

HTTP et Servlets

Introduction

Formulaires

HTTP

CGI

Servlets

Introduction

- WWW : World Wide Web
- créé en 1989 par Tim Berners-Lee au CERN :
mise en ligne de documentation scientifique

- Les concepts :

hypertexte
client/serveur
schéma de désignation

langage HTML
protocole HTTP
nommage par URL

Introduction : hypertexte

Texte classique

- organisation linéaire
- avec index, sommaire, glossaire, notes,...

Hypertexte

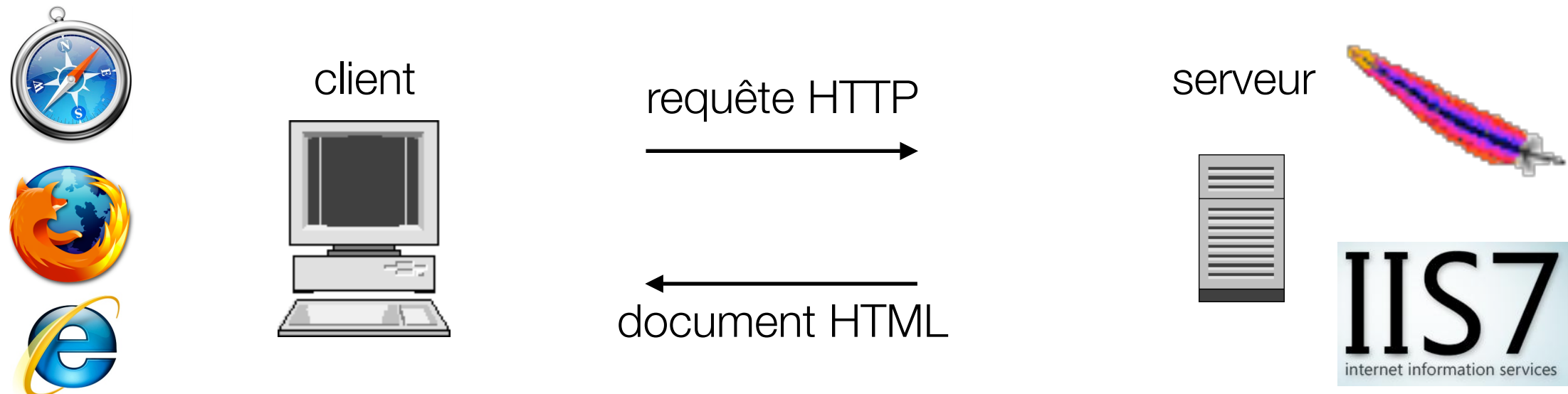
- organisation non linéaire (hiérarchique, graphe, ...)
- texte enrichi de liens
 - renvoi vers un document
 - renvoi vers une partie du même document
 - renvoi vers une partie d'un autre document

Introduction : client/serveur

- Client : le navigateur (Firefox, Safari, Internet Explorer, ...),
- Serveur : le serveur Web (Apache, Microsoft IIS, ...)
- Le protocole : HTTP

1. Le client émet la 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)

- désigne une page Web
- Chaque page a un nom unique => pas d'ambiguïté possible

protocole://serveur/page

`http://www.lifl.fr/index.html`

- Organisation hiérarchique possible des pages

protocole://serveur/repertoire/.../page

`http://www.lifl.fr/seminaires/patarin.html`

Introduction : url dans sa forme générale

```
protocole"://" [utilisateur[":"motdepasse]@] (nom|adresse) [:port]  
["/"chemin] ["/"nomdefichier] [#ancrelocale] [options]
```

Le format complet est défini dans les RFC 1738 et 1808

<http://www.w3.org/Addressing/>

Exemples :

- <http://www.w3.org/Addressing/>
- <ftp://user:foo@research.digital.com/personal/latex.sty>
- <http://123.87.54.251/index.html>
- <http://www.google.fr/search?q=lille+gmi>
- <http://xenon.inria.fr:8080/hello.html>
- <http://alken.lifl.fr/index.html#annuaire>

Introduction : historique

- 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, depuis 2003 par le MIT et l'ERCIM (Groupement Européen de Recherche en Informatique et en Mathématiques)
- 1994 : navigateur Netscape (Win, Unix, Mac) développé notamment par Marc Andreessen un des auteurs de Mosaic
- 1996 : Microsoft Internet Explorer 
- 1996 : 1ère version d'Opera en téléchargement (1994 Telenor R&D, Norway)
- 2003 : création de la Mozilla Foundation (Firefox, Thunderbird) 
- 2008 : Google Chrome 

Introduction : standards

HTTP

- 0.9 : version de base avec requête/réponse
- 1.0 : version standardisée IETF (RFC 1945)
- 1.1 : version étendue (connexions persistantes) (RFC 2616)

HTML

- 1.0 version initiale
- 2.0 version standardisée W3C
- 3.0 version étendue (tableaux, images cliquables, applets)
- 4.0 version étendue (frames, feuilles de styles)
- 5.0

XML

- XHTML 1.1, groupe de travail sur XHTML 2.0 (abandonné)

URL : format stable depuis 1989

Formulaires

HTML 1.0 est essentiellement mono directionnel :
informations fournies par le serveur (suite à une commande client)

Utilisation professionnelle : nécessité de flux d'information bidirectionnels
(client ↔ serveur)

HTML 2.0 introduit les formulaires

- permettent aux clients de saisir des informations
- qui seront envoyées aux serveurs

un formulaire est écrit en HTML et peut contenir :

