

Filters and Interceptors

Filters and Interceptors are introduced from JAX-RS 2.0 release. These can be configured on both Client and Server side.

Filters can modify inbound and outbound requests and responses such as modification of headers, entity and other request/response parameters. If we want to supply any additional parameters through a set of Requests/Responses, we can add those parameters into a filter and configure it to the Client or WebTarget instance.

Interceptors are used to modify the entity input and output streams such as to zip and unzip output and input entity streams.

Server Request Filters:

These filters can be used to modify the request level parameters like headers at Server side.

In order to create a request level filter at the server side, add a class by implementing from `ContainerRequestFilter` interface and annotate it with `@Provider` annotation to register it with the JAX-RS runtime. This filter will be applied globally to all the incoming requests to REST resource after the resource method is matched by the JAX-RS runtime. It has a single method *filter* which will take `ContainerRequestContext` instance as a parameter, a mutable class that provides request-specific information for the filter. This method will be executed before dispatching the request to resource method and after identifying the resource method.

```
public class ServerRequestFilter implements ContainerRequestFilter
{
    @Override
    public void filter(ContainerRequestContext requestContext) {
        // Implement your logic here
    }
}
```

If there are multiple filters configured, the filters will be executed in a chain. The order of execution can be controlled by applying `@Priority` annotation to the filter class. Execution will be in the ascending order of the `@Priority` annotation value.

This filter may abort the request by sending a custom Response from it. In this case, the corresponding filters and resource method execution will be stopped and delegates the request to Response filters.

If you want to apply a filter before matching the resource method, you can do so by applying `@PreMatching` annotation on the respective filter class. Same priority rules will be applied to the prematching filters also.

Server Response Filters:

These filters can be used to modify the response level parameters like headers, entity at Server side.

In order to create a response level filter at the server side, add a class by implementing from `ContainerResponseFilter` interface and annotate it with `@Provider` annotation to register it with the JAX-RS runtime. This filter will be applied globally to all the outgoing responses from the REST resource. It has a single method *filter* which will take `ContainerRequestContext`, `ContainerResponseContext` instances as parameters. `ContainerResponseContext` will be used to provide response specific information to the filter.

```
public class ServerResponseFilter implements ContainerResponseFilter {  
    @Override  
    public void filter(ContainerRequestContext requestContext,  
        ContainerResponseContext responseContext) {  
        // Implement your logic here  
    }  
}
```

Unlike `ContainerRequestFilter`, this will be applied even if there is no matching Resource identified (404 error) by the runtime.

If there are multiple filters configured, the filters will be executed in a chain. The order of execution can be controlled by applying `@Priority` annotation to the filter class. Execution will be in the descending order of the `@Priority` annotation value.

Client Request Filters:

These filters can be used to modify the request level parameters like headers at client side.

In order to create a request filter at the client side, add a class by implementing from `ClientRequestFilter` interface and implement the filter method. You need to register this filter manually on `Client/WebTarget` instances.

```
public class ClientRequestFilterTest implements ClientRequestFilter  
{  
    @Override  
    public void filter(ClientRequestContext requestContext) {  
        // Implement your logic here  
    } }  
}
```

Filters registered at Client instance will be available to all the WebTargets created thereafter, but not to the targets created prior to the registration.

Filters registered at WebTarget instance will not be reflected in Client instance. However they will be available for the targets if their parent target is configured with any of the filters.

This filter may abort the request by sending a custom Response from it. In this case, the corresponding filters and resource method execution will be stopped and delegates the request to Response filters.

If there are multiple filters configured, the filters will be executed in a chain. Order of execution will depend on how we register them with Client and WebTarget instances. First registered filter will get executed first and so on. The order of execution can even be controlled by applying @Priority annotation to the filter class. In this case, the registration order will be ignored and filters will get executed in the ascending order of the @Priority annotation value.

The execution order of client level request filters will follow the same rules that of server request filters.

Client Response Filters:

These filters can be used to modify the response level parameters like headers, entity at client side.

In order to create a response filter at the client side, add a class by implementing from ClientResponseFilter interface and implement the filter method. You need to register this filter manually on Client/WebTarget instances.

```
public class ClientResponseFilterTest implements ClientResponseFilter {  
  
    @Override  
  
    public void filter(ClientRequestContext requestContext, ClientResponseContext  
        responseContext) {  
  
        // Implement your logic here  
  
    }  
  
}
```

Same registration rules of Request filters will be applied for response filters as well.

