Les servlets

Yves Bekkers

Plan

- Introduction : Pages WEB
- Servlet
- Application WEB
- Pages JSP
- Actions (tags)

Pages WEB ?

Pages WEB

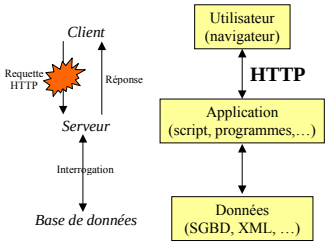
- Mise à disposition de pages sur le WEB
  - Protocole HTTP
- Cycle requête/réponse du protocole HTTP
  - Un client émet une requête vers un serveur
    - La requête possède le format HTTP
  - Le serveur est obligé de répondre au client
    - La réponse est formatée HTTP
  - Les requêtes sont indépendantes les unes des autres

Pages WEB dynamiques ?

- Protocole HTTP
- Pages WEB dynamiques pourquoi ?
  - Adapter les pages à la demande du client (outils de recherche, …)
  - Présenter des informations changeantes (météo, catalogue de produit avec leur prix, ...)
- Pages WEB dynamiques comment ?
  - Des ressources statiques - pages html, images, …,
  - Des ressources semi-statiques – SGBDR, BD XML
  - Des scripts et des programmes - CGI, PHP, JSP, javascript

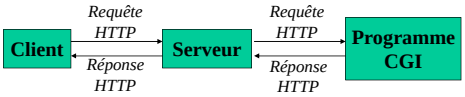
Pages dynamiques - architecture

- Au moins trois niveaux applicatifs



Extensibilité des serveurs HTTP

- Mécanisme standard d'extension : CGI Common Gateway Interface
  - Le serveur passe la requête à un programme adapté CGI
  - Le programme génère la réponse et la retourne au serveur
- Programme CGI écrit en n'importe quel langage



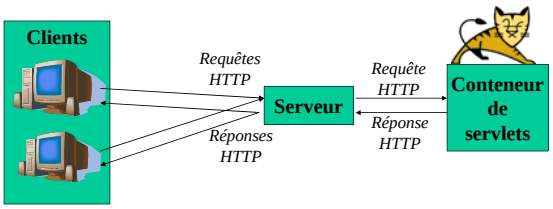
Technologies d'extension concurrentes

- CGI : Common Gateway Interface
  - programme de réponse écrit en n'importe quel langage
- ASP : Active Serveur Page
  - html, JavaScript, VBScript
  - Technologie propriétaire : *Microsoft*
- PHP :
  - Orienté SGBD, Technologie open source
- Servlet, pages JSP
  - CGIs basés sur Java, open source

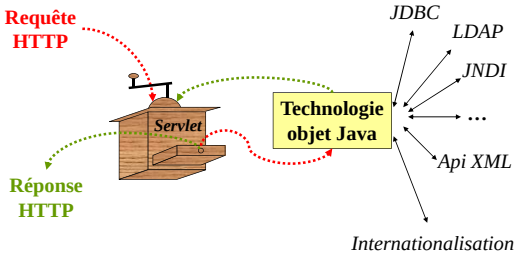
Servlets et pages JSP

- Des CGIs écrits en Java
- Une *technologie*
  - Les *servlets* et les *pages JSP*
  - L'*application*,
    - sa structure, ses paramètres de configuration
- Un *outil* : le conteneur de servlets
  - Ex : Tomcat d'Apache : open source
  - Nombreuses implémentations commerciales

Services WEB en java



Les Servlets = une ouverture vers la technologie *Objet* de Java



Non limité au protocole HTTP

- Fonctionnalités
  - *Servlets* en général
    - Champ d'application pas limité au protocole HTTP,
    - Permet d'étendre d'autres protocoles (FTP, mail, ...)
  - *Servlets HTTP*
    - Implémente le protocole requête/réponse HTTP
    - Cas d'usage le cas le plus courant
    - Mêmes rôles que les CGIs

Serveurs de servlet disponibles

- *Tomcat* d'Apache (JWSDK)
  - `http://jakarta.apache.org/`
- *Jrun* d'Allaire
  - `http://www.allaire.com/products/jrun/`
- *ServletExec* de NewAtlanta
  - `http://newatlanta.com/`
- ...

Applets versus servlets

- Deux technologies Java pour introduire le dynamisme sur le Web
  - *Applet* : dynamisme produit par le client
  - *Servlet* : dynamisme produit par le serveur

Servlet versus Javascript

- Javascript comme les Applets permet de gérer le dynamisme chez le client
- Toutes les données doivent être connues localement chez le client
- Javascript ne peut pas accéder à des données d'un serveur

CGI versus Servlet ressemblances

- Dynamisme généré par le serveur
  - Les données du serveurs peuvent être accédées
- Modèle de programme commun
  - schéma "question-réponse"
  - `get, post, head` du protocole http

CGI versus Servlet - différences

- Portabiblité
  - Un CGI est lié au type de la plateforme serveur et du langage dans lequel il a été développé
  - Les Servlets sont indépendants de la plateforme
- Passage à l'échelle
  - Les CGIs posent des problèmes d'échelle
    - Un processus par CGI ...
  - Un serveur de Servlet est conçu pour gérer des problèmes d'échelle
    - La JVM reste active et gère plusieurs processus pour une même classe

