Pages dynamiques sur le Web

Michel Daydé
ENSEEIHT-IRIT
2 rue Camichel
31071 TOULOUSE CEDEX FRANCE
dayde@enseeiht.fr

March 29, 2004

Contents

1 Introduction

Le Web peut servir de support à l'exécution d'application réparties en plus de sa fonction d'accès à l'information.

Intérêt : Interface familière + Disponibilité d'outils de base :

- Espace universel de désignation (URL/URI)
- Protocole de transfert de l'information (HTTP)
- Gestion d'information sous format standard (HTML)

Web = système d'exploitation primitif pour application réparties ?

Problèmes :

- Où et comment sont exécutés les programmes ?
  - Sur le site serveur → scripts CGI ou PHP, servlets
  - Sur le site client → scripts dans extension du navigateur (JavaScript, plugin) ou applets

Comment assurer la sécurité ?

- Problème majeur pas complètement résolu
  - Protection des sites
  - Encryptage de l'information
  - Restrictions sur les conditions d'exécution

1.1 Internet et le Web

1.1.1 Exemple requête sur le Web

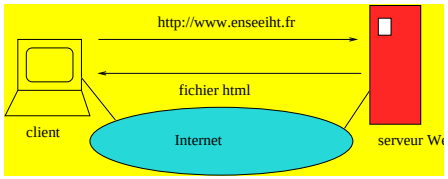


Figure 1: Fonctionnement d'une requête sur le Web.

- Vision de l'utilisateur

  - L'utilisateur clique sur un lien
  - La page Web s'affiche

- Station client
  - Le navigateur envoie une requête au serveur attaché à l'URI (Uniform Resource Identifier) associé au lien
  - Si tout est OK, le navigateur reçoit un fichier HTML qu'il affiche à l'écran.

- Du point de vue réseau, il faut :
  - Trouver le bon serveur
  - Transporter la requête du client vers le serveur
  - Transporter le fichier HTML du serveur vers le client

Localisation du serveur

- Via la service de noms de l'Internet (DNS) associant nom ↔ "adresse IP".
- Toute machine connectée à Internet a une adresse IP
- Comment trouver l'annuaire : on connaît l'adresse IP d'un point d'entrée

Envoyer la requête au serveur :

- Requête transmise sous forme de message (suite de bits)
- Découpée en "paquets" de taille fixe
- Chaque paquet est envoyé sur le réseau

- Internet = interconnexion de réseaux reliés entre eux par des routeurs.
- Chaque paquet contient : numéro + adresse de destination. En arrivant sur un réseau si le destinataire n'en fait pas partie, le paquet est transmis à un routeur qui possède des informations permettant d'acheminer le paquet vers le destinataire.

- Fonctionnement du serveur :

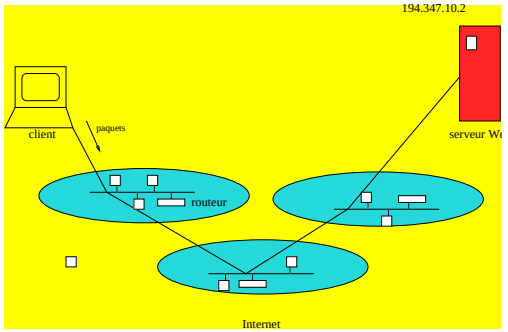


Figure 2: Routage des données

  - Reconstruit le message initial en mettant les paquets dans l'ordre des numéros.
  - Interprète la requête
  - Envoie le fichier au client (sous forme de message découpé en paquets)
  - Cas de pb de transmission (paquet perdu ou mal transmis) : le destinataire peut le détecter et fait une demande de renvoi.

- Transmission de l'information :
  - Paquet = suite de bits
  - Transmission sur un support physique (cable, fibre optique, ...) sur lequel on envoie des signaux.

1.1.2 Services usuels de l'Internet

Services principaux :

- Courrier électronique → protpocole SMTP (Simple Mail Transfer Protocol) : grâce au format MIME inclut tout type de documents (Multipurpose Internet Mail Extensions)
- Forums de discussion (news) → protocole NNTP (Network News Transfer Protocol)

- Transferts de fichiers (ftp) → protocole FTP (File Transfer Protocol)
- Accès à une machine distante (telnet) → protocole TELNET de terminal virtuel
- X-Window : service de fenêtres pour client distant
- Accès au Web → protocole HTTP, formats HTML, XML, ...
- Services divers basés sur le Web
- Extension diverses pour la sécurité

Basés sur les protocoles de transport TCP et UDP ainsi que le service de noms DSN

1.1.3 World Wide Web : principes et composants

Historique :

- Idée de base : ensemble de documents répartis reliés entre eux par des liens hypertexte. Objectif initial (Tim Berners-Lee, CERN, 89-90) : créer un outil pour la partage dela connaissance pour la communauté des physiciens
- Fin 93 : 250 serveurs, 1% du trafic Internet
- A partir de 94 :
  - Premiers navigateurs : Mosaic, Netscape
  - Premiers moteurs de recherche : AltaVista, Yahoo!
  - Création du Worl Wide Web Consortium
  - 10,000 serveurs fin 94

Eléments de base :

- Espace de noms global pour la désignation des ressources (URL puis URI)
- Protocole client-serveur pour le transfert d'informations : HTTP
- Langage de balisage pour la description de documents : HTML

Extensions

- Utilisation de langages de script : applets, servlets, ...
- Utilisation de types de données multiples

1.1.4 Désignation sur le Web

- Format d'un URI (Uniform Resource Identifier)
  <protocole>:<chemin d'accès>
- Format du chemin d'accès dépend du protocole :
- Protocole HTTP :

```
http://<identite du serveur>[:<numero de porte>]
[<chemin d'acces local>][? requete][#etiquette]
```

- **ftp** : idem sans requête ni étiquette
- Par défaut : HTTP pour le protocole, 80 pour numéro de porte, index.html pour le nom de fichier dans un répertoire, ...
- URI : voies d'accès à toutes les ressources (documents, serveurs, programmes, ...) et constituants des liens hypertexte

1.2 Protocole HTTP

- Protocole standard du Web
  - Protocole client-serveur construit au-dessu de TCP
  - Utilisation principale : entre navigateur et serveur Web mais peut être utilisé de façon autonome par toute application

Quelques commandes :

- GET <URI> : demande au serveur indiqué dans l'URI la page désignée (option : envoi uniquement si elle a été changée depuis une date spécifiée)
- HEAD <URI> : demande au serveur d'envoyer l'en-tête de la page
- PUT <URI><page> : envoie au serveur une page pour la rendre disponible à l'adresse spécifiée (remplace le contenu si existante)
- DELETE <URI> : efface la page à l'adresse spécifiée
- Vérification des droits avant exécution de commande
- Réponse comporte un code et éventuellement un résultat
- Commandes d'envoi de données utilisent une convention standard (MIME) pour les données non textuelles.