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

Formulaires : exemple

```
<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>

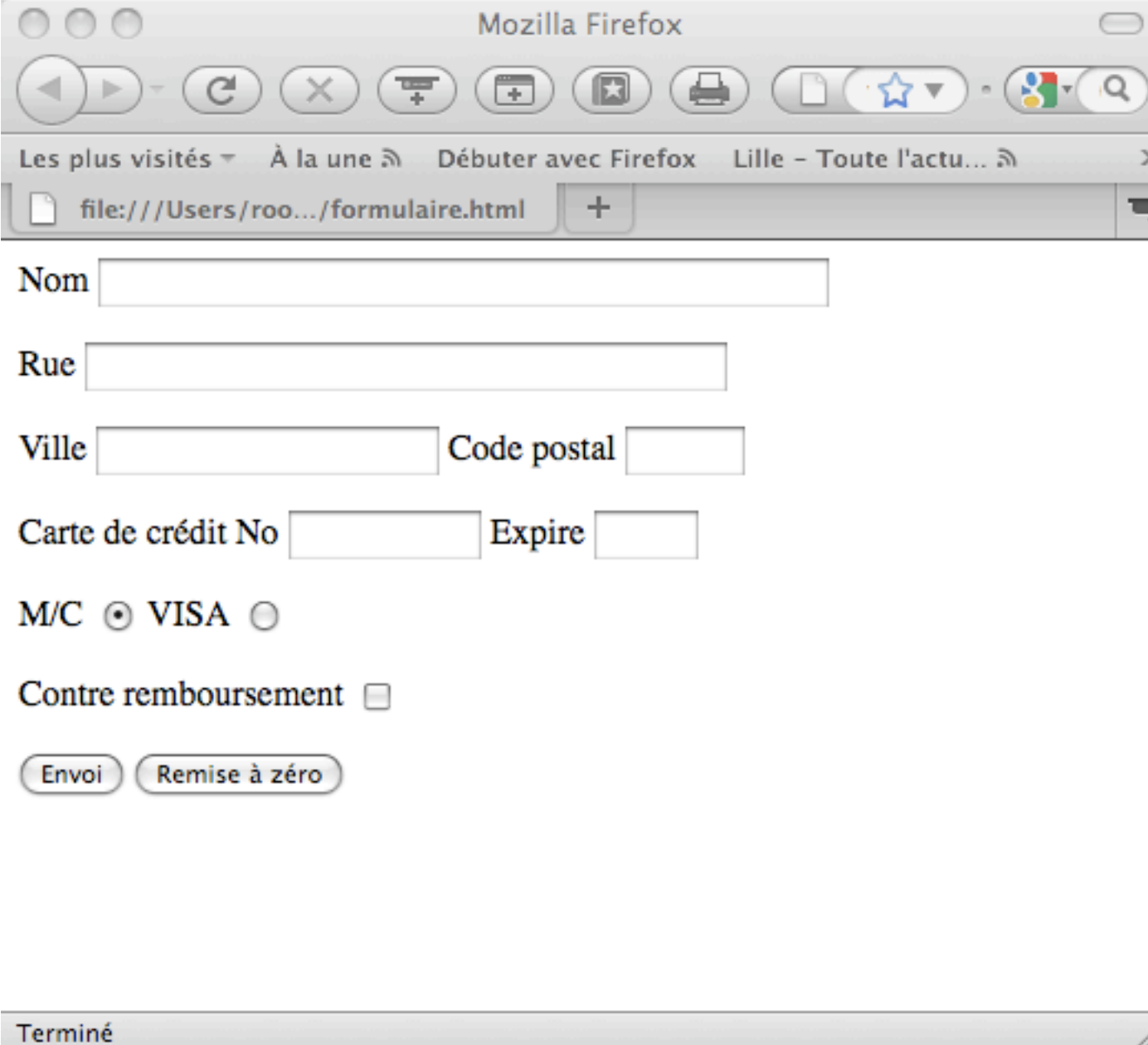
M/C <INPUT NAME="cc" TYPE=RADIO VALUE="mc" CHECKED>
VISA <INPUT NAME="cc" TYPE=RADIO VALUE="vis"> <P>

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



The screenshot shows a Mozilla Firefox browser window with the following elements:

- Address Bar:** Displays the file path `file:///Users/roo.../formulaire.html`.
- Form Fields:**
 - Nom:** A single-line text input field.
 - Rue:** A single-line text input field.
 - Ville:** A single-line text input field.
 - Code postal:** A single-line text input field.
 - Carte de crédit No:** A single-line text input field.
 - Expire:** A single-line text input field.
- Radio Buttons:** Labeled "M/C", with "VISA" selected (indicated by a filled circle).
- Checkbox:** Labeled "Contre remboursement", which is currently unchecked.
- Buttons:** Two buttons labeled "Envoi" and "Remise à zéro".
- Status Bar:** At the bottom, it displays the word "Terminé".

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 frame 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 maximum
- 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 séparés 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 (sur 1 seule ligne)

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

Formulaires : autres types 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 NAME="passe" SIZE=16 TYPE=PASSWORD> <P>
```

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

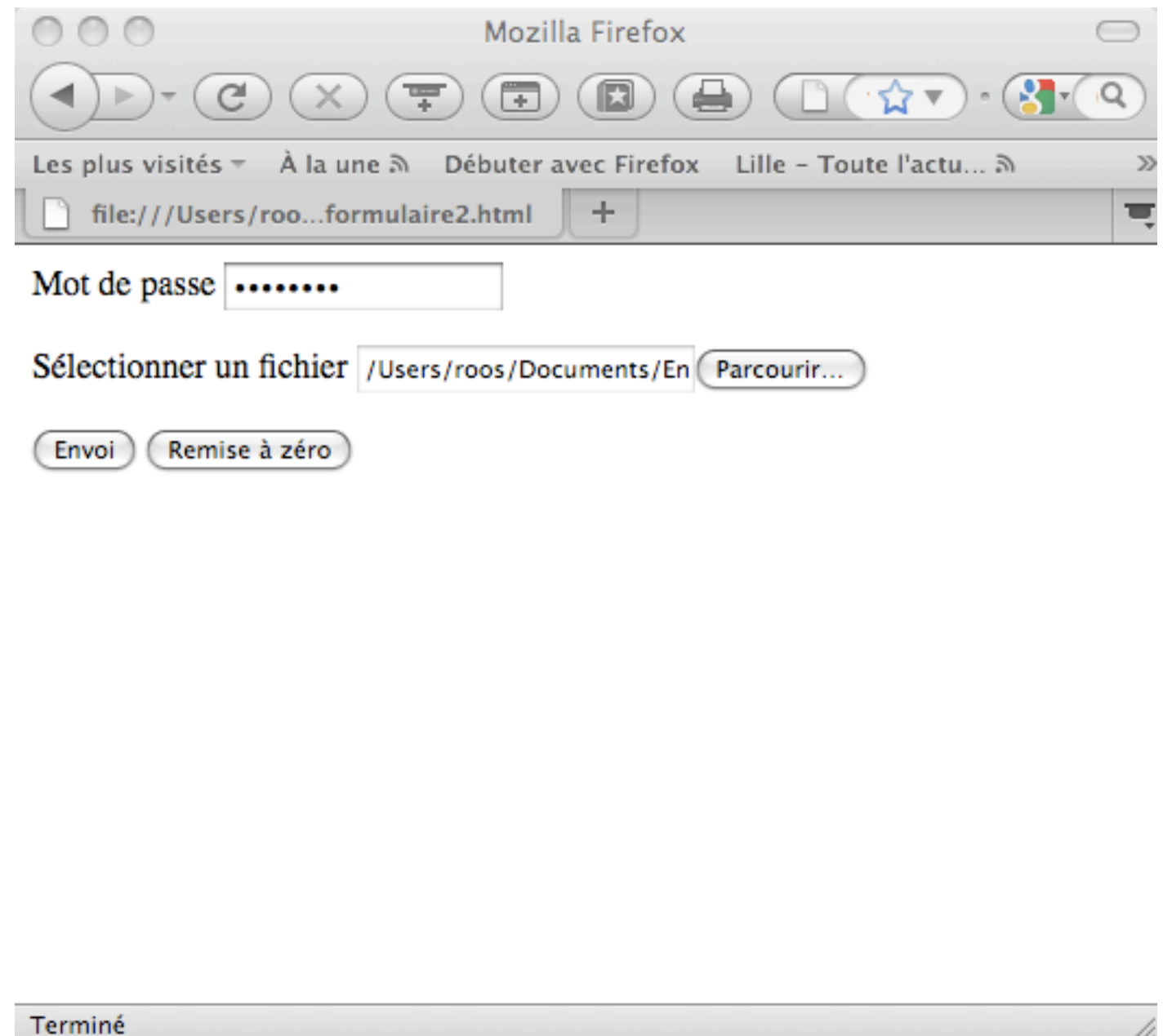
```
<INPUT TYPE=SUBMIT VALUE="Envoi">
```