Autres avantages des servlets

- Contextes d'application et de session implémentés automatiquement par le conteneur de servlet
- Disponibilité d'outils de programmation de haut niveau : Java

Méthodes du protocole http

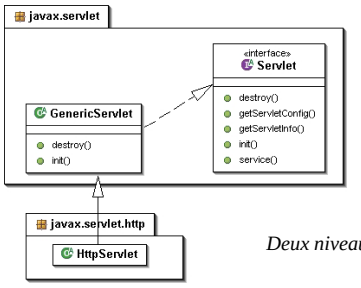
- Requête associée à une méthode
  - méthode `HEAD` pour extraire des informations relative au document
  - méthodes `GET` et `POST` utilisées pour exécuter un programme sur le serveur

`GET` versus méthode `POST`

- Méthode `GET` est utilisée par défaut pour toute requête WEB
  - les données d'entrée sont ajoutées à l 'URL de la requête (limite la taille des données en entrées)
  - Utilise le cache si la page y est déjà
  - « Esprit » *Extraction d'information*
- Méthode `POST` remplit les mêmes services mais
  - les données d'entrée sont transmises dans le corps de la requête (elles peuvent être plus grosses)
  - Il n'y a jamais d'accès au cache
  - « Esprit » *Mise à jour d'information sur le serveur*

Servlet

Deux librairies de base



Deux niveaux de classes

Librairies sur les servlets

- 9 librairies
  - javax.servlet
  - javax.servlet.http
  - javax.servlet.jsp
  - javax.servlet.jsp.jstl.core
  - javax.servlet.jsp.jstl.fmt
  - javax.servlet.jsp.jstl.sql
  - javax.servlet.jsp.jstl.tlv
  - javax.servlet.jsp.tagext

Servlet : mise en oeuvre

- 3 méthodes de mise en oeuvre
  - Implémenter l'interface `javax.servlet.Servlet`
    - Méthodes liées au cycle de vie d'une servlet
      - `init, service, destroy`
  - Etendre la classe `javax.servlet.GenericServlet`
    - Protocole non implémenté (à la charge de l'utilisateur)
  - Etendre la classe `javax.servlet.http.HttpServlet`
    - Spécialisation des `GenericServlet`
    - Protocole http implémenté (méthodes `doget`, `dopost`)

Votre première servlet

```
public class Bonjour extends HttpServlet {
  public void doGet(
    HttpServletRequest request,
    HttpServletResponse response)
    throws IOException, ServletException {
      response.setContentType("text/html");
      printWriter out = response.getWriter();
      out.println("<html><head>");
      out.println("<title>Bonjour</title>");
      out.println("</head><body>");
      out.println("<h1>Bonjour !</h1>");
      out.println("</body></html>");
    }
}
```


Classe `HttpServlet` (1)

- C'est la classe que l'on étend en général !
- Méthode `service()` déjà écrite
  - Ne pas la surcharger !
- Méthodes à implémenter
  - **`void`** `init()`
  - **`void`** `doGet(HttpServletRequest, HttpServletResponse)` *Requête GET*
  - **`void`** `doPost(HttpServletRequest, HttpServletResponse)` *Requête POST*
  - `String getServletName()`
  - `String getServletInfo()`
  - **`void`** `destroy()`

Classe `HttpServlet` (2)

- Méthodes prédéfinies utiles
  - **`void`** `log(String)`
  - **`void`** `log(String,Throwable)`
    - Générer des traces
  - `ServletConfig getServletConfig()`
    - Accès aux informations de configuration de cette servlet sur le serveur
  - `ServletContext getServletContext()`
    - Explication à venir …
- Exceptions: `ServletException`

Cycle de vie d'une servlet

1. Le conteneur de servlet créé une seule instance de la servlet
2. Il appelle sa méthode `init()` une seule fois
3. A chaque demande de service un processus est lancé sur la méthode `service()`
4. La méthode `destroy()` est appelée pour la destruction de l'instance

La vie d'une servlet

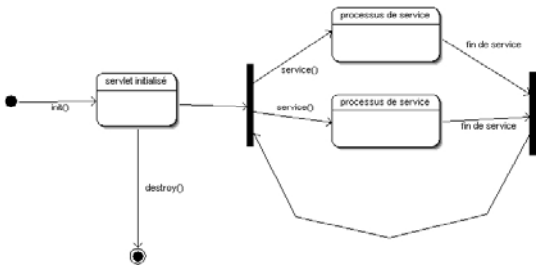
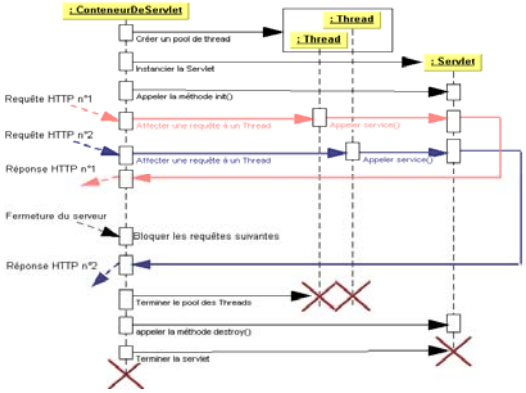


Diagramme des temps



Une servlet = plusieurs `Threads`

- Plusieurs Threads peuvent traverser la méthode `service()` en même temps
  - Les variables doivent être locales aux méthodes
  - Les attributs de classes ne peuvent être que de constantes de configuration initialisées dans la méthode `init()`

Variables d'instance et partage de code

- Les variables d'instance sont partagées entre toutes les exécutions de la méthode `service()`
- Elles ne sont pas "*Thread safe*".
- Leur usage doit être limité à des déclarations de constantes
- Les variables d'instance des objets manipulés doivent elles aussi être rendues *Thread safe*
- Attention aux modifications de fichiers !

