Web Component Development with Servlet & JSP Technologies (EE 6)

Module-9: More Controller Facilities

Team Emertxe

ORACLE
Certified Expert
Java EE 6 Web Component Developer

EMERTXE

Objectives

Upon completion of this module, you should be able to:

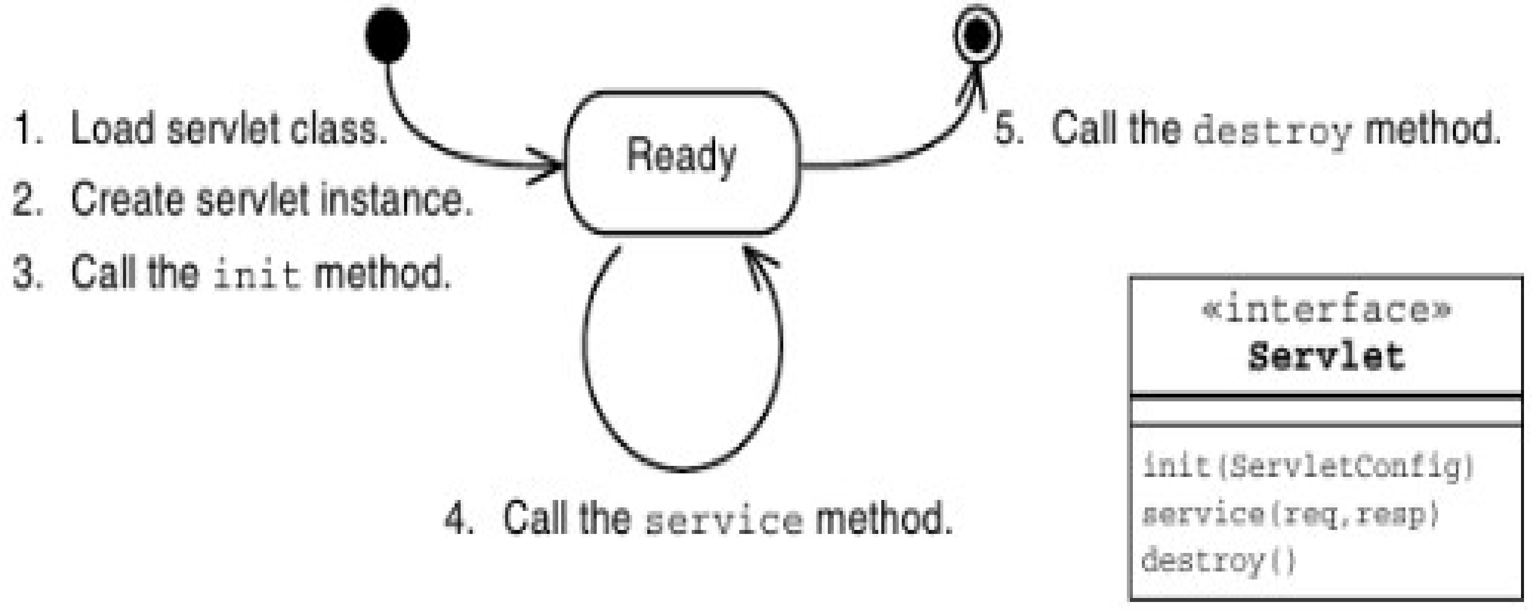
- Understand the lifecycle of a servlet
- Understand the threading model of a servlet
- Write filters and apply them to groups of servlets or JSPs handle multipart form data