All the client level Request and Response filters will get executed even if the underlying resource is not available.

If there are multiple filters configured, the filters will be executed in a chain. Order of execution will depend on how we register them with Client and WebTarget instances. First registered filter will get executed first and so on. The order of execution can even be controlled by applying

@Priority annotation to the filter class. In this case, the registration order will be ignored and filters will get executed in the descending order of the @Priority annotation value.

Interceptors:

Interceptors are intended to manipulate input and output entity streams on both Client and Server sides. There are two kinds of interceptors, ReaderInterceptor and WriterInterceptor. These two are executed before message body readers or writers execution and their primary intention is to wrap the entity streams that will be used in message body reader and writers.

If request/response is not associated with any entity, Interceptors will not be executed. So it is advised to use the Interceptors for entity streams modification and the filters for modification of headers, media types etc.

You need to annotate the server side interceptor classes with @Provider annotation to register with JAX-RS runtime.

Reader interceptors:

Reader interceptors are used to manipulate inbound entity streams on both Client and Server side.

By using a reader interceptor, you can manipulate request entity stream on the server side when an entity is sent through the client request and response entity stream on the client side when an entity is returned from the server response.

To create a Reader Interceptor, add a class by implementing from ReaderInterceptor and override its method *aroundReadFrom* which takes a single parameter ReaderInterceptorContext, a context class which is capable of modifying the entity stream and delegates the call to MessageBodyReader.readFrom().

```
public class ServerReaderInterceptor implements ReaderInterceptor {  
  
    @Override  
  
    public Object aroundReadFrom(ReaderInterceptorContext  
interceptorContext)  
  
        throws IOException, WebApplicationException {  
  
        // Implement your logic here  
  
    }  
  
}
```

In case multiple reader interceptors are configured, you need to call `ReaderInterceptorContext.proceed()` manually to execute all the filters in a chain. Invocation of this method on the last filter will invoke the appropriate `MessageBodyReader` of Client/Server. Therefore every interceptor must call the `proceed()` method otherwise the entity would not be read.

Writer interceptors:

Writer interceptors are used to manipulate outbound entity streams on both Client and Server side.

By using a writer interceptor, you can manipulate request entity stream on the client side before sending the request (which contains entity) to the server and response entity stream on the server side before returning that entity stream to Client.

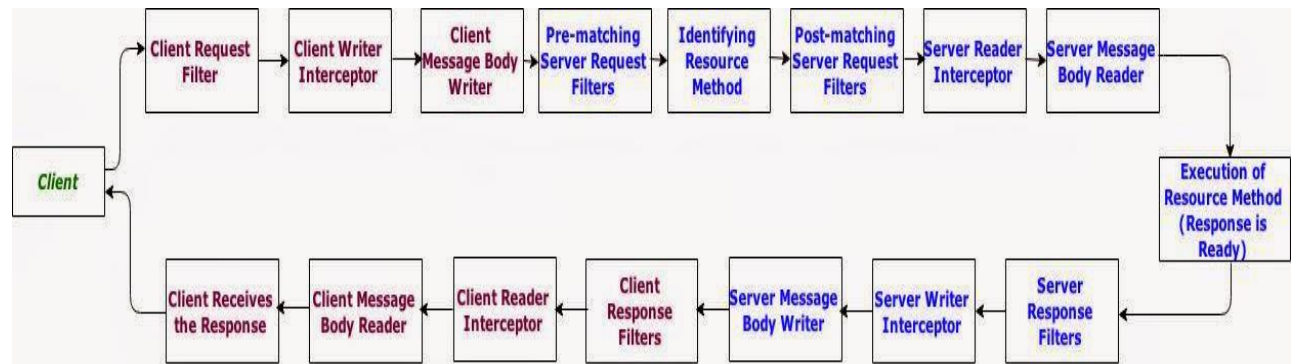
To create a Writer Interceptor, add a class by implementing from `WriterInterceptor` and override its method `aroundWriteTo` which takes a single parameter `WriterInterceptorContext`, a context class which is capable of modifying the entity stream and delegates the call to `MessageBodyWriter.writeTo()`.

```
public class ServerWriterInterceptor implements WriterInterceptor
{
    @Override
    public void aroundWriteTo(WriterInterceptorContext
interceptorContext)
        throws IOException, WebApplicationException {
        // Implement your logic here
    }
}
```

In case multiple writer interceptors are configured, you need to call `WriterInterceptorContext.proceed()` manually to execute all the filters in a chain. Invocation of this method on the last filter will invoke the appropriate `MessageBodyWriter` of Client/Server. Therefore every interceptor must call the `proceed()` method otherwise the entity would not be written.