Modification d'un compteur golbal

```
public class monThread extends HttpServlet {
  private int compteur = 0;

  public void doGet(…)…{
    …
    synchronized(this) {
      int tmpCompteur = ++compteur;
    }
    …
  }
}
```


Une servlet sûre par rapport aux Threads ?

- Utiliser seulement de variables locales et des paramètres
- Eviter les variables statiques et les variables d'intance
- Sinon utilisez des sections critiques
- N'appelez que des méthodes de classes qui respectent les règles ci-dessus

Implémenter les deux méthodes HTTP

```
public void doGet(HttpServletRequest request,
   HttpServletResponse response)
   throws IOException, ServletException {
      …
}
public void doPost(HttpServletRequest request,
   HttpServletResponse response)
   throws IOException, ServletException {
     doGet(request, response);
}
```


Arguments de `doGet()` et `doPost()`

- L'objet `HttpServletRequest` contient toutes les informations sur la requête
- L'objet `HttpServletResponse` est utilisé pour produire la réponse

La requête

- Rendue accessible par l'objet
  `javax.servlet.http.HttpServletRequest`
- Méthodes utiles
  - `String getParameter(String)`
    Compile et décode l'URL (pour GET) le corps (POST)
  - `InputStream getInputStream()`
    Pour lire de manière brute le corps de la requête POST
  - `String getRemoteHost()`
    L'adresse IP du client
  - …

Extraits Apidoc de l'objet `request`

`java.lang.String` **`getMethod`**`()`
Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.

`java.lang.StringBuffer` **`getRequestURL`**`()`
Reconstructs the URL the client used to make the request.

`java.lang.String` **`getPathInfo`**`()`
Returns any extra path information associated with the URL the client sent when it made this request.

`Cookie[]` **`getCookies`**`()`
Returns an array containing all of the Cookie objects the client sent with this request.

`java.lang.String` **`getQueryString`**`()`
Returns the query string that is contained in the request URL after the path.

`java.lang.String` **`getRemoteUser`**`()`
Returns the login of the user making this request

Utilisation de l'objet requête - Exemples

- Lire un paramètre

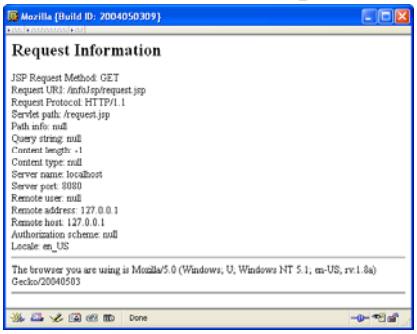
```
String s = request.getParameter("somme");
int somme = Integer.parseInt(s);
```

- Informations contextuelles

```
req.setAttribute("personneId", "blabla");
…
String maVal = req.getAttribute("personneId");
```

- Entêtes de la requête HTTP
  - `String getHeader(String)`
  - `Enumeration getHeaders(String)`

Entêtes d'une requête



La réponse

- Rendue accessible par l'objet
  `javax.servlet.http.HttpServletResponse`
- Méthodes utiles
  - **`void`** `addHeader(String name, String value)`
    - Ajoute une paire nom/valeur à l'entête de la réponse
    `void` `setStatus(`**`int`**` sc)`
    - Affecte le statut (valeur par défaut `SC_OK` =200)
  - **`void`** `sendError(int sc, String msg)`
    - Envoie une réponse erreur
  - **`void`** `sendRedirect(String url)`
    - Redirige le navigateur vers une nouvelle page
  - `ServletOutputStream getOutputStream()`
    - Le Flux de sortie pour la réponse

Utilisation de l'objet réponse Exemples

- Pour définir le type MIME et envoyer du binaire

```
ServletOutputStream out = getOutputStream();
response.setContentType("img/jpeg");
```

- Pour interdire la mise en « *cache* » méthodes GET

```
response.addDateHeader("Expires", 0);
```

- Pour signaler que la méthode POST est interdite

```
response.sendError(
   response.SC_METHOD_NOT_ALLOWED,
   "Désolé, POST non disponible !"
);
```

Entêtes d'une réponse

- Toute réponse possède des champs entêtes
  - `response.addDateHeader("Expires", 0);`
- Entête du protocole HTTP
  - Code de statut `setStatus(int sc);`
    - utilisé pour donner les raisons pour les quelles la requête a échouée
  - Cookies `addCookie(Cookie c)`
    - Pour ranger de l'information chez le client, par exemple pour suivre une session

Attention faire les choses dans l'ordre

- Configurer la sortie d'abord
  - `response.setContentType("text/html");`
  - `response.addDateHeader("Expires", 0);`
- Ecrire sur la sortie ensuite
  - `out.write(10);`
- *Reconfigurer la sortie après avoir commencé à produire les données est impossible*

