



SERVLETS: Dispatching, monitoring, filtering

Dispatching

```
RequestDispatcher dispatch =  
    cntx.getRequestDispatcher("/SecondServlet");  
dispatch.forward(req,res);
```

```
RequestDispatcher dispatch =  
    cntx.getRequestDispatcher("/SecondServlet");  
dispatch.include(req,res);
```

Dispatching example

```
package servlets;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.RequestDispatcher;

public class SecondServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,HttpServletResponse res)
        throws IOException,ServletException {
        PrintWriter out=res.getWriter();
        System.out.println("Second Servlet Called");
    }
}
```

Dispatching example

```
package servlets;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.ServletContext;
import javax.servlet.RequestDispatcher;

public class FirstServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,HttpServletResponse res)
        throws IOException,ServletException {
        PrintWriter out=res.getWriter();
        out.println("First Servlet Called");
        ServletConfig config = getServletConfig();
        ServletContext cntx = config.getServletContext();
        RequestDispatcher dispatch =
            cntx.getRequestDispatcher("/SecondServlet");
        dispatch.forward(req,res);
    }
}
```

Dispatching example

```

<servlet>
  <servlet-name>FirstServlet</servlet-name>
  <servlet-class>servlets.FirstServlet</servlet-class>
</servlet>

<servlet>
  <servlet-name>SecondServlet</servlet-name>
  <servlet-class>servlets.SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>FirstServlet</servlet-name>
  <url-pattern>/firstservlet/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>SecondServlet</servlet-name>
  <url-pattern>/SecondServlet/*</url-pattern>
</servlet-mapping>

```

Monitoring Servlets Lifecycle

Web context	Initialization and Destruction	ServletContextListener	ServletContextEvent
	Attribute added, removed, or replaced	ServletContextAttributeListener	ServletContextAttributeEvent
Session	Creation, invalidation, activation, passivation, and timeout	HttpSessionListenerHttpSessionActivationListener	HttpSessionEvent
	Attribute added, removed, or replaced	HttpSessionAttributeListener	HttpSessionBindingEvent
Request	A servlet request has started being processed by Web components	ServletRequestListener	ServletRequestEvent
	Attribute added, removed, or replaced	ServletRequestAttributeListener	ServletRequestAttributeEvent

Monitoring Servlets Lifecycle - Example

```
/* File : ApplicationWatch.java */
import javax.servlet.ServletContextListener;
import javax.servlet.ServletContextEvent;
public class ApplicationWatch implements
    ServletContextListener {
    public static long applicationInitialized = 0L;
    /* Application Startup Event */
    public void contextInitialized(ServletContextEvent ce) {
        applicationInitialized = System.currentTimeMillis(); }
    /* Application Shutdown Event */
    public void contextDestroyed(ServletContextEvent ce) {}
}
```

Monitoring Servlets Lifecycle - Example

```
/* File : SessionCounter.java */
import javax.servlet.http.HttpSessionListener;
import javax.servlet.http.HttpSessionEvent;
public class SessionCounter implements HttpSessionListener
{
    private static int activeSessions = 0;
    /* Session Creation Event */
    public void sessionCreated(HttpSessionEvent se) {
        activeSessions++; }
    /* Session Invalidation Event */
    public void sessionDestroyed(HttpSessionEvent se) {
        if(activeSessions > 0) activeSessions--; }
    public static int getActiveSessions() { return activeSessions;
    }
}
```

Monitoring Servlets Lifecycle - Example

```
<!-- Web.xml -->
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD
Web Application 2.3//EN"
"http://java.sun.com/j2ee/dtds/web-app_2.3.dtd">
<web-app>
<!-- Listeners -->
<listener>
  <listener-class> com.stardeveloper.web.listener.SessionCounter
</listener-class>
</listener>
<listener>
  <listener-class>
com.stardeveloper.web.listener.ApplicationWatch </listener-
class>
</listener>
</web-app>
```

Scope Objects

Web context	ServletContext	Web components within web context <code>servlet.getServletConfig().getServletCon text</code>
Session	HttpSession	Web components handling requests that belong to a session
Request	ServletRequest	Web component handling the request
Page	PageContext	Web component in the JSP page

Main Methods:
Object `getAttribute(String name)`
void `setAttribute(String name, Object o)`
Enumeration `getAttributeNames()`

AOP

The programming paradigms of **aspect-oriented programming** (AOP), and **aspect-oriented software development** (AOSD) attempt to aid programmers in the separation of concerns, specifically cross-cutting concerns, as an advance in modularization.

Logging and authorization offer two examples of crosscutting concerns:

a logging strategy necessarily affects every single logged part of the system. Logging thereby crosscuts all logged classes and methods.

Same is true for authorization.

Filters (javax.servlet.filter)

Other classes that preprocess/postprocess request/response

A filter is an object that performs filtering tasks on either the request to a resource (a servlet or static content), or on the response from a resource, or both.

Filters perform filtering in the doFilter method. Every Filter has access to a FilterConfig object from which it can obtain its initialization parameters, a reference to the ServletContext which it can use, for example, to load resources needed for filtering tasks.

Filters are configured in the deployment descriptor of a web application

Examples that have been identified for this design are

- 1) Authentication Filters
- 2) Logging and Auditing Filters
- 3) Image conversion Filters
- 4) Data compression Filters
- 5) Encryption Filters
- 6) Tokenizing Filters
- 7) Filters that trigger resource access events
- 8) XSL/T filters
- 9) Mime-type chain Filter

<http://java.sun.com/products/servlet/Filters.html>

Filters

Filters are important for a number of reasons. First, they provide the ability to encapsulate recurring tasks in reusable units.

Second, filters can be used to transform the response from a servlet or a JSP page. A common task for the web application is to format data sent back to the client. Increasingly the clients require formats (for example, WML) other than just HTML.

Filters

Filters can perform many different types of functions.

- * Authentication-Blocking requests based on user identity.
- * Logging and auditing-Tracking users of a web application.
- * Image conversion-Scaling maps, and so on.
- * Data compression-Making downloads smaller.
- * Localization-Targeting the request and response to a particular locale.
- * XSL/T transformations of XML content-Targeting web application responses to more than one type of client.

There are many more, such as encryption, tokenizing, triggering resource access events, mime-type chaining, and caching.