Relevance

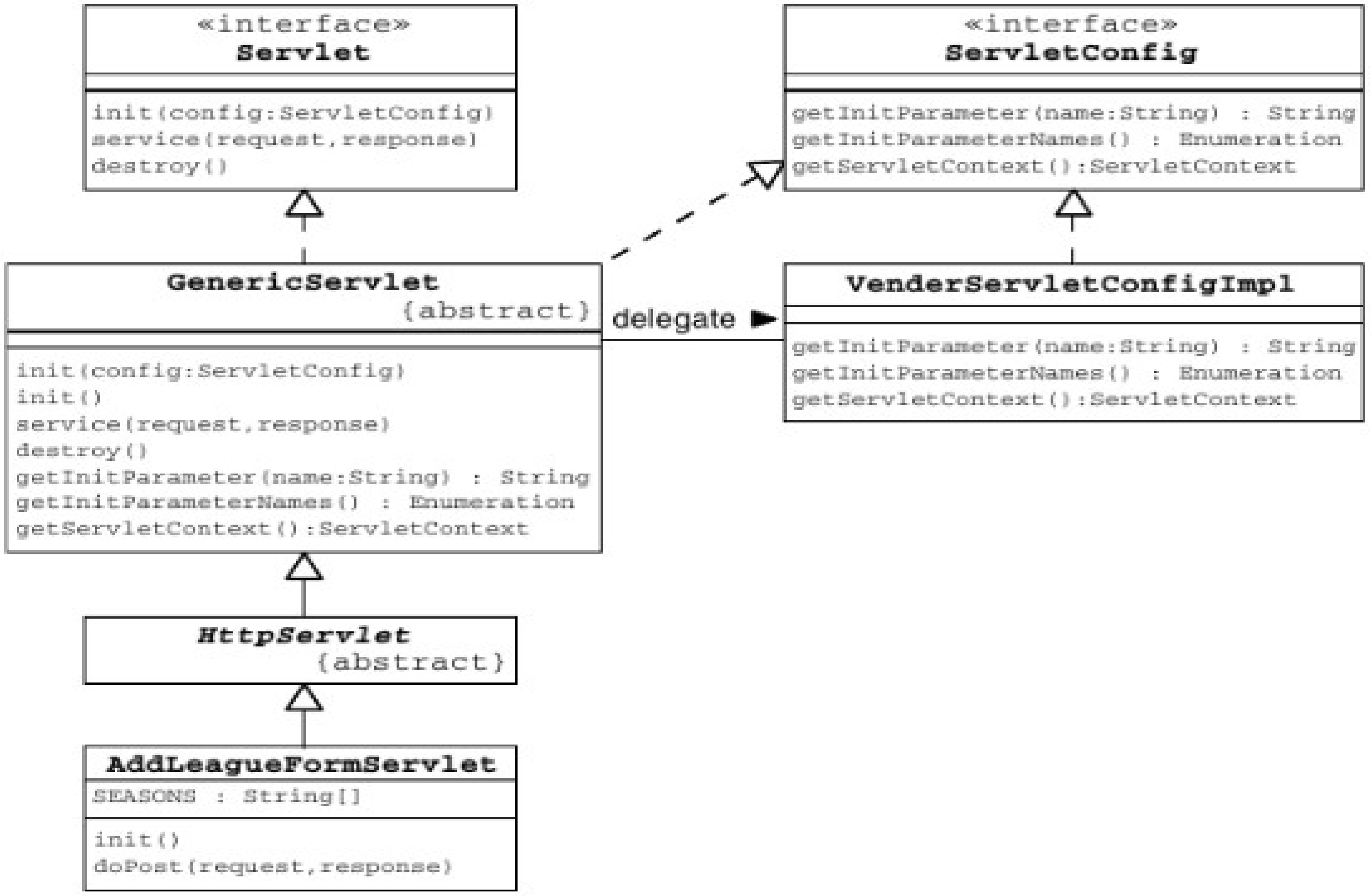
Discussion – The following questions are relevant to understanding what technologies are available for developing web applications and the limitations of those technologies:

- How are servlets loaded, when, and how many instances are created?
- What happens when mulitple clients request service from the same servlet concurrently?
- What happens when a piece of behavior should be applied to multiple pages? Does the code have to be duplicated in multiple servlets?

