
USTL – Master mention Informatique – M1

Construction d'applications réparties

III. Internet et servlet

Lionel.Seinturier@lifl.fr

Plan

1. Introduction
2. HTML
3. Formulaires
4. HTTP
5. Servlet

Introduction

WWW : World Wide Web

Créé en 1989 au CERN par Tim Berners-Lee

Mettre en ligne de la documentation (initialement technique pour physiciens)

Services de l'Internet

- Web
- messagerie
- transfert de fichiers (FTP)
- forum de nouvelles (*news*), de discussion (*chat*)
- *peer-to-peer* (Gnutella, Freenet, ...), diffusion de flux audio et vidéo (*streaming*)

Principes du Web

- **hypertexte** : HTML
- **client/serveur** : HTTP
- **schéma de désignation** : URL

Introduction

Hypertexte

Texte "normal"

- organisation linéaire
- éventuellement renvois sous forme de sommaires, index, notes de bas de page

Hypertexte

- origine : Vanevar Bush (MEMEX, 1945)
- organisation pas forcément linéaire
- texte enrichi de **liens**
 - ⇒ renvoi vers un document
 - ⇒ renvoi vers une partie du même document
 - ⇒ renvoi vers une partie d'un autre document

Introduction

Client/serveur

Client : le navigateur (Internet Explorer, Netscape, ...)
Serveur : le serveur Web (Apache, Microsoft IIS, ...)

1. Le client **émet** une requête
2. Le serveur **répond** en fournissant le document demandé ou un message d'erreur si le document n'existe pas



Introduction

Schéma de désignation

Uniform Resource Locator (URL)

Permet de désigner une page Web
Chaque page a un nom unique ⇔ pas d'ambiguïté possible

```
protocole://serveur/page  
http://www.lip6.fr/index.html
```

Organisation hiérarchique possible pour les pages (≡ hiérarchie fichier disque)

```
protocole://serveur/répertoire/.../page  
http://www.lip6.fr/congres/2002/index.html
```

Introduction

Historique du Web

1989 : création par **Tim Berners-Lee** (CERN)
1993 : **Mosaic** 1er navigateur graphique (NCSA)
1994 : World Wide Web Consortium (**W3C**)
- créé par le CERN et le MIT
- organisme de **standardisation du Web**
- administré par le MIT et l'INRIA
1994 : **Netscape** navigateur (Win, Unix, Mac)
développé notamment par
Marc Andreessen (un des auteurs de Mosaic)
1996 : Microsoft **Internet Explorer**
1996 : Microsoft **Internet Explorer**
1996 : 1ère version d'**Opera** en téléchargement (1994 Telenor R&D, Norway)
2003 : création de la Mozilla Foundation (**Firefox**, Thunderbird)



Introduction

Standards

HTTP

- 0.9 : version de base avec requête/réponse
- 1.0 : version standardisée IETF (RFC 1945)
- 1.1 : version étendue (connexions persistantes)

HTML

- 1.0 : version initiale
- 2.0 : version standardisée W3C
- 3.2 : version étendue (tableaux, images cliquables, applets)
- 4.0 : version étendue (*frames*, feuilles de style)

en devenir : XHTML, XML

URL

format stable depuis 1989

Introduction

Bibliographie (pléthorique)

Ouvrages sur HTML www.amazon.fr
- 96 en français
- 228 en anglais

Tutoriels HTML en ligne www.yahoo.fr
- 34 en français
- 105 en anglais

Sélection

WebMaster in a Nutshell. S. Spainhour, R. Eckstein. 2ème édition. O'Reilly.

Concis et couvrant beaucoup d'aspects :
HTML, CSS, XML, XSL, JavaScript, CGI, PHP, HTTP, Apache.

HyperText Markup Language (HTML)

Lionel Seinturier

Université des Sciences et Technologies de Lille

Lionel.Seinturier@lifl.fr

16/9/03

HTML

HyperText Markup Language (HTML)

HTML : langage de balisage issu de SGML
Document HTML contient :
- du texte
- des **balises** (ou *tags*) : directives de mises en forme

Exemple : `<I>Ce texte est en italique</I>`

SGML (Standard Generalized Markup Language) créé en 1986
- format de document des professionnels de l'édition et de la gestion documentaire
- norme ISO 8879
- langage très complet
- suite bureautique SGML (éditeur + navigateur) très lourde à gérer

HTML : version allégée de SGML

HTML

Structure d'un document HTML

Déclaration version HTML utilisée

En-tête

Corps du document

```
<!DOCTYPE ... >
<HTML>
  <HEAD>
    En-tête
  </HEAD>
  <BODY>
    Corps du document
  </BODY>
</HTML>
```

- Déclaration de version et en-tête **facultatifs**
- Absence de corps $\Rightarrow$ document vide (sauf cas spéciaux, ex.: *frames*)

HTML

Règles de balisage

2 types de balises

- autonome `< ... >` (ex. : `<! DOCTYPE >`)
- délimitant une zone
 - balise de début de zone `< ... >` (ex. : `<HTML>`)
 - balise de fin de zone `</ ... >` (ex. : `</HTML>`)

- identifiants de balise non sensibles à la casse (`<BODY>` $\equiv$ `<body>`)
- certaines balises sont associées à un ou +sieurs attributs
 - chaque attribut à un identifiant et une valeur (entre "" et après =)
 - ex. : `<IMG SRC="photo.gif" ALT="une photo">`

- commentaires `<!-- ceci est un commentaire -->`
- encodage de caractère `& ... ;`
 - `< > &` `< > &`
 - caractères accentués `& lettre accent ;`
 - é à ô `&ecute; ` ô`

HTML

Déclaration version HTML utilisée

Langage et version utilisés par le document

⇒ les navigateurs savent s'ils pourront afficher le document

```
<! DOCTYPE langage statut version >
```

Déclarations couramment rencontrées

```
<! DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 3.2" >
```

```
<! DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0" >
```

statut de la version : PUBLIC | PRIVATE

numérotation W3C

HTML

En-tête des documents HTML

Informations sur le document

```
<TITLE>un titre</TITLE>
```

titre du document / apparaît dans la barre supérieur du navigateur

```
<META NAME="propriété" CONTENT="valeur">
```

propriété du document / non affichée / +sieurs propriétés peuvent être déf.
pas de contrainte ni d'obligation sur les propriétés définies

```
<BASE HREF="préfixe URL">
```

préfixe commun à toutes les URLs utilisées dans le document / non affiché

```
<HEAD>
  <TITLE>Bienvenue sur la page web de Lionel Seinturier</TITLE>
  <META NAME="auteur" CONTENT="Lionel Seinturier">
  <META NAME="contenu" CONTENT="Cours Web">
  <META NAME="mis à jour" CONTENT="tous les ans"
</HEAD>
```

