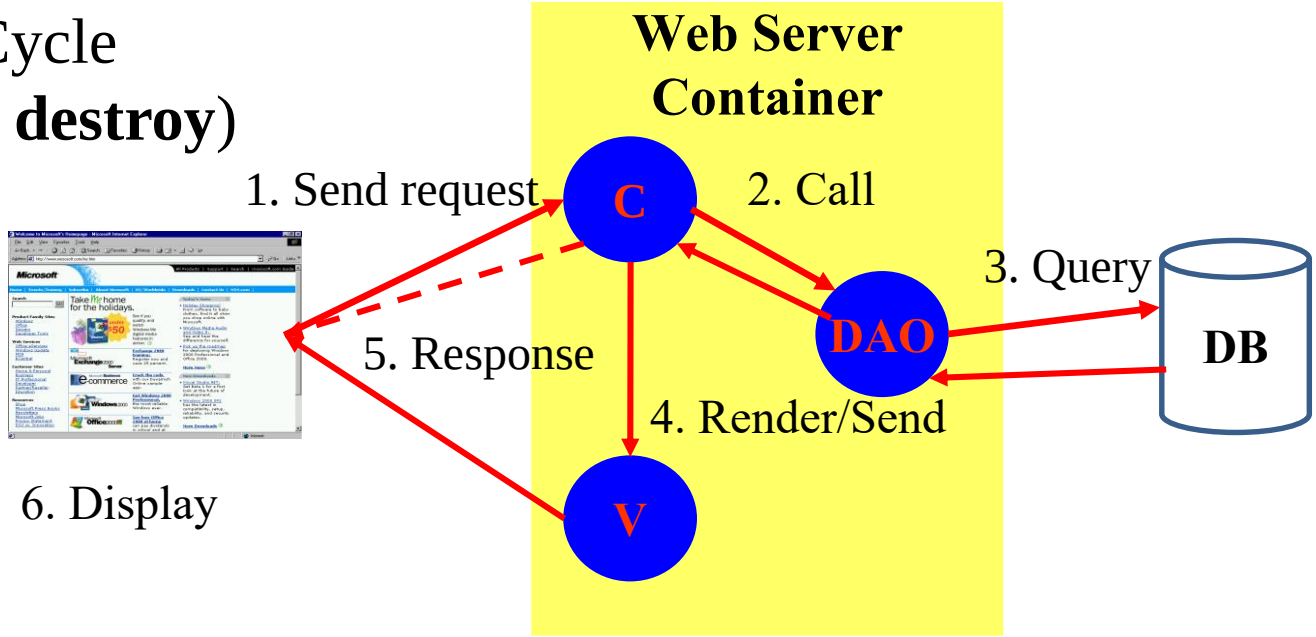
Fpt University

Web Applications & Web Containers

Web Applications
The Web Container Model

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**)

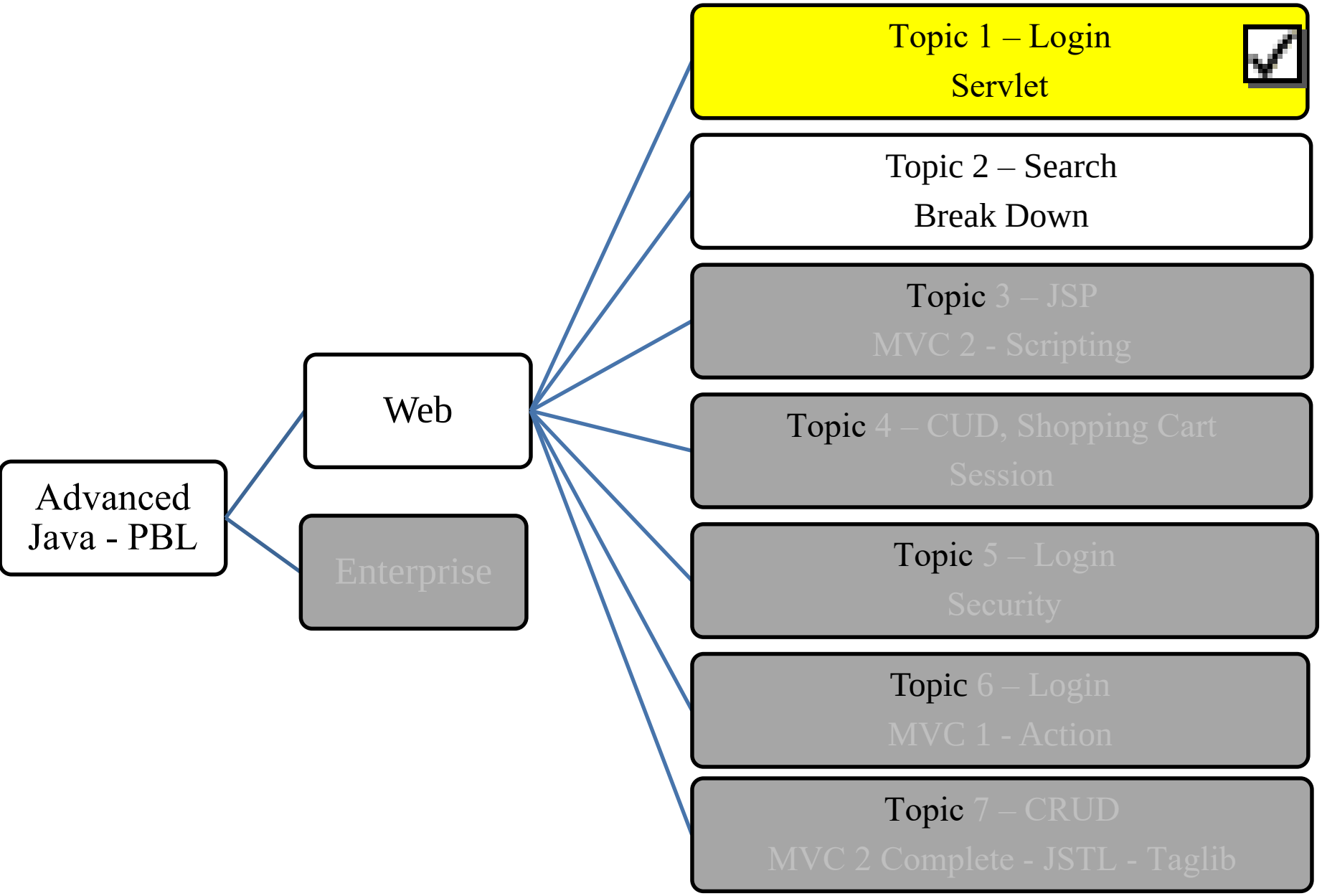


Fpt University

Objectives

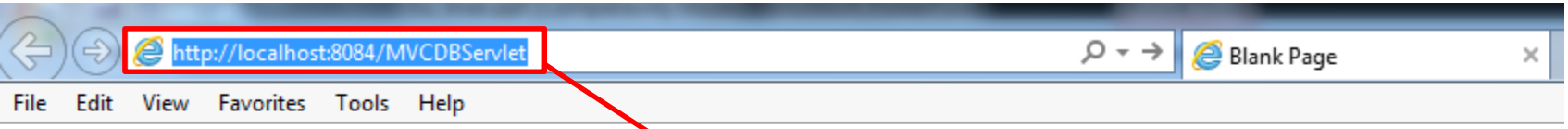
- **How to deploy the Web Application to Web Server without using Netbeans/ Eclipse tools?**
  - Web applications Structure
  - Request Parameters vs. Context Parameters vs. Config/Servlet Parameters
  - Application Segments vs. Scope
- **How to transfer from resources to others with/without data/objects?**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

Objectives

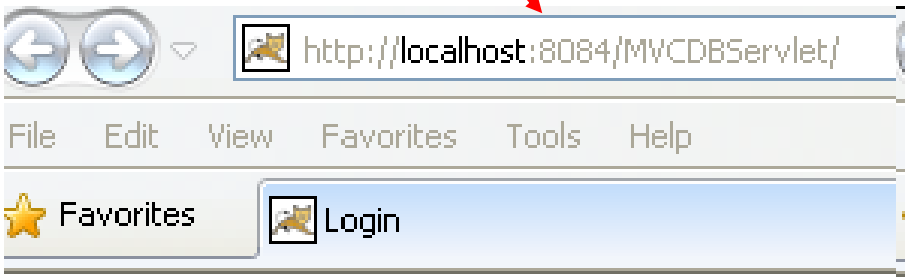

Advanced Java - PBL
Web
Enterprise
Topic 1 – Login Servlet
Topic 2 – Search Break Down
Topic 3 – JSP MVC 2 - Scripting
Topic 4 – CUD, Shopping Cart Session
Topic 5 – Login Security
Topic 6 – Login MVC 1 - Action
Topic 7 – CRUD MVC 2 Complete - JSTL - Taglib

Deploy Application

Expectation



Typed and Press Enter



Login Page

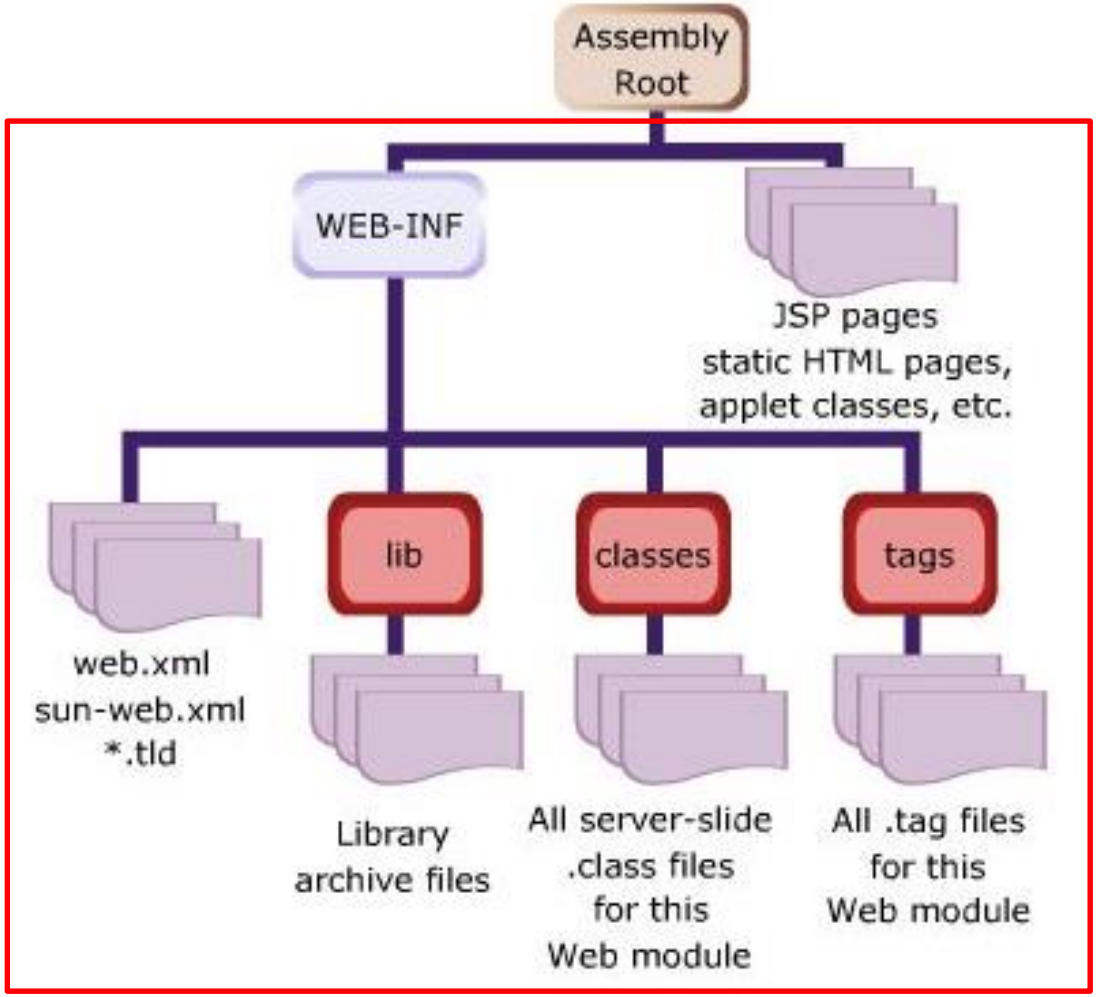
Username

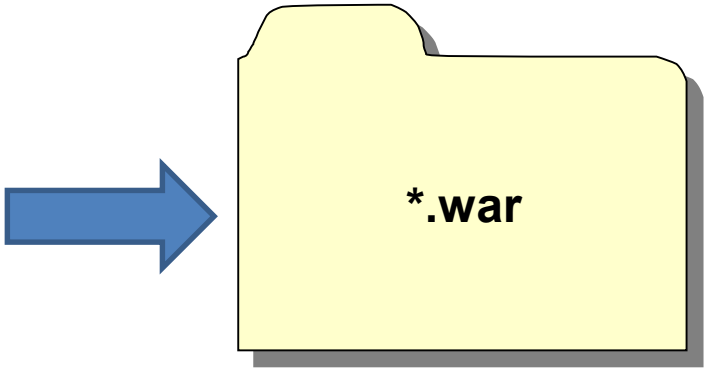
Password

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

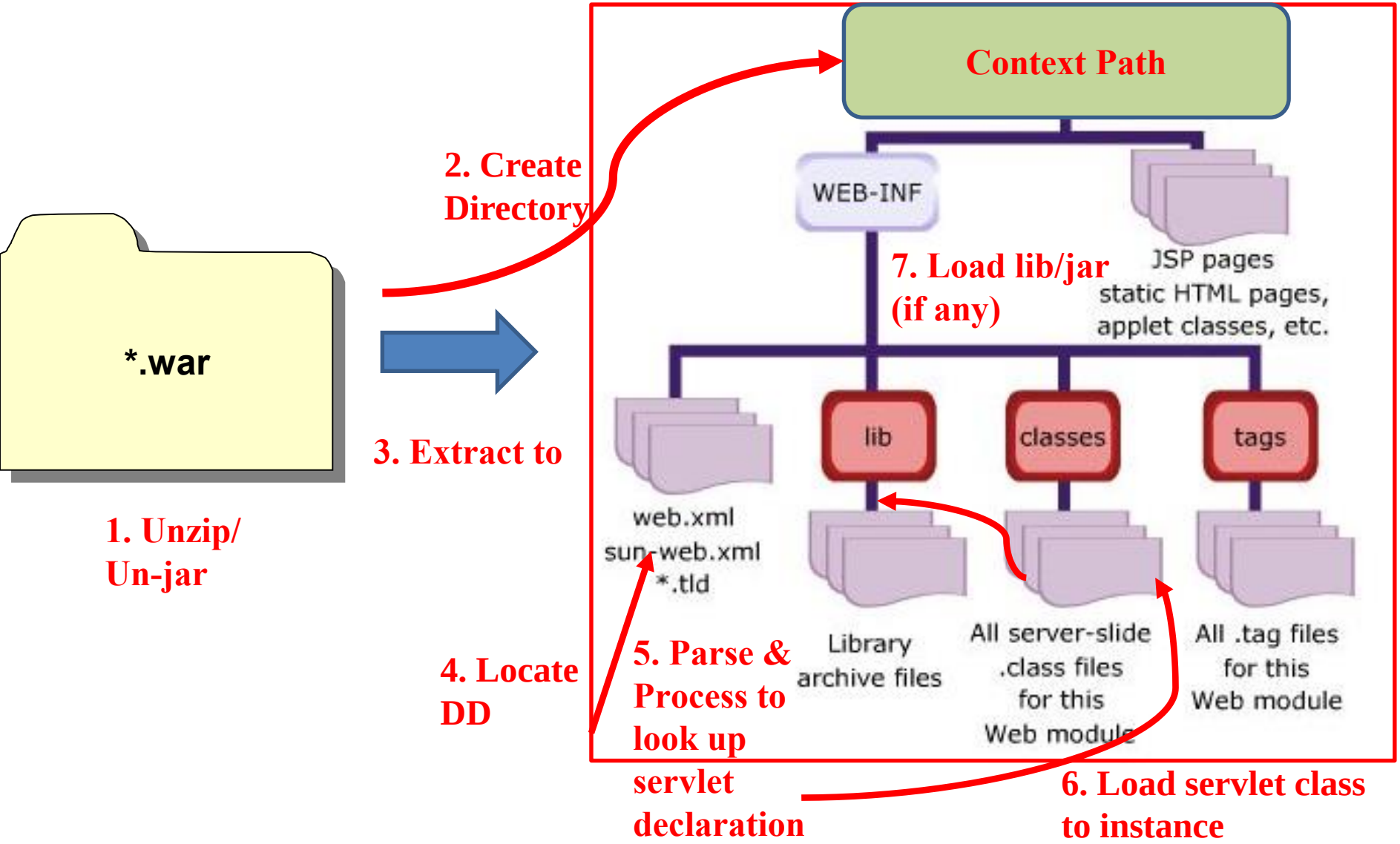
Web Applications

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.