Servlet Life Cycle Overview



The ServletConfig API


«interface»
Servlet
init(config:ServletConfig)
service(request,response)
destroy()
«interface»
ServletConfig
getInitParameter(name:String) : String
getInitParameterNames() : Enumeration
getServletContext():ServletContext
GenericServlet
{abstract}
init(config:ServletConfig)
init()
service(request,response)
destroy()
getInitParameter(name:String) : String
getInitParameterNames() : Enumeration
getServletContext():ServletContext
delegate
VenderServletConfigImpl
getInitParameter(name:String) : String
getInitParameterNames() : Enumeration
getServletContext():ServletContext
HttpServlet
{abstract}
AddLeagueFormServlet
SEASONS : String[]
init()
doPost(request,response)

Servlet Lifecycle and Annotations

Java EE 5 introduced some key annotations for container managed objects. In the web container, these objects are Servlets, Filters, and many listeners.

A variety of annotations provide dependency injection for different resource types. Some of these are listed below:

- @EJB
- @Resource
- @PersistenceContext
- @PersistenceUnit
- @WebServiceRef

Lifecycle Method Annotations

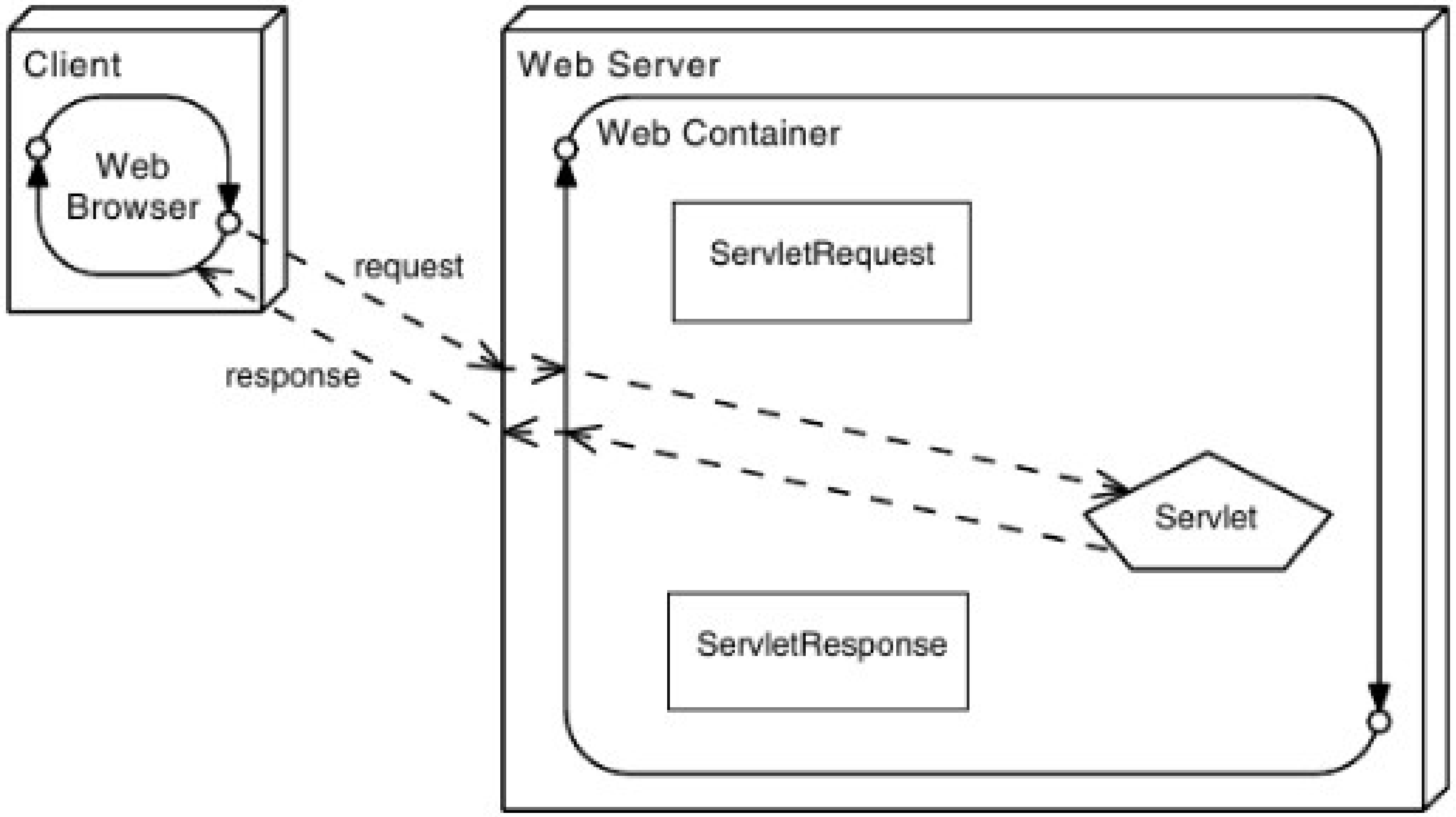
The Java EE specification requires that objects that qualifies dependency injection must also be supported with two lifecycle annotations. These are @PostConstruct and @PreDestroy. These methods are very similar in function to the init() and destroy() methods of a servlet.

