Les filtres et Listeners Web

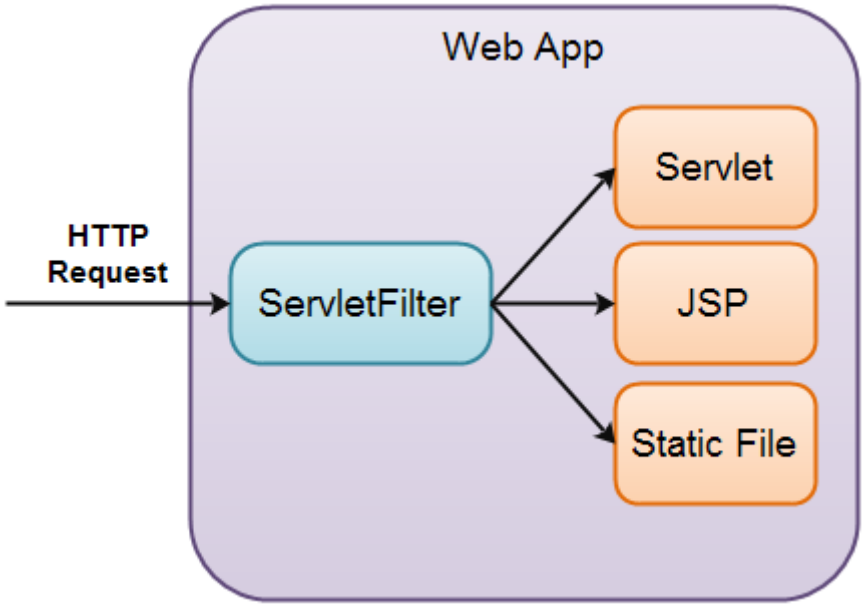
1. Les filtres de Servlet

Un filtre de servlet est un objet java qui peut intercepter les requêtes HTTP ciblées sur votre application Web.
Les filtres sont des classes Java qui peuvent intercepter les requêtes d'un client avant d'accéder à une ressource; manipuler les demandes des clients; intercepter les réponses provenant des ressources avant qu'elles ne soient renvoyées au client; et manipuler les réponses avant qu'elles ne soient envoyées au client.

Les filtres ont de nombreuses utilisations :

- filtres d'authentification
- journalisation et audit des filtres
- filtres de conversion d'image
- filtres de compression de données
- filtres de chiffrement
- les filtres qui déclenchent les événements d'accès aux ressources
- Filtres de chaîne de type MIME

Un filtre de servlet peut intercepter des demandes à la fois pour des servlets, des JSP, des fichiers HTML ou d'autres contenus statiques, comme illustré dans le diagramme ci-dessous:



Un filtre de servlet dans une application Web Java

Pour créer un filtre de servlet, vous devez implémenter l'interface javax.servlet.Filter . Voici un exemple de mise en œuvre de filtre de servlet:

```
 import javax.servlet. *;
 import java.io.IOException;

 / **

  * /
 classe publique SimpleServletFilter implements Filter {

    public void init (FilterConfig filterConfig) throws ServletException {
    }
```

```
    public void doFilter (requête ServletRequest, réponse ServletResponse,
                    FilterChain filterChain)
    throws IOException, ServletException {

    }

    public void destroy () {
    }
}
```

Lorsque le filtre de servlet est chargé la première fois, sa méthode init() est appelée, comme avec les servlets.
Lorsqu'une requête HTTP arrive à votre application Web que le filtre intercepte, le filtre peut inspecter l'URI de la requête, les paramètres de la requête et les en-têtes de la requête et décider s'il veut bloquer ou transmettre la requête au servlet cible, JSP etc.
C'est la méthode doFilter() qui effectue l'interception. Voici un exemple de mise en œuvre:

```
public void doFilter(ServletRequest request, ServletResponse response,
                FilterChain filterChain)
throws IOException, ServletException {

   String myParam = request.getParameter("myParam");

   if(!"blockTheRequest".equals(myParam)){
      filterChain.doFilter(request, response);
   }
}
```

Remarquez comment la méthode doFilter() vérifie un paramètre de requête, myParam , pour voir s'il est égal à la chaîne "blockTheRequest". Sinon, la requête est transmise à la cible de la requête en appelant la méthode filterChain.doFilter() . Si cette méthode n'est pas appelée, la requête n'est pas transmise, mais simplement bloquée.
Le filtre de servlet ci-dessus ignore simplement la requête si le paramètre de requête myParam est égal à "blockTheRequest". Vous pouvez également écrire une réponse différente dans le navigateur en utilisant l'objet ServletResponse .
On peut caster l'objet ServletResponse en HttpResponse pour en obtenir un PrintWriter . Sinon, vous n'avez que l'objet OutputStream disponible via getOutputStream() .
Voici un exemple:

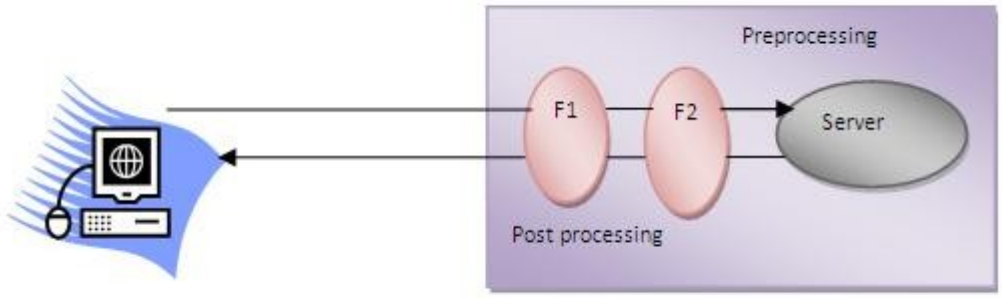
```
public void doFilter(ServletRequest request, ServletResponse response,
                FilterChain filterChain)
throws IOException, ServletException {

   String myParam = request.getParameter("myParam");

   if(!"blockTheRequest".equals(myParam)){
      filterChain.doFilter(request, response);
      return;
   }

   HttpResponse httpResponse = (HttpResponse) httpResponse;
    httpResponse.getWriter (). write ("une réponse différente ... par exemple en HTML");
  //  ou bien se rédiriger vers une autre page ou servlet
  HttpServletResponse resp = (HttpServletResponse) response;
   resp.sendRedirect(/autreservlet);



}
```

Configuration du filtre de servlet dans web.xml

Vous devez configurer le filtre de servlet dans le fichier web.xml de votre application Web, avant qu'il ne fonctionne.

```
<filter>
   <filter-name>myFilter</filter-name>
   <filter-class>servlets.SimpleServletFilter</filter-class>
</filter>

<filter-mapping>
   <filter-name>myFilter</filter-name>
   <url-pattern>*.simple</url-pattern>
</filter-mapping>
```

Avec cette configuration, toutes les requêtes dont l'URL se termine par .simple seront interceptées par le filtre de servlet. Tous les autres seront laissés intacts.



Il est parfois intéressant d'effectuer des opérations avant et/ou après l'invocation de la servlet. Il s'agit souvent d'opérations communes à un ensemble de requêtes d'une application Web.
Un filtre est un composant d'une application web qui agit comme un intercepteur sur une servlet. Il est déclaré dans le descripteur de l'application web.xml, et posé sur une ou plusieurs servlets. Lorsqu'une requête doit être traitée par une servlet sur laquelle un filtre est déclaré, alors le serveur, plutôt que d'invoquer directement la méthode doGet() ou doPost() de la servlet, va invoquer la méthode doFilter() de ce filtre.
Cette méthode reçoit trois paramètres :

- les deux premiers sont les objets requête et réponse qui auraient été envoyés à la servlet ;
- le troisième est un objet de type FilterChain, qui modèlise la servlet interceptée.

Le filtre peut donc agir sur la requête, qui est complète. S'il choisit d'autoriser le traitement de cette requête par la servlet, alors il doit invoquer la méthode doFilter() de l'objet FilterChain, en lui passant la requête et la réponse qu'il a reçues en paramètre.
Il peut aussi choisir de ne pas traiter cette requête et de la rediriger vers une autre ressource de l'application web.
Un filtre de Servlet est une classe implémentant l'interface Filter. Un filtre a son propre cycle de vie. Une fois créé, le conteneur initialise le filtre en appelant sa méthode init et il signalera la destruction du filtre en appelant sa méthode destroy. L'opération de filtrage est réalisée grâce à la méthode doFilter.
Exemple d'implémentation d'un filtre Web

```
package fr.epsi;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
```

```
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MyFilter implements Filter {

  @Override
  public void init(FilterConfig filterConfig) throws ServletException {
    // ...
  }

  @Override
  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
      throws IOException, ServletException {
    // ...
  }

  @Override
  public void destroy() {
    // ...
  }

}
```

Déclaration des filtres

La déclaration d'un filtre Web auprès du conteneur se fait soit par l'annotation **@WebFilter** soit dans le fichier de déploiement **web.xml**.
Comme pour une Servlet, un filtre est associé à un ou plusieurs motifs d'URL (URL pattern) indiquant au conteneur pour quelles requêtes HTTP le filtre doit être appelé.
Déclaration d'un filtre Web par annotation

```
package fr.epsi;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;

@WebFilter({"/subpart/*", "/otherpart/*"})
public class MyFilter implements Filter {

  @Override
  public void init(FilterConfig filterConfig) throws ServletException {
    // ...
  }

  @Override
  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
      throws IOException, ServletException {
    // ...
```

```
  }

  @Override
  public void destroy() {
    // ...
  }

}
```

Si l'**on ne souhaite pas utiliser une annotation**, il est également possible de déclarer un listener dans le fichier de déploiement web.xml grâce aux balises filter et filter-mapping.
Déclaration d'un filtre dans le fichier web.xml

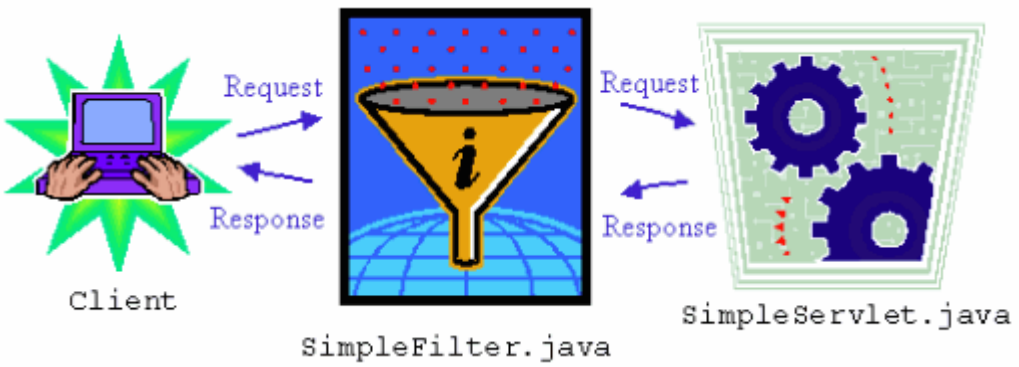
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                      http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  version="3.0">

  <filter>
            <filter-name>MyFilter</filter-name>
            <filter-class>fr.epsi.MyFilter</filter-class>
  </filter>

  <filter-mapping>
            <filter-name>MyFilter</filter-name>
            <url-pattern>/subpart/*</url-pattern>
  </filter-mapping>

