Fpt University

Web Applications & Web Containers

Web Applications
The Web Container Model

Review

- **The Servlet Model**
  - Form Parameters
  - HTTP Methods: **GET**, **POST**, HEAD, OPTIONS, PUT, DELETE, TRACE
  - Servlet (Servlet → Generic Servlet → HttpServlet → YourServlet)
  - Servlet Life Cycle (**init, service, destroy**)
- **The Web Application Process**
  - **Step 1:** Creating a Web application project
  - **Step 2:** Creating the html with(out) form parameters, Servlets
  - **Step 3:** Writing the code for Servlet & Compile
  - **Step 4:** Building the Web application project
  - **Step 5:** Deploying to a Web Server
  - **Step 6:** Executing the application

Objectives

- **Web applications**
  - File and Directory Structure
  - Deployment Descriptor Elements
  - WAR files
- **The Web Container Model**
  - ServletContext
  - Attributes, Scope, and Multithreading
  - Request Dispatching
  - Filters and Wrappers

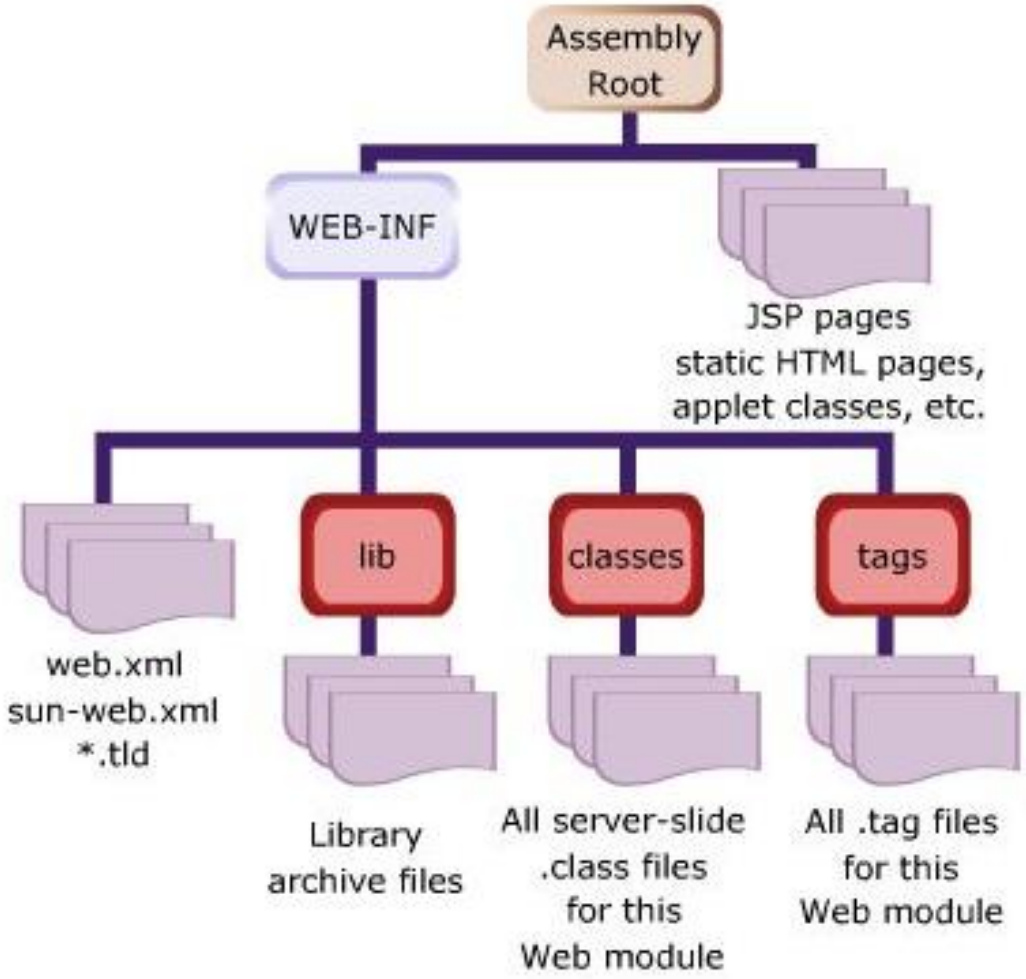
Web Applications

Overview

- A **web application** or **webapp**
  - Is **an application** that is **accessed via web browser** over a network such as the Internet or an intranet.
  - Is also a **computer software application** that is coded in a **browser-supported language** (such as HTML, JavaScript, Java, etc.) and **reliant** on a common web **browser to render** the application executable.
- Web applications are popular due to the **ubiquity** of web browsers, and the convenience of using a web browser as a client, sometimes called a **thin client**.

Web Applications

File and Directory Structure



Above structure is packaged into *.war file to deploy on Web Server

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

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

The Deployment Descriptor

- The Web Deployment Descriptor file **describes all** of **Web components**
- It is an **XML file**. Given that the name is **web.xml.**

```
<web-app>
    <description>
    <display-name>
    <icon>
    <distributable>
    <context-param>
    <filter>
    <filter-mapping>
    <listener>
    <servlet>
    <servlet-mapping>
    <session-config>
    <mime-mapping>
    <welcome-file-list>
    <error-page>
    <jsp-config>
    <security-constraint>
    <login-config>
    <security-role>
```

Web Applications

The Deployment Descriptor – web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <servlet>
                <servlet-name>servlet name</servlet-name>
                <servlet-class>[package.]classname</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>servlet name</servlet-name>
        <url-pattern>/context Path/root</url-pattern>
    </servlet-mapping>
    <session-config>
      <session-timeout>30</session-timeout>
     </session-config>
    <welcome-file-list>
        <welcome-file>default page to show</welcome-file>
    </welcome-file-list></web-app>
```

Web Applications

The Deployment Descriptor – Example

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>servlet.sample.HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/HelloServlet</url-pattern>
    </servlet-mapping>
    <session-config>
      <session-timeout>30</session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>HelloServlet</welcome-file>
    </welcome-file-list></web-app>
```

Web Applications

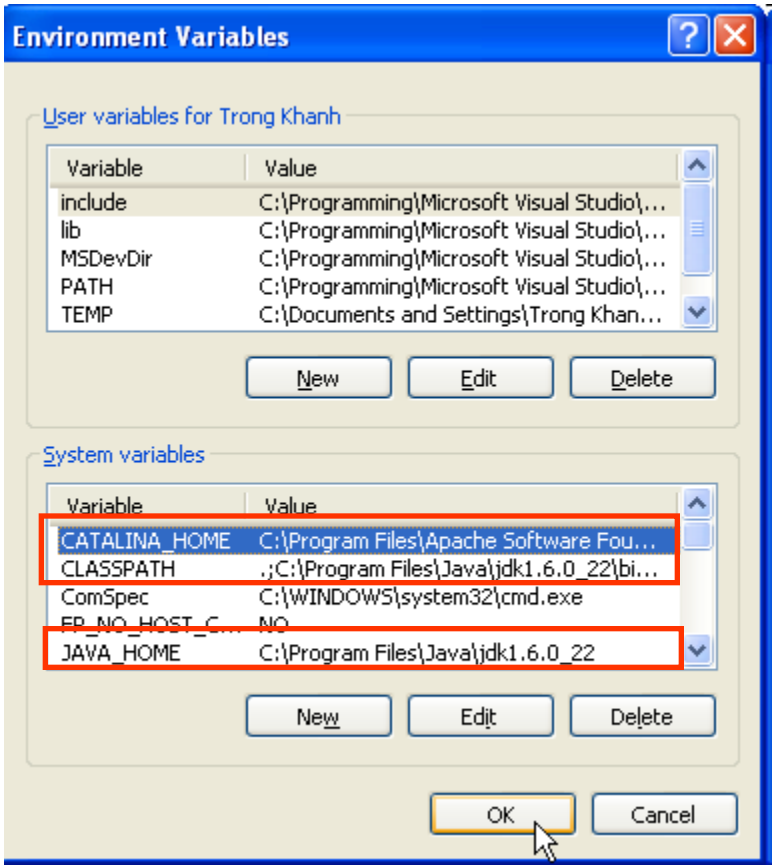
Packaging Your Web Application

- A **WAR** Is Not a **JAR**
  - Although a WAR file can be produced in the same way as a JAR file, and has the **same underlying file format**, it is **different**. The most obvious difference is the file extension naming convention: .jar for **Java ARchive**, and .war for **Web (Application) ARchive**.
  - WARs are packaged for a different purpose: to make it as easy as possible for a **web container** to deploy an application.
- A WAR file
  - Several web containers have automatic deployment mechanisms.
  - The server recommended for this course – Tomcat 6.0.26 – has a "**webapps**" directory. Place a WAR file in this directory, and Tomcat (by default) **will un-jar** the contents into the file system under the webapps directory. It provides a context root directory with the same name as the WAR file (**but without the .war extension**) – then makes the application available for use.