In case of multiple interceptors, the order of execution can be controlled by applying `@Priority` annotation to the class. Interceptors will be executed in the ascending order of `@Priority` annotation. For Client side interceptors, in the absence of `@Priority` value, they will be executed in the same order you register them with Client/WebTarget.

Below diagram will show the execution order of Filters and Interceptors in a JAX-RS application.



JAX-RS Interceptors Example

how to create and configure a RESTful application based on the underlying implementation of JAX-RS specification.

web.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xmlns="http://java.sun.com/xml/ns/javaee"
4  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"
5  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6  http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"
7  id="WebApp_ID" version="2.5">
8  <description>Testing Interceptors in JAX-RS</description>
9  <servlet>
10 <servlet-name>restcontainer</servlet-name>
11 <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
12 <init-param>
13 <param-name>jersey.config.server.provider.packages</param-name>
14 <param-value>com.javatech.rest.interceptor.resource,com.javatech.rest.interceptors.providers</param-value>
15 </init-param>
16 </servlet>
  
```

```
17 <servlet-mapping>
18 <servlet-name>restcontainer</servlet-name>
19 <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
    </web-app>
```

InterceptorResource.java

```
1  package com.javatech.rest.interceptor.resource;
2
3  import javax.ws.rs.POST;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.core.Response;
6  import javax.ws.rs.core.Response.Status;
7
8  @Path("interceptor")
9  public class InterceptorResource {
10
11      @POST
12      public Response testInterceptor(String inputMessage) {
13          System.out.println("Input Message: " + inputMessage);
14
15          Response response = Response.ok("\nOrder successfully
placed")
16              .status(Status.OK).build();
17
18          return response;
19      }
20  }
```

ServerReaderInterceptor1.java

```
1  package com.javatech.rest.interceptors.providers;
2
3  import java.io.ByteArrayInputStream;
4  import java.io.IOException;
5  import java.io.InputStream;
6
7  import javax.annotation.Priority;
8  import javax.ws.rs.WebApplicationException;
9  import javax.ws.rs.ext.Provider;
10 import javax.ws.rs.ext.ReaderInterceptor;
11 import javax.ws.rs.ext.ReaderInterceptorContext;
12
13 @Priority(value = 1000)
14 @Provider
15 public class ServerReaderInterceptor1 implements ReaderInterceptor {
16
17     @Override
18     public Object aroundReadFrom(ReaderInterceptorContext interceptorContext)
19         throws IOException, WebApplicationException {
20         System.out.println("ServerReaderInterceptor1 invoked");
21         InputStream inputStream = interceptorContext.getInputStream();
22         byte[] bytes = new byte[inputStream.available()];
23         inputStream.read(bytes);
24         String requestContent = new String(bytes);
25         requestContent = requestContent
26             + "\nRequest changed in ServerReaderInterceptor1.";
27         interceptorContext.setInputStream(new ByteArrayInputStream(
```



```
28         requestContent.getBytes()));
29     return interceptorContext.proceed();
30 }
31 }
```

ServerReaderInterceptor2.java

```
1  package com.javatech.rest.interceptors.providers;
2
3  import java.io.ByteArrayInputStream;
4  import java.io.IOException;
5  import java.io.InputStream;
6
7  import javax.annotation.Priority;
8  import javax.ws.rs.WebApplicationException;
9  import javax.ws.rs.ext.Provider;
10 import javax.ws.rs.ext.ReaderInterceptor;
11 import javax.ws.rs.ext.ReaderInterceptorContext;
12
13 @Priority(value = 2000)
14 @Provider
15 public class ServerReaderInterceptor2 implements ReaderInterceptor {
16
17     @Override
18     public Object aroundReadFrom(ReaderInterceptorContext
19 interceptorContext)
20         throws IOException, WebApplicationException {
21         System.out.println("ServerReaderInterceptor2 invoked.");
22     }
23 }
```

```

22     InputStream inputStream = interceptorContext.getInputStream();
23     byte[] bytes = new byte[inputStream.available()];
24     inputStream.read(bytes);
25     String requestContent = new String(bytes);
26     requestContent = requestContent
27         + "\nRequest changed in ServerReaderInterceptor2.";
28     interceptorContext.setInputStream(new ByteArrayInputStream(
29         requestContent.getBytes()));
30     return interceptorContext.proceed();
31 }
    }

```

ServerWriterInterceptor1.java