  <filter-mapping>
            <filter-name>MyFilter</filter-name>
            <url-pattern>/otherpart/*</url-pattern>
  </filter-mapping>
</web-app>
```

Servlet-Filter url-pattern

TIl ya 3 facons de configure **url-pattern** pour Filter:

URL Pattern	Exemple
/*	http://example.com/contextPath
	http://example.com/contextPath/status/abc
/status/abc/*	http://example.com/contextPath/status/abc
	http://example.com/contextPath/status/abc/mnp
	http://example.com/contextPath/status/abc/mnp?date=today
	~~http://example.com/contextPath/test/abc/mnp~~
*.map	http://example.com/contextPath/status/abc.map
	http://example.com/contextPath/status.map?date=today
	~~http://example.com/contextPath/status/abc.MAP~~

Il est également possible de déclarer qu'un filtre doit être utilisé pour des requêtes traitées par des Servlets spécifiques plutôt que d'utiliser un modèle d'URL.
Exemple 1 : filtre très simple qui imprime un message à la console du serveur Web pendant qu'il filtre la requête, appelle le Servlet, puis imprime un autre message sur la console pendant qu'il filtre la réponse. La figure 1 est un diagramme montrant comment le filtre simple s'adapte au modèle de demande-réponse du servlet.



Listing 1: Un exemple de filtre très simple (SimpleFilter.java)

```
package com.filters;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

import java.io.IOException;
import javax.servlet.ServletException;

public class SimpleFilter implements Filter
{
    public  void init(FilterConfig filterConfig) { }
public void doFilter (ServletRequest request,
ServletResponse response,
FilterChain chain)
{

try
{
System.out.print ("Within Simple Filter ... ");
System.out.println ("Filtering the Request ...");

chain.doFilter (request, response);

System.out.print ("Within Simple Filter ... ");
System.out.println ("Filtering the Response ...");

} catch (IOException io) {
System.out.println ("IOException raised in SimpleFilter");
} catch (ServletException se) {
System.out.println ("ServletException raised in SimpleFilter");
}
}
```

Déploiement de filtres dans Tomcat

. Le Listing 2 montre le fichier web.xml qui a déployé le SimpleFilter.java le mappant à un Servlet.
Listing 2: Le descripteur de déploiement pour le filtre simple. (web.xml)

```
<?xml version = "1.0" encoding = "ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">

<web-app>

<!-- Define the filters within the Web Application -->

<filter>
<filter-name>
Simple Filter Example
</filter-name>
<filter-class>
com.filters.SimpleFilter
</filter-class>
</filter>


<!-- Map the filter to a Servlet or URL -->

<filter-mapping>
<filter-name>
Simple Filter Example
</filter-name>
<url-pattern>
/simple
</url-pattern>
</filter-mapping>


<!-- Define the Servlets within the Web Application -->

<servlet>
<servlet-name>
Simple Servlet
</servlet-name>
<servlet-class>
com.servlets.SimpleServlet
</servlet-class>
</servlet>


<!-- Define Servlet mappings to urls -->

<servlet-mapping>
<servlet-name>
Simple Servlet
</servlet-name>
<url-pattern>
/simple
</url-pattern>
</servlet-mapping>

</web-app>
```

La figure 2 montre la console du serveur Tomcat sur laquelle cette application Web s'exécutait après que la servlet ait traité une demande

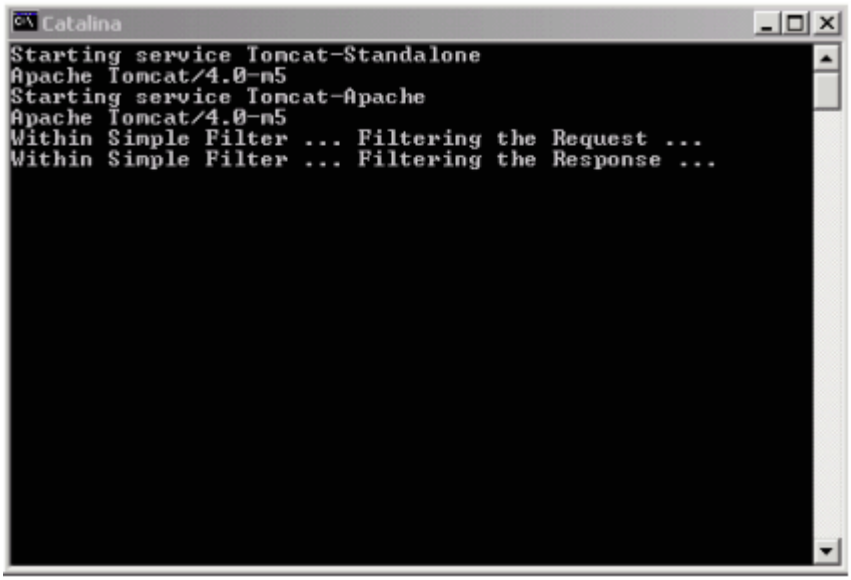


Figure 2. Console de Tomcat exécutant le filtre simple.

Chaînage de filtres

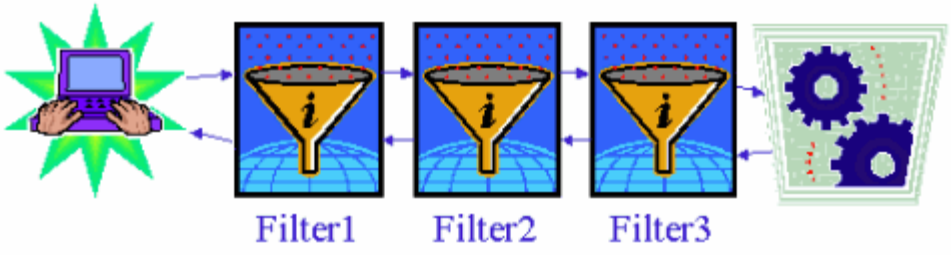


Figure 3. Un diagramme de chaînage de filtres.

Plusieurs filtres peuvent être écrits et appliqués au même modèle d'URL. L'ordre d'exécution est déterminé par l'ordre dans le descripteur de déploiement. La méthode doFilter() permet d' appele l'élément suivant dans la chaîne avec

chain.doFilter(request, response)

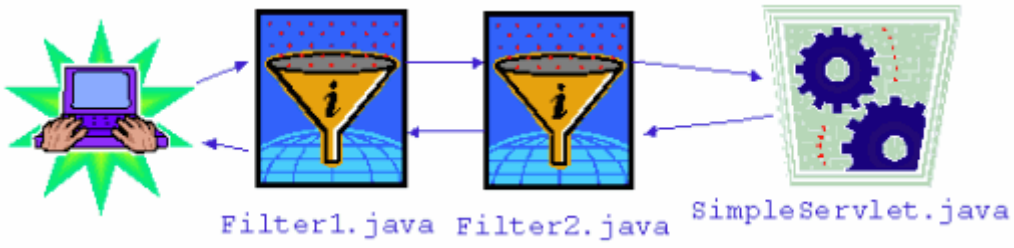


Figure 4. Diagramme de l'exemple de chaînage de filtres.

Le Listing 3 montre le code pour Filter1.java , qui est un simple filtre qui Filter1.java un message à la console du serveur web. Le message vous permet de suivre la demande à travers les filtres.

Listing 3: Un filtre simple qui génère un message sur la console. (Filter1.java)

```
package com.filters;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
```

```
import java.io.IOException;
import javax.servlet.ServletException;

public class Filter1 implements Filter
{
   public  void init(FilterConfig filterConfig) { }


public void doFilter (ServletRequest request,
ServletResponse response,
FilterChain chain)
{

try
{
System.out.print ("Within First Filter ... ");
System.out.println ("Filtering the Request ...");

chain.doFilter (request, response);

System.out.print ("Within First Filter ... ");
System.out.println ("Filtering the Response ...");

} catch (IOException io) {
System.out.println ("IOException raised in Filter1 Filter");
} catch (ServletException se) {
System.out.println ("ServletException raised in Filter1 Filter");
}
}

}
```

Le listing 4 montre le code du second filtre de la chaîne qui imprime un message à la console du serveur Web pour nous aider à suivre la demande et la réponse..

Listing 4. Un filtre simple qui génère un message sur la console. (Filter2.java)

```
package com.filters;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

import java.io.IOException;
import javax.servlet.ServletException;

public class Filter2 implements Filter
{
   public  void init(FilterConfig filterConfig) { }

public void doFilter (ServletRequest request,
ServletResponse response,
FilterChain chain)
{

try
{
System.out.print ("Within Second Filter ... ");
```

```
System.out.println ("Filtering the Request ...");

chain.doFilter (request, response);

System.out.print ("Within Second Filter ... ");
System.out.println ("Filtering the Response ...");

} catch (IOException io) {
System.out.println ("IOException raised in Filter2 Filter");
} catch (ServletException se) {
System.out.println ("ServletException raised in Filter2 Filter");
}
}

}
```

Après avoir écrit les filtres, ils doivent être définis et mappés à un modèle d'URL dans le descripteur de déploiement. L'ordre dans lequel ils sont définis est important. Le conteneur va exécuter les filtres dans l'ordre dans lequel ils sont définis. Le Listing 5 montre le fichier web.xml pour déployer Filter1.java et Filter2.java .

Listing 5: Le descripteur de déploiement pour le chaînage de filtres. (web.xml)

```
<?xml version = "1.0" encoding = "ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">

<web-app>

<!-- Define the filters within the Web Application -->

<filter>
<filter-name>
First Filter in Chain
</filter-name>
<filter-class>
com.filters.Filter1
</filter-class>
</filter>

<filter>
<filter-name>
Second Filter in Chain
</filter-name>
<filter-class>
com.filters.Filter2
</filter-class>
</filter>


<!-- Map the filter to a Servlet or URL -->

<filter-mapping>
<filter-name>
First Filter in Chain
</filter-name>
<url-pattern>
/simple
</url-pattern>
</filter-mapping>
```

```
<filter-mapping>
<filter-name>
Second Filter in Chain
</filter-name>
<url-pattern>
/simple
</url-pattern>
</filter-mapping>


<!-- Define the Servlets within the Web Application -->

<servlet>
<servlet-name>
Simple Servlet
</servlet-name>
<servlet-class>
com.servlets.SimpleServlet
</servlet-class>
</servlet>


<!-- Define Servlet mappings to urls -->

<servlet-mapping>
<servlet-name>
Simple Servlet
</servlet-name>
<url-pattern>
/simple
</url-pattern>
</servlet-mapping>

</web-app>
```

La sortie des messages dans la console est illustrée à la figure 5. Remarquez que la requête est passée par Filter1 puis Filter2 , mais la réponse est passée par Filter2 puis Filter1 .

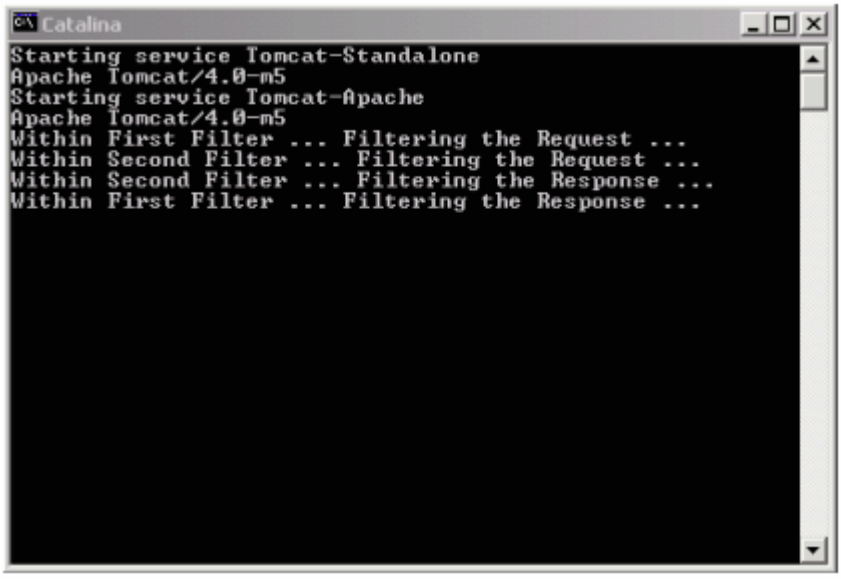


Figure 5. Sortie de la chaîne de filtres.

2. Mise en place d'un filtre

2.1. Écriture d'un filtre

Techniquement un filtre est une classe qui doit implémenter l'interface Filter. Cette inferface définit trois méthodes :

- init(FilterCongif) : méthode callback, appelée lors de la construction de cet objet par le serveur d'applications.
- destroy() : méthode callback, appelée lors de la destruction de cet objet par le serveur d'applications.
- doFilter(ServletRequest, ServletResponse, FilterChain) : cette méthode est appelée sur requête HTTP, au lieu de l'appel d'une servlet sur lequel ce filtre est déclaré. La propagation de la requête au filtre suivant, ou à la servlet est faite en invoquant filterChain.doFilter(request, response).

Exemple 30. Construction d'un filtre