1.3 Exécution de programmes sur un serveur Web

- Intérêt
  - Exécution de programmes interactifs : formulaires d'inscriptions, enquêtes, ...
  - Requêtes (moteur de recherche, ...)
- Connexion avec le monde extérieur
  - Systèmes de bases de données
  - Serveur d'applications ou de calcul

1.4 Mécanismes

- Scripts CGI (Common Gateway Interface)
  - Le plus ancien
  - Marche avec de multiples langages, interface commune standard
- Scripts PHP
  - Interfaces avec SGBD
  - Très léger (pas de création de processus)
  - Interpréteur intégré au serveur
- Servlets
  - Programmes Java activés sur le serveur, interface spécifique Java
- Applets
  - Programmes Java activés par le navigateur sur le client
  - Applications légères
  - Source en Java, compilé en *bytecode* chargé et interprété par le navigateur

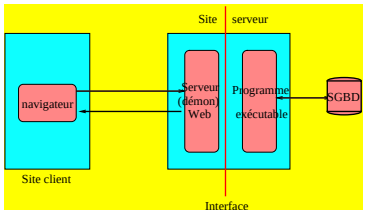


Figure 3: Exemple de système de bases de données sur le Web

2 Langage HTML

2.1 Introduction

- Langage de balisage :
  - Comporte des marques (balises) insérées dans le texte donnant des indications de formatage.
  - Inspiré de SGML utilisé dans l'édition de documents
  - Intérêt du balisage : séparer le contenu de la présentation ou de l'interprétation (interprétation différente selon les capacités d'affichage (WAP ?))
- HTML vient du milieu universitaire
- Intégration du multimédia annexe importante de HTML
- Permet de suggérer la structure d'un document
- N'est pas un langage de mise en page (e.g. on ne peut prédire police et taille absolue utilisées pour afficher un texte)
- Documents HTML produits à la main ou avec des éditeurs
- HTML permet aussi l'interaction entre client et serveur (INPUT, ...)

Exemple de page HTML

```
<html>
<head>
<title> Exemple simple de document HTML </title>
```

```
</head>
<body>
<h2> Ceci est un exemple <i> simpliste </i> de document HTML </h2>
<hr>
<!-- Bien sur avec un commentaire -->
<p>

L'auteur s'excuse <br>
Il s'agit de <a href="http://www.enseeiht.fr/~dayde">Michel Dayd&eacute;</a>
</body>
</html>
```

2.2 Squelette HTML

- En-tête et corps : head, body
- Délimités par marqueurs d'ouverture et de fermeture
- En-tête : informations sur le document → titre
- Corps : contenu du document à afficher dans le navigateur

2.3 Hyperliens

- Hypertexte cœur d'un document HTML
- Permet avec un clic sur un mot ou une phrase de récupérer ou d'afficher un autre document
- Deux éléments : adresse du document (URL) et jeter une ancre

Les ancres

```
<a href="http://...../mypage.html">Voici mon document</a>
```

- href : définit la source d'un hyperlien
- Texte ou images entre <**a**> **...** </**a**> zone activée dans le navigateur à sélectionner
- Aspect en général différent

Exemples d'hyperliens


```
<html>
<head>
<title> Exemples simples de liens</title>
</head>
<body>
<hr>
<ul>
<li> Voici un exemple <a href="http://localhost/COURS/exemple-simple.html">
de lien hypertexte </a> vers une page HTML.
<li> Voici un exemple avec un <a href="ftp://localhost/COURS/exemples-liens.html">
transfert de fichier </a> par ftp
<li> Et enfin un lien avec <a href="mailto:dayde@enseeiht.fr">
envoi de mail </a> &agrave; Michel Dayd&eacute;
</ul>
<hr>
Auteur : <a href="http://www.enseeiht.fr/~dayde">Michel Dayd&eacute;</a>
</body>
</html>
```

2.4 Eléments importants d'un document HTML

- Peu d'éléments obligatoires à part :

  `<html>, <head>, <body>, <title>`

- Commentaires

  ```
  <!-- .......
       ....... -->
  ```

  Attention en chargeant le texte source des documents via la navigateur, on voit les commentaires !!!

2.4.1 Texte

- Marqueurs spéciaux pour la structure du texte (listes, titres, tableaux, font, ...)
- Marqueurs pour indiquer au navigateur comment formatter et afficher le texte

2.4.2 Listes

Listes non ordonnées : simple indentation avec un symbole (puce)

```
<ul>
   <li> ....
   <li> ....
</ul>
```

Listes ordonnées : indentation avec un numéro d'ordre

```
<ol>
   <li> ....
   <li> ....
</ol>
```

Listes de définition : chaque item = terme + définition

```
<dl>
   <dt> terme </dt>
      <dd> définition .....
                      </dd>
   ....
</dl>
```

2.4.3 Tableaux

Créés ligne par ligne :

```
<table border>
<caption align=bottom> Ceci est un tableau </caption>
<tr>
   <td colspan=2 rowspan=2></td>
   <th colspan=2 align=center>Pr&eacute;f&eacute;rences</th>
</tr>
<tr>
   <th>Shopping</th>          <th>M&eacute;nage</th>
</tr>
<tr>
   <th rowspan=2> Sexe </th>  <th> Masculin </th>
   <td> 50% </td> <td> 50% </td>
</tr>
<tr>
```



```
   <th> F&eacute;minin </th>  <td> 50% </td> <td> 50% </td>
</tr>
</table>
```

2.5 Contenu multimédia

- Images et autres éléments multimédia ne font pas partie du document HTML
- Images, séquences audio, vidéo, et autres sont séparées du documen t
- Références aux éléments multimédia via des marqueurs spéciaux utilisés par la navigateur

