Chapter-10

Introducing to Filtering

Some popular use of filter s include –
Authentication
Auditing
Compression
Encryptions
On the fly format transformation

- Common filter application:
  1. Filter can intercept request header information before it reaches the resource in the processing pipeline and can therefore be used to create customization authentication.
  2. Filter can be written to perform sophistication logging and auditing.
  3. Filter is also useful in data transformation.
  4. Filter can present the serving of a particular resource altogether and generate their own response.
- The big picture of filtering:
  The middle tier component of the java EE archive true consists of an application server often fronted by a web server.
  The middle- tier server becomes a server that serves one of the three type of resources based on incoming requests-
  1. Static content (HTML , images and so on)
  2. A servlet
  3. A JSP page

Filters are positioned in the processing pipeline. between the application server and the client.
A filter provides application level access in the request handling pipeline of the container.

Filtering pipeline :

Filters can be used to do the following:-

1) Inspect the request header and data before it reaches the resource.
2) Inspect the response header and data after it has been send by the resource.
3) Provide a modified version of the request to the resource being processed by the container.
4) Access and modify the response from the resource before returning it.
5) Stop a request from reaching a resource altogether.

The Filtering interface :

→A filter is simply a class that implement the javax.servlet.Filter interface similar to the javax.servlet.Servlet interface.
→There are three life cycle method that a filter must implement.

1) Public void init(FilterConfic config) throws Servlet Exception.
2) Public void doFilter(SrevletRequest req,ServletResponse res,FilterChain chain) throws IOException ,ServletException.
3) Public void destroy() when filter is being taken out of service.

Configuration and deployment of filters:

→ Filter are typically created by developers and delivered as bytecode classes within a web application.
→ A web application filters are configured and by specifying the following-

a) Which filter(s) to use
b) Any initialization parameters to the filter
c) Where to apply the filter
d) The order in which multiple filters are chained together(if applicable)

→ The servlet container supporting filters will parse the following two types of filter related declarations present within this file-

1) Filter definition :-Tells the container the textual name associated with the filter.
2) Filter mapping :- Tells the container which resources the filter will be applied to.

The life cycle of a filter:

→ The filter life cycle is almost identical to that of servlet life cycle-

1) Instantiation
2) Initialization[init()](explicit stage)
3) Filter execution[doFilter()](explicit stage)
4) Destruction[destroy()]

→ The constructor is called during instantiation phase.

→The init() method is called for initialization.

→ The doFilter() method contains all the filter processing logic,handles the filter execution phase.

→ The destroy() method is used in the destruction phase.

→ the initialization must occur before the first request is mapped through the filter interface.

→During the initialization, the container passes a FilterConfig object via the init() method of the javax.servlet.Filter interface.

→The filterConfig can be used by the filter to obtain-

-- initialization parameter of the filter

--the textual name of the filter

The FilterConfig interface

→ the FilterConfig interface declares four methods

1) public String getFilterName(): obtain the textual name of the filter, as defined in the web.xml deployment descriptor.
2) public String getInitParameter(String paramName) : obtain the string value of the specific initialization parameter by name.
3) public Enumeration getInitParameterName(): obtain a java.util.Enumeration consisting of all the names of the initialization parameter for this interface in web.xml<fliter>
4) public ServletContext getServletContext(): obtain the ServletContext that the filter is execute within the server.xml.

→ we must take care of to write filters that are thread safe.

→ the filter instance will kept alive to process a request until the container(or a VM in the container) shuts down or the associated web application is undeployed.

Filter Defination:

The filter definition appear in the web.xml deployment descriptor inside the <filter> element. Each <filter> element must have the following child element.

```
<filter>
<filter-name>AuditFilter</filter-name>
<filter-class>Filters.Audit Filter</filter-class >
</filter>
```

And optionally one or more <init-param> child element. It contains <param-name> and <param-value> subelement.

Filter Mapping:

→Filter mapping are XML-based entries, specified per web application within the web.xml file

```
<filter-mapping>
<filter-name>VusialAudit Filter</filter-name>
<servlet-name>mylocate</servlet-name>
</filter-mapping>
```

→ the servlet name is defined within the same web.xml file like this

```
<servlet>
<servlet-name>mylocate</servlet-name>
<servlet-class>FindProd</servlet-class>
</servlet>
```

Matching URL patterns:

→ the real power of mapping becomes evident when we use URL patterns matching

URL Pattern	Matches
/*	Everything that is served by this web application including static pages, servlets and JSP pages.
/servlet/*	All servlets
/jsp/*.jsp	All jsp pages located on the /jsp path
/dept/accounting/*	All resources in the accounting department branch of web application.

```
<filter-mapping>
<filter-name> </filter-name>
<url-pattern>/* </ url-pattern>

</filter-mapping>
```

