Listener and Filters

LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Listener
- Filters

Introduction to Listeners

- Various events occur during the lifecycle of a web application
  - Context is created and destroyed (ServletContextEvent)
  - Session is created and destroyed (HttpSessionEvent)
  - Request is created and destroyed (ServletRequestEvent)
  - Attributes are added and removed from context, request and session objects
  - Session is activated and passivated

- listener
  - Is a class which listens to the various events generated in a web application
  - Provides methods that are executed when the events occur

3

Listener Implementation

- A listener class can be created
  - By implementing one of the listener interfaces provided in javax.servlet and javax.servlet.http package
  - Overriding the methods in the listener interface

- javax.servlet.ServletContextListener
  - interface which provides methods for responding to lifecycle events of a Web Application
  - Contains two methods
    - contextDestroyed(ServletContextEvent sce)

      Receives notification that the ServletContext is about to be shut down
    - contextInitialized(ServletContextEvent sce)

      Receives notification that the web application initialization process is starting

4

Creating Listener class

```
public class MyWebAppListener implements ServletContextListener{

    @Override
    public void contextInitialized(ServletContextEvent event) {
        ServletContext ctx = event.getServletContext();
        Calendar c1 = Calendar.getInstance();
        int curYear = c1.get(Calendar.YEAR);
        ctx.setAttribute("currentYear", curYear);
    }

    @Override
    public void contextDestroyed(ServletContextEvent event) {
        System.out.println("WebApp context has been destroyed");
    }
}
```

Registering Listener class

- Listener class has to be registered with the web application to respond to the events
- Listener classes are registered in web.xml as given below

```
<listener>
    <listener-class>com.manipal.MyWebAppListener</listener-class>
</listener>
```

- Listener classes can also be registered using annotation

```
@WebListener
public class MyWebAppListener implements ServletContextListener{
```

Listener Interfaces

- ServletRequestListener
  - Receives notification when a ServletRequest is about to come/go into the scope of web Application
    - requestDestroyed(ServletRequestEvent sre) requestInitialized(ServletRequestEvent sre)

- ServletRequestAttributeListener
  - Receives notification that an attribute has been added/removed/replaced from ServletRequest
    - attributeAdded(ServletRequestAttributeEvent srae)
    - attributeRemoved(ServletRequestAttributeEvent srae)
    - attributeReplaced(ServletRequestAttributeEvent srae)

Listener Interfaces

- HttpSessionListener
  - Receives notification when a session is created or destroyed
    - sessionCreated(HttpSessionEvent se)
    - sessionDestroyed(HttpSessionEvent se)

    Find number of active sessions in a web application?

- HttpSessionAttributeListener
  - Receives notification when an attribute is added/removed/replaced from HttpSession
    - attributeAdded(HttpSessionBindingEvent event)
    - attributeRemoved(HttpSessionBindingEvent event
    - attributeReplaced(HttpSessionBindingEvent event)

Listener Interfaces

- HttpSessionActivationListener
  - container that migrates session between VMs is required to notify all attributes bound to sessions implementing HttpSessionActivationListener.
    - sessionDidActivate(HttpSessionEvent se)
    - sessionWillPassivate(HttpSessionEvent se)

- HttpSessionBindingListener
  - Causes an object to be notified when it is bound to or unbound from a session
    - valueBound(HttpSessionBindingEvent event)
    - valueUnbound(HttpSessionBindingEvent event)