```
public class TransparentFilter implements Filter {

   public void init(FilterConfig filterConfig) {
     // l'objet filterConfig encapsule les paramètres
     // d'initialisation de ce filtre

  }

   public void destroy() {
     // callback de destruction de ce filtre
  }


   public void doFilter(ServletRequest request, ServletResponse response,
                        FilterChain filterChain)
   throws IOException, ServletException {

     // propagation de la requête le long de la chaîne
    filterChain.doFilter(request, response) ;
  }
}
```

2.2. Déclaration du filtrage

Un filtre doit ensuite être déclaré dans le fichier web.xml de l'application. De même qu'une servlet, un filtre est déclaré en deux temps : d'une part par sa classe d'implémentation, d'autre part par l'URL qu'il filtre. On peut mapper un filter à des classes de servlet ou url-patterns .
Voyons ceci sur un exemple.

Exemple 31. Déclaration d'un filtre

```
<web-app>
  <!-- déclaration de la classe d'implémentation du filtre -->
  <filter>
    <filter-name>SimpleFilter</filter-name>
    <filter-class>org.paumard.cours.servlet.SimpleFilter</filter-class>
  </filter>

  <!-- déclaration des URL interceptées par ce filtre -->
  <filter-mapping>
    <filter-name>SimpleFilter</filter-name>
    <!-- on aurait pu aussi définir une servlet filtrée, par un élément servlet-name -->
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <!-- suivent les déclarations des servlets -->
  <servlet>
    <servlet-name>TestServlet</servlet-name>
```

```
    <servlet-class>org.paumard.cours.servlet.TestServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>TestServlet</servlet-name>
    <url-pattern>/test</url-pattern>
  </servlet-mapping>

</web-app>
```

Implémentation d'un filtre

L'opération de filtrage est réalisée par la méthode doFilter qui va filtrer la requête et la réponse
Principe général d'implémentation d'un filtre

```
@Override
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    throws IOException, ServletException {
  // réaliser des opérations avant le traitement de la requête

  // appeler l'élément suivant dans la chaîne de filtrage
  chain.doFilter(request, response);

  // réaliser des opérations après le traitement de la requête
}
```

Si plusieurs filtres doivent être déclenchés pour le traitement d'une requête, alors l'appel à chain.doFilter(...) permet de passer au filtre suivant. Un fois le dernier filtre appelé, l'appel à chain.doFilter(...) passera au traitement normal de la requête et de la manipulation de réponse à retourner vers le client (Servlet, JSP ou ressource statique).
Il est recommandé d'implémenter des filtres de manière à ce qu'ils soient indépendants les uns des autres. En effet, si plusieurs filtres sont appelés pour le traitement d'une requête, l'ordre dans lequel ces filtres seront appelés n'est pas prédictible s'ils ont été déclarés avec l'annotation @WebFilter. En revanche, ils seront appelés dans l'ordre des balises filter-mapping s'ils ont été déclarés à partir du fichier de déploiement web.xml
Une implémentation de filtre peut très bien ne pas appeler chain.doFilter(...) et choisir de générer directement une réponse.

Cas d'utilisation de filtres

3. Filtrage d'une requête
Comme nous l'avons vu, un filtre peut être utilisé pour valider ou modifier une requête, ou une réponse. Commençons par examiner la façon dont on peut agir sur la requête.
Écrivons par exemple un filtre simple, qui enregistre les paramètres de la requête pour toutes les servlets de notre application.

Exemple 32. Filtrage d'une requête pour de la journalisation

```
public class LoggerFilter implements Filter {

  private static Logger logger = Logger.getLogger(LoggerFilter.class) ;

  // méthodes init() et destroy()

  private void beforeProcessing(ServletRequest request, ServletResponse response)
  throws IOException, ServletException {
```

```
      HttpServletRequest httpRequest = (HttpServletRequest)request ;
      String host = httpRequest.getRemoteHost() ;
      String url = httpRequest.getRequestURL().toString() ;

      logger.debug("L'hôte [" + host +  "] fait une requête sur [" + url +  "]") ;

  }

   public  void doFilter(ServletRequest request, ServletResponse response,
                  FilterChain filterChain)
   throws IOException, ServletException {

     beforeProcessing(request, response) ;
     filterChain.doFilter(request, response) ;
  }
}
```

La mise en place ici est très simple : il s'agit juste d'enregistrer les informations du client qui a soumis cette requête.
On peut aussi interdire l'accès à une ressource, comme dans l'exemple suivant.

Exemple 33. Filtrage d'une requête avec validation de sécurité

```
 public  class LoggerFilter  implements Filter {

   // méthodes init() et destroy()

   public  void doFilter(ServletRequest request, ServletResponse response,
                  FilterChain filterChain)
   throws IOException, ServletException {

     if (request.isUserInRole("admin")) {

       // l'utilisateur a été identifié comme étant un administrateur
       // il est autorisé à accéder à la servlet filtrée
      filterChain.doFilter(request, response) ;

    }  else {

       // l'utilisateur n'est pas authentifié correctement
       // on le redirige vers une page d'erreur
      RequestDispatcher requestDispatcher =
        request.getRequestDispatcher("/userNotAdmin.jsp") ;
      requestDispatcher.forward(request, response) ;
    }
  }
}
```

Deux exemples d'implémentation de filtres simples mais efficaces.

Gestion de l'UTF-8

Un cas facilement compréhensible est celui d'une application Web qui poste les données de tous ses formulaires HTML en UTF-8. Nous avons vu que par défaut, le conteneur Web utilise l'encodage ISO-8859-1 (Latin-1). Il est donc nécessaire de positionner le bon encodage grâce à la méthode ServletRequest.setCharacterEncoding(String). Cette opération répétitive est source d'oubli (et donc de bug). Il serait préférable de garantir que cette méthode soit systématiquement appelée avant chaque traitement de Servlet. Ce type de comportement peut très facilement être implémenté au moyen d'un filtre Web.

Filtre UTF-8

```
package fr.epsi;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;

@WebFilter("/*")
public class Utf8RequestEncodingFilter implements Filter {

  @Override
  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
      throws IOException, ServletException {
    request.setCharacterEncoding("UTF-8");
    chain.doFilter(request, response);
  }

  @Override
  public void init(FilterConfig filterConfig) throws ServletException {
  }

  @Override
  public void destroy() {
  }

}
```

Génération de log

Il peut être intéressant de garder une trace des paramètres HTTP reçus lors des tests ou pour des statistiques. Le filtre ci-dessous écrit dans les logs du serveur le nom et la valeur de tous les paramètres reçus :
Filtre de log de paramètres

```
package fr.epsi;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;

@WebFilter("/*")
```

```
public class LogFilter implements Filter {

  @Override
  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
      throws IOException, ServletException {
    request.getServletContext().log("parameters received: " + parametersToString(request));
    chain.doFilter(request, response);
  }

  private List<String> parametersToString(ServletRequest request) {
    List<String> parameters = new ArrayList<>();
    request.getParameterMap().forEach((k, v) -> parameters.add(k + "=" + Arrays.toString(v)));
    return parameters;
  }

  @Override
  public void init(FilterConfig filterConfig) throws ServletException {
  }

  @Override
  public void destroy() {
  }

}
```

L'utilisation conjointe des deux filtres ci-dessus peut poser problème. En effet, la méthode request.setCharacterEncoding(...) dans la classe Utf8RequestEncodingFilter doit être appelée avant que les paramètres de la requête ne soient accédés. Le filtre Utf8RequestEncodingFilter doit donc être placé **avant** le filtre LogFilter. Malheureusement, cela ne peut pas être garanti par l'utilisation de l'annotation @WebFilter.
On peut imaginer des traitements bien plus complexes grâce aux filtres : contrôle des droits d'accès (autorisation), optimisation d'image, chiffrement des données...

Example of counting number of visitors for a single page

MyFilter.java

```
import java.io.*;
import javax.servlet.*;

public class MyFilter implements Filter{
    static int count=0;
    public void init(FilterConfig arg0) throws ServletException {}

    public void doFilter(ServletRequest req, ServletResponse res,
            FilterChain chain) throws IOException, ServletException {
          PrintWriter out=res.getWriter();
        chain.doFilter(request,response);
        out.print("<br/>Total visitors "+(++count));
        out.close();

      }
      public void destroy() {}
    }
```

2. **Les listeners Web**

Un listener (écouteur) est utilisé en Java pour désigner un objet qui sera notifié lors d'une modification de son environnement. Pour le conteneur Web Java EE, un listener designe une classe qui implémente une des interfaces définies dans l'API servlet.
Les différents types de listeners sont représentés dans l'API Servlet par les **interfaces** Java suivantes :

javax.servlet.ServletContextListener

Permet d'écouter les changements d'état du ServletContext représentant l'application Web. Le conteneur Web avertit l'application de la création du ServletContext grâce à la méthode contextInitialized et de la destruction du ServletContext grâce à la méthode contextDestroyed. Ces méthodes prennent en paramètre le même objet événement, de type ServletContextEvent ayant une unique méthode : getServletContext(), qui retourne l'objet ServletContext. Un ServletContextListener est un moyen de réaliser des traitements au moment du lancement de l'application Web et/ou au moment de son arrêt.

javax.servlet.ServletContextAttributeListener

Permet d'écouter les changements d'état des attributs stockés dans le ServletContext (les attributs de portée application). Le conteneur avertit l'application de l'ajout d'un attribut au contexte considéré (appel à la méthode attributeAdded), de la suppression d'un attribut du contexte (appel à la méthode attributeRemoved) et de la modification d'un attribut (appel à la méthode attributeReplaced).

javax.servlet.ServletRequestListener

Permet d'écouter l'entrée et/ou la sortie d'une requête de l'environnement de l'application Web. Le conteneur avertit l'application de l'entrée d'une requête à traiter grâce à la méthode requestInitialized et de la sortie de la requête grâce à la méthode requestDestroyed. Ces deux méthodes prennent en paramètre le même objet, instance de ServletRequestEvent. Cet objet permet d'accéder au contexte de l'application web par la méthode getServletContext(), et à l'objet requête par la méthode getServletRequest().

javax.servlet.ServletRequestAttributeListener

Permet d'écouter les changements d'état des attributs stockés dans la HttpServletRequest (les attributs de portée requête). Le conteneur avertit l'application de l'ajout d'un attribut dans le contexte (appel à la méthode attributeAdded), de la suppression d'un attribut du contexte (appel à la méthode attributeRemoved) et de la modification d'un attribut (appel à la méthode attributeReplaced).

javax.servlet.http.HttpSessionListener

Permet d'écouter la création et la suppression d'une HttpSession. Le conteneur avertit l'application de la création d'une session grâce à la méthode sessionCreated et de la suppression d'une session grâce à la méthode sessionDestroyed. Ces méthodes prennent en paramètre un objet HttpSessionEvent ayant unique méthode : getSession(), qui retourne la session concernée.
La suppression d'une session signifie que soit elle a été invalidée par l'application elle-même grâce à la méthode HttpSession.invalidate() soit elle est arrivée à expiration et le conteneur a décidé de l'invalider.
L'interface HttpSessionListener est utilisée pour surveiller quand les sessions sont crées et supprimées dans un serveur d'application. Il est pratiquement utilisé pour tracer des statistiques d'utilisation d'une session pour un serveur.

javax.servlet.http.HttpSessionAttributeListener

Permet d'écouter les changements d'état des attributs stockés dans la HttpSession (les attributs de portée session). Le conteneur avertit l'application de l'ajout d'un attribut (appel à la méthode attributeAdded), de la suppression d'un attribut (appel à la méthode attributeRemoved) et de la modification d'un attribut (appel à la méthode attributeReplaced).

Objets événement associés

Pour les interfaces ServletContextAttributeListener, ServletRequestAttributeListener et HttpSessionAttributeListener, les trois méthodes attributeAdded(), attributeRemoved(), attributeReplaced() prennent des paramètres différents en fonction du contexte, qui sont les événements. Ces objets exposent tous les mêmes méthodes :

- getName() : le nom de l'attribut concerné ;
- getValue() : la valeur de l'attribut concerné.

Il existe également deux autres listeners Web :
HttpSessionBindingListenerer : Cette interface expose deux méthodes :

- valueBound(HttpSessionBindingEvent) : appelée lorsque l'objet est attaché à une session ;
- valueUnbound(HttpSessionBindingEvent) : appelée lorsque l'objet est détachée de la session.

Ces deux méthodes reçoivent en paramètre un objet de type **HttpSessionBindingEvent**. Cet événement expose les deux méthodes classiques **getName() et getValue() et** également **getSession(),** qui retourne la session à laquelle cet objet est attaché, ou de laquelle il est détaché.
HttpSessionActivationListener : exposant les méthodes sessionDidActivate() et sessionWillPassivate(). Ces deux méthodes prennent en paramètre le même objet **HttpSessionEvent.** pour signaler à un objet qu'il a été ajouté ou retiré d'une session et exposant en plus une méthode **getSession().**

Déclaration des listeners Web
Une classe implémentant une ou plusieurs interfaces la désignant comme un listener Web doit également être déclarée auprès du conteneur Web. Pour cela, il suffit d'ajouter l'annotation @WebListener à la classe :

Exemple 1 :de ServletContextListener

L'exemple (simple) ci-dessous consiste en un ServletContextListener dont le rôle est de réaliser un log applicatif signalant respectivement le lancement et l'arrêt de l'application Web :