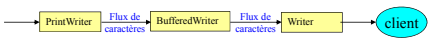
Tampon de sortie

- Par défaut tout contenu de réponse est envoyé directement au client via le flux de sortie (`OutputStream` ou `PrintWriter`)
- Utiliser un tampon permet de retarder le moment où les données sont envoyées au client et permet de retarder les actions suivantes
  - Changer les bits d'état de la réponse
  - Modifier les entêtes de la réponse
  - Détourner la réponse par un **`forward`**

Mise en place d'un tampon

```
PrintWriter sortie =
    new PrintWriter(
       new BufferedWriter(
           response.getWriter()));

sortie.print(0.123);
```



Contrôle de flux
Navigation de page en page

Utiliser d'autres ressources WEB

- Naviguer à partir d'une page statique : HTML

```
<img src="w3nation.gif" alt="…"/>
<a href="suite.html">Aller à la suite</a>
```

- Mais aussi inclure des ressources
  - Inclure une bannière, un copyright …
- Les deux méthodes à partir d'une servlet
  - Inclusion
  - Forward
- Dans tous les cas il faut obtenir un objet
  - `javax.servlet.RequestDispatcher`

L'objet `RequestDispacher`

```
RequestDispatcher dispatch;
```

- URL relatives : par le contexte de la requête
  `dispatch=request.getRequestDispatcher("urlRelative");`
  - Accepte un chemin relatif en paramètre
  - Ceux qui ne commencent pas par un « / »
- URL absolues : par le contexte global
  `dispatch=this.getServletContext().getRequestDispatcher("urlAbsolue");`
  - Accepte seulement les URLs abolues
- Attention `getRequestDispatcher()` est susceptible de retourner une référence **`null`**

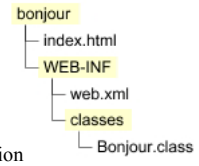
Utiliser un `RequestDispacher`

- Inclure un contenu
  `dispatch.include(request,response);`
  - Ressource statique : inclusion simple
  - Ressource dynamique :
    - Envoi d'une requête
    - Exécution puis inclusion de la réponse
    - Le ressource dynamique appelée ne peut pas modifier les entêtes
- Détourner vers un autre producteur
  `dispatch.forward(request,response);`
  - Toute la sortie de l'appelante est remplacée par celle de l'appelée … ou presque !

Notion d'application

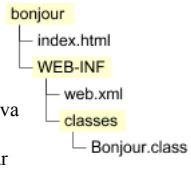
Hiérarchie d'une application

- Une racine
  - Le nom de la racine est le nom de l'application
- Une partie invisible de l'extérieur
  - Le répertoire WEB-INF, contient les informations de gestion de l'application
- Tout le reste visible de l'extérieur
  - Structure non imposée



Le répertoire WEB-INF

- Le répertoire `WEB-INF` contient
  - un document xml `web.xml` qui décrit l'installation de l'application
  - un répertoire `classes` pour le code java (classes)
  - un répertoire `lib` pour les librairies jar
  - Le reste est libre :
    - source, fichiers xml, …
- Chargement des classes
  - D'abord dans le répertoire `classes`, puis dans les librairies du répertoire `lib`



Le fichier web.xml

- Fichier de description de l'application
  - Utilisé par le conteneur de Servlet pour effectuer le déploiement de l'application
  - Document xml dont l'élément racine est `<web-app>`

```
<web-app>
   <servlet>
        <servlet-name>servlet1</servlet-name>
        <servlet-class>fr.ifsic.web.MaServlet</servlet-class>
   </servlet>
   <servlet-mapping>
        <servlet-name>servlet1</servlet-name>
        <url-pattern>*.do</url-pattern>
   </servlet-mapping>
</web-app>
```

`web.xml` - Informations de déploiement

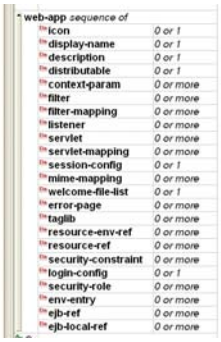
- Le document `web.xml` contient
  - Paramètres d'initialisation
  - Configuration de la session
  - Déclarations de servlets
  - Interprétation d'URL, *mapping*
  - Classes *d'écoute* liées au cycle de vie des servlets
  - Définition de filtres
  - Types MIME
  - Liste de documents d'accueil, *Welcome files*
  - Pages de récupération d'erreur
  - Librairies de tag JSP
  - Identifications JNDI

Structure du document `web.xml`

- Servlet 2.2
- Le fichier `web.xml` contient la séquence suivante d'éléments xml

Structure du document `web.xml` (2)

- Servlet 2.3
- Le fichier `web.xml` contient la séquence suivante d'éléments xml



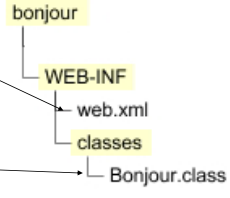
Déploiement d'une application

- Deux formats de déploiement
  - La hiérarchie complète non compressée
  - Un fichier WAR (Web Archive) qui est une compression zip de la hiérarchie
- On déploie l'un ou l'autre sur le répertoire de déploiement du serveur,
  - Pour Tomcat, dans son répertoire `webapps`
  - Le serveur introspecte périodiquement son répertoire de déploiement