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 **tomcat6w.exe**



Web Applications

Manual Deploying

Tomcat

```
IDE\;C:\Programming\Microsoft Visual Studio 8\Common7\IDE\PrivateAssemblies\;
:\Programming\Microsoft Visual Studio\Common\Tools\WinNT;C:\Programming\Micro
t Visual Studio\Common\MSDev98\Bin;C:\Programming\Microsoft Visual Studio\Com
\Tools;C:\Programming\Microsoft Visual Studio\VC98\bin;C:\Programming\jwsdp\j
p-shared\bin;c:\Programming\Adobe\Flex Builder 3 Plug-in\sdks\2.0.1\bin;c:\Pr
amming\Sun\AppServer5.1\bin\
08-11-2010 23:03:49 org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-8080
08-11-2010 23:03:49 org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 500 ms
08-11-2010 23:03:49 org.apache.catalina.core.StandardService start
INFO: Starting service Catalina
08-11-2010 23:03:49 org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/6.0.10
- Deploying module: addressing-1.2
- Deploying module: soapmonitor-1.2
- Deploying Web service: TemperatureWS.aar
- Deploying Web service: version.aar
08-11-2010 23:03:52 org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
- JK: ajp13 listening on /0.0.0.0:8009
- Jk running ID=0 time=0/32  config=null
08-11-2010 23:03:52 org.apache.catalina.startup.Catalina start
INFO: Server startup in 2313 ms
```

c:\Programming\Tomcat 6.0\webapps*.*

Name	Ext	Size
[..]		<DIR>
[axis]		<DIR>
[axis2]		<DIR>
[basic]		<DIR>
[Day1Servlet0430]		<DIR>
[docs]		<DIR>
[host-manager]		<DIR>
[ROOT]		<DIR>
Day1Servlet0430	war	8.406

[..]		<DIR>
[bin]		<DIR>
[conf]		<DIR>
[data]		<DIR>
[Ex]		<DIR>
[lib]		<DIR>
[logs]		<DIR>
[server]		<DIR>
[temp]		<DIR>
[webapps]		<DIR>
[work]		<DIR>
LICENSE		11.560
tomcat	ico	21.630
Uninstall	exe	66.284

```
INFO: Server startup in 2313 ms
08-11-2010 23:07:12 org.apache.catalina.startup.HostConfig deployWAR
INFO: Deploying web application archive Day1Servlet0430.war
```

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

Web Applications

Web Application Development Process

- **Requirement tools: NetBeans 6.9.1**
- **Step 1:** Creating a Web application project
- **Step 2:** Creating the Servlets
- **Step 3:** Writing the code for Servlet & Compile
- **Step 4:** Package Servlet into WAR file
- **Step 5:** Deploying to a Web Server
- **Step 6:** Executing the application

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

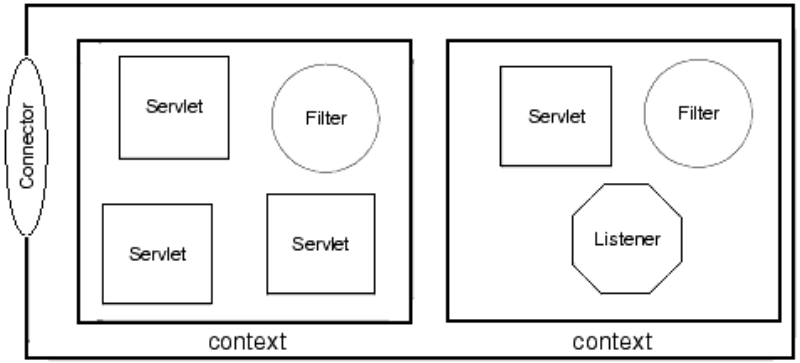
The Web Container Model

The ServletContext

- A servlet container can manage any number of distinct applications.
  - An application consists of any number of servlets, filters, listeners, and static Web pages.
  - A set of components working together is a Web application.
- The container uses a *context* to
  - Group related components. The container loads the objects within a context as a group, and objects within the same context can easily share data.
  - Provide a set of services for the web application to work with the container
- Each context usually corresponds to a distinct Web application.

→ **A servlet context is considered as a memory segment that**

- **Collects all method that is used for particular Web application in server side and they support to interact with Servlet container**
- **Stores some object in server side that all web's component can access**



The Web Container Model

The ServletContext – Example

- For example, the directory structure below describes two contexts, one named orders and one named catalog. The catalog context contains a static HTML page, intro.html.

webapps

\orders

\WEB-INF

web.xml

\catalog

intro.html

\WEB-INF

web.xml

The Web Container Model

The ServletContext – Initialization Parameters

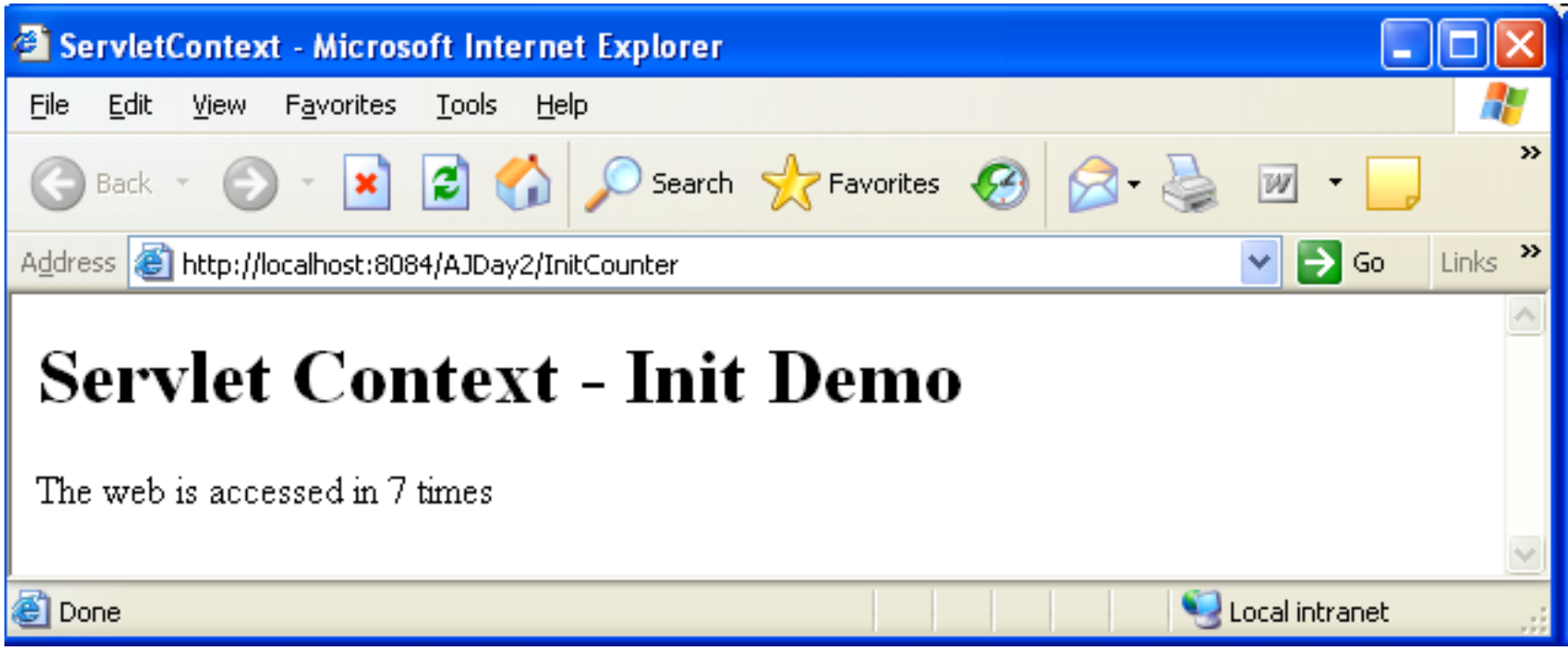
- Providing some fundamental information available to all the dynamic resources (servlets, JSP) within the web application is allowed by
  - Using servlet initialization parameters in the deployment descriptor with the getInitParameter(String parName) method to provide initialization information for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

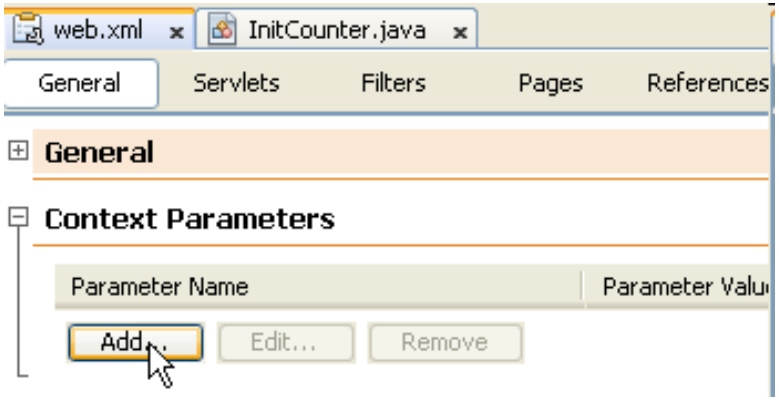


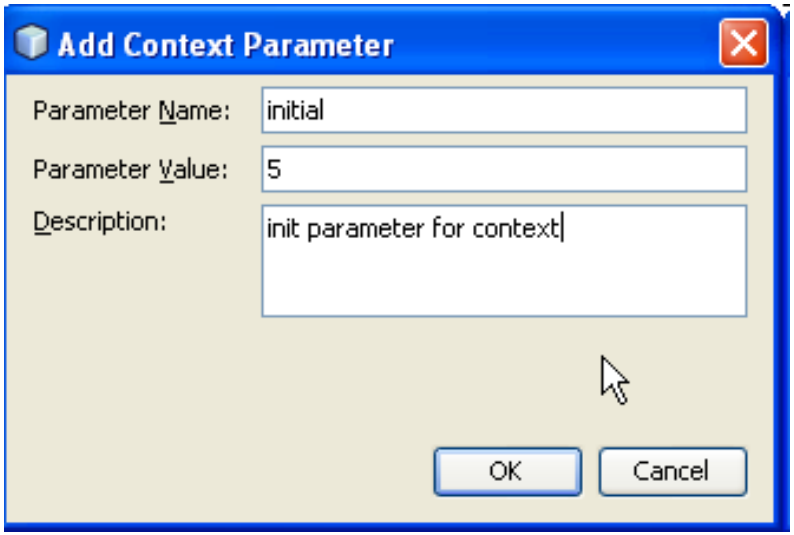
The Web Container Model

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

Context Parameters

Parameter Name	Parameter
initial	5

Add... Edit... Remove

web.xml * | InitCounter.java

General | Servlets | Filters | Pages | References | Security | XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee
    <context-param>
        <description>init parameter for context</description>
        <param-name>initial</param-name>
        <param-value>5</param-value>
    </context-param>
```

```java
int count = 0;

public void init() throws ServletException {
    super.init();
    ServletContext sc = getServletContext();
    String initNo = sc.getInitParameter("initial");
    count = Integer.parseInt(initNo);
}
```

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

```
38      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
39      throws ServletException, IOException {
40          response.setContentType("text/html;charset=UTF-8");
41          PrintWriter out = response.getWriter();
42          try {
43              out.println("<html>");
44              out.println("<head>");
45              out.println("<title>ServletContext</title>");
46              out.println("</head>");
47              out.println("<body>");
48              out.println("<h1>The ServletContext-Init Demo</h1>");
49
50              ServletContext context = getServletContext();
51              String intNo = context.getInitParameter("initial");
52              count = Integer.parseInt(intNo);
53
54              count++;
55              out.println("The web is accessed in " + count + "times");
56
57              out.println("</body>");
58              out.println("</html>");
59          } finally {
60              out.close();
61          }
62      }
```

Address http://localhost:8084/ServletMDL1/InitCounter

The ServletContext-Init Demo

The web is accessed in 6times

The Web Container Model

The ServletContext – **Initialization Parameters**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    <context-param>
        <description>init parameter for context</description>
        <param-name>initial</param-name>
        <param-value>5</param-value>
    </context-param>
    <context-param>
        <description>init parameter for context</description>
        <param-name>initial</param-name>
        <param-value>6</param-value>
```