HTML

Corps des documents HTML

Balises principales

```
<BODY attr1="val1" ... attrn="valn">
```

attributs possibles **valeurs par défaut** en cas d'absence

- BGCOLOR : couleur de fond
- TEXT : couleur du texte
- BACKGROUND : URL de l'image de fond d'écran
- LINK : couleur des liens non encore visités
- VLINK : couleur des liens visités
- ...

```
<H1>Titre</H1>
```

attribut possible

```
<H2> <H3> <H4> <H5> <H6>
```

ALIGN="left|center|right"

```
<P> ... </P>
```

paragraphe

```
<BR>
```

passage à la ligne

```
<HR>
```

trait horizontal

```
<B> ... </B>
```

gras

```
<I> ... </I>
```

italique

HTML

Couleurs

2 solutions

- identificateurs prédéfinis : green, yellow, purple, blue, red, ...
- nombre de 6 chiffres hexadécimaux (ex. : #1A7FC0)
codant les intensités de rouge/vert/bleu
- intensité

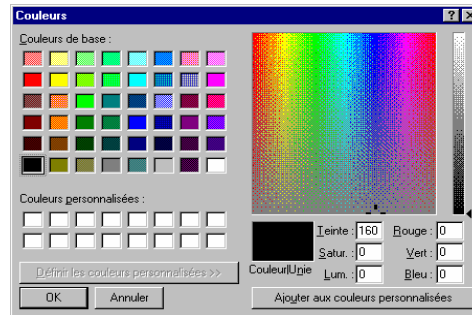
Equivalence entre les 2

blue ≡ #0000FF

purple ≡ #800080

...

La plupart des éditeurs HTML fournissent des outils de type "palette de couleurs"



HTML

Liens hypertexte

Portion de texte permettant d'atteindre un document désigné par une **URL**

texte du lien

URL absolue LIP6

URL relative sommaire

plan d'accès

accueil

Définition de **signets (anchors)** ≡ marques placées dans un document

Désignation de signets à l'aide du caractère # dans les URLs

conclusion du rapport

conclusion du rapport

HTML

Compléments sur les URLs

Éléments constitutifs (optionnels) d'une URL

protocole://	protocoles utilisables : HTTP, FTP, MAILTO, FILE, NEWS
utilisateur@	login pour les pages protégées
serveur	nom ou adresse IP (ex. : 163.136.27.6) du serveur
:numéro	numéro de port TCP du serveur
/répertoire	chemin d'accès dans la hiérarchie de répertoires
fichier	nom du document à atteindre
#signet	signet dans le document à atteindre
?options	information transmise au serveur (ex. : formulaire)

Exemple

http://bob@12.7.6.1:8080/pub/cv.html#diplomes
ftp://anonymous@ftp.lip6.fr
http://www.altavista.com/query.html?jussieu

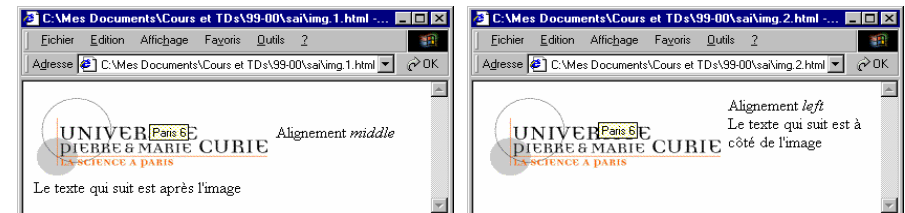
HTML

Images

Insertion d'une image (gif ou jpeg) dans un document

Attributs possibles

- HEIGHT="99" / WIDTH : hauteur/largeur en pixels de l'image
- ALT=" ..." : légende associée à l'image
- HSPACE="99" / VSPACE : espacement en pixels horiz./vert. autour de l'image
- ALIGN="bottom|middle|top|left|right" : alignement de l'image / au texte



HTML

Listes

Listes numérotées (...) ou non (...)

```
<UL>                                balise de début de liste
  <LI> Rouge </LI>                  balises pour un élément de la liste
  <LI> Vert </LI>                   balises pour un élément de la liste
  ...
</UL>                                balise de fin de liste
```

Attributs possibles

```
<UL TYPE="disc|circle|square">    : type de puce
<OL TYPE="1|I|i|A|a"               : type de numérotation
  START="99">                       : départ de la numérotation

<LI TYPE="disc|circle|square">     : type de puce (liste UL)
<LI TYPE="1|I|i|A|a ">             : type de numérotation (liste OL)
```

HTML

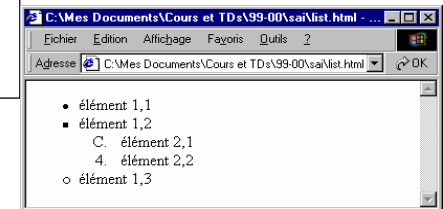
Listes

Entrelacement possible des définitions de listes

```
<UL>
<LI> élément 1,1 </LI>
<LI type="square"> élément 1,2 </LI>

  <OL type="A" start="3">
    <LI> élément 2,1 </LI>
    <LI type="1"> élément 2,2 </LI>
  </OL>

<LI type="circle">
  élément 1,3 </LI>
</UL>
```



HTML

Tableaux

Tableaux à 2 dimensions (<TABLE> ... </TABLE>)

```
<TABLE>                                balise de début de tableau
  <TR>                                balise de début de ligne (row)
    <TD> cellule </TD>                balises pour une colonne (data)
    ...
  </TR>                                balise de fin de ligne
  ...
</TABLE>                                balise de fin de tableau
```

Attributs possibles pour <TABLE>

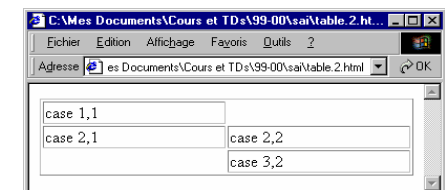
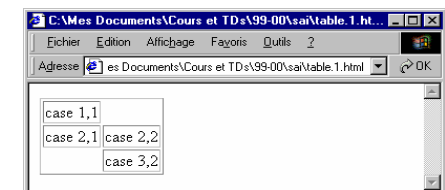
- BORDER="99" (0) : épaisseur en pixels des bordures du tableaux
- ALIGN="left|right" : alignement du tableau / au texte
- WIDTH="99|99%" : largeur du tableau en pixels | en % largeur fenêtre
- CELSPACING="99" (2) : espacement en pixels entre les cellules
- CELLPADDING="99" (1) : marge intérieure en pixels des cellules

HTML

Tableaux

Possibilité de laisser des cellules vides

```
<TABLE BORDER="1">
<TR> <TD> case 1,1 </TD> </TR>
<TR> <TD> case 2,1 </TD>
      <TD> case 2,2 </TD> </TR>
<TR> <TD></TD>
      <TD> case 3,2 </TD> </TR>
</TABLE>
```



WIDTH="100%"

HTML

Tableaux

Attributs possibles pour <TR>

- ALIGN="**left**|right|center|justify" : alignement horiz. du texte (≠ <TABLE>)
- VALIGN="**middle**|top|bottom" : alignement vertical du texte

Attributs possibles pour <TD>

- ALIGN/VALIGN : idem <TR>, supprime <TR ALIGN/VALIGN >
- WIDTH="99|99%" : largeur de la colonne en pixels | en % largeur tableau
- COLSPAN="99" : aggrège plusieurs cellules horizontalement
- ROWSPAN="99" : aggrège plusieurs cellules verticalement