Trois exemples d'application WEB

Exemple1 - l'application « bonjour »

- Une hiérarchie application
  - Racine `bonjour`
- Un fichier de configuration
  - Met en relation un nom du service (URL) et la servlet
- Une servlet
  - Classe `Bonjour` hérite de `javax.servlet.http.HttpServlet`



La servlet `Bonjour`

```
public class Bonjour extends HttpServlet {
  public void doGet(
    HttpServletRequest request,
    HttpServletResponse response)
    throws IOException, ServletException {
      response.setContentType("text/html");
      printWriter out = response.getWriter();
      out.println("<html><head>");
      out.println("<title>Bonjour</title>");
      out.println("</head><body>");
      out.println("<h1>Bonjour !</h1>");
      out.println("</body></html>");
    }
}
```


Choix de la servlet dans le document `web.xml`

```
<servlet>
  <servlet-name>salut</servlet-name>
  <servlet-class>Bonjour</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>salut</servlet-name>
  <url-pattern>hello</url-pattern>
</servlet-mapping>
```

nom interne

Classe Java

Patron d'URL

URL

- Trouver l'application

`http://bekkers:8080/bonjour/hello`

serveur application service

Chemin de contexte

- Le conteneur gère un contexte par application
  - Chaque application possède son *chemin de contexte*
  - C'est le plus grand chemin de contexte trouvé qui détermine l'application cible
- Le reste de l'URL sert à déterminer la servlet

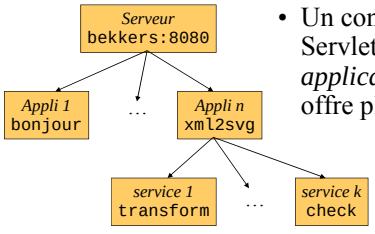
Choix de la servlet

- Deux étapes
  1. Choix du contexte
     - à l'aide des noms d'application
  2. Choix de la servlet (service)
     - à l'aide de document `web-xml` de l'application
     - Mécanisme de modèle (patron)

Utilisation d'une servlet

1. Compiler les classes et servlets de l'application
2. Déployer l'application sur le serveur
3. Lancer le serveur
4. Démarrer un navigateur et demander l'URL de la servlet
   - `http://localhost:8080/bonjour/hello`

Plusieurs applications indépendantes



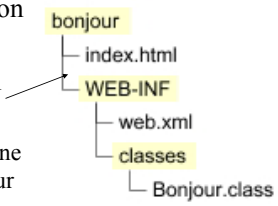
- Un conteneur de Servlets gère plusieurs *applications*, chacune offre plusieurs *services*

Exemple - choix du contexte

- Deux applications
  - **`monAppli`** et **`monAppli-demo`**
- Chemin de contexte dans une URL (en rouge)
  - **`http://www.serveur.fr/monAppli-demo/serv`**
  - Le plus long chemin de contexte est choisi

Ajouter une page d'accueil à l'application « bonjour »

- Une hiérarchie application
  - Racine `bonjour`
- Une page d'accueil html
  - À la racine du projet
  - Toutes les pages à la racine sont visibles de l'extérieur
- Le fichier de configuration
- La servlet



Passer par une page d'accueil

- La page d'accueil est un formulaire `index.html`

```
<html>
   <head>
      <title>Requête Bonjour</title>
   </head>
   <body>
      <h1>Requête Bonjour</h1>
      <form method="GET" action= "hello">
         <input type="SUBMIT">
      </form>
   </body>
</html>
```

Appel du service

Formulaires html

- Elément `<form>` envoie une requête http

```
<form action="http://www.xxx.fr/cgi-bin/reg"
      method="post">
   ... Composants du formulaire ...
</form>
```

  - L'attribut `action` est l'url du service http
  - L'attribut `post` est la méthode d'appel
    - Deux méthodes `get, post`

Composant `<input type="submit">`

- Bouton de soumission

```
<input type="submit"
  [name=" ... "][value=" ... "]>
```

  - L'attribut `name` est un paramètre envoyé au service
    - S'il est absent pas de paramètre envoyé
  - L'attribut `value` s'affiche sur le bouton
    - s'il est absent une valeur par défaut s'affiche par exemple « `Submit Query` »
    - Si l'attribut `name` est présent alors la valeur du paramètre envoyé est celle de l'attribut `value`

Déroulement de l'accès

- Premier accès au serveur par la page d'accueil
  - `http://localhost:8080/bonjour/index.html`

- Click sur le bouton « soumettre »
- Second accès au serveur par la servlet
  - `http://localhost:8080/bonjour/hello`

Comprendre l'activité de la servlet

Activités gérées par la servlet

1. Lire les données expédiées par le demandeur
2. Lire les méta-informations sur la demande et le demandeur
3. Calculer la réponse (accés aux SGBD …)
4. Formater la réponse (généralement en html)
5. Envelopper la réponse http (type de document en retour, cookies …)
6. Transmettre le document dans un format adapté
   - texte (eg. html)
   - binaire (eg. gif)
   - compressé (eg. gzip)

Activités de la servlet « Bonjour »

```
public class Bonjour extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
  throws IOException, ServletException
  {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<body>");
    out.println("<head>");
    out.println("<title>Bonjour !</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Bonjour !</h1>");
    out.println("</body>");
    out.println("</html>");
  }
}
```

