DÉVELOPPEMENT WEB JAVA

SERVLETS/JSP/BD

LI385 – Jean-Loup Guillaume

Programme

- La base : les servlets
  - Du pur java
  - Création de page web directement en Java avec des println
- Le JSP : dissocier le html du code Java
  - Appels java au milieu du code HTML (similaire au php)
- Les bases de données en Java
  - JDBC et MySQL
  - NoSQL (MongoDB)

SERVLETS - INTRODUCTION

Servlets

- Composant Java générant du contenu dynamiquement :
  - Similaire à Php…
  - Les Servlets s'exécutent sur le serveur Web
  - Peuvent être considérées comme le Contrôleur dans le modèle MVC
    - JSP = Vue, classes Java = Modèle

- Avantages = code Java :
  - Tout est géré par la JVM
  - Portable
  - Utilisation de processus légers (threads)
  - Langage compilé (potentiellement plus efficace, code source "caché")

Servlets

- La spécification décrit l'API Servlet (interface)
  - L'implémentation dépend du serveur utilisé
- En pratique, on doit implémenter l'interface javax.servlet.Servlet
  - Soit directement
  - Soit en étendant une classe qui l'implémente
    - GenericServlet ou HttpServlet
    - C'est ce qu'on fera par la suite
- Actuellement version 3.0 de l'API

Servlets - remarques

- Exécution sur le serveur
  - Etablissement de connexions (RMI, …)
  - Appels systèmes/connexions BD/…
  - Manipulation des ressources locales (fichiers, …)

- Pas d'interface graphique
  - Java pur : génération de pages HTML
  - Le navigateur fait le reste

Servlet Container

- Servlet container :
  - Partie du serveur web qui gère les services réseaux
    - Support HTTP obligatoire
    - Autres protocoles possibles
  - Ou ajouté à un serveur d'application gérant le web
  - Gère le cycle de vie des servlets (création, gestion, mort)

- Mécanismes pouvant être implémentés :
  - Cache
  - Modification des requêtes (resp. réponses) avant transmission à la Servlet (resp. client)

Fonctionnement classique

1. Un client (navigateur) fait une requête vers un serveur
2. La requête est transmise au container par le serveur
3. Le container décide à quelle Servlet passer la requête
4. La Servlet :
    1. Utilise l'objet requête pour récupérer les informations nécessaires (utilisateur, paramètres passés, cookies, etc.)
    2. Exécute le code nécessaire (appels BD, génération de code html, etc.)
    3. Retourne un objet réponse au container (code HTML en général)
5. Le container s'assure de l'envoi de la réponse au client et rend la main au serveur

Servlet Container

- Les serveurs implémentent les servlets comme ils veulent
- Respect du cycle de vie des Servlets :
  - Création et initialisation de la Servlet
  - Gestion d'un certain nombre de requêtes
  - Destruction de la Servlet et libération des ressources
- Le reste dépend de l'implémentation :
  - Le serveur peut créer une Servlet par requête (peu efficace)
  - Savoir quand une Servlet va être détruite dépend donc de l'implémentation
  - Persistance des informations dans la Servlet

Développement

- Nombreux outils gratuits, nécessite :
  - Java installé (et savoir programmer)
  - Un serveur web :
    - Serveur classique + moteur de servlets ou
    - Serveur avec support intégré des servlets (Jboss, Apache Tomcat, Glassfish, …)
- Pour que tout soit plus simple :
  - Utiliser un IDE (Eclipse typiquement)
  - Installer les plugins associés au serveur web.

Exemple de configuration

- Java Enterprise Environment 7 :
  - JDK : Java Development Kit dernière version.
  - JRE : Java Runtime Environment dernière version.
- Jboss AS 7.1.1 : serveur d'application.
  - Ou autre
- Eclipse :
  - Version récente (type Juno 4.2 ou plus récent)
  - Avec plugin Jboss : permet de tout gérer dans Eclipse

Une application Web

- Une application web = un espace virtuel
  - Contient html, js, images, servlets, jsp, classes Java, etc.
  - Un fichier web.xml

- Fichier web.xml
  - Décrit ce qui est disponible et comment y accéder :
    - Nom de la servlet, Classe associée, URLs associées
    - À placer dans le répertoire /WEB-INF
    - Généré automatiquement par Eclipse (case à cocher)
  - Optionnel depuis la version 3.0 des servlets (voir annotations)
  - Inutile si aucune servlet/listener/filter