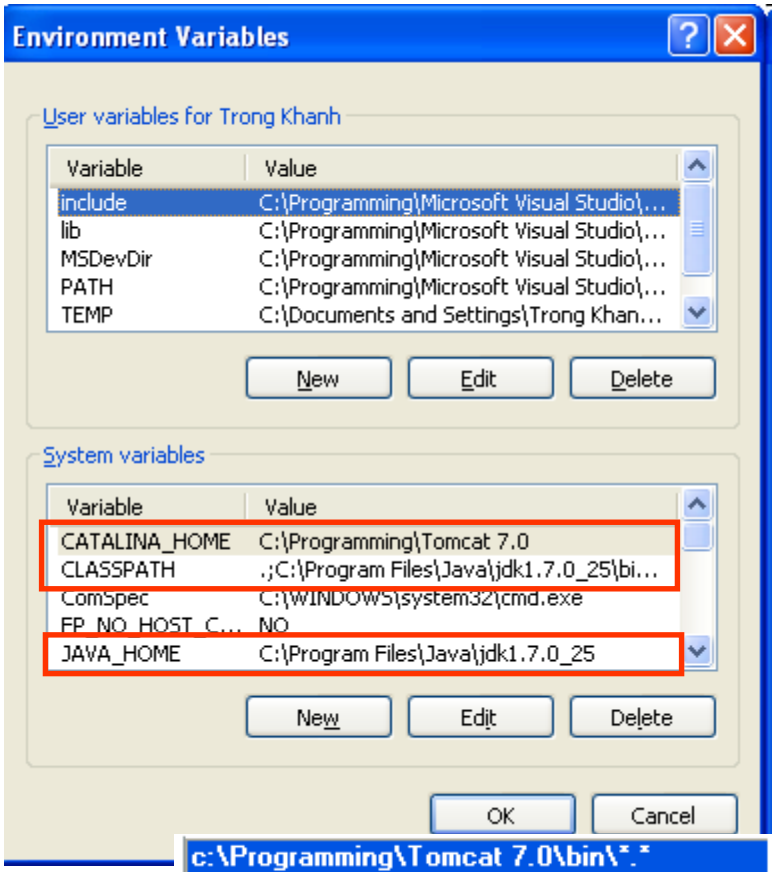
Web Applications

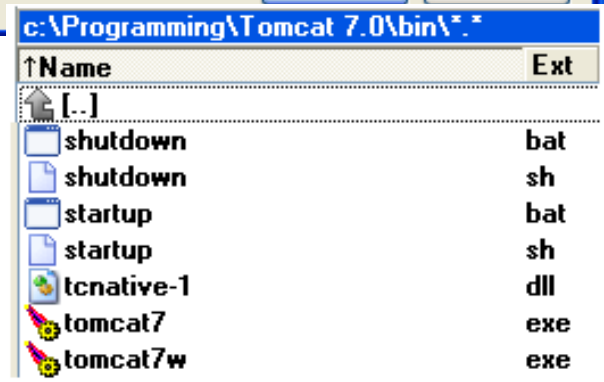
Deploy Mechanism



Web Applications
Manual Deploying

- Setup the environment for JAVA and TOMCAT
  - **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

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

Tomcat

```
INFO: Server startup in 619 ms
thg 9 21, 2013 8:29:03 CH org.apache.catalina.startup.HostConfig deployWAR
INFO: Deploying web application archive C:\Programming\Tomcat 7.0\webapps\AJDay1
_7.war
```

- 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

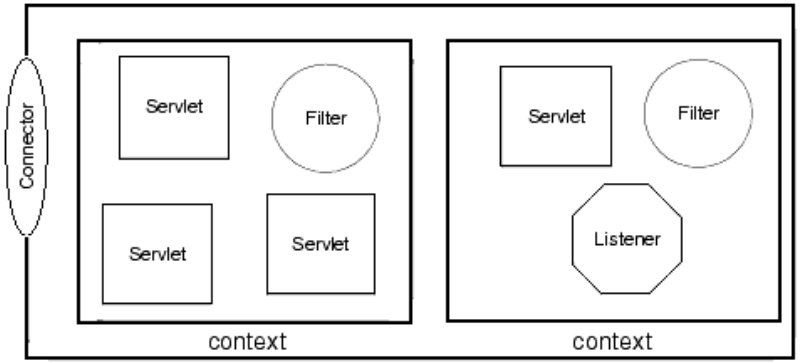
The Servlet Container

- 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

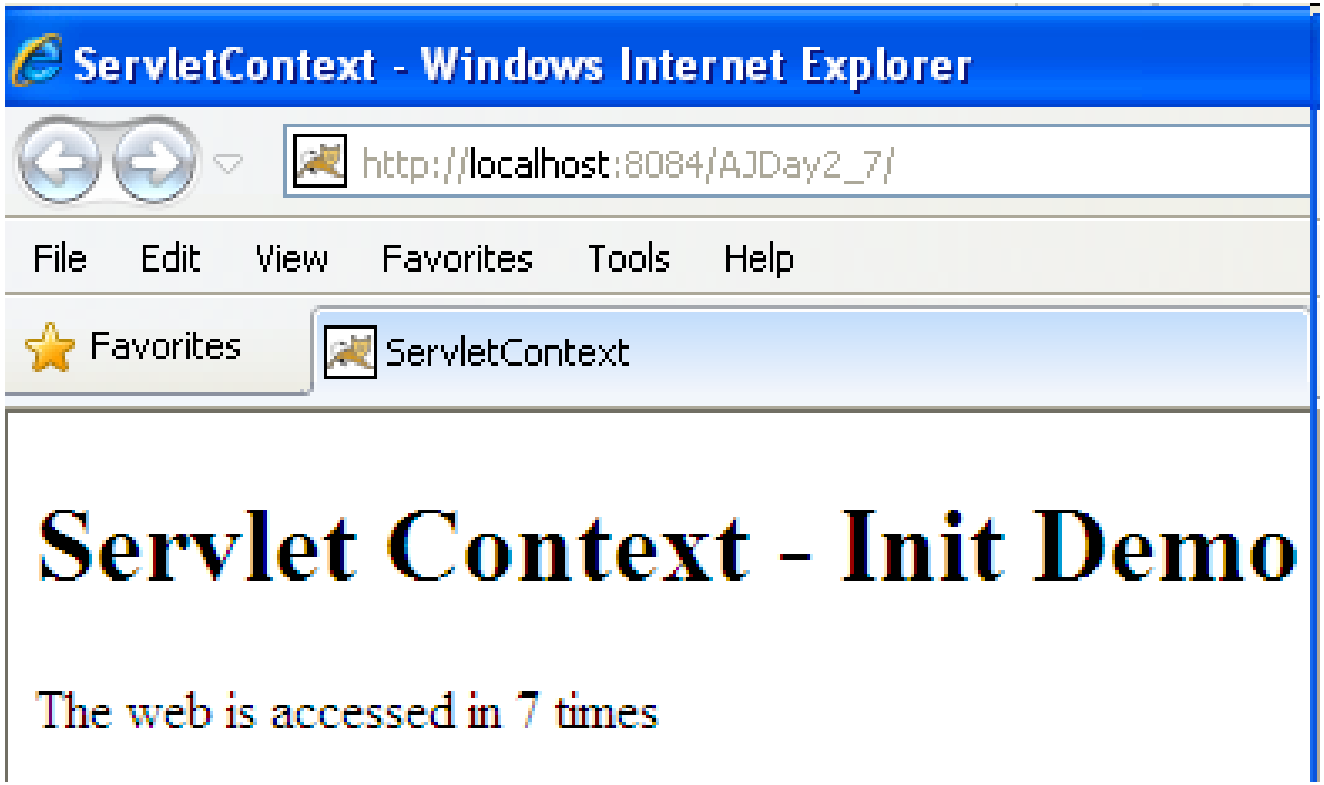
- Providing some fundamental information available to all the dynamic resources (servlets, JSP) within the web application is allowed by
  - **Using** servlet **initialization parameters in** the **deployment descriptor** with the **getInitParameter(String parName)** method to provide **initialization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

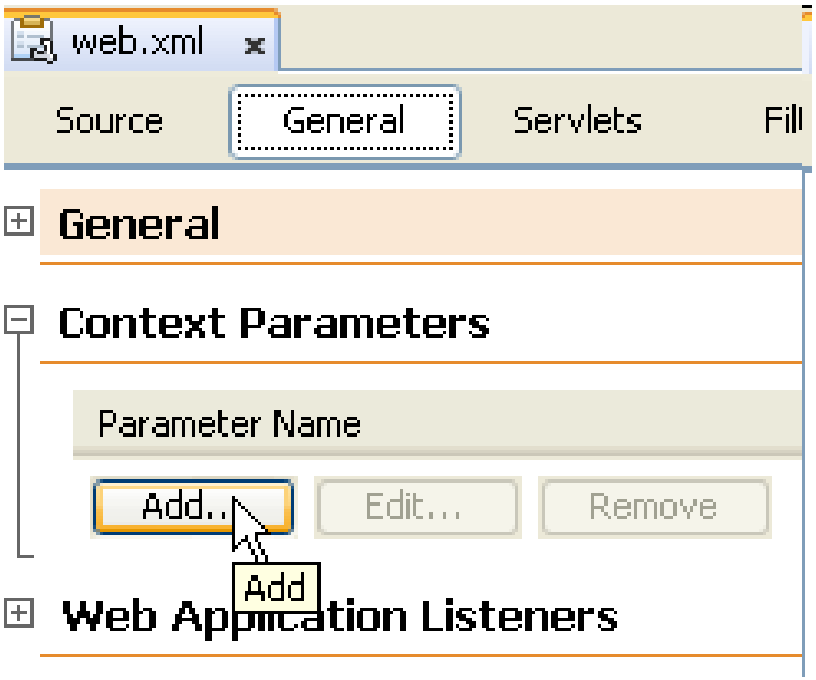


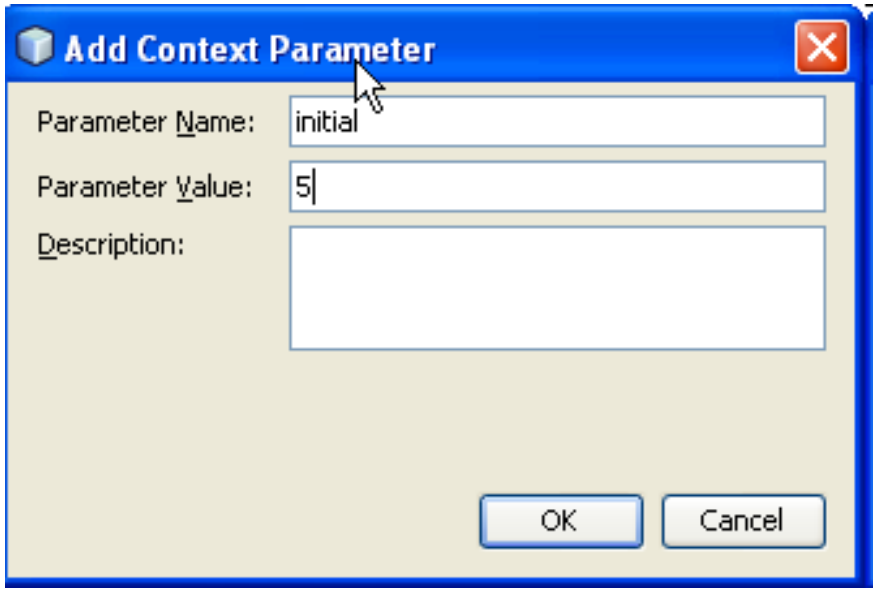
The Web Container Model

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

Source | General | Servlets | Filters | Pages | Re

General

Context Parameters

Parameter Name	Parameter Value
initial	5

Add... | Edit... | Remove

web.xml

Source | General | Servlets | Filters | Pages | Refe

```
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>
```

