Java côté serveur

Eugen Dedu
Eugen.Dedu@pu-pm.univ-fcomte.fr
http://lifc.univ-fcomte.fr/~dedu/

UFC, IUP 3ème année
Montbéliard
septembre 2004

Contexte et motivations

- Moteurs de recherche, e-commerce, météo, news, ...
- => Besoin de pages Web dynamiques
  - scripts exécutés par le serveur ou par le client
- Deux façons :
  - page HTML incluant un script :
    - JavaScript, Flash, Java
  - page écrite en un autre langage :
    - CGI (Common Gateway Interface)
    - PHP (Php's Hypertext Preprocessor)
    - Java (servlets = server applets)

Contexte et motivations - méthodes côté serveur -

- CGI
  - utilisation de tout langage
  - lent au démarrage (1 processus / appel)
- PHP
  - facile à la programmation
- Servlet
  - puissance du langage, des bibliothèques
    - création de sites de chats, sites e-commerce
  - portabilité
  - sécurité, ...

Servlets / JSP
- utilisation dans le monde réel -

- Avantages Java côté serveur :
  - sites complexes
- Exemples :
  - Moteur de recherche excite : http://www.excite.com
  - IEEE : http://www.ieee.org
  - Delta Airlines : http://www.delta.com
  - Banque First USA : http://www.firstusa.com
  - OFoto (Kodak) : http://www.ofoto.com
- http://www.securityspace.com/s_survey/data/index.html
  - Apache Modules, Technology Penetration, Cookie Usage

Installation logiciel sous Debian GNU/Linux

- Installer paquetage tomcat4 (et tomcat4-webapps), qui fournit serveur Web, moteur de servlet, bibliothèque servlets, ...
  - port 8180
- Installer compilateur java
- Plusieurs configurations à faire, plus d'info : http://www.coreservlets.com/Apache-Tomcat-Tutorial/

Biblionetographie

- “Programmation Java côté serveur”, Andrew Patzer, janvier 2000
- http://courses.coreservlets.com/Course-Materials/
- http://java.sun.com/products/servlet/
- http://java.sun.com/products/jsp/
- http://java.sun.com/products/javabeans/
- http://home.earthlink.net/~alxdark/software/wcd-guide/, “Component Developer Certification”
- http://jakarta.apache.org/tomcat/tomcat-4.1-doc/servletapi/

Plan

- Servlet HelloWWW
  - rappel : en-têtes en HTTP
- Cycle de vie d'une servlet
- Classes de base
- Gestion des erreurs, journalisation
  - rappel : formulaires HTML
- Sécurité dans les données d'entrée
- Validation HTML
- Sessions
- Contexte des servlets

En-têtes en HTTP
- envoi des paramètres -

- Client -> serveur : en-tête (avec adresse)
- Serveur -> client : en-tête + contenu
- URL classique :
  - http://www.google.com/search
- Comment envoyer des paramètres (le mot cherché) ?
  - GET : ajoutés à l'adresse, paires clé-valeur
    - http://www.google.com/search?hl=en&q=iup+montbeliard
  - POST : ajoutés à l'en-tête du document
  - (HEAD : récupère juste l'ên-tête du document)

En-têtes en HTTP
- GET vs. POST -

- Apparition :
  - méthode POST peut apparaître si et seulement si formulaire avec méthode POST
- Utilisation :
  - GET ne doit pas avoir des effets de bord (i.e. peut être répétable)
  - POST peut avoir des effets de bord et peut ne pas être répétable (e.g. payer une commande)

En-têtes en HTTP - requête du client -

- GET /search?hl=en&q=stgi+montbeliard HTTP/1.1
- Accept: image/png, image/jpg, image/gif, */*
- Cookie: uid=158793
- Host: www.google.com
- Referer: http://www.google.com
- User-Agent: Mozilla/5.0 [en] (...)
- ...
- (ligne vide)

En-têtes en HTTP - réponse du serveur -

- HTTP/1.1 200 OK
- Date: Mon 06 Oct 2003 09:00:00 CEST
- Server: Apache/2.0.47 (Unix) PHP/4.0
- Last-Modified: Mon, 17 May 1999 ...
- Content-Length: 2398
- Content-Type: text/html; charset=utf-8
- (ligne vide)
- <HTML><HEAD>
- ...