Insertion of filters into the request flow:

→Tomcat 5.5 have full flexibility to insert filters into the request flow even in between programmatically dispatched point

Consider a filter mapping similar to the following;

```
<filter-mapping>
<filter-name>F1</filter-name>
<url-pattern>/* </ url-pattern>
</filter-mapping>

<filter-mapping>
<filter-name>F2</filter-name>
<url-pattern>/* </ url-pattern>
</filter-mapping>

<filter-mapping>
<filter-name>F3</filter-name>
<url-pattern>/* </ url-pattern>
</filter-mapping>
```

→ incoming request→

←Outgoing response←

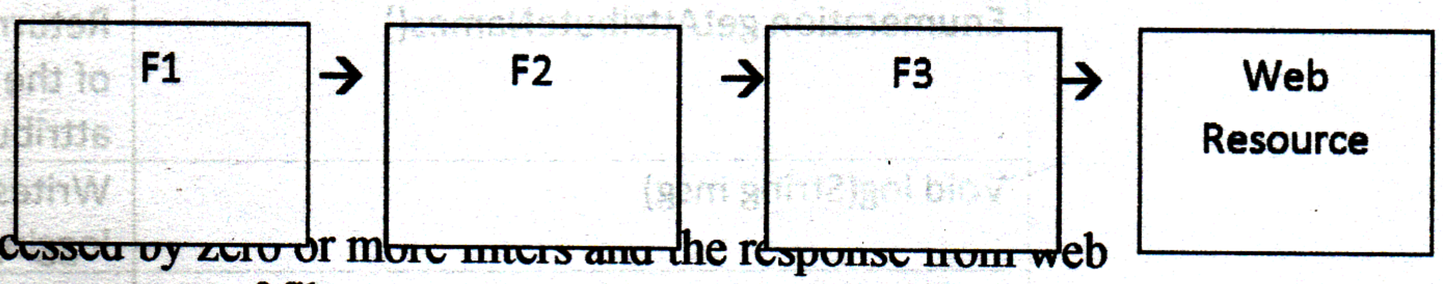

Fig: an incoming request is processed by zero or more filters and the response from web resource can be processed by the some set of filters.

Note: when the primary web resource used the request dispatcher to include other processing resources, the include processing resource cannot be filtered. The same thing happens when the web resource use the request dispatcher to forward the request to other processing resources.

→ The <dispatcher> subelement of a filter mapping allow us to specify the filtering is to be performed during regular request.

→ <dispatcher> subelement of <filter-mapping> is optional.

<filter-mapping>
<filter-name>F3</filter-name>
<url-pattern>/* </ url-pattern>
<dispatcher>REQUEST</dispatcher>
</filter-mapping>

<dispatcher> Value:

1) REQUEST
2) FOEWARD
3) INCLUDE
4) ERROR

Filter chaining :

Chaining is the action of passing a request through multiple filters in a sequence before accessing the resource request.

The FilterChain interface :

Javax.servlet.FilterChain interface through the doFilter() method
public void doFilter(ServletRequest req , ServletResponse res) throws IOException ,ServletException.

***All filters are Chainable:**

>All filter are intrinsically chainable.

***** Unique properties of filters chaining:**

Each of the filterchain dofilter() method calls are stacked upon one another. So program flow is blocked in a nested fashion across all the filters involved.
The order of chained filters execution on the incoming request is down the filter stack, whereas the order of chained filter execution on the outgoing path is up the filter stack.

- Initial Parameters for filters:

```
<filter>
<filter-name> stop Games filter</filter-name>
<filter-class> filters StopGamesFilter</filter-class>
<init-param>
<param-name> starthour</param-name>
<param-value>10</param-value>
</init-param>
</filter>
```

  - The starthour will be accesssiable within the filter via a call to the filterconfig object:
    String start Hour= filterConfig.getInit Parameters("starthour");

- **Filter writers method:**

Method	Service
Object getAttribute(String name)	Obtain the value of a named attribute.
Void setAttribute(String name, Object object)	Attaches a named attribute to the Servlet Contest.
Void removeAttribute(String name)	Remove a previously attached attribute.
Enumeration getAttributeNames()	Return a java.util. Enumeration consisting of the nnames of all the currently attached attributes.
Void log(String msg)	Writes a String to the currently active logging facility associated with the context.
Void log(String msg, Throwable throw)	Writes a String and an Stack trace to the log.

