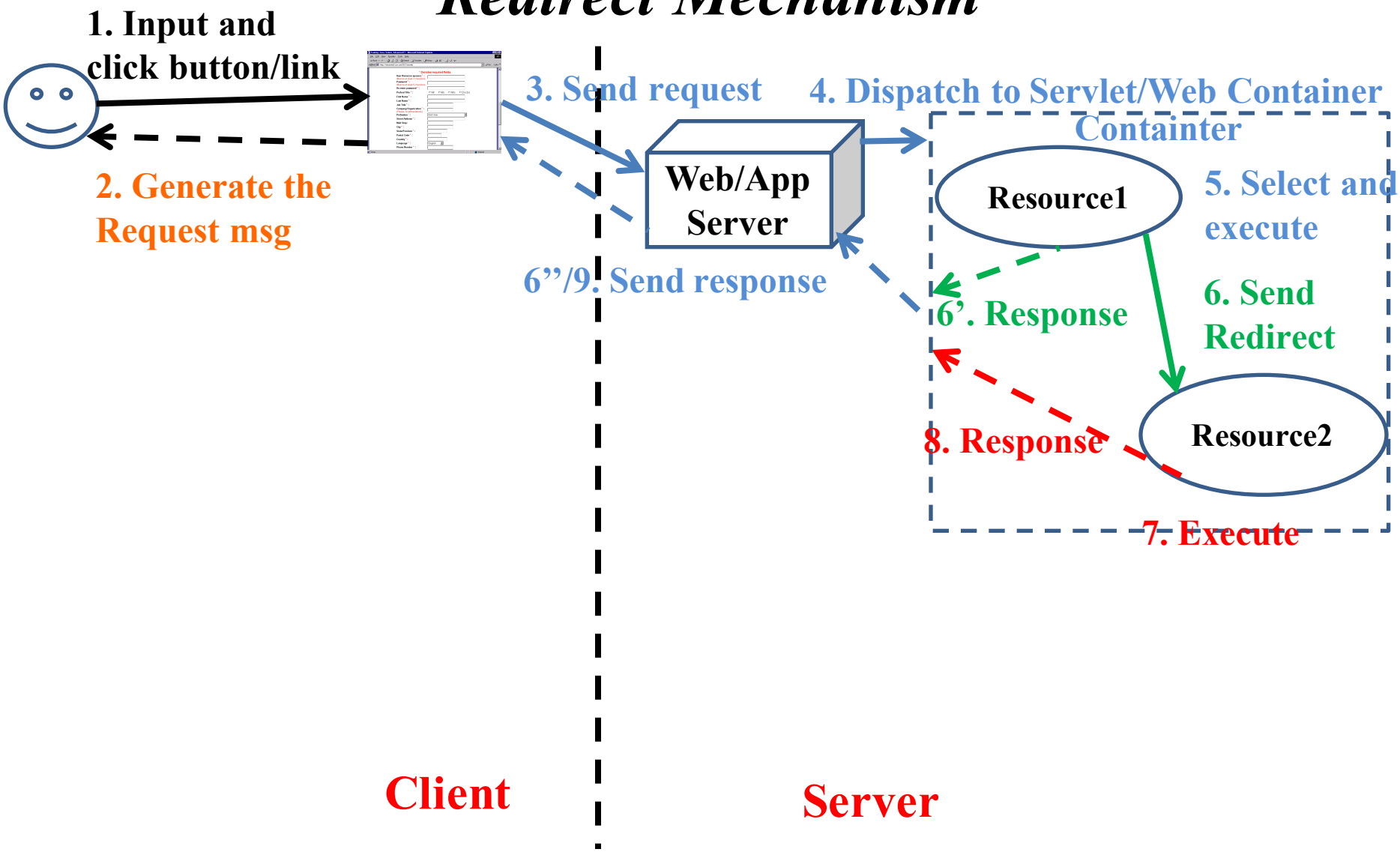




The Web Container Model

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 **getRequestDispacher(String path) method**
  - **Returns** RequestDispacher 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 **RequestDispacher 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	- **Redirect** the **output** to another servlet - **Forward** the **request** to another Servlet to process the client request. - **Ex**: RequestDispatcher rd = request.getRequestDipatcher("home.jsp"); rd.forward(request, response);
include	- **Include** the **content** of another servlet into the current output stream - Include the **output** of another Servlet to process the client request - **Ex** RequestDispatcher rd = request.getRequestDipatcher("home.jsp"); rd.include (request, response);

The Web Container Model

Using RequestDispatcher – Example

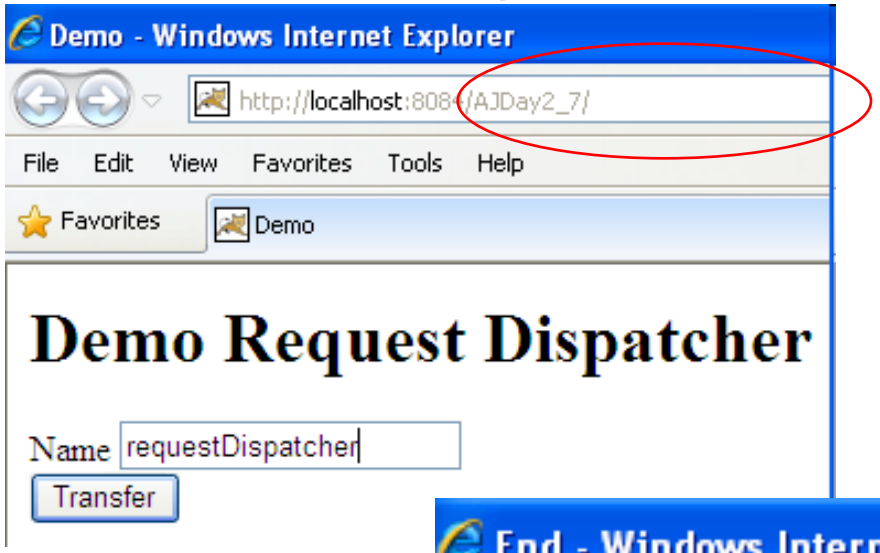
```
<body>
    <h1>Demo Request Dispatcher</h1>
    <form action="MiddleServlet">
        Name <input type="text" name="txtName" value="" /><br/>
        <input type="submit" value="Transfer" />
    </form>
</body>
```