Fichier web.xml - exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    <servlet>
        <servlet-name>Hello</servlet-name>
        <servlet-class>Hello</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Hello</servlet-name>
        <url-pattern>/Hello.html</url-pattern>
    </servlet-mapping>
</web-app>
```

Fichier web.xml - exemple

- Servlet associée à toutes les URLs

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    <servlet>
        <servlet-name>Hello</servlet-name>
        <servlet-class>Hello</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Hello</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

La gestion des erreurs

- Récupération des erreurs :
  - Java (exceptions)
  - http : page inexistante (erreur 404) ou autre
- Association d'une page à ces erreurs dans web.xml

```
<error-page>
    <exception-type>java.lang.ArithmeticException</exception-type>
    <location>/exception.jsp</location>
</error-page>

<error-page>
   <error-code>404</error-code>
   <location>/erreur</location>
</error-page>
```

SERVLETS – LA BASE

Cycle de vie

- Création de la Servlet
  - Géré par le container (création d'une instance)
- Initialisation
  - Méthode "init"
- Appel
  - Méthode "service"
  - Récupération de la requête et création de la réponse
- Destruction
  - Méthode "destroy"

Méthodes d'une Servlet

```
void init(ServletConfig config)
throws ServletException
```

- init est appelé avant la première requête :
  - On peut faire ce qu'on veut, notamment des choses longues qu'on ne souhaite pas faire à chaque fois (connection à une BD)
  - Ne pas passer de paramètres à la fonction si on étend la méthode init (voir GenericServlet plus loin)
    - Sinon on récupère des informations générales sur le contexte d'exécution

Méthodes d'une Servlet

```
void service(ServletRequest req, ServletResponse res)
throws ServletException, IOException
```

- ServletRequest : informations venant du client :
  - Paramètres passés (par GET ou POST)
  - Session, Cookies, …
- ServletResponse : réponse pour le client :
  - Typiquement envoi du code HTML

- On verra plus loin comment utiliser ces objets

Méthodes d'une Servlet

```
void destroy()
```

- Destruction de la servlet :
  - On libère tout ce qu'on aurait pu allouer
  - Fermeture de connexion BD
  - ...

Méthodes d'une Servlet

```
ServletConfig getServletConfig()
```

- Getter pour la configuration de la Servlet :
  - L'objet ServletConfig a été passé à la méthode init
  - init doit sauver des objets pour que cette méthode le retourne à la demande

Méthodes d'une Servlet

```
abstract String getServletInfo()
```

- Informations sur la Servlet :
  - Auteur, version, …
  - Retourne une chaîne en texte brut

Exemple – compteur d'appels

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

public class Hello implements Servlet {
          private int count;
          private ServletConfig config;

          public void init(ServletConfig config) throws ServletException {
                    this.config=config;
                    count=0;
          }
          public ServletConfig getServletConfig() {
                    return this.config;
          }
          public String getServletInfo() {
                    String t = "info sur la servlet";
                    return t;
          }
          public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {
                    count++;
          }
          public void destroy() {}
}
```

Paramètres d'initialisation

- Paramètre pour une Servlet :
  - Dans la définition de la Servlet
- Paramètre de l'application :
  - Lié à l'application complète et pas à une Servlet particulière

```
<servlet>
  <servlet-name>Hello</servlet-name>
  <servlet-class>Hello</servlet-class>
  <init-param>
    <param-name>message1</param-name>
    <param-value>niveau servlet</param-value>
  </init-param>
</servlet>
<context-param>
  <param-name>message2</param-name>
  <param-value>niveau application (contexte)</param-value>
</context-param>
```

Paramètres d'initialisation

- Récupération via ServletConfig
  - Passé en paramètre à init()
  - Ou récupéré ailleurs avec getServletConfig()

```
private String message1;
private String message2;

public void init(ServletConfig c)
throws ServletException {
  super.init(c);
  message1 = c.getInitParameter("message1");
  message2 = c.getServletContext().getInitParameter("message2");
}
```

Concurrence

- Les Servlets sont potentiellement exécutées en parallèle (requête concurrentes) :
  - Faire attention aux problèmes de synchronisation, type écriture dans un fichier par plusieurs clients en même temps
  - En dehors du cadre du cours

- Remarques :
  - SingleThreadModel déprécié
  - Synchronized : ne pas utiliser sur la méthode service (pb de performances)

GENERIC SERVLET

Classe GenericServlet

- Une implémentation de Servlet et de ServletConfig.
  - En pratique on utilise GenericServlet ou HttpServlet
  - Versions de base des méthodes :
    - Simplifie la conception
  - La méthode service doit par contre être codée entièrement (et éventuellement getServletInfo)