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

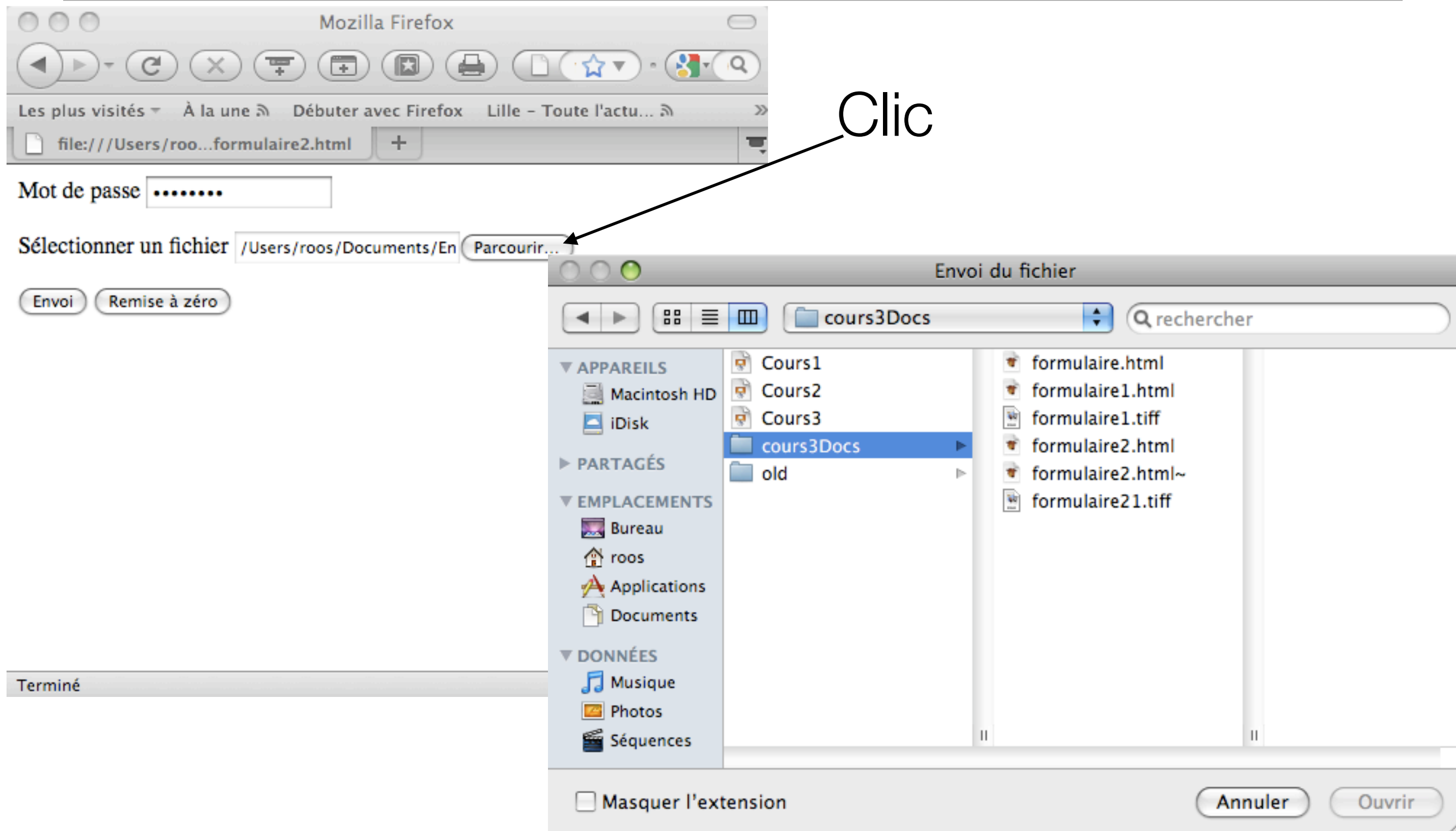
```
</FORM></BODY> </HTML>
```

Formulaires : autres types pour la balise <INPUT>

- `PASSWORD` les caractères saisis sont masqués
- `FILE` provoque l'affichage :
 - d'un champ de saisie du nom du fichier
 - d'un bouton Parcourir pour sélectionner le fichier via une fenêtre



Formulaires : autres types pour la balise <INPUT>



Formulaires : autres types pour la balise <INPUT>

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 et ajouté dans les en-têtes HTTP de la requête

```
Content-Type: multipart/form-data;
```

```
boundary=-----7d225420d803c8
```

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

Formulaires : autres types 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>
```

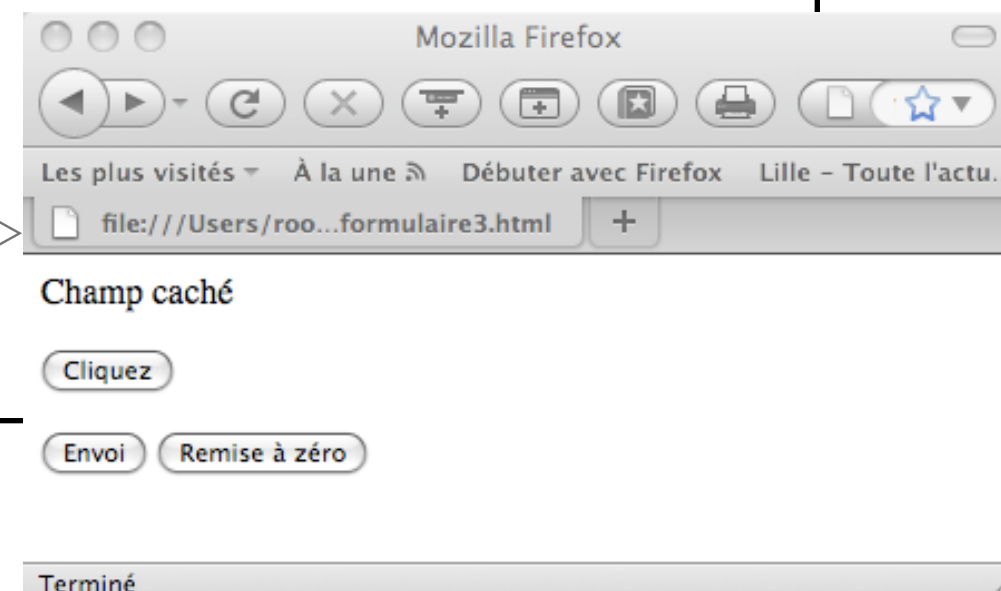
```
Champ caché <INPUT NAME="cache" VALUE="JF" TYPE=HIDDEN> <P>
```