Balise <CAPTION> Légende associée au tableau

- ALIGN="**top**|bottom" : placement de la légende

HTML

Tableaux

```
<TABLE BORDER="1" WIDTH="100%">

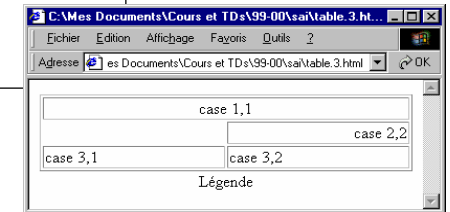
<CAPTION ALIGN="bottom"> Légende </CAPTION>

<TR ALIGN="center">
  <TD COLSPAN="2"> case 1,1 </TD> </TR>

<TR> <TD></TD>
  <TD ALIGN="right"> case 2,2 </TD> </TR>

<TR> <TD> case 3,1 </TD>
  <TD> case 3,2 </TD> </TR>

</TABLE>
```



HTML

Frames

Permettent de **scinder** la fenêtre du navigateur en +sieurs parties (*frames*)

Principe

- un découpage (<FRAMESET> ... </FRAMESET>) défini dans un fichier HTML
- chaque *frame* du *frameset* est associée à une balise <FRAME>
- autant de fichiers HTML qu'il y a de *frames* définies dans le *frameset*

Attributs possibles pour la balise <FRAMESET>

- ROWS="99|99%, .." : liste des tailles de ligne du découpage
- COLS="99|99%, .." : liste des tailles de colonne du découpage

Attributs possibles pour la balise <FRAME>

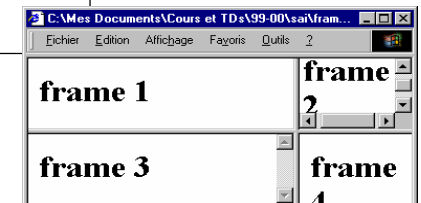
- SRC="URL" : URL du fichier à afficher dans la *frame*
- NORESIZE : interdit le redimensionnement de la *frame*
- SCROLLING="**auto**|yes|no" : barres de défilement verticale et horizontale

HTML

Frames

Exemple de fichier de *frameset* - index.html

```
<HTML>
<FRAMESET ROWS="50%, 50%" COLS="70%, *">
  <FRAME SRC="frame1.html" NORESIZE>
  <FRAME SRC="frame2.html">
  <FRAME SRC="frame3.html" SCROLLING="yes">
  <FRAME SRC="frame4.html" SCROLLING="no">
</FRAMESET>
</HTML>
```



Attributs ROWS/COLS

- * : taille restante
- $\sum > 100\%$ | taille fenêtre en pixels $\Rightarrow$ règle de 3
- $\sum < 100\%$ | taille fenêtre $\Rightarrow$ dernière taille ignorée

HTML

Frames

Entrelacement possible des définitions de *framesets*

Liens hypertexte vers les *frames*

Chaque *frame* peut être associée à un nom

```
<FRAME SRC="..." NAME="frameDeGauche">
```

Les liens peuvent référencer cette *frame* à l'aide de l'attribut TARGET

```
<A HREF="URL" TARGET="frameDeDroite">
```

Identificateurs de TARGET réservés

- TARGET="_self" : affichage du document dans la **frame du lien**
- TARGET="_blank" : affichage du document dans une **nouvelle fenêtre**
- TARGET="_parent" : affichage dans la **fenêtre contenant le frameset**
- TARGET="_top" : affichage dans la **fenêtre du lien**

HTML

Images cliquables

Définition d'un lien () autour d'une image ()

- ⇒ l'image acquiert un cadre bleu (enlève le cadre)
- ⇒ un clic sur l'image provoque le chargement du lien

Image cliquable sous forme de plan

- définition d'un plan (<MAP> ... </MAP>)
- chaque zone du plan (<AREA ... >)
 - a une forme : rectangle, cercle ou polygone
 - est associée à une URL
- plaquage du plan sur une image ()

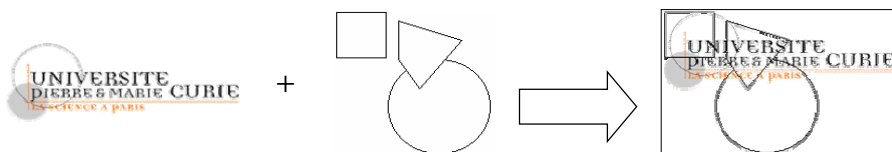
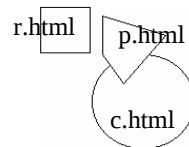
- rectangles : coordonnées des **coins** supérieur gauche et inférieur droit
- cercles : coordonnées du **centre** et **rayon**
- polygones : coordonnées de **tous les coins**

HTML

Images cliquables

```
<IMG SRC="map.gif" USEMAP="#plan">

<MAP NAME="plan">
  <AREA SHAPE="rectangle" COORDS="0,0,50,50"
    HREF="r.html">
  <AREA SHAPE="circle" COORDS="100,100,50"
    HREF="c.html">
  <AREA SHAPE="polygon"
    COORDS="60,10,60,50,80,80,120,30,60,10"
    HREF="p.html">
</MAP>
```



Formulaires HTML

Lionel Seinturier

Université Pierre & Marie Curie

Lionel.Seinturier@lip6.fr



Formulaires

Formulaires

HTML 1.0 essentiellement "mono"-directionnel

- informations fournies **par le serveur** (suite à une commande client)

Utilisation professionnelle

- nécessité de flux d'information bi-directionnels (client ↔ serveur)
- HTML 2.0 introduit les **formulaires**
- permettent aux clients de saisir des informations
- qui seront envoyées aux serveurs

Un formulaire est défini **en HTML** et peut contenir

- zones de saisie de texte
- boîtes à cocher
- boutons radio
- menus déroulants
- boutons

Formulaires

Exemple de formulaire HTML

```
<HTML> <BODY>

<FORM ACTION="http://monserveur.com/prog.php" METHOD=POST>

Nom <INPUT NAME="client" SIZE=46> <P>
Rue <INPUT NAME="rue" SIZE=40> <P>
Ville <INPUT NAME="ville" SIZE=20>
Code postal <INPUT NAME="cp" SIZE=5> <P>
Carte de crédit No <INPUT NAME="carte" SIZE=10>
Expire <INPUT NAME="expire" TYPE=TEXT SIZE=4> <P>

M/C <INPUT NAME="cc" TYPE=RADIO VALUE="mc" CHECKED>
VISA <INPUT NAME="cc" TYPE=RADIO VALUE="vis"> <P>

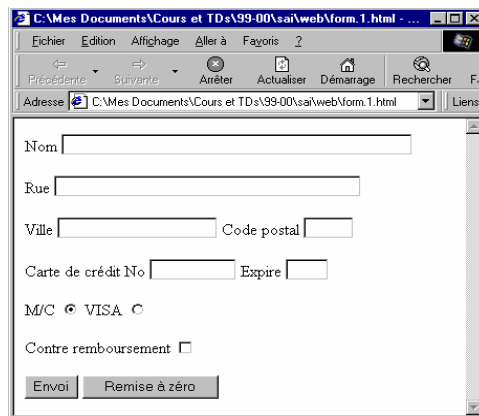
Contre remboursement <INPUT NAME="cr" TYPE=CHECKBOX> <P>

<INPUT TYPE=SUBMIT VALUE="Envoi">
<INPUT TYPE=RESET VALUE="Remise à zéro"> <P>

</FORM> </BODY> </HTML>
```