Première servlet : HelloWWW - exécution d'une servlet -

- Écrire la servlet :
  - lire les éventuels paramètres de la requête
  - faire le traitement nécessaire
  - écrire les éventuels en-têtes
  - obtenir l'objet d'écriture et écrire à la sortie la page Web
- La compiler
  - export CLASSPATH=/usr/share/java/servlet-2.3.jar
    - utilise le JSDK (Java Servlet Development Kit)
- mv *.class ~/public_html/WEB-INF/classes
- Lire la page avec un navigateur : ~user/servlet/...

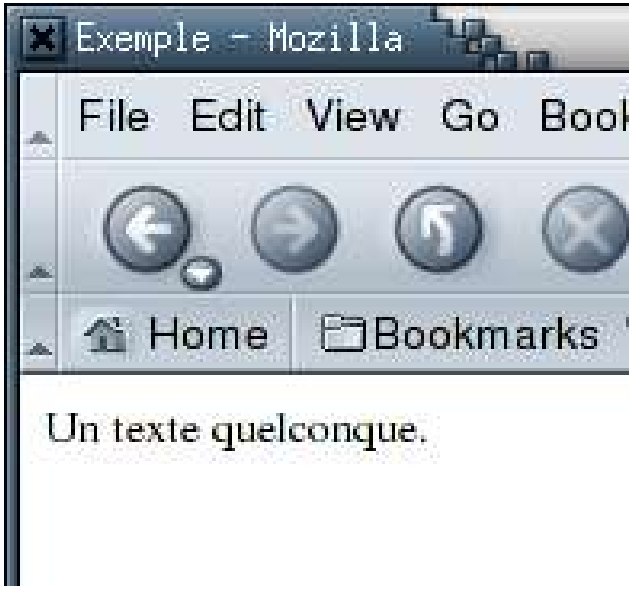
Première servlet : HelloWWW - réponse texte -

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

import java.io.*;

public class HelloWWW extends HttpServlet{
    public void doGet (HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException{
        res.getWriter().println (“Hello WWW!”);
    }
}
```

Première servlet : HelloWWW - réponse HTML -

- Méthode doGet () :