```

1  package com.javatech.rest.interceptors.providers;
2
3  import java.io.IOException;
4  import java.io.OutputStream;
5
6  import javax.annotation.Priority;
7  import javax.ws.rs.WebApplicationException;
8  import javax.ws.rs.ext.Provider;
9  import javax.ws.rs.ext.WriterInterceptor;
10 import javax.ws.rs.ext.WriterInterceptorContext;
11
12 @Priority(value = 1000)
13 @Provider
14 public class ServerWriterInterceptor1 implements WriterInterceptor {
15

```

```

16     @Override
17     public void aroundWriteTo(WriterInterceptorContext interceptorContext)
18         throws IOException, WebApplicationException {
19         System.out.println("ServerWriterInterceptor1 invoked");
20         OutputStream outputStream = interceptorContext.getOutputStream();
21         String responseContent = "\nResponse changed in
22 ServerWriterInterceptor1.";
23         outputStream.write(responseContent.getBytes());
24         interceptorContext.setOutputStream(outputStream);
25
26         interceptorContext.proceed();
27     }
28 }

```

ServerWriterInterceptor2.java

```

1     package com.javatech.rest.interceptors.providers;
2
3     import java.io.IOException;
4     import java.io.OutputStream;
5
6     import javax.annotation.Priority;
7     import javax.ws.rs.WebApplicationException;
8     import javax.ws.rs.ext.Provider;
9     import javax.ws.rs.ext.WriterInterceptor;
10    import javax.ws.rs.ext.WriterInterceptorContext;
11
12    @Priority(value = 2000)
13    @Provider

```

```

14 public class ServerWriterInterceptor2 implements WriterInterceptor {
15
16     @Override
17     public void aroundWriteTo(WriterInterceptorContext interceptorContext)
18         throws IOException, WebApplicationException {
19         System.out.println("ServerWriterInterceptor2 invoked.");
20         OutputStream outputStream = interceptorContext.getOutputStream();
21         String responseContent = "\nResponse changed in
22 ServerWriterInterceptor2.";
23         outputStream.write(responseContent.getBytes());
24         interceptorContext.setOutputStream(outputStream);
25
26         interceptorContext.proceed();
27     }
28 }

```

JAX-RS Interceptors Client Example

InterceptorClient.java

```

1 package com.javatech.rest.interceptor.client;
2
3 import javax.ws.rs.client.Client;
4 import javax.ws.rs.client.ClientBuilder;
5 import javax.ws.rs.client.Entity;
6 import javax.ws.rs.client.WebTarget;
7 import javax.ws.rs.core.Response;
8
9 import com.javatech.rest.interceptors.providers.ClientReaderInterceptor1;
10 import com.javatech.rest.interceptors.providers.ClientReaderInterceptor2;
11 import com.javatech.rest.interceptors.providers.ClientWriterInterceptor1;

```

```

12  import com.javatech.rest.interceptors.providers.ClientWriterInterceptor2;
13
14  public class InterceptorClient {
15      public static void main(String[] args) {
16          Client client = ClientBuilder.newClient();
17          WebTarget target = client
18              .target("http://localhost:7001/JAXRSInterceptors/rest/interceptor");
19
20          target = target.queryParam("UserName", "admin");
21          target.register(ClientWriterInterceptor2.class);
22          target.register(ClientWriterInterceptor1.class);
23          target.register(ClientReaderInterceptor2.class);
24          target.register(ClientReaderInterceptor1.class);
25          Response response = target.request()
26              .buildPost(Entity.text("Request Content.")).invoke();
27          if (response.hasEntity()) {
28              System.out.println("Response Status: " + response.getStatus()
29                  + "\nResponse Content: "
30                  + response.readEntity(String.class));
31          } else
32              System.out.println("No Response");
33      }
34  }

```

ClientWriterInterceptor1.java