```
<INPUT VALUE="Cliquez" onClick="fonctionJavaScript()" TYPE=BUTTON>  
<P>
```

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

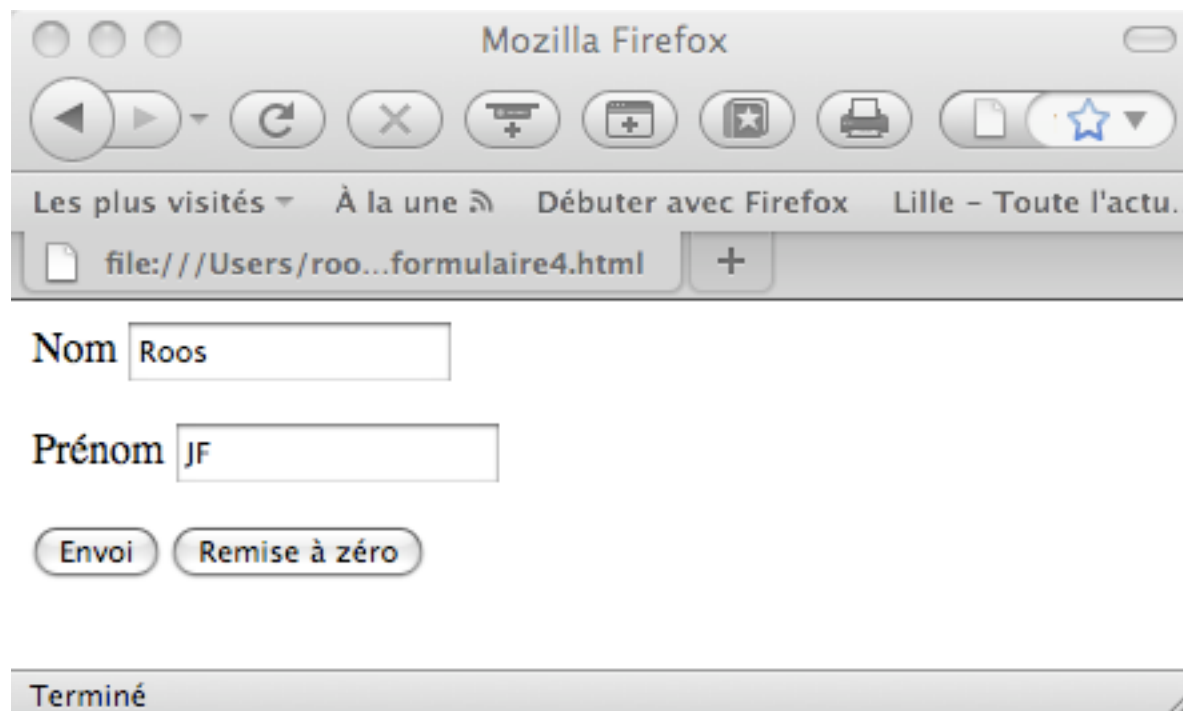
```
<INPUT TYPE=RESET VALUE="Remise à zéro"> <P>
```

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



Formulaires : autres types pour la balise <INPUT>

- champ HIDDEN : transmission d'informations furtives dans une chaîne de formulaire
- 1 à 2 : insertion d'un champ caché dans le formulaire 2 (VALUE="JF")



A screenshot of a Mozilla Firefox browser window displaying a web form. The address bar shows the file path: file:///Users/roo...formulaire4.html. The form contains two text input fields: "Nom" with the value "Roos" and "Prénom" with the value "JF". Below these fields are two buttons: "Envoi" and "Remise à zéro". At the bottom of the form, there is a status bar that says "Terminé". A circled number "1" is placed below the form.



A screenshot of a Mozilla Firefox browser window displaying a web form. The address bar shows the file path: file:///Users/roo...formulaire5.html. The form has a title "Saisissez votre commande" in bold. It contains two text input fields: "Article" with the value "DVD" and "Quantité" with the value "3". Below these fields are two buttons: "Envoi" and "Remise à zéro". A circled number "2" is placed above the form.

Formulaires : autres types pour la balise <INPUT>

- 2 à 3 : récupération de la valeur du champ caché pour générer le formulaire 3

3

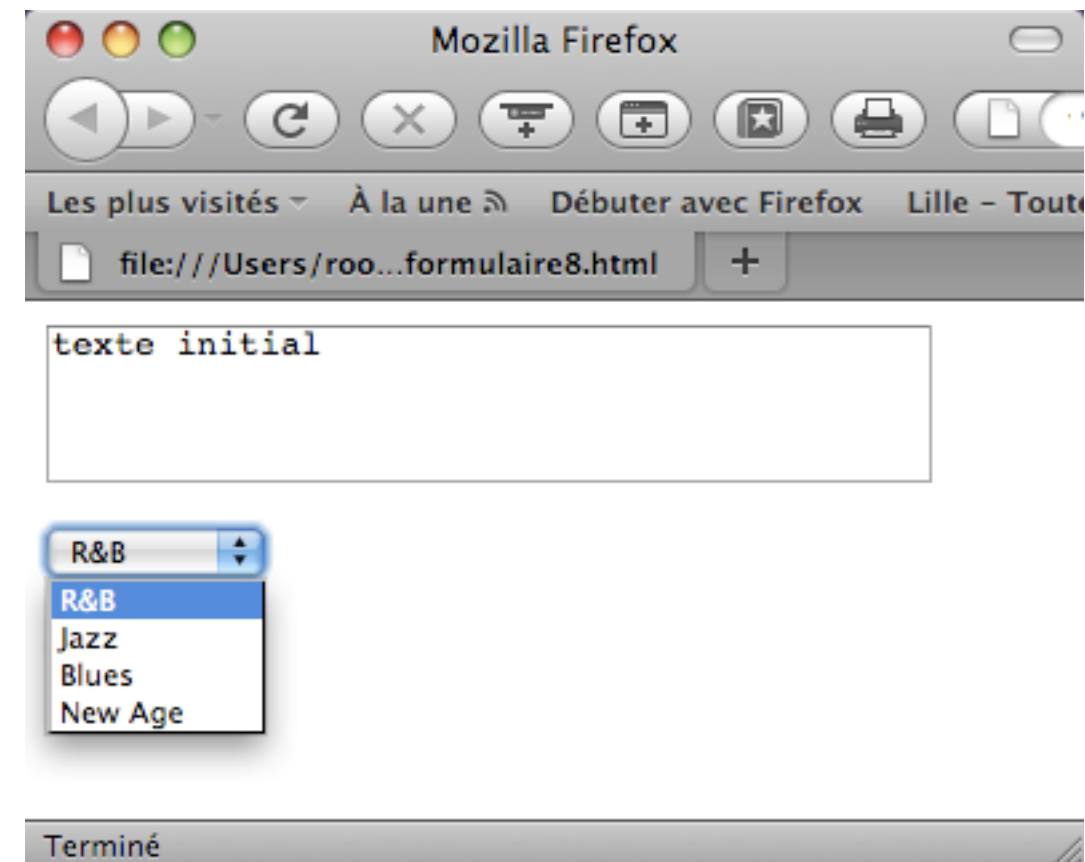
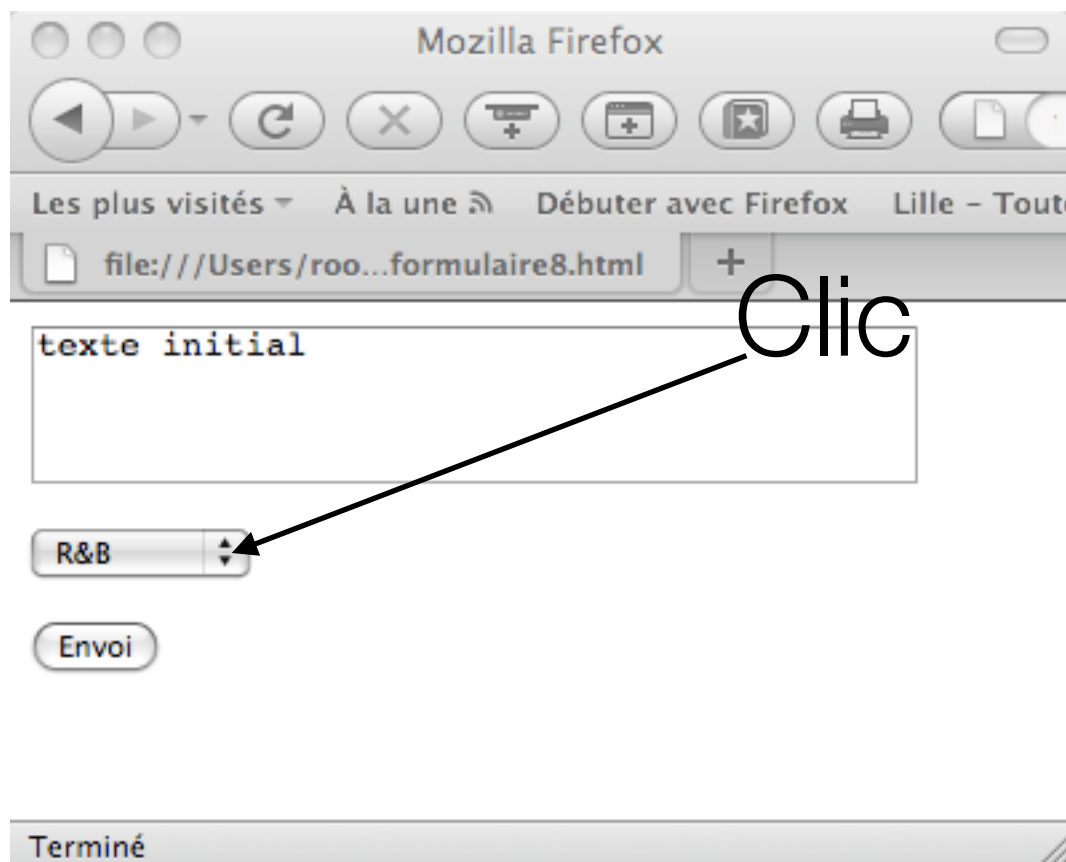


Formulaires : autres balise possibles

- 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 balise possibles



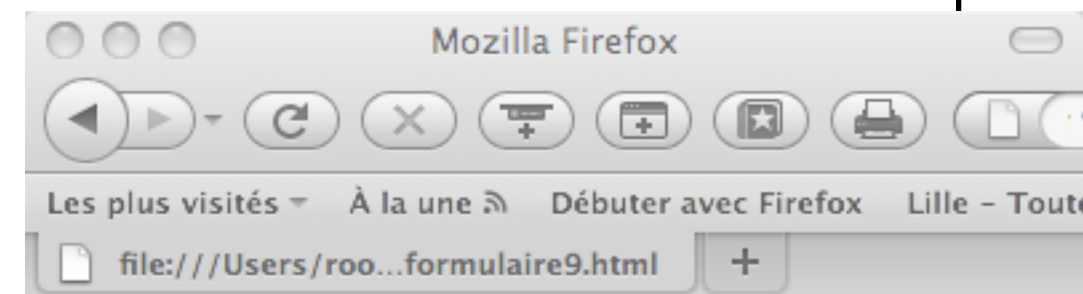
Formulaires : autres balise possibles

