## Chapter 12 : Filters and Interceptors

1. Server side, **two kinds of filters**:
   1. **Request Filter** : They get executed before a **JAX-RS Method** is invoked.
   2. **Response Filter** : After a JAX-RS method is invoked.  
      **NOTE**: **By default,** they are executed for all HTTP requests but can be bound to a specific **JAX-RS Method** too.
2. Internally, the algorithm for executing an HTTP on the server side looks something like this:  
   Graphical user interface, text

   Description automatically generated with medium confidence
3. **NOTE**: JAX-RS filters are different from those in Servlet API.
   1. Servlet Filters wrap around the servlet processing and are run in the same Java Call Stack.   
      Whereas JAX-RS filters run in their **Java Call stacks** as **JAX-RS** has **asynchronous API**.  
      Each request filter runs to completion before the JAX-RS method is invoked.  
      Each response filter runs to completion only after a response becomes available to send back to the client.  
      In the asynchronous case, response filters run after **resume()**, **cancel()** or a **timeout** happens.  
      See Chapter 13 for more details on the Asynchronous API.

# Server Request Filters

1. **Request Filters** are implementation of **ContainerRequestFilter.**Graphical user interface, text, application

   Description automatically generated
2. ContainerRequestFilter comes in two flavors:
   1. **Pre-Matching Filters**:
      1. Designated with **@PreMatching**.
      2. Will be executed before the **JAX-RS resource method** is matched with the incoming **HTTP Request**.
      3. **Usage**: They are often used to modify request attributes to change how they match to a specific resource.
         1. **For Example**:
         2. Some firewalls don’t allow PUT and/or DELETE invocations.
         3. To circumvent this limitation, many applications tunnel the HTTP method through HTTP header **X-HTTP-Method-Override**.  
            Text

            Description automatically generated
         4. A filter can modify pretty much everything about the incoming request through methods on **ContainerRequestContext**.  
            But once the request is matched to a **JAX-RS Method**, a filter can’t modify the request **URI or HTTP Method**.
      4. **Another use case** for request filters is implementing **Custom Authentication Protocols**:
         1. **For Example**, OAuth 2.0 has a token protocol that is transmitted through the **Authorization HTTP header**.
         2. Here is what the implementation of that might look like:
         3. A picture containing text, screenshot, font, line

            Description automatically generated
   2. **Post-Matching Filters**:
      1. Those filters which are not annotated with @PreMatching annotation, come under the category of **post-Matching filters** which mean they will be invoked after the received request has been mapped to **JAX-RS Method**.

## Server Response Filter

1. **Response Filters** are implementations of the **ContainerResponseFilter** interface.
2. A picture containing text, screenshot, font, line

   Description automatically generated
3. Generally, Response filters are used to decorate Response by adding or modifying response headers.  
   Such as for each GET Request Response, we want to set **Cache-Control header.**A screen shot of a computer code

   Description automatically generated with low confidenceThe **ContainerResponseContext** allows us to view**, modify, add the response before it is marshalled and sent back to the client**.

# Reader and Writer Interceptors

1. While filters deal with headers, **Reader & Writer** deal with **message bodies**.
2. They work in conjunction with a **MessageBodyReader & MessageBodyWriter** and are usable on both the client and server.
3. **Reader Interceptors** implement the **ReaderInterceptor** interface and **Writer Interceptors** implement the **WriterInterceptor**.
4. A screen shot of a computer code

   Description automatically generated with low confidence
5. These interceptors are triggered when a **MessageBodyReader** or **MessageBodyWriter** is needed to unmarshal or marshal a Java Object to and from the HTTP message body.
6. Unlike filters, they are invoked in the **same Java Call Stack**.
7. In other words, a **ReaderInterceptor** wraps around the invocation of **MessageBodyReader.readFrom()** and a **writeInterceptor** wraps around the invocation of the **MessageBodyWriter.writeTo()**.
8. d

WE WILL COME BACK TO INTERCEPTOR