Formulaires

Exemple de formulaire HTML



Formulaires

Balises <FORM> </FORM>

Déclaration d'un formulaire

Attributs principaux (<FORM ACTION=.. METHOD=.. NAME=.. >)

ACTION	URL vers laquelle envoyer les données saisies	
METHOD	commande HTTP à utiliser pour effectuer l'envoi	⇒ POST
NAME	nom du formulaire	
TARGET	nom de la <i>frame</i> dans laquelle le résultat doit être affiché	

- toutes les balises HTML sont permises entre <FORM> </FORM>
- ⇒ images, tableaux, ... peuvent être inclus dans un formulaire

- des formulaires peuvent être insérés à l'intérieur d'un autre formulaire

Formulaires

Balise <INPUT>

Déclaration des champs de saisie (exclusivement entre <FORM> </FORM>)

NAME nom du champ de saisie (unique à l'intérieur d'un formulaire)

TYPE type du champ de saisie

Types possibles (TYPE= . . .)

TEXT **zone de saisie texte** (type par défaut en cas d'omission de TYPE)

SIZE : taille apparente MAXLENGTH : taille max

RADIO **bouton radio**

tous les boutons ayant même nom (NAME) ∈ au même groupe
dans ce cas, les attributs (VALUE) permettent de les différencier

CHECKBOX **boîte à cocher**

SUBMIT **bouton d'envoi** des données au serveur

RESET **bouton d'effacement** du formulaire

Formulaires

Envoi des données au serveur

Lorsque l'utilisateur appuie sur le bouton SUBMIT, le navigateur

- construit une **chaîne de caractères** contenant toutes les données du formulaire
- envoie cette chaîne au serveur

Chaîne

- ensemble de couples séparées par le caractère &
- chaque couple est de la forme nom de champ = valeur saisie
- les espaces sont remplacés par le caractère +
- les caractères + & = sont encodés %2B %26 %3D

Exemple (1 seule ligne)

client=Jean+Vier&rue=54+rue+Gambetta&ville=Paris&
cp=75001&carte=0123456789&cc=vis&cr=on

Rq : pour les boîtes à cocher =on si cochée, rien sinon

Formulaires

Autres types possibles pour la balise <INPUT>

PASSWORD zone de saisie d'un mot de passe + attribut ENCTYPE

FILE sélection d'un fichier à envoyer

```
<HTML> <BODY>
```

```
<FORM ACTION="http://monserveur.com/prog.php" METHOD=POST  
  ENCTYPE="multipart/form-data">
```

```
  Mot de passe <INPUT TYPE=PASSWORD NAME="passe" SIZE=16> <P>
```

```
  Sélectionner un fichier <INPUT TYPE=FILE NAME="fichier"> <P>
```

```
  <INPUT TYPE=SUBMIT VALUE="Envoi">
```

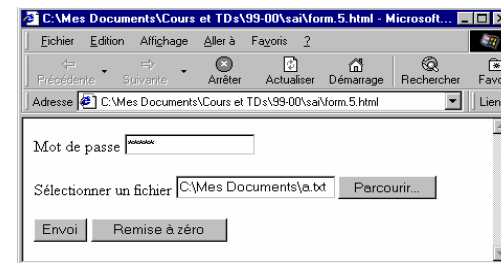
```
  <INPUT TYPE=RESET VALUE="Remise à zéro"> <P>
```

```
</FORM>
```

```
</BODY> </HTML>
```

Formulaires

Autres types possibles pour la balise <INPUT>



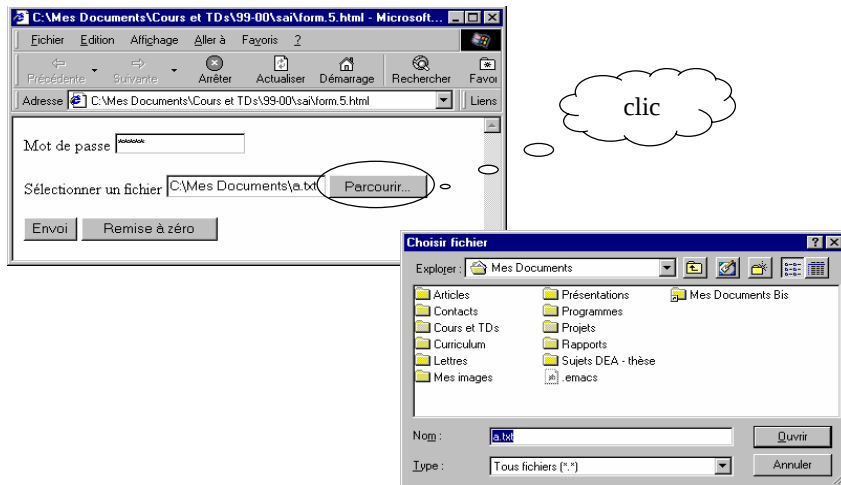
PASSWORD les caractères saisis sont remplacés par des *

FILE provoque l'affichage

- d'un champ de saisie du nom du fichier
- d'un bouton **Parcourir** pour sélectionner le fichier via une fenêtre de parcours du disque

Formulaires

Autres types possibles pour la balise <INPUT>



Formulaires

Encodage fichiers joints

```
-----7d225420d803c8
Content-Disposition: form-data; name="fichier"; filename="..."
Content-Type: image/gif
```

```
GIF89a& ... contenu binaire du fichier ...
-----7d225420d803c8
Content-Disposition: form-data; name="passe"
```

en clair

```
-----7d225420d803c8--
```

- séparateur déterminé aléatoirement à chaque upload par le navigateur
 - + dans les en-têtes HTTP de la requête
- ```
Content-Type: multipart/form-data;
boundary=-----7d225420d803c8
```

- filename navigateurs Windows : nom local complet (C:\Mes documents\toto.doc)
- filename navigateurs Unix/Mac : nom local simple (toto.doc)

## Formulaires

### Autres types possibles pour la balise <INPUT>

HIDDEN champ caché (ne provoque aucun affichage)  
BUTTON un bouton simple ⇒ association avec un traitement JavaScript

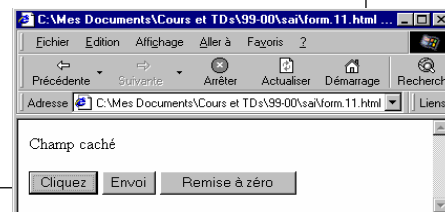
```
<HTML> <BODY>
<FORM ACTION="http://monserveur.com/prog.php" METHOD=POST>
```

```
Champ caché <INPUT TYPE=HIDDEN NAME="cache" VALUE="Lionel"> <P>
<INPUT TYPE=BUTTON VALUE="Cliquez" onClick="fonctionJavaScript()>
```

```
<INPUT TYPE=SUBMIT VALUE="Envoi">
<INPUT TYPE=RESET
VALUE="Remise à zéro"> <P>
```

```
</FORM>
```

```
</BODY> </HTML>
```