2.5.1 Inclusion d'images

- Eléments multimédia pouvant être inclus grâce à des ancres pour un téléchargement séparé et affichage.
- HTML possède des spécifications pour l'affichage d'images en ligne contrairement aux autres éléments multimédia
- Navigateurs possèdent des décodeurs (consensus autour de GIF et JPEG).
- Marqueur disponible :

  ```
  <img src="mon_images.jpg" align="top">
  ```

- Image traitée comme caractère spécial → bas de l'image a ligné avec le bas de la liogne courante de texte (modifiable avec **align** et optio ns **top, bottom, middle**).
- Incorporables dans des hyperliens

Exemples d'insertion d'images

```
<html>
<head>
<title> Exemple simple de document HTML avec image en fond</title>
</head>
<body background="FondPoster.jpg">
<blink> <h2> <font color=red> Ceci est un exemple de document HTML
```

```
avec une image en fond</font> </h2>
</blink>
<hr>
L'auteur s'excuse <br>. Il s'agit de
<a href="http://www.enseeiht.fr/~dayde">Michel Dayd&eacute;</a>
</body>
</html>
```

2.5.2 Images réactives

- Images dans une ancre comportant un attribut spécial : elles peuvent contenir plusieurs hyperlines
- Ajouter ismap au marqueur <img>
- En cliquant sur une zone de l'image, le navigateur renvoit les coordonnées relatives x, y de la souris au serveur désigné dans l'ancre → actions spécifiques
- Les attributs $usemap, map, area$ autorisent un comportement similaire sans faire appel à un serveur en insérant toutes les infos dont la navigateur a besoin dans le même document que l'image.

2.6 Frames (cadre)

- Division de la fenêtre du navigateur en plusieurs zones d'affichage - cadres- chacune contenant un document différent
- Chaque cadre peut recevoir tout type de contenu valide
- Définition dans un document spécial ou le marqueur body est remplacé par un ou plusieurs éléments frameset indiquant le découpage de la fenêtre
- Attributs rows, cols définissent la taille et le nombre de lignes (resp. colonnes) en pixels ou en % de taille fenêtre

Exemple de frames

```
<html>
<head>
<title> Exemple simple de frames </title>
```



```
</head>
<!--  Decoupage de la page en 2 frames -->
<frameset rows="60%,*" cols="40%,*">
        <frame src="exemple-fond.html" scrolling="no" noresize>
        <frame name="contents" src="exemple-images.html">
        <frame name="contents" src="exemples-liens.html" noresize>
        <frame name="results" src="exemple-table.html">
<noframes>
      Your browser does not support frames.
      <a href="exemple-simple.html">Please visit the frameless page.</a>
</noframes>
</frameset>
</html>
```

2.7 Formulaires

- Sections de formulaire crées avec les marqueurs

```
<form>
.....
</form>
```

- Insertion de champs de saisie, cases à cocher, boutons radio, ...
- Utilisateur remplit le formulaire et clique sur le bouton spécial d'envoi
- Programme spécifique côté serveur traite les données envoyées → script CGI (ou *applets* côté client)

2.8 Exemple simple de formulaire

```
<html>
<head> <title> Exemple simple de formulaire </title> </head>
<body>
<!-- On transfere une zone de texte, un fichier, et des
    valeurs de bouton -->
<form action="http://lisa/cgi-bin/formulaire-simple.pl"
             enctype="multipart/form-data" method="post">
      Entrer votre nom : <INPUT type="text" name ="nom" size="40"> <br>
      Nom du fichier <br> <input type="file" name="toto" value="" size="60"><br>
      <INPUT TYPE="checkbox" NAME="data" VALUE=1 CHECKED > Texte <br>
```

```
      <INPUT TYPE="checkbox" NAME="data" VALUE=2> Image <br>
      <INPUT TYPE="checkbox" NAME="data" VALUE=3> Autre <p>
      Quelques commentaires ? <br>
      <TEXTAREA ROWS=5 COLS=40 NAME="commentaires">
      </TEXTAREA><p>
<input type="submit" value="Envoyer la forme">
<INPUT TYPE="reset" VALUE="Annuler">
</body>
</html>
```

2.9 Feuilles de style

- CSS : Cascading Style Sheets
- Contrôle de l'apparence des pages Web
  - Style + taille des polices de caractères
  - Couleur du texte et du fond
  - Alignement
  - ...

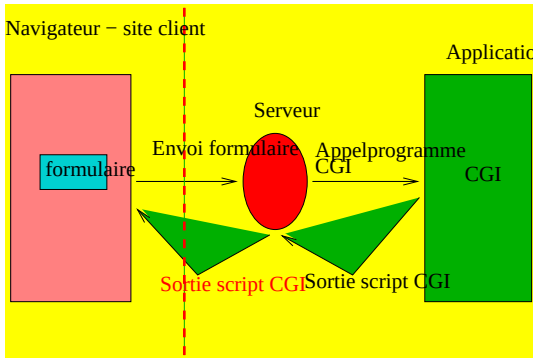
underlineExemple de CSS ([5])

```
<html>
<head>
<title> Exemple simple de CSS </title>
<!-- On dissimule les CCS dans des commentaires pour eviter
des pbs avec les anciens navigateurs. -->
<style type="text/CSS">
   <!--
   H1 {color: red}
   -->
</style>
</head>
<body>
<h1> Je dois apparaitre en rouge </h1>
si toutefois votre navigateur comprend
<h1> les CSS !</h1>
</body>
</html>
```


3 Principes de la programmation CGI

Interface commune pour l'exécution de programmes sur le site d'un serveur Web

- Utilisables avec de multiples langages : scripts (Perl, Tcl, Shell, ...) langages compilés (C, C++, ...)
- Localisation des programmes dans des répertoires spécifiques (/cgi-bin et /cgi-src)



3.1 Exécution d'un script CGI

- Conventions communes pour le passage des paramètres via HTTP (avec commandes HTTP GET, HTTP POST pour envoi des paramètres, retour des résultats dans un document HTML ou autre, sous forme d'une URI pointant vers le résultat)