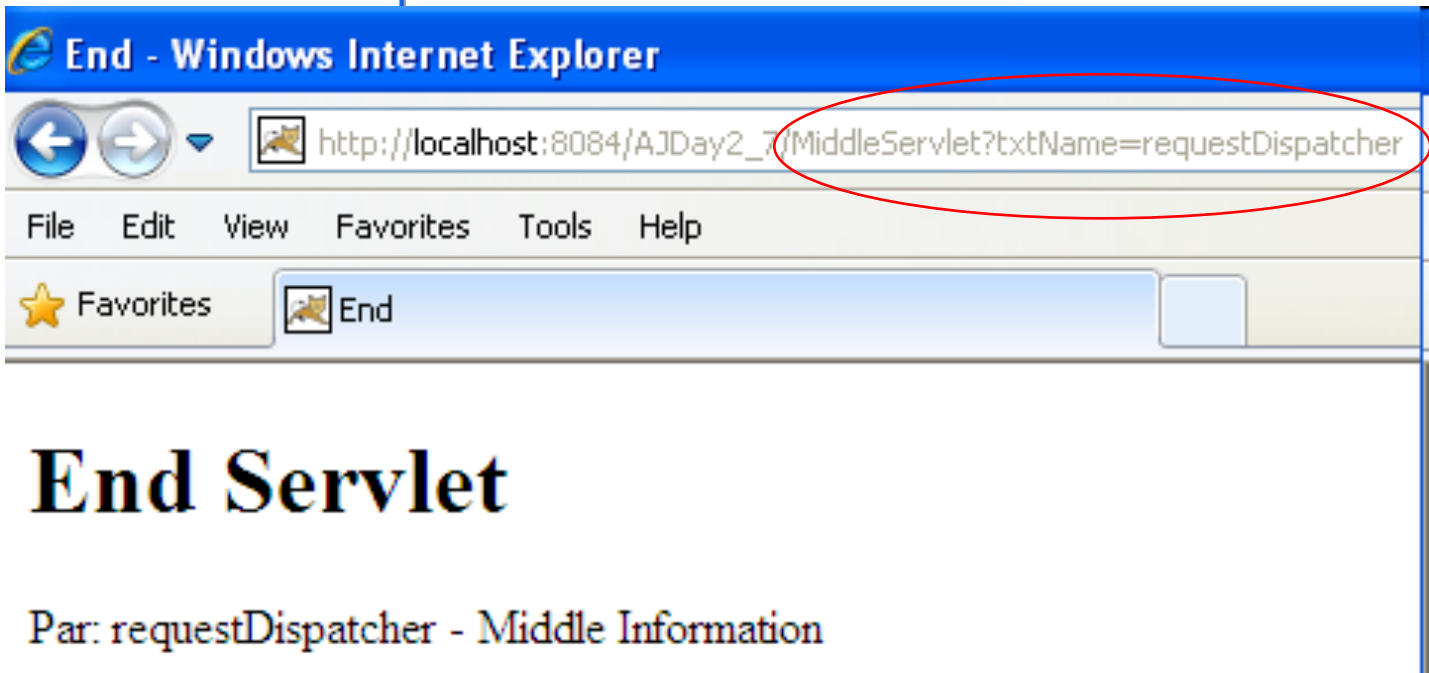
MiddleServlet.java | EndServlet.java

```
28          */
29         protected void processRequest(HttpServletRequest request, HttpServletRespo
30         throws ServletException, IOException {
31             response.setContentType("text/html;charset=UTF-8");
32             PrintWriter out = response.getWriter();
33             try {
34
35                 out.println("<html>");
36                 out.println("<head>");
37                 out.println("<title>Middle</title>");
38                 out.println("</head>");
39                 out.println("<body>");
40                 out.println("<h1>Middle Servlet</h1>");
41
42                 request.setAttribute("Middle", "Middle Information");
43
44                 RequestDispatcher rd = request.getRequestDispatcher("EndServlet");
45                 rd.forward(request, response);
46
47                 out.println("</body>");
48                 out.println("</html>");
49
50             } finally {
```