```
package com.exa;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

@WebListener
public class LoggingListener implements ServletContextListener {
 ServletContext context;
 @Override
  public void contextInitialized(ServletContextEvent sce) {
    sce.getServletContext().log("## Lancement de l'application ##");
System.out.println("ServletContextListener started");
  context = sce.getServletContext();
                        // set variable to servlet context
 context.setAttribute("TEST", "TEST_VALUE");

  }

  @Override
  public void contextDestroyed(ServletContextEvent sce) {
    sce.getServletContext().log("## Arrêt de l'application ##");
    System.out.println("ServletContextListener destroyed");
  }
}
```

Exemple 2 : de ServletRequestListener

```
package com.exa;

import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
import javax.servlet.annotation.WebListener;

@WebListener
public class MyServletRequestListener implements ServletRequestListener {
```

```
  @Override
  public void requestInitialized(ServletRequestEvent sre) {
    // ...
  }

  @Override
  public void requestDestroyed(ServletRequestEvent sre) {
    // ...
  }
}
```

Si l'**on ne souhaite pas utiliser une annotation**, il est également possible de déclarer un listener dans le fichier de déploiement web.xml grâce à la balise listener.
Déclaration d'un listener dans le fichier web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                      http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  version="3.0">

  <listener>
            <listener-class>com.exa.LoggingListener </listener-class>
  </listener>
<listener>
            <listener-class>com.exa.MyServletRequestListener</listener-class>
  </listener>


</web-app>
```

Utlisation de l'interface ServletContextListener

Introduction:

A propos de ServletContext.

1. Il n'y aura qu'un seul ServletContext pour chaque application Web.
2. ServletContext sera créé lors du déploiement de l'application. Une fois le ServletContext créé, il sera utilisé par tous les servlets et fichiers jsp dans la même application.
3. ServletContext sera détruit lors de la destruction de l'application Web ou de l'arrêt du serveur.

L'interface ServletContextListener définit les 2 méthodes suivantes.

1. public void contextInitialized (ServeletContextEvent e)

Cette méthode sera exécutée lors de la création de l'objet ServletContext, c'est-à-dire au moment du déploiement de l'application.

2.public void contextDestroyed (ServletContextEvent e)

Cette méthode sera exécutée automatiquement par le conteneur Web au moment de la destruction de l'objet contextuel, c'est-à-dire au moment de la désinstallation de l'application ou de l'arrêt du serveur.

Comment le ServletContextListener est utile:

1. ServletContextListener est averti lorsque le contexte est initialisé.
   - ServletContextListener récupère les paramètres init du contexte à partir de ServletContext.
   - Il stocke la connexion de la base de données en tant qu'attribut, de sorte que les autres composants de l'application Web puissent y accéder.
2. Il sera notifié lorsque le contexte est détruit. Il ferme la connexion à la base de données.

ServletContextEvent

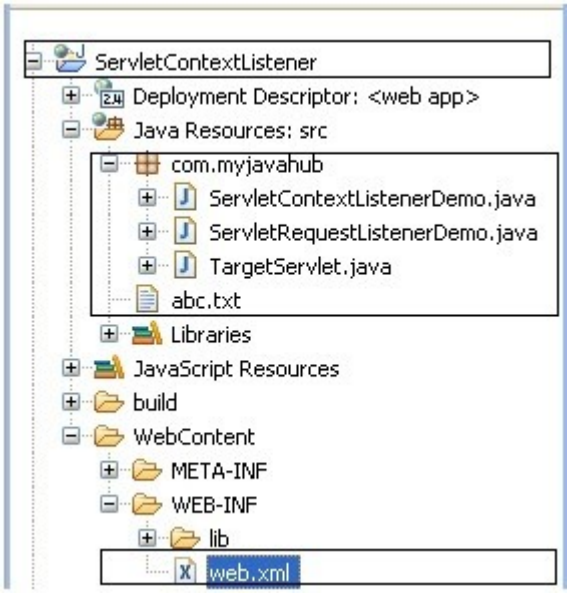
Cette classe contient une seule méthode getServetContext.

```
public ServletContext getServletContext ()
```

Programme de démonstration:

Dans l'exemple, nous devons maintenir une requête Listener pour incrémenter la valeur du compteur pour chaque requête et maintenir un écouteur de contexte qui stocke la valeur de comptage dans un fichier texte lors de l'arrêt du serveur et redéploiement de l'application et un écouteur de contexte va lire la valeur de compteur du fichier texte au démarrage du serveur de temps ou au déploiement de l'application et attribuez-la à la variable de comptage de l'écouteur de la demande.

Étape 1: **Structure du projet dans Eclipse**



Étape 2:

Créez une classe qui implémente les méthodes d'interface ServletContextListener : contextInitialized () et contextDestroyed ()

ServletContextListenerDemo.java

```
package com.myjavahub;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
```

```
public class ServletContextListenerDemo  implements ServletContextListener
{
        public void contextDestroyed(ServletContextEvent e)
        {
           System.out.println("context destroy");
           try
           {
             String path =          e.getServletContext().getRealPath("abc.txt");
             String count = e.getServletContext().getAttribute("counter").toString();
             BufferedWriter bw = new BufferedWriter(new FileWriter(path));
             bw.write(count);
             bw.flush();
             bw.close();
              }
           catch (Exception e1)
           {
              }
        }

        public void contextInitialized(ServletContextEvent e)
        {
               try
               {
                       String  path = e.getServletContext().getRealPath("abc.text");
                       BufferedReader br = new BufferedReader(new FileReader(path));
                        System.out.println(path);
                       System.out.println("context intialize");
                      String s = br.readLine();
                       if (s!= null)
                       {
                         int c = Integer.parseInt(s);
                         ServletRequestListenerDemo.count = c;
                         System.out.println("correct hitcount value :"+c);
                              }
                       }
                       catch (Exception e2)
                       {
                       }
        }
}
```

Étape 3: Créer une classe implémentant **ServletRequestListener** pour incrémenter la valeur du compteur pour chaque requête.

ServletRequestListenerDemo.java

```
package com.myjavahub;
import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
public class ServletRequestListenerDemo implements ServletRequestListener
{
   public static int count = 0;
        public void requestDestroyed(ServletRequestEvent arg0)
        {
               count ++;
        }
```

```
        public void requestInitialized(ServletRequestEvent arg0)
        {

        }
}
```

Étape 4: créez une classe de servlet qui gère une requête,

TargetServlet.java

```
package com.myjavahub;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TargetServlet extends HttpServlet
{

        private static final long serialVersionUID = 1L;
        protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                throws ServletException, IOException
   {
      PrintWriter out = resp.getWriter();
      getServletContext().setAttribute("counter", ServletRequestListenerDemo.count);
      out.print("<h1>The numbers of hits for this application : :"+ServletRequestListenerDemo.count);
   }
}
```

Étape 5: Configurez l'écouteur dans le descripteur de déploiement web.xml,

```
<listener>
        <listener-class>com.myjavahub.ServletRequestListenerDemo</listener-class>
        </listener>
        <listener>
        <listener-class>com.myjavahub.ServletContextListenerDemo</listener-class>
</listener>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd">
        <listener>
        <listener-class>com.myjavahub.ServletRequestListenerDemo</listener-class>
        </listener>
        <listener>
        <listener-class>com.myjavahub.ServletContextListenerDemo</listener-class>
        </listener>
        <servlet>
          <servlet-name>TargetServlet</servlet-name>
          <servlet-class>com.myjavahub.TargetServlet</servlet-class>
        </servlet>
        <servlet-mapping>
           <servlet-name>TargetServlet</servlet-name>
           <url-pattern>/hitcount</url-pattern>
        </servlet-mapping>
</web-app>
```

Étape 6: Déployez l'application sur le serveur d'applications / le conteneur Web et vérifiez la sortie suivante dans le journal du serveur / de la console,

Sortie:

Utlisation de l'interface de ServletContextEvent et ServletContextListener

Quel est le besoin de ServletContextListener?

Parfois, nous pouvons exiger qu'un code soit exécuté avant qu'une application Web ne démarre. Par exemple, nous devons créer une connexion à la base de données afin que l'application Web puisse l'utiliser chaque fois qu'elle effectue certaines opérations et lorsque l'application s'arrête, nous pouvons fermer la connexion à la base de données.

Comment pouvons-nous y parvenir?

La spécification Java EE fournit une interface nommée ServletContextListener qui sert notre but. ServletContextListener écoute les événements du cycle de vie d'un contexte de servlet. Cette interface est notifiée chaque fois qu'une application avec laquelle l'auditeur est associé démarre et s'arrête.

Les implémentations de cette interface reçoivent des notifications concernant les modifications apportées au contexte de servlet de l'application Web dans laquelle elles font partie. Pour recevoir des événements de notification, la classe d'implémentation doit être configurée dans le descripteur de déploiement pour l'application Web.

Si vous souhaitez écouter lorsque l'application Web démarre, utilisez la **méthode contextInitialized (ServletContextEvent event)** .

Notification selon laquelle le processus d'initialisation de l'application Web commence. Tous les ServletContextListeners sont informés de l'initialisation du contexte avant que n'importe quel filtre ou servlet de l'application Web ne soit initialisé.

Si vous souhaitez écouter lorsque l'application Web s'arrête, utilisez la méthode contextDestroyed (ServletContextEvent event) .