Output

Apache Tomcat 6.0.26 Log | Apache Tomcat 6.0.26 | ServletMDL1 (run)

```
SEVERE: Parse error in application web.xml file at jndi:/localhost/ServletMDL1/WEB-INF/web.xml
java.lang.IllegalArgumentException: Duplicate context initialization parameter initial
        at org.apache.tomcat.util.digester.Digester.createSAXException(Digester.java:2806)
```

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

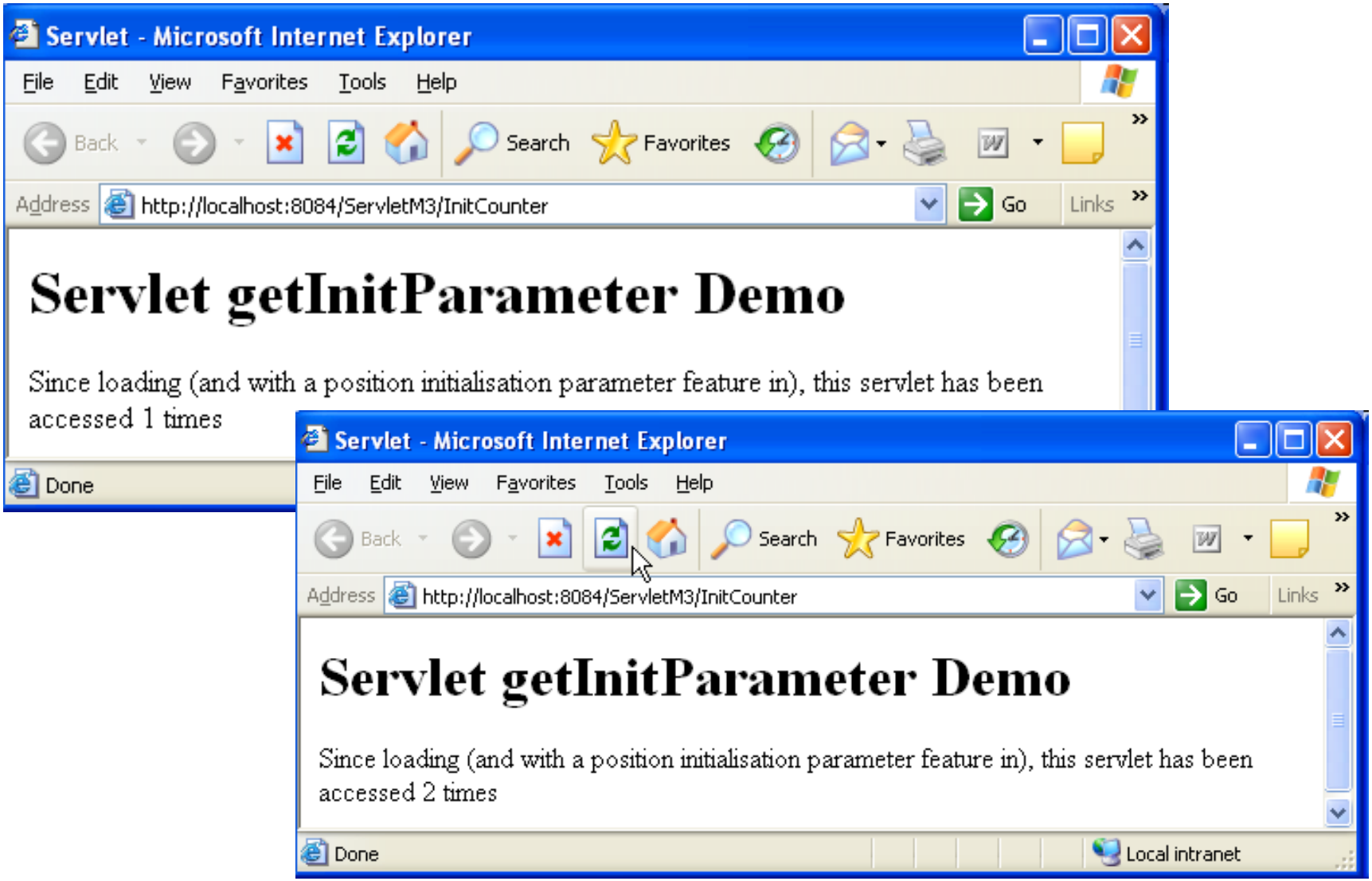
```
<servlet>
    <servlet-name>servletName</servlet-name>
    <servlet-class>servletClass</servlet-class>
      <init-param>
        <param-name>parName</param-name>
        <param-value>parValue</param-value>
      </init-param>
</servlet>
```