If an @PostConstruct method throws any exceptions the container must abandon the object. No methods may be called on the object and the object must not be put into service.

Servlets and Threading

- If multiple clients invoke behavior that calls the same method, in the same servlet instance, at the same time, then that one servlet method (in a single servlet instance), might be concurrently executing many times under the control of many different threads—one thread for each client.

Web Container Request Cycle



Applying Filters to an Incoming Request

Filters are components that you can write and configure to perform some additional pre-processing and post-processing tasks. When a request is received by the web container, several operations occur:

1. The web container performs its pre-processing of the request. What happens during this step is the responsibility of the container provider.

2. The web container checks if any filter has a URL pattern that matches the URL requested.

3. The web container locates the first filter with a matching URL pattern. The filter's code is executed.

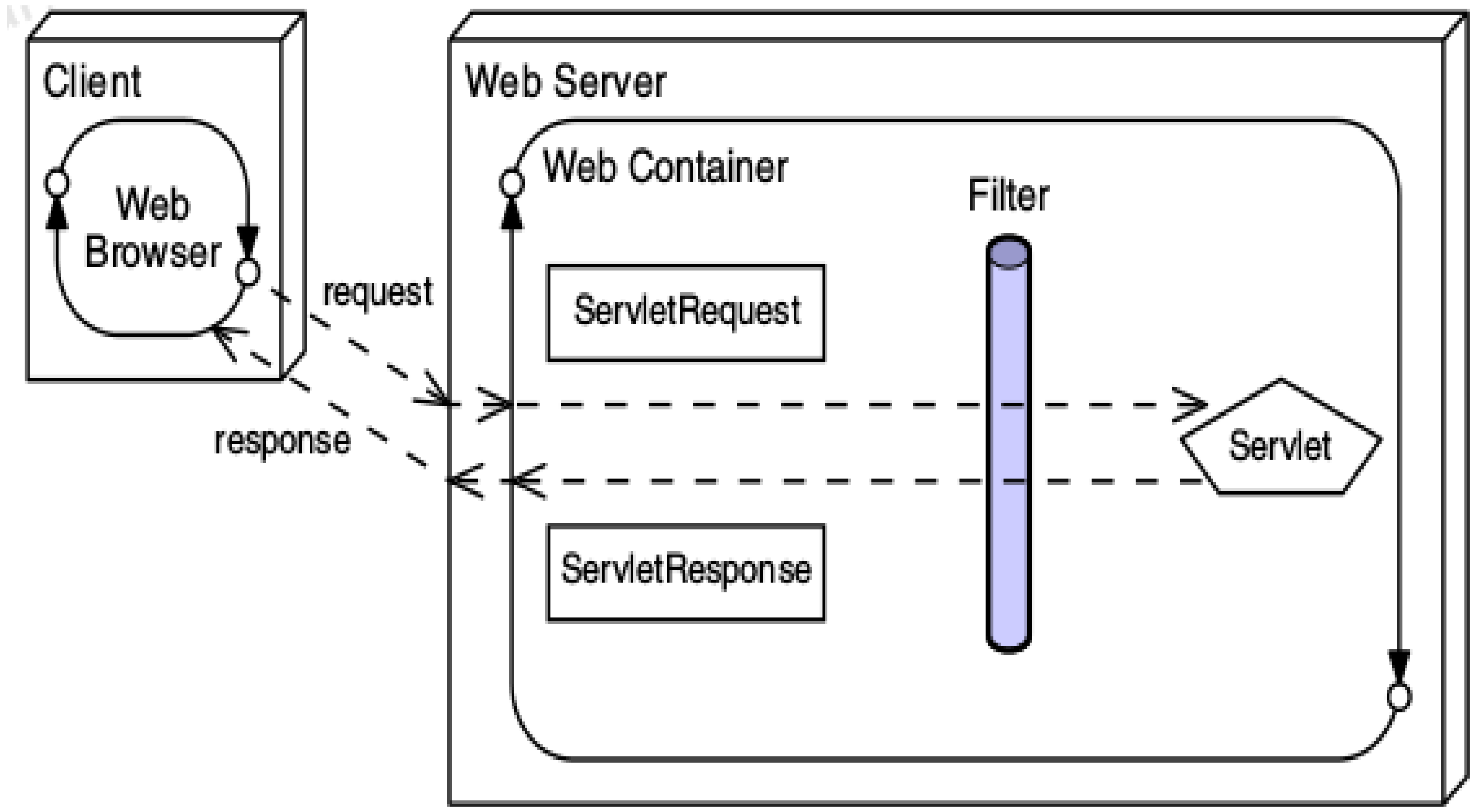
Applying Filters to an Incoming Request

4. If another filter has a matching URL pattern, its code is then executed. This continues until there are no more filters with matching URL patterns.

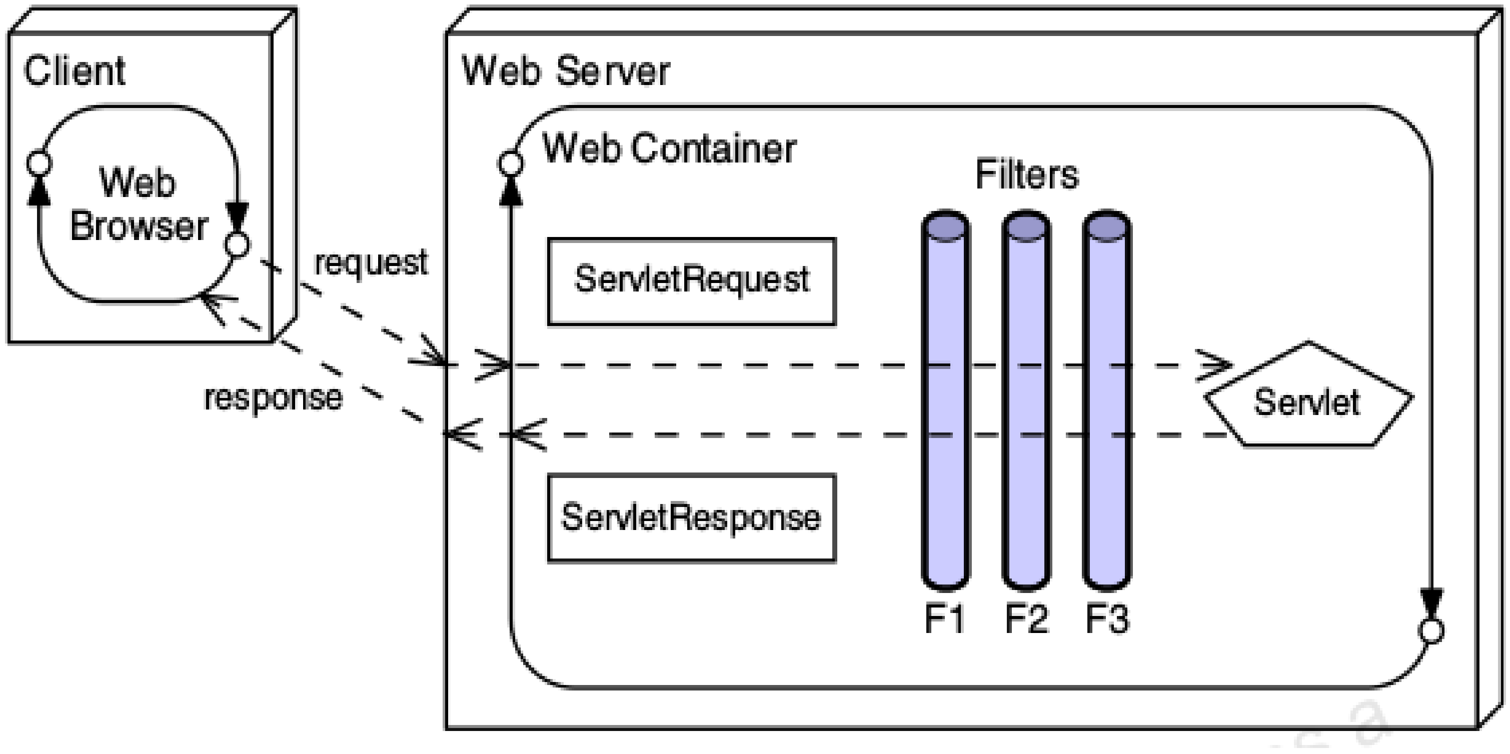
5. If no errors occur, the request passes to the target servlet. 6. The servlet passes the response back to its caller. The last filter that   was applied to the request is the first filter applied to the response.