❖ Testing the filter:

1. Go to Tomcat's logs subdirecting and delete all files.
2. Create a web application with the filter and index.jsp and deploy it.
3. Start Tomcat 5.
4. Start a browser and navigate to http://localhost:8080/filters1/index.jsp.
5. After the web page gas located in the browser, shut down Tomcat.

❖ **Make the code Thread –Safe:**

There is typically only one instance of a filter per java VM. This makes it inevitable that the dofilter() method will be encounter by many threads simultaneously. So the filter must be thread-safe. This means the following:

1. Local variable in the filter class scope should be read-only, or the scope access must be synchronized.
2. Beware of calling methods that may modify instance variable indirectly or outside of synchronization.

What is filter?

- ✓ Filters enable web-application programmers to tap into the request-processing pipeline of the container.
- ✓ Filters are packaged code components in a web-application, at the same level as servlet and JSP pages.
- ✓ Filters are deployed at the same way as servlet and JSP pages, through the deployment descriptor for the application.
- ✓ The filter has access to in coming requests before they reach the final resource and to the outgoing response immediately after the resource processing.
- ✓ Filter can also substitute their own version of the request and response (typically wrapped) for consumption of the resource.
- ✓ Symbiotic, well-define interactions between the request dispatcher and filter enable us to create filter that act as in-series processors for request- switching web-application and application frameworks such as Apaches Struts and Turbine.

Chapter-11

❖ **Filter for five problem domains:**

Application Domain	Filter Sample
1.Auditing	A visual auditing filter that includes audit information inline with every resource that it services.
2. Authorization	A filter that disallows access to the server during certain hour of the day.
3.Adapater	An adapter filter that allows newly formatted queries to work with a legacy set of resources.
4.Authentication	An ad hoc authentication filter that can add simple login protection to any (group of) resource.
5.Pipeline Processing	A data-processing filter that takes advantage of the request flow along the processing pipeline.

❖ **Techniques and filters:**

Techinque Illustrated	Filter
1.Transforming incoming request headers	Adapter filter
2.Transforming outgoing respose content	Auditing
3.Wrapping request objects	Adapter
4.Wrapping response	Aduting

5.Dinamically adapting filter behavior based on incoming requests	Authentication filter
6.Stopping downstream request flow	Authorization filter Authentication filter
7.Generating response	Authorization filter Authentication filter
8.Adding to or modifying the attributes of a request is a processing pipeline	Pipeline processing filter Authentication filter
9.Interacling with the request dispatcher's include() and forward actions	Pipeline processing

❖ **A brief word on terminology:**

- The request originates from the Clint and foes though a chain of filters before reaching the final resource processor.
- First the request travels from the client, through the first filter, the second filter and soon. This call the down stream or inbound trip. The downstream trip is always toward the final goal.
- After the resource process has finished with the request a response then will travel upstream or out bound back through all the filter and onward to the client.

Form client downstream-→inbound
Client ->filter1->filter2->filter3-> response
Upstream <--outbound to client

❖ **Wrapping the response object for content modification:**

- We must wrap the response with our own custom version during the request's inbound trip, before calling the chain dofilter()
  In fact the following happens

1. The filter supplies a custom wrapped version of the response to downstream fitness when it calls the chain. Dofilter().
2. The custom wrapped response object hands down a custom output stream or printwriter object that is actually a byte array managed in the code.
3. When downstream filters, ot the resource processor, writ4e to the custom output stream or printwriter, its buffering all the output.
4. When downstream filters, or the resource processor flush or close the custom output stream or Printwriter,we examine the buffering all the output.
5. When downstream filters or the resource processor, flush or close the custom output stream or Printwriter, we examine the buffered output for the closing </body> tah and insert the auditing information just before it(if found).

❖ **The Replace ContentOutputStream class:**

- This class wraps an outputStream and does the following:
  1. Supplies its own byte array-based stream for the write() method, called by downstream filter and the resource processor.
  2. Handles the close() method by calling a child replaceContext() method to transform the byte array stream.
  3. Provides the transformed content through the getResult() method. This method is called by the filter when the filter write the response.
- ServletOutputStream is the base class of OutputStream which is refused by getoutputStream() on a response.
  ServletOutputStream requires the write(int) method to be implemented by its subclass.
  It implements all the other writer() variants based on this method.

- The ReplaceContextOutputStream requires the replaceContext( ) method to be implemented by all in subclass.
- The flag variable closed, initially false, is used to ensure that the wrapped stream is closed only once, regardless of how many times the close( ) method maybe called.