- Writing Code to Retrieve ServletConfig Initialization Parameters

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

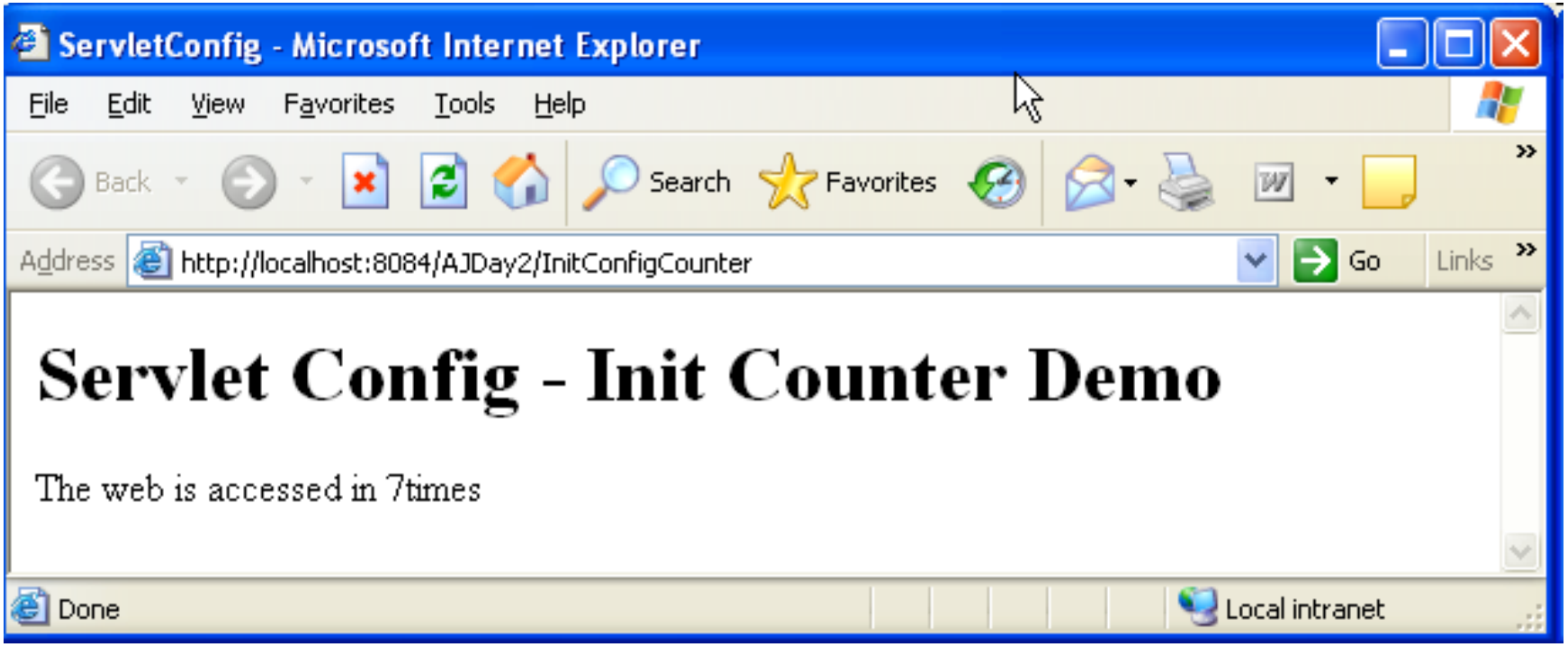
The Web Container Model

The ServletConfig interface – Example



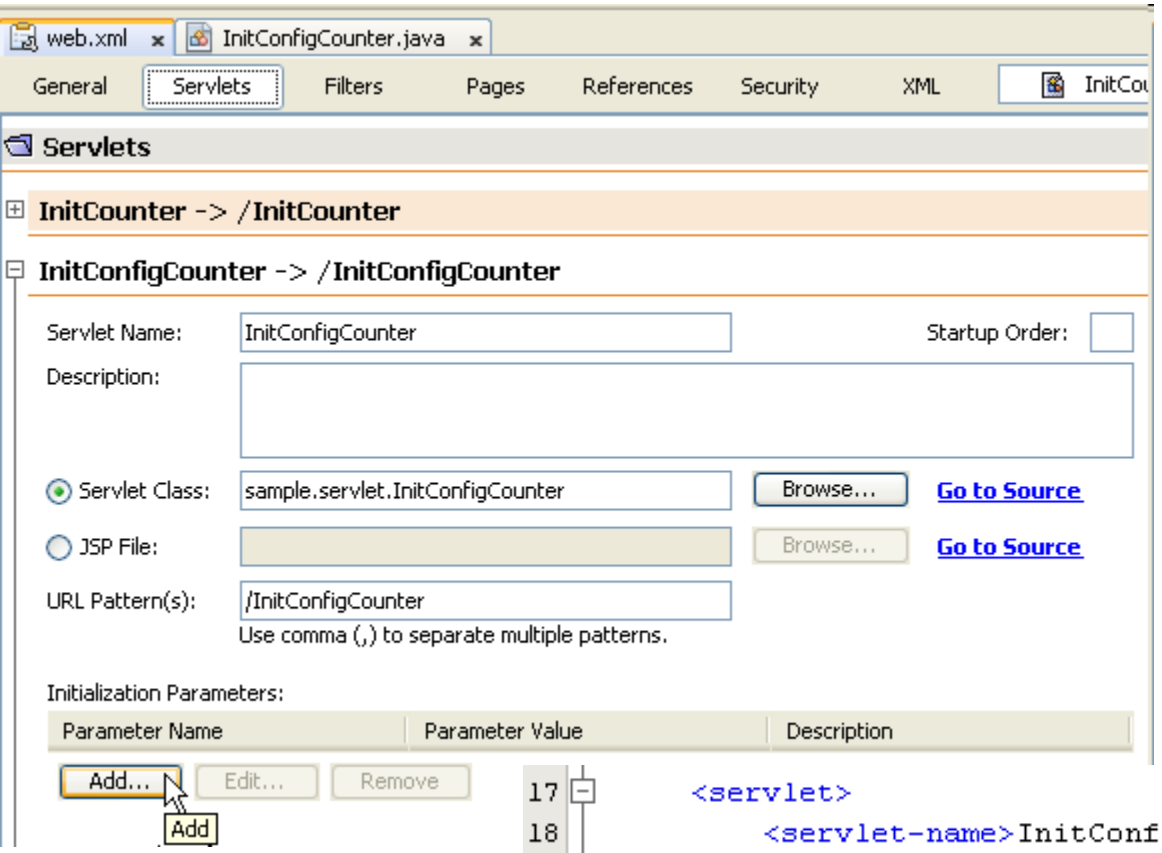
The Web Container Model

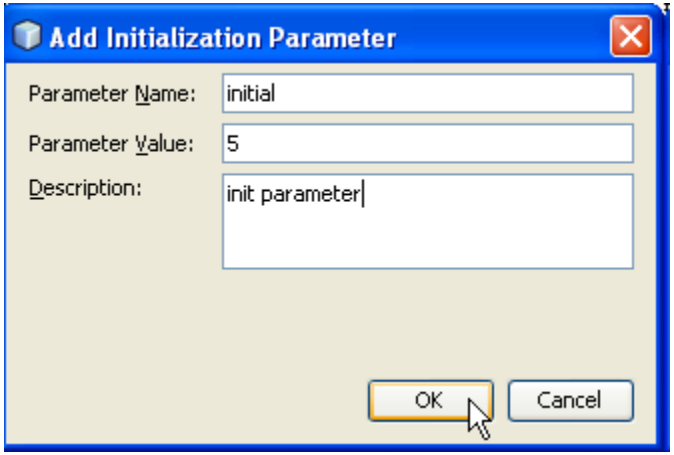
The ServletConfig interface – Example



The Web Container Model

The ServletConfig interface – Example






```
17  <servlet>
18      <servlet-name>InitConfigCounter</servlet-name>
19      <servlet-class>sample.servlet.InitConfigCounter</servlet-class>
20      <init-param>
21          <description>init parameter</description>
22          <param-name>initial</param-name>
23          <param-value>5</param-value>
24      </init-param>
25  </servlet>
```

The Web Container Model

The ServletConfig interface – Example

web.xml | InitConfigCounter.java

```
40      */
41      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
42              throws ServletException, IOException {
43          response.setContentType("text/html;charset=UTF-8");
44          PrintWriter out = response.getWriter();
45          try {
46              out.println("<html>");
47              out.println("<head>");
48              out.println("<title>ServletConfig</title>");
49              out.println("</head>");
50              out.println("<body>");
51              out.println("<h1>Servlet Config - Init Counter Demo</h1>");
52
53              ServletConfig scc = getServletConfig();
54              String tmp = scc.getInitParameter("initial");
55              try {
56                  count = Integer.parseInt(tmp);
57              } catch (NumberFormatException e) {
59                  e.printStackTrace();
                }
60              count++;
61              out.println("The web is accessed in " + count + "times");
62
63              out.println("</body>");
64              out.println("</html>");
65          } finally {
66              out.close();
67          }
68      }
```

Address http://localhost:8084/AJDay2/InitConfigCounter

Servlet Config - Init Counter Demo

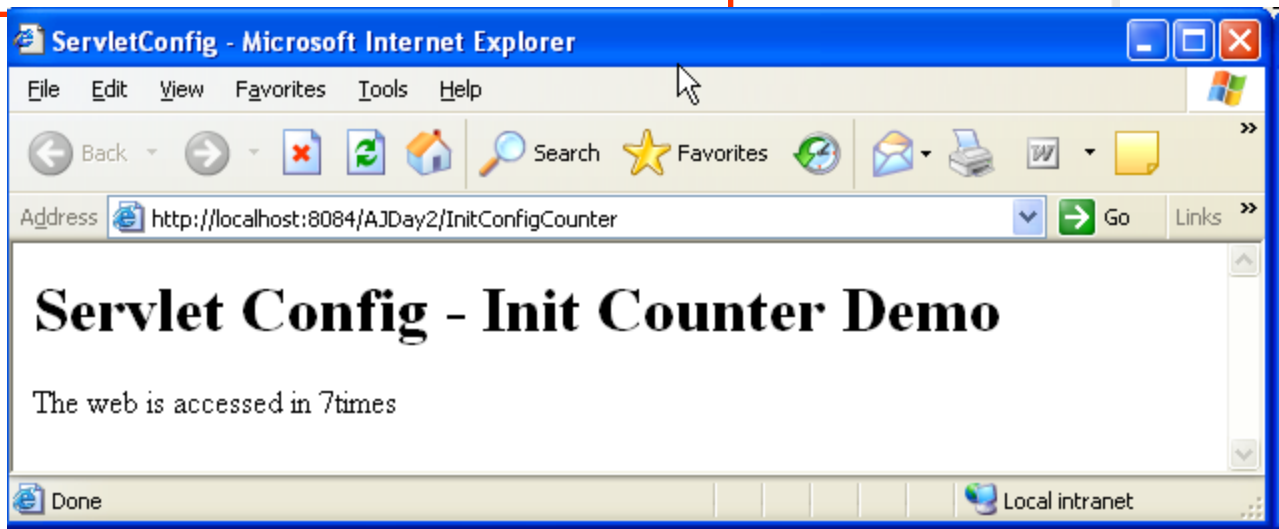
The web is accessed in 6times

Done

The Web Container Model

The ServletConfig interface – Example

```
17      <servlet>
18          <servlet-name>InitConfigCounter</servlet-name>
19          <servlet-class>sample.servlet.InitConfigCounter</servlet-class>
20          <init-param>
21              <description>init parameter</description>
22              <param-name>initial</param-name>
23              <param-value>5</param-value>
24          </init-param>
25          <init-param>
26              <description>init parameter</description>
27              <param-name>initial</param-name>
28              <param-value>6</param-value>
29          </init-param>
30      </servlet>
```



The Web Container Model

Attributes, Scope, and Multithreading

- **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

- **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**
- 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

- Defines **how long** a attribute is available in its scope.
- There are **3 scopes**
  - **Request Scope**
    - **Lasts** from the moment an HTTP **request** hits a servlet in the **web container to** the moment the servlet is done **with delivering** the HTTP **response**.
    - **javax.servlet.ServletRequest**
  - **Session Scope**
    - Session scope comes into play from the point where a **browser window establishes/ open session** contact with the web application up to the point where that **browser window is closed, session is closed, session is time out, server is crashed**.
    - **javax.servlet.http.HttpSession**
    - HttpSession session = request.getSession();
  - **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

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

- **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**

- **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

```
<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

```
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

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

The Web Container Model

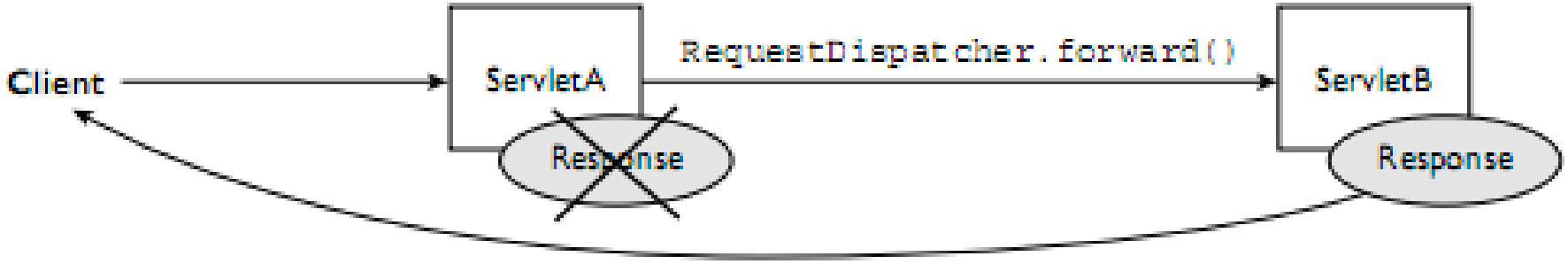
Request Dispatching

- 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

Address http://localhost:8084/AJDay2/requestDispatcher.html

Demo Request Dispatcher

Name RequestDispatcher

Transfer

Address http://localhost:8084/AJDay2/MiddleServlet?txtName=RequestDispatcher

End Servlet

Par: RequestDispatcher - Middle Information

The Web Container Model

Using RequestDispatcher – Example

```html
<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

```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

Change the RequestDispatch – forward method to include method

Address http://localhost:8084/AJDay2/requestDispatcher.html

Demo Request Dispatcher

Name includeRD

Transfer

Address http://localhost:8084/AJDay2/MiddleServlet?txtName=includeRD

Middle Servlet

End Servlet

Par: includeRD - Middle Information

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

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

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

The Web Container Model

Filter

- **Are components that add functionality to the request and response processing of a Web Application**
- Is tool that acts as an interface or a passage between the client and the web application, such as JSP and servlet in the server
- Are basically a set of steps through which request and response must pass for required modifications
- **Supports dynamic modification of requests and responses between client and web applications.**
- Categorized according to the services they provide to the web applications
- Resides in the web container along with the web applications
- **Intercept the requests and response that flow between a client and a Servlet/JSP.**
- **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**
- Was introduced as a Web component in Java servlet specification version 2.3

The Web Container Model

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

The Web Container Model

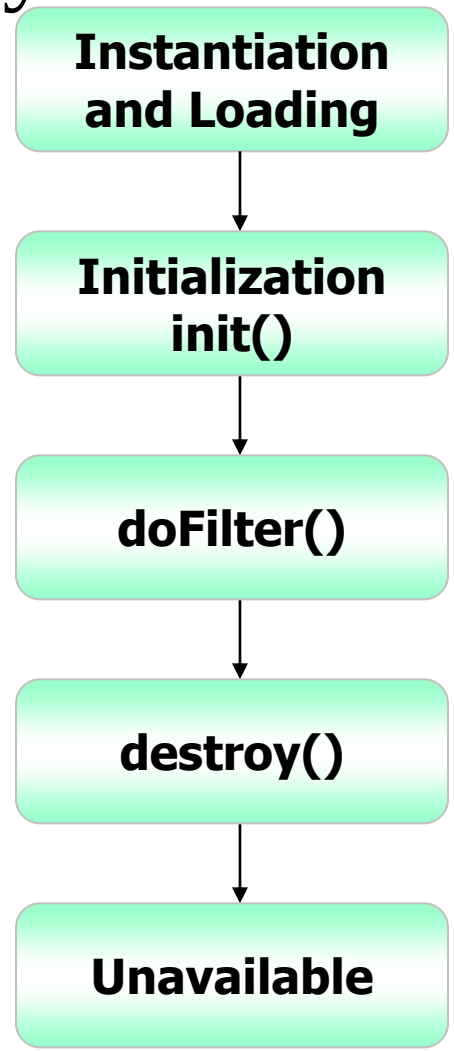
Filter

- **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

The Web Container Model

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



The Web Container Model

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.