web.xml | InitCounter.java

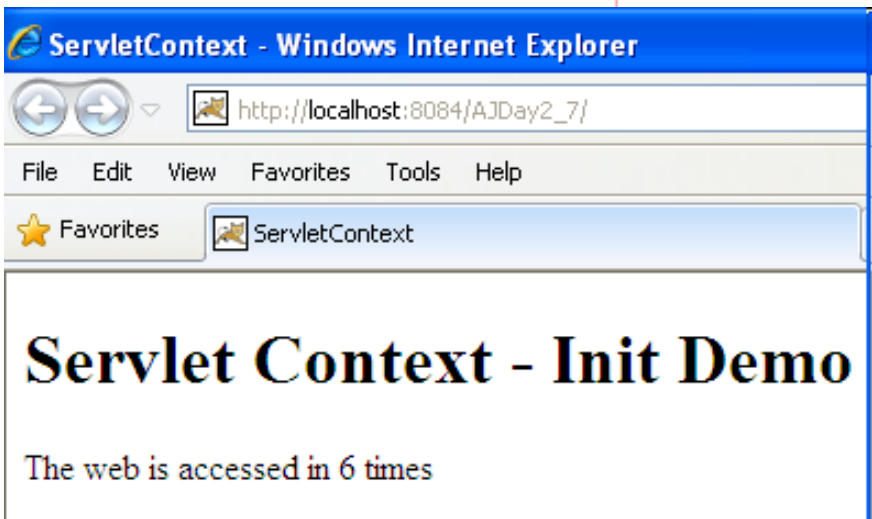
Source | History

```
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) {
                   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**

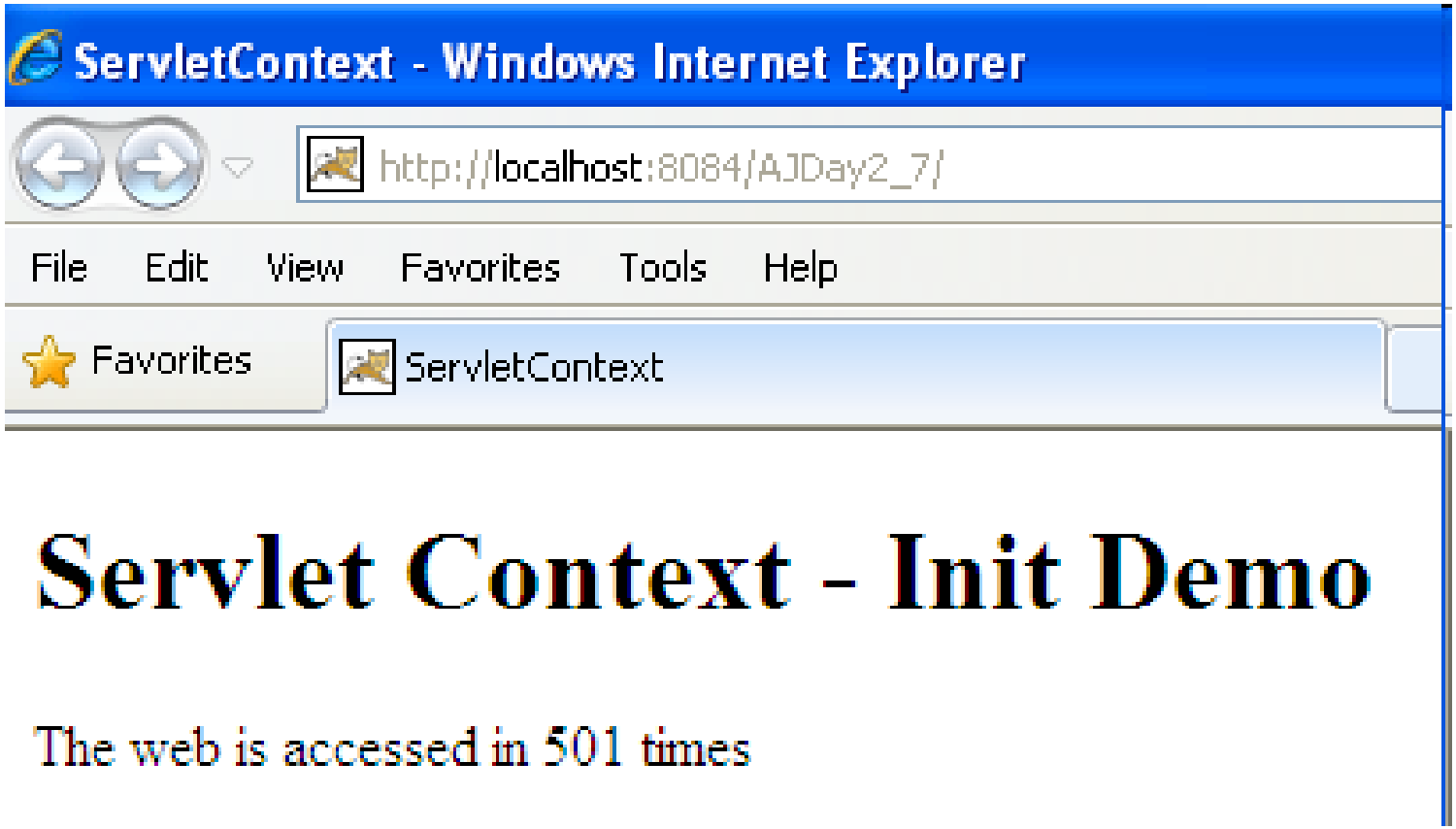
web.xml

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

web.xml | InitConfigCounter.java

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

Add... | Edit... | Remove

Add

Security Role References:

Add Initialization Parameter

Parameter Name: initial

Parameter Value: 5

Description: init parameter

OK | Cancel

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 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

Home - Windows Internet Explorer

http://localhost:8084/MVCDBServlet/ProcessServlet

File Edit View Favorites Tools Help

Favorites Home

Welcome to DB Servlet

Name a

Search

http://localhost:8084/MVCDBServlet/ProcessServlet?txtName=a&btAction=Search

File Edit View Favorites Tools Help

Favorites Search Result

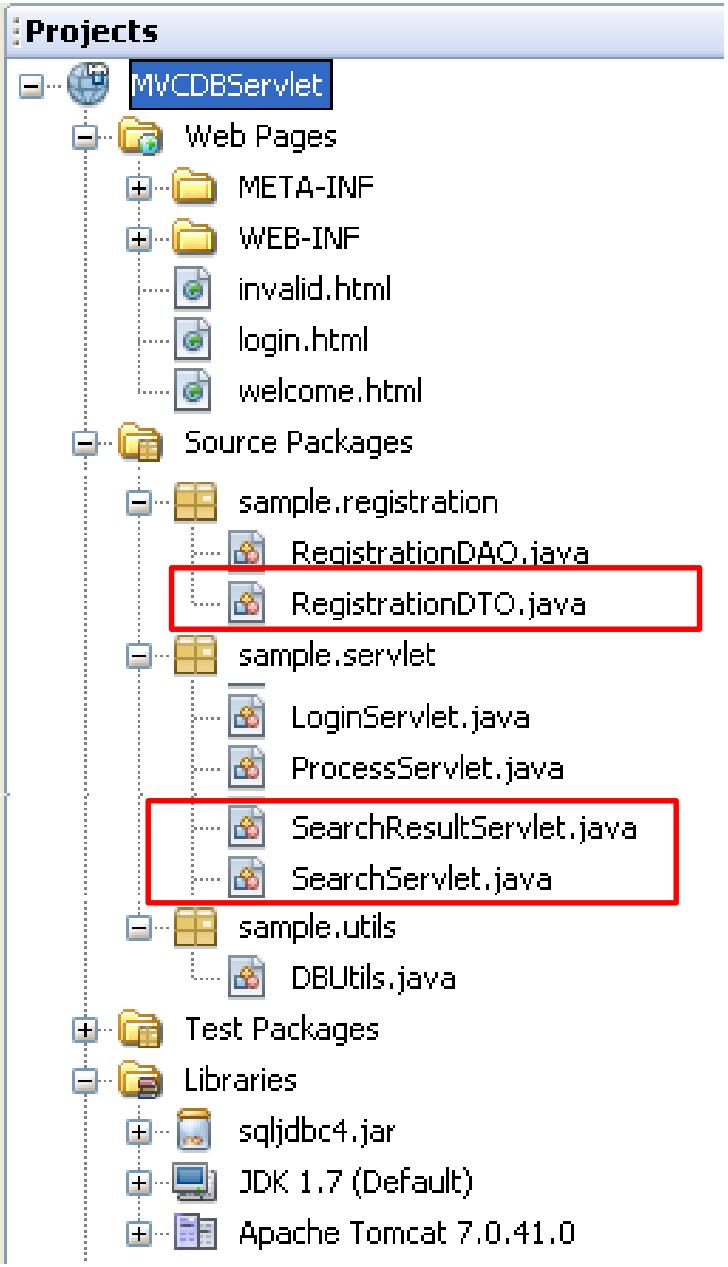
Search Result

No.	Username	Password	Lastname	Roles	Delete
1	class861	3455454	Khoa	false	Delete
2	EntityClass	EJB32	Day11	true	Delete
3	khanh	kieu123	Kieu Khanh	true	Delete
4	quangabc	123456	Quang di tre	false	Delete
5	Taidaica	123456	Tai Dai Dai Ca	false	Delete
6	TienDan	12345634	Dan Quan Dai	true	Delete

http://localhost:8084/MVCDBServlet/ProcessServlet?btAction=del&txtUsername=khanh&searchValue=a

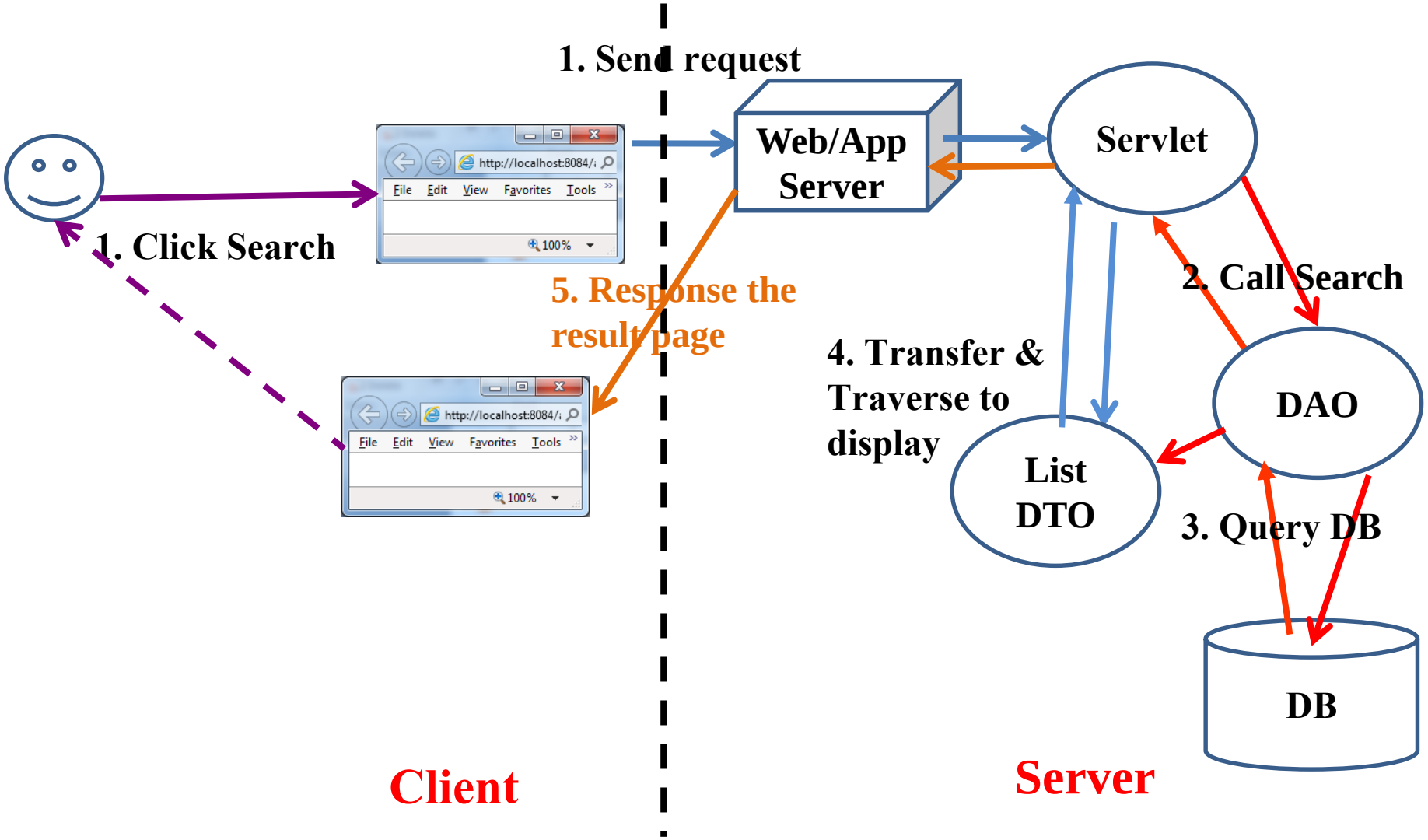
How To Transfer

Expectation



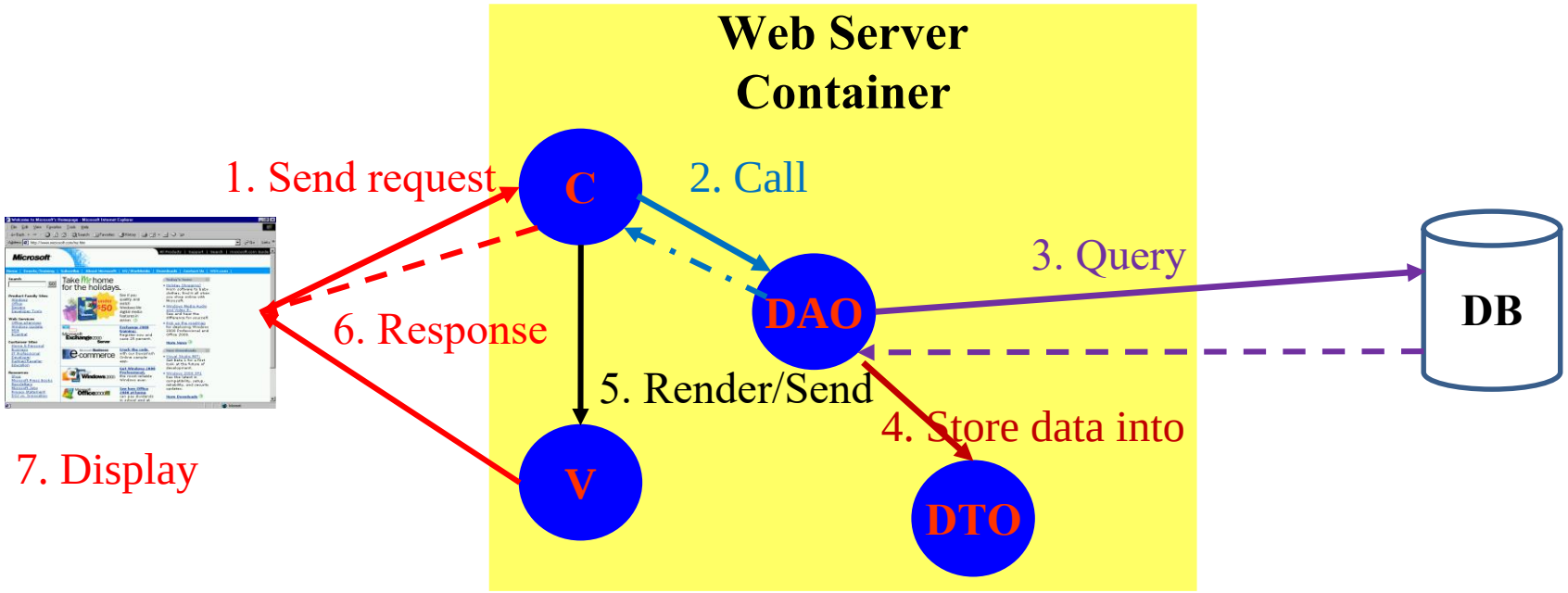
How To Transfer

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**: 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	- **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

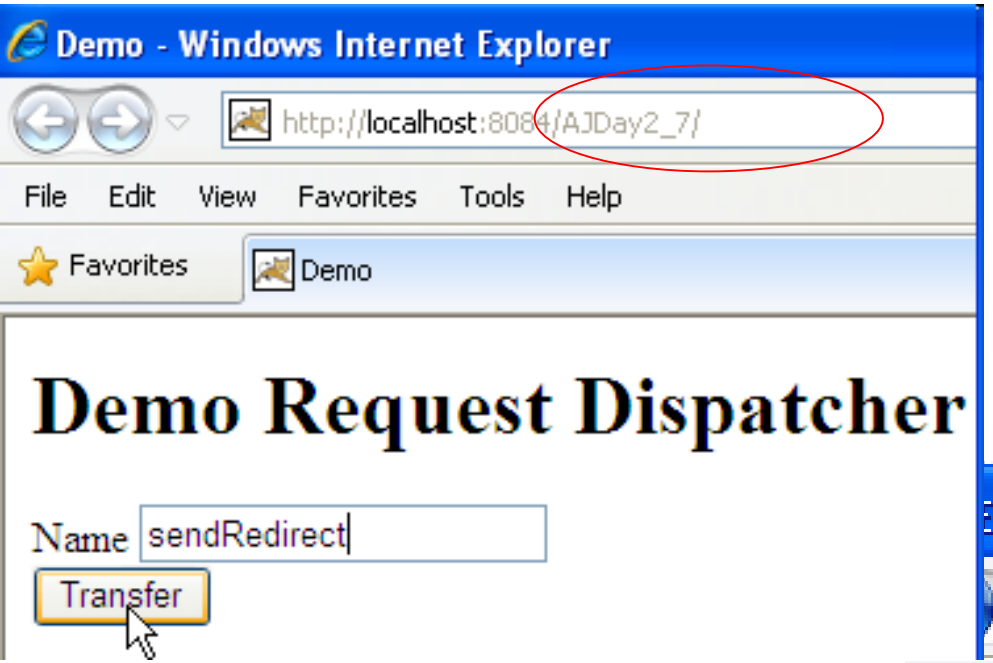
Need for using RequestDispatcher – Redirect

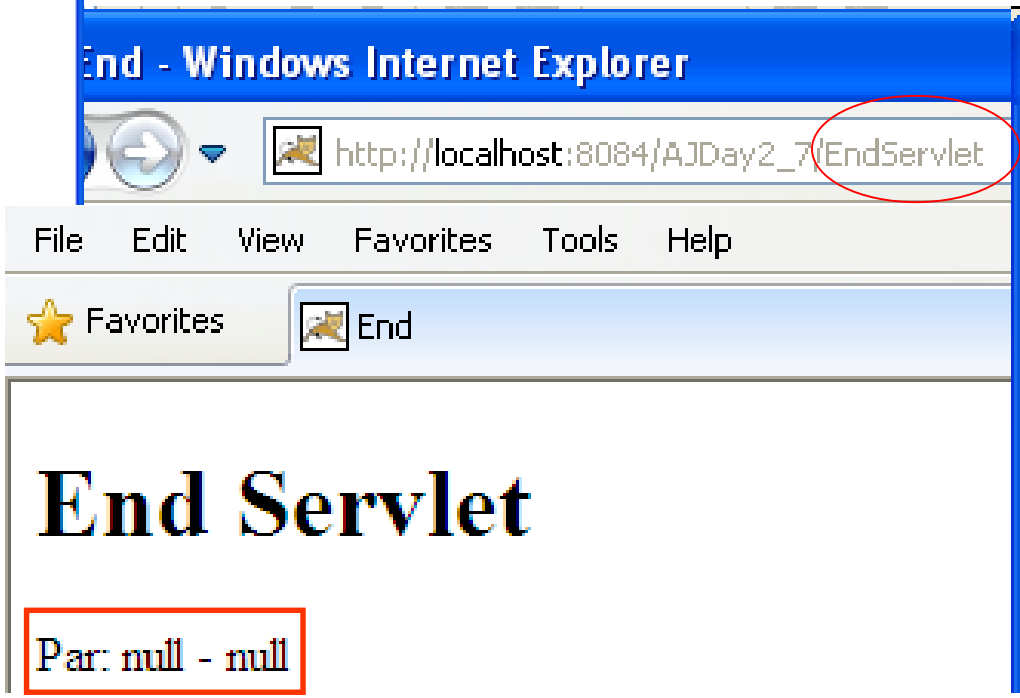
MiddleServlet.java | EndServlet.java

```
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

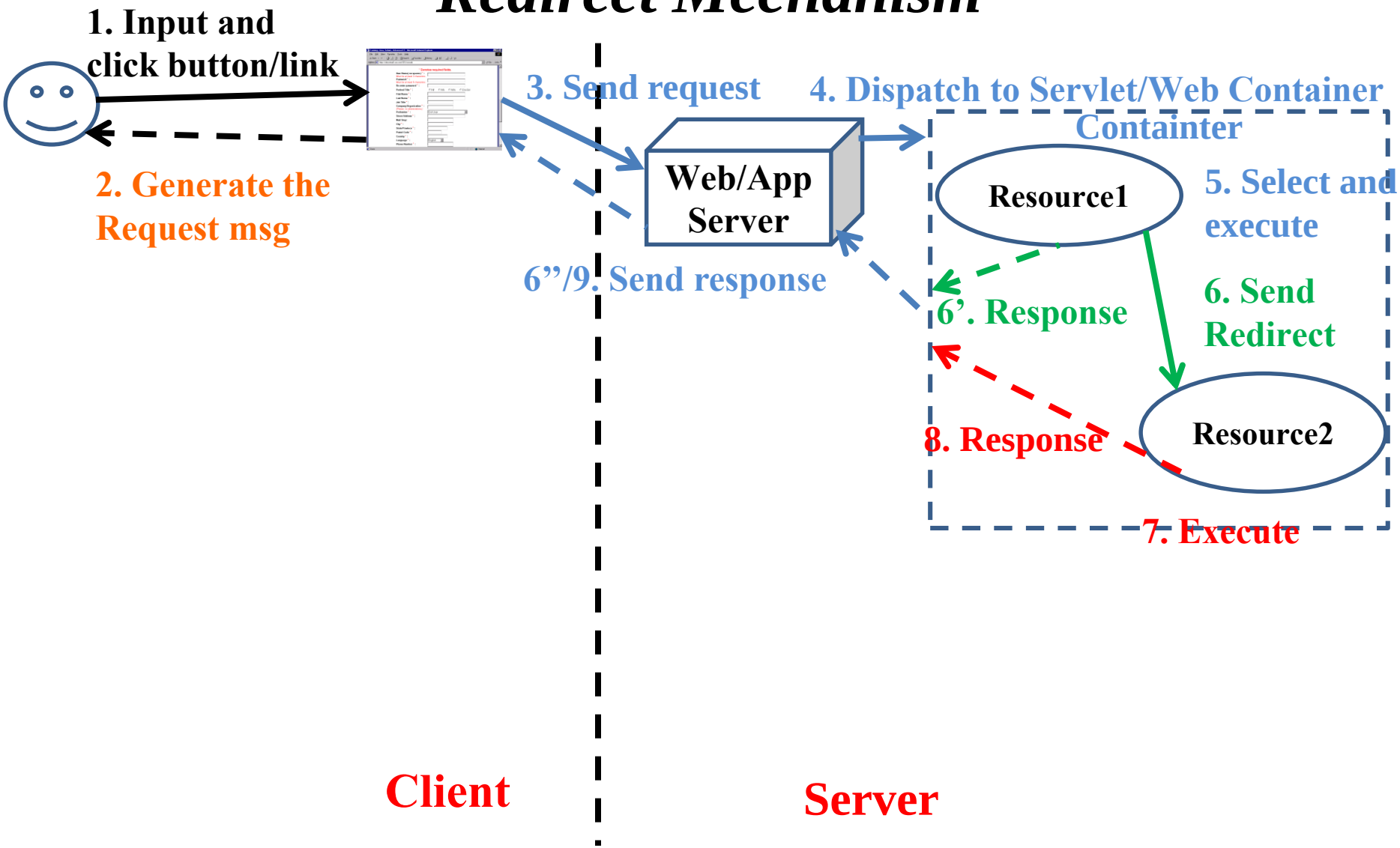




The Web Container Model

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

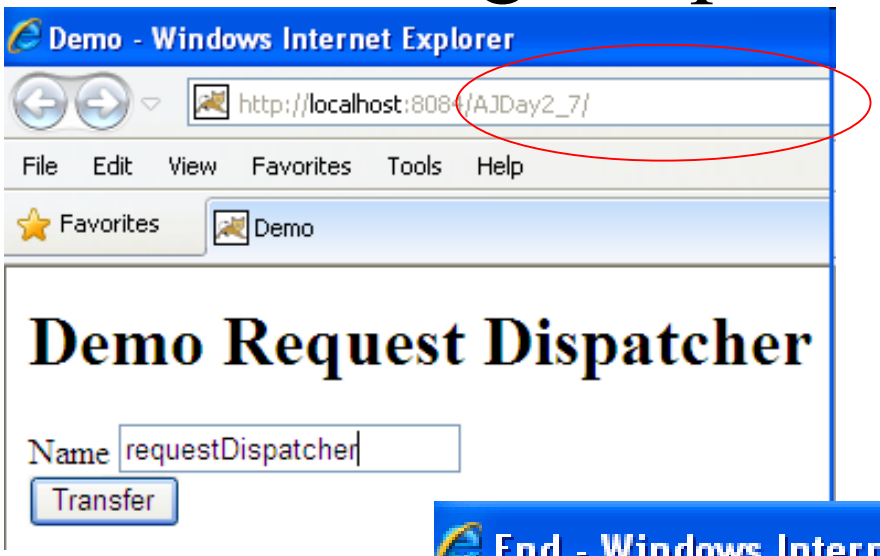
```
<body>
    <h1>Demo Request Dispatcher</h1>
    <form action="MiddleServlet">
        Name <input type="text" name="txtName" value="" /><br/>
        <input type="submit" value="Transfer" />
    </form>
</body>
```