- The write( ) method writes to the in-memory by array stream called stream, this ensure that all writes on this stream write to Stream(essentially a buffer).
- The processStream( ) method call the replaceContext() method to transform the in-memory stream and write the transform output to the wrapped output stream.

Authorization filtes

It allows or disallows an incoming request to reach it destined resource processor depending on the time of day.

Fig. when a request is made outside the allowable hours, the filter denies access to the request resource.

- Thread safety considerations:

After a instance of a filter has been created and before the very fast do filter() method is called. The container calls the init() method to set the filterconfig object for the filter. This is a natural point at which to perform any initialization required and it is conceptually equivalent to the init() method of a servelet.

Filter for adapting to legacy resources

- Writing the legacy AdapterFilter:

This filter will perform several tasks:-

1. Determining the awning IP address.
2. Map the IP address from which the request originates to department.
3. Add the DEPT parameter to the request before it reaches the underlying jsp resource.

Ad Hoc authentication Filter

- When we need simple, temporary protection of selected resources, the AdHokAuthenticate filters can be the best choice.
- The action of this filter is Straightforward. It triggers basic authentication on the client browser.
- Almost all known browsers, including even the earliest version support basic authentication.

The authentication process works like this:-

1. A client attempts to access a protected resource.
2. The server examines the clients request to determine whether theirs any authorization data in the "Authorization" heder.
3. If authorization data is not found, the server sends back HTTP status code 401(Unauthorized access) and a header with www-authenticate:BASIC realm=<realm>, where realm is a next string that will be displayed to the client.
4. The client browser pops up a login screen in which the user should enter a username and password.
5. When the user enters the username and password, the client encodes the username and password by using simple base64 encoding and sends both to the server.
6. The server examines the client request to determine whether there's any authorization data, decoding the base64 extended password if necessary. If there is no authorization data, the process goes back to step 3.
7. The server verifies the password and either allows or reject access.

Filter in the request-processing pipeline

➢ Understanding the pipeline model:-

There are many name given to pipeline data processing model. It's often identified as the enabling element of Model-view controller(MVC) web application design, and sometimes it refferd to as the push model of application design, contrast with the more conventional pull model.

Some of the highly desirable properties of a application model:-

1. The request stays intact as it traverse the pipeline with work being carried out only on attached attributes (sometimes called decorators).
2. Data management, business logic, and presentation logic can be clearly separated into different processors and renderers.
3. The processor can be designed to be completely composable (that is chained in any order) or highly specialized.
4. Multiple renderers can compose the final output(such as data to XML to HTML via XSTL).
5. The state of the request travels with the request through the pipeline.

- In general, designing request-handling login by using the pipeline model provide the following-
  - Processors and renders those are readily reusable as components.
  - Strong and clear separation between data management, business logic, and presentation logic in a application.
  - Application that are more maintainable.
  - Application that are more adaptable to changing business request requirements
  - Robust applications that will be sealable with container technology and hardware advancer.

Web applications whose performance is highly optimizable.

Chapter-12

#Overview of Application security:

➔ Authentication is the process by while a web application verifies that "you are who you say you are". For exampie-when a user iogs into web page with a user name and password, the web application validates the entered credentials against it used data store

➔ Authorization: to delete a user from the database, we need to be a authorization-administrator.

➔ The security features that all servlet containers provide are as follows-

1) Authentication: The process of providing the identify to an application

2) Access control for resources: this means by which interactions are limited to users, roles or programs

3) Data integrity: That means for providing that a third party hasn't modified information while it was in transit

4) Confidentiality or data privacy: The means used to ensue that is made available only to users who are authorized to access it.

➔ DOS attack: Denial of service attack, in which a application or site is attacked by an enmous number of hits at the same time

✓ By using Container-managed-security, an application will be much easier to maintain and comprehend. We can easily add new roles to the application by adding or altering a few lines in the application's deployment descriptor.

✓ Using Container-managed-security also makes it easy to switch from using a file-based user data store (or realm) to a database or LAPD-based realm.

✓ Deployment descriptor is the primary vehicle to implement security

Using Authentication:

✓ One of the most popular JSP/Servlet sample applications produced today is the "how to log in"application.