The Web Container Model

Filter

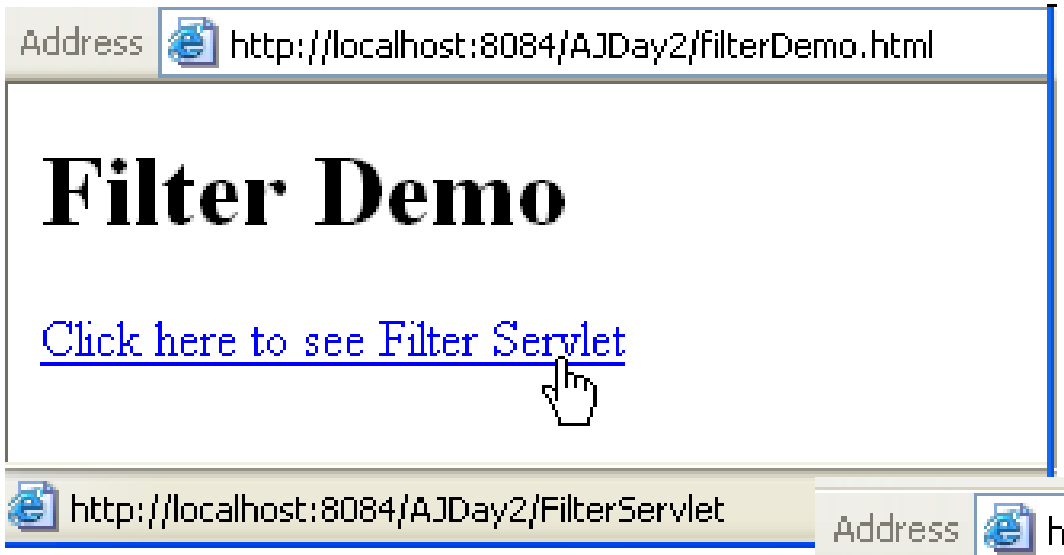
- **In Web Deployment Descriptor**

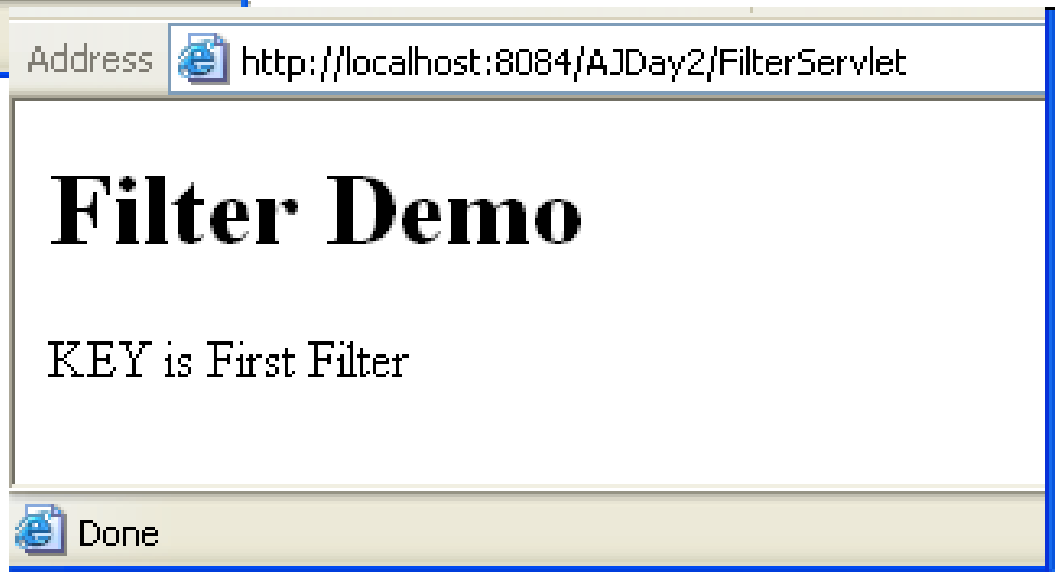
```
<web-app>
….
<filter>
    <filter-name>Name of Filters</filter-name>
    <filter-class>implemented Filter Class</filter-class>
     [<init-param>
          <param-name>parameter name</param-name>
           <param-value>value </param-value>
     </init-param>]
  </filter>
  <filter-mapping>
    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

The Web Container Model

Filter – Example

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





The Web Container Model

Filter – Example

filterDemo.html

Preview

```
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>
```

The Web Container Model

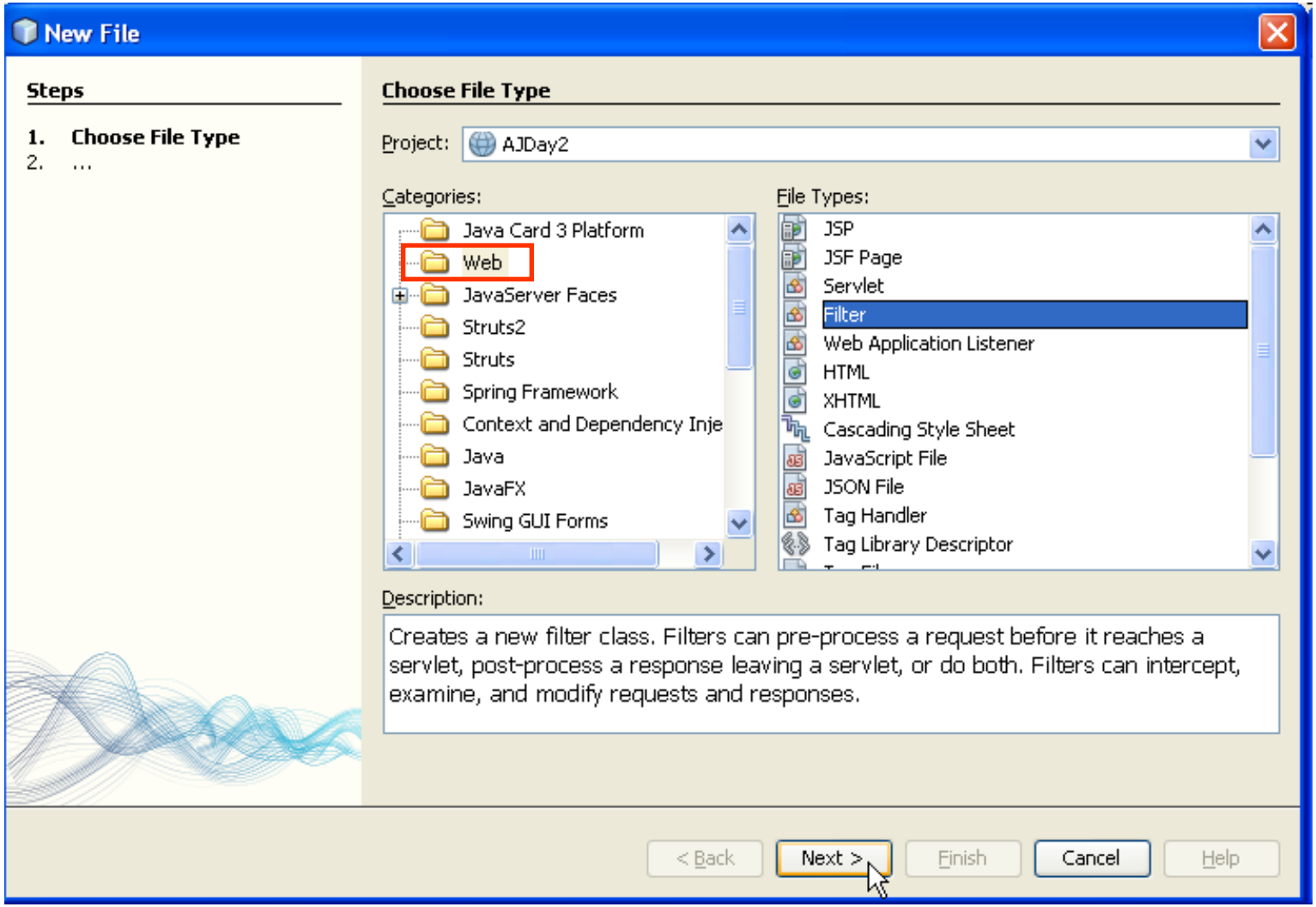
Filter – Example

FilterServlet.java

```java
 * @author Trong Khanh
 */
public class FilterServlet extends HttpServlet {

    /**...*/
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Filter</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Filter Demo</h1>");

            String test = (String)request.getAttribute("KEY");
            out.println("KEY is " + test);