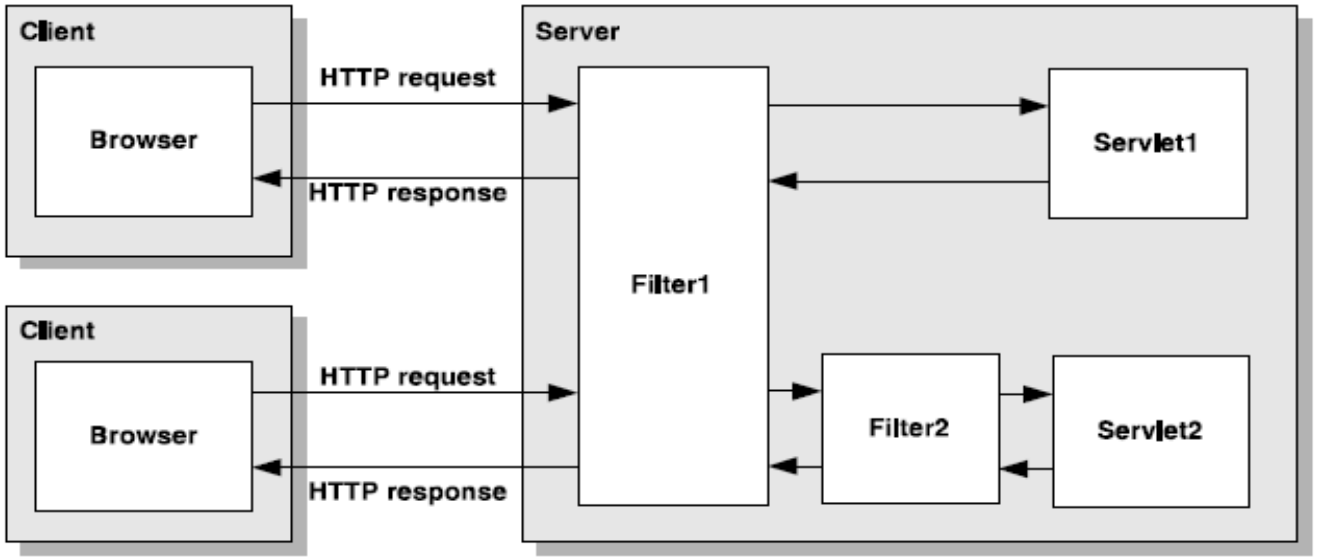
What is Filter

- Filters are used to intercept and modify HTTP request and responses
  - Intercept request and do some processing before requested servlet or JSP is executed
  - Intercept response to do some processing before response is returned to browser
- Ideal for handling cross cutting concerns i.e. aspects that cut across different parts of the application
- Types of tasks that can be performed by filters
  - Logging
    - Log the requests and responses
  - Authentication
    - Allow only authorized used to certain parts of application
  - Compression
    - Compress responses to improve performance

How Filters Work

- Filter1 is mapped to Servlet1
  - Filter1 can execute some code before and after the servlet code is executed
- Filter1 and Filter2 are mapped to Servlet2 (Filter Chaining)

- Mapping of Filters to Servlet is done in web.xml.
- Execution of Filters can be turned on or off through configuration in web.xml
- Filters can be added or removed without impacting the servlets behaviour



Creating filters

- A filter class can be created by implementing the Filter Interface

- Filter interface has the following methods
  - init(FilterConfig filterConfig)
    - Called by the web container once to initialize the filter
  - doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
    - called by the container each time a request/response pair is passed through the chain
  - destroy()
    - Called by the web container to remove the filter object

Creating filters

```
public class LogFilter implements Filter {
  private FilterConfig config = null;

    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        System.out.println("Logging the request details");
        chain.doFilter(request, response);
        System.out.println("Logging the response details");
    }

    public void init(FilterConfig fConfig) throws ServletException {
        this.config = fConfig;
    }

    public void destroy() {
        config = null;
    }
}
```

Configuring Filters

```
<filter>
  <filter-name>first</filter-name>
  <filter-class>filters.LogFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>first</filter-name>
  <url-pattern>/Servlet1</url-pattern>
</filter-mapping>
```

OR

```
<filter>
  <filter-name>first</filter-name>
  <filter-class>filters.LogFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>first</filter-name>
  <servlet-name>filters.Servlet1</servlet-name>
</filter-mapping>
```


14

<dispatcher> Element

<dispatcher> is a sub tag of <filter-mapping>

INCLUDE: Use this for the filter to be applied to any include targets matching a specified servlet

FORWARD: Use this for the filter to be applied to any forward targets matching a specified servlet

REQUEST: Use this for the filter to be applied to all request targets matching a specified servlet

ERROR: Use this for the filter to be applied under the error page mechanism.

SUMMARY

Servlets

SUMMARY

In this lesson, you've learned to:

- Listener
- Filters