7. The first filter originally applied to the request passes the response to the web container. The web container might perform post-processing tasks on the response.

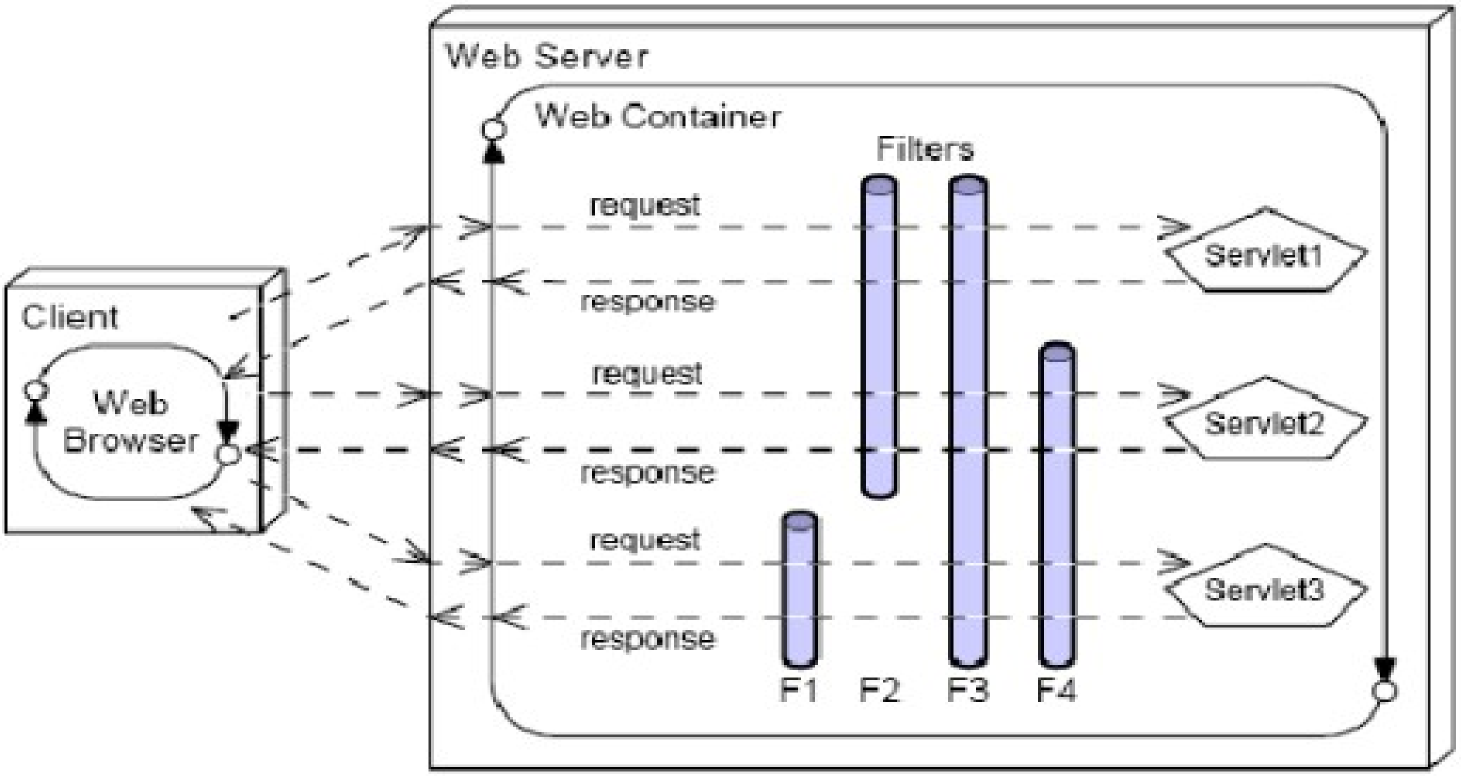