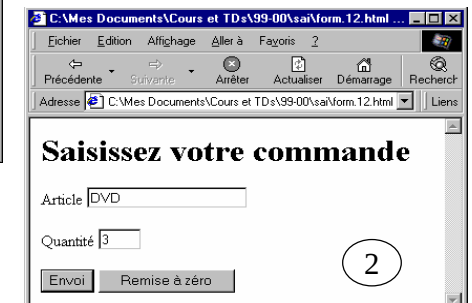
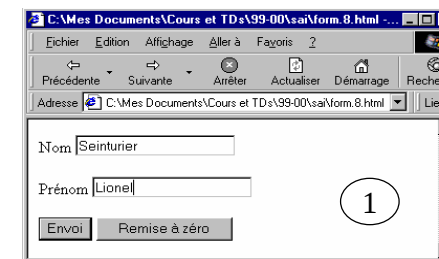


## Formulaires

### Autres types possibles pour la balise <INPUT>

#### Champ HIDDEN

⇒ transmission d'informations "furtives" dans une chaîne de formulaires

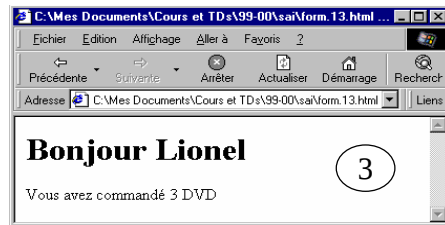


## Formulaires

### Autres types possibles pour la balise <INPUT>

#### Champ HIDDEN

⇒ transmission d'informations "furtives" dans une chaîne de formulaires



prog. 1→2      insertion d'un champ caché dans le form. 2 (VALUE="Lionel")  
prog. 2→3      récupération de caché pour générer form. 3

## Formulaires

### Autres balises possibles dans un formulaire

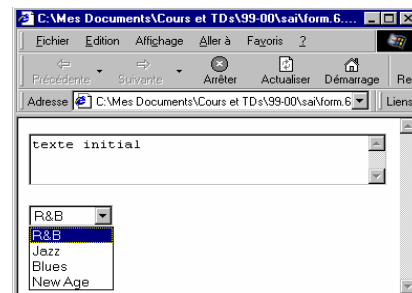
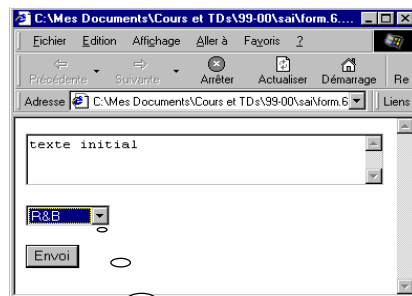
TEXTAREA      zone de saisie d'un texte sur plusieurs lignes  
SELECT          définition d'un menu déroulant  
                 balise OPTION pour définir les choix du menu

```
<HTML> <BODY>
<FORM ACTION="http://monserveur.com/prog.php" METHOD=POST>

<TEXTAREA NAME="zone" ROWS=3 COLS=40>texte initial</TEXTAREA> <P>
<SELECT NAME="musicTypes">
 <OPTION> R&B
 <OPTION> Jazz
 <OPTION> Blues
 <OPTION> New Age
</SELECT> <P>
<INPUT TYPE=SUBMIT VALUE="Envoi">
</FORM> </BODY> </HTML>
```

## Formulaires

### Autres balises possibles dans un formulaire



## Formulaires

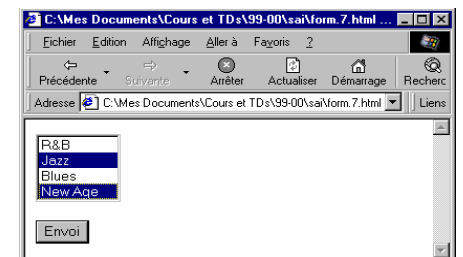
### Autres balises possibles dans un formulaire

SELECT MULTIPLE      définition d'un menu déroulant à choix multiples

```
<HTML> <BODY>
<FORM ACTION="http://monserveur.com/prog.php" METHOD=POST>

<SELECT MULTIPLE NAME="musicTypes">
 <OPTION> R&B
 <OPTION> Jazz
 <OPTION> Blues
 <OPTION> New Age
</SELECT> <P>
<INPUT TYPE=SUBMIT VALUE="Envoi">
</FORM> </BODY> </HTML>
```

Par défaut taille zone = 4,  
sinon attribut size



# HyperText Transfert Protocol (HTTP)

Lionel Seinturier

Université Pierre & Marie Curie

Lionel.Seinturier@lip6.fr

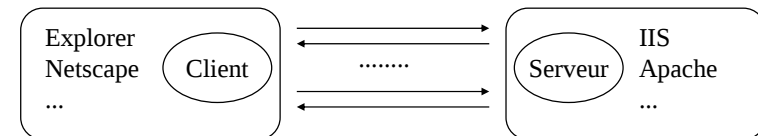
# HTTP

## HyperText Transfert Protocol (HTTP)

Protocole réseau d'échange de l'information sur le Web

Principe de base

- **un** couple requête/réponse **par document** à charger
- + **autant** de req./rép. qu'il y a d'**éléments inclus** (images, frames, sons, ...)



Les éléments inclus sont désignés par des **URLs**

Ils peuvent être localisés

- sur le même site (que le document) et dans le même répertoire
- sur le même site mais dans un répertoire différent
- sur un site différent

# HTTP

## Caractéristiques HTTP

- Protocole applicatif (niveau 7)
- Utilise TCP (niveau 4) ⇒ garantie d'un transport **fiable** (sans erreur)
- **Pas** de notion de **connexion** HTTP

- Toutes les commandes HTTP sont émises en **mode texte** (ASCII)

⇒ Protocole simple, facilement implantable

Permet de

- charger des documents textes (HTML) ou binaires (GIF, JPG, son, ...)
- envoyer des informations au serveur Web
- déposer des documents textes ou binaires sur le serveur Web
- charger/déposer n'importe quel type de fichier

Version actuelle : HTTP/1.1

# HTTP

## Exemple de session HTTP

Client	Serveur
eole:~ > telnet www-src.lip6.fr 80	--> Demande cx TCP
Trying 132.227.64.100 ...	
Connected to www-src.lip6.fr.	<-- Cx TCP acceptée
Escape character is '^['.	
GET /index.html HTTP/1.1	--> Envoi commande HTTP
HTTP/1.1 200 OK	<-- Réception réponse HTTP
Server: Apache/1.3b2	
Last-Modified: Mon, 01 Feb 1999 13:55:20 GMT	
Content-Length: 9872	
Content-Type: text/html	
<HTML>	
<HEAD> <TITLE>Syst&egrave;mes</TITLE> </HEAD>	
<BODY BGCOLOR="#FFFFFF">	
Le code HTML de la page	
</BODY>	
</HTML>	
Connection closed by foreign host.	<-- Fermeture cx TCP