- `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`

Terminé

HTTP : HyperText Transfert Protocol

protocole réseau d'échange de l'information sur le Web
principe de base :

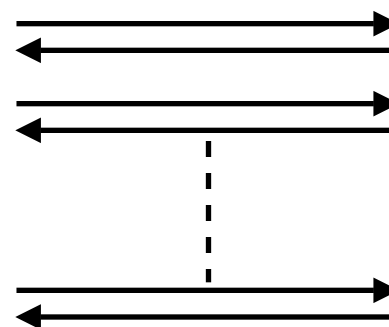
- un couple requête/réponse par document à charger
- et autant de requête/réponse que d'éléments inclus (images, sons, ...)

Les éléments inclus sont désignés par des URLs et peuvent être localisés

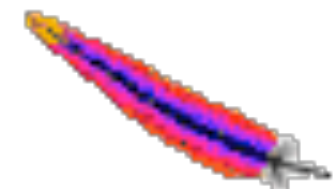
- sur le même site que le document et dans le même répertoire
- sur le même site mais dans un répertoire différent
- sur un site différent



client



serveur



HTTP

protocole applicatif au dessus de TCP : garantie d'un transport fiable
protocole ASCII

- le client initie la connexion TCP vers le serveur sur le port 80
- le serveur accepte la connexion TCP venant du client
- des messages http sont échangés entre navigateur et serveur Web
- la connexion TCP est fermée
- pas de notion de connexion HTTP
- http est “sans état” : le serveur ne maintient aucune information sur les requêtes précédentes du client
 - les protocoles qui gèrent un état sont complexes!
 - l'histoire passée (état) doit être maintenue
 - si le client/serveur tombe, leurs vues de l'état peuvent être incohérentes, et doivent donc être réconciliées

HTTP : exemple

Supposons qu'un utilisateur demande l'URL

`http://www.UneUniversite.edu/unDepartement/home.html` qui contient du texte et 10 images

1a le client HTTP initie la connexion TCP vers le serveur HTTP, port 80 de `www.UneUniversite.edu`

1b le serveur `www.UneUniversite.edu`, en attente de connexion, accepte la demande du client et le notifie

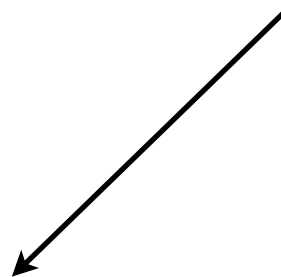
2 le client http envoie une requête http (contenant l'URL) sur la socket de connexion TCP