3.2 Script Perl pour Formulaire simple

```
#!/usr/bin/perl
use CGI qw(:standard);
print "Content-type: text/html\n\n";
print "<html>\n";
print "<head>\n";
```

```
print "<title> R&eacute;sultat retourne par le script CGI </title>\n";
print "</head>\n";
print "<body>\n";
print "Starting Formulaire-simple.pl <br>\n" ;
# Impression de toutes les variables d'environnement
foreach $key (keys %ENV) {
        print "$key --> $ENV{$key}<br>";
}
print "<hr> \n";

# Affichage des donnees disponibles sur le client

$machine_cliente = $ENV{'REMOTE_ADDR'};
$nom_client      = $ENV{'REMOTE_HOST'};
print "Votre port d'attache est ",$machine_cliente, "\n";
print "de nom ",$nom_client,"\n";

print "<hr> \n";

# Recuperation des parametres

$taille_donnees_formulaire = $ENV{'CONTENT_LENGTH'};
print "Taille donnees envoyees par POST :",$taille_donnees_formulaire,"<br>\n\n" ;

$nom = param('nom') ;
print "Nom =", $nom," <br>\n";
$myfile = param('toto') ;
print "Nom fichier donnees =", $myfile," <br>\n\n";
$filesize = ( stat($myfile)) [7] ;
print "<p> Taille du fichier (",$myfile,") =", $filesize," </p> <br>\n" ;
$bouton = param('data') ;
print "Type de donnees =",$bouton,"<br>\n\n";
$commentaires = param('commentaires') ;
print "Commentaires :", $commentaires, "<br>\n\n";

print "<hr> \n";

# Transfert du fichier /tmp/toto

$localfile = "/tmp/toto" ;
```



```
print "fichier recepteur :", $localfile, "\n\n" ;

open(SFILE,">$localfile") || die "Failed to save submission data\n" ;

print SFILE "coucou \n" ;
while (read $myfile, $buf, 16384) {
    print "Je suis dans la boucle de lecture du fichier \n";
    print $buf;
    print SFILE $buf;
}
close (SFILE) ;
close ($fichier) ;

print "<hr> \n";

# Copie de la zone commentaires dans /tmp/comments

print "fichier recevant les commentaires /tmp/comments \n";
open(FSCRIPT,">/tmp/comments") || die "Failed to save comments\n" ;

print FSCRIPT $commentaires;
close (FSCRIPT) ;

print "</body> \n";
print "</html>";
```

3.3 Protocoles de transfert de données

Dexu protocoles différents :

- GET : a pour effet d'inclure dans l'URL les données du formulaire :

  ```
  GET /cgi-bin/formulaire-simple.pl?nom=Dayde....
  ```

- POST : données acheminées sous forme d'un flux d'entrée

Exemple méthode GET

```
<HTML>
<HEAD><TITLE> Test formulaire simple avec GET</TITLE></HEAD>
<BODY>
```

```
<H1> Formulaire simple </H1>
<HR>
<FORM ACTION="/cgi-bin/COURS/simple-get.pl" METHOD="GET">
Commande : <INPUT TYPE="text" NAME="commande" SIZE=40>
<P>
<INPUT TYPE="submit" VALUE="Envoi !">
<INPUT TYPE="reset" VALUE="Annulation">
</FORM>
<HR>
</BODY>
</HTML>

#!/usr/bin/perl

print "Content-type: text/plain\n\n";
print "Starting simple.pl \n" ;

$requete = $ENV{'QUERY_STRING'} ;
print "Requete = ",$requete, " \n\n";
($nom_champ, $commande) = split (/=/, $requete) ;
print "Commande = ",$commande, " \n\n";
```

Exemple méthode POST

```
<HTML>
<HEAD><TITLE> Test formulaire simple avec POST</TITLE></HEAD>

<BODY>
<H1> Formulaire simple </H1>
<HR>
<FORM ACTION="http://lisa/cgi-bin/COURS/simple-post.pl" METHOD="POST">
Commande : <INPUT TYPE="text" NAME="commande" SIZE=40>
<P>
<INPUT TYPE="submit" VALUE="Envoi !">
<INPUT TYPE="reset" VALUE="Annulation">
</FORM>
<HR>
</BODY>
</HTML>

#!/usr/bin/perl
```



```
use CGI qw(:standard);

print "Content-type: text/plain\n\n";

print "Starting simple-post.pl \n" ;

$taille_donnees_formulaire = $ENV{'CONTENT_LENGTH'};
print "Taille donnees envoyees par POST :",$taille_donnees_formulaire,"\n\n" ;
read( STDIN, $requete, $taille_donnees_formulaire ) ;
print "Requete = ",$requete, " \n\n";
```

3.4 Problèmes inhérent à CGI

Sécurité !!!

- Principal problème
  - CGI permet à un client d'exécuter un programme quelconque sur un serveur en lui passant de paramètres quelconques
  - Dangers : pénétration dans le système du serveur (via exécution de scripts UNIX) ou extraction d'informations confidentielles
- Quelques solution pour améliorer la sécurité :
  - Protéger les répertoires /cgi-bin et /cgi-src
  - Eviter les commandes exécutant leurs paramètres comme un programme → system, eval,...
  - Filtrer les paramètres des programmes (attention aux paramètres provoquant une exception dans le programme)
  - Processus exécutant le programme CGI doit avoir des droits limités

3.5 Directives SSI

- SSI : *Server Side Includes*
- Permet d'inclure dans un document HTML des informations simples (heure, ...)
- De lancer des applications
- Le contenu de variables d'environnement
- ...