5 Envelopper la réponse

4 Formater la réponse

1 Pas de données

2 Pas de métadonnées

3 Pas de calcul

6 Transmettre la réponse

Envelopper/transmettre la réponse

- Paramètre `ServletResponse` `response` de la méthode `doGet`
- L'objet `javax.servlet.ServletResponse` assiste la servlet à coder la réponse au client
  - Créé par le moteur de servlet
  - Envoyé en paramètre d'entrée
- Réponse « flux de caractères »

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
```

  - Attention, appeler `setContentType()` avant `getWriter()`
- Réponse en binaire
  - utiliser le `ServletOutputStream` retourné par `response.getOutputStream()`

Exemple 2 - l'application « *conversion* »

- Nom de l'application (nom de la racine)
  - `convertirEnEuro`
- Une page d'accueil
  - `index.hml`
- Une servlet
  - `ConvertirEnEuro`
- Un fichier de configuration
  - `web.xml`

Paramètres

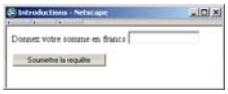
`http://www.my.com/convertirEnEuro/conversion?somme=12`

Partie serveur | application | service | params

Chemin de contexte | Partie service

Utilisation de la servlet

- Accès à la page d'accueil :
  - `http://localhost:8080/convertirEnEuro/index.html`



- Cliquer sur le bouton
- Accès à la servlet :
  - `http://localhost:8080/convertirEnEuro/conversion?somme=14`

Formulaire de saisie

- Envoie du paramètre `somme`

```
<HTML>
  <HEAD>
    <TITLE>Introductions</TITLE>
  </HEAD>
  <BODY>
    <FORM METHOD=GET ACTION="conversion">
      Donnez votre somme en francs
      <INPUT TYPE=TEXT NAME="somme"><P>
      <INPUT TYPE=SUBMIT>
    </FORM>
  </BODY>
</HTML>
```

Saisie du paramètre `somme`

Récupération du paramètre

- Le paramètres dans l'url

  `http://localhost:8080/convertirEnEuro/conversion?somme=12`

- Réception du paramètre dans la servlet
  - Objet `javax.servlet.ServletRequest`
    - Instance créé par le moteur de servlet
    - Envoyée comme paramètre d'entrée de `doGet()`
  - Méthode `getParameter()` de l'objet `request`

```
String val = request.getParameter("somme");
```


Récupération du paramètre

```
public class ConvertirEnEuro
     extends HttpServlet {
  public void doGet(HttpServletRequest req,
    HttpServletResponse response)
      throws IOException, ServletException {
    …
    int francs =
     Integer.parseInt(
        req.getParameter("somme")
     );
    …
  }
} // fin class ConvertirEnEuro
```

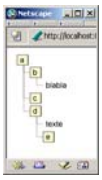

Récupération du paramètre - 2

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html>");
out.println("<head><title>Convertir en Euro</title></head>");
out.println("<body>");
out.println("<h1>Résultat</h1>");
out.println(francs + " Frs valent ");
out.println(francs / 6.55957 + " Euro");
out.println("</body></html>");
```


Troisième exemple d'application

Manipulation de documents xml

Exemple 3 – Transformation xslt



Formulaire de la page d'accueil

```
<form action="xml2svg" method="post">
   <textarea name="document"
     rows="20" cols="80">
       ... donnez ici votre document xml
   </textarea><br/>
   ...
</form>
```


Produire un document svg

- Définir le type de document de sortie

```
response.setContentType("image/svg+xml");
```

- Produire la sortie

```
StreamResult result =
   new StreamResult(response.getWriter());
transformer.transform(source, result);
```


Compiler les documents xml

- Compiler un document reçu en paramètre

```
/* créer un analyseur DOM */
DocumentBuilderFactory factory =
  DocumentBuilderFactory.newInstance();
DocumentBuilder builder =
  factory.newDocumentBuilder();

/* analyser le document passé en paramètre */
document = builder.parse(
  new InputSource(
    new StringReader(
      request.getParameter("document"))));
```

Transforme la chaîne en un flux d'octets

Lecture de la chaîne paramètre« Document »

Transformer un document

```
String xsl = context.getRealPath("/xsl/xml2svg.xslt");
// création d'un transformeur
TransformerFactory transFactory =
   TransformerFactory.newInstance();
Transformer transformer = transFactory.newTransformer(
    new StreamSource(xsl));
transformer.setParameter("box.color",
    request.getParameter("box.color"));
transformer.setParameter("text.color",
    request.getParameter("text.color"));
// fabrication de la source
DOMSource source = new DOMSource(document);
// transformation
StreamResult result = new
     StreamResult(response.getWriter());
transformer.transform(source, result);
```

Passage de parametre *à la feuille de style xsl*

entrée

sortie

Configurer l'application : document `web.xml`

Document `web.xml`

- **Interpréter les URLs**
- Divers

Interprétation par défaut des URLs

- Servlets
  - Mécanisme d'alias
- Fichiers auxiliaires, pages JSP
  `http://HOST:PORT/monapplication/fichier`