3 le serveur http reçoit la requête, construit un message de réponse contenant l'objet demandé, envoie le message sur la socket

HTTP : exemple

4 le serveur HTTP ferme la connexion TCP

5 le client http reçoit le message de réponse contenant le fichier HTML
Il l'analyse et affiche le HTML. Il trouve 10 objets image à télécharger



6 les étapes 1 à 5 sont répétées pour les 10 objets image

HTTP : connexion TCP

non persistante en HTTP 1.0

- le serveur traite la requête, répond et ferme la connexion TCP
- 2 échanges pour retrouver un objet :
 - établissement de la connexion TCP
 - échange requête/réponse
- chaque transfert souffre du ralentissement dû à l'ouverture de connexion TCP
- beaucoup de navigateurs ouvrent de multiples connexions en parallèle

persistante en HTTP 1.1 : les connexions TCP restent ouvertes

- pendant x secondes (configuration du serveur)
- les éléments d'un document (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 : format du message de requête

- deux types de messages : requête/réponse

ligne de requête (commande GET,POST,HEAD)

GET /somedir/page.html HTTP/1.0

User-agent: Mozilla/4.0

Accept: text/html, image/gif, image/jpeg

Accept-language: fr

lignes d'en-tête

(extra carriage return, line feed)

CR/LF indique la fin du message

HTTP : format du message de requête

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

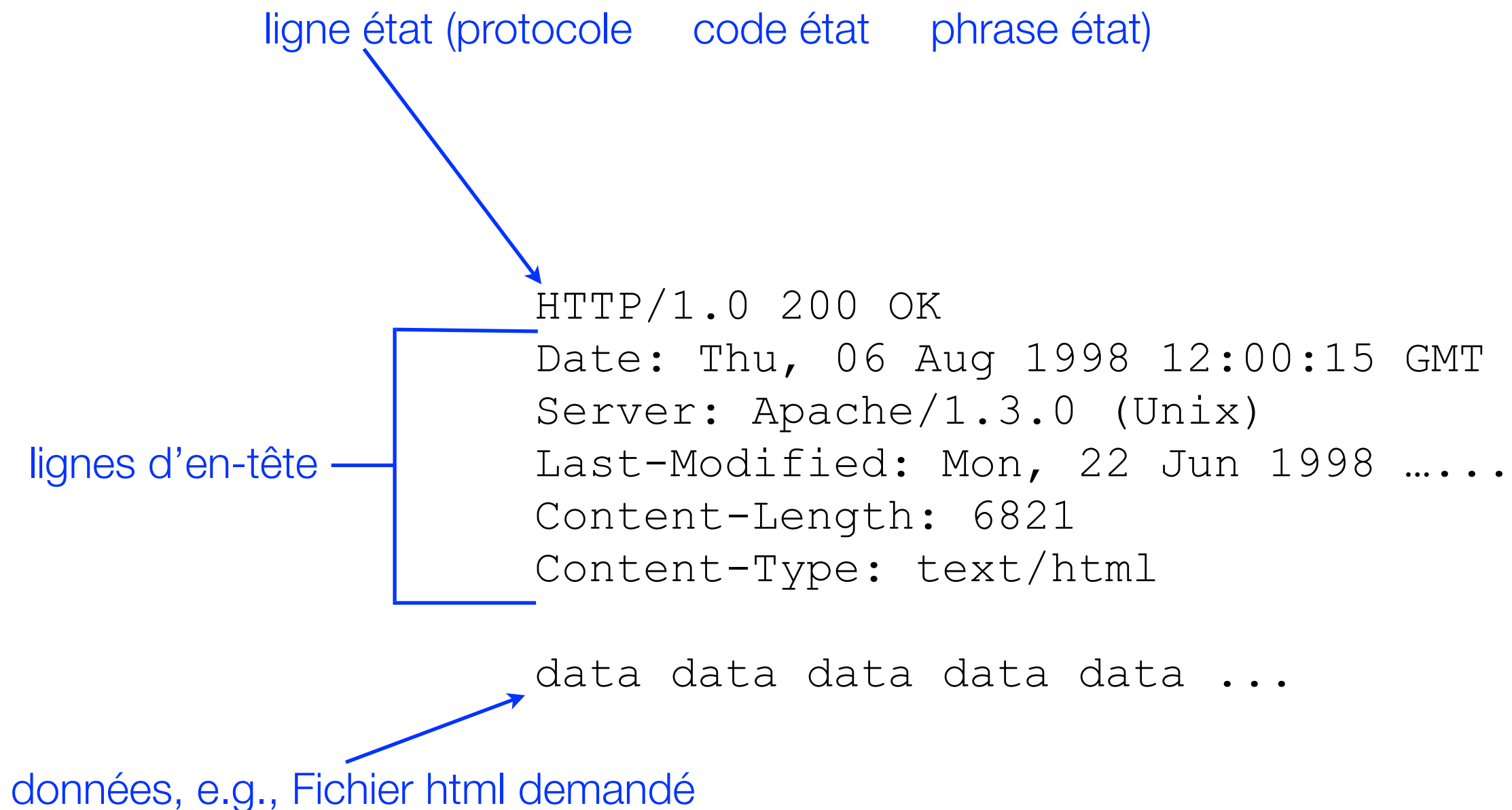
- GET demande d'un document
- HEAD demande de l'en-tête (HTTP) d'un document
- POST dépose d'information sur le serveur
- GET : requête principale, 80 % des requêtes Web
- HEAD : au travers de l'en-tête, permet de savoir si un document a changé
- POST : envoi d'informations saisies dans un formulaire client....