```
res.setContentType ("text/html");
PrintWriter out = res.getWriter ();
out.println (“<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Strict//FR\">”);
out.println ("<HTML>”);
out.println (“<HEAD>”);
out.println ("<TITLE>Exemple</TITLE>");
out.println ("</HEAD><BODY>");
out.println ("Un texte quelconque.");
out.println ("</BODY></HTML>");
out.close ();
```



Cycle de vie d'une servlet - conteneur (moteur) de servlet -

- Il s'interpose (fait la liaison) entre le serveur et la servlet
- Buts :
  - gestion des requêtes client
  - transmission des requêtes à une servlet
  - envoi des requêtes au client

Cycle de vie d'une servlet - non optimisé -

- Une instance par requête
- Le conteneur boucle sur chaque requête :
  - il crée une instance de la servlet
  - il appelle sa méthode init ()
  - il appelle sa méthode service ()
  - il appelle sa méthode destroy ()
  - il détruit l'instance (=> garbage collector)

Cycle de vie d'une servlet - optimisé -

- Évite la création / destruction d'instances :
  - le conteneur crée une instance de la servlet
  - il appelle sa méthode init ()
  - boucle sur chaque requête :
    - il appelle sa méthode service ()
  - il appelle sa méthode destroy ()
  - il détruit l'instance (=> garbage collector)

- En réalité, pool de threads => utiliser synchronized en service ()

Classes de base
- interface Servlet -

- Toute servlet doit l'hériter :
  - directement
  - indirectement, par :
    - GenericServlet
    - HttpServlet
- void init (ServletConfig)
  - initialisation de la servlet
  - appelée automatiquement

Classes de base
- interface Servlet -

- void service (ServletRequest, ServletResponse)
  - exécution de la servlet
  - appelée automatiquement
  - ServletRequest : accès aux données d'entrée
  - ServletResponse : construction de la réponse (la sortie)
- void destroy ()
  - destruction de la servlet

Classes de base
- interface Servlet -

- ServletConfig getServletConfig ()
  - informations sur la configuration des servlets
  - des informations passées du conteneur à la servlet (e.g. contexte d'exécution)
- String getServletInfo ()
  - retourne des informations sur la servlet (auteur, version, ...)

Classes de base
- classe GenericServlet -

- Servlet élémentaire
- Implante toutes les méthodes, sauf service()
  - => la classe est abstraite
- Classe indépendante du protocole

Classes de base
- classe HttpServlet -

- Extension de GenericServlet spécifique à HTTP
- Utilisée dans la plupart des cas
- service() détermine le type de la requête et appelle la méthode appropriée :
  - doGet (), doPost (), doHead (), ...
  - par défaut, ces méthodes renvoient une erreur
- En pratique, on surcharge la méthode doGet ()
- long getLastModified (HttpServletRequest)
  - le temps écoulé entre 1970 et la dernière modification de la servlet

Classes de base
- interface HttpServletRequest -

- Permet d'accéder aux paramètres de l'adresse et à l'en-tête
  - http://www.google.com/search?hl=en&lr=&ie=UTF-8&oe=UTF-8&q=servlet+online+course&spell=1
- String getParameter (String)
  - getParameter ("hl") retourne "en"
- String[] getParameterValues (String)
  - renvoie toutes les valeurs d'un même paramètre
- Enumeration getParameterNames ()
- String getHeader (String)
- cookie, informations sur le client, ...

Classes de base
- interface HttpServletResponse -

- Permet de générer la réponse (la sortie)
- void setContentType (String)
  - "text/html" pour HTML
- void setHeader (String, String)
  - en-têtes personnalisées

Classes de base
- interface HttpServletResponse -

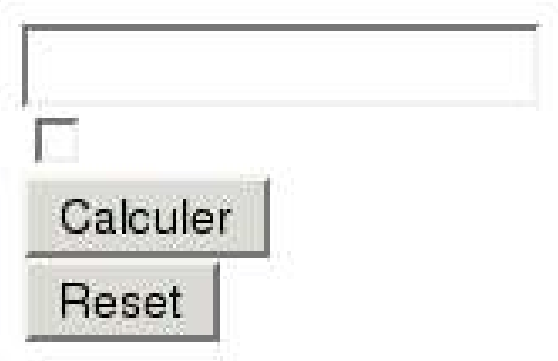
- Écrire sur la sortie, 2 méthodes exclusives :
  - PrintWriter getWriter ()
    - pour écrire du texte
    - .println (String)
  - ServletOutputStream getOutputStream ()
    - pour écrire du binaire
- Forcer l'écriture :
  - .flush ()

Formulaires en HTML
- balises -

- <FORM>...</FORM>, formulaire :
  - ACTION, METHOD, ENCTYPE
- <INPUT>, champ d'entrée :
  - NAME, TYPE, VALUE, CHECKED, SIZE, MAXLENGTH