The Web Container Model

Using RequestDispatcher – Example

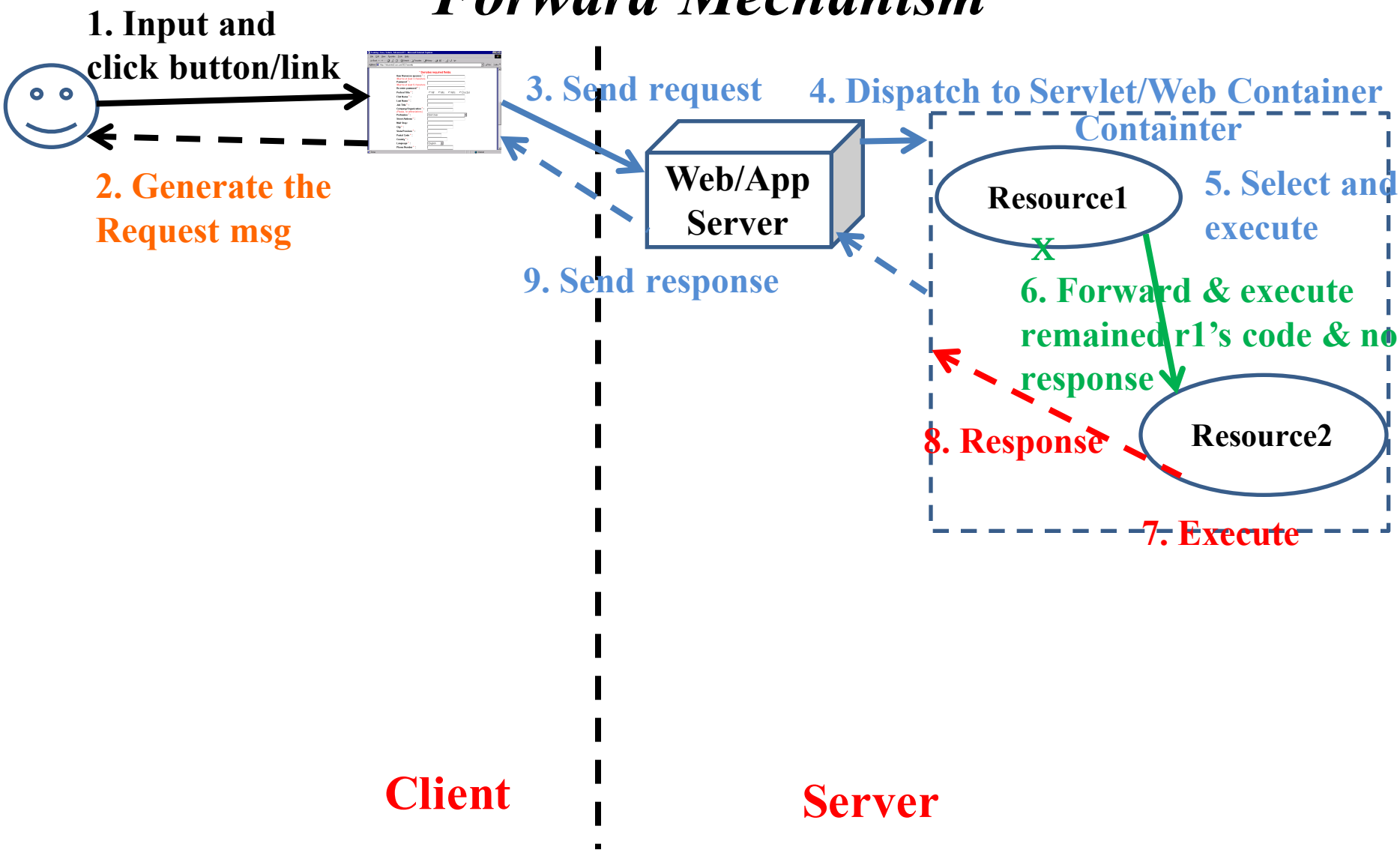




The Web Container Model

Need for using RequestDispatcher

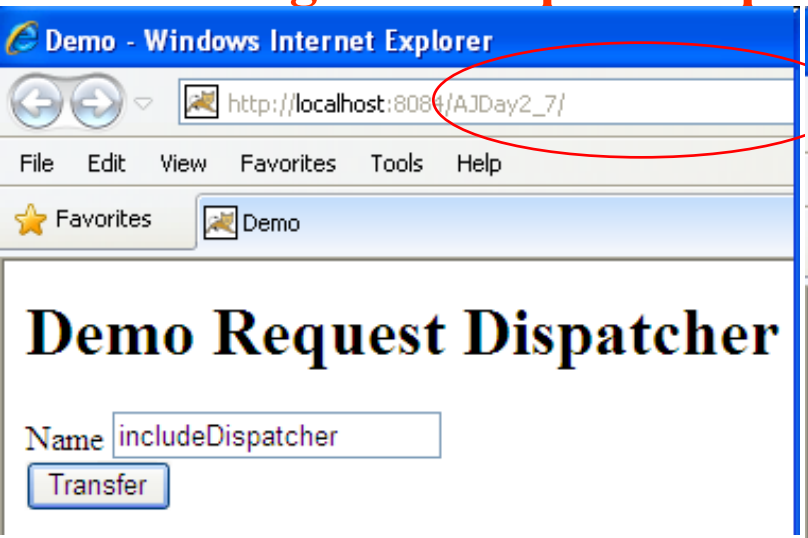
Forward Mechanism

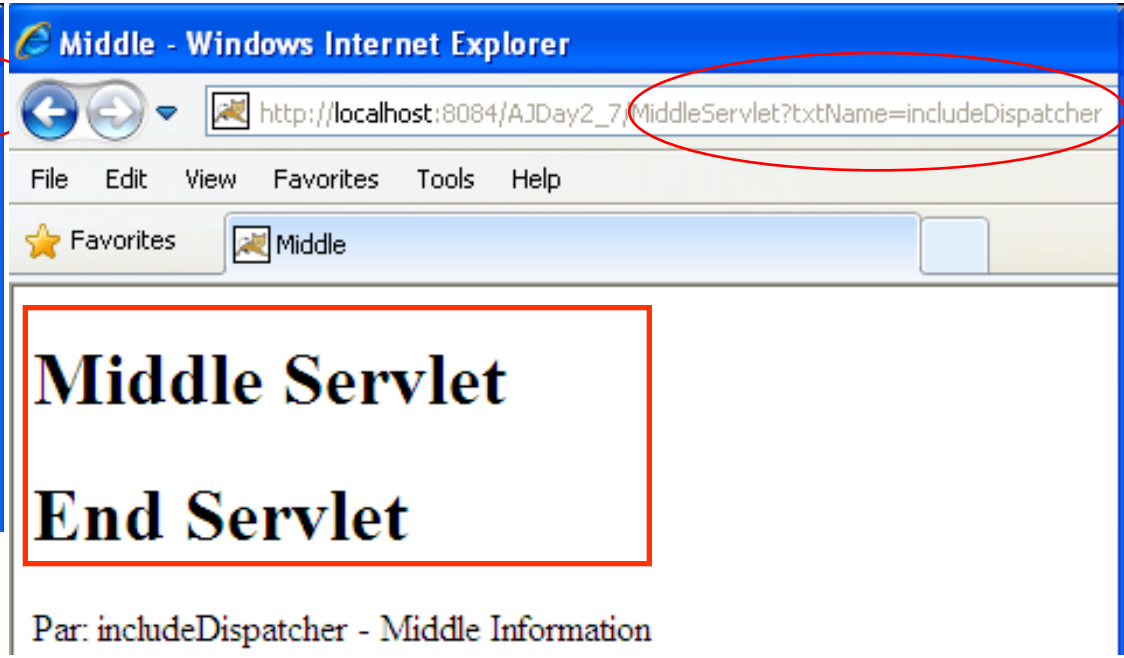


The Web Container Model

Using RequestDispatcher – Example

Change the RequestDispatch – forward method to include method