```
1      package com.javatech.rest.interceptors.providers;
2
3      import java.io.IOException;
4      import java.io.OutputStream;
5
6      import javax.annotation.Priority;
7      import javax.ws.rs.WebApplicationException;
8      import javax.ws.rs.ext.WriterInterceptor;
9      import javax.ws.rs.ext.WriterInterceptorContext;
10
11     @Priority(value = 1000)
12     public class ClientWriterInterceptor1 implements WriterInterceptor {
13
14         @Override
15         public void aroundWriteTo(WriterInterceptorContext
16             interceptorContext)
17             throws IOException, WebApplicationException {
18             System.out.println("ClientWriterInterceptor1 invoked");
19             OutputStream outputStream = interceptorContext.getOutputStream();
20             String requestContent = "\nRequest changed in
21 ClientWriterInterceptor1.";
22             outputStream.write(requestContent.getBytes());
23             interceptorContext.setOutputStream(outputStream);
24
25             interceptorContext.proceed();
26         }
27     }
```

ClientWriterInterceptor2.java

```
1    package com.javatech.rest.interceptors.providers;
2
3    import java.io.IOException;
4    import java.io.OutputStream;
5
6    import javax.annotation.Priority;
7    import javax.ws.rs.WebApplicationException;
8    import javax.ws.rs.ext.WriterInterceptor;
9    import javax.ws.rs.ext.WriterInterceptorContext;
10
11    @Priority(value = 2000)
12    public class ClientWriterInterceptor2 implements WriterInterceptor {
13
14        @Override
15        public void aroundWriteTo(WriterInterceptorContext
16        interceptorContext)
17            throws IOException, WebApplicationException {
18            System.out.println("ClientWriterInterceptor2 invoked");
19            OutputStream outputStream = interceptorContext.getOutputStream();
20            String requestContent = "\nRequest changed in
21            ClientWriterInterceptor2.";
22            outputStream.write(requestContent.getBytes());
23            interceptorContext.setOutputStream(outputStream);
24
25            interceptorContext.proceed();
26        }
27    }
```

```
}
```

ClientReaderInterceptor1.java

```
1    package com.javatech.rest.interceptors.providers;
2
3    import java.io.ByteArrayInputStream;
4    import java.io.IOException;
5    import java.io.InputStream;
6
7    import javax.annotation.Priority;
8    import javax.ws.rs.WebApplicationException;
9    import javax.ws.rs.ext.ReaderInterceptor;
10   import javax.ws.rs.ext.ReaderInterceptorContext;
11
12   @Priority(value = 1000)
13   public class ClientReaderInterceptor1 implements ReaderInterceptor {
14
15       @Override
16       public Object aroundReadFrom(ReaderInterceptorContext
17       interceptorContext)
18           throws IOException, WebApplicationException {
19           System.out.println("ClientReaderInterceptor1 invoked");
20           InputStream inputStream = interceptorContext.getInputStream();
21           byte[] bytes = new byte[inputStream.available()];
22           inputStream.read(bytes);
23           String responseContent = new String(bytes);
24           responseContent = responseContent
25               + "\nResponse changed in ClientReaderInterceptor1.";
```



```
26         interceptorContext.setInputStream(new ByteArrayInputStream(
27             responseContent.getBytes()));
28
29         return interceptorContext.proceed();
30     }
    }
```

ClientReaderInterceptor2.java

```
1     package com.javatech.rest.interceptors.providers;
2
3     import java.io.ByteArrayInputStream;
4     import java.io.IOException;
5     import java.io.InputStream;
6
7     import javax.annotation.Priority;
8     import javax.ws.rs.WebApplicationException;
9     import javax.ws.rs.ext.ReaderInterceptor;
10    import javax.ws.rs.ext.ReaderInterceptorContext;
11
12    @Priority(value = 2000)
13    public class ClientReaderInterceptor2 implements ReaderInterceptor {
14
15        @Override
16        public Object aroundReadFrom(ReaderInterceptorContext
17            interceptorContext)
18            throws IOException, WebApplicationException {
19            System.out.println("ClientReaderInterceptor2 invoked");
20            InputStream inputStream = interceptorContext.getInputStream();
```

```
21     byte[] bytes = new byte[inputStream.available()];
22     inputStream.read(bytes);
23     String responseContent = new String(bytes);
24     responseContent = responseContent
25         + "\nResponse changed in ClientReaderInterceptor2.";
26     interceptorContext.setInputStream(new ByteArrayInputStream(
27         responseContent.getBytes()));
28
29     return interceptorContext.proceed();
30 }
}
```

JAX-RS Filters Client Example

FiltersClient.java