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

The Web Container Model

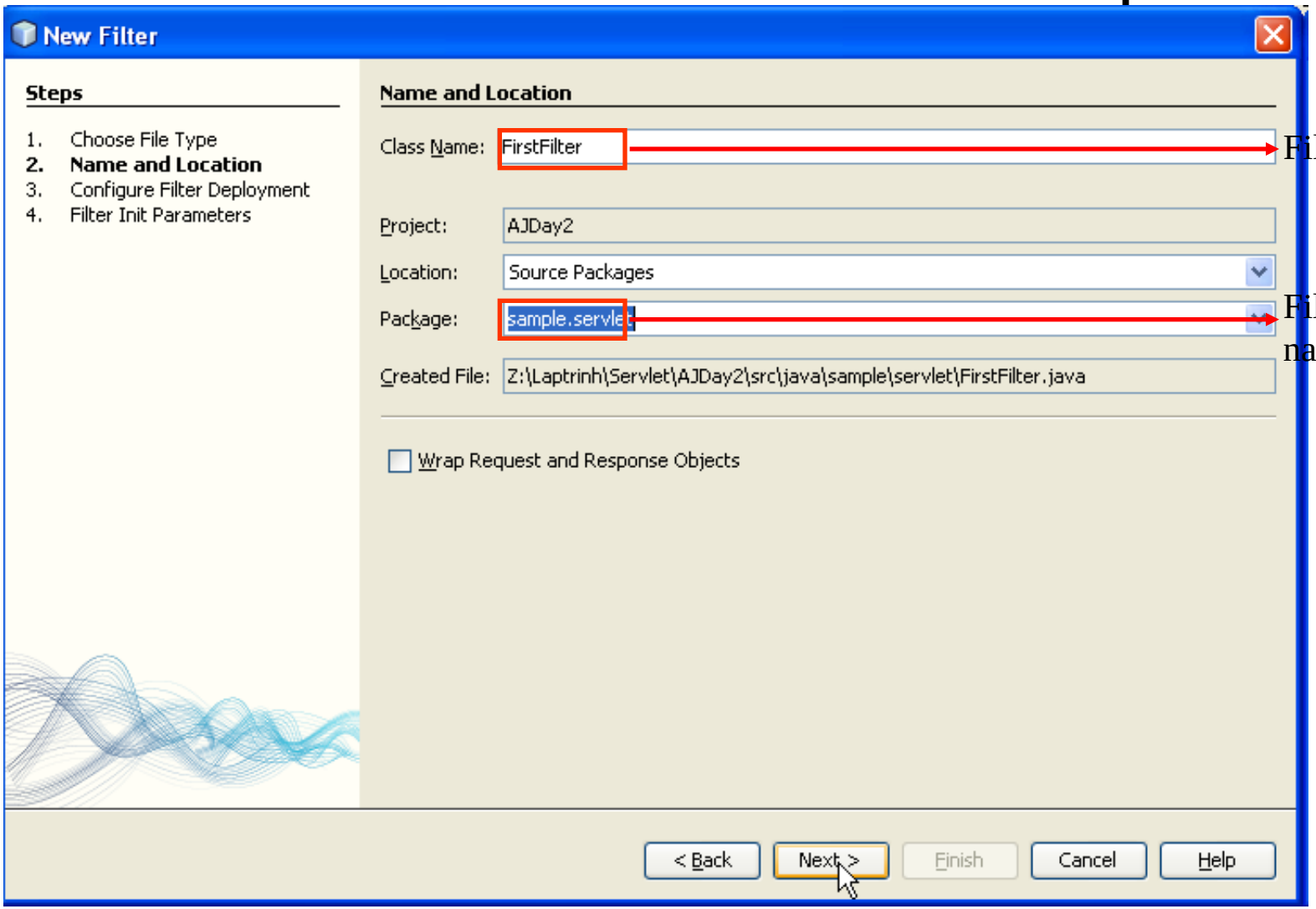
Filter – Example



- Click Next Button

The Web Container Model

Filter – Example



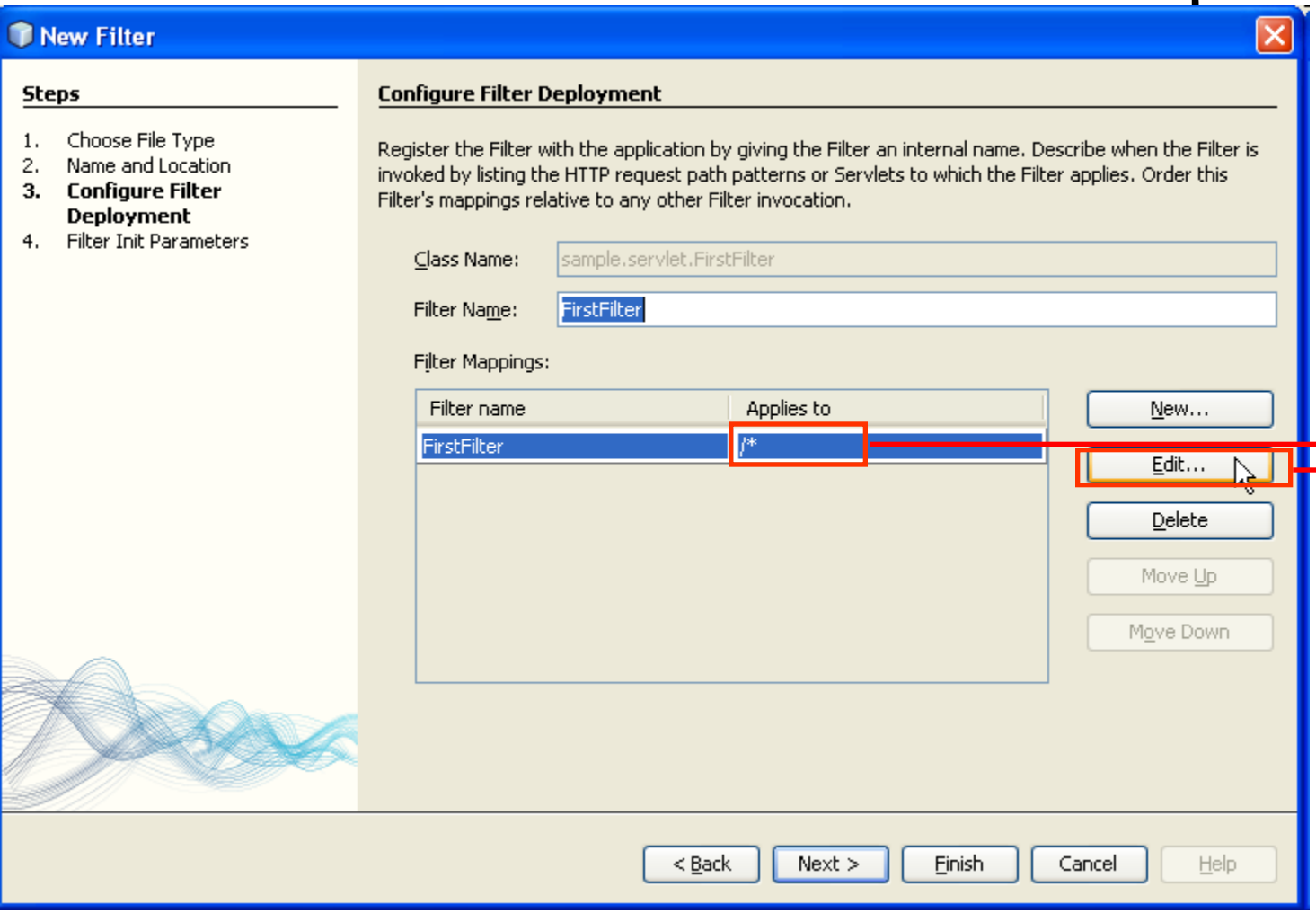
Fill your filter name

Fill/choose package name

- Click Next Button

The Web Container Model

Filter – Example



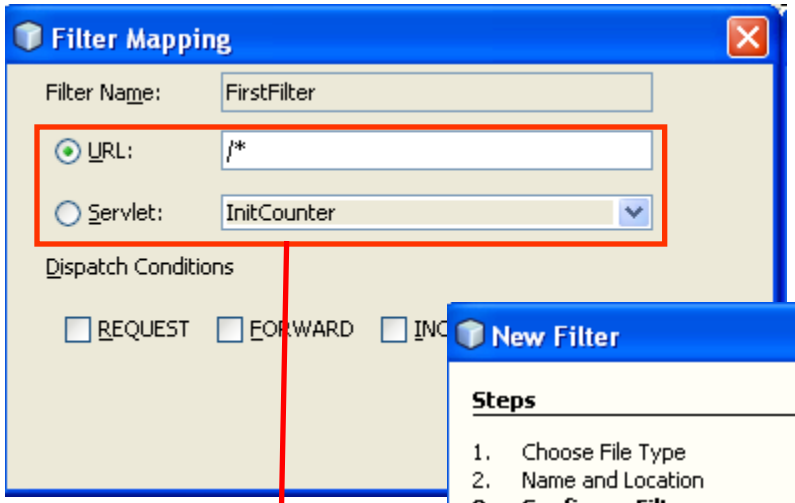
Apply filter

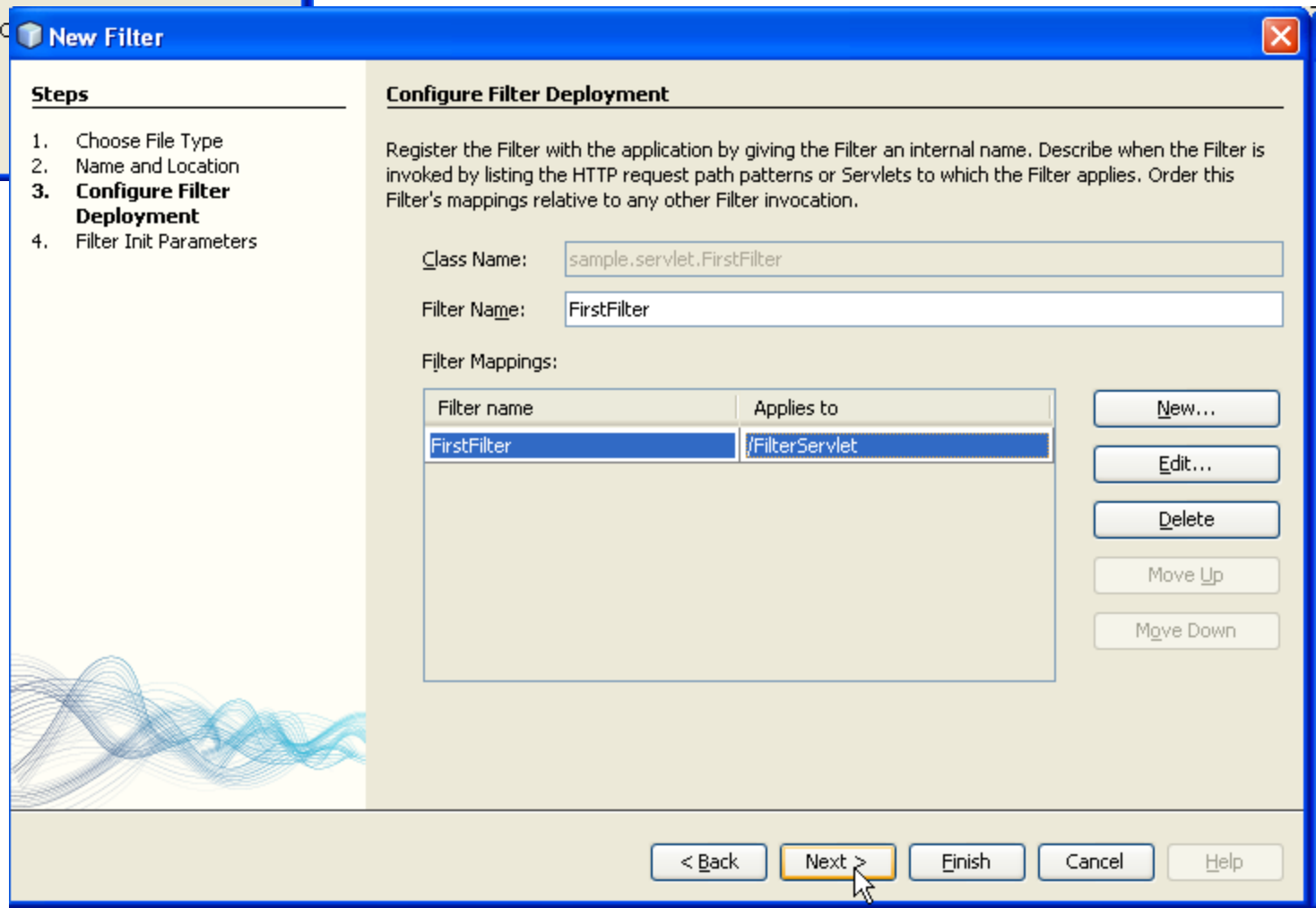
Edit the Apply filter

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

The Web Container Model

Filter – Example





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

The Web Container Model

Filter – Example

FirstFilter.java | web.xml

General | Servlets | Filters | Pages | References | Security | XML

```xml
<filter>
    <filter-name>FirstFilter</filter-name>
    <filter-class>sample.servlet.FirstFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>FirstFilter</filter-name>
    <url-pattern>/FilterServlet</url-pattern>
</filter-mapping>
<servlet>
```

FirstFilter.java | web.xml

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

The Web Container Model

Filter – Example

Output

Apache Tomcat 6.0.26 Log | Apache Tomcat 6.0.26

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

Output

Apache Tomcat 6.0.26 Log | Apache Tomcat 6.0.26 | AJDay2 (run)

```
21-06-2011 20:09:31 org.apache.catalina.core.ApplicationContext log
INFO: FirstFilter:Initializing filter
```

FirstFilter.java | web.xml

General | Servlets | Filters | Pages | References | Security | XML

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

The Web Container Model

Filter – Example

Address http://localhost:8084/AJDay2/filterDemo.html

Filter Demo

Click here to see Filter Servlet

Output

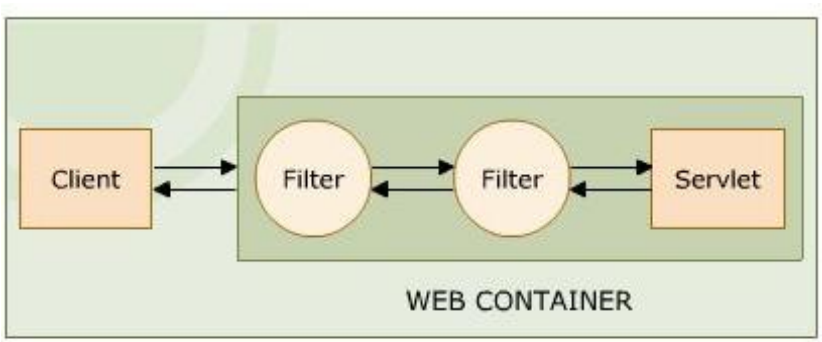
Apache Tomcat 6.0.26 Log | Apache Tomcat 6.0.26

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

The Web Container Model

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

The Web Container Model

Filter Chain – Example

The Web Container Model

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();
    }
}
```

The Web Container Model

Filter Chain – Example

web.xml

General | Servlets | Filters | Pages | References | Security | XML

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

The Web Container Model

Filter Chain – Example

web.xml

General | Servlets | Filters | Pages | References | Security | XML

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:
3      <filter>
4          <filter-name>SecondFilter</filter-name>