3.5.1 Syntaxe des directives

```
<!--#directive paramètre="argument"-->
```

- Insertion de variables d'environnement

  ```
  <!--#echo var="nom_de_variable"-->
  ```

- Insertion de fichiers

  ```
  <!--#include file="nom_de_fichier.html"-->
  ```

- Taille d'un fichier, date dernière modif, ...

  ```
  <!--#size file="nom_de_fichier"-->
  <!--#flastmod file="nom_de_fichier"-->
  ```

- Exécution de programmes externes

  ```
  <pre>
  <!--#exec cmd="nom d exécutable"-->
  </pre>
  ```

  Délimiteur *pre* permet de prendre en compte les blancs (balises dans le bloc interprétées)

- Exécution d'un script CGI

  ```
  <!--#exec cgi="/cgi-bin/toto.pl"-->
  ```

3.5.2 Exemple de directive SSI

```
<html>
<head>
<title> Exemple simple de document HTML </title>
</head>
<body>
<h2> Ceci est un exemple <i> simpliste </i> de SSI </h2>
<hr>
Il est exactement <!--#echo var="date_local--> <p>
On peut aussi inclure un fichier avec
```



```
<!--#include file="COURS/horloge.html"--> <p>
Ou enfin ex\'ecuter des programmes externes :<br>
<!--#exec cmd="ls"-->
Auteur <a href="http://www.enseeiht.fr/~dayde">Michel Dayd&eacute;</a>
</body>
</html>
```

4 Servlets

4.1 Introduction

- Servlets : classes Java pour gérer les requêtes entrantes vers un serveur et y répondre.
- Le conteneur de servlets peut charger les intances de ces classes de manière dynamique
- Les mêmes servlets compilées et empaquetées sous forme d'application Web peuvent être déployées sous n'importe quelle machine dotée d'un conteneur de servlets compatible tel Tomcat.

Avantage des servlets :

- Chargées dans le serveur pour être exécutées et utilisent moins d'espace et de ressources que les scripts CGI ou Perl qui doivent créer de nouveaux processus pour gérer les requêtes.
- Basées sur Java → large éventail de fonctionnalités (XML, communication,.....).
- Adaptées à des développements lourds : élément de la plate-forme J2EE (Java 2 Enterprise Edition).

Particularité des servlets :

- La spécification stipule que tous les serveurs doivent gérer l'état des sessions.
- Difficile car HTTP est un protocole sans état.
- En écrivant des servlets ou du JSP on n'a pas à gérer des sessions car elles sont automatiquement crées et gérées (objet session).

Servlets = Alternative à CGI utilisant Java

- Classe *Servlet* (et ses extensions) → outils pour traiter les requêtes HTTP et y répondre (documents HTML)
  - *doGetn, doPost* : traitent les opérations GET et POST de HTTP
  - *Servlet Request* : objet permettant de récupérer les paramètres fournis au client
  - *Servlet Response* : objet permettant de préparer une réponse HTML au client

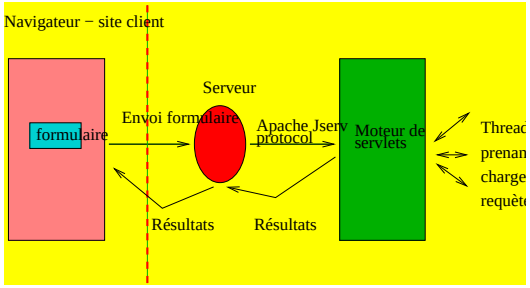


Figure 4: Exécution des Servlets

Autres outils

- Synchronisation entre clients multiples d'un servlet
- Communication entre servlets
- Communication avec d'autres sites

CGI versus Servlets

- Avantages des Servlets : meilleure sécurité, exécution plus efficace (threads au lieu de processus lourds)
- Avantage de CGI : pas de restriction de langage, développement rapide avec langages interprétés (prototypage rapide)
- Servlets préférables pour applis nécessitant connexions externes (accès SGB)

http://java.sun.com/docs/books/tutorial/servlets/

4.2 Exemple simple de Servlet (hello)

Réponse à une opération GET en renvoyant un message prédéfini dans une page HTML.

Objets définis :

- HttpServletRequest = la requète du client (pas utilisée)
- HttpSerletResponse = réponse construite par le serveur renvoyée au client

```
public class SimpleServlet extends HttpServlet
{
/* Repondre a un HTTP GET en renvoyant une page web simple.
 * Surcharge le doGet de la classe HttpServlet
*/
   public void doGet ( HttpServletRequest request,
          HttpServletResponse response )
          throws Servlet Exception, IOException
     {
          PrintWriter  out ;
          String title = "Example Apache JServ Servlet";

          // On fixe le type de la reponse
          response.setContentType("text/html") ;

          // Ecriture du texte de la reponse
          out = response.getWriter() ;

          out.printla("<HTML><HEAD><TITLE>") ;
          out.printla(title) ;
          out.printla("</TITLE></HEAD><BODY>") ;
          out.printla("<H1>" + title + "</H1>") ;
          out.printla("<P>Bonjour de SimpleServelt !") ;
          out.printla("</BODY></HTML>") ;
          out.close() ;
```

4.3 JSP (JavaServer Pages)

- Technologie permettant d'intégrer du code Java exécutable dans des pages HTML ou d'autres documents basés sur du texte comme documents XML.

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

public class RequestInfo 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>Request Information Example</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h3>Request Information Example</h3>");
        out.println("Method: " + request.getMethod());
        out.println("Request URI: " + request.getRequestURI());
        out.println("Protocol: " + request.getProtocol());
        out.println("PathInfo: " + request.getPathInfo());
        out.println("Remote Address: " + request.getRemoteAddr());
        out.println("</body>");
        out.println("</html>");
    }

    /**
     * We are going to perform the same operations for POST requests
     * as for GET methods, so this method just sends the request to
     * the doGet method.
     */

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

```
<html>
<!--
  Copyright (c) 1999 The Apache Software Foundation.  All rights
  reserved.
-->

<body bgcolor="white">
<h1> Request Information </h1>
<font size="4">
JSP Request Method: <%= request.getMethod() %>
<br>
Request URI: <%= request.getRequestURI() %>
<br>
Request Protocol: <%= request.getProtocol() %>
<br>
Servlet path: <%= request.getServletPath() %>
<br>
Path info: <%= request.getPathInfo() %>
<br>
Path translated: <%= request.getPathTranslated() %>
<br>
Query string: <%= request.getQueryString() %>
<br>
Content length: <%= request.getContentLength() %>
<br>
Content type: <%= request.getContentType() %>
<br>
Server name: <%= request.getServerName() %>
<br>
Server port: <%= request.getServerPort() %>
<br>
Remote user: <%= request.getRemoteUser() %>
<br>
Remote address: <%= request.getRemoteAddr() %>
<br>
Remote host: <%= request.getRemoteHost() %>
<br>
Authorization scheme: <%= request.getAuthType() %>
<hr>
The browser you are using is <%= request.getHeader("User-Agent") %>
<hr>
</font>
</body>
</html>
```

Figure 5: Comparaison Servlets et Pages JSP : Extraits exemples JSP Snoop et Servlets request info tiré de Apache Tomcat/4.1.27.