Association URL ↔ service

`http://www.my.com/bonjour``/hello?i=5&j=7`

Contexte service params

Le « mapping » service/Servlet est défini dans le document web.xml

Correspondances d'URL

- On test l'URL en regard de *patrons* donnés dans le fichier `web.xml`
- On choisit dans l'ordre
  - Modèle d'url correspondant exactement
  - Le plus long chemin **`/…/*`**
  - Dernier segment avec une extension **`*.do`**
  - Par défaut **`/`**
- Ambiguités
  - La première dans l'ordre ci-dessus qui réussit gagne

Correspondance exacte

- Description :

```
<servlet-mapping>
      <servlet-name>salut</servlet-name>
      <url-pattern>/hello<url-pattern>
</servlet-mapping>
```

- URL

  `http://localhost:8080/appli``/hello`

  ~~**`http://localhost:8080/appli`**`/hello/bekkers`~~

Chemin le plus long

- Chaîne commençant par « `/` » et finissant par « `/*` »

```
<servlet-mapping>
      <servlet-name>salut</servlet-name>
      <url-pattern>/hello/bekkers/*</url-pattern>
</servlet-mapping>
```

- Exemple

  `http://localhost:8080/appli``/hello/bekkers`

  `http://localhost:8080/appli``/hello/bekkers/yves`

  ~~**`http://localhost:8080/appli`**`/ss/hello/bekkers/yves`~~

Correspondance avec une extension

- Le derbier segment contient un « *.*XXX*»
  - description :

```
<servlet-mapping>
      <servlet-name>salut</servlet-name>
      <url-pattern>*.hello</url-pattern>
</servlet-mapping>
```

  - URL

http://localhost:8080/appli/bonjour.hello
http://localhost:8080/appli/salut.hello
~~**http://localhost:8080/appli/**hello~~

Correspondance par défaut

- Chaîne « / » seule
  - description :

```
<servlet-mapping>
      <servlet-name>salut</servlet-name>
      <url-pattern>/</url-pattern>
</servlet-mapping>
```

  - URL

http://localhost:8080/appli/*n-importe-quoi*

Exemples

- **Associations**
  - Servlet1 ⇔ /foo/bar/*
  - Servlet2 ⇔ /foo/*
  - Servlet3 ⇔ /catalogue
  - Servlet4 ⇔ *.bop
- **Exemples d'URL**
  - /foo/bar/index.html → servlet1
  - /foo/bar/index.bop → servlet1
  - /foo → servlet2
  - /foo/index.html → servlet2
  - /catalogue/catalogue/index.html → 404 Error page
  - /catalogue/racecar.bop → Servlet4
  - /index.bop → servlet4

Document `web.xml`

- Interpréter les URLs
- **Divers**

Élément `<welcome-file>`

http://www.my.com/bonjour/hello

Contexte Service

- Une requète dont la partie contexte est correcte mais dont la partie service est non reconnue peut être détournée vers une page de d'accueil unique

```
<welcome-file-list>
   <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```


Récupération d'erreur

```
<error-page>
  <error-code>404</error-code>
  <location>/common/404.jsp</location>
</error-page>

<error-page>
  <exception-type>
    javax.servlet.ServletException
  </exception-type>
  <location>/common/error.jsp</location>
</error-page>
```

Contraintes de sécurité

```
<security-constraint>
   <web-resource-collection>
      <web-resource-name>
         Entire Application
      </web-resource-name>
      <url-pattern>/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <role-name>manager</role-name>
   </auth-constraint>
</security-constraint>
```


Authentification du client

```
<login-config>
   <auth-method>BASIC</auth-method>
   <realm-name>
      Tomcat Manager Application
   </realm-name>
</login-config>
```


Paramétrage d'une application

- **Problème** : rendre des informations accessibles aux applications WEB sans les "cabler" dans le code JSP ou Java
  - Exemple : Paramètres d'accès à une base de données :
    - l'URL
    - le nom de classe Driver
    - le nom d'utilisateur
    - le mot de passe
- **Une méthode** : Utiliser le fichier web.xml

Paramètrage par le fichier web.xml

- Entrées d'environnement (tout objet Java)

```
<env-entry>
   <env-entry-name>sgbd.nomenclature.driver
                                   </env-entry-name>
   <env-entry-value>sun.jdbc.odbc.JdbcOdbcDriver
                                   </env-entry-value>
   <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

- Paramètres de contexte (String uniquement)

```
<context-param>
   <param-name>sgbd.nomenclature.driver</param-name>
   <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>
```


Récupération des paramètres

- Entrées d'environnement (tout objet Java)

```
// Obtenir le contexte d'environnement
Context initCtx = new InitialContext();
Context envCtx = (Context)
  initCtx.lookup("java:comp/env");
// rechercher l'information
String driver = (java.lang.String)
  envCtx.lookup("sgbd.nomenclature.driver");
```

- Paramètres de contexte (String uniquement)

```
String driver = application.
   getInitParameter("sgbd.nomenclature.driver");
```

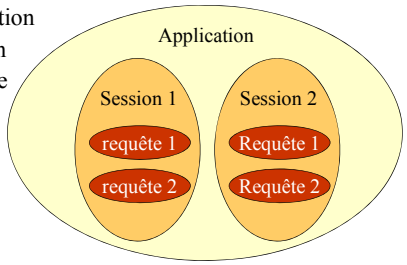

Notions de contexte

Problème du suivi de session

- Protocoles connectés – Session explicite
  - FTP, Telnet
  - Notion de session
    - Connection/session/déconnection
- Protocole déconnecté – Session non explicite
  - HTTP
  - Chaque requête est une transaction isolée
  - Nécessité de déterminer si un ensemble de requêtes vient du même client
    - Introduire un suivi de session