5          <filter-class>sample.servlet.SecondFilter</filter-class>
6      </filter>
7      <filter>
8          <filter-name>FirstFilter</filter-name>
9          <filter-class>sample.servlet.FirstFilter</filter-class>
10     </filter>
11     <filter-mapping>
12         <filter-name>SecondFilter</filter-name>
13         <url-pattern>/FilterServlet</url-pattern>
14     </filter-mapping>
15     <filter-mapping>
16         <filter-name>FirstFilter</filter-name>
17         <url-pattern>/FilterServlet</url-pattern>
18     </filter-mapping>
19     <servlet>
20         <servlet-name>SessionServlet</servlet-name>
21         <servlet-class>sample.servlet.SessionServlet</servlet-class>
22     </servlet>
23     <servlet>
24         <servlet-name>FilterServlet</servlet-name>
25         <servlet-class>sample.servlet.FilterServlet</servlet-class>
26     </servlet>
```

The Web Container Model

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 {
```

The Web Container Model

Filter Chain – Example

Output

Apache Tomcat 6.0.26 Log | Apache Tomcat 6.0.26 | AJDay2 (run)

```
21-06-2011 20:24:24 org.apache.catalina.core.ApplicationContext log
INFO: FirstFilter:Initializing filter
21-06-2011 20:24:24 org.apache.catalina.core.ApplicationContext log
INFO: SecondFilter:Initializing filter
```

Output

Apache Tomcat 6.0.26 Log | Apache Tomcat 6.0.26 | AJDay2 (run)

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

The Web Container Model

Filter Chain – Example – **Change pos**

web.xml | FilterServlet.java

General | Servlets | Filters | Pages | References | Security | XML

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

The Web Container Model

Filter Chain – Example

Output

Apache Tomcat 6.0.26 Log | Apache Tomcat 6.0.26 | AJDay2 (run)

```
21-06-2011 20:29:01 org.apache.catalina.core.ApplicationContext log
INFO: FirstFilter:Initializing filter
21-06-2011 20:29:01 org.apache.catalina.core.ApplicationContext log
INFO: SecondFilter:Initializing filter
```

Output

Apache Tomcat 6.0.26 Log | Apache Tomcat 6.0.26 | AJDay2 (run)

```
INFO: Container org.apache.catalina.core.ContainerBase.[Catalina].[localhos
Passed First Filter - request
Pass Second Filter - request
Pass Second Filter - response
Passed First Filter - response
```

The Web Container Model

Why need a Wrapper Class

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");
        PrintWriter out = response.getWriter();
        out.println("<br/>The slide is licensed to KhanhKT");
    } catch (Throwable t) {
        t.printStackTrace();
    }
}
```