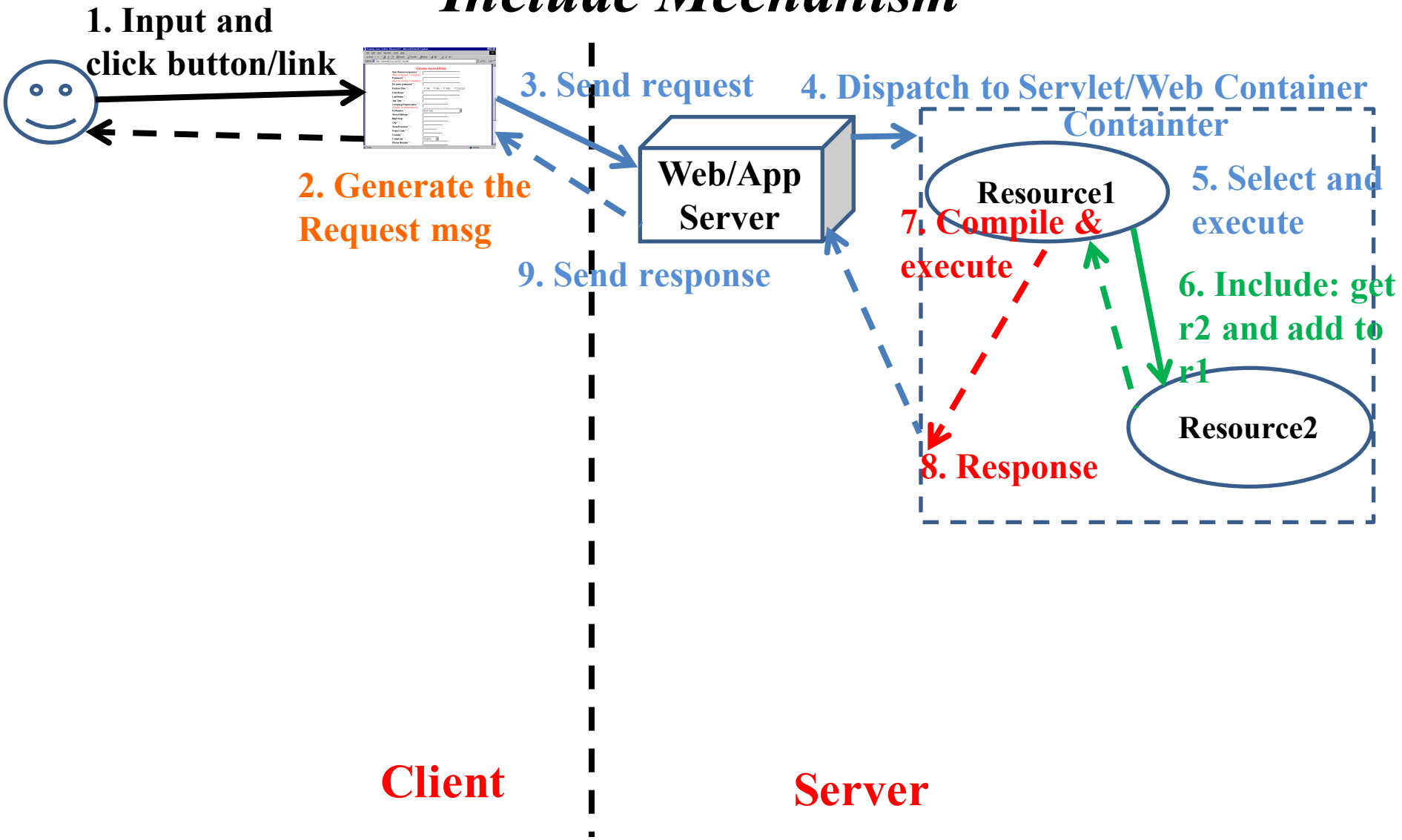



```
request.setAttribute("Middle", "Middle Information");

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

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

The Web Container Model

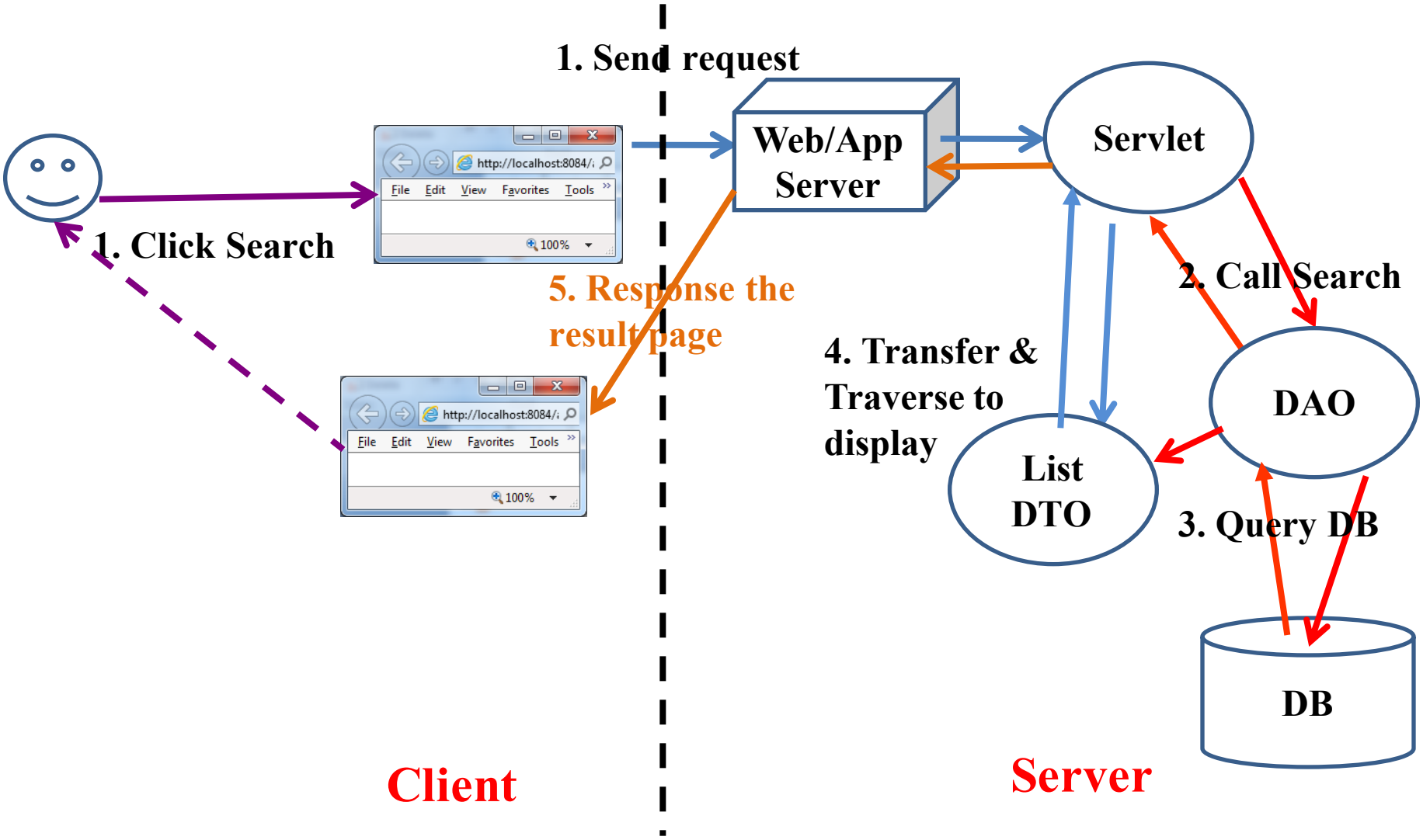
Need for using RequestDispatcher

Include Mechanism



How To Transfer

Interactive Server Model – Implementation