Conserver l'information

- Les servlets offrent trois portées de contextes par application
  1. L'application
  2. La session
  3. La requête



Trois contextes

1. Application
   `javax.servlet.ServletContext`
2. Session
   `javax.servlet.http.HttpSession`
3. Requête
   `javax.servlet.http.HttpServletRequest`

Contexte global à l'application

- Objet `javax.servlet.ServletContext`
  - Visible par toutes les sessions et toutes les pages d'une même application
- Ce contexte peut servir à
  - dialoguer avec le conteneur de servlet
    - Obtenir des information sur un fichier présent sur le serveur,
    - Gérer les requêtes d'une même application,
    - Écrire dans un fichier de log.
  - communiquer de l'information entre servlets d'une même application

Utiliser le contexte de l'application

- Obtenir le contexte de l'application (2 étapes)

```
ServletConfig config =
  this.getServletConfig();
ServletContext appli =
  config.getServletContext();
```

- Mémoriser un objet

```
Objetc myObject = ...
appli.setAttribute("racine", myObject );
```

- Extraire un objet mémorisé

```
myObject = appli.getAttribute("racine");
```


Utiliser le contexte de l'application bis

- *Problème* :
  - Accéder à un fichier local à l'application
- *Solution* :
  - utiliser la méthode `getRealPath()`
  - chemin absolu de la racine de l'application

```
String racine = context.getRealPath("/");
```

  - chemin absolu vers un fichier quelconque de l'application

```
String fichier =
  context.getRealPath("/xml/index.xml");
```

Notez la présence du "/" en début de nom

Suivi de session

- Session
  - Ensemble de requêtes apparentées entre un client unique et un serveur WEB
  - Permet au client de parcourir un site web tout en remplissant son cadie
- Trois méthodes de suivi de session
  1. Suivi automatique par le serveur de servlet à partir des informations envoyées par le client
     - Un chronomètre permet de borner dans le temps la durée de la session
  2. Réécriture d'URL avec des paramètres cachés
  3. Utilisation des cookies

`HttpSession`

- Rôle
  - suivi de session automatique par le serveur de servlet
- Objet `javax.servlet.http.HttpSession`
- Conserver de l'information entre deux requêtes liées d'un même utilisateur
  - Une session dure un temps borné et traverse une ou plusieurs connections d'un et un seul utilisateur.
  - L'utilisateur peut ainsi visité un même site plusieurs fois en étant reconnu.
  - Le serveur dispose de diverses méthodes pour mainenir la notion de session : cookies ou le réécriture d'URLs.

Objet `HttpSession`

- Obtenir la session
  - `HttpSession session = request.getSession(true);`
- Gérer des attributs de session

```
void setAttribute(String name,
  Object value)
void removeValue(String name)
Object getAttribute(String name)
java.util.Enumeration getAttributeNames()
```

- Divers

```
Long getCreationTime()
String getId()
```


Problèmes pour obtenir la session

- L'identification de la session est faite de façon transparente par le conteneur de servlet, qui utilise des cookies ou la réécriture d'URL
- Etant donnée que les cookies sont stockés dans le en-têtes HTTP, et que celles-ci doivent être les premières informations envoyées, la méthode *getSession()* doit être appelée avant tout envoi de données au navigateur (la méthode doit être invoquée avant toute écriture sur le flot de sortie de la servlet)

Gérer la durée d'une session

- `int setMaxInactiveInterval(int interval)`
  - Définit l'intervalle de temps maximum entre deux requêtes avant que la session n'expire
- `int getMaxInactiveInterval(int interval)`
  - Retourne l'intervalle de temps maximum entre deux requêtes avant que la session n'expire

Les filtres

Filtrer les requètes et les réponses

- Un filtre est un programme qui peut transformer les entêtes et les contenus des requètes et des réponses
- Exemples d'Applications des filtres
  - authentication,
  - logging,
  - image conversion, data
  - compression,
  - encryption,
  - tokenizing streams,
  - XML transformations,
  - and so on.

Interface à implémenter

- Les interface à implémenter sont
  - `Filter`, `FilterChain`, et `FilterConfig`
  - Dans la paquetage `javax.servlet`.
- Exemple

```
public final class monFiltre
    implements Filter {
  private FilterConfig filterConfig = null;
  public void init(FilterConfig filterConfig)
    throws ServletException {
      this.filterConfig = filterConfig;
  }
  public void doFilter(ServletRequest request,
    ServletResponse response, FilterChain chain)
    throws IOException, ServletException {
      …
  }
}
```


Références

- Tutorial de sun (1500 pages)
  - (html) http://java.sun.com/j2ee/1.4/docs/tutorial-update2/doc/index.html
  - (pdf) http://java.sun.com/j2ee/1.4/docs/tutorial-update2/doc/J2EETutorial.pdf
- Tutorial Sun sur les servlets et jsp (600 pages gratuites en pdf)
  - http://csajsp-chapters.corewebprogramming.com/Core-Servlets-and-JSP.pdf
- Un tutorial sympa
  - http://j2ee.masslight.com/