✓ A Realm is a database of usernames and passwords that identify valid users of a web application

Configuration setting in web.xml file

```
<Security-constraint>

    <web-resource-collection>
        <web-resource-name> My Application </ web-resource-name >
        <url-pattern> /* </ url-pattern >
    </web-resource-collection>

    <auth-constraint>
      <role-name>  *  < /role-name >
    </auth-constraint>

</Security-constraint>
<login-config>
    <auth-method> BASIC </auth-method>
</login-config>
```

- ✓ The significant elements in the preceding code are <url-pattern> , <role-name> , <auth-method>
  1) The <url-pattenrn> element defines the characters to look for in a client's request. This value can be a path based pattern (such as /admin/*)or an extension based pattern (such as /admin/*.jsp) and it doesn't include the context path. Any resources that match this path will be secured by the container.
  2) The <role-name> element indicates the roles allowed to view the secured resources
  3) The <auth-method> defines the type of authentication mechanism to use such as basic (a simple dialog box with the username/password) or form (a direct to an HTML based login page)

- ✓ When we add the <security-constraint> element to the deployment descriptor of a web application installed in Tomcat , the application will be protected by Tomcat's default memory realm.

```
<tomcat-users>
<role rolename="tomcat"/>
<role rolename="role1"/>
<role rolename="manager"/>
<role rolename="admin"/>
<user username="tomcat" password="tomcat" roles="tomcat"/>
<user username="both" password="tomcat" roles="tomcat,role1"/>
</tomcat-users>
```

- ⇨ Authentication options:

Mechanism	Configuration
• HTTP basic authentication	<auth-method> BASIC </auth-method>
• HTTP digest authentication	<auth-method> DIGEST </auth-method>
• HTTP client authentication	<auth-method> CLIENT-CERT </auth-method>

• Form- based authentication	<auth-method> FORM </auth-method>

⇨ When using HTTP basic authentication , the server will authenticate a user by using a username and password from the client.

- The target server is not authenticated; therefore this is not a secure mechanism. The client has no proof that the server is who it says it to for a server to prove it identically, it needs to obtain a SSL certificate from a certificate authority (such as VeriSign)
- If we need greater security but still wish to use basic authentication , we combine it with a virtual private network(VPN)

⇨ HTTP digest authentication , the used is prompted with a username/password dialog box that looks similar to the basic authentication dialog box

⇨ HTTP client authentication requires the user to posses a Public Key Certificate(PKC) and is based on HTTP over SSL, hence the name HTTPS

- PKCs are useful in applications with strict security requirements and also for single sign on form within the browser.

⇨ Form-based authentication is the final option when using the declarative security.Unlike the others it allows the developer to customize the look&feel on the login screen.

⇨ session.invalidate() method use to loging out a JSP page or Servlet.

⇨ JDBC Realm:
The JDBC Realm allows us to configure declaratively the location for storing user's information

⇨ JNDIRealm:
Configuring a JNDI Realm is similar to configuring a JDBC Realm,however the realm attributes are slightly different.

⇨ SSL:
Secure Sockets Layer(SSL) is a technology that allows web browsers and web servers to communicate over a secure channel.

⇨ SSL hand Shake:
IN SSL, data is encrypted at the browser (before transmission) and then decrypted at the server before reading the data. This same process takes place when the server returns data to the client. The process is known as SSL handshake.

⇨ There currently three levels of encryption supported by this protocol:40-bit, 50-bit,128-bit

⇨ With HTTP basic authentication, user may find if frustrating when their login fails.

<u>SSL Configuration</u>

```
<user-data-constraint>
  <description>
Encryption is not required for the Application in general.
  </description>
<transport-gurantee> NONE </transport-gurantee>

</user-data-constraint>
```

Specify how data should be sent between the client and server.

<u><transport-gurantee> values:</u>

Value	Description
NONE	The application does not required any transport-gurantee.This is the same as not including <user-data-constraint> element in web.xml
INTEGRAL	The data can not be changed in transit
CONFIDENTIAL	The data will be transmitted in a fashion that presents other entities from observing the contents of the transmission.

SSL drawback

One drawback to using SSL in a web application is that it tends to significantly decrease the throughput of the server. This is mainly due to the encryption and decryption process on each end of the connection.

Java Authentication and Authorization Service(JAAS)

- ⇨ JAAS provides a framework and standard programming interface for authenticating users and for assigning priviliges.
- ⇨ JASS can be helpful when we use complex authentication schema or when we grant resource specific privileges to users(for example-inserting a row in a database, or assigning write permission to a file)

#Servlet 2.5 Security Changes:

HttpServlenRequest interface gives us the following methods -

1. getRemoteUser()
2. idUserInRole(String rootName)
3. getUserPrincipal()

```
<Security-role-ref>
        <role-name> admin </role-name>
        <role-link> accounting </role-link>
</Security-role-ref>
```

#the purpose of <security-role-ref> is to create link.