- <SELECT>...</SELECT>, liste de sélection :
  - NAME, MULTIPLE, SIZE
- <OPTION>, option dans une liste de sélection :
  - SELECTED
- <TEXTAREA>...</TEXTAREA>, zone d'édition :
  - NAME, ROWS, COLS

Formulaires en HTML
- exemple -

- <FORM METHOD=POST ACTION="http://...">
  - <INPUT NAME="nom" TYPE=text>
  - <INPUT NAME="error" TYPE=checkbox>
  - <INPUT TYPE="submit" VALUE="Calculer">
  - <INPUT TYPE=reset>
- </FORM>



http://www.uwa.edu.au/web/office/authors/howto/writing_html_forms

Traitement des erreurs - modalités -

- Comment spécifier à l'utilisateur que les données qu'il a introduites sont incorrectes ?
- Faire une page avec le texte d'erreur
- Ou utiliser les erreurs HTTP
  - meilleur, car elles sont consignées dans le journal (*log*) d'erreurs HTTP
  - en regardant le journal on peut voir les erreurs fréquentes des utilisateurs

Traitement des erreurs - envoi d'erreurs HTTP -

- Méthodes de HttpServletResponse :
  - void sendError (int codeErreur)
  - void sendError (int codeErreur, String msg)
    - spécifie le message donné sur la page d'erreur
- Donnent la main au serveur Web => finissent le programme

Traitement des erreurs - codes d'erreurs HTTP -

- 100-199 Informations
- 200-299 Succès de la requête client
  - 200, SC_OK
- 300-399 Requête client réorientée
- 400-499 Requête client incomplète
  - 400, SC_BAD_REQUEST, syntaxe incorrecte
  - 404, SC_NOT_FOUND, ressource inaccessible

Traitement des erreurs - codes d'erreurs HTTP -

- 500-599 Erreur de serveur
  - 500, SC_INTERNAL_SERVER_ERROR, erreur dans le serveur
  - 503, SC_SERVICE_UNAVAILABLE, serveur surchargé
- http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

- ServletException renvoie généralement erreur 500

Journalisation des événements - pourquoi -

- Stockent des événements dans un fichier journal (*log*), exemples :
  - site e-commerce :
    - les destinations d'avion pour lesquelles les billets ont été achetés
    - les livres les plus recherchés
  - site visite virtuelle :
    - les peintures les plus accédées
  - site enseignement à distance :
    - les cours les plus accédés
  - débogage des servlets !

Journalisation des événements - comment -

- Dans une classe héritant ServletConfig (e.g. HttpServlet) :
  - ServletContext servletContext = getServletContext ()
- servletContext :
  - void log (String)
  - void log (String, Throwable)
    - sauvegarde la trace de la pile aussi

Sécurité dans les données d'entrée - exemple -

- Une application Web (e.g. servlet) qui affiche le texte tapé par le client dans un editbox
  - le client introduit le texte “<script>piège</script>”
    - (ou <script src='http://bad-site/badfile'></script>)
  - il clique sur le bouton Submit
  - l'application fait copier-coller du texte dans la sortie qu'il génère
  - => le client / navigateur exécute le script à son insu
    - (il faut que le client autorise l'exécution des scripts)

Sécurité dans les données d'entrée - exemple -

- Un client peut recevoir du texte d'un autre client :
  - sur un même site (e.g. une application chat)
  - sur un autre site (e.g. lien piège sur site A : http://B/comment.pl?mycomment=<script>...</script>)

Sécurité dans les données d'entrée - considérations -

- Apparition de la vulnérabilité :
  - on écrit des données qu'on ne valide pas
- Conséquences :
  - lecture des cookies
  - lecture des données d'un formulaire
    - clic sur http://I/..., ensuite remplissage formulaire, puis Submit modifié
  - envoi d'un message / e-mail à son insu
  - exécution d'une requête non autorisée à une base de données
- Cross-site scripting = liaison inattendue entre deux sites
  - insertion de balises dans un autre site Web
- http://www.cert.org/advisories/CA-2000-02.html

Sécurité dans les données d'entrée - résolution -

- On valide l'entrée / les paramètres de l'URL lors de leur affichage
  - < > : partout
  - &, “ : à l'intérieur des attributs HTML
- Si problème, plusieurs solutions :
  - renvoi d'erreur
  - remplacement des charactères concernés avec leur contrepartie :