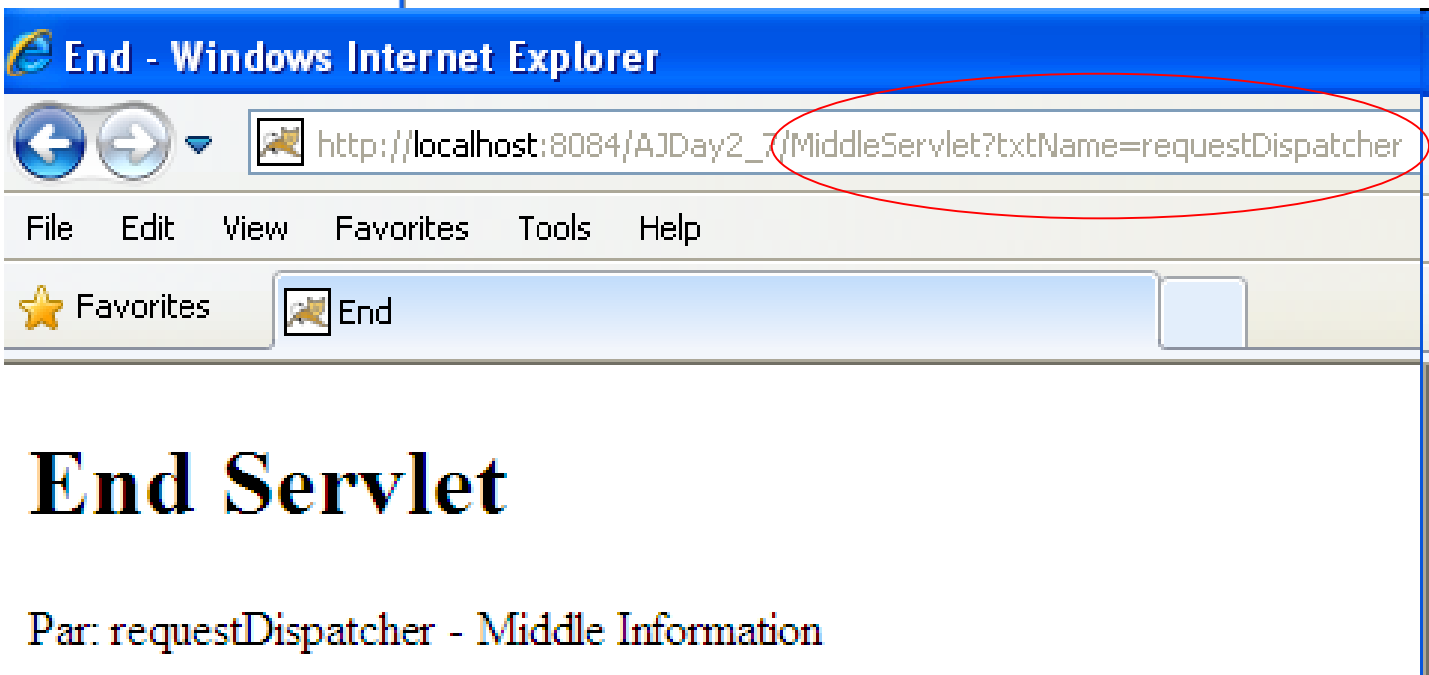
MiddleServlet.java | EndServlet.java

```
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

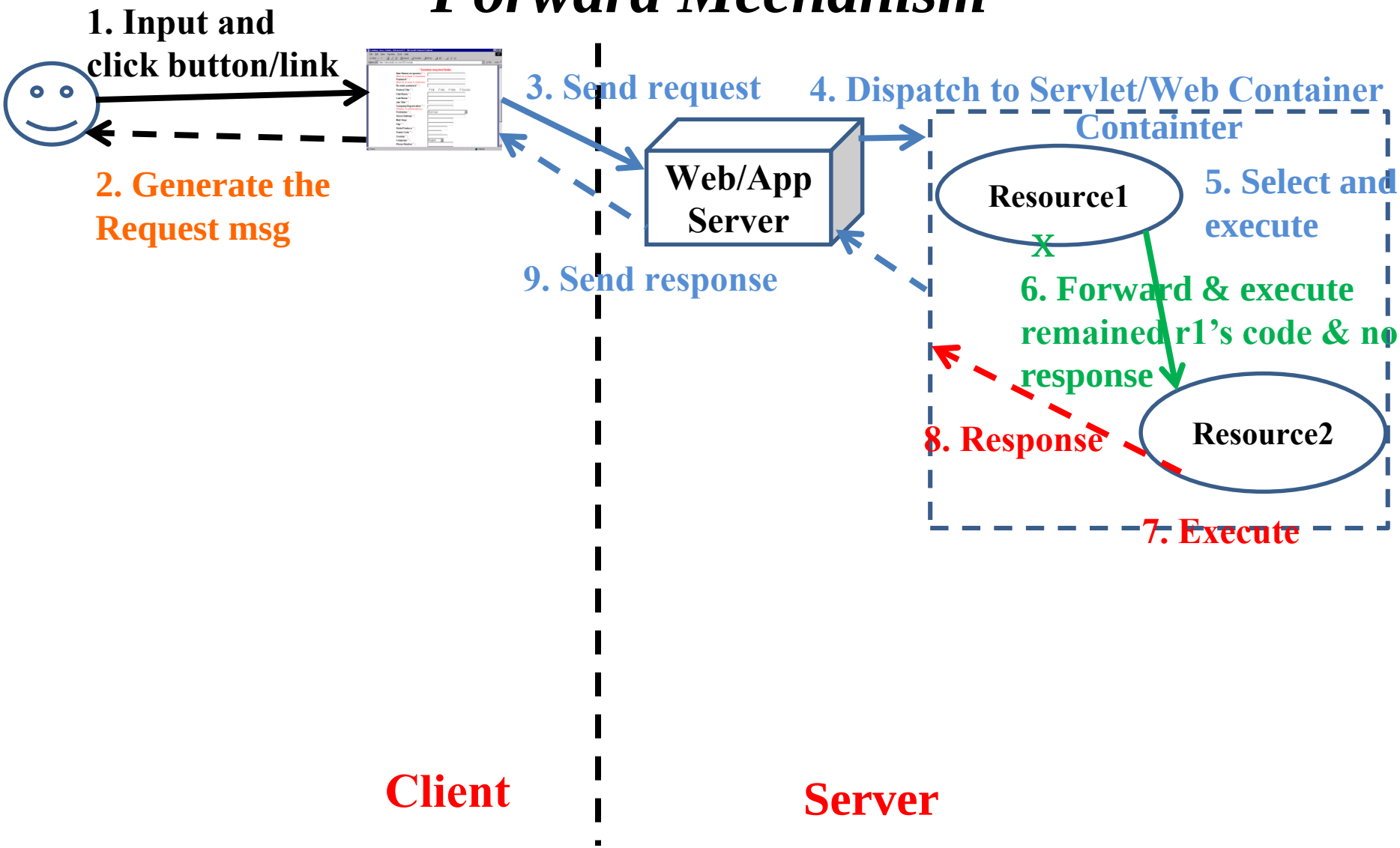




The Web Container Model

Need for using RequestDispatcher

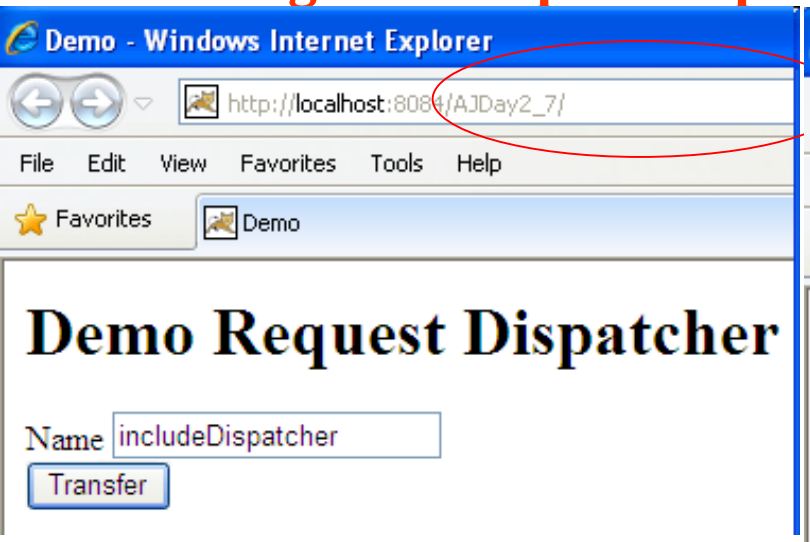
Forward Mechanism

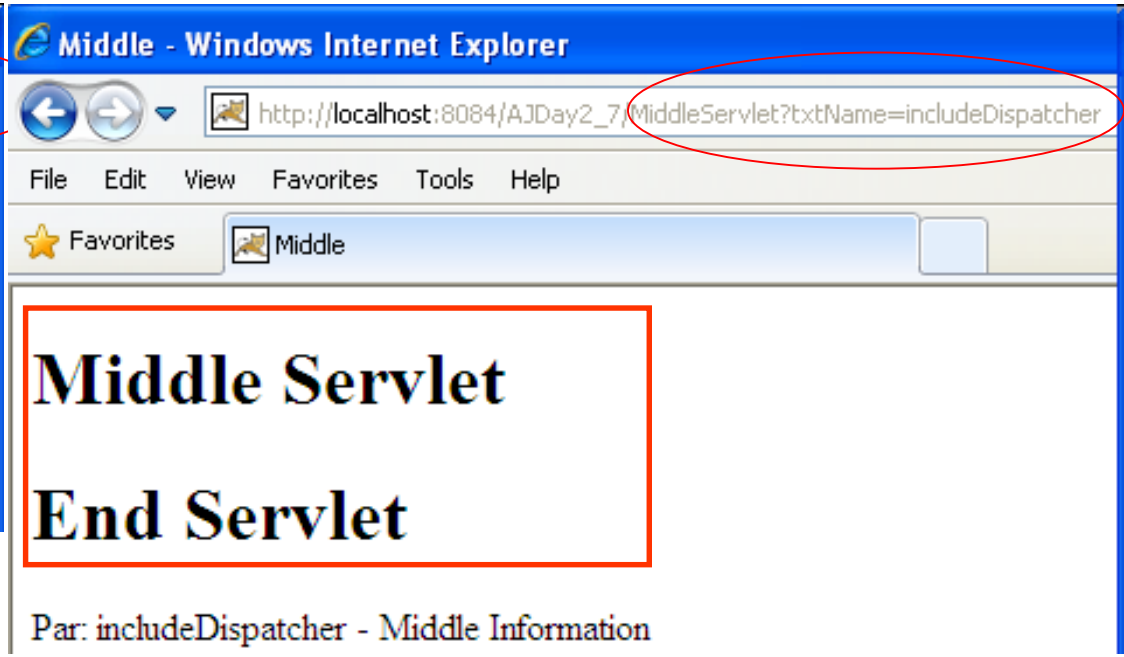


The Web Container Model

Using RequestDispatcher – Example

Change the RequestDispatch – forward method to include method



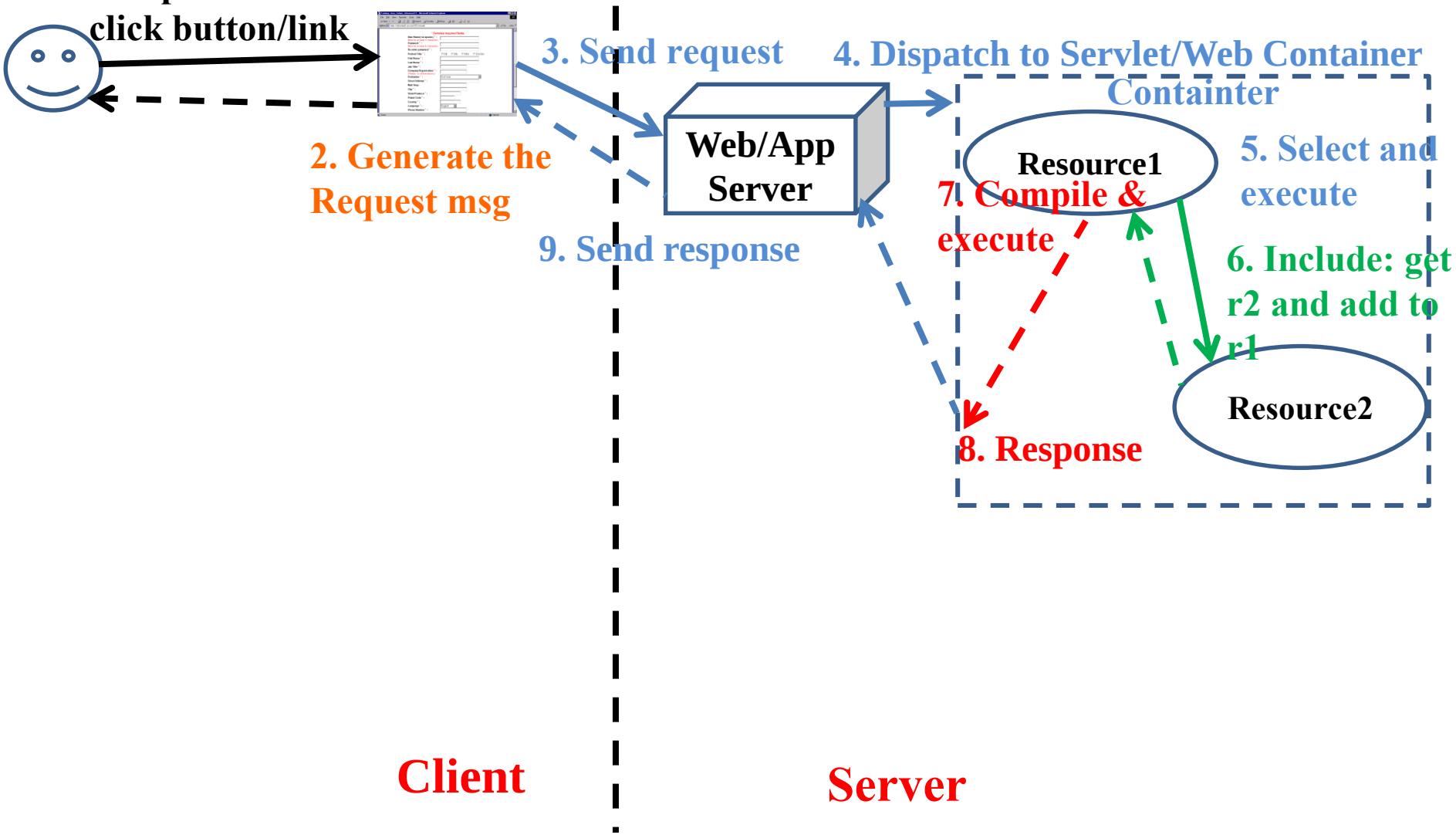



```
request.setAttribute("Middle", "Middle Information");

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

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

The Web Container Model

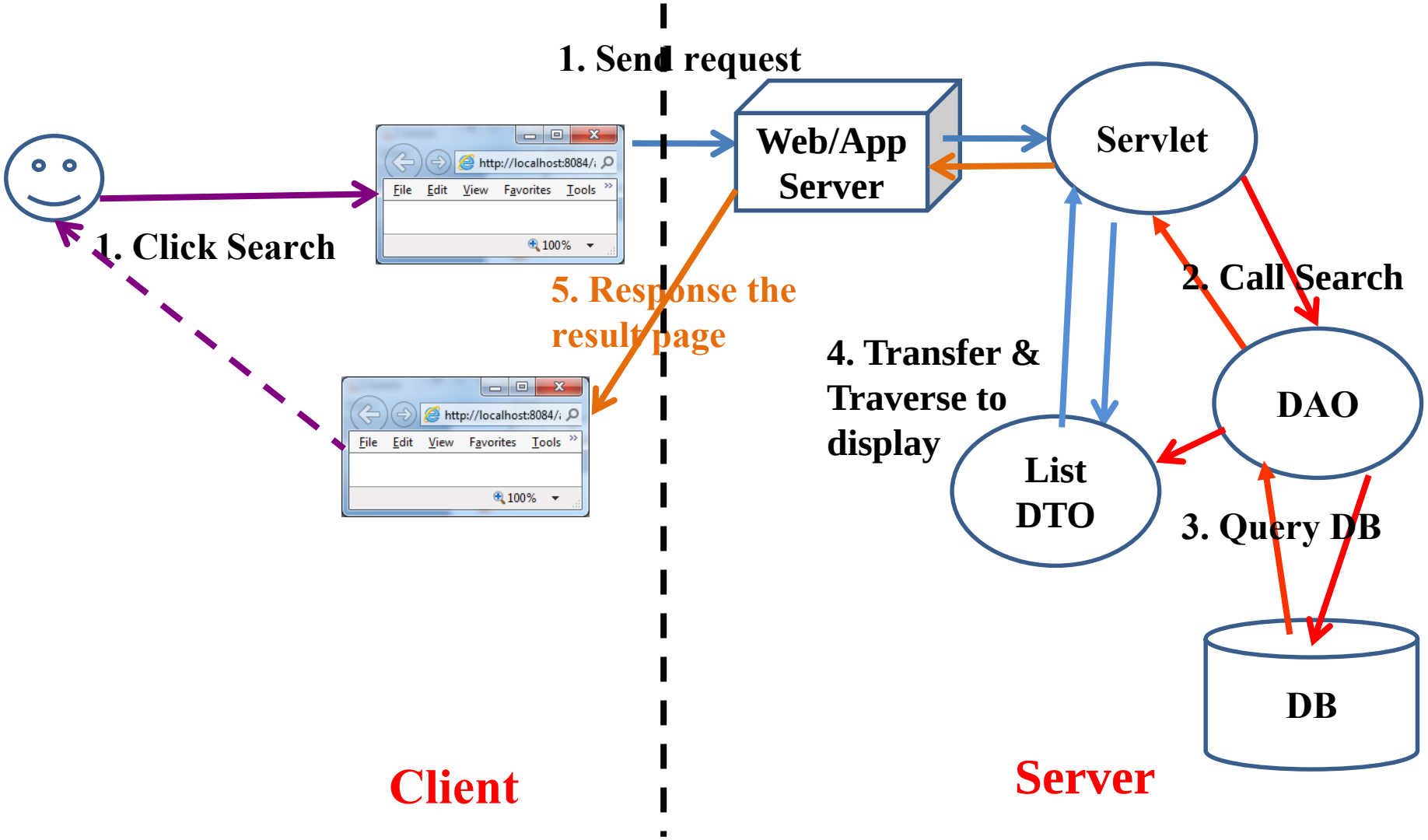
Need for using RequestDispatcher

Include Mechanism



How To Transfer

Interactive Server Model – Implementation

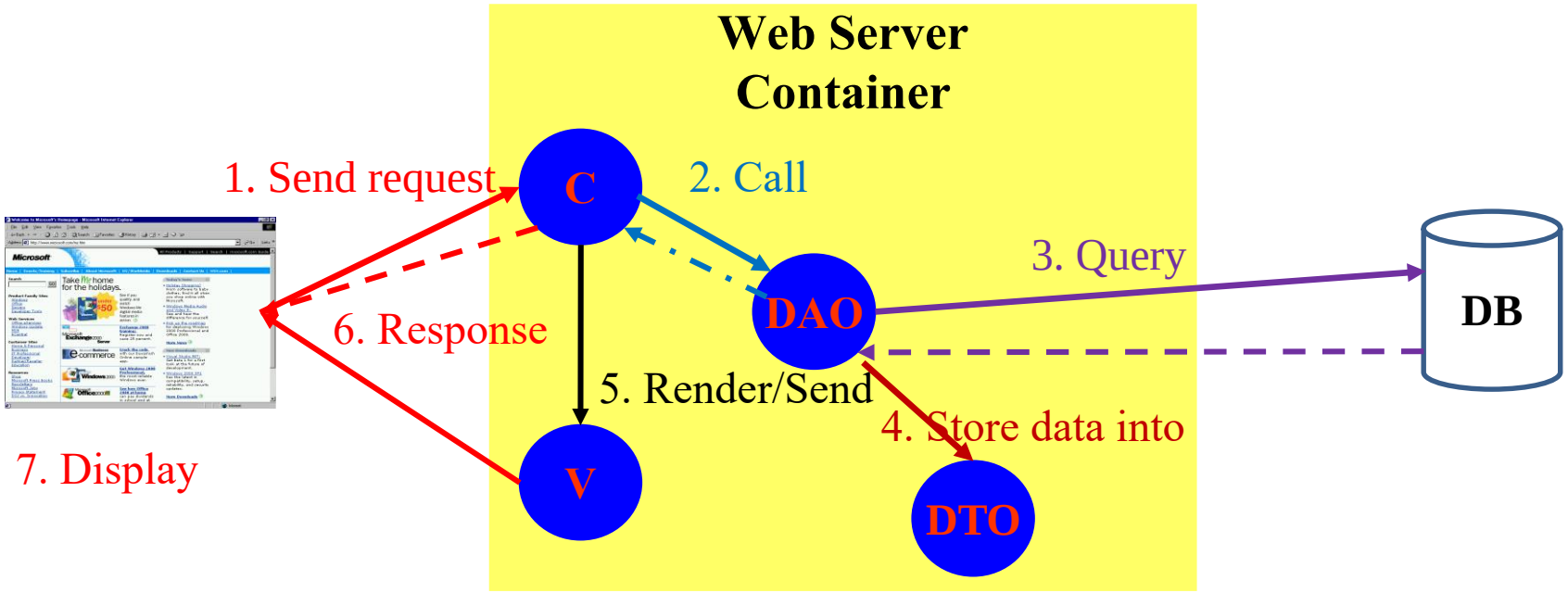


Summary

- **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
  - RequestDispatcher vs. Filter