```
1  package com.javatech.rest.filters.client;
2
3  import javax.ws.rs.client.Client;
4  import javax.ws.rs.client.ClientBuilder;
5  import javax.ws.rs.client.Invocation;
6  import javax.ws.rs.client.WebTarget;
7  import javax.ws.rs.core.Response;
8
9  import com.javatech.rest.filters.providers.ClientRequestFilterTest1;
10 import com.javatech.rest.filters.providers.ClientRequestFilterTest2;
11 import com.javatech.rest.filters.providers.ClientResponseFilterTest1;
12 import com.javatech.rest.filters.providers.ClientResponseFilterTest2;
13
```

```
14 public class FiltersClient {
15     public static void main(String[] args) {
16         System.out.println("Registering Client Level Filters");
17
18         Client client = ClientBuilder.newClient();
19         client.register(new ClientResponseFilterTest2());
20         WebTarget target = client
21             .target("http://localhost:7001/RestfulServiceFilters/rest");
22         target.register(new ClientRequestFilterTest2());
23
24         WebTarget resourceTarget = target.path("/select");
25         resourceTarget = resourceTarget.queryParam("OrderID", "111")
26             .queryParam("UserName", "VaraPrasad");
27         resourceTarget.register(new ClientResponseFilterTest1());
28         resourceTarget.register(new ClientRequestFilterTest1());
29
30         System.out.println("Invoking REST Service: "
31             + resourceTarget.getUri().toString());
32         Invocation invocation = resourceTarget.request().buildGet();
33         Response response = invocation.invoke();
34         String respContent = "";
35
36         if (response.hasEntity())
37             respContent = response.readEntity(String.class);
38
39         System.out.println("Response--> " + respContent);
40         System.out
41             .println("Content Type after changing in ClientResponseFilter: "
```

```
42         + response.getHeaderString("Content-Type"));
43     }
44 }
```

ClientRequestFilterTest1.java

```
1     package com.javatech.rest.filters.providers;
2
3     import javax.annotation.Priority;
4     import javax.ws.rs.client.ClientRequestContext;
5     import javax.ws.rs.client.ClientRequestFilter;
6
7     @Priority(value = 1)
8     public class ClientRequestFilterTest1 implements
9         ClientRequestFilter {
10
11         public ClientRequestFilterTest1() {
12             System.out.println("ClientRequestFilterTest1 Instantiation");
13         }
14
15         @Override
16         public void filter(ClientRequestContext requestContext) {
17             System.out.println("filter() on ClientRequestFilterTest1");
18             requestContext.getHeaders().add("Content-Type", "text/html;");
19             // requestContext.abortWith(Response.ok("Not Valid")
20             // .status(Status.BAD_REQUEST).build());
21         }
22     }
```

ClientRequestFilterTest2.java

```
1    package com.javatech.rest.filters.providers;
2
3    import javax.annotation.Priority;
4    import javax.ws.rs.client.ClientRequestContext;
5    import javax.ws.rs.client.ClientRequestFilter;
6
7    @Priority(value = 2)
8    public class ClientRequestFilterTest2 implements
       ClientRequestFilter {
9
10   public ClientRequestFilterTest2() {
11       System.out.println("ClientRequestFilterTest2 Instantiation");
12   }
13
14   @Override
15   public void filter(ClientRequestContext requestContext) {
16       System.out.println("filter() on ClientRequestFilterTest2");
17   }
18   }
```

ClientResponseFilterTest1.java

```
1    package com.javatech.rest.filters.providers;
2
3    import javax.annotation.Priority;
4    import javax.ws.rs.client.ClientRequestContext;
```

```
5    import javax.ws.rs.client.ClientResponseContext;
6    import javax.ws.rs.client.ClientResponseFilter;
7
8    @Priority(value = 2)
9    public class ClientResponseFilterTest1 implements
10    ClientResponseFilter {
11
12    public ClientResponseFilterTest1() {
13        System.out.println("ClientResponseFilterTest1 Instantiation");
14    }
15
16    @Override
17    public void filter(ClientRequestContext requestContext,
18        ClientResponseContext responseContext) {
19        System.out.println("filter() on ClientResponseFilterTest1");
20        responseContext.getHeaders().putSingle("Content-Type",
21        "text/plain;");
22    }
23    }
```

ClientResponseFilterTest2.java

```
1    package com.javatech.rest.filters.providers;
2
3    import javax.annotation.Priority;
4    import javax.ws.rs.client.ClientRequestContext;
5    import javax.ws.rs.client.ClientResponseContext;
6    import javax.ws.rs.client.ClientResponseFilter;
7
8    @Priority(value = 1)
```