# HTTP

## Requête HTTP

Permet d'envoyer des commandes au serveur Web

3 commandes principales (présentes dans HTTP/1.0 et 1.1)

GET	demande d'un document
HEAD	demande de l'en-tête (HTTP) d'un document
POST	dépose d'information sur le serveur

GET : requête principale, 80 % des requêtes Web  
HEAD : au travers de l'en-tête, permet de savoir si un document a changé  
POST : envoi d'informations saisies dans un formulaire client

# HTTP

## Requête HTTP

commande	URL	version HTTP comprise par le client
en-tête 1 HTTP: valeur		
....		
en-tête n HTTP: valeur		

informations envoyées par le client

```
GET /index.html HTTP/1.1
Accept-language: fr
User-Agent: Mozilla/4.0
If-modified-since: Tue, 23 Jul 1997 10:24:05
```

```
POST /index.php HTTP/1.1
Accept-language: fr

nom=Seinturier&prenom=Lionel
```

# HTTP

## Requête HTTP

En-têtes client

• informations sur le client	From, Host, User-Agent
• information sur la page contenant le lien	Referer
• <i>login</i> , mot de passe	Authorization
• préférences pour le document demandé	Accept ...
• conditions sur le document demandé	If ...

Accept: liste de types MIME  
Accept-Charset, Accept-Encoding, Aspect-Language  
If-Modified-Since, If-Unmodified-Since

# HTTP

## Réponse HTTP

version HTTP du serveur	code retour	commentaire
en-tête 1 HTTP: valeur		
....		
en-tête n HTTP: valeur		

document texte (HTML, ...) ou binaire (GIF, JPG, ...)

```
HTTP/1.1 200 OK
Content-Length: 9872
Content-Type: text/html
```

```
<HTML>
...
</HTML>
```

# HTTP

## Réponse HTTP

code retour : renseigne sur le succès (200) ou l'échec (4xx) de la requête

- 200 : ok
- 404 : document inconnu
- 401 : authentification nécessaire
- 500 : erreur du serveur HTTP dans le traitement requête (servlet, PHP, ...)
- 503 : serveur temporairement surchargé
- ...

en-têtes HTTP : informations transmises par le serveur sur le document envoyé

- Content-Length : taille du document
- Last-Modified : date de dernière modification du document
- Server : nom du logiciel serveur
- Expire : date d'expiration du document
- Content-Type : type (MIME) du document
- ... **nombreux** autres en-têtes possibles

# HTTP

## Réponse HTTP

Type (MIME) du document

Classification des documents de la forme **médium/format**

- text/html : document texte HTML
- text/plain : document texte "brut"
- image/gif : image GIF
- image/jpeg : image JPEG
- audio/wav : fichier son au format .wav
- video/mpeg : fichier vidéo au format .mpeg
- application/ps : fichier généré par une application au format PostScript

- application/octet-stream : fichier d'octets "brut"

+ **nombreux** autres types existants (voir RFC 1341 et 1521)

# HTTP

## HTTP 1.1

Depuis janvier 1997 (RFC 2067)

Apport principal

### • connexions persistantes

- ⇒ les connexions TCP restent ouvertes
- ⇒ pendant x secondes (dépend de la configuration du serveur)
- ⇒ les éléments inclus dans un document (*frames*, images, sons, ...) peuvent être chargés "dans la foulée" du document HTML (s'ils sont situés sur le même serveur)

Raison : pour les "petits" fichiers (< 10 Ko, 80 % des documents Web)  
le coût de l'ouverture de cx TCP est **non négligeable** / coût du transfert  
⇒ gain de temps important

# HTTP

## HTTP 1.1

Autres apports

3 commandes supplémentaires

- |        |                                                          |
|--------|----------------------------------------------------------|
| PUT    | déposer un fichier sur le serveur                        |
| DELETE | effacer un fichier du serveur (si autorisation !)        |
| TRACE  | obtenir un diagnostic des requêtes reçues par le serveur |

# HTTP

## Evolutions de HTTP

### HTTP-NG (New Generation)

- plusieurs types de protocoles en fonction des données (audio/vidéo/HTML/...)
- RPC (plutôt que TCP), ASN.1
- authentification des clients

### S-HTTP (Secure)

- sécurité au niveau HTTP
  - nouvelles URL (shhttp://)
  - protection des données échangées
    - cryptage (DES)
    - authentification (RSA)
  - prise en compte de la protection au travers des caches
- ⇒ solution supplantée par sécurité au niveau socket (SSL) + HTTP "normal"
- ⇒ URL https://

# Servlet

Lionel Seinturier

Université des Sciences et Technologies de Lille

Lionel.Seinturier@lifl.fr

# Servlet

## Servlet

Programme Java s'exécutant **côté serveur Web**

servlet            prog. "autonome" stockés dans un fichier **.class** sur le serveur  
JSP                prog. **source** Java embarqué dans une page **.html**

	côté client	côté serveur
.class autonome	applet	servlet
embarqué dans .html	JavaScript	JSP

### Servlet et JSP

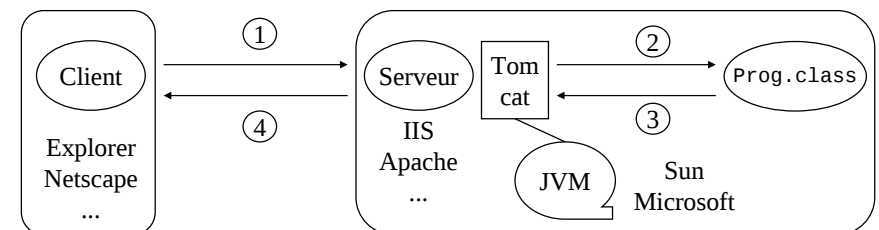
- exécutable avec tous les serveurs Web (Apache, IIS, ...)
- auxquels on a ajouté un "moteur" de servlet/JSP (le plus connu : **Tomcat**)
- JSP compilées automatiquement en servlet par le moteur

# Servlet

## Servlet

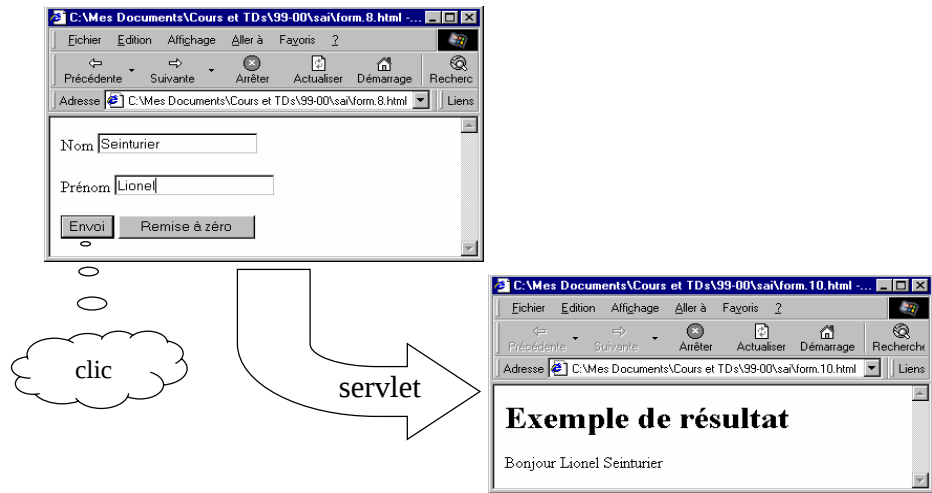
### Principe

- les fichiers de *bytecode* (.class) sont stockés sur le serveur (comme des docs)
- par convention dans le répertoire **servlet/**
- ils sont désignés par une URL    `http://www.lifl.fr/servlet/Prog`
- le chargement de l'URL provoque l'exécution de la servlet