Q&A

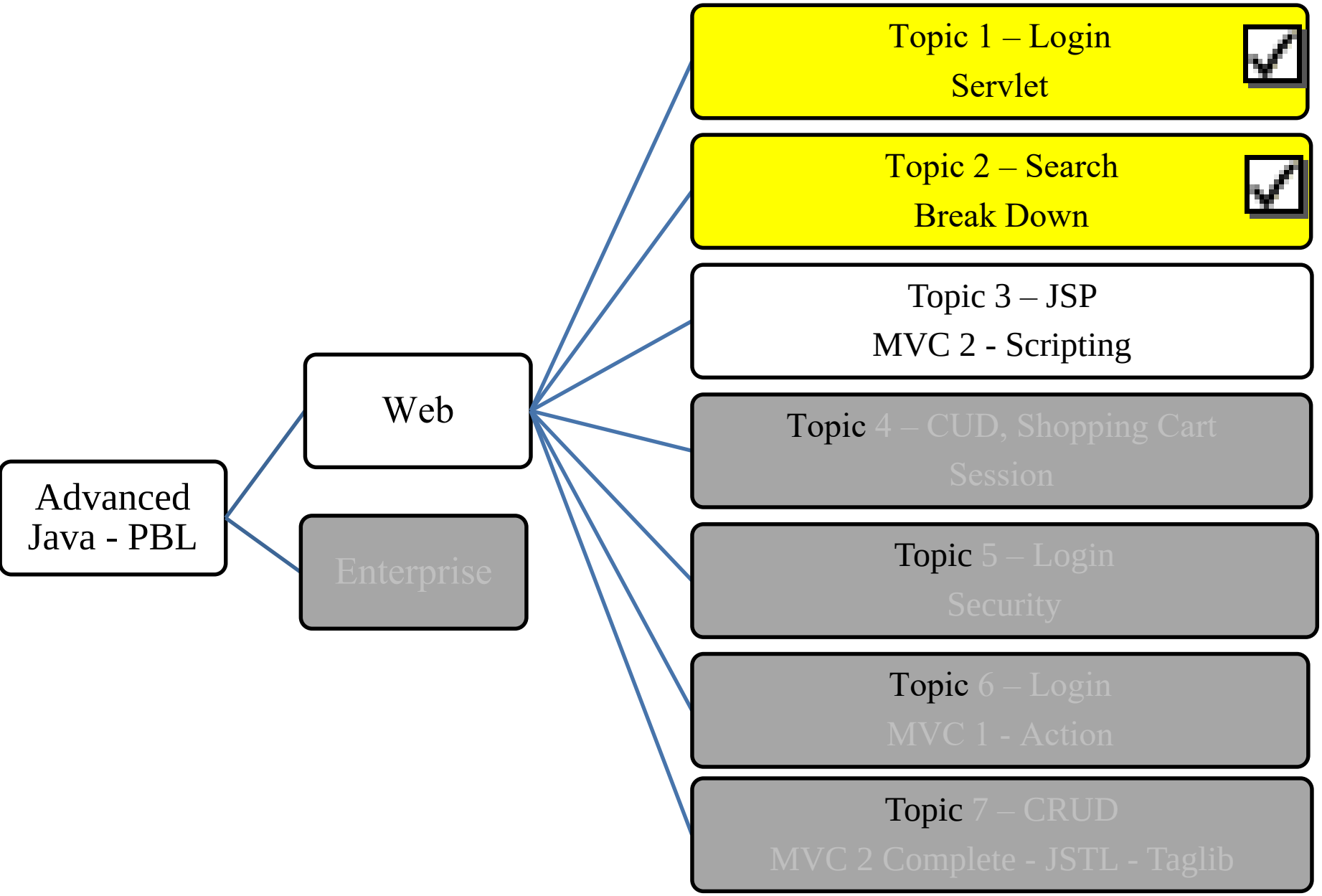
Summary


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

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

Source History

```
13      * @author Trong Khanh
14      */
15     public class RegistrationDTO implements Serializable {
16         private String username;
17         private String password;
18         private String lastname;
19         private boolean roles;
20         public RegistrationDTO() {...2 lines }
22         public RegistrationDTO(String username, String password,
23                 String lastname, boolean roles) {
24             this.username = username;
25             this.password = password;
26             this.lastname = lastname;
27             this.roles = roles;
28         }
29         /**...3 lines */
32         public String getUsername() {...3 lines }
35         /**...3 lines */
38         public void setUsername(String username) {...3 lines }
41         /**...3 lines */
44         public String getPassword() {...3 lines }
47         /**...3 lines */
50         public void setPassword(String password) {...3 lines }
53         /**...3 lines */
56         public String getLastname() {...3 lines }
59         /**...3 lines */
62         public void setLastname(String lastname) {...3 lines }
65         /**...3 lines */
68         public boolean isRoles() {...3 lines }
71         /**...3 lines */
74         public void setRoles(boolean roles) {...3 lines }
```

How to Transfer DAO

RegistrationDAO.java

Source History

```
18  *
19  * @author Trong Khanh
20  */
21 public class RegistrationDAO implements Serializable {
22
23     public boolean checkLogin(String username, String password) {...39 lines }
62
63     private List<RegistrationDTO> listAccounts;
64
65     public void searchLikeLastname (String name) {
66         Connection con = null;
67         PreparedStatement stm = null;
68         ResultSet rs = null;
69         try {
70             con = DBUtils.makeConnection();
71             if (con != null) {
72                 //1. tao truy van lay du lieu duoi DB
73                 String sql = "Select * From Registration Where lastname Like ?";
74
75                 stm = con.prepareStatement(sql);
76                 stm.setString(1, "%" + name + "%");
77
78                 //2. xu ly du lieu
79                 rs = stm.executeQuery();
80                 listAccounts = new ArrayList<RegistrationDTO>();
81                 while (rs.next()) {
82                     String username = rs.getString("username");
83                     String password = rs.getString("password");
84                     String lastname = rs.getString("lastname");
85                     boolean roles = rs.getBoolean("isAdmin");
```

How to Transfer DAO

```
86
87                      RegistrationDTO reg = new RegistrationDTO(username,
88                              password, lastname, roles);
89                      listAccounts.add(reg);
90                  }
91              }
92          } catch (SQLException ex) {
            ex.printStackTrace();
94          } finally {
95              try {
96                  if (rs != null) {
97                      rs.close();
98                  }
99                  if (stm != null) {
100                     stm.close();
101                 }
102                 if (con != null) {
103                     con.close();
104                 }
105             } catch (SQLException ex) {
                ex.printStackTrace();
107             }
108         }
109     }
110
111     public List<RegistrationDTO> getListAccounts() {
112         return listAccounts;
113     }
```

How to Transfer

Process Servlet

ProcessServlet.java

Source History

```
22
23      private final String loginServlet = "LoginServlet";
24      private final String searchServlet = "SearchServlet";
25
26      /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> ...9
35      protected void processRequest(HttpServletRequest request, HttpServletResponse res
36              throws ServletException, IOException {
37          response.setContentType("text/html;charset=UTF-8");
38          PrintWriter out = response.getWriter();
39          try {
40              String button = request.getParameter("btAction");
41
42              if (button.equals("Login")) {
43                  RequestDispatcher rd = request.getRequestDispatcher(loginServlet);
44                  rd.forward(request, response);
45              } else if (button.equals("Search")) {
46                  RequestDispatcher rd = request.getRequestDispatcher(searchServlet);
47                  rd.forward(request, response);
48              }
```

How to Transfer

Search Servlet

SearchServlet.java

Source History

```
21      * @author Trong Khanh
22      */
23     public class SearchServlet extends HttpServlet {
24
25         private final String resultPage = "welcome.html";
26         private final String showSearchResult = "SearchResultServlet";
27
28         /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> ...9 lines
37         protected void processRequest(HttpServletRequest request, HttpServletResponse response)
38                 throws ServletException, IOException {
39             response.setContentType("text/html;charset=UTF-8");
40             PrintWriter out = response.getWriter();
41             try {
42                 /* TODO output your page here. You may use following sample code. */
43                 String name = request.getParameter("txtName");
44
45                 RegistrationDAO dao = new RegistrationDAO();
46                 dao.searchLikeLastname(name);
47                 List<RegistrationDTO> result = dao.getListAccounts();
48
49                 request.setAttribute("INFO", result);
50                 RequestDispatcher rd = request.getRequestDispatcher(showSearchResult);
51                 rd.forward(request, response);
52             } finally {
53                 out.close();
54             }
55         }
```

How to Transfer

Search Result Servlet

SearchResultServlet.java

Source History

```
21    * @author Trong Khanh
22    */
23   public class SearchResultServlet extends HttpServlet {
24
25       /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> ...9 lines
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 name = request.getParameter("txtName");
49               List<RegistrationDTO> result = (List<RegistrationDTO>)
50                       request.getAttribute("INFO");
```

How to Transfer

Search Result Servlet

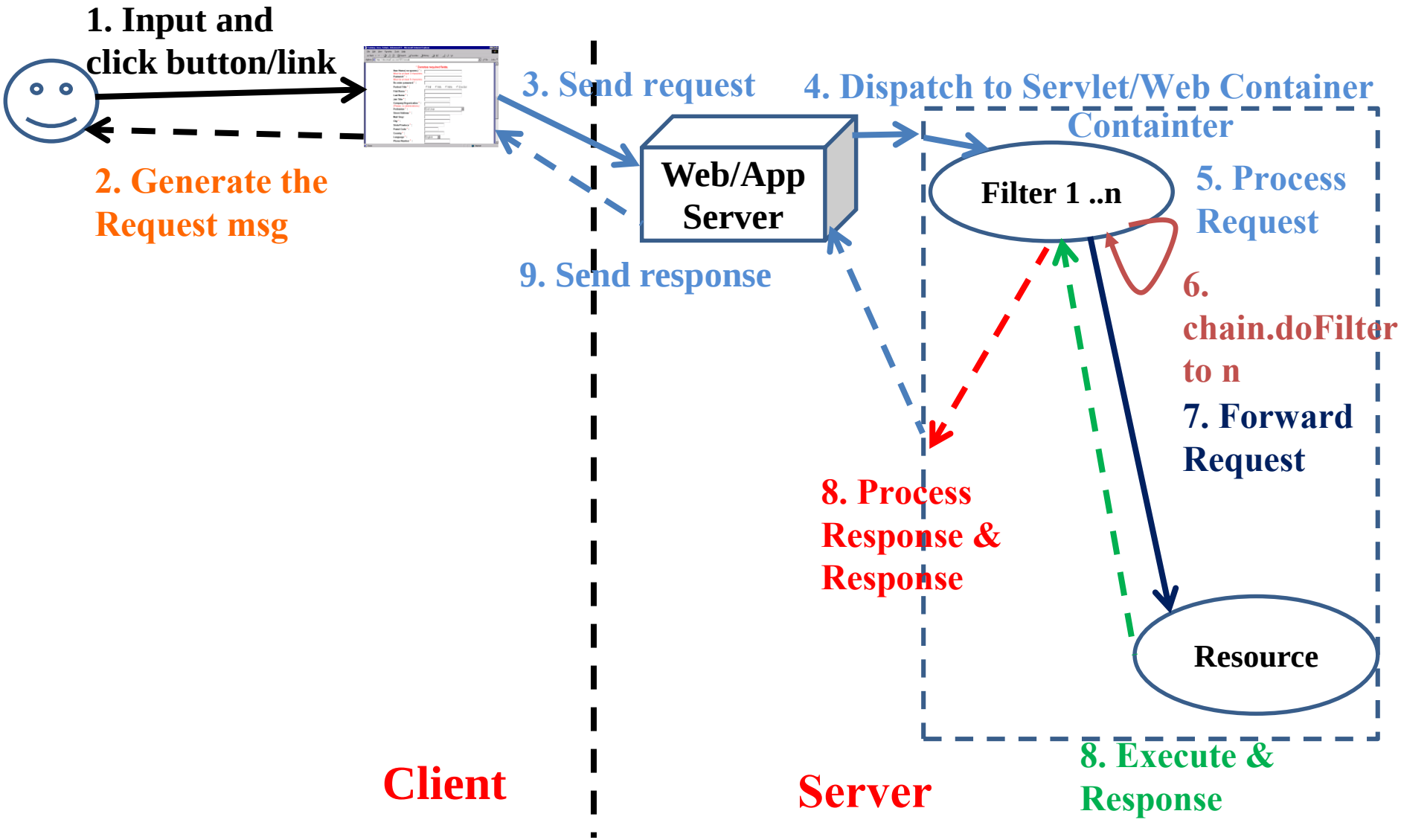
```
52              if (result != null) {
53                  if (result.size() > 0) {
54                      out.println("<table border='1'>");
55                      out.println("<thead>");
56                      out.println("<tr>");
57                      out.println("<th>No.</th>");
58                      out.println("<th>Username</th>");
59                      out.println("<th>Password</th>");
60                      out.println("<th>Lastname</th>");
61                      out.println("<th>Roles</th>");
62                      out.println("</tr>");
63                      out.println("</thead>");
64                      out.println("<tbody>");
65
66                      for (int i = 0; i < result.size(); i++) {
67                          RegistrationDTO reg = result.get(i);
68
69                          out.println("<form action='ProcessServlet' method='POST'>");
70                          out.println("<tr>");
71                          out.println("<td>" + (i + 1) + "</td>");
72                          out.println("<td>"
73                                  + reg.getUsername()
74                                  + "<input type='hidden' name='txtUsername' value='"
75                                  + reg.getUsername() + "' />"
76                                  + "</td>");
77                          out.println("<td>"
78                                  + "<input type='text' name='txtPassword' value='"
79                                  + reg.getPassword() + "' />"
80                                  + "</td>");
```

How to Transfer

Search Result Servlet

```
81                      out.println("<td>" + reg.getLastname() + "</td>");
82                      if (reg.isRoles()) {
83                          out.println("<td>"
84                                  + "<input type='checkbox' name='chkAdmin' value='ADMIN' checked='checked' />"
85                                  + "</td>");
86                      } else {
87                          out.println("<td>"
88                                  + "<input type='checkbox' name='chkAdmin' value='ADMIN' />"
89                                  + "</td>");
90                      }
91                      out.println("</tr>");
92                      out.println("</form>");
93                  }//end for
94
95                  out.println("</tbody>");
96                  out.println("</table>");
97
98                  return;
99              }
100         }
101
102         out.println("<h2>No record is matched!!!!</h2>");
103
104         out.println("</body>");
105         out.println("</html>");
106     } finally {
107         out.close();
108     }
109 }
```

Appendix
Filter



Appendix

Filter

- **Are components that add functionality to the request and response processing of a Web Application**
  - **Intercept** the requests and response that flow between a client and a Servlet/JSP.
  - Supports **dynamic modification of requests and responses** between client and web applications.
  - Dynamically **access incoming requests** from the user before the servlet processes the request
  - **Access the outgoing response** from the web resources before it reaches the user
- **Categorized** according to the **services** they provide to the web applications
- **Resides** in the **web container** along with the web applications
- Was introduced as a Web component in Java servlet specification version 2.3

Appendix

Filter

- **Usage**
  - Authorize request
  - Altering request headers and modify data
  - Modify response headers and data
  - Authenticating the user
  - Comprising files
  - Encrypting data
  - Converting images
  - Logging and auditing filters
  - Filters that trigger resource access events

Appendix

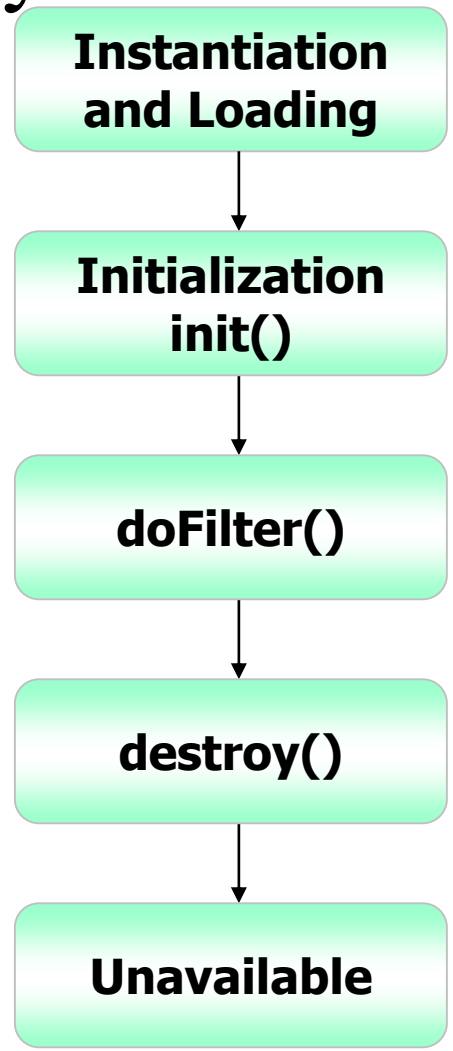
Filter

- **Benefits – Advantages**
  - Optimization of the time taken to send a response
  - Compression of the content size before sending
  - Optimization of the bandwidth
  - Security
  - Identify the type of request coming from the Web client, such as HTTP and FTP, and invoke the Servlet that needs to process the request.
  - Retrieve the user information from the request parameters to authenticate the user.
  - Validate a client using Servlet filters before the client accesses the Servlet.
  - Identify the information about the MIME types and other header contents of the request.
  - Facilitate a Servlet to communicate with the external resources.
  - Intercept responses and compress it before sending the response to the client

Appendix

Filter Life Cycle

- Working of Filter
  - The filter intercepts the request from a user to the servlet
  - The filter then provides customized services
  - The filter sends the serviced response or request to the appropriate destination



Appendix

Filter API

- **Creates and handles** the functionalities of a filter
- Contains **three interfaces**
  - Filter Interface, FilterConfig Interface, FilterChain Interface
- **Filter Interface**
  - Must be implemented to create a filter class **extends javax.servlet.Filter**
  - An object performs filtering tasks on the request and the response

Methods	Descriptions
init	- **public void init(FilterConfig fg);** - Called by the servlet container to initialize the filter - Called only once - Must complete successfully before the filter is asked to do any filtering work
doFilter	- **public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException, ServletException** - Called by the container each time a request or response is processed - Then examines the request/response headers & customizes them as per the requirements - Passed the request/response through the FilterChain object to the next entity in the chain
destroy	- **public void destroy();** - Called by the servlet container to inform the filter that its service is no more required - Called only once.