Notification selon laquelle le contexte de la servlet est sur le point d'être arrêté. Tous les servlets et les filtres ont été détruits () avant que ServletContextListeners ne soit notifié de la destruction du contexte.

Dans cet exemple, on recupère les données d'une table d'une base de données. Pour cela, on crée un objet connection dans une classe listener et on utilise cet objet connection dans une servlet.

index.html

```
<a href="servlet1">fetch records</a>
```

MyListener.java

```
package com.exa ;
import java.sql.*;
//import javax.servlet.*;
import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
public class MyListener implements ServletContextListener{
public void contextInitialized(ServletContextEvent event) {
try{
        Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(  " jdbc:mysql://localhost/Base
","user","user");
  /*    ServletContext sc = event.getServletContext();

            String url = sc.getInitParameter("url");
            String user_name = sc.getInitParameter("user_name");
            String password = sc.getInitParameter("password");
Connection con=DriverManager.getConnection(  url,user_name,password);
*/
              //storing connection object as an attribute in ServletContext
        ServletContext ctx=event.getServletContext();
        ctx.setAttribute("mycon", con);
        }catch(Exception e){e.printStackTrace();}
        }
          public void contextDestroyed(ServletContextEvent arg0) {}
.       }
```

FetchData.java

```
package com.exa ;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
.
public class FetchData extends HttpServlet {

        public void doGet(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException {  .
        response.setContentType("text/html");
        PrintWriter out = response.getWriter(); .
        try{
        //Retrieving connection object from ServletContext object
        ServletContext ctx=getServletContext();
        Connection con=(Connection)ctx.getAttribute("mycon");
          //retieving data  from emp table
        PreparedStatement ps=con.prepareStatement("select * from emp",
        ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
          ResultSet rs=ps.executeQuery();
        while(rs.next()){
        out.print("<br>"+rs.getString(1)+" "+rs.getString(2));
.       }

        con.close();
        }catch(Exception e){e.printStackTrace();}

        out.close();
        }
        }
```

Dans le repertoire lib, mettre mysql-connector-java-5.1.7-bin.jar

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
<listener>
        <listener-class>
                com.exa.MyListener
        </listener-class>
  </listener>

<context-param>
    <param-name>url</param-name>
    <param-value>jdbc:mysql://localhost/testdb</param-value>
  </context-param>
   <context-param>
        <param-name>user_name</param-name>
        <param-value>jean</param-value>
  </context-param>
  <context-param>
        <param-name>password</param-name>
        <param-value>jean</param-value>
  </context-param>
  <servlet>
    <description></description>
    <display-name>testClass</display-name>
    <servlet-name>testClass</servlet-name>
    <servlet-class>com.exa.FetchData</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>testClass</servlet-name>
    <url-pattern>/testClass</url-pattern>
  </servlet-mapping>


</web-app>
```

Utlisation de l'interface ServletContextAttributeListener

Introduction:

Cet écouteur est utilisé lorsque l'on veut savoir quand un attribut a été ajouté ou supprimé ou remplacé par un autre attribut dans un

Cette interface définit les 3 méthodes suivantes.

1. public void attributeAdded (ServletContextAttributeEvent e)

Cette méthode sera exécutée automatiquement par le conteneur Web au moment de l'ajout de l'attribut à l'objet ServletContext.

2. public void attributeRemoved (ServletContextAttributeEvent e)

Cette méthode sera exécutée automatiquement par le conteneur Web au moment où l'attribut existant est supprimé de l'objet ServletContext.

3. public void attributeReplaced (ServletContextAttributeEvent e)

Cette méthode sera exécutée automatiquement par le conteneur Web au moment où la valeur de l'attribut est remplacée par une autre valeur.

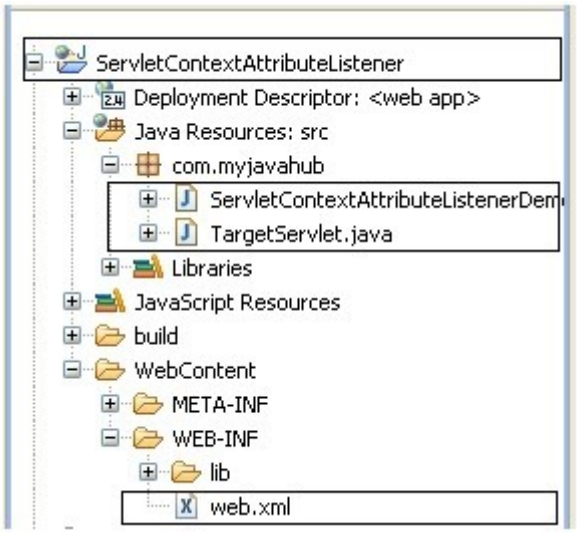
ServletContextAttributeEvent est une classe d'événements utilisée pour les notifications lorsque les modifications sont apportées aux attributs de ServletContext dans une application.
La classe ServletContextAttributeEvent a deux méthodes:

getName (): renvoie le nom de l'attribut qui a été modifié sur le ServletContext.

getValue (): renvoie la valeur de l'attribut qui a été ajouté, supprimé ou remplacé par un autre attribut.

Exemple:

Étape 1: Structure du projet dans Eclipse



Étape 2: créez une classe qui implémente les méthodes d'interface ServletContextAttributeListener : attributeAdded (), attributeReplaced () et attributeRemoved ()

ServletContextAttributeListenerDemo .java

```
package com.myjavahub;
import javax.servlet.*;

public class ServletContextAttributeListenerDemo  implements ServletContextAttributeListener
{
```

```
  public void attributeAdded(ServletContextAttributeEvent scae)
  {
    System.out.println("New Context attribute added");
    System.out.println("Name of Attribute added: " + scae.getName());
    System.out.println("Value of Attribute added: " + scae.getValue());
  }

  public void attributeRemoved(ServletContextAttributeEvent scae) {
    System.out.println("Context attribute removed");
    System.out.println("Name of Attribute removed: " + scae.getName());
    System.out.println("Value of Attribute removed: " + scae.getValue());
  }

  public void attributeReplaced(ServletContextAttributeEvent scae) {
    System.out.println("Context attribute replaced");
    System.out.println("Name of Attribute replaced: " + scae.getName());
    System.out.println("Value of Attribute replaced: " + scae.getValue());
  }
}
```

Étape 3: configurez l'écouteur dans le descripteur de déploiement web.xml,

```
<listener>
  <listener-class>
    com.myjavahub. ServletContextAttributeListenerDemo
  </listener-class>
</listener>
```

Étape 4 :
Créez une classe de servlets qui ajoute, remplace et supprime un attribut du ServletContext.

```
package com.myjavahub;

import java.io.*;

import javax.servlet.ServletContext;
import javax.servlet.http.*;

public class TargetServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException
{
    PrintWriter out = response.getWriter();
    out.print("plz check server logs");
          ServletContext sc = getServletContext();
     sc.setAttribute("attr1", "value1");
     sc.setAttribute("attr1", "value2");
     sc.removeAttribute("attr1");
  }
}
```

Étape 5: Configurez le servlet dans le descripteur de déploiement web.xml,

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<listener>
  <listener-class>com.myjavahub.ServletContextAttributeListenerDemo</listener-class>
</listener>
        <servlet>
  <servlet-name>TargetServlet</servlet-name>
  <servlet-class>com.myjavahub.TargetServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>TargetServlet</servlet-name>
  <url-pattern>/target</url-pattern>
</servlet-mapping>
</web-app>
```

Étape 6: Testez l'exemple avec l'URL suivante dans le navigateur et vérifiez le journal du serveur

http: // localhost: 8080 / ServletContextAttributeListener / target

Sortie:

```
New Context attribute added
Name of Attribute added: attr1
Value of Attribute added: value1
Context attribute replaced
Name of Attribute replaced: attr1
Value of Attribute replaced: value1
Context attribute removed
Name of Attribute removed: attr1
Value of Attribute removed: value2
```

Utlisation de l'interface ServletRequestListener

Cet écouteur permet de recevoir une notification chaque fois qu'on soumet une requete cliente venant d'une application pour une ressource.
L'interface ServletRequestListener permet de créer une classe de gestionnaire d'événements qui écoute les événements sur l'objet **ServletRequest**, à savoir la création et la destruction de l'objet de requête.
L'interface implémente 2 méthodes :

1. `public void requestInitialized (ServletRequestEvent e)`

Cette méthode sera exécutée automatiquement par le conteneur Web au moment de la création de l'objet de requête. C'est à dire juste avant d'appeler la méthode de service (doget, doPost, service).

2.`public void requestDestroyed (ServletRequestEvent e)`

Cette méthode sera exécutée par le conteneur Web au moment de la destruction de l'objet de requête, c'est-à-dire juste après la fin du service ().
ServletRequestEvent:
Cette classe définit les 2 méthodes suivantes pour obtenir l'objet Request et le contexte.

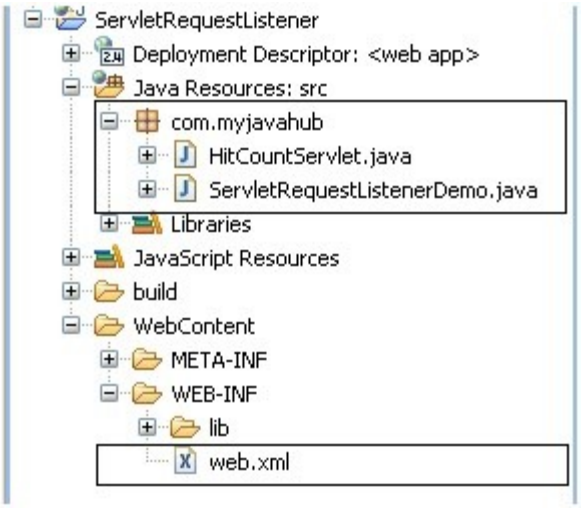
1. public ServletRequest getServletRequest ()

Renvoie la requête ServletRequest en cours de modification.

2. ServletContext public getServletContext ();

Renvoie le ServletContext de cette application Web.

Étape 1: Structure du projet dans Eclipse



Étape 2:
Créez une classe qui implémente les méthodes d'interface ServletRequestListener : requestInitialized () et requestDestroyed ().

ServletRequestListenerDemo .java

```
package com.myjavahub;

import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;

public class ServletRequestListenerDemo  implements ServletRequestListener
{
        public static int count = 0;
        public void requestDestroyed(ServletRequestEvent arg0)
        {
          System.out.print("The Request object destroyed at  :"+new java.util.Date());
        }
        public void requestInitialized(ServletRequestEvent arg0)
        {
                count++;
             System.out.print("Request Object created At:"+ new java.util.Date());
             System.out.print("The hit count for this web application :"+count);
        }
}
```

Étape 3: Créez une classe de servlet qui gère une requête,

```
package com.myjavahub;

import java.io.IOException;
```

```
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HitCountServlet extends HttpServlet
{
        private static final long serialVersionUID = 1L;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
              throws ServletException, IOException
    {
            PrintWriter out = resp.getWriter();
            out.print("<h2> This is target Servlet </h2>");
            out.print("<h2>the no of hits for this application is "+ServletRequestListenerDemo.count);
    }
}
```

Étape 4: Configurer l'écouteur dans le descripteur de déploiement web.xml,

```
</ listener >
< listener-class > com.myjavahub.ServletRequestDemoListener </ listener-class >
</ listener >
```

Étape 5: Configurez le servlet dans le descripteur de déploiement Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd">
        <listener>
        <listener-class>com.myjavahub.ServletRequestListenerDemo</listener-class>
        </listener>
        <servlet>
          <servlet-name>HitCountServlet</servlet-name>
          <servlet-class>com.myjavahub.HitCountServlet</servlet-class>
        </servlet>
        <servlet-mapping>
          <servlet-name>HitCountServlet</servlet-name>
          <url-pattern>/hitcount</url-pattern>
        </servlet-mapping>
</web-app>
```

Étape 6:

Testez l'exemple avec l'URL suivante dans le navigateur et vérifiez le journal du serveur / de la console

Sortie: http: // localhost: 8080 / ServletRequestListener / hitcount

Utlisation de l'interface ServletRequestAttributeListener
Introduction:

1. Cet écouteur écoute les événements liés à l'attribut de portée de requête comme l'ajout, la suppression et le remplacement d'attribut.
2. Cette interface définit les trois méthodes suivantes

```
public void attributeAdded (ServletRequestAttributeEvent e)
```

Cette méthode sera exécutée automatiquement par le conteneur Web lorsque nous ajouterons un attribut dans la portée de la demande.