HTTP : format du message de requête

En-têtes client

- informations sur le client `From, Host, User-Agent`
 - information sur la page contenant le lien `Referer`
 - login, 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` `version`

HTTP : format du message de réponse



HTTP : format du message de réponse

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

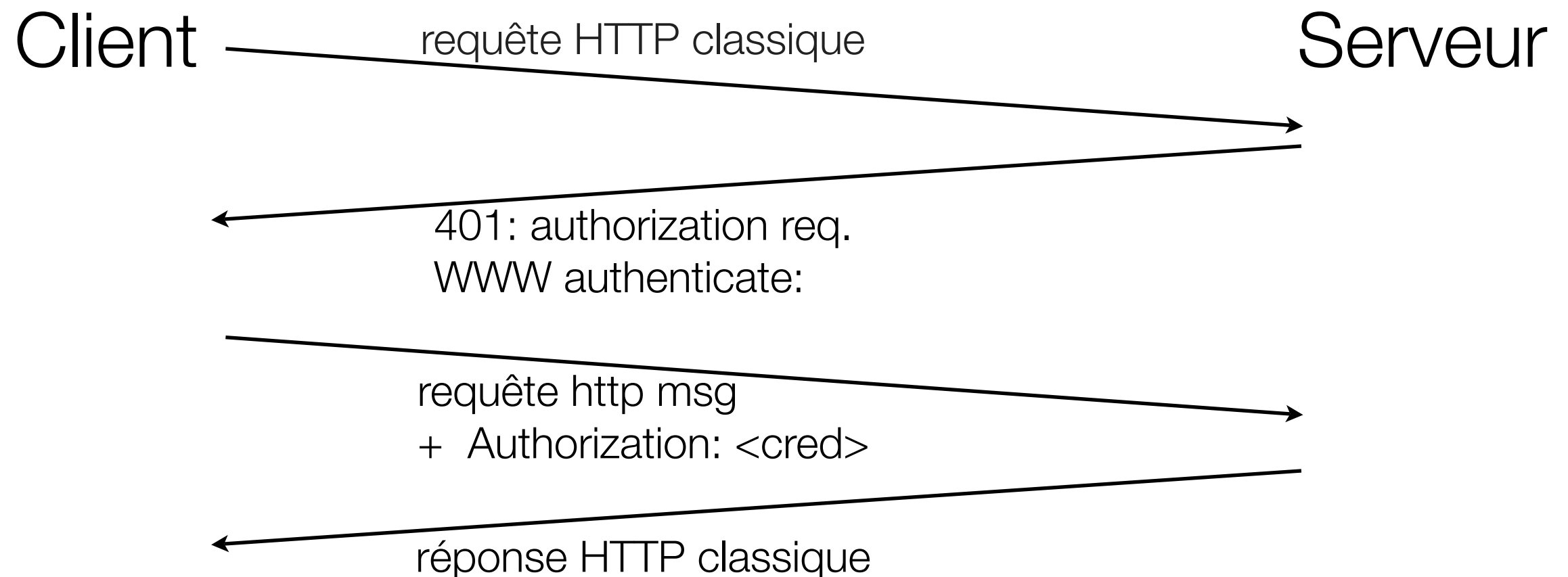
- 200 : ok
- 404 : document inconnu
- 401 : authentification nécessaire
- 500 : erreur du serveur HTTP dans le traitement requête (servlet, PHP, ...)
- 503 : serveur temporairement surchargé
- ...

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 : authentication

- Authentification : contrôle d'accès au contenu du serveur
- Les crédits d'autorisation : noms, mots de passe
- Sans état: le client doit présenter son autorisation à chaque requête
- autorisation: lignes d'en-tête dans chaque requête
- Si non autorisation: le serveur refuse l'accès, et envoie WWW authenticate: Ligne d'entête dans la réponse



HTTP : cookies garder un état

généré par le serveur, # à rappeler au serveur, utilisé plus tard pour:

- authentification
- se souvenir des préférences utilisateurs, de ses choix
- le serveur envoie le “cookie” au client dans la réponse

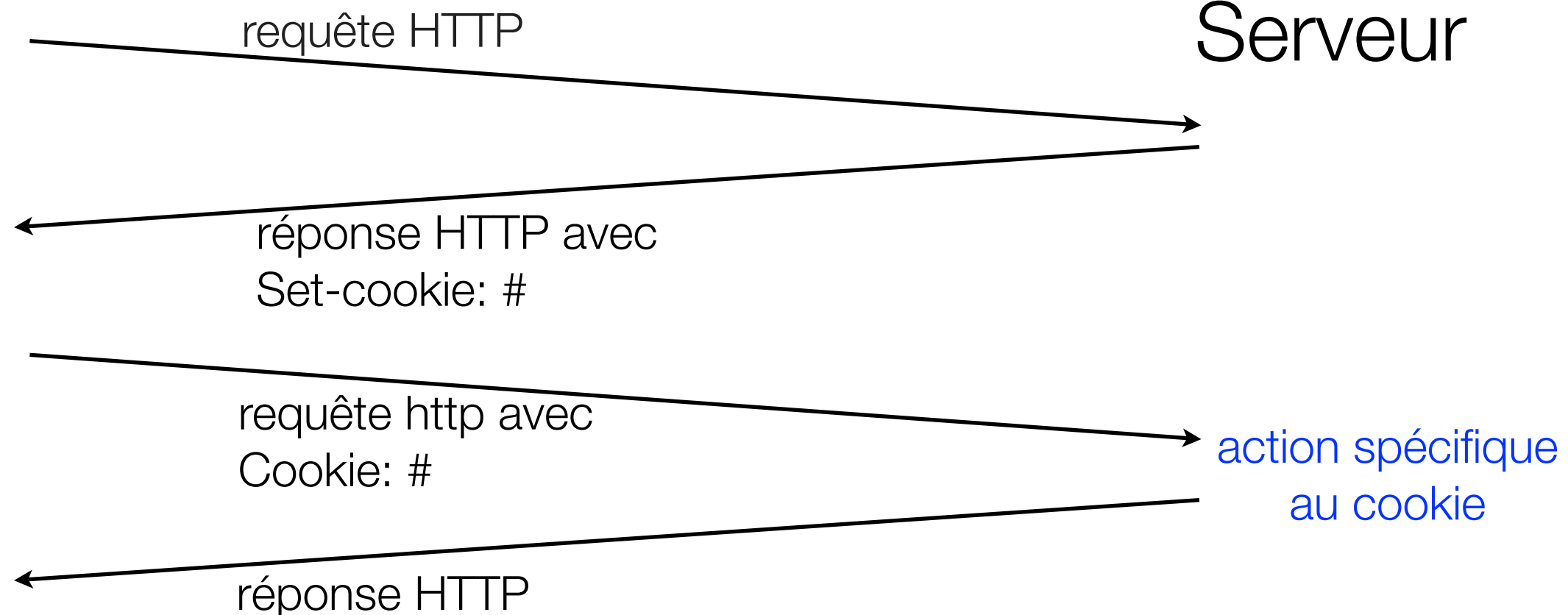
`Set-cookie: 1678453`

- Le client présente le cookie dans ses futures requêtes

`cookie: 1678453`

Client

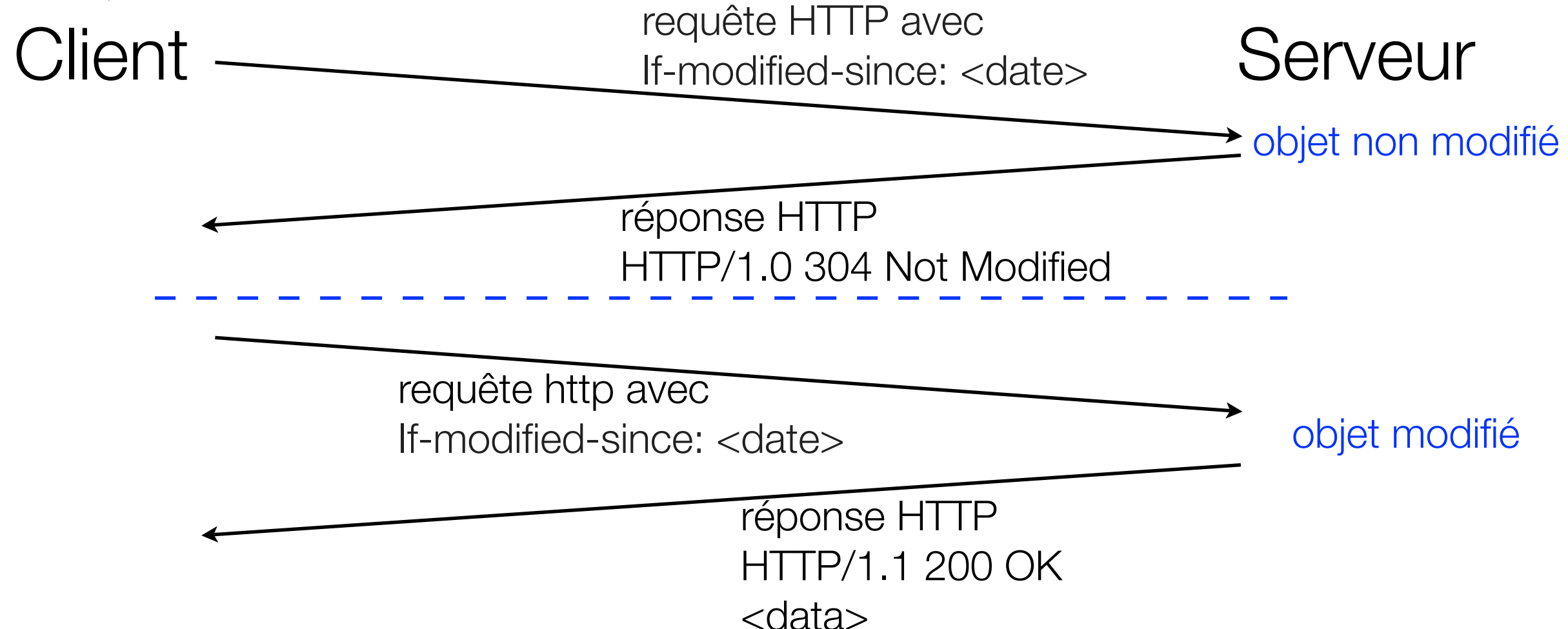
Serveur



HTTP : GET conditionnel, un cache côté client

Objectif: ne pas envoyer d'objet si le client a une version à jour dans son cache

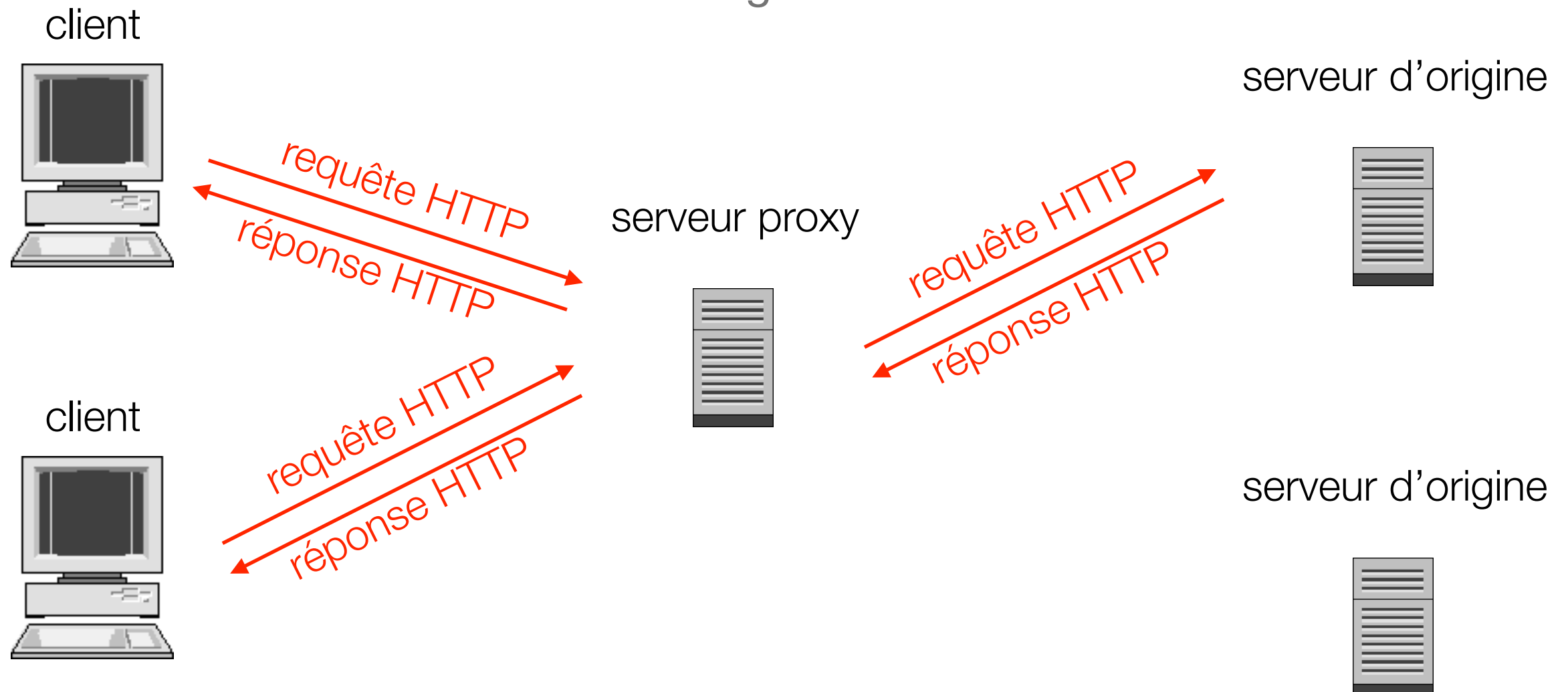
- le client donne la date de la copie cachée dans la requête
`If-modified-since: <date>`