Appendix
Filter

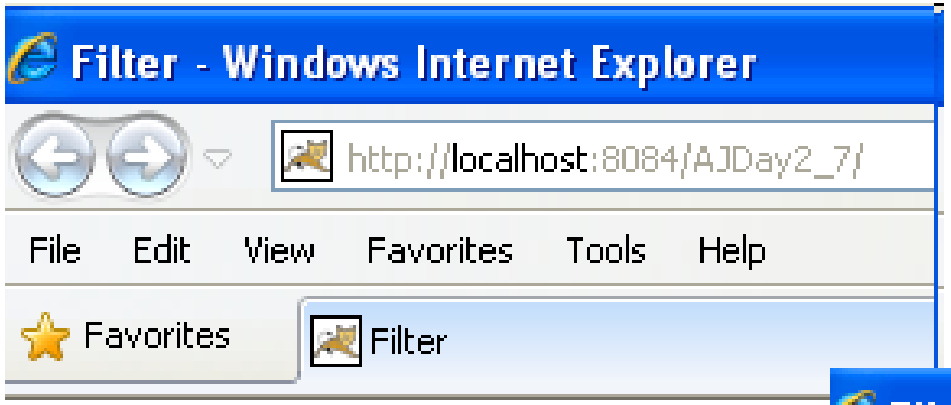
- **In Web Deployment Descriptor**

```
<web-app>
….
<filter>
    <filter-name>Name of Filters</filter-name>
    <filter-class>implemented Filter Class</filter-class>
     [<init-param>
          <param-name>parameter name</param-name>
           <param-value>value </param-value>
     </init-param>]
  </filter>
  <filter-mapping>
     <filter-name>FilterName</filter-name>
     <url-pattern>/context</url-pattern>
  </filter-mapping>
  ….
</web-app>
```

Appendix

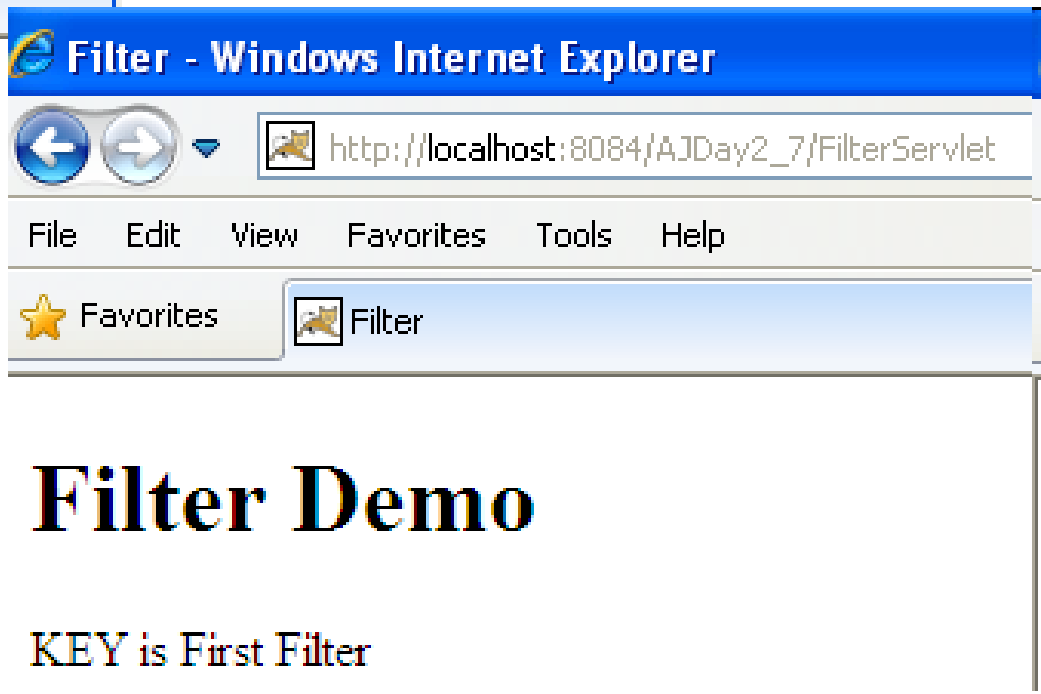
Filter – Example

- Building the web application shows as the following GUI in sequence



Filter Demo

Click here to see Filter Servlet



Filter Demo

KEY is First Filter

Appendix

Filter – Example

filterDemo.html

```
1  ...
5  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
6  <html>
7    <head>
8      <title>Filter</title>
9      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
10   </head>
11   <body>
12       <h1>Filter Demo</h1>
13
14       <a href="FilterServlet">Click here to see Filter Servlet</a>
15   </body>
16 </html>
```

Appendix

Filter – Example

FilterServlet.java

```
17      * @author Trong Khanh
18      */
19     public class FilterServlet extends HttpServlet {
20
21         /**...*/
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                 out.println("<html>");
34                 out.println("<head>");
35                 out.println("<title>Filter</title>");
36                 out.println("</head>");
37                 out.println("<body>");
38                 out.println("<h1>Filter Demo</h1>");
39
40                 String test = (String)request.getAttribute("KEY");
41                 out.println("KEY is " + test);
42
43                 out.println("</body>");
44                 out.println("</html>");
45             } finally {
46                 out.close();
47             }
48         }
```

Appendix

Filter – Example

New File

Steps

1. **Choose File Type**
2. ...

Choose File Type

Project: AJDay2_7

Categories:

- Web
- JavaServer Faces
- Bean Validation
- Struts
- Spring Framework
- Contexts and Dependency Inj
- Java
- Swing GUI Forms
- JavaBeans Objects
- AWT GUI Forms

File Types:

- JSP
- JSF Page
- Servlet
- Filter
- Web Application Listener
- WebSocket Endpoint
- HTML
- XHTML
- Cascading Style Sheet
- JavaScript File
- JSON File
- Tag Handler

Description:

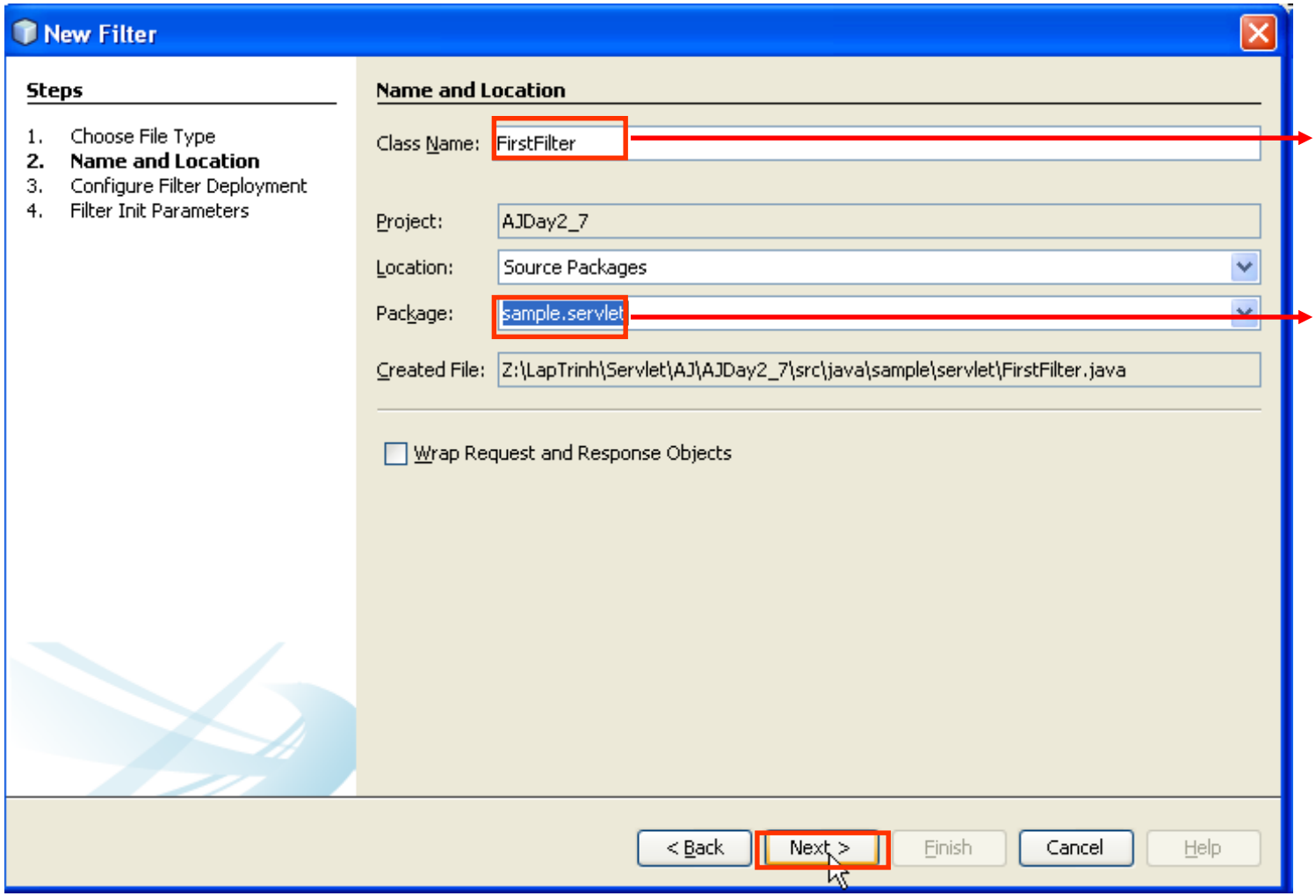
Creates a new filter class. Filters can pre-process a request before it reaches a servlet, post-process a response leaving a servlet, or do both. Filters can intercept, examine, and modify requests and responses.

- Click Next Button

< Back | Next > | Finish | Cancel | Help

Appendix

Filter – Example



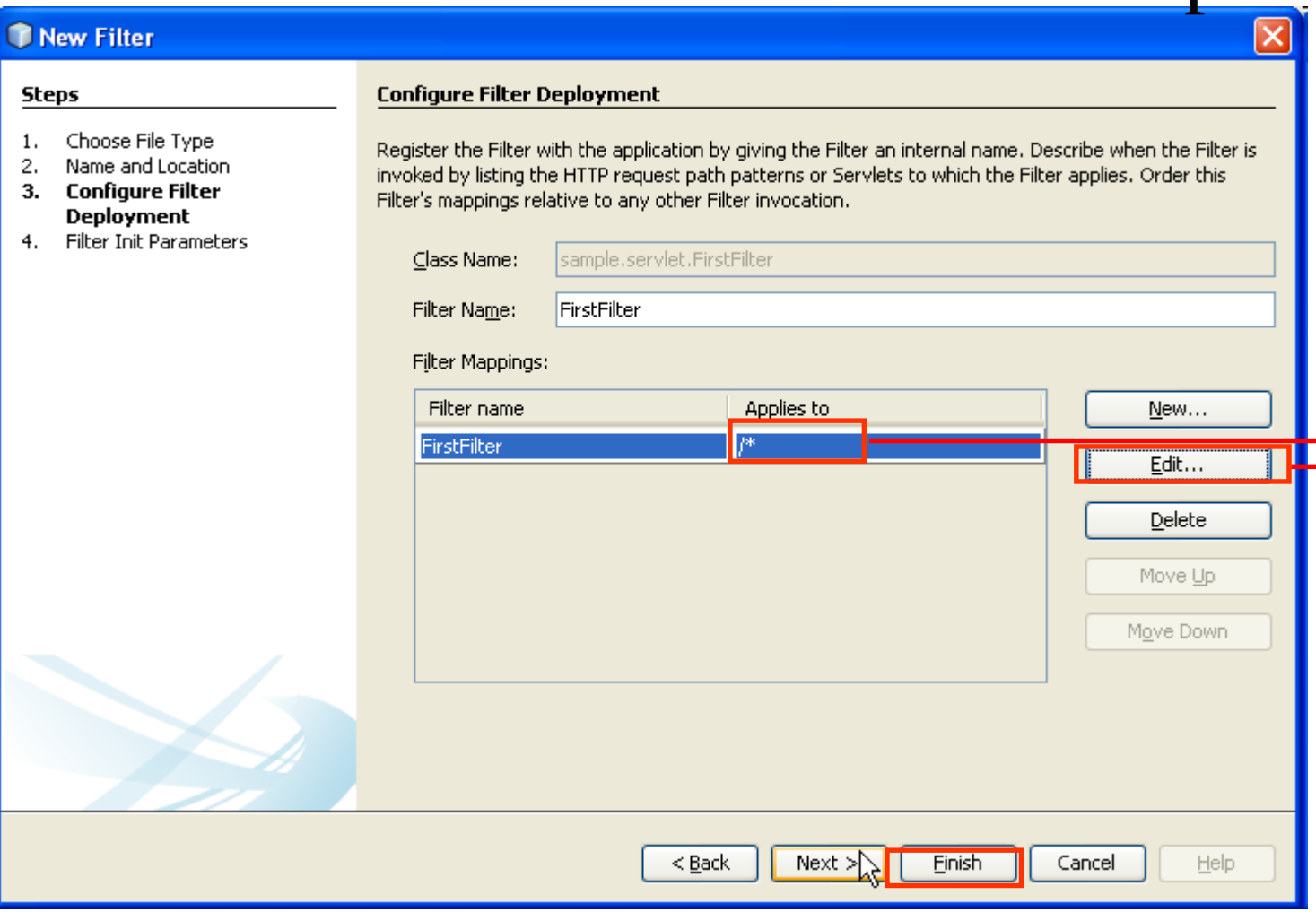
Fill your filter name

Fill/choose package name

- Click Next Button

Appendix

Filter – Example



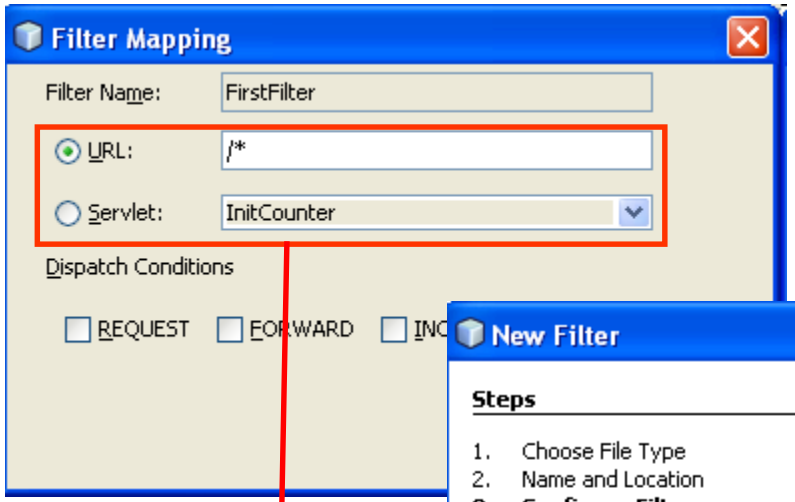
Apply filter

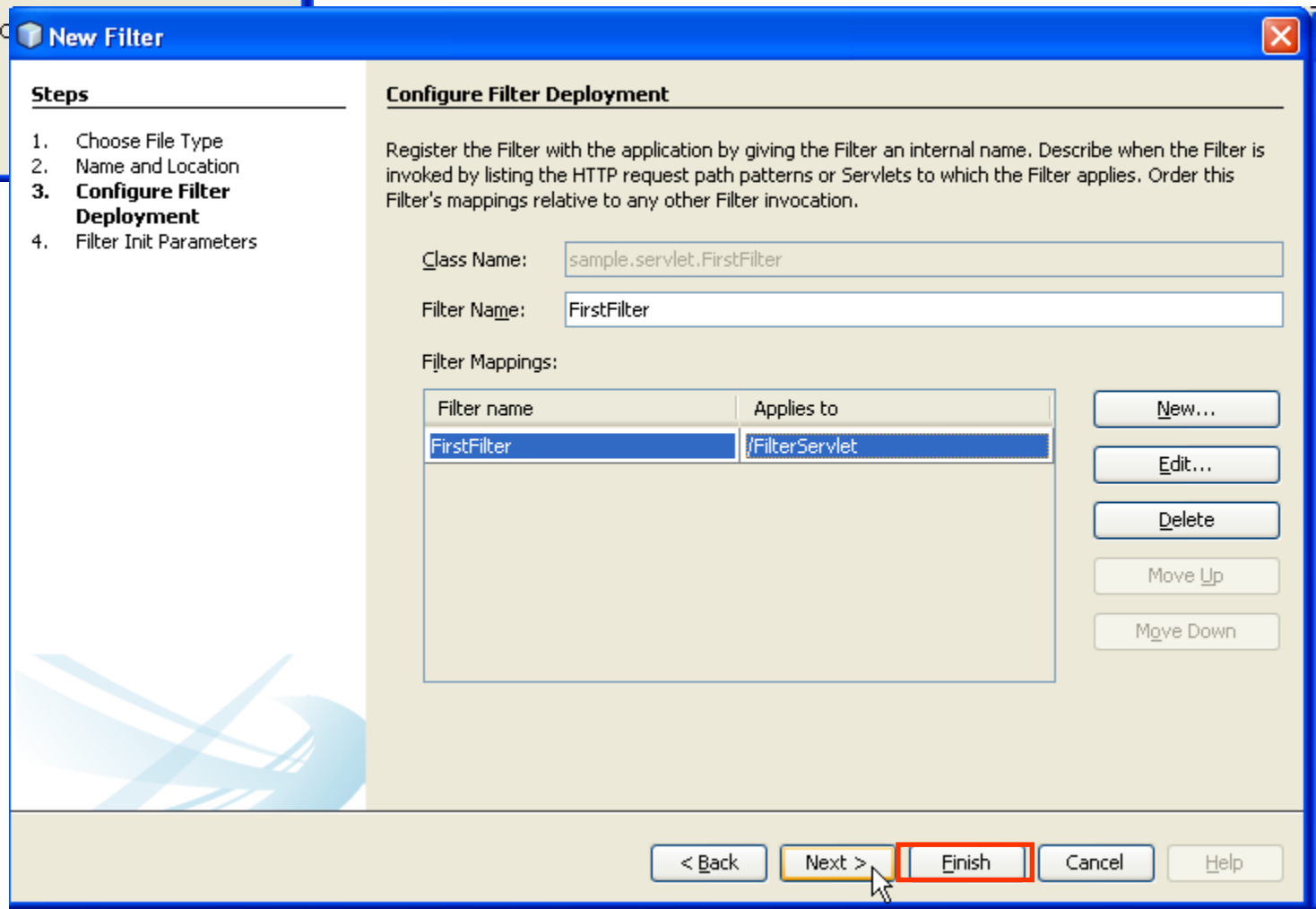
Edit the Apply filter

- Click Edit Button to apply Filter the selected Servlet
- Otherwise, click Finish Button