# Servlet

## Illustration du fonctionnement des servlets



# Servlet

## Mécanismes mis en œuvre

- écriture d'une servlet = écriture d'une **classe Java**
  - lors du premier chargement d'une servlet (ou après modification), le **moteur**
    - **instancie** la servlet
      - ⇒ servlet = objet Java présent dans le moteur
  - puis, ou lors des chargements suivants, le **moteur**
    - exécute le code dans un **thread**
- ⇒ le code produit un résultat qui est envoyé au client  
⇒ en cas d'erreur dans le code Java de la servlet message récupéré dans le navigateur

# Servlet

## Développement

Utilisation des packages Java `javax.servlet.*` et `javax.servlet.http.*`

- extension de la classe `javax.servlet.http.HttpServlet`
- redéfinition de la méthode `service` de cette classe
- définit le code à exécuter lorsque la servlet est invoquée
- elle est appelée automatiquement par le "moteur" de servlet

```
void service(ServletRequest request , ServletResponse response);
```

représente la **requête**  
envoyée par le client  
⇒ renseigné automatiquement  
par le "moteur"

représente la **réponse**  
retournée par la servlet  
⇒ à **renseigner** dans  
le code de la servlet

# Servlet

## Aperçu de l'API servlet

Méthodes appelables sur un objet request

- `String getParameter(String param)`  
retourne la **valeur du champ** param transmis dans les données du formulaire
- `java.util.Enumeration getParameterNames()`  
retourne l'ensemble des noms de paramètres transmis à la servlet
- `String getMethod()`  
retourne la méthode HTTP (GET ou POST) utilisée pour invoquer la servlet

Méthodes appelables sur un objet response

- `void setContentType(String type)`  
définit le **type MIME** du document retourné par la servlet
- `PrintWriter getWriter()`  
retourne un **flux de sortie** permettant à la servlet de produire son résultat  
la servlet "print" le code HTML sur ce flux de sortie

## Servlet

### Exemple de servlet

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

public class HelloServlet extends HttpServlet {

 public void service(ServletRequest request,
 ServletResponse response)

 throws ServletException, IOException

 {
 response.setContentType("text/html");

 PrintWriter out = response.getWriter();

 }
```

imposé

imposé par l'API servlet

ce qui suit est en HTML

recupère un flux pour générer le résultat

## Servlet

### Exemple de servlet (suite)

```
out.println("<html><body>");
out.println("<h1>Hello depuis une servlet</h1>");
out.println("</body></html>");

} }
```

génération du code HTML

### Compilation

⇒ HelloServlet.class installé dans l'arborescence de Tomcat

Chargement via une URL de type `http://.../servlet/HelloServlet`  
⇒ exécution de HelloServlet.class



## Servlet

### Exemple de servlet (code complet)

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

public class HelloServlet extends HttpServlet {

 public void service(ServletRequest request,
 ServletResponse response)

 throws ServletException, IOException {

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

 out.println("<html><body>");
 out.println("<h1>Hello depuis une servlet</h1>");
 out.println("</body></html>");

 } }
```

## Servlet

### Deuxième exemple de servlet

Chaque servlet n'est instanciée **1 seule fois**

⇒ persistance de ces données entre 2 invocations

```
public class CompteurServlet extends HttpServlet {

 int compteur = 0;

 public void service(ServletRequest request,
 ServletResponse response)

 throws ServletException, IOException {

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

 out.println("<html><body>");
 out.println("<h1> " + compteur++ + "</h1>");
 out.println("</body></html>");

 } }
```

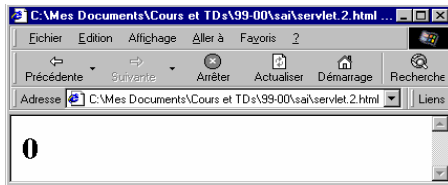
## Servlet

### Deuxième exemple de servlet

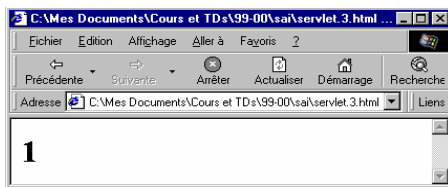
Chaque servlet n'est instanciée **1 seule fois**

⇒ persistance de ces données entre 2 invocations

1ère invocation



2ème invocation



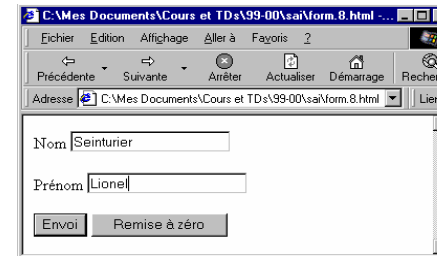
## Servlet

### Récupération des données d'un formulaire

Méthode `String getParameter(String)` d'un objet `request`

⇒ retourne le texte saisi

⇒ ou null si le nom de paramètre n'existe pas



```
<HTML> <BODY>
<FORM ACTION="http://..."
 METHOD=POST>
Nom <INPUT NAME="nom"> <P>
Prénom <INPUT NAME="prenom"> <P>
<INPUT TYPE=SUBMIT VALUE="Envoi">
<INPUT TYPE=RESET
 VALUE="Remise à zéro">
</FORM>
</BODY> </HTML>
```

## Servlet

### Récupération des données d'un formulaire

```
public class CompteurServlet extends HttpServlet {

 public void service(ServletRequest request,
 ServletResponse response)
 throws ServletException, IOException {

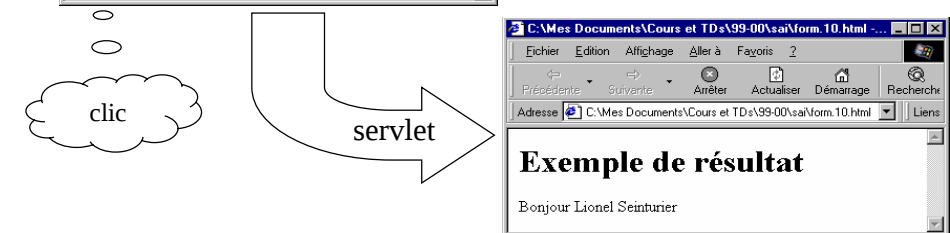
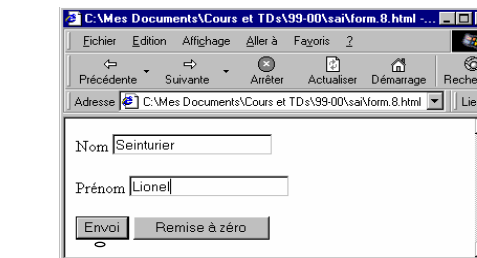
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();

 String nom = request.getParameter("nom");
 String prenom = request.getParameter("prenom");

 out.println("<html><body>");
 out.println("<h1>Exemple de résultat</h1>");
 out.println("Bonjour " + prenom + " " + nom);
 out.println("</body></html>");
 }
}
```