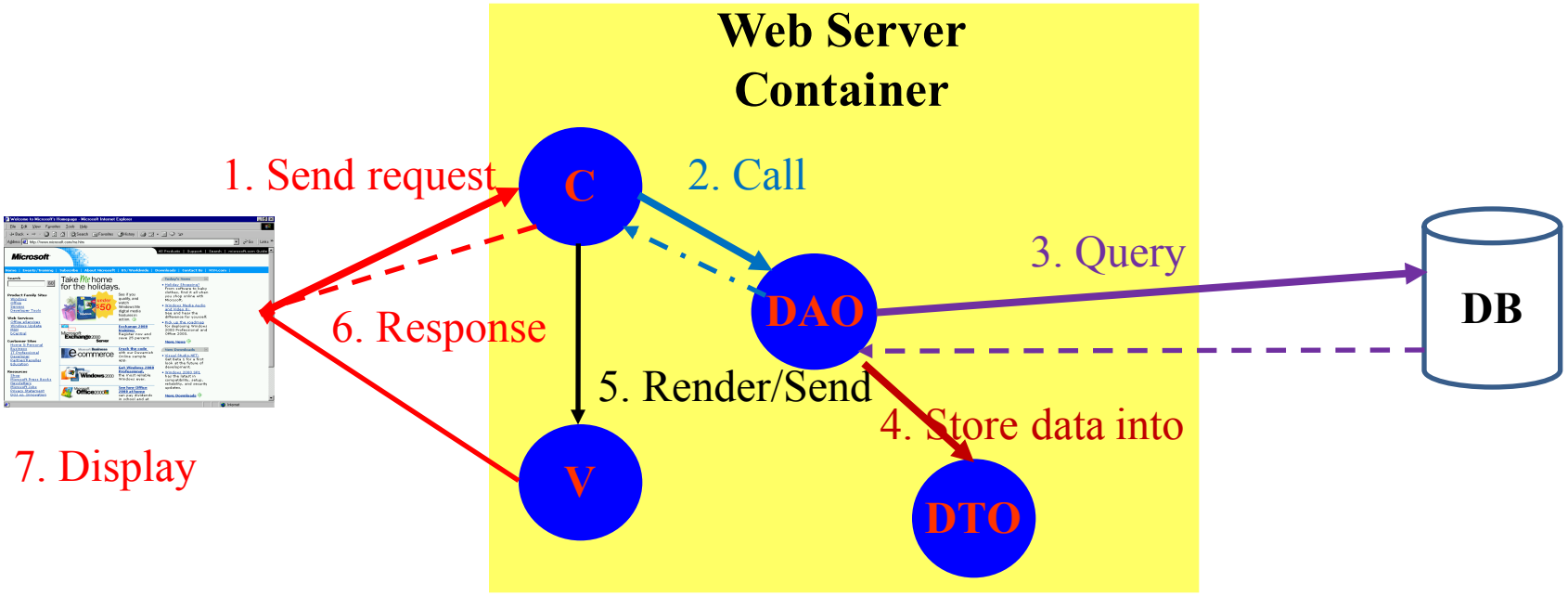


Summary

- **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

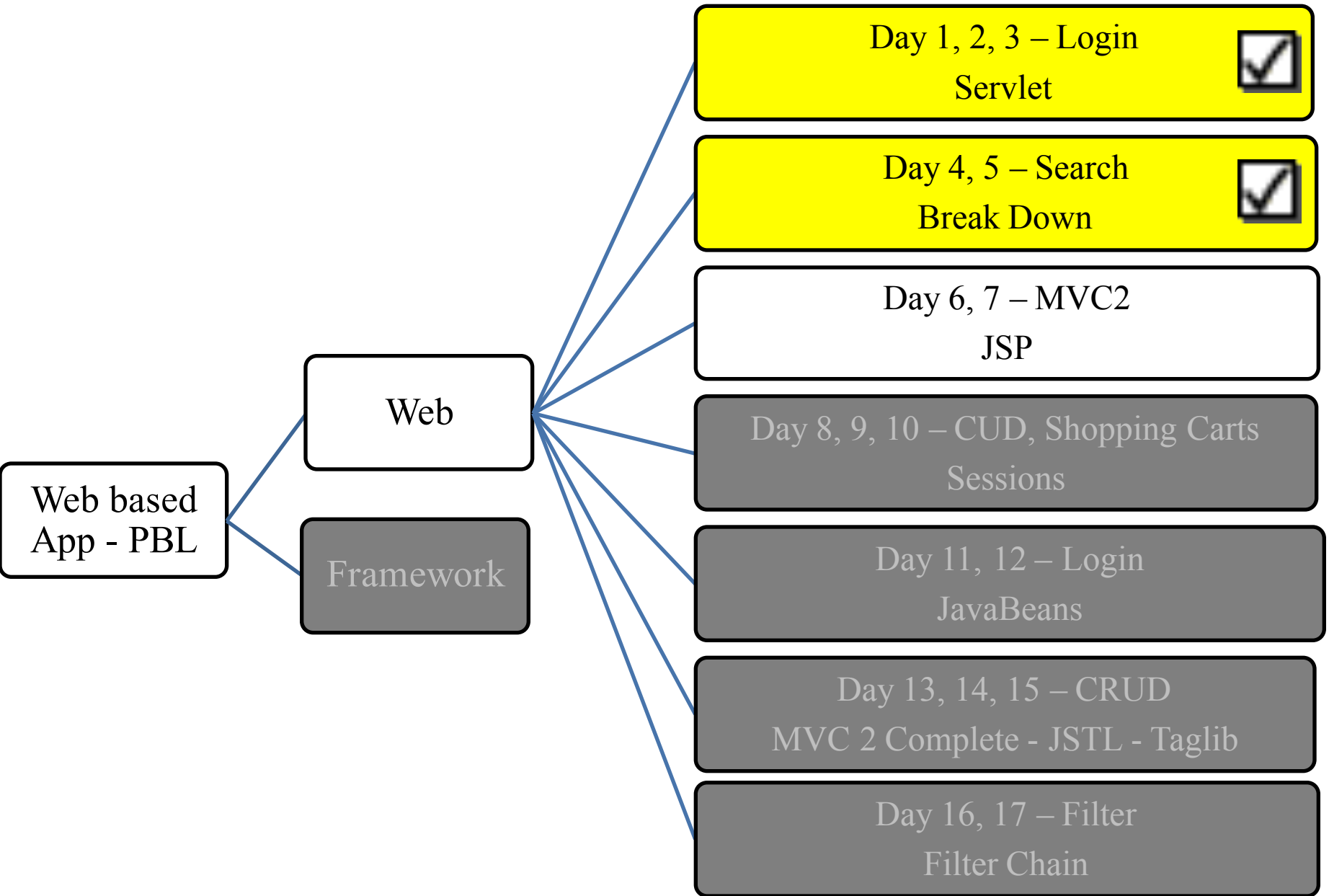

Web Server
Container
1. Send request
C
2. Call
3. Query
DB
DAO
6. Response
5. Render/Send
4. Store data into
7. Display
V
DTO

Fpt University

Next Lecture

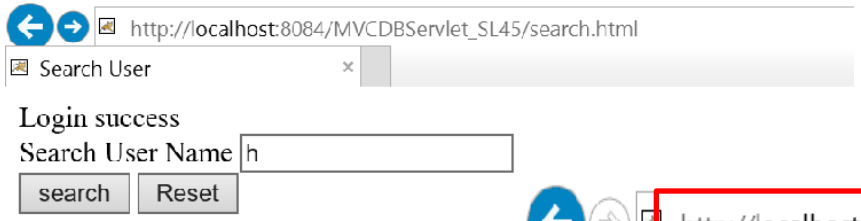
- **How to upgrade Application in previous topics approach MVC Model**