Appendix

Filter – Example





Select the URL and typing the URL string, or Select the Servlet and choose the approximate Servlet in combo box

Appendix

Filter – Example

web.xml | FilterServlet.java | FirstFilter.java

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

```
11      <filter>
12          <filter-name>FirstFilter</filter-name>
13          <filter-class>sample.servlet.FirstFilter</filter-class>
14      </filter>
15      <filter-mapping>
16          <filter-name>FirstFilter</filter-name>
17          <url-pattern>/FilterServlet</url-pattern>
18      </filter-mapping>
19      <servlet>
```

FirstFilter.java | web.xml

```
95      */
96      public void doFilter(ServletRequest request, ServletResponse response,
97                           FilterChain chain)
98          throws IOException, ServletException {
99          try {
100             System.out.println("Passed First Filter - request");
101             request.setAttribute("KEY", "First Filter");
102             chain.doFilter(request, response);
103             System.out.println("Passed First Filter - response");
104             request.setAttribute("KEY", "First Filter Again");
105         }
106         catch(Throwable t) {
                t.printStackTrace();
108         }
109     }
```

Appendix

Filter – Example

Output

Apache Tomcat 7.0.27.0 | Apache Tomcat 7.0.27.0 Log | AJDay2_7 (run-deploy)

```
thg 10 30, 2013 8:08:38 SA org.apache.catalina.core.ApplicationContext log
INFO: FirstFilter:Initializing filter
```

Output

Apache Tomcat 7.0.27.0 | Apache Tomcat 7.0.27.0 Log | AJDay2_7 (run-deploy)

```
Passed First Filter - request
Passed First Filter - response
```

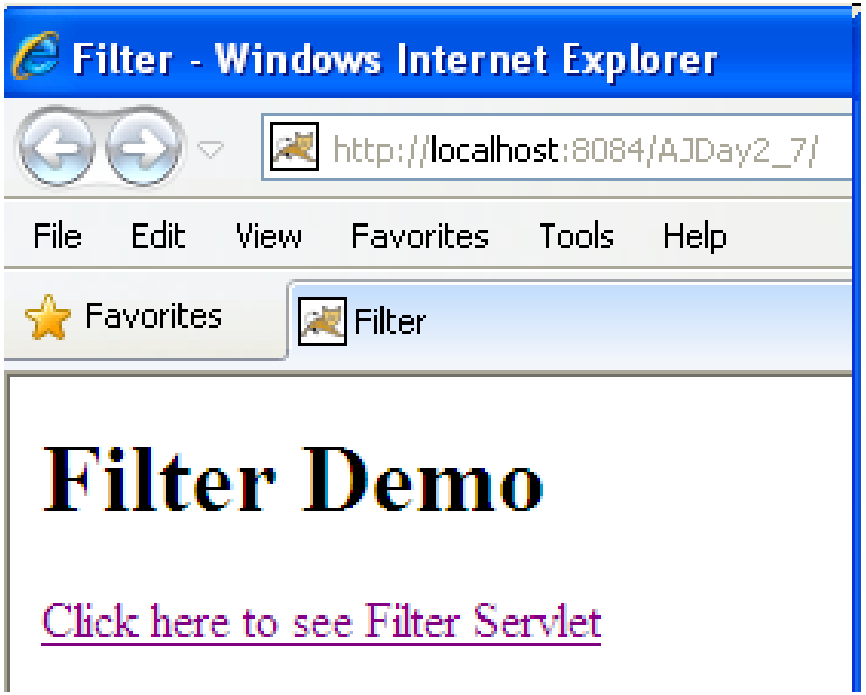
web.xml | FilterServlet.java | FirstFilter.java

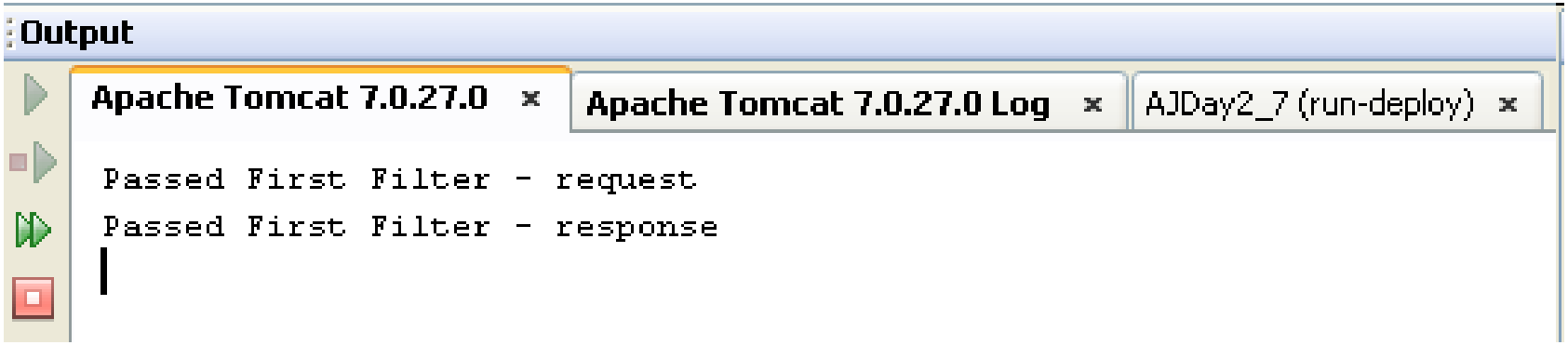
Source | General | Servlets | Filters | Pages | References | Security | History

```
11  <filter>
12      <filter-name>FirstFilter</filter-name>
13      <filter-class>sample.servlet.FirstFilter</filter-class>
14  </filter>
15  <filter-mapping>
16      <filter-name>FirstFilter</filter-name>
17      <url-pattern>/*</url-pattern>
18  </filter-mapping>
```

Appendix

Filter – Example

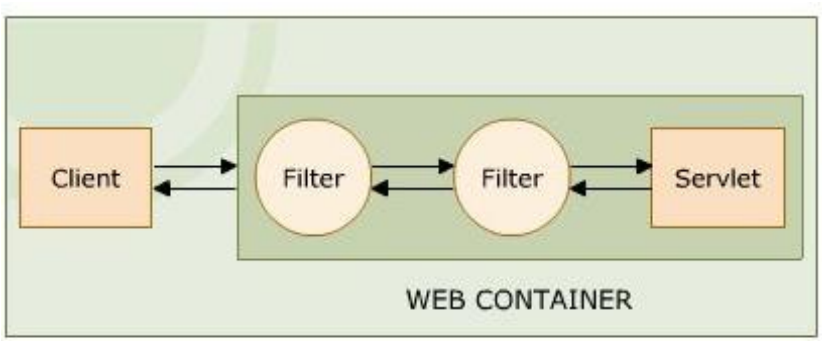




Appendix

Filter Chain

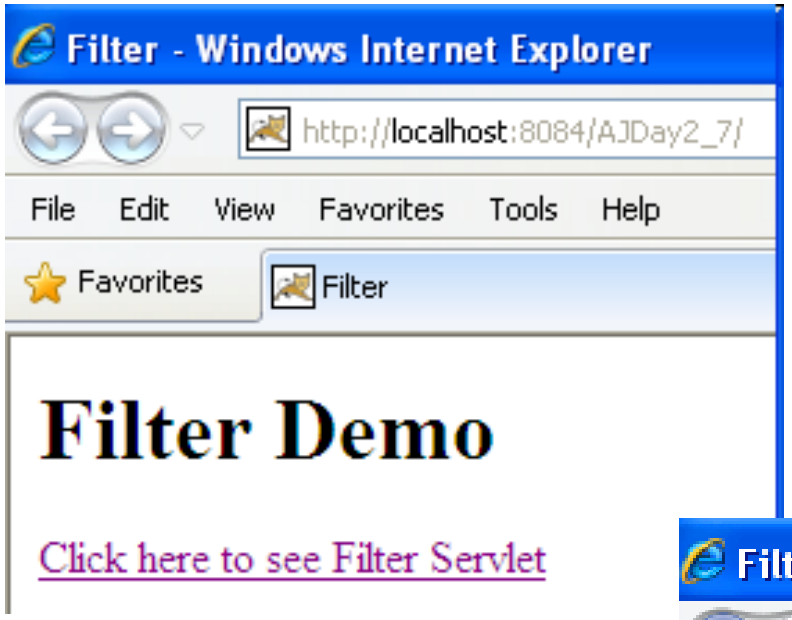
- There can be **more than one filter** between the user and the endpoint – Invoke a **series of filters**
- A request or a response is **passed through one** filter to the **next** in the filter chain. So each request and response has to be serviced by each filter forming a filter chain
- If the Calling filter is last filter, will invoke web resource
- **FilterChain Interface**
  - Provides an object through the web container
  - The object invokes the next filter in a filter chain starting from the first filter from a particular end. If the calling filter is the last filter in the chain, it will invoke the web resource, such as JSP and servlet.
  - Only implement doFilter() method.
  - Forces the next filter in the chain to be invoked

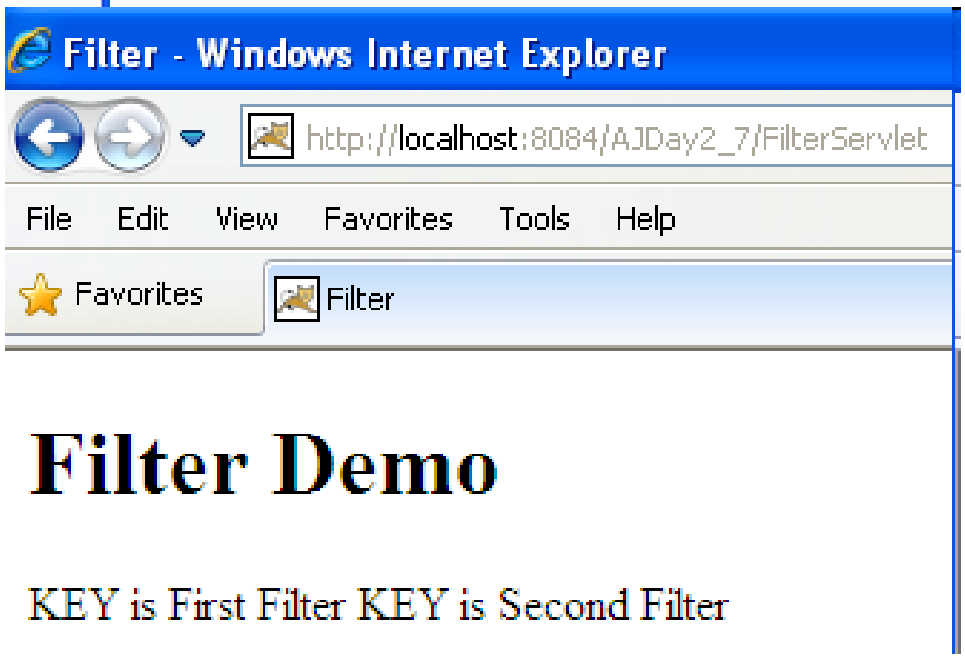


Filter Chain

Appendix

Filter Chain – Example





Appendix

Filter Chain – Example

web.xml | SecondFilter.java

```
public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
    try {
        System.out.println("Pass Second Filter - request");
        request.setAttribute("KEY1", "Second Filter");
        chain.doFilter(request, response);
        System.out.println("Pass Second Filter - response");
        request.setAttribute("KEY1", "Second Filter Again");
    } catch (Throwable t) {
        t.printStackTrace();
    }
}
```

Appendix

Filter Chain – Example

web.xml | FilterServlet.java

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

```
11  <filter>
12      <filter-name>FirstFilter</filter-name>
13      <filter-class>sample.servlet.FirstFilter</filter-class>
14  </filter>
15  <filter>
16      <filter-name>SecondFilter</filter-name>
17      <filter-class>sample.servlet.SecondFilter</filter-class>
18  </filter>
19  <filter-mapping>
20      <filter-name>SecondFilter</filter-name>
21      <url-pattern>/FilterServlet</url-pattern>
22  </filter-mapping>
23  <filter-mapping>
24      <filter-name>FirstFilter</filter-name>
25      <url-pattern>/FilterServlet</url-pattern>
26  </filter-mapping>
```

Appendix

Filter Chain – Example

web.xml | FilterServlet.java

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

```
11  <filter>
12      <filter-name>SecondFilter</filter-name>
13      <filter-class>sample.servlet.SecondFilter</filter-class>
14  </filter>
15  <filter>
16      <filter-name>FirstFilter</filter-name>
17      <filter-class>sample.servlet.FirstFilter</filter-class>
18  </filter>
19  <filter-mapping>
20      <filter-name>SecondFilter</filter-name>
21      <url-pattern>/FilterServlet</url-pattern>
22  </filter-mapping>
23  <filter-mapping>
24      <filter-name>FirstFilter</filter-name>
25      <url-pattern>/FilterServlet</url-pattern>
26  </filter-mapping>
27  <servlet>
```

Appendix

Filter Chain – Example

web.xml | FilterServlet.java

```
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              out.println("<html>");
34              out.println("<head>");
35              out.println("<title>Filter</title>");
36              out.println("</head>");
37              out.println("<body>");
38              out.println("<h1>Filter Demo</h1>");
39
40              String test = (String)request.getAttribute("KEY");
41              out.println("KEY is " + test);
42
43              String test1 = (String) request.getAttribute("KEY1");
44              out.println("KEY is " + test1);
45
46              out.println("</body>");
47              out.println("</html>");
48          } finally {
```

Appendix

Filter Chain – Example

Output

Apache Tomcat 7.0.27.0 | Apache Tomcat 7.0.27.0 Log | AJDay2_7 (run-deploy)

```
thg 10 30, 2013 8:11:16 SA org.apache.catalina.core.ApplicationContext log
INFO: FirstFilter:Initializing filter
thg 10 30, 2013 8:11:16 SA org.apache.catalina.core.ApplicationContext log
INFO: SecondFilter:Initializing filter
```

Output

Apache Tomcat 7.0.27.0 | Apache Tomcat 7.0.27.0 Log | AJDay2_7 (run-deploy)

```
Pass Second Filter - request
Passed First Filter - request
Passed First Filter - response
Pass Second Filter - response
```

Appendix

Filter Chain – Example – **Change pos**

web.xml | FilterServlet.java

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

```
11  <filter>
12      <filter-name>FirstFilter</filter-name>
13      <filter-class>sample.servlet.FirstFilter</filter-class>
14  </filter>
15  <filter>
16      <filter-name>SecondFilter</filter-name>
17      <filter-class>sample.servlet.SecondFilter</filter-class>
18  </filter>
19  <filter-mapping>
20      <filter-name>FirstFilter</filter-name>
21      <url-pattern>/FilterServlet</url-pattern>
22  </filter-mapping>
23  <filter-mapping>
24      <filter-name>SecondFilter</filter-name>
25      <url-pattern>/FilterServlet</url-pattern>
26  </filter-mapping>
27  <servlet>
```

Appendix

Filter Chain – Example

Output

Apache Tomcat 7.0.27.0 | Apache Tomcat 7.0.27.0 Log | AJDay2_7 (run-deploy)

```
thg 10 30, 2013 8:06:32 SA org.apache.catalina.core.ApplicationContext log
INFO: FirstFilter:Initializing filter
thg 10 30, 2013 8:06:32 SA org.apache.catalina.core.ApplicationContext log
INFO: SecondFilter:Initializing filter
```

Output

Apache Tomcat 7.0.27.0 | Apache Tomcat 7.0.27.0 Log | AJDay2_7 (run-deploy)

```
Passed First Filter - request
Pass Second Filter - request
Pass Second Filter - response
Passed First Filter - response
```

Appendix

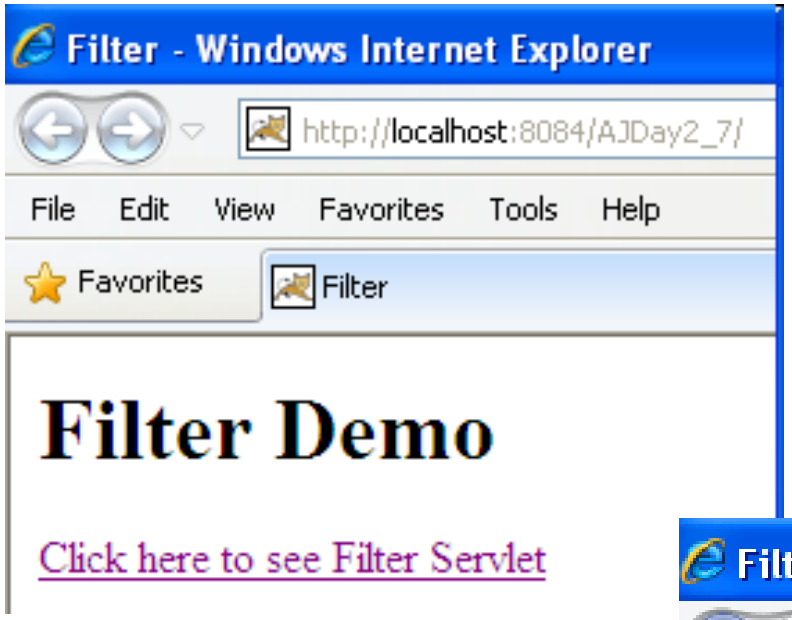
Why need a Wrapper Class

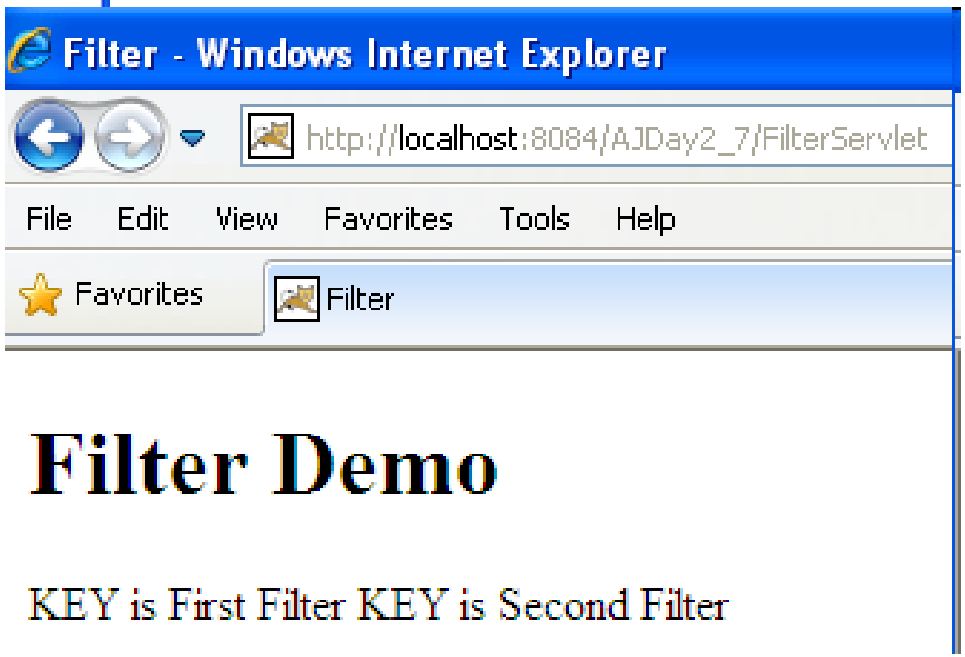
SecondFilter.java

```
99      public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain)
100         throws IOException, ServletException {
101         try {
102             System.out.println("Pass Second Filter - request");
103             request.setAttribute("KEY1", "Second Filter");
104             chain.doFilter(request, response);
105             System.out.println("Pass Second Filter - response");
106             request.setAttribute("KEY1", "Second Filter Again");
107             PrintWriter out = response.getWriter();
108             out.println("<br/>The slide is licensed to KhanhKT");
109         } catch (Throwable t) {
                t.printStackTrace();
111         }
112     }
```