```
public void attributeRemoved (ServletRequestAttributeEvent e)
```

Cette méthode sera exécutée automatiquement par le conteneur Web lorsque nous supprimons un attribut dans la portée de la demande.

```
public void attributeReplaced (ServletRequestAttributeEvent e)
```

Cette méthode sera exécutée automatiquement par le conteneur Web lorsque nous remplacerons un attribut dans la portée Request.

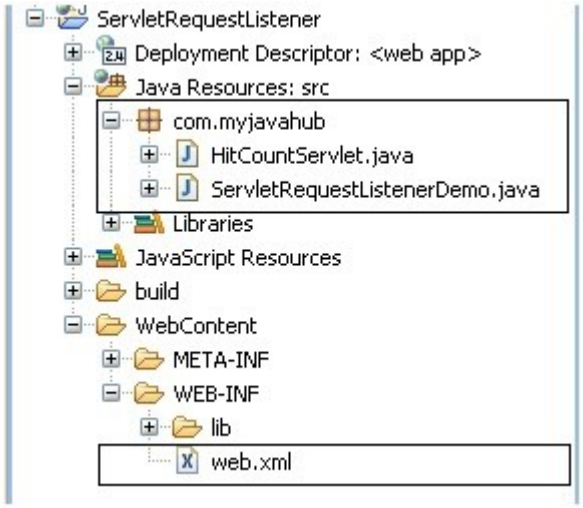
Dans les méthodes ci-dessus, vous pouvez voir que nous avons utilisé la classe ServletRequestAttributeEvent en tant que paramètre pour les méthodes ci-dessus. Cette classe est une classe d'événements utilisée pour les notifications lorsque les modifications sont apportées aux attributs de ServletRequest dans une application.
La classe **ServletRequestAttributeEvent** a deux méthodes:

getName (): Cette méthode renvoie le nom de l'attribut qui a été modifié dans ServletRequest.

getValue (): Cette méthode renvoie la valeur de l'attribut qui a été ajouté, supprimé ou remplacé par un autre attribut.

Exemple:

Étape 1: Structure du projet dans Eclipse



Étape 2:

Créez une classe qui implémente les méthodes d'interface ServletRequestAttributeListener : attributeAdded (), attributeRemoved () et attributeReplaced ()

ServletRequestAttributeListenerDemo.java

```
package com.myjavahub;

import javax.servlet.ServletRequestAttributeEvent;
import javax.servlet.ServletRequestAttributeListener;

public class ServletRequestAttributeListenerDemo implements ServletRequestAttributeListener
{
	public void attributeAdded(ServletRequestAttributeEvent e)
	{
	System.out.println(e.getName()+" : Attribute added . .Value "+e.getValue());
	}

	public void attributeRemoved(ServletRequestAttributeEvent e)
	{
	 System.out.println(e.getName()+" : Attribute removed . " );
	}

	public void attributeReplaced(ServletRequestAttributeEvent e)
	{
	 System.out.println(e.getName()+" : Attribute replaced . .Value ");
	}
}
```

Étape 3: Créer une classe de servlet qui gère une requête,

TargetServlet.java

```
package com.myjavahub;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TargetServlet extends HttpServlet
{
        private static final long serialVersionUID = 1L;
        @Override
        protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                        throws ServletException, IOException
        {
                PrintWriter  out = resp.getWriter();
                out.print("plz check the server logs");
               // setting “site” attribute in  request object
                req.setAttribute("site", "myjavahub.com");
               // replacing  “site” attribute in request object
                req.setAttribute("site", "myjavahub.blogspot.com");
               // remove “site” attribute from request object.
                req.removeAttribute("site");
         }
}
```

Étape 4: Configurer l'écouteur dans le descripteur de déploiement web.xml,

```
<listener>
<listener-class>com.myjavahub.ServletRequestAttributeListenerDemo</listener-class>
</listener>
```

Étape 5: C onfigurer le servlet dans le descripteur de déploiement web.xml,
Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

<listener>
<listener-class>com.myjavahub.ServletRequestAttributeListenerDemo</listener-class>
</listener>

<servlet>
        <servlet-name>TargetServlet</servlet-name>
         <servlet-class>com.myjavahub.TargetServlet</servlet-class>
</servlet>
<servlet-mapping>
          <servlet-name>TargetServlet</servlet-name>
          <url-pattern>/target</url-pattern>
</servlet-mapping>
</web-app>
```

Étape 6: Testez l'exemple avec l'URL suivante dans le navigateur et vérifiez les journaux du serveur uniquement.

Résultat: http: // localhost: 8080 / ServletRequestAttributeListener / target

```
site : Attribute added . .Value myjavahub.com
site : Attribute replaced . .Value
site : Attribute removed .
```

Utlisation de l'interface HttpSessionListener pour compter le nombre de sessions actives
Introduction:

1. L'interface HttpSessionListener est utilisée pour surveiller quand les sessions sont créées et détruites sur le serveur d'applications.
2. L'objectif principal de cet écouteur est de notifier chaque fois qu'il y a un changement dans la liste des sessions actives dans une application Web.

Méthodes:

Cette interface définit les deux méthodes suivantes

1. `public void sessionCreated (HttpSessionEvent e)`

La méthode sessionCreated () sera appelée par le conteneur de servlet dès qu'une nouvelle session est créée pour cette application. **La** classe **HttpSessionEvent** est transmise en tant qu'argument à la méthode sessionCreated. Cet objet peut être utilisé pour obtenir les informations relatives à la session, y compris l'ID de session.

2. `public void sessionDestroyed (HttpSessionEvent e)`

Cette méthode sera exécutée automatiquement par le conteneur au moment de la destruction de l'objet session.

HttpSessionEvent

L'objet HttpSessionEvent peut être utilisé pour obtenir les informations relatives à la session, y compris l'ID de session.

```
public HttpSession getSession ()
```

Étape 1: Créer un package et une classe HTTP Session Listener

```
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

public class SessionListener implements HttpSessionListener {
    private int sessionCount = 0;

    public void sessionCreated(HttpSessionEvent event) {
        synchronized (this) {
            sessionCount++;
        }

        System.out.println("Session Created: " + event.getSession().getId());
        System.out.println("Total Sessions: " + sessionCount);
    }

    public void sessionDestroyed(HttpSessionEvent event) {
```

```
        synchronized (this) {
            sessionCount--;
        }
        System.out.println("Session Destroyed: " + event.getSession().getId());
        System.out.println("Total Sessions: " + sessionCount);
    }
}
```

Étape 2: Créer une entrée HttpSessionListener dans Web.xml
Par conséquent, pour chaque création et invalidation de session, les méthodes sessionDestroyed de sessionCreated seront appelées par le conteneur de servlet.

```
<listener>
        <listener-class>com.myjavahub.SessionListener</listener-class>
</listener>
```

Étape 3: Créer des fichiers JSP pour le suivi de session
Nous allons créer une petite application Web pour tester la fonctionnalité de SessionListener. Il y aura 3 fichiers JSP :
index.jsp affichera une liste d'utilisateurs. Ces utilisateurs sont stockés dans une variable de session.
AddUser.jsp : ajoutera l'utilisateur dans la variable de session. Et le DestroySession.jsp : invalidera la session.
Index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@page import="java.util.List"%>
<%@page import="java.util.ArrayList"%>
<html>
<head>
    <title>Servlet Session Listener example - viralpatel.net</title>
</head>
<body>
    <h2>Add User Screen</h2>
<span style="float: right">
<a href="DestroySession.jsp">Destroy this session</a>
</span>
    <form method="post" action="AddUser.jsp">
        <h3>Enter Username to Add in List</h3>
        <input type="text" name="user"/>
        <input type="submit" value="Add User"/>
    </form>

    <%
        List<String> users = (List<String>)session.getAttribute("users");
        for(int i=0; null!=users && i < users.size(); i++) {
            out.println("<br/>" + users.get(i));
        }
    %>
</body>
</html>
```

AddUser.jsp

```
<%@page import="java.util.ArrayList"%>
```

```
<%@page import="java.util.List"%>
<%
   String username = request.getParameter("user");
   List<String> users = (List<String>)session.getAttribute("users");

   if(null == users) {
      users = new ArrayList<String>();
   }
   users.add(username);
   session.setAttribute("users", users);
   response.sendRedirect("index.jsp");
%>
```

DestroySession.jsp

```
<%
   session.invalidate();
%>
<h2>Session Destroyed successfully.. </h2>
<a href="javascript:history.back()">Click here to go Back</a>
```

Étape 4: Exécutez l'application Web
Ajouter des utilisateurs et voir la liste des utilisateurs ajoutés et également vérifier la sortie de la console pour voir les journaux pour la création de session et le nombre de sessions.

Une fois que vous cliquez sur Détruire la session, la session est invalidée et le compte de session est décrémenté.


```
package com.exa;

import javax.servlet.http.HttpSessionListener;
import javax.servlet.http.HttpSessionEvent;

public class SessionCounter implements HttpSessionListener {
```

```
private static int activeSessions = 0;

public void sessionCreated(HttpSessionEvent se) {
activeSessions++;
}

public void sessionDestroyed(HttpSessionEvent se) {

if(activeSessions > 0)
activeSessions--;
}



public static int getActiveSessions() {
return activeSessions;
}

}
```

web.xml

```
<listener>
        <listener-class>com.exa.SessionCounter </listener-class>
  </listener>
```

Utlisation de l'interface HttpSessionEvent et HttpSessionListener pour compter le total et les utilisateurs courants logés

Fichiers nécessaires:

1. index.html: to get input from the user.
2. MyListener.java: A listener class that counts total and current logged-in users and stores this information in ServletContext object as an attribute.
3. First.java: A Servlet class that creates session and prints the total and current logged-in users.
4. Logout.java: A Servlet class that invalidates session.

index.html

```
1.  <form action="servlet1">
2.  Name:<input type="text" name="username"><br>
3.  Password:<input type="password" name="userpass"><br>
4.
5.  <input type="submit" value="login"/>
6.  </form>
```

MyListener.java

```
1.  import javax.servlet.ServletContext;
2.  import javax.servlet.http.HttpSessionEvent;
3.  import javax.servlet.http.HttpSessionListener;
4.
5.  public class CountUserListener implements HttpSessionListener{
6.     ServletContext ctx=null;
7.     static int total=0,current=0;
8.
9.     public void sessionCreated(HttpSessionEvent e) {
10.    total++;
11.    current++;
12.
13.    ctx=e.getSession().getServletContext();
14.    ctx.setAttribute("totalusers", total);
```

```
15.         ctx.setAttribute("currentusers", current);
16.              System.out.println("An User session has been started");
   HttpSession session = e.getSession();
       System.out.print(getTime() + " (session) Created:");
       System.out.println("ID=" + session.getId() + " MaxInactiveInterval="
 + session.getMaxInactiveInterval());
17.         }
18.
19.         public void sessionDestroyed(HttpSessionEvent e) {
20.             current--;
21.             ctx.setAttribute("currentusers",current);
      System.out.println("An User session has been destroyed");
       HttpSession session = e.getSession();
       // session has been invalidated and all session data
//(except Id)is no longer available
       System.out.println(getTime() + " (session) Destroyed:ID="
+ session.getId());