  - **Using JSP to View**
  - **MVC Pattern Design**

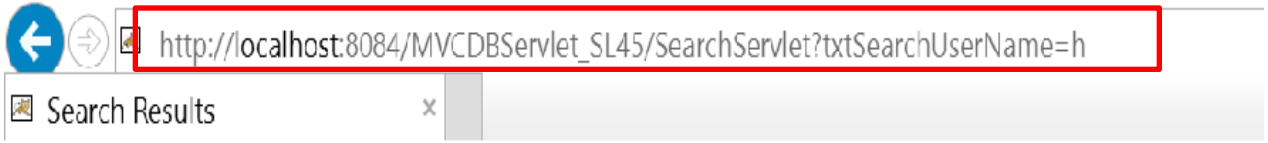
Next Lecture



Appendix :How To Transfer

Expectation

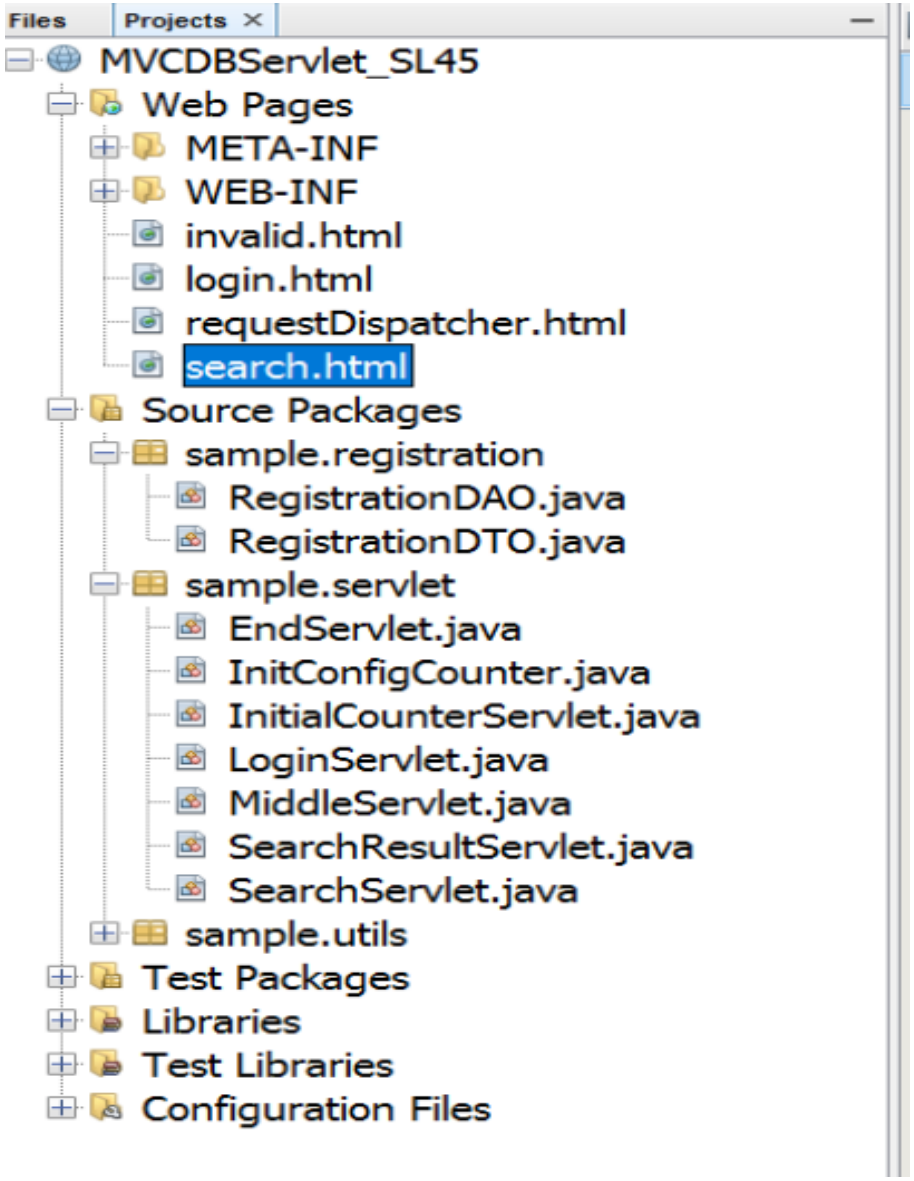




Search value: h

No.	UserID	Name	Password	Role
1	hoadnt	Hòa	123	AD
2	SE123	Hòa Đoàn	123	AD
3	Se124	Hồng Kim	123	ST

Appendix :How To Transfer Expectation



Login.html

login.html

Source | History