## Servlet

### Récupération des données d'un formulaire



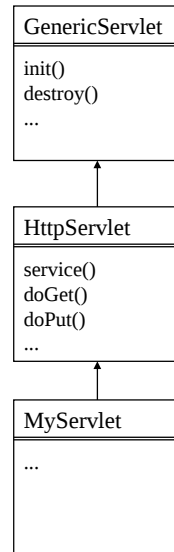
# Servlet

## Cycle de vie d'une servlet

- void init(ServletConfig conf)  
méthode appelée par le moteur au **démarrage** de la servlet
  - ⇒ peut être utilisée pour initialiser la servlet
  - ⇒ propager l'initialisation par super.init(conf)
  - ⇒ ne jamais utiliser de constructeur
- void destroy()  
lorsque l'on veut **détruire** la servlet

## Différenciation des méthodes HTTP

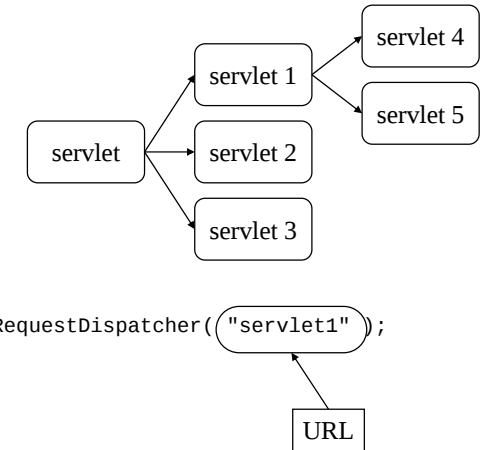
- service() traite toutes les requêtes HTTP
- doGet() doHead() doPost() doPut() doDelete() doTrace() peuvent être redéfinies pour traiter chaque requête HTTP de façon différenciée



# Servlet

## Chaînage des servlets

- agrégation des résultats fournis par plusieurs servlets
  - ⇒ meilleure modularité
  - ⇒ meilleure réutilisation



## Utilisation d'un *RequestDispatcher*

- obtenu via un objet request

```
RequestDispatcher rd = request.getRequestDispatcher("servlet1");
```

## Inclusion du résultat d'une autre servlet

```
rd.include(request, response);
```

## Délégation du traitement à une autre servlet

```
rd.forward(request, response);
```

# Servlet

## Compléments sur l'API

### Méthodes appelables sur un objet request

- String getProtocol()  
retourne le protocole implanté par le serveur (ex. : HTTP/1.1)
- String getServerName() / String getServerPort()  
retourne le nom/port de la machine serveur
- String getRemoteAddr() / String getRemoteHost()  
retourne l'adresse/nom de la machine cliente (ayant invoqué la servlet)
- String getScheme()  
retourne le protocole utilisé (ex. : http ou https) par le client

# Servlet

## Gestion de la concurrence

Par défaut les servlets sont exécutées de façon multi-*threadée*

Si une servlet doit être exécutée en exclusion mutuelle

(ex. : accès à des ressources partagées critiques)

implantation de l'interface **marqueur** `SingleThreadModel`

```
public class CompteurServlet
 extends HttpServlet
 implements SingleThreadModel {

 public void service(ServletRequest request,
 ServletResponse response)
 throws ServletException, IOException {
 /** Du code en exclusion mutuelle avec lui-même */
 } };
```

Autre solution : définir du code `synchronized` dans la servlet

## Servlet

### Cookies

Permettent à un serveur Web de stocker de l'information chez un client

- ⇒ moyen pour savoir "par où passe" un client, quand, en venant d'où, ...
- ⇒ débat éthique ??
- ⇒ l'utilisateur a la possibilité d'interdire leur dépôt dans son navigateur

- définis dans la classe `javax.servlet.http.Cookie`
- on les crée en donnant un nom (String) et une valeur (String)  
`Cookie uneCookie = new Cookie( "sonNom", "saValeur" );`
- on les positionne via un objet `response`  
`response.addCookie( uneCookie );`
- on les récupère via un objet `request`  
`Cookie[] desCookies = request.getCookies();`

Quelques méthodes :      `String getName() / String getValue()`

## Servlet

### Suivi de session

- HTTP protocole non connecté
- pour le serveur, 2 requêtes successives d'un même client sont **indépendantes**

Objectif : être capable de "suivre" l'activité du client sur +sieurs pages

Notion de session

- ⇒ les **requêtes** provenant d'un **utilisateur** sont associées à une même session
- ⇒ les sessions ne sont pas éternelles, elles **expirent** au bout d'un délai fixé

Sur un objet `request`

- `HttpSession session = request.getSession(true)`  
retourne la session courante pour cet utilisateur ou une nouvelle session
- `HttpSession session = request.getSession(false)`  
retourne la session courante pour cet utilisateur ou `null`

## Servlet

### Suivi de session

Méthodes appelables sur un objet de type `HttpSession`

- `void setAttribute( String name, Object value )`  
ajoute un couple (name, value) pour cette session
- `Object getAttribute( String name )`  
retourne l'objet associé à la clé name ou `null`
- `void removeAttribute( String name )`  
enlève le couple de clé name
- `java.util.Enumeration getAttributeNames()`  
retourne tous les noms d'attributs associés à la session
- `void setMaxIntervalTime( int seconds )`  
spécifie la durée de vie maximum d'une session
- `long getCreationTime() / long getLastAccessedTime()`  
retourne la date de création / de dernier accès de la session  
en ms depuis le 1/1/1970, 00h00 GMT → `new Date(long)`

## Servlet

### Partage de données entre servlets

Notion de contexte d'exécution

= ensemble de couples (name, value) partagées par toutes les servlets **instanciées**

`ServletContext ctx = getServletContext()` (héritée de `GenericServlet`)

Méthodes appelables sur un objet de type `ServletContext`

- `void setAttribute( String name, Object value )`  
ajoute un couple (name, value) dans le contexte
- `Object getAttribute( String name )`  
retourne l'objet associé à la clé name ou `null`
- `void removeAttribute( String name )`  
enlève le couple de clé name
- `java.util.Enumeration getAttributeNames()`  
retourne tous les noms d'attributs associés au contexte

# Servlet

---

## Conclusion

Permettent d'étendre le comportement des serveurs Web avec des prog. Java

Résumé des fonctionnalités

- + portabilité, facilité d'écriture (**Java**)
- + gestion des applications requérant un suivi entre plusieurs programmes (**persistance des données** dans les servlets)
- + servlet chargée et instanciée **une seule fois**
- + servlet exécutée avec des processus légers (*threads*)