- Avantage : plus léger que les servlets.

5 Apache et Tomcat

- Tomcat : Conteneur de servlets et de JSP (JavaServer Pages)
- Développé sous l'égide de la foundation Apache (créateur sur seveur HTTP Apache le + puissant le sans doute le plus utilisé dans le monde du Web)
- Serveur Apache initialement version "patchée" du serveur NCSA HTTPd en anglais "a patchy version"
- Ensuite création de la fondation Apache

5.1 Foundation Apache

Parmi les projets actuels dans la foundation Apache :

- Serveur HTTP Apache
- Jakarta : ensemble de solutions serveurs open source pour la plateforme Java dont Tomcat, Turbine (développement d'applications Web sécurisées), JMeter (mesure de perf. de servlets), ...
- Projet Apache XML

5.2 Tomcat

- Principal moteur de servlets mais pas le premier qui est JServ.
- Succès de Tomcat : intègre un moteur JSP contrairement à JServ qui a à peu près disparu.
- Tomcat est donc le principal moteur de servlets et l'implantation officielle des servlets et de JSP.

6 PHP ([1])

- Langage interprété exécuté côté serveur
- Syntaxique à la C ou Perl
- Disponible sur UNIX et Windows et intégré aux serveurs récents
- Une fois le script interprété par le serveur → page envoyée au client
- Avantage : le client n'a poas accès au script !!
- Interface simple avec de multiples SGBD

6.1 Exemple simple de script PHP

```
<html>
<head><title> Example PHP : formulaire de saisie </title> </head>

<body>

<form method = "get" action="http://lisa/COURS/simple.php">
```

```
Nom : <input type=text size=20 name=nom>    <p>
Pr&eacute;nom : <input type=text size=20 name=prenom>    <p>
Adresse : <input type=text size=20 name=adresse>    <p>

<input type=submit value=envoyer>
</form>
</body>
</html>
```

```
<html>
<head><title> Example PHP : affichage du script</title> </title>
<body>
<?php
        echo "<h1> Valeur des variables </h1>";
        echo "Nom:$nom <p>";
        echo "Prenom:$prenom <p>";
        echo "Adresse:$adresse <p>";
?>
<body> </body> </html>
```

7 Exécution d'un programme sur un client Web

- Intérêt :
  - Décharger le serveur en utilisant la capacité de traitement locale
  - Rend l'exécution indépendante de la charge du réseau (affichage, ...)
  - Meilleure interactivité
- Mécanismes :
  - Scripts : Tcl, Perl, JavaScript
  - Applet Java

8 Applets

- Définition :

  Programme Java inclus dans une page HTML. Lorsque la page est chargée l'applet est exécutée sur la machine virtuelle Java incluse dans le navigateur.

- Forme :

```
<applet code="MyApplet.class" width=120 height=150></applet>
```

Le programme (en bytecodes) est dans MyApplet.class, dans le même répertoire qui contient l'applet.

Le programme d'une applet doit dériver de la classe standard Applet

```
public class MyApplet extends Applet
{
  ....
}
```

8.1 Restrictions

- Restrictions : capacités d'une applet limitées pour des raisons de sécurité. Interdiction de :
  - Charger des bibliothèques (importation de code nouveau)
  - Accéder aux fichiers locaux sur le site client (encore que ...)
  - Ouvrir une connexion Internet sauf vers le serveur d'où est issue l'applet
  - Lancer un nouveau processus sur le site client
  - Lire certaines caractéristiques du système local

8.2 Capacités des applets

- Sons
- Connexions réseau au serveur d'où elles viennent
- Affichage de pages HTML
- Invocation de méthodes publiques d'autres applets dans la même page
- ...

8.3 Exemple d'applet ([3])

- Applet réagissant à son premier chargement de la page HTML support, démarrage, changement de page, destruction) en affichant un message approprié dans un fenêtre.

- *init, start, ...* sont définies dans la classe Applet et surchargées dans cet exemple. Automatiquement appelées par la navigateur ou l'Appletviewer (inspecteur d'applets) lors des évènements correspondants.
- http://java.sun.com/docs/books/tutorial/aplet/

Applet : Hello

```
import java.applet.Applet;
import java.awt.Graphics;
public class Simple extends Applet {
   StringBuffer buffer;
   public void init() {
      buffer = new StringBuffer();
      addItem("initializing... ");
   }
   public void start() {
      addItem("starting... ");
   }
   public void stop() {
      addItem("stopping... ");
   }


   public void destroy() {
      addItem("preparing for unloading...");
   }
   void addItem(String newWord) {
       System.out.println(newWord);
       buffer.append(newWord);
       repaint();
   }
   public void paint(Graphics g) {
      //Draw a Rectangle around the applet's display area.
      g.drawRect(0, 0, size().width - 1, size().height - 1);

      //Draw the current string inside the rectangle.
      g.drawString(buffer.toString(), 5, 15);
   }
}
```


HTML avec applet

```
<HTML>
<HEAD><TITLE> Applet simple : Hello </TITLE></HEAD>

<BODY>
<H1> Exemple d'applet </H1>
<HR>
<APPLET CODE=Simple.class WIDTH=500 HEIGHT=20>
<HR>
<EM> Votre navigateur ne supporte pas les applets java</EM>
</HR>
</APPLET>
<HR>
Un navigateur qui comprend les applets ignore tout entre hr et /hr <p>
Au contraire un navigateur qui ne comprend pas les applets
ignore tout jusqu'a hr.
</BODY>
</HTML>
```

8.4 Cycle de vie d'une applet

- Lancement de l'applet au chargement de la page :
  - Instance de la sous-classe applet créés
  - L'applet s'initialise elle-même (initializing)
  - Elle démarre son exécution (starting)
- Quand l'utilisateur va vers une autre page → applet stoppe (stopping)
- Redémarrage lorsqu'on revient sur la page
- Arrêt du navigateur → applet stoppe + nettoyage final

8.5 Balise APPLET

Forme le plus simple d'une applet :

```
<APPLET CODE =xxx.class CODEBASE=localisation WIDTH=500 HEIGHT=60>
</APPLET>
```

CODEBASE = répertoire ou URL absolue (permet de charger une applet d'un autre serveur)

8.6 Passage de paramètres

Utilisation de la balise PARAM :

```
<APPLET CODE =xxx.class CODEBASE=localisation WIDTH=500 HEIGHT=60>
<PARAM NAME=parameter1_name VALUE=parameter1_value>
<PARAM NAME=parameter2_name VALUE=parameter2_value>
....
</APPLET>
```

8.7 Récupération des paramètres dans l'applet

Utilisation de *getParameter*

```
public String getParameter(String name)
```

Conversion éventuelle de la chaine de caractère (entier, ...)