```

9  public class ClientResponseFilterTest2 implements
10 ClientResponseFilter {
11
12  public ClientResponseFilterTest2() {
13      System.out.println("ClientResponseFilterTest2 Instantiation");
14  }
15
16  @Override
17  public void filter(ClientRequestContext requestContext,
18      ClientResponseContext responseContext) {
19      System.out.println("filter() on ClientResponseFilterTest2");
20  }
21  }

```

JAX-RS Filters Example

how to create and configure a RESTful application based on the underlying implementation of JAX-RS specification.

web.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xmlns="http://java.sun.com/xml/ns/javaee"
4      xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"
5      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6      http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"
7      id="WebApp_ID" version="2.5">
8      <description>Testing Filters in JAX-RS</description>
9      <servlet>
10         <servlet-name>restcontainer</servlet-name>
11         <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
12         <init-param>

```

```
13     <param-name>jersey.config.server.provider.packages</param-name>
14     <param-
15 value>com.javatech.rest.filters.resource,com.javatech.rest.filters.providers</param-value>
16     </init-param>
17 </servlet>
18 <servlet-mapping>
19     <servlet-name>restcontainer</servlet-name>
20     <url-pattern>/rest/*</url-pattern>
21 </servlet-mapping>
22 </web-app>
```

SelectOrderResource.java

```
1  package com.javatech.rest.filters.resource;
2
3  import javax.ws.rs.GET;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.QueryParam;
6  import javax.ws.rs.core.Context;
7  import javax.ws.rs.core.HttpHeaders;
8  import javax.ws.rs.core.Response;
9  import javax.ws.rs.core.Response.Status;
10 @Path("/select")
11 public class SelectOrderResource {
12
13     public SelectOrderResource() {
14         System.out.println("Instantiation of SelectOrderResource");
15     }
16     @Context
17     private HttpHeaders headers;
```



```
18     public HttpHeaders getHeaders() {
19         return headers;
20     }
21
22     public void setHeaders(HttpHeaders headers) {
23         this.headers = headers;
24     }
25
26     @GET
27     public Response getOrder(@QueryParam("OrderID") String orderId) {
28         String responseContent = "";
29         responseContent = "Content Type: "
30             + getHeaders().getHeaderString("Content-Type");
31
32         Response response = null;
33         if (orderId != null && !"".equals(orderId))
34             responseContent = responseContent + "\nOrder Successfully placed.";
35         else
36             responseContent = responseContent + "\nInvalid Order.";
37
38         response = Response.ok(responseContent).status(Status.OK).build();
39         System.out.println("Resource Method executed. Response: "
40             + response.toString());
41
42         return response;
43     }
44 }
45
```

ServerAuthenticationRequestFilter.java

```
1    package com.javatech.rest.filters.providers;
2
3    import javax.annotation.Priority;
4    import javax.ws.rs.container.ContainerRequestContext;
5    import javax.ws.rs.container.ContainerRequestFilter;
6    import javax.ws.rs.container.PreMatching;
7    import javax.ws.rs.core.Response;
8    import javax.ws.rs.core.Response.ResponseBuilder;
9    import javax.ws.rs.core.Response.Status;
10   import javax.ws.rs.ext.Provider;
11
12   @PreMatching
13   @Priority(value = 3)
14   @Provider
15   public class ServerAuthenticationRequestFilter implements
16       ContainerRequestFilter {
17
18       public ServerAuthenticationRequestFilter() {
19           System.out.println("ServerAuthenticationRequestFilter
20 initialization");
21       }
22
23       @Override
24       public void filter(ContainerRequestContext requestContext) {
25           ResponseBuilder responseBuilder = null;
```

```

26     Response response = null;
27     String userName = null;
28     System.out.println("filter() on
29     ServerAuthenticationRequestFilter");
30
31     userName = requestContext.getUriInfo().getQueryParameters()
32         .getFirst("UserName");
33     if (userName == null || "".equals(userName)) {
34         System.out.println("Authentication Filter Failed");
35         responseBuilder = Response.serverError();
36         response =
37         responseBuilder.status(Status.BAD_REQUEST).build();
38         requestContext.abortWith(response);
39     } else
40         System.out.println("Authentication Filter Passed; UserName is "
41             + userName);
42     }
43 }

```

ServerRequestFilter2.java

```

1     package com.javatech.rest.filters.providers;
2
3     import javax.annotation.Priority;
4     import javax.ws.rs.container.ContainerRequestContext;
5     import javax.ws.rs.container.ContainerRequestFilter;
6     import javax.ws.rs.ext.Provider;
7
8     @Provider