Appendix

Why need a Wrapper Class



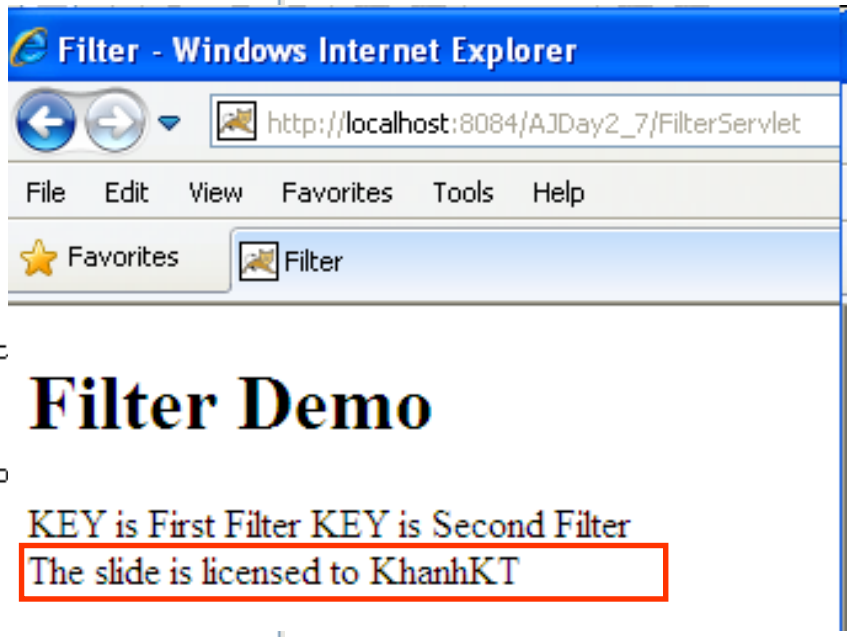


Appendix

Why need a Wrapper Class

FilterServlet.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              out.println("<html>");
34              out.println("<head>");
35              out.println("<title>Filter</title>");
36              out.println("</head>");
37              out.println("<body>");
38              out.println("<h1>Filter Demo</h1>");
39
40              String test = (String)request.getAttribut
41              out.println("KEY is " + test);
42
43              String test1 = (String) request.getAttrib
44              out.println("KEY is " + test1);
45
46              out.println("</body>");
47              out.println("</html>");
48          } finally {
49              //out.close();
50          }
51      }
```



The slide is licensed to KhanhKT

Appendix

Wrapper Class

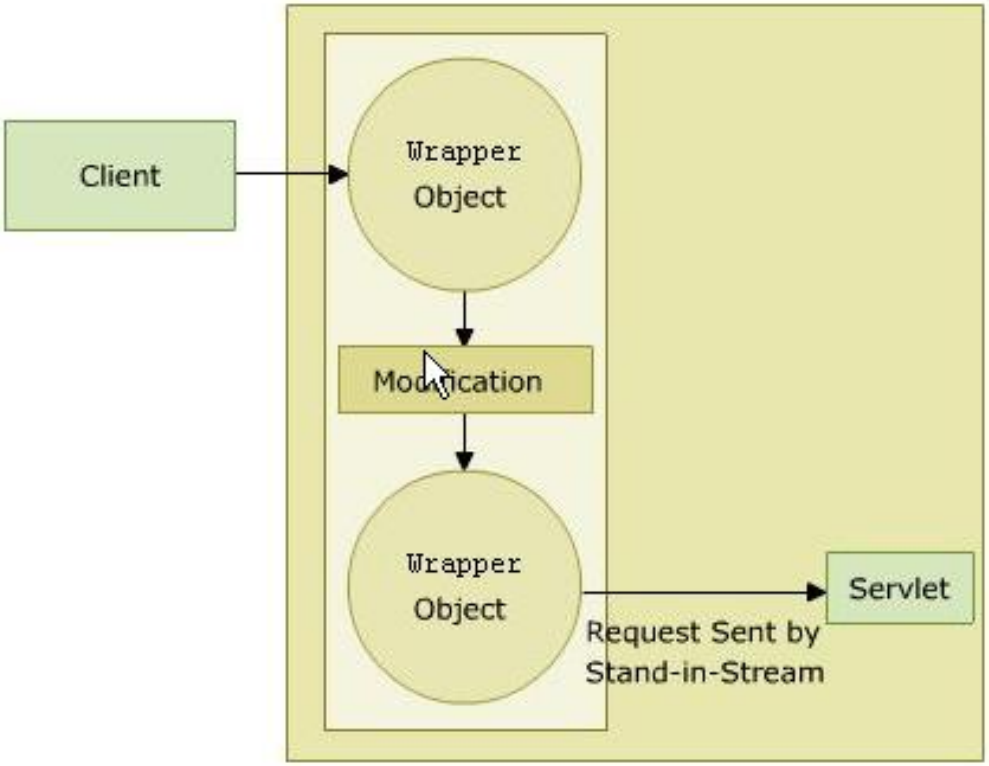
- To modify or intercept the request or response before they can reach their logical destination, the required object can dynamically capture the request or response
- Wrapper class
  - Creates the object to capture the request and response before they reach server and client respectively
  - The wrapper object generated by the filter implements the getWriter() and getOutputStream(), which returns a stand-in-stream. The stand-in-stream is passed to the servlet through the wrapper object
  - The wrapper object captures the response through the stand-in-stream and sends it back to the filter

Classes	Descriptions
ServletRequestWrapper	- Provides a convenient implementation of the ServletRequest interface - Can be sub-classed by developers wishing to send the request to a servlet - To override request methods, one should wrap the request in an object that extends ServletRequestWrapper or HttpServletRequestWrapper
ServletResponseWrapper	- Provides a convenient implementation of the ServletResponse interface - Can be sub classed by developers wishing to send the response from a servlet.

Appendix

Wrapper Class – Altering Request

- Create filter class extends to the **ServletRequestWrapper** or **HttpServletRequestWrapper** class.
- The object captures the HttpRequest object from the client and sends it to the filers
- Through the objects filter extends some services to the request.

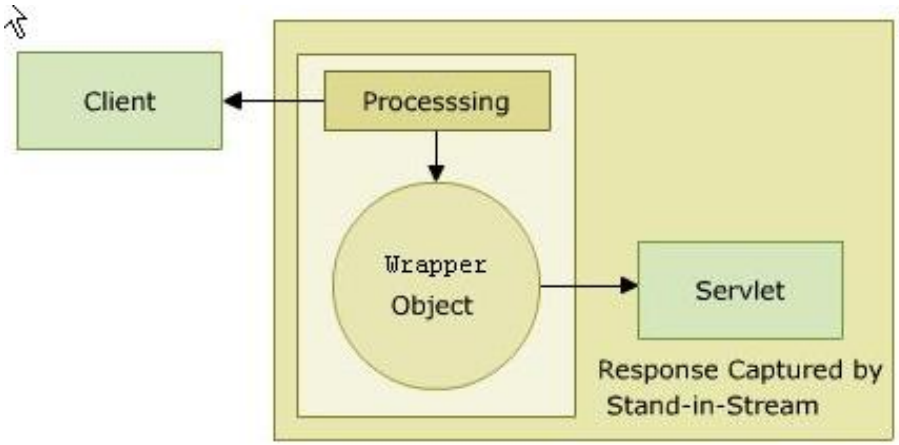


Modifying the Request

Appendix

Wrapper Class – Altering Response

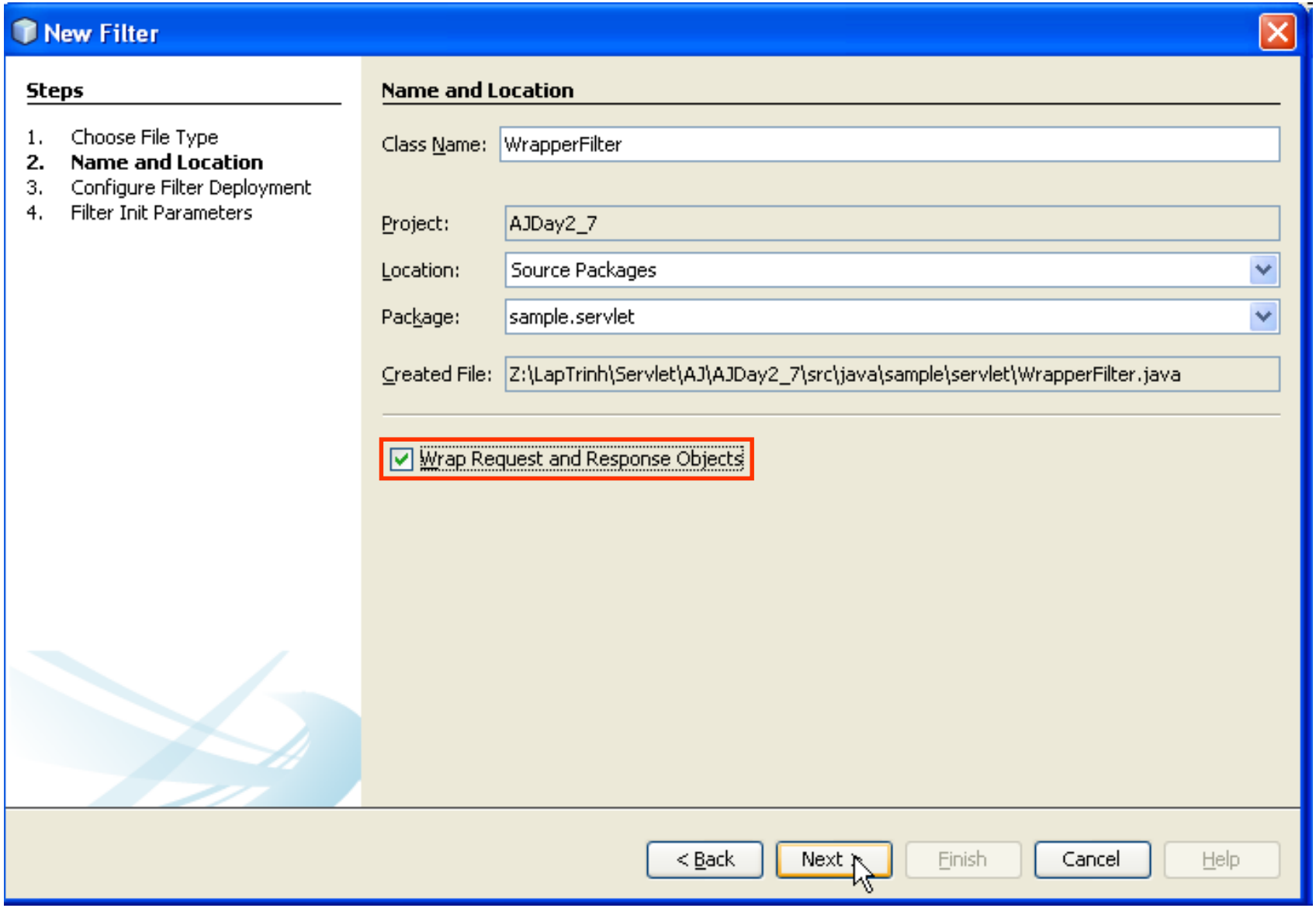
- Create filter class extends to the **ServletResponseWrapper** or **HttpServletResponseWrapper** class.
- The object captures the httpRequest object from the client and sends it to the filers
- Through the objects filter extends some services to the request.



Modifying the Response

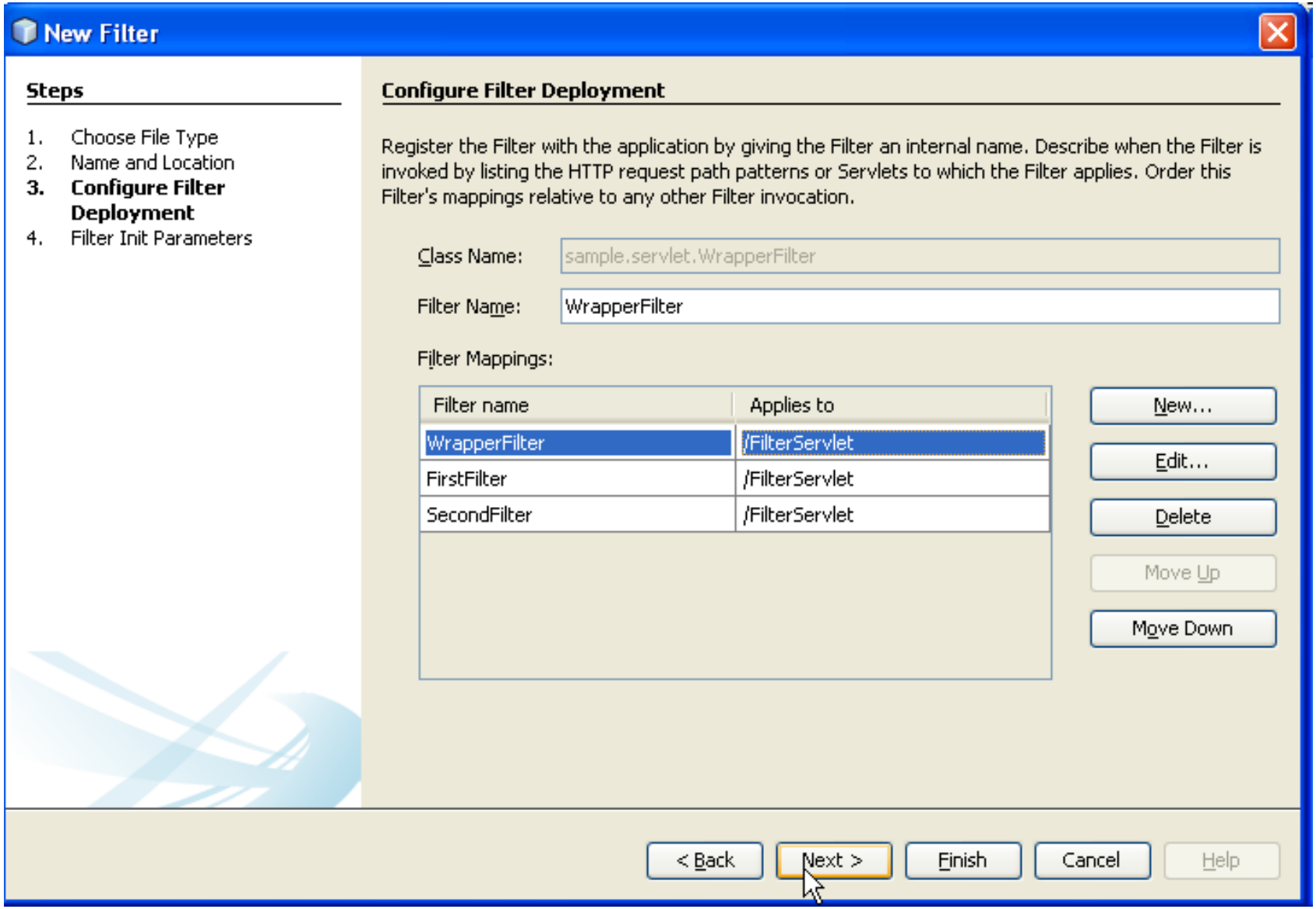
Appendix

Wrapper Class – Example



Appendix

Wrapper Class – Example



Appendix

Wrapper Class – Example

- Adding the MyPrinter class extends PrintWriter in FilterWrapper class

web.xml | WrapperFilter.java

Source | History

```
404                 */
405         }
406         class MyPrinter extends PrintWriter {
407             public MyPrinter (Writer out) {
408                 super(out);
409             }
410
411             public void close() {
412                 super.close();
413             }
414         }
415     }
```

Appendix

Wrapper Class – Example

- Modifying the ResponseWrapper class uses MyPrinter to output stream

WrapperFilter.java

```
323     class ResponseWrapper extends HttpServletResponseWrapper {
324         private MyPrinter out;
325         public ResponseWrapper(HttpServletResponse response) {
326             super(response);
327             try {
328                 out = new MyPrinter(response.getWriter());
329             }catch (IOException e) {
                    e.printStackTrace();
331             }
332         }
333
            public PrintWriter getWriter() throws IOException {
335             return out;
336         }
```

Appendix

Wrapper Class – Example

WrapperFilter.java

```
133         */
            public void doFilter(ServletRequest request, ServletResponse response,
135                                 FilterChain chain)
136             throws IOException, ServletException {
137             HttpServletResponse resp = (HttpServletResponse)response;
138             ResponseWrapper wrapperResp = new ResponseWrapper(resp);
139             try {
140                 chain.doFilter(request, wrapperResp);
141                 PrintWriter out = wrapperResp.getWriter();
142                 out.println("<br/>The slide is licensed to KhanhKT");
143                 out.close();
144             }
145             catch(Throwable t) {
                    t.printStackTrace();
147             }
148         }
```