The Web Container Model

Why need a Wrapper Class

Address http://localhost:8084/AJDay2/filterDemo.html

Filter Demo

Click here to see Filter Servlet

http://localhost:8084/AJDay2/FilterServlet

Address http://localhost:8084/AJDay2/FilterServlet

Filter Demo

KEY is First Filter KEY is Second Filter

The Web Container Model

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.getAt
41              out.println("KEY is " + test);
42
43              String test1 = (String) request.get
44              out.println("KEY is " + test1);
45
46              out.println("</body>");
47              out.println("</html>");
48          } finally {
49              //out.close();
50          }
51      }
```

Address http://localhost:8084/AJDay2/FilterServlet

Filter Demo

KEY is First Filter KEY is Second Filter

The slide is licensed to KhanhKT

Done

The Web Container Model

Wrapper Class

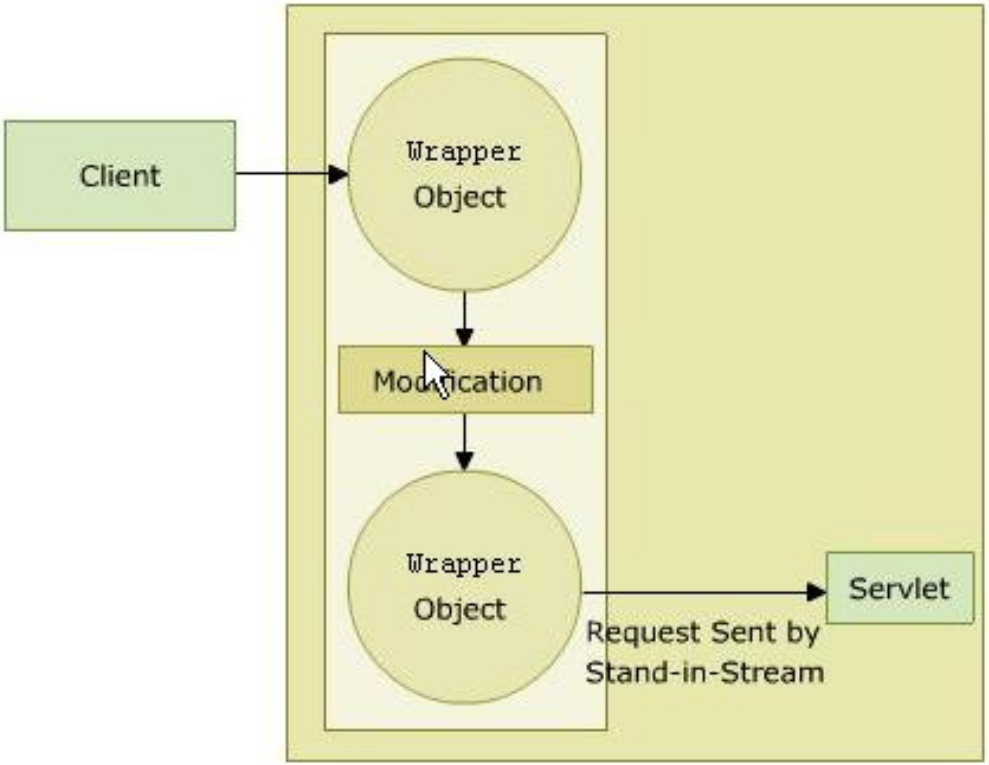
- To modify or intercept the request or response before they can reach their logical destination, the required object can dynamically capture the request or response
- Wrapper class
  - Creates the object to capture the request and response before they reach server and client respectively
  - The wrapper 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.

The Web Container Model

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.

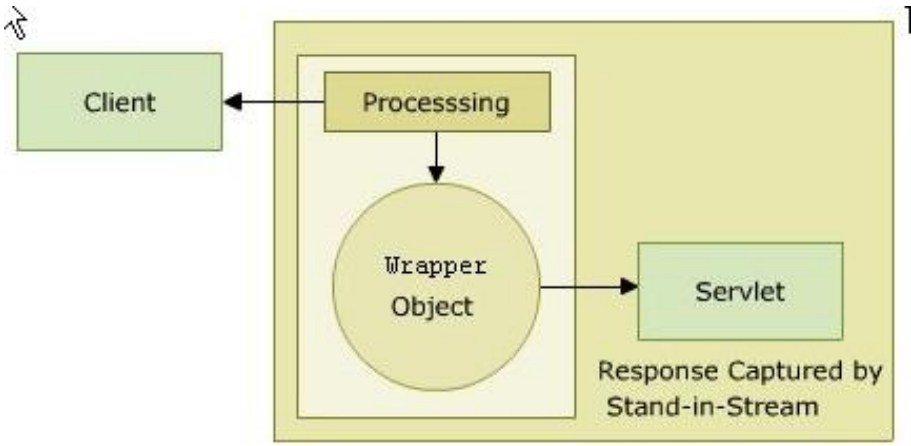


Modifying the Request

The Web Container Model

Wrapper Class – Altering Response

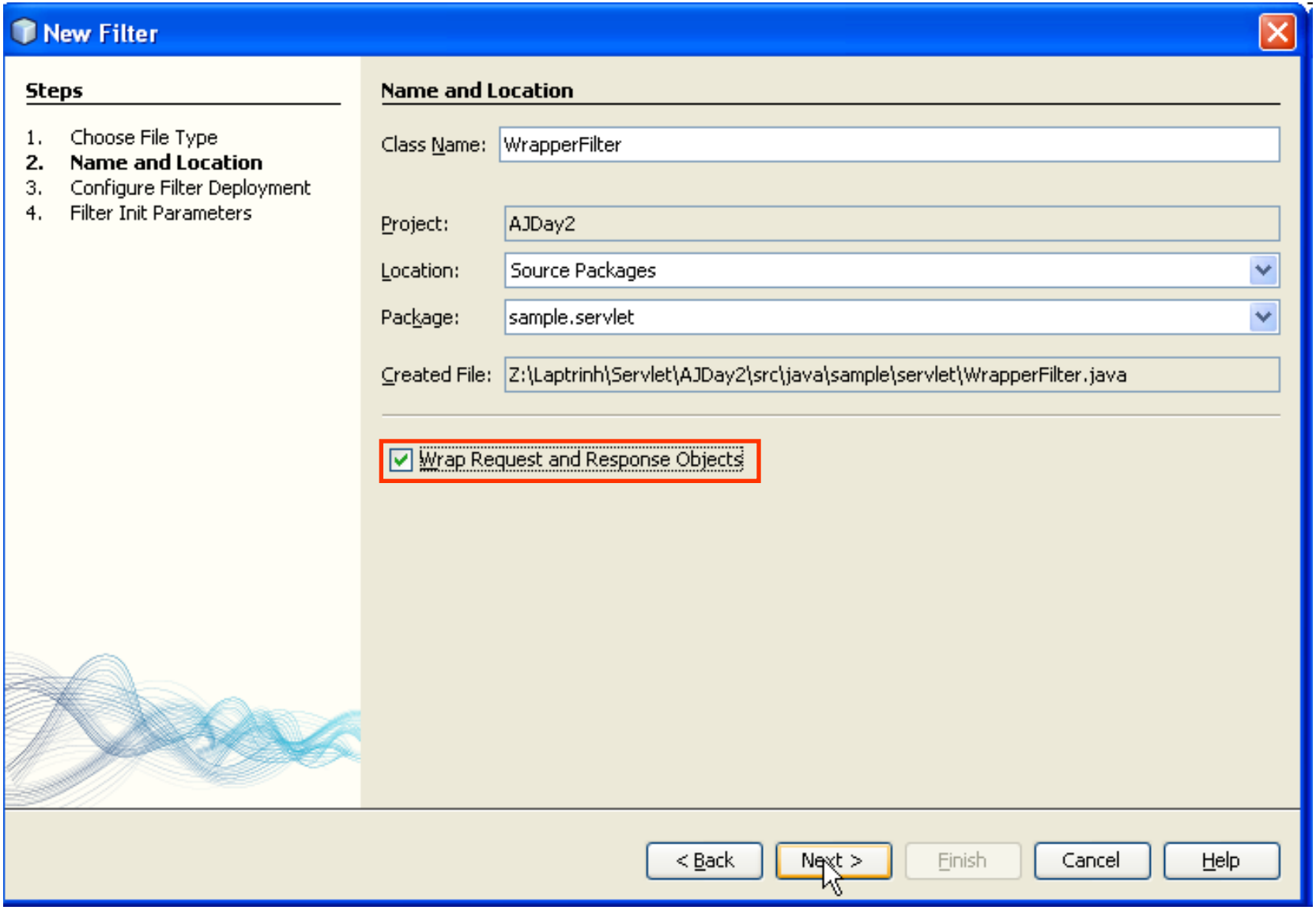
- Create filter class extends to the **ServletResponseWrapper** or **HttpServletResponseWrapper** class.
- The object captures the httpRequest object from the client and sends it to the filers
- Through the objects filter extends some services to the request.



Modifying the Response

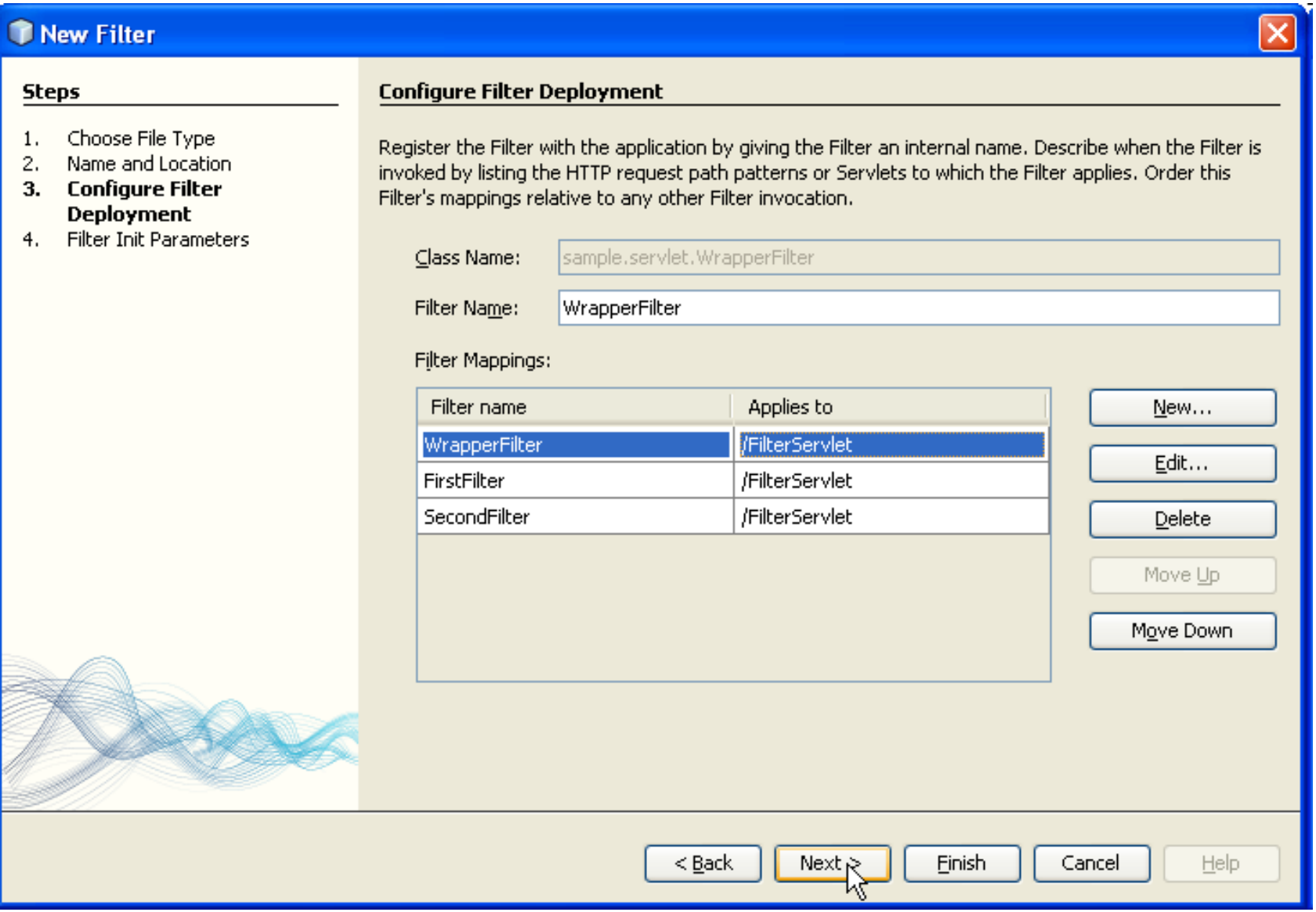
The Web Container Model

Wrapper Class – Example



The Web Container Model

Wrapper Class – Example



The Web Container Model

Wrapper Class – Example

- Adding the MyPrinter class extends PrintWriter in FilterWrapper class

WrapperFilter.java

```
363         class MyPrinter extends PrintWriter {
364             public MyPrinter (Writer out) {
365                 super(out);
366             }
367
                public void close() {
369                 super.close();
370             }
371         }
372     }
```

The Web Container Model

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         }
```

The Web Container Model

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         }
```

Summary

- **Web Applications**
- **The Web Contain Model**

Q&A

Next Lecture

- **Sessions in Web Application**
  - Mechanism
  - 4 Techniques
- **Errors Handling in Servlets**
  - Reporting
  - Logging