22.         }
23.        private String getTime()
   {
      return new Date(System.currentTimeMillis()).toString();
   }

24.      }
```

First.java

```
1.       import java.io.IOException;
2.       import java.io.PrintWriter;
3.
4.       import javax.servlet.ServletContext;
5.       import javax.servlet.ServletException;
6.       import javax.servlet.http.HttpServlet;
7.       import javax.servlet.http.HttpServletRequest;
8.       import javax.servlet.http.HttpServletResponse;
9.       import javax.servlet.http.HttpSession;
10.
11.      public class First extends HttpServlet {
12.      public void doGet(HttpServletRequest request,
13.       HttpServletResponse response)
14.          throws ServletException, IOException {
15.
16.              response.setContentType("text/html");
17.              PrintWriter out = response.getWriter();
18.
19.              String n=request.getParameter("username");
20.              out.print("Welcome "+n);
21.
22.              HttpSession session=request.getSession();
23.              session.setAttribute("uname",n);
24.
25.              //retrieving data from ServletContext object
26.              ServletContext ctx=getServletContext();
27.              int t=(Integer)ctx.getAttribute("totalusers");
28.              int c=(Integer)ctx.getAttribute("currentusers");
29.              out.print("<br>total users= "+t);
30.              out.print("<br>current users= "+c);
31.
```

```
32.             out.print("<br><a href='logout'>logout</a>");
33.
34.             out.close();
35.         }
36.
37.     }
```

Remarque : à la place d'une servlet , on peut créer une page jsp équivalente (First.jsp)

Logout.java

```
1.      import java.io.IOException;
2.      import java.io.PrintWriter;
3.
4.      import javax.servlet.ServletException;
5.      import javax.servlet.http.HttpServlet;
6.      import javax.servlet.http.HttpServletRequest;
7.      import javax.servlet.http.HttpServletResponse;
8.      import javax.servlet.http.HttpSession;
9.
10.
11.     public class LogoutServlet extends HttpServlet {
12.     public void doGet(HttpServletRequest request,
13.      HttpServletResponse response)
14.             throws ServletException, IOException {
15.
16.             response.setContentType("text/html");
17.             PrintWriter out = response.getWriter();
18.
19.             HttpSession session=request.getSession(false);
20.             session.invalidate();//invalidating session
21.
22.             out.print("You are successfully logged out");
23.
24.
25.             out.close();
26.         }
27.
28.     }
```

Utlisation de l'interface HttpSessionBindingListener

- **HttpSessionListener** réagit chaque fois qu'une nouvelle session est créée ou supprimée
- **HttpSessionBindingListener** réagit chaque fois qu'un objet de ce listener est ajouté ou spprimé de la session

Cet écouteur définit les 2 méthodes suivantes.

```
1. public void valueBound (HttpSessionBindingEvent e)
```

Cette méthode sera exécutée automatiquement chaque fois que nous ajouterons un type particulier d'objet à la portée de la session.

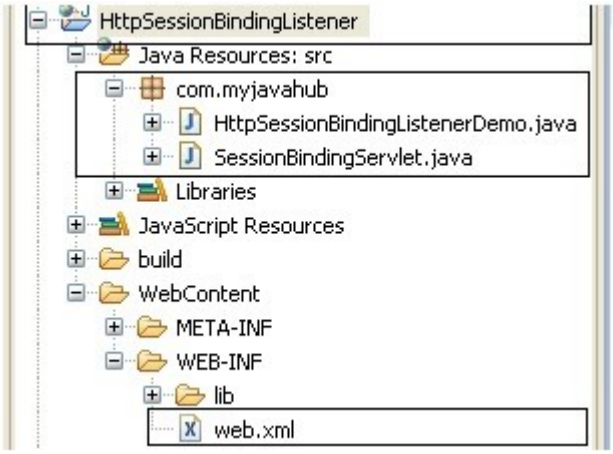
```
2. public void valueUnbound (HttpSessionBindingEvent e)
```

Cette méthode sera exécutée lors de la suppression d'un objet particulier de la session.
Pour l'opération de remplacement, valueBound () sera exécuté suivi de valueUnbound ().

- **HttpSessionBindingEvent**

  Cette classe a deux méthodes:
- **getName (): Renvoie** le nom avec lequel l'objet est lié ou non lié à la session.

- **getSession ():** renvoie la session vers ou à partir de laquelle l'objet est lié ou non lié.
- **Exemple:**
-
- **Étape 1: Structure du projet dans Eclipse**
-



-
- **Étape 2:** Créez une classe d'attributs qui implémente les méthodes d'interface **HttpSessionBindingListener** , valueBound () et valueUnbound ()
-

```
package com.myjavahub;

import javax.servlet.*;
import javax.servlet.http.*;

public class HttpSessionBindingListenerDemo  implements HttpSessionBindingListener
{
	ServletContext context;
	public HttpSessionBindingListenerDemo(ServletContext context)
	{
			this.context = context;
	}
	public void valueBound(HttpSessionBindingEvent event)
	{
			context.log("The value bound is " + event.getName());
	}
	public void valueUnbound(HttpSessionBindingEvent event)
	{
			context.log("The value unbound is " + event.getName());
	}
}
```

- **Étape 3:** Configurer l'écouteur dans le descripteur de déploiement web.xml,
  Remarque: Pour la classe HttpSessionBindingListener, la configuration dans le fichier web.xml avec l'élément <listener> n'est pas requise.

- **Étape 4:** créez une classe de servlet qui ajoute et supprime l'attribut de et vers la session,
-

```
package com.myjavahub;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SessionBindingServlet  extends HttpServlet
```

```
{
	private static final long serialVersionUID = 1L;
	public void doGet(HttpServletRequest req, HttpServletResponse res)
	throws ServletException, IOException
	{
			res.setContentType("text/html");
			PrintWriter out = res.getWriter();
			// Get the current session object, create one if necessary
			HttpSession session = req.getSession();
			// Add a UserName
			session.setAttribute("name", new
HttpSessionBindingListenerDemo(getServletContext()));
			out.println("This is the example of HttpSessionBindingListener");
	}
}
```

- **Étape 5:** Configurez le servlet dans le descripteur de déploiement web.xml,

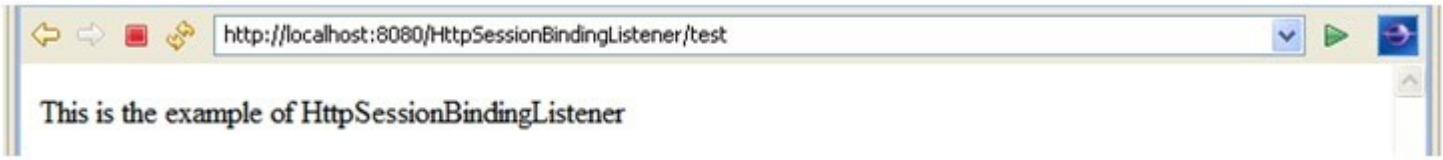
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

<servlet>
	<servlet-name>SessionBindingServlet</servlet-name>
	<servlet-class>com.myjavahub.SessionBindingServlet</servlet-class>
</servlet>
<servlet-mapping>
	<servlet-name>SessionBindingServlet</servlet-name>
	<url-pattern>/test</url-pattern>
</servlet-mapping>
</web-app>
```

- **Étape 6:**

  Testez l'exemple avec l'URL suivante dans le navigateur et vérifiez le journal du serveur / de la console

- http: // localhost: 8080 / HttpSessionBindingListener / test
-
- **Sortie du browser:**

•



- **ServerConsole logs:**

```
INFO: Server startup in 625 ms
Feb 14, 2012 6:40:36 PM org.apache.catalina.core.ApplicationContext log
INFO: The value bound is name
```

Exemple 2 de HttpSessionBindingListener

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpSessionBindingEvent;
import javax.servlet.http.HttpSessionBindingListener;

public class Binder extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
      IOException {
    resp.setContentType("text/html");
    PrintWriter out = resp.getWriter();
    HttpSession session = req.getSession(true);
    SessionObject o = new SessionObject(getServletContext());
    session.setAttribute("Binder.object", o);
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Session Binder</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("Object bound to session " + session.getId());

    out.println("</body>");
    out.println("</html>");
    out.flush();
  }
}

class SessionObject implements HttpSessionBindingListener {
  ServletContext context;

  public SessionObject(ServletContext context) {
    this.context = context;
  }

  public void valueBound(HttpSessionBindingEvent event) {
    context.log("" + (new java.util.Date()) + " Binding " + event.getName() + " to session "
        + event.getSession().getId());
  }

  public void valueUnbound(HttpSessionBindingEvent event) {
    context.log("" + (new java.util.Date()) + " Unbinding " + event.getName() + " from session "
        + event.getSession().getId());
  }

}
```

Exemple 2 de HttpSessionBindingListener

```
package com.exa ;
import java.io.IOException;
```

```
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class HttpSessionAttributeTestServlet extends HttpServlet
{
   private static final long serialVersionUID = 1L;

   public void init() throws ServletException
   {
      System.out.println("----------------------------------------");
      System.out.println(" Init method is called in "
            + this.getClass().getName());
      System.out.println("----------------------------------------");
   }

   public void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      HttpSession httpSession = request.getSession();

      sleep();

      User user = new User("Ram", 43);
      httpSession.setAttribute("userinfo", user);

      sleep();

      httpSession.invalidate();

      out.println("This is the example of HttpSessionBindingListener");

   }

   private void sleep()
   {
      try
      {
         Thread.sleep(10000);
      }
      catch (InterruptedException e)
      {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }
   }

   public void destroy()
   {
      System.out.println("----------------------------------------");
```

```
        System.out.println(" destroy method is called in "
                + this.getClass().getName());
        System.out.println("----------------------------------------");
    }

}
```

User.java : Listener

```
import javax.servlet.http.HttpSessionBindingEvent;
import javax.servlet.http.HttpSessionBindingListener;

public class User implements HttpSessionBindingListener
{

    private String name;
    private int age;

    public User(String name, int age)
    {
        super();
        this.name = name;
        this.age = age;
    }

    public User(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public int getAge()
    {
        return age;
    }

    public void setAge(int age)
    {
        this.age = age;
    }

    @Override
    public void valueBound(HttpSessionBindingEvent httpSessionBindingEvent)
    {
        System.out.println("\n#####################################\n");

        System.out.println("valueBound method has been called in "
                + this.getClass().getName());
```

```
        System.out.println("Added/Replaced Attribute Name ="
                + httpSessionBindingEvent.getName() + ",value = "
                + httpSessionBindingEvent.getValue());

        System.out.println("\n#######################################\n");

        /*
         * if particular Attribute is added/replaced, based on that if you want
         * to perform any operation then you can do it here.
         */

    }

    @Override
    public void valueUnbound(HttpSessionBindingEvent httpSessionBindingEvent)
    {
        System.out.println("\n#######################################\n");

        System.out.println("valueUnbound method has been called in "
                + this.getClass().getName());

        System.out.println("Removed Attribute Name ="
                + httpSessionBindingEvent.getName() + ",value = "
                + httpSessionBindingEvent.getValue());

        System.out.println("\n#######################################\n");

        /*
         * if particular Attribute is removed, based on that if you want to
         * perform any operation then you can do it here.
         */

    }

    @Override
    public String toString()
    {
        return "User [name=" + name + ", age=" + age + "]";
    }

}
```

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_3_0.xsd"
    metadata-complete="true" version="3.0">

    <display-name>HttpSessionBindingListenerDemo</display-name>

    <description>
        This is a simple web application with a source code organization
        based on the recommendations of the Application Developer's Guide.
    </description>
<!--
```