Applying Filters to an Incoming Request



Applying Filters to an Incoming Request



Modular Use of Filters



Applying Filters

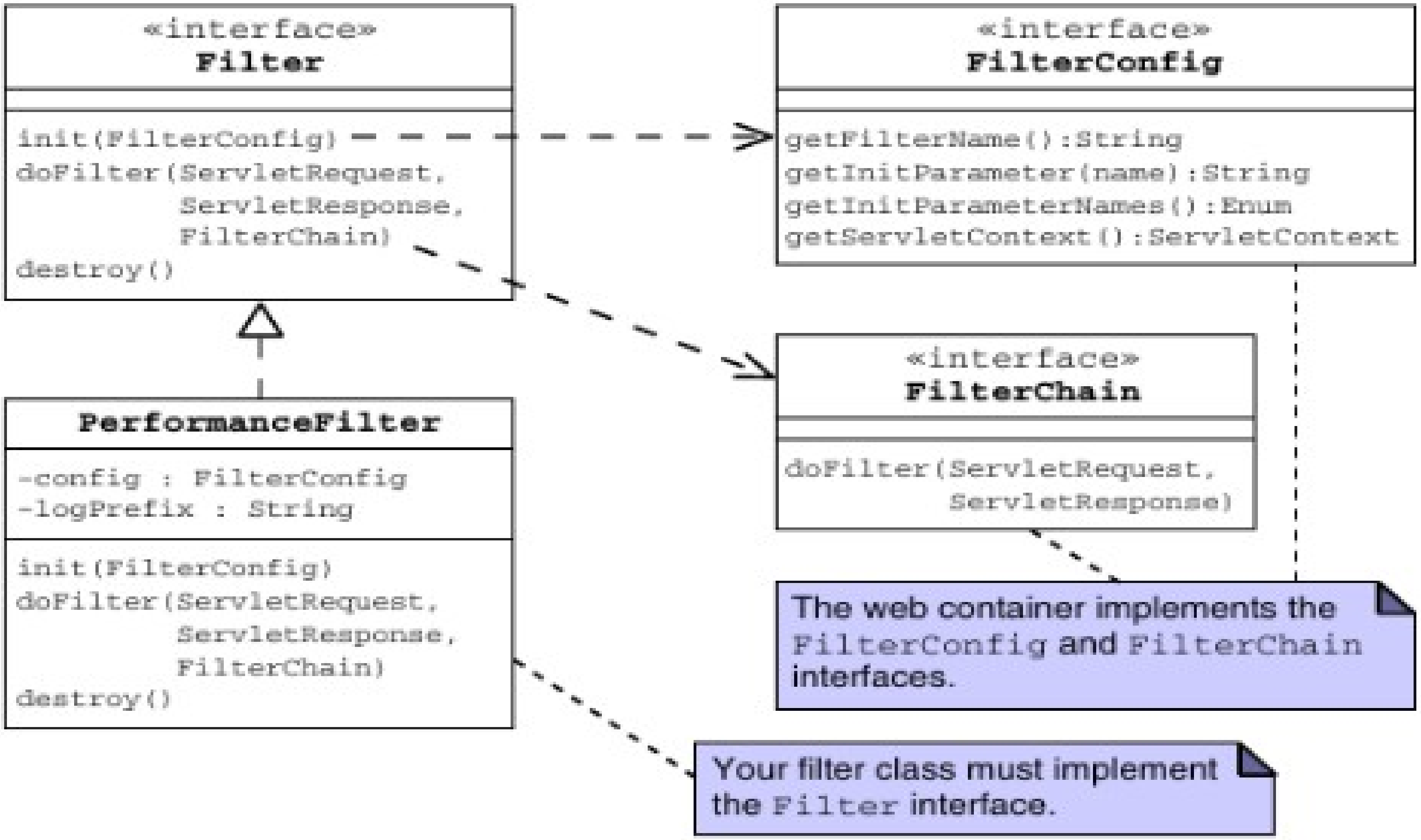
Filters can be used for operations such as:

