Objets principaux de l'API Servlet

P.Mathieu

IUT de Lille
http://www.iut-a.univ-lille.fr
prenom.nom@univ-lille.fr

Plan

Université de Lille

Forme générale d'une Servlet

Université de Lille

```
import ....

@WebServlet("/nompublic")
public class MaServlet extends HttpServlet
{
  public void init(ServletConfig config) throws ServletException
  {
    super.init(config);
    .....
  }

  public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
    {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      .....
    }

  public void destroy()
  {
    super.destroy();
    .....
  }
}
```

Pour être exécutée, une servlet doit être placée dans un conteneur Java EE

- Gère les contextes
  les contextes sont indépendants et peuvent être démarrés ou arrêtés séparément
- Gère les servlets
  - l'invocation des méthodes demandées par les clients
  - le multi-threading
    Chaque requête crée un thread du serveur
  - durée de vie des servlets
    les servlets sont persistantes
  - sécurité

Université de Lille

Un contexte complet peut être archivé sous la forme d'une Web Application Resource (**WAR**)

Création : `jar cvf mnoappli.war monappli`

- Le war est placé tel quel dans `webapps`
- Au lancement du serveur, le war est décompressé s'il n'existe pas déjà
- Le serveur s'occupe du `CLASSPATH` pour l'exécution des objets
- Un serveur Java EE peut aussi déployer automatiquement un WAR "à chaud" via la console d'administration

Tomcat est fournit avec une console d'administration des contextes et de déploiement à chaud

- ajoutez vous le rôle `manager-gui` et/ou `manager-script` dans `conf/tomcat-users.xml`

Plan

Université de Lille

Le Descripteur de déploiement

Le descripteur de déploiement `WEB-INF/web.xml`

Université de Lille

Afin de faciliter leur déploiement, les applications WEB possèdent en général de nombreux paramètres

- Le nom de cette application (`display-name`, `description`)
- La page d'accueil (`welcome-file-list`)
- L'icône à afficher dans le navigateur (`icon`)
- Les constantes à utiliser dans les différents objets (`context-param`)
- ....
- Ces différents paramètres se rangent dans le descripteur de déploiement `web.xml`
- Depuis Tomcat 6, cela peut se faire en partie grâce aux annotations

Le Descripteur de déploiement

La description d'une servlet

Université de Lille

- **Le client** voit une URL. Il n'a pas besoin de connaitre les chemins et répertoires d'installation.
- **Le développeur** connait le nom de la classe et son chemin d'accès
- **Le déployeur** doit pouvoir établir le lien entre l'url et la classe

Distinguer les trois

- Améliore la flexibilité. On peut changer les emplacements sans avoir à toucher au code (même si on n'a pas le source !)
- Améliore la sécurité. Le client ne connait pas la hiérarchie et les différents répertoires.

Le Descripteur de déploiement

Déclaration des servlets

```
<web-app>
    ....
    <servlet>
        <servlet-name>Nom Interne1</servlet-name>
        <servlet-class>but3.Servlet1</servlet-class>
    </servlet>

    <servlet>
        <servlet-name>Nom Interne2</servlet-name>
        <servlet-class>but3.Servlet2</servlet-class>
    </servlet>
    ....
    <servlet-mapping>
        <servlet-name>Nom Interne1</servlet-name>
        <url-pattern>/nomPublic1</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>Nom Interne2</servlet-name>
        <url-pattern>/nomPublic2</url-pattern>
    </servlet-mapping>
    ....
</web-app>
```

Le Descripteur de déploiement

Déclaration de paramètres

Université de Lille

Le descripteur de déploiement (web.xml) autorise des paramètres au niveau des objets (Servlet, Filter, ...), ou au niveau des contextes

```
<web-app>
    ....
    <servlet>
        <servlet-name>Nom Interne1</servlet-name>
        <servlet-class>but3.Servlet1</servlet-class>
        <init-param>
            <param-name>email</param-name>
            <param-value>philippe.mathieu@univ-lille1.fr</param-value>
        </init-param>
    </servlet>
    ....
    <context-param>
        <param-name>driver</param-name>
        <param-value>org.postgresql.Driver</param-value>
    </context-param>
    <context-param>
        <param-name>login</param-name>
        <param-value>Dupont</param-value>
    </context-param>
    ....
</web-app>
```

Le Descripteur de déploiement

Accès aux paramètres du Descripteur de déploiement

Université de Lille

- Les paramètres pour l'ensemble du contexte :
  Accessibles par tous les objets du contexte
  (par ex déclarations relatives au SGBD)
  `getServletContext().getInitParameter(String)`
- Les paramètres d'une servlet :
  Accessibles uniquement par la servlet concernée
  `getServletConfig().getInitParameter(String)`

Ces paramètres ne peuvent pas être modifiés "à chaud" !

Plan

Université de Lille

Maintenir un état entre deux requêtes

Le problème

Université de Lille

- HTTP est un mode sans état (Contrairement à FTP, Telnet ou aux Sockets)
- Après avoir "servi" une page, la connexion est rompue
- Le serveur web ne maintient pas les informations à propos du client entre deux requêtes
- Le serveur ne sait pas déterminer si une requête ou une réponse provient du même client.
- De nombreuses applications nécessitent pourtant d'identifier les requêtes venant du même utilisateur pour conserver un état entre ces requêtes.

Maintenir un état entre deux requêtes

Un exemple

Université de Lille

Pb : On souhaite afficher un compteur pour chaque client qui charge une page

```
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;

@WebServlet("/servlet-Cpt1")
public class Cpt1 extends HttpServlet
{ int cpt=0;

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        cpt++;
        out.println("<head> <title>Implémenter un compteur</title> ");
        out.println("</head> <body>");
        out.println("<h1> Le nombre de chargements est : "+ cpt + "</h1>");
        out.println("</body>");
    }
}
```

Maintenir un état entre deux requêtes

Différentes possibilités

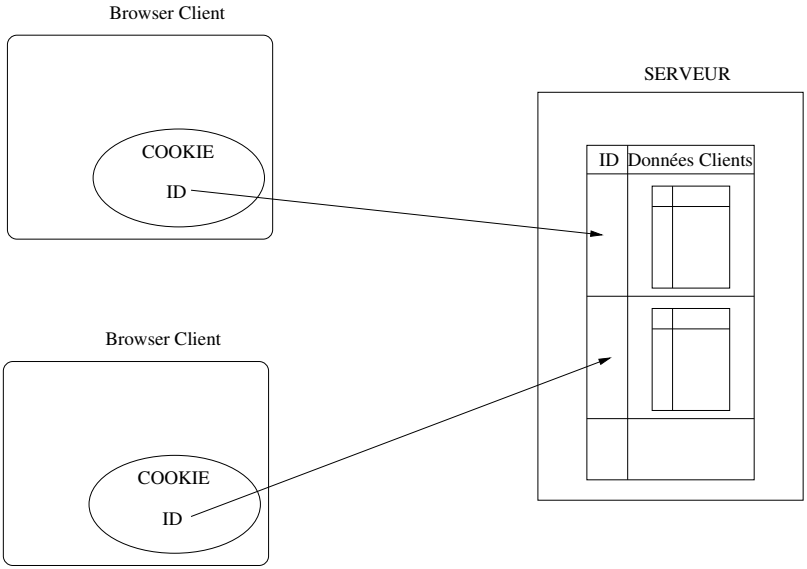
Université de Lille

- Champs cachés
  `<input type=hidden name="cle" value="mavaleur">`
- Concaténation d'URL
  Ajouter l'identifiant du client à la fin de chaque lien
  `<a href="mypage?login=paul&item=article1>...</a>`
- Les cookies
  La première page crée le cookie chez le client
  Les pages suivantes accèdent à ce cookie préalablement.

Inconvénient pour tous : Fonctionne uniquement pour des données textuelles (pas d'objet). Pb de confidentialité !.

Maintenir un état entre deux requêtes

Fonctionnement Idéal



Maintenir un état entre deux requêtes

L'objet `session`

Université de Lille

- HTTP est un protocole non connecté : il n'y a aucun lien permanent entre le serveur et le client.
- Nécessite un dispositif pour créer des attributs par utilisateur.
- Objet `HTTPsession` : dictionnaire rangé dans le serveur permettant l'accès aux attributs des utilisateurs par une clé
- Une `session` est propre à un utilisateur, une machine, un browser et une W.A.R.
- Un cookie stocke l'identifiant de sessions (`JSESSIONID` pour Tomcat) chez le client

Maintenir un état entre deux requêtes

Méthodes principales

Université de Lille

getSession méthode de l'objet `HTTPServletRequest` qui renvoie l'objet `HttpSession` du serveur. S'il n'existait pas, il est alors créé.

getAttribute méthode de l'objet `HttpSession` qui permet de récupérer un objet identifié par une clé dans la session.

setAttribute méthode de l'objet `HttpSession` qui permet de ranger un objet identifié par une clé dans la session.

invalidate méthode de l'objet `HttpSession` qui permet de détruire la session.

setMaxInactiveInterval durée de vie maxi d'une session inactive en millisec

Maintenir un état entre deux requêtes

Principe

Université de Lille

- La session est persistante dans le serveur
- C'est le serveur qui décide quand la détruire
  - S'il est trop chargé
  - Si le time-out est dépassé (`setMaxInactiveInterval`)
- Implémentée physiquement soit à l'aide de Cookies, soit en réécriture d'URL

Maintenir un état entre deux requêtes

compteur revisité

Université de Lille

```java
import java.io.*;
import jakarta.servlet.*;       // pour les servlets
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;

@WebServlet("/servlet-Cpt2")
public class Cpt2 extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        HttpSession session = req.getSession( true );
        Integer cpt = (Integer)session.getAttribute( "compteur" );
        cpt = new Integer( cpt == null ? 1 : cpt.intValue() + 1 );
        session.setAttribute( "compteur", cpt );

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<head> <title>Implémenter un compteur</title>");
        out.println(" </head> <body>");
        out.println("<h1> Le nombre de chargements est : "+ cpt + "</h1>");
        out.println("</body>");
    }
```

Plan

Université de Lille

Persistance de l'information

Université de Lille

	Durée de vie	Technique à utiliser
page	visualisation de la page	variable déclarée dans la methode de service (`doGet() doPost()`)
requête	durée de vie de la requête, y compris si forward du `requestDispatcher`	`req.setAttribute()` `req.getAttribute()`
session	durée de vie du navigateur	`s=req.getSession()` `s.getAttribute()` `s.setAttribute()`
Database	A vie	JDBC

Plan

Université de Lille

Les filtres

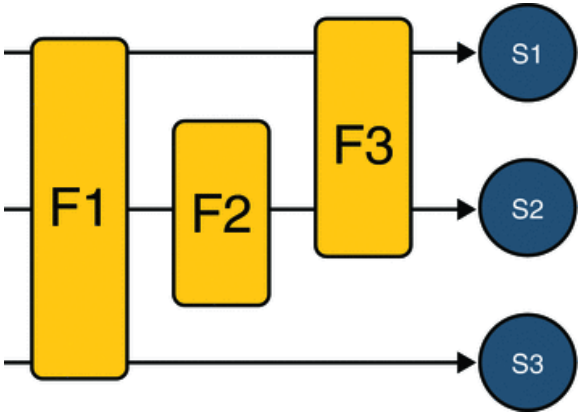
Université de Lille

- Un `filter` filtre des URL !
  en ce sens, il s'intercale entre la requête et les Servlet
- Il permet d'effectuer toute opération précédant ou succédant à l'exécution d'une servlet
- Positionner un filtre n'implique aucune opération sur les servlets
- Les filtres JEE implémentent le pattern "chaine de responsabilité"
- Un filtre hérite de la classe `HttpFilter` (ou implémente `Filter`) et la méthode `doFilter`

Principe

Université de Lille

- L'avantage du filtre est qu'il peut être factorisé pour plusieurs servlet
- Plusieurs filtres peuvent être appliqués en cascade à la même requête.



Usages courants des filtres

Université de Lille

- Encoder ou décoder les données des requêtes
- Loguer les différents appels
- Mesurer des performances d'exécution de requêtes
- Permettre une authentification
- ...

Comment programmer un filtre ?

Université de Lille

- Hériter de `HttpFilter` ou implémenter `Filter`
- Définir une `urlPattern` de filtrage des requêtes

```
@WebFilter(urlPatterns = {"/*"})
public  class MyFilter extends HttpFilter {

    public  void doFilter(HttpServletRequest req, HttpServletResponse res,
                          FilterChain filterChain)
    throws IOException, ServletException {

      // travail effectif du filtre
      // loguer la requête par exemple

       // puis propagation de la requête le long de la chaîne
      filterChain.doFilter(req, res) ;
   }
}
```

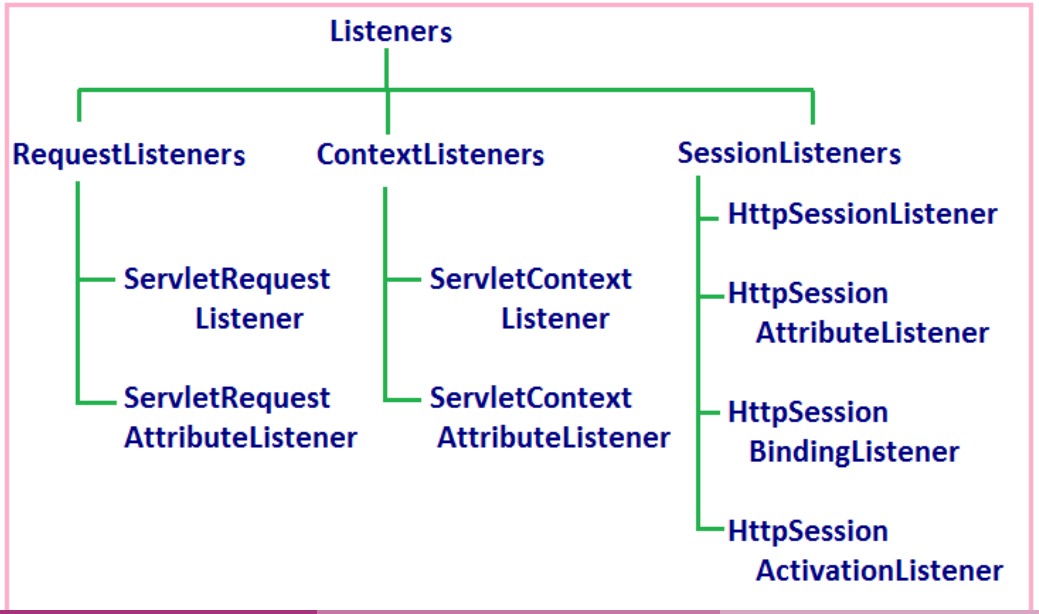
Les Listeners (Observateurs)

Université de Lille

- Un objet Listener est un objet qui écoute certains événements dans une application
- Il informe sur le cycle de vie des objets Request, Session et Context
- Le concepteur configure les observateurs qu'il souhaitent, ceux ci seront déclenchés quand les évènements correspondants se déclencheront
- Attention : contrairement à un filtre, un évènement n'est pas forcément lié à une requête : SessionTimeOut par exemple

Les Listeners sont divisés en 3 groupes

Université de Lille



Comment programmer un Listener ?

Université de Lille

```java
@WebListener
public class MyContextListener implements ServletContextListener
{
   public void contextInitialized(ServletContextEvent event)
   {
     // pour accéder au contexte : event.getServletContext()
   }

   public void contextDestroyed(ServletContextEvent event)
   {

   }
}
```

Listeners et Events

Université de Lille

Servlets Listeners Summary

Object	Event	Event Class	Listener Interface
Web context or Servlet Context	Attribute added, removed, or replaced	ServletContextAttributeEvent	javax.servlet.ServletContextAttributeListener
Session	Attribute added, removed, or replaced	HttpSessionBindingEvent	javax.servlet.http.HttpSessionAttributeListener
Request	Attribute added, removed, or replaced	ServletRequestAttributeEvent	javax.servlet.ServletRequestAttributeListener
Web context or Servlet Context	Initialization and destruction	ServletContextEvent	javax.servlet.ServletContextListener
Session	Session creation Session invalidation Session timeout	HttpSessionEvent	javax.servlet.http.HttpSessionListener
Session	activation, passivation	HttpSessionEvent	javax.servlet.http.HttpSessionActivationListener
Session	Notifies the object that it is being bound to a session Notifies the object that it is being unbound from a session	HttpSessionBindingEvent	javax.servlet.http.HttpSessionBindingListener
Request	Request is Initialized Request is Destroyed	ServletRequestEvent	javax.servlet.ServletRequestListener

Usage de Listeners

Université de Lille

- Surveillance de bon fonctionnement
- Log
- Création de ressources à l'initialisation d'un objet (par ex une connexion au démarrage du contexte)

Annotations vs web.wml

Université de Lille

```
@WebFilter(urlPatterns = "/restreint/*"
initParams=@WebInitParam(name="param1",
                         value="valeur1")
)
```

```
@WebListener
```

```
@WebServlet(urlPatterns={"/test","/ok"}
 initParams=@WebInitParam(name="param1",
 value="valeur1")
 loadOnStartup=1)
```

```
<filter>
    <filter-name>nomInterne</filter-name>
    <filter-class>classeDuFiltre</filter-class>
        <init-param>
            <param-name>param1</param-name>
            <param-value>valeur1</param-value>
        </init-param>
</filter>

<filter-mapping>
    <filter-name>nomInterne</filter-name>
    <url-pattern>/restreint/*</url-pattern>
</filter-mapping>

<listener>
    <listener-class>
      classeDuListener
    </listener-class>
 </listener>

<servlet>
    <servlet-name>nominterne</servlet-name>
    <servlet-class>classeinterne</servlet-class>
  <init-param>
    <param-name>param1</param-name>
    <param-value>valeur1</param-value>
  </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>nominterne</servlet-name>
    <url-pattern>/urlpublique</url-pattern>
</servlet-mapping>
```

Conclusion

Université de Lille

- Les 4 objets principaux de JEE sont `Servlet`, `Session`, `Filter` `Listener`
- `Listener` permet d'auditer facilement une application
- `Filter` et `Listener` se lancent "à chaud" sans aucun lien direct avec l'application
- Ces objets constituent l'épine dorsale de frameworks comme `Spring`