LA DECLARATION PAS OBLIGATOIRE

```
<listener>
   <listener-class>com.exa.User</listener-class>
</listener>

-->
   <servlet>
      <servlet-name>HttpSessionAttributeTestServlet</servlet-name>
      <servlet-class>HttpSessionAttributeTestServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>HttpSessionAttributeTestServlet</servlet-name>
      <url-pattern>/listenerTest</url-pattern>
   </servlet-mapping>

</web-app>
```

index.html

```
<html>
<body>
   <a href="listenerTest">listenerTest</a>
</body>
</html>
```

Exemple 3 de HttpSessionBindingListener

```
import java.sql.*;
import javax.servlet.http.*;

public class JDBCQueryBean implements HttpSessionBindingListener
{
  public void JDBCQueryBean() {   }

  private Connection conn = null;

  private void runQuery() {
    StringBuffer sb = new StringBuffer();
    Statement stmt = null;
    ResultSet rset = null;
    try {
      if (conn == null) {
        DriverManager.registerDriver(new .OracleDriver());
        conn = DriverManager.getConnection("jdbc:oracle:oci8:@",
                                    "scott", "tiger");

      }

      stmt = conn.createStatement();
      rset = stmt.executeQuery ("SELECT * from ....");

    } catch (SQLException e) {
       e.printStackTrace();
  }
    finally {
      try {
        if (rset != null) rset.close();
        if (stmt != null) stmt.close();
      }
```

```
      catch (SQLException ignored) {}
    }
  }

  public void valueBound(HttpSessionBindingEvent event) {
    // do nothing -- the session-scoped bean is already bound
  }

  public synchronized void valueUnbound(HttpSessionBindingEvent event) {
    try {
      if (conn != null) conn.close();
    }
    catch (SQLException ignored) {}
  }
}
```

This is a sample code for JDBCQueryBean, a JavaBean that implements the **HttpSessionBindingListener** interface. The Connection object is bound into the Session and the listener takes care to close the connection as soon as the Session expires.

Utilisation de l'interface de HttpSessionAttributeListener

L'interface HttpSessionAttributeListener est utilisée pour écouter et surveiller les changements (added, removed, and replaced) apportés aux attributs dans une session au sein d'un serveur. HttpSessionAttributeListener définit les trois méthodes suivantes

1. public void attributeAdded (événement HttpSessionBindingEvent)

Cette méthode sera exécutée automatiquement par le conteneur Web au moment de l'ajout de l'attribut à l'objet de session.

2.public void attributeRemoved (événement HttpSessionBindingEvent)

Cette méthode sera exécutée automatiquement par le conteneur Web au moment où l'attribut existant est supprimé de l'objet de session.

3.public void attributeReplaced (événement HttpSessionBindingEvent)

Cette méthode sera exécutée automatiquement par le conteneur Web au moment où la valeur de l'attribut est remplacée par une nouvelle valeur.

HttpSessionBindingEvent

La classe HttpSessionBindingEvent est une classe d'événements utilisée pour les notifications lorsque les modifications sont apportées aux attributs d'une session.

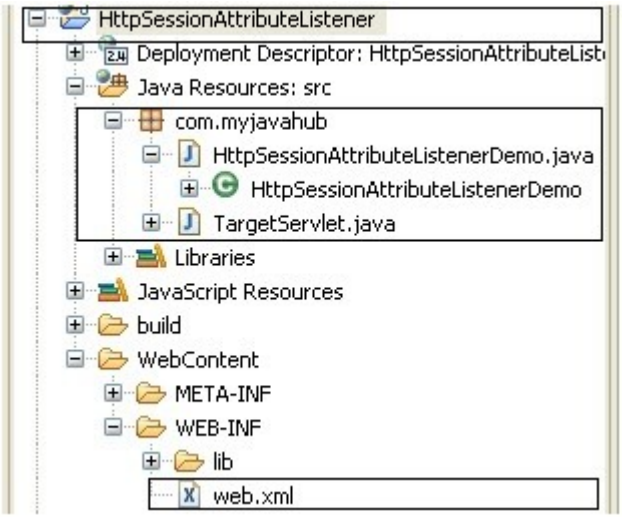
Méthodes:

getName () : renvoie le nom de l'attribut qui a été modifié dans la session.

getValue (): renvoie la valeur de l'attribut qui a été ajouté, supprimé ou remplacé par un autre attribut.

getSession (): retourne la session qui a été modifiée.

Exemple:
Étape 1: Structure du projet dans Eclipse



Étape 2:

Créez une classe qui implémente les méthodes d'interface HttpSessionAttributeListener : attributeAdded (), attributeRemov ed () et attributeReplaced ()
HttpSessionAttributeListenerDemo.java

```
package com.myjavahub;

import javax.servlet.http.HttpSessionAttributeListener;
import javax.servlet.http.HttpSessionBindingEvent;

public class HttpSessionAttributeListenerDemo implements HttpSessionAttributeListener
{

	public void attributeAdded(HttpSessionBindingEvent event)
	{
		String attributeName = event.getName();
		Object attributeValue = event.getValue();
		System.out.println("Attribute added : " + attributeName + " : " + attributeValue);
	}

	public void attributeRemoved(HttpSessionBindingEvent event)
	{
		String attributeName = event.getName();
		Object attributeValue = event.getValue();
		System.out.println("Attribute removed : " + attributeName + " : " + attributeValue);
	}

	public void attributeReplaced(HttpSessionBindingEvent event)
	{
		String attributeName = event.getName();
		Object attributeValue = event.getValue();
		System.out.println("Attribute replaced : " + attributeName + " : " + attributeValue);
	}
}
```

Étape 3: Configurer l'écouteur dans le descripteur de déploiement web.xml,

```
< listener >
</ listener-class >
com.myjavahub.HttpSessionAttributeListenerDemo </ listener-class >
</ listener >
```

Étape 4:

Créer une classe de servlet qui démarre une session et ajoute, remplace et supprime un attribut de session,
TargetServlet.java

```
package com.myjavahub;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class TargetServlet  extends HttpServlet
{
		private static final long serialVersionUID = 1L;

		protected void doGet(HttpServletRequest req, HttpServletResponse resp)
				throws ServletException, IOException
		 {
			PrintWriter out = resp.getWriter();
			out.print("Plz check server logs");

			 HttpSession session = req.getSession();
			 // adding Attribute "site" to session object
			 session.setAttribute("site", "myjavahub");
			 //replacing "site" attribute value with new value
			 session.setAttribute("site","myjavahub.blogspot.com");
			 // removing "site" attribute form session.
			 session.removeAttribute("site");
		}
}
```

Étape 5:

Configurez le servlet dans le descripteur de déploiement web.xml,
web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
	<display-name>HttpSessionAttributeListener</display-name>
<listener>
<listener-class>com.myjavahub.HttpSessionAttributeListenerDemo</listener-class>
</listener>
	<servlet>
	<servlet-name>TargetServlet</servlet-name>
```

```
        <servlet-class>com.myjavahub.TargetServlet</servlet-class>
        </servlet>
        <servlet-mapping>
        <servlet-name>TargetServlet</servlet-name>
        <url-pattern>/test</url-pattern>
        </servlet-mapping>
</web-app>
```

Étape 6:

Testez l'exemple avec l'URL suivante dans le navigateur et vérifiez le journal du serveur / de la console

http: // localhost: 8080 / HttpSessionAttributeListener / test

Sortie:

```
INFO: Server startup in 587 ms
Attribute added : site : myjavahub
Attribute replaced : site : myjavahub
Attribute removed : site : myjavahub.blogspot.com
```

```
package com.exa;
import javax.servlet.http.HttpSessionAttributeListener;
import javax.servlet.http.HttpSessionBindingEvent;

public final class MySessionAttributeListener
      implements HttpSessionAttributeListener {
          //Called after the attribute is added  by method setAttribute


          public void attributeAdded(HttpSessionBindingEvent event) {
System.out.println("one attribute is added in User session");
System.out.println(event.getSession().getId());
            System.out.println(event.getName() + "', '" + event.getValue());
          }

//          Called after the attribute is removed  from session
public void attributeRemoved(HttpSessionBindingEvent event) {
          System.out.println("one attribute is removed form User session");
            System.out.println(event.getSession().getId());
            System.out.println(event.getName() + "', '" + event.getValue());
   }

  //  Called after the attribute is replaced
     public void attributeReplaced(HttpSessionBindingEvent event) {
         System.out.println("one attribute is replaced in User session");
              System.out.println(event.getSession().getId());
              System.out.println(event.getName() + "', '" + event.getValue());
   }
}
```

web.xml

```
          <listener>
    <listener-class>com.exa. MySessionAttributeListener</listener-class>
```

</listener>

Différence entre HttpSessionAttributeListener et HttpSessionBindingListener

L'interface HttpSessionAttributeListener : utilisée pour écouter les attributs ajoutés ou supprimés d'une session. Cette interface possède trois méthodes de rappel: attributeAdded, attributeDeleted et attributeReplaced.
L'interface HttpSessionBindingListener : utilisée pour savoir quand un objet implémentant l'interface est ajouté, modifié ou retiré de la session.. HttpSessionBindingListener n'est pas requis pour configurer dans web.xml
Lorsqu'on configure à la fois attributeListener et bindingListener, BindingListener est exécuté en premier, suivi de AttributeListener.

ListenerDemoServlet.java

```
1       import javax.servlet.http.HttpServletRequest;
2       import javax.servlet.http.HttpServletResponse;
3       import javax.servlet.http.HttpSession;
4
5       @WebServlet(name = "ListenerDemoServlet", urlPatterns = {"/ListenerDemoServlet"})
6       public class ListenerDemoServlet extends HttpServlet {
7
8          @Override
9          protected void doGet(HttpServletRequest request,
10                            HttpServletResponse response)
11             throws ServletException, IOException {
12            HttpSession session = request.getSession();
13            session.setAttribute("stringvalue1", "hello");
14            session.setAttribute("stringvalue2", "world");
15            session.setAttribute("objvalue1", new MyObjectInSession());
16         }
17
18      }
```

MyObjectInSession.java

```
1       package com.quickprogrammingtips.sample.articleproject;
2
3       import javax.servlet.http.HttpSessionBindingEvent;
4       import javax.servlet.http.HttpSessionBindingListener;
5
6       // the listener annotation is optional since container infers it from the interface
7       public class MyObjectInSession implements HttpSessionBindingListener{
8          public MyObjectInSession() {
9          }
10         @Override
11         public void valueBound(HttpSessionBindingEvent event) {
12            System.out.println("My Object is added to session: Key is :"
13                            + event.getName());
14         }
15
16         @Override
17         public void valueUnbound(HttpSessionBindingEvent event) {
18         }
19      }
```

MySessionAttributeListener.java

```
1       package com.quickprogrammingtips.sample.articleproject;
2
3       import javax.servlet.annotation.WebListener;
4       import javax.servlet.http.HttpSessionAttributeListener;
```

```
5       import javax.servlet.http.HttpSessionBindingEvent;
6
7       // The following annotation marks this class as a listener and container automatically registers it.
8       @WebListener
9       public class MySessionAttributeListener implements HttpSessionAttributeListener {
10
11          @Override
12          public void attributeAdded(HttpSessionBindingEvent event) {
13             System.out.println("Attribute added is "+event.getName());
14          }
15
16          @Override
17          public void attributeRemoved(HttpSessionBindingEvent event) {
18          }
19
20          @Override
21          public void attributeReplaced(HttpSessionBindingEvent event) {
22             System.out.println("Attribute replaced is "+event.getName());
23          }
24      }
```

Lorsque la servlet est exécutée, la sortie suivante est visible sur la console du serveur d'applications,

Attribute added is stringvalue1
Attribute added is stringvalue2
My Object is added to session: Key is :objvalue1
Attribute added is objvalue1