    - < > & "
    - plus simple : String URLEncoder.encode (String)

Validation HTML

- HTML (et XHTML) sont des standards
- Exemples d'erreurs :
  - balise non fermée
  - type de document non spécifié (1ère ligne du HTML)
    - <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//FR">
  - encodage de la page non spécifié
    - champ Content-Type de l'en-tête HTTP
    - balise méta Content-Type du document HTML
- Valider les servlets
  - http://validator.w3.org/

Sessions
- pourquoi -

- Session : ensemble d'interactions apparentées entre un client et le serveur
  - quand un client ajoute un produit dans un site d'achat en ligne, comment le serveur sait ce qu'il y a dans son chariot ?
  - gestion d'un compte bancaire, lecture e-mails, ...
- Assure le suivi des données d'un utilisateur
  - confidentialité : ça peut aussi être des données que vous voulez cacher au client !
- Connecté vs. non connecté :
  - connecté : telnet, ftp
  - non connecté : http

Sessions
- comment -

- Méthodes :
  - le serveur envoie à chaque appel des données uniques au client et lui demande de les renvoyer toutes à chaque appel ultérieur
    - récriture de l'URL
    - formulaires cachés
    - cookies
  - pour les servlets, on peut combiner avec :
    - objets session

Sessions
- récriture de l'URL -

- On génère des pages contenant des liens intégrant les paramètres antérieurs
  - exemple : transmission d'un uid dans chaque lien :
    - <A HREF="http://.../?uid=dedu">Lien de session</A>
    - tous les liens de session de la page générée doivent contenir l'uid
  - => uniquement pages dynamiques
  - le transfert utilise obligatoirement la méthode GET :
    - => URLs longs
    - => pas de confidentialité

Sessions
- formulaires cachés -

- On rajoute des champs cachés :
  - <INPUT TYPE="hidden" NAME="uid" VALUE="dedu">
- Mêmes inconvénients que récriture de l'URL
  - (pas de confidentialité : on peut voir la source de la page Web)
  - toutes les pages doivent provenir d'un formulaire

Sessions
- cookies -

- Cookie = paire (clé, valeur) associée à une adresse serveur, exemples :
  - (LANG, ro)
  - (UID, dedu)
- Le serveur les envoie dans l'en-tête quand il veut ajouter ou modifier un cookie
  - le client les stocke
- À chaque connexion, le client envoie dans l'en-tête tous les cookies associés à l'adresse URL
  - délai d'expiration des cookies

Sessions
- cookies dans une servlet -

- Servlet => côté serveur !
- Envoi :
  - Cookie c = new Cookie ("uid", "dedu");
  - c.setMaxAge (2*24*60*60); // expire en 2 jours
  - rep.addCookie (c); // ajoute à l'en-tête de la réponse
    - avant l'écriture des données !
- Lecture :
  - Cookie[] c = req.getCookies ();
  - .getName (), .getValue (), ...

Sessions
- cookies, conclusions -

- Plus intuitif que les autres méthodes
- Pas de confidentialité
- Impossible d'avoir plusieurs fenêtres avec des sessions différentes !
- On peut faire persister le cookie longtemps
  - => crainte de surveillance de la part des utilisateurs
  - les utilisateurs / navigateurs peuvent les refuser

Sessions
- HttpSession, but -

- Les méthodes précédentes échouent si :
  - session qui nécessite beaucoup de données
  - données confidentielles aux clients
- Solution :
  - on échange que le uid (par exemple)
    - fait la correspondance entre l'utilisateur et la session
    - cookie, récriture de l'URL, ...
  - on stocke sur le serveur toutes les données associées => persistance
    - objet HttpSession
- .getSession () de HttpServletRequest

Sessions
- HttpSession, derrière le rideau -

- .getSession () lit le cookie de session (e.g. JSessionId)
  - d'autres serveurs utilisent la récriture de l'URL
- S'il existe, elle retourne la session qui lui est associée
- Sinon :
  - elle crée le cookie
  - elle le transmet au client
  - elle retourne la session qui lui est associée
- => Pas besoin de gérer soi-même l'uid

Sessions
- HttpSession, méthodes (1/2) -

- long getCreationTime ()
- String getID () : un identificateur chaîne de caractères