8.8 Exemple AppletButton (http://journals.ecs.soton.ac.uk/java/tutorial)

```
String windowClass;
String buttonText;
String windowTitle;
int requestedWidth = 0;
int requestedHeight = 0;
. . .
public void init() {
    windowClass = getParameter("WINDOWCLASS");
    if (windowClass == null) {
        windowClass = "TestWindow";
    }
    buttonText = getParameter("BUTTONTEXT");
    if (buttonText == null) {
        buttonText = "Click here to bring up a " + windowClass;
    }
    windowTitle = getParameter("WINDOWTITLE");
    if (windowTitle == null) {
        windowTitle = windowClass;
    }

    String windowWidthString = getParameter("WINDOWWIDTH");
    if (windowWidthString != null) {
```



```
        try {
            requestedWidth = Integer.parseInt(windowWidthString);
        } catch (NumberFormatException e) {
            //Use default width.
        }
    }
    String windowHeightString = getParameter("WINDOWHEIGHT");
    if (windowHeightString != null) {
        try {
            requestedHeight = Integer.parseInt(windowHeightString);
        } catch (NumberFormatException e) {
            //Use default height.
        }
     }
}
```

Source HTML

```
<p>
<applet code=AppletButton.class codebase=example width=350 height=60>
<param name=windowClass value=BorderWindow>
<param name=windowTitle value="BorderLayout">
<param name=buttonText value="Click here to see a BorderLayout in action">
<blockquote>
<hr>
<em>
Your browser can't run 1.0 Java applets,
so here's a picture of the window the program brings up:</em>
<p>
<img src=images/BorderEx1.gif width=302 height=138>
<hr>
</blockquote>
</applet>
</blockquote>
```

9 JavaScript

Présentation

- Langage de script (interprété) destiné à être exécuté sur un site client Web (utilisation possible côté serveur)

- Programme JavaScript inclus dans une page HTML (comme applet)

  `<script language="JavaScript"> ... texte du programme ... </script>`

- Interpréteur de JavaScript inclus dans le navigateur (issu de Netscape)

Usages : interactions locales sur le site client (évite la consultation du serveur)

- Modifier dynamiquement le contenu d'une page HTML
- Afficher sélectivement des images
- Afficher des boîtes de dialogue, contrôle de validité du remplissage d'un formulaire
- Gérer un historique des documents visités
- ...

JavaScript versus applets

- Domaine d'application plus restreint pour JavaScript (formulaires, images, affichage local, ...)
- Langage de script contre langage compilé
- Sécurité moindre pour JavaScript

9.1 Exemple : affichage heure courante ([3])

```
<HTML>
   <HEAD>
      <SCRIPT LANGUAGE="javascript">
         <!--
         function clock () {
            var date= new Date();
            string=" "+date.getHours()+' : '+
                   date.getMinutes()+' : '+
                   date.getSeconds();
            document.forms[0].elements[0].value=string;
            setTimeout('clock()',150);
         }
         //-->
      </SCRIPT>
    </HEAD>
```



```
    <BODY onLoad="clock();">
       ...
       <FORM>
         Il est tres exactement :
         <INPUT Type="text" Value="hh : mm : nn">
       </FORM>
       ...
    </BODY>
</HTML>
```

9.2 Exemple : alternance entre deux images ([3])

Une image est affichée (et sert d'ancre à un lien hypertexte). Quand la souris entre ou sort du cadre de l'image, une image différente est affichée (on alterne 2 images).

```
...
<SCRIPT language="JavaScript">
var temp="";
var image1 = image.gif ;
var image2 = une_autre_image.gif ;
</SCRIPT>
...
<A HREF="htpp://toto..."
onMouseOver="temp=image1;
image1=image2; image2=temp;
document.mon_image.src=image1;">
<IMG SRC="image.gif" NAME=mon_image></A>
...
```

9.3 Exemple : commerce électronique sur le Web

- Sur le site client : des applets assurent présentation des produits et saisie de la commande (on peut aussi directement utiliser des documents HTML)
- Sur le site serveur : des servlets assurent l'interfaçage avec d'autres composants (base de données de produits, gestion des stocks, gestions des transactions, ...) et renvoient les réponses sous forme de documents HTML

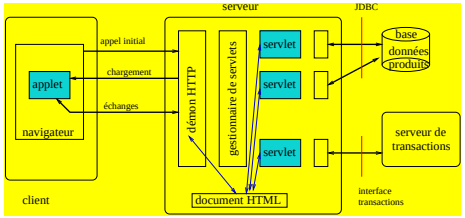


Figure 6: Exemple de commerce électronique.

10 XML et XHTML

10.1 HTML

- HTML né du besoin d'assembler du texte, des graphiques ...
- Norme HTML 4.01 définie à partir de SGML (Standardized Generalized Markup Language)
- Mais SGML trop complexe pour étendre et améliorer les caractéristiques de HTML
- Apparition de XML permettant de développer de nouveaux langages de marquage → XHTML
- XML : métalangage qui permet de définir des langages de marquage tel XHTML

Cela sert à quoi ?