- la réponse du serveur ne contient aucun objet si la copie cachée est à jour
`HTTP/1.0 304 Not Modified`



HTTP : Caches Web (serveurs proxy)

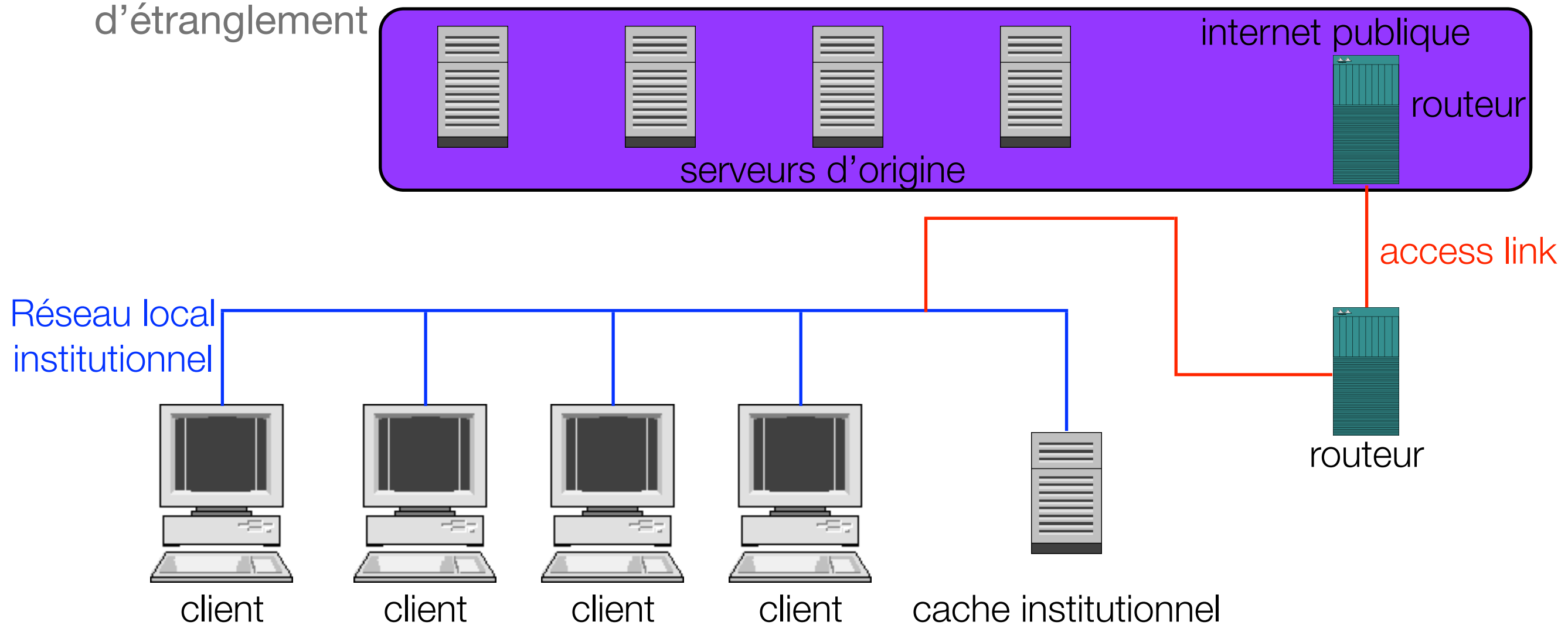
Objectif: satisfaire la requête du client sans interroger le serveur d'origine

- l'utilisateur paramètre son navigateur pour accéder au web via le proxy
- le client envoie toutes les requêtes HTTP vers le cache web
- le cache web retourne l'objet s'il le possède
- sinon il le demande au serveur d'origine et le retourne au client



HTTP : Caches Web (serveurs proxy)

- le cache est prêt du client (e.g., sur le même réseau)
- le temps de réponse est donc plus petit
- le trafic vers les serveurs distants diminue
- le lien entre le réseau local et le reste du monde est souvent un goulot d'étranglement



HTTP : évolution

- depuis janvier 1997 (RFC 2616) HTTP 1.1
autre apport que les connexions persistantes : 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-NG (New Generation) : tentative avortée d'évolution de HTTP
 - plusieurs types de protocoles en fonction des données (audio, vidéo, HTML, ...)
 - RPC (plutôt que TCP), ASN.1
 - authentification des clients

CGI : Common Gateway Interface

- protocole entre un serveur HTTP et des programmes à lancer côté serveur
- les programmes peuvent être
 - interprétés : shells scripts sh, ksh, Perl, PHP, Python, ...
 - compilés : programmes C, C++, Ada, etc ...
- Exemples d'utilisation :
 - traitement des données saisies dans un formulaire
 - génération automatique de pages Web
 - ...

CGI : fonctionnement

- Les scripts sont désignés par une URL (comme les documents HTML)
- **par exemple** : `http://www.lifl.fr/cgi-bin/anniversaire.pl`
- en fonction du chemin d'accès, le serveur sait si l'URL correspond à un script ou