```

```
9    @Priority(value = 1)
10   public class ServerRequestFilter2 implements
11   ContainerRequestFilter {
12
13       public ServerRequestFilter2() {
14           System.out.println("ServerRequestFilter2 initialization");
15       }
16
17       @Override
18       public void filter(ContainerRequestContext requestContext) {
19           System.out.println("filter() on ServerRequestFilter2");
20       }
21   }
```

ServerRequestFilter3.java

```
1   package com.javatech.rest.filters.providers;
2
3   import javax.annotation.Priority;
4   import javax.ws.rs.container.ContainerRequestContext;
5   import javax.ws.rs.container.ContainerRequestFilter;
6   import javax.ws.rs.ext.Provider;
7
8   @Provider
9   @Priority(value = 2)
10  public class ServerRequestFilter3 implements
11  ContainerRequestFilter {
12
13      public ServerRequestFilter3() {
```

```
13    System.out.println("ServerRequestFilter3 initialization");
14    }
15
16    @Override
17    public void filter(ContainerRequestContext requestContext) {
18        System.out.println("filter() on ServerRequestFilter3");
19    }
20    }
```

ServerResponseFilter.java

```
1    package com.javatech.rest.filters.providers;
2
3    import javax.annotation.Priority;
4    import javax.ws.rs.container.ContainerRequestContext;
5    import javax.ws.rs.container.ContainerResponseContext;
6    import javax.ws.rs.container.ContainerResponseFilter;
7    import javax.ws.rs.ext.Provider;
8
9    @Provider
10    @Priority(value = 1)
11    public class ServerResponseFilter implements
12    ContainerResponseFilter {
13
14    public ServerResponseFilter() {
15        System.out.println("ServerResponseFilter initialization");
16    }
17    }
```

```
18  @Override
19  public void filter(ContainerRequestContext requestContext,
20      ContainerResponseContext responseContext) {
21      System.out.println("filter() on ServerResponseFilter");
22
23      if ("/select".equals(requestContext.getUriInfo().getPath())
24          && responseContext.hasEntity()) {
25          responseContext.setEntity(responseContext.getEntity().toString()
26              + "\nResponse changed in Response Filter");
27      }
28  }
    }
```

ServerResponseFilter2.java

```
1  package com.javatech.rest.filters.providers;
2
3  import javax.annotation.Priority;
4  import javax.ws.rs.container.ContainerRequestContext;
5  import javax.ws.rs.container.ContainerResponseContext;
6  import javax.ws.rs.container.ContainerResponseFilter;
7  import javax.ws.rs.ext.Provider;
8
9  @Provider
10 @Priority(value = 2)
11 public class ServerResponseFilter2 implements
12     ContainerResponseFilter {
13
```

```
14 public ServerResponseFilter2() {
15     System.out.println("ServerResponseFilter2 initialization");
16 }
17
18 @Override
19 public void filter(ContainerRequestContext requestContext,
20     ContainerResponseContext responseContext) {
21     System.out.println("filter() on ServerResponseFilter2");
22
23     if ("/select".equals(requestContext.getUriInfo().getPath())
24         && responseContext.hasEntity()) {
25         responseContext.setEntity(responseContext.getEntity().toString()
26             + "\nResponse changed in Response Filter2");
27     }
28 }
}
```

ServerResponseFilter3.java

```
1 package com.javatech.rest.filters.providers;
2
3 import javax.annotation.Priority;
4 import javax.ws.rs.container.ContainerRequestContext;
5 import javax.ws.rs.container.ContainerResponseContext;
6 import javax.ws.rs.container.ContainerResponseFilter;
7 import javax.ws.rs.ext.Provider;
8
9 @Provider
```

```
10  @Priority(value = 3)
11  public class ServerResponseFilter3 implements
12  ContainerResponseFilter {
13
14  public ServerResponseFilter3() {
15      System.out.println("ServerResponseFilter3 initialization");
16  }
17
18  @Override
19  public void filter(ContainerRequestContext requestContext,
20      ContainerResponseContext responseContext) {
21      System.out.println("filter() on ServerResponseFilter3");
22
23      if ("/select".equals(requestContext.getUriInfo().getPath())
24          && responseContext.hasEntity()) {
25          responseContext.setEntity(responseContext.getEntity().toString()
26              + "\nResponse changed in Response Filter3");
27      }
28  }
29  }
```