- long getLastAccessedTime ()
- void invalidate ()
- boolean isNew ()
  - true si session créée mais pas encore envoyée au client

Sessions
- HttpSession, méthodes (2/2) -

- Données : paires (clé, valeur) à clé unique
- String[] getValueNames ()
  - toutes les clés stockées dans cette session
- Object getAttribute (String clé)
  - retourne la valeur de la clé
- void setAttribute (String clé, Object valeur)
- void removeAttribute (String clé)
- int setMaxInactiveInterval (int), int get... ()
  - temps max. mémorisation session entre deux connexions

Sessions
- HttpSession, méthodologie -

- Obtenir l'objet de type HttpSession
  - NB : avant toute écriture de sortie !
  - HttpSession getSession (boolean create)
    - si create=true, crée une session s'il n'en existe pas une
    - retourne null si aucune session n'existe et create=false
- Lire / écrire des données
- (Fermer la session ou la laisser être fermée par le moteur de servlet, e.g. après 30 minutes)
- (Fichier connu par le moteur de servlet ?)

Sessions
- HttpSession, exemple -

- Écriture, exécution A :
  - HttpSession session = req.getSession (true);
  - Integer item = new Integer (2003);
  - session.setAttribute ("annee", item);
- Lecture, exécution B :
  - HttpSession session = req.getSession (true);
  - Integer item = (Integer) session.getAttribute ("annee");
  - if (item == null)
    - ... // "annee" non encore utilisé
  - int annee = item.intValue ();

Sessions
- conclusion -

- Si peu de données
  - utiliser cookie ou récriture de l'URL
- Si beaucoup de données (peu de données aussi)
  - utiliser HttpSession
  - en combinaison avec cookie, récriture de l'URL, ...

Contexte de servlets - problème -

- Étude de cas : un site de réservation de vols aériens (e.g. http://anyway.com)
- En utilisant les sessions on peut assurer le suivi d'un client : recherche de vols, choix d'un vol, choix de l'horaire, ..., paiement d'un billet
- Problème :
  - s'il reste un billet à un vol et plusieurs clients veulent l'acheter ?
  - généralement : gérer plusieurs clients sur une même ressource

Contexte de servlets - solution -

- Solution :
  - permettre à des ressources (e.g. objets) d'être partagées par plusieurs servlets
    - les servlets communiquent en fait avec le conteneur de servlets
  - l'ensemble de ces ressources = contexte des servlets
- Autrement dit :
  - stocker des informations propres pas à un utilisateur, mais à une application Web
  - un site Web peut avoir plusieurs applications Web / contextes :
    - réservation de vols
    - achat de livres

Contexte de servlets - configuration (1/2) -

- Fait la correspondance URL <-> servlet (.class)
- Exemple :
  - http://psm-serv.pu-pm.univ-fcomte.fr/vols/rech-vol...
- Fichier server.properties (hypothétique) :
    - (préfixe URL unique pour chaque contexte / application)
  - /vols
  - /livres

Contexte de servlets - configuration (2/2) -

- Fichier servlet.properties (hypothétique) :
  - (correspondance nom court <-> nom long)
  - RechVol fr.univ-fcomte.pu-pm.psm-serv.RechVol
  - ChoixVol fr.univ-fcomte.pu-pm.psm-serv.ChoixVol
  - ...
- Fichier rules.properties (hypothétique) :
  - (correspondance préfixe URL unique <-> nom court)
  - /rech-vol RechVol
  - /choix-vol ChoixVol
  - ...

Contexte de servlets - méthodes -

- Classe de base : ServletContext (.getServletContext ())
  - void setAttribute (String clé, Object valeur)
    - si clé existe, remplace la valeur
    - peut être utilisé dans la méthode init () pour initialisation :
      - String getInitParameter (String), pour paramètres d'initialisation
  - Object getAttribute (String clé)
  - void removeAttribute (String clé)

À venir...

- Pages avec contenu mixte
  - renvoyer des images générées dynamiquement (à la volée)
- JSP (Java Server Pages)
  - insérer, ”à la PHP”, du code servlets dans des fichiers HTML
- Java Beans
  - passer des objets entre plusieurs servlets
  - instancier des classes Java sans programmation explicite en Java