```
1  <!DOCTYPE html>
2  <!--
3  To change this license header, choose License Headers in Project Properties.
4  To change this template file, choose Tools | Templates
5  and open the template in the editor.
6  -->
7  <html>
8      <head>
9          <title>Login page</title>
10         <meta charset="UTF-8">
11         <meta name="viewport" content="width=device-width, initial-scale=1.0">
12     </head>
13     <body>
14         <form action="LoginServlet" method="post">
15             User ID <input type="text" name="txtUserID"/> <br/>
16             Password <input type="password" name="txtPassword"/><br/>
17             <input type="submit" value="Login" name="btnAction"/>
18             <input type="reset" value="Reset"/>
19
20         </form>
21     </body>
22 </html>
23
```

Fpt University

Search.html

search.html

Source | History

```
1  <!DOCTYPE html>
2  <!--
3  To change this license header, choose License Headers in Project Properties.
4  To change this template file, choose Tools | Templates
5  and open the template in the editor.
6  -->
7  <html>
8      <head>
9          <title>Search User</title>
10         <meta charset="UTF-8">
11         <meta name="viewport" content="width=device-width, initial-scale=1.0">
12     </head>
13     <body>
14         <div>Login success  </div>
15         <form action="SearchServlet" >
16             Search User Name <input type="text" name="txtSearchUserName"/> <br/>
17
18             <input type="submit" value="search" />
19             <input type="reset" value="Reset"/>
20
21         </form>
22     </body>
23 </html>
24
```

Fpt University

DTO

RegistrationDTO.java

Source | History

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package sample.registration;

import java.io.Serializable;

/**
 *
 * @author HD
 */
public class RegistrationDTO implements Serializable{
    private String userID;
    private String userName;
    private String password;
    private String role;

    public RegistrationDTO(String userID, String userName, String password, String role) {
        this.userID = userID;
        this.userName = userName;
        this.password = password;
        this.role = role;
    }

    public String getUserID() {...3 lines }

    public void setUserID(String userID) {...3 lines }

    public String getUserName() {...3 lines }

    public void setUserName(String userName) {...3 lines }

    public String getPassword() {...3 lines }

    public void setPassword(String password) {...3 lines }

    public String getRole() {...3 lines }

    public void setRole(String role) {...3 lines }


}