- Blocking access to a resource based on user identity or role membership
- Auditing incoming requests
- Compressing the response data stream
- Transforming the response
- Measuring and logging servlet performance

Filter API

The Filter API is part of the base servlet API. The interfaces can be found in the javax.servlet package.

Filter API


«interface»
Filter
init(FilterConfig)
doFilter(ServletRequest,
ServletResponse,
FilterChain)
destroy()
«interface»
FilterConfig
getFilterName():String
getInitParameter(name):String
getInitParameterNames():Enum
getServletContext():ServletContext
«interface»
FilterChain
doFilter(ServletRequest,
ServletResponse)
PerformanceFilter
-config : FilterConfig
-logPrefix : String
init(FilterConfig)
doFilter(ServletRequest,
ServletResponse,
FilterChain)
destroy()
The web container implements the FilterConfig and FilterChain interfaces.
Your filter class must implement the Filter interface.

The init() method of Filter API

```
public void init(FilterConfig config)
```

init() method is invoked only once. It is used to initialize the filter.

The doFilter() Method

public void doFilter(HttpServletRequest request,HttpServletResponse response, FilterChain chain)

doFilter() method is invoked every time when user request to any resource, to which the filter is mapped.It is used to perform filtering tasks.

The destroy() method of Filter API

public void destroy()

This is invoked only once when filter is taken out of the service.

Configuring the Filter

Configuring a Filter Using Annotations:

The annotation is @WebFilter, and this allows specification of the filter name, the url patterns to which it responds, and the types of invocation for which it should be invoked. The annotation also allows the specification of init parameters.

Configuring the Filter

Configuring a filter using web.xml:

```
<web-app>

<filter>
<filter-name>...</filter-name>
<filter-class>...</filter-class>
</filter>

<filter-mapping>
<filter-name>...</filter-name>
<url-pattern>...</url-pattern>
</filter-mapping>

</web-app>
```

Simple example of filter

```
public class MyFilter implements Filter{

public void init(FilterConfig arg0) throws ServletException {}

public void doFilter(ServletRequest req, ServletResponse resp,
    FilterChain chain) throws IOException, ServletException {

    PrintWriter out=resp.getWriter();
    out.print("filter is invoked before");

    chain.doFilter(req, resp);//sends request to next resource

    out.print("filter is invoked after");
    }
    public void destroy() {}
}
```

Handling Multipart Forms

Java EE 6 introduced a mechanism to simplify handling of multipart form data. Three key elements make this up, which are: additional methods in the javax.servlet.http.HttpServletRequest object, an annotation javax.servlet.annotation.MultipartConfig, and an interface javax.servlet.http.Part

Stay connected

About us: Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies,
No-1, 9th Cross, 5th Main,
Jayamahal Extension,
Bangalore, Karnataka 560046
T: +91 80 6562 9666
E: training@emertxe.com

https://www.facebook.com/Emertxe https://twitter.com/EmertxeTweet https://www.slideshare.net/EmertxeSlides

Thank You