- XML permet de définir des langages de marquage adaptés à des besoins spécifiques
- Langage de marquage satisfaisant pour l'échange d'informations ou le traitement de données dans chaque secteur
- En HTML : navigateur sait quoi faire du marqueur <h1> car définit dans la DTD HTML (Document Type Definition)
- Avec XML on peut créer de nouvelles DTD

- Exemple pour la description de recettes de cuisine introduire les marqueurs *ingredient, portions*
- XML est prometteur auusi pour le partage et la gestion d'informations

Exemple : DTD d'adresses [5]

- Définition d'un élément **adresse** avec plusieurs champs et* un attribut spécifiant adresse domicile ou bureau :

```
<!ELEMENT adresse (nom, rue+, ville, pays, codepostal?)>
<!ATTLIST adresse type (domicile|bureau) #REQUIRED>
```

- adresse = nom + un ou plusieurs éléments d'adressage + un élément ville + un élément pays et un code postal optionnel
- Définition de l'élément **nom** :

```
<!ELEMENT nom (premier, milieu?, dernier)>
<!ELEMENT premier (#PCDATA)>
<!ELEMENT milieur (#PCDATA)>
<!ELEMENT dernier(#PCDATA)>
```

- **nom** = prénom + second prénom optionnel + nom de famille.
- prénom, second prénom et nom n'ont pas d'éléments imbriqués et ne contiennent que des données caractères analysées (i.e. nom véritable de la personne)
- Eléments de l'adresse :

```
<!ELEMENT rue (#PCDATA)>
<!ELEMENT ville (#PCDATA)>
<!ELEMENT pays (#PCDATA)>
<!ELEMENT codepostal (#PCDATA)>
<!ATTLIST codepostal longueur CDATA "5">
```

- L'élément **codepostal** a un attribut **longueur** indiquant la longueur du code postal. S'il n'est pas spécifié, attribut = 5

Utilisation pour marquer des documents :

```
<adresse type="domicile">
   <nom>
       <premier>Michel</premier>
       <dernier>Dayde</dernier>
   </nom>
   <rue> 123 avenue Bidule </rue>
   <ville> Saint Pipi les Agassous </ville>
   <pays> France </pays>
   <codepostal longueur="5">31000</codepostal>
</adresse>
```

Mise en page XML [2]

- XML : format de description de données et non de leur représentation
- Mise en page assurée par un langage tiers :
  - Feuilles de style CSS
  - XSL : langages de feuilles de style extensible développé pour XML
  - XSLT langage Microsotf pour Internet Exeplorer transformant un document XML en document HTML avec feuilles de style

XML pour l'échange de documents

- Vrai puissance de XML → échange de documents
- Partage et échange de données au sein d'applications au sein d'applications réparties complexes (schémas de stockage différents, erreurs, lourd, ...)
- XML générique, pas propriétaire, indépendant des plates-formes
- Permet d'effectuer presque toutes les tâches de capture ou de siasie des données
- Echange des documents indépendants de leur présentation
- XML utilsé par un application existante pour intégrer des documents issus d'une autre source

10.2 XHTML

- Un des premiers langages définit avec XML est une version de HTML "XHTM L"
- XHTML version plus disciplinée de HTML (pbs de versions et de norme, ...)
- HTML : ensemble limité de procédures de création de documents
- Pas de possibilités pour formules mathématique, notations musicales, ...
- Pbs avec affichage sur PDA, téléphones cellulaires, ...
- ...
- Créatrion de documents XHTML similaire à HTML
- Déclaration de la DTD utilisée pour créer le document au début et définition d'un espace de noms au début du document XHTML

Exemple de document XHTML ([5])

```
<?xml version="1.0" encoding ="UTF-8"?>
<!DOCTYPE html
          PUBLIC "-//W"C//DTD XML 1.0 Transitional//EN"
          " http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd ">
<html xmls="http://www.w3.org/1999/xhtml" xml:lang="en" lang="eng">
<head>
   <title> Mon premier document XHTML</title>
</head>
<body>
  .....
</body>
</html>
```

Explications :

- Utisation de version 1 de XHTML avec encodage Unicode 8 bits
- DOCTYPE → on suit les règles de la DTD transitoire XHTML 1.0
- Attribut xmlns indique que l'espace de noms XHTML est celui par défaut de tout le document

- eng : anglais utilisé à la fois dans les espaces de noms XML et XHT ML

11 Aspects systèmes

Intérêts d'un cache

- Introduit un niveau intermédiaire, d'accès rapide (car local)
- Réduit temps moyen d'accès en conservant informations les plus utilisées
- Réduit le trafic entre les niveaux de stockage de l'information

Web bien adapté

- Informations changent peu souvent dans la plupart des cas
- On peut regrouper les demandes :
  - Cache individuel sur disque
  - Cache local pour un département
  - Cache régional

Problèmes

- Choix des informations à conserver
- Politique de mise à jour du cache en particulier quand il est plein
- Rafraîchissement des informations
- Coopération entre caches

11.1 Gestion des caches

Politique de remplacement

- FIFO : dans l'ordre des arrivées
- RANDOM : choisir un document au hasard
- SIZE : éliminer le document le plus gros, gestion à court terme
- LRU (Least Recently Used) : hypothèse de localité, fréquemment utilisé

Cohérence : comment garantir que les documents sont à jour ?

- Invalidation : le serveur prévient le cache quand l'original est modifié : idéal mais coût gestion par le serveur qui doit garder trace des copies
- TTL (Time To Live) : durée de vie limitée (élimination ou rappel à la date d'expiration)
- Durée de vie proportionnelle à l'age du document

Coopération entre caches

- Hiérarchie : tout cache à un parent auquel il transmet la requête s'il ne peut la résoudre, et ainsi de suite, si pas de parent contacter le serveur puis réponse au fils éventuel
- Entre égaux : un cache transmet la requête aux caches frères et au serveur : il prend la première réponse
- Mode de coopération pas fixé et peut dépendre de la nature des requêtes

References

[1] Max Buvry, Support de cours Base de Données, 1ère année Télécommunications et Réseaux, 2001.

[2] V. Charvillat et Romulus Grigoras, Un peu plus loin avec les technologies multimédia, Polycope ENSEEIHT, 2001.

[3] Sacha Krakowiak, *Introduction aux Systèmes et Réseaux Informatiques*, Université J. Fourier, Grenoble, http://sirac.imag.fr.

[4] Michel Gabassy et Bertrand Dupouy, *L'Informatique Répartie sour UNIX*, Collection de la Direction des Etudes et Recherches d'Electricité de France, Eyrolles, 1992.

[5] Chuck Musciano et Bill Kennedy, *HTML et XHTML, La référence*, O'Reilly, Paris, 2001.

[6] Miche Riveill, *Construction d'applications réparties - Introduction*, Notes de Cours INPG / ENSIMAG, 1999.