```

Fpt University

DAO

RegistrationDAO.java

Source | History

```
16  *
17  * @author Hoa Doan
18  */
19  public class RegistrationDAO {
20      public boolean checkLogin(String userID, String password) throws SQLException {...37 lines }
57
58      public List<RegistrationDTO> getListUsers(String searchValue) throws SQLException, ClassNotFoundException{
59          Connection conn=null ;
60          PreparedStatement stm=null;
61          ResultSet rs=null;
62          List<RegistrationDTO> listUsers= null;
63          try{
64             conn=DBUtils.createConnection();
65             String sql="select userID, userName,password,role from tblUsers where username like ?";
66             stm=conn.prepareStatement(sql);
67             stm.setString(1, "%"+searchValue+"%");
68             rs= stm.executeQuery();
69             while(rs.next()){
70                 String userID= rs.getString("userID");
71                 String userName=rs.getString("userName");
72                 String password=rs.getString("password");
73                 String role=rs.getString("role");
74                 if(listUsers==null)
                       listUsers= new ArrayList<RegistrationDTO>();
76                 listUsers.add(new RegistrationDTO(userID, userName, password, role));
77             }
78          }finally{
79              if(rs!=null)
80                  rs.close();
81              if(stm!=null)
82                  stm.close();
83              if(conn!=null)
84                  conn.close();
85          }
86          return listUsers;
87      }
88
89
```

Login Servlet

LoginServlet.java

Source | History

```
16  import javax.servlet.http.HttpServletResponse;
17  import sample.registration.RegistrationDAO;
18
19  /**
20   *
21   * @author HD
22   */
23  public class LoginServlet extends HttpServlet {
24      private final String searchPage = "search.html";
25      private final String invalidPage = "invalid.html";
26
27      /**
28       * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
29       * methods.
30       *
31       * @param request servlet request
32       * @param response servlet response
33       * @throws ServletException if a servlet-specific error occurs
34       * @throws IOException if an I/O error occurs
35       */
36      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException, SQLException {
38          response.setContentType("text/html;charset=UTF-8");
39          try (PrintWriter out = response.getWriter()) {
40              /* TODO output your page here. You may use following sample code. */
41              String button = request.getParameter("btnAction");
42              String url = invalidPage;
43              if (button.equals("Login")) {
44                  String userID = request.getParameter("txtUserID");
45                  String password = request.getParameter("txtPassword");
46                  RegistrationDAO registrationDAO = new RegistrationDAO();
47                  if (registrationDAO.checkLogin(userID, password)) {
48                      url = searchPage;
49                  }
50
51              }
52  //              sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer.
53              response.sendRedirect(url);
54          }
55      }
56
57      HttpServlet methods. Click on the + sign on the left to edit the code.
103
104 }
105
```

Search Servlet

SearchServlet.java

Source | History

```
22      *
23      * @author HD
24      */
25     public class SearchServlet extends HttpServlet {
26
27         /**
28          * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
29          * methods.
30          *
31          * @param request servlet request
32          * @param response servlet response
33          * @throws ServletException if a servlet-specific error occurs
34          * @throws IOException if an I/O error occurs
35          */
36         private final String searchpage = "search.html";
37         private final String showSearchResult = "SearchResultServlet";
38
39         protected void processRequest(HttpServletRequest request, HttpServletResponse response)
40                 throws ServletException, IOException, SQLException, ClassNotFoundException {
41             response.setContentType("text/html;charset=UTF-8");
42             String url = searchpage;
43             String searchValue = request.getParameter("txtSearchUserName");
44             try  {
45                 if (!searchValue.isEmpty()) {
46                     RegistrationDAO regDAO = new RegistrationDAO();
47                     List<RegistrationDTO> resultList = regDAO.getListUsers(searchValue);
48                     request.setAttribute("SEARCHRESULT", resultList);
49                     url = showSearchResult;
50                 }
51             } finally {
52                 try {
53                     RequestDispatcher rd = request.getRequestDispatcher(url);
54                     rd.forward(request, response);
                    } catch (Exception ex) {
                        ex.printStackTrace();
57                  }
58     //              rd.include(request, response);
59              }
60          }
61
62          HttpServlet methods. Click on the + sign on the left to edit the code.
112
113     }
114
```

Search Result Servlet

SearchResultServlet.java

Source | History

```
31          */
32      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
33              throws ServletException, IOException {
34          response.setContentType("text/html;charset=UTF-8");
35          try (PrintWriter out = response.getWriter()) {
36              /* TODO output your page here. You may use following sample code. */
37              out.println("<!DOCTYPE html>");
38              out.println("<html>");
39              out.println("<head>");
40              out.println("<title>Search Results</title>");
41              out.println("</head>");
42              out.println("<body>");
43
44              String searchValue= request.getParameter("txtSearchUserName");
45              out.println("<h1>Search value: "+searchValue+"</h1>");
46              List<RegistrationDTO> searchResult= (List<RegistrationDTO>) request.getAttribute("SEARCHRESULT");
47              if(searchResult!=null){
48                  out.println("<table border='1'>");
49                  out.println("<thead>");
50                  out.println("<tr>");
51                  out.println("<th>No.</th>");
52                  out.println("<th>UserID</th>");
53                  out.println("<th>Name</th>");
54                  out.println("<th>Password</th>");
55                  out.println("<th>Role</th>");
56                  out.println("</tr>");
57                  out.println("</thead>");
58                  out.println("<tbody>");
59                  int count=0;
60                  for(RegistrationDTO dto:searchResult){
61                      out.println("<tr>");
62                      out.println("<td>"+ ++count +"</td>");
63                      out.println("<td>"+ dto.getUserID() +"</td>");
64                      out.println("<td>"+ dto.getUserName()+"</td>");
65                      out.println("<td>"+ dto.getPassword()+"</td>");
66                      out.println("<td>"+ dto.getRole()+"</td>");
67                      out.println("</tr>");
68                  }
69                  out.println("</tbody>");
70                  out.println("</table>");
71              }else{
72                  out.println("<h1>No record is matched</h1>");
73              }
74              out.println("</body>");
75              out.println("</html>");
76          }
77      }
```

sample.servlet.SearchResultServlet > processRequest > try > if (searchResult != null) else >

Output | HTTP Server Monitor | Search Results