- La méthode service utilise :
  - ServletRequest
  - ServletResponse

ServletRequest

- Récupération des valeurs des paramètres :
  - `String getParameter(String name)`
  - `String[] getParameterValues(String name)`

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

public class Hello extends GenericServlet {
  public void service(ServletRequest req, ServletResponse res)
  throws ServletException, IOException {
    String t = req.getParameter("test");
    PrintWriter out = res.getWriter();
    out.println(t + "<br />");
  }
}
```

ServletRequest

- Récupération des paramètres :
  - `Enumeration getParameterNames()`
  - `Map getParameterMap()`

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

public class Hello extends GenericServlet {
  public void service(ServletRequest req, ServletResponse res)
  throws ServletException, IOException {
    PrintWriter out = res.getWriter();
    for (Enumeration e = req.getParameterNames () ; e.hasMoreElements() ;) {
      out.println(e.nextElement());
    }
  }
}
```

ServletRequest – autres méthodes

- Entête de la requête :
  - int getContentLength()
  - String getContentType()
  - String getProtocol()

- Contenu de la requête (transfert de fichier par ex.)
  - ServletInputStream getInputStream()
  - BufferedReader getReader()

ServletResponse

- Données à envoyer au client.

- Création du contenu :
  - `ServletOutputStream getOutputStream()`
  - `PrintWriter getWriter()`
  - (Il faut choisir entre les deux).

- Création de l'entête (par exemple HTTP) :
  - `void setContentLenght(int length)`
    - Taille du contenu, placé dans l'entête HTTP "Content-Length"
  - `void setContentType(String mime)`
  - `void setCharacterEncoding(String encoding)`
  - ...

Exemple : un compteur d'appels

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

public class Hello extends GenericServlet {
	int count;

	public void init()
	throws ServletException {
		count=0;
	}

	public void service(ServletRequest req, ServletResponse res)
	throws ServletException, IOException {
		count++;
		res.setContentLength(10);
		PrintWriter out = res.getWriter();
		out.println(count + " appels et du texte qui ne va pas forcément s'afficher");
	}
}
```

ServletResponse – autres méthodes

- `void setBufferSize(int size) / int getBufferSize()`
  - Taille du buffer avant envoi.
  - Utile si on a beaucoup de données à envoyer.
- `int flushBuffer()`
  - Force l'envoi au client
- `void reset()`
  - Vide tout, entêtes inclus
- `boolean isCommitted()`
  - Permet de savoir si la réponse a été envoyée
- …

HTTPSERVLET

HttpServlet

- Sous-classe de GenericServlet spécifique pour le web :
  - A utiliser dans la grande majorité des cas

- 2 méthodes à utiliser en remplacement de service() :
  - void doGet(HttpServletRequest req, HttpServletResponse res)
    - requêtes HTTP GET
  - void doPost(HttpServletRequest req, HttpServletResponse res)
    - requêtes HTTP POST
  - service() appelle automatiquement la bonne méthode en fonction du type de la requête

  - Au moins une des deux doit être redéfinie
    - L'une peut appeler l'autre si les traitements sont les mêmes.

Discussion client – serveur

- Comme avant, on utilise les objets
  - HttpServletRequest : pour la réception de données
    - via la méthode getParameter() par exemple
  - HttpServletResponse : pour l'envoi de données
    - en créant un writer avec getWriter() par exemple

Un exemple

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

public class Hello extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException {
    String value1 = req.getParameter("param1");
    if(value1 == null) {...}

    PrintWriter out = res.getWriter();
    out.println("Bonjour");
    out.flush();
  }
}
```

Un autre exemple (version HTML)

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

public class Hello extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println(
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n" +
      "<html>\n" +
      "<head><title>Bonjour</title></head>\n" +
      "<body><h1>Bonjour</h1></body></html>");
  }
}
```

Informations sur la requête

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
       res.setContentType("text/plain");
       PrintWriter out = res.getWriter();
       out.println("Protocol: " + req.getProtocol());
       out.println("Scheme: " + req.getScheme());
       out.println("ServerName: " + req.getServerName());
       out.println("ServerPort: " + req.getServerPort());
       out.println("RemoteAddr: " + req.getRemoteAddr());
       out.println("RemoteHost: " + req.getRemoteHost());
       out.println("Method: " + req.getMethod());
       out.println("requestURI: " + req.getRequestURI());
       out.println("ServletPath: " + req.getServletPath());
       out.println("PathInfo: " + req.getPathInfo());
       out.println("PathTranslated: " + req.getPathTranslated());
       out.println("QueryString: " + req.getQueryString());
       out.println("RemoteUser: " + req.getRemoteUser());
       out.println("AuthType: " + req.getAuthType());
       out.flush();
}
```

Redirection et inclusion

- Objectif :
  - Utiliser d'autres servlets pour répondre au client :
    - transfert de la requête à une autre requête
    - ou inclusion de la réponse d'une autre servlet
  - Transparent pour le client : tout se fait au niveau du serveur.

- Se fait via l'interface RequestDispatcher et deux fonctions :
  - forward(request,response)
  - include(request,response)

- Pour obtenir un RequestDispatcher :
  - getServletContext().getRequestDispatcher(path)

Forward

- Faire suivre la requête à une autre servlet :
  - Sert à rediriger uniquement
  - Pas à inclure des choses avant ou après.

```
protected void doGet(HttpServletRequest req,
HttpServletResponse res)
throws ServletException, IOException {
  RequestDispatcher rd = req.getRequestDispatcher("Hello");
  rd.forward(req,res);
}
```

Include

- Inclure le résultat d'une autre requête.
- Permet d'inclure plusieurs résultats d'autres pages
  - Attention aux ajouts de éventuels de balises <html> ou <body> par les servlets appelées.

```
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
     res.setContentType("text/html");
     RequestDispatcher rd = req.getRequestDispatcher("Hello");
     rd.include(req, res);
     rd.include(req, res);
}
```

Exemple

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

public class Hello extends GenericServlet {
  boolean hasBeenCalled = false;
  public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {
    if(hasBeenCalled) {
      PrintWriter out = res.getWriter();
      out.println("Ceci est affiché par la servlet fille");
      this.hasBeenCalled=false;
    } else {
      this.hasBeenCalled=true;
      RequestDispatcher rd = req.getRequestDispatcher("Hello.html");
      rd.include(req, res);
      PrintWriter out = res.getWriter();
      out.println("Ceci est affiché par la servlet pere");
    }
  }
}
```

Remarques

- On perd du temps si on doit retaper les entêtes HTML à chaque fois :
  - Créer des méthodes, attributs ou toute autre solution pour alléger le doGet();
- Passer au JSP :
  - Seulement après avoir fait des servlets "à la main"

SERVLETS
CONTEXTE ET SESSIONS

Problématique

- HTTP = protocole non connecté
  - Chaque requête est indépendante des précédentes.
  - Différent de Telnet, Ftp, …
- Besoin de stocker des informations entre des requêtes successives
  - Equivalent à être en mode connecté

Solutions avec les servlets

- Utilisation des cookies
- Réécriture d'URL
  - Ajouter des informations d'identification dans l'URL
  - Problème : taille, caractères autorisés, sécurité
- Utilisation de champs de formulaire "hidden"
  - Mêmes problèmes

- Utilisation de ServletContext ou de HttpSession

Sauvegarde d'information

- Deux moyens de sauvegarder de l'information

- Contexte = configuration de la servlet
  - Via l'objet ServletContext.
    - Ou indirectement avec ServletConfig (cf fonction init).
  - Méthodes pour obtenir la configuration, échanger des données…

- Session = session HTTP
  - HTTP est sans état, on maintient une session en faisant passer des informations d'un manière ou d'une autre (cookie, URL rewriting, …)

ServletContext

- Obtenu par la méthode getServletContext()

- On peut y placer des attributs
  - `void setAttribute(String name, Object value)`
    - Stocke un objet quelconque accessible par son nom
- En récupérer
  - `Enumeration getAttributeNames()`
    - Liste les noms d'attributs disponibles dans le contexte
  - `Object getAttribute(String name)`
    - Récupérer un objet par son nom
- Ou en supprimer
  - `void removeAttribute(String name)`

L’objet session

- Simple avec l'API des servlets
  - objet HttpSession

- Principe :
  - Un objet "session" peut être associé à chaque requête
  - Il va servir de "container" pour des informations persistantes
  - Durée de vie limitée et réglable

Exemple de base

```
HttpSession session = request.getSession(true);
Caddy caddy = (Caddy) session.getValue("caddy");

if(caddy != null) {
  // le caddy n'est pas vide !
  afficheLeContenuDuCaddy(caddy);
} else {
  caddy = new Caddy();
  caddy.ajouterUnAchat(request.getParameter("id"));
  session.putValue("caddy", caddy);
}
```

Méthodes de la classe HttpSession

- getID()
  - Génération de l'identifiant de session
- isNew()
  - Vrai si la session a été créée mais pas encore envoyée au navigateur.
- getCreationTime() / getLastAccessedTime()
  - Heure de création de la session
- getMaxInactiveInterval()
  - Durée de vie de la session
- invalidate()
  - Supprime la session en cours

FILTRES

Principe

- Modification à la volée :
  - Des entêtes et contenus
  - Avant le passage de la requête à la Servlet
  - Avant l'envoi de la réponse au client
- Exemples :
  - Authentification
  - Compression
  - Encryption
  - Conversion d'image
- Equivalent à un include

Exemple - Filtre

- Utilisation de doFilter :
  - Transfère la requête au filtre/servlet suivant

```
public class HelloFilter implements Filter {
  ...
  public void doFilter(ServletRequest req,
                       ServletResponse res,
                       FilterChain chain)
  throws IOException, ServletException {
    PrintWriter out = res.getWriter();
    out.println("Avant Servlet");
    chain.doFilter(req, res);
    out.println("Après Servlet");
  }
}
```

Exemple – web.xml

```
...
<filter>
  <filter-name>Filtre inutile</filter-name>
  <filter-class>HelloFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>Filtre inutile</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
...
```

LISTENERS

Les listeners

- Permet de suivre le cycle de vie d'une Servlet :
  - Lié au contexte :
    - Création/destruction : javax.servlet.ServletContextListener
    - Modification des attributs : javax.servlet.ServletContextAttributeListener
  - Lié à la Servlet :
    - Création/destruction : javax.servlet.ServletRequestListener
    - Modification des attributs : javax.servlet.ServletRequestAttributeListener
  - Lié à la session http :
    - Création/destruction : javax.servlet.HttpSessionListener
    - Javax.servlet.HttpSessionAttributeListener
- Selon le Listener créé on peut observer différents événements

Exemple

```
import javax.servlet.ServletContext;
import javax.servlet.ServletRequest;
import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
import javax.servlet.http.HttpServletRequest;

public class RequestListener implements ServletRequestListener {
  private static long reqCount;

  public void requestInitialized(ServletRequestEvent sre) {
    ServletContext context = sre.getServletContext();
    HttpServletRequest request = (HttpServletRequest) sre.getServletRequest();
    context.log("request.getRequestURI() + " : appels=" + ++reqCount);
  }

  public void requestDestroyed(ServletRequestEvent sre) {}
}
```

Annotations

- Objectif : alléger le fichier web.xml
  - Annotation mise avant la déclaration de la classe
  - Plus de détails dans la doc

- Servlets :
  - @WebServlet("/Test")
- Filtres :
  - @WebFilter("/*")
- Listener :
  - @WebListener

SERVLETS ET COOKIES

Manipuler les cookies

- Tout est inclus dans l 'API des servlets :
  - Méthode addCookie(cookie) de HttpServletResponse
    - Ajoute le cookie à la réponse (pour stockage chez le client)
  - Méthode getCookies() de HttpServletRequest
    - Retourne tous les cookies envoyés par le client
  - Classe Cookie pour créer des cookies :
    - Cookie(String name, String value)

Méthodes de la classe Cookie

- `String getValue() / void setValue(String)`
  - Valeur contenue dans le cookie
- `String getName()`
  - Nom du cookie
- `String getComment() / void setComment(String)`
  - Ajout d'un commentaire sur le cookie
- `int getMaxAge() / void setMaxAge(int)`
  - Durée de vie du cookie en secondes
  - Par défaut -1 : jusqu'à la fin de la session courante
- …

Récupération des cookies

```
public class Hello extends HttpServlet {
  int count=0;

  private static String getCookieValue(Cookie [] cookies, String cookieName, String defaultValue) {
    for(int i=0 ; i < cookies.length ; i++)
      if(cookieName.equals(cookies[i].getName()))
        return(cookies[i].getValue());
      return(defaultValue);
  }

  public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
    Cookie [] cookies = request.getCookies();
    String cookieName="nom";
    String cookieValue = getCookieValue(cookies, cookieName, "inconnu");
    PrintWriter out = response.getWriter();
    out.println("contenu du cookie " + cookieName + " : " + cookieValue);
    response.addCookie(new Cookie("nom",String.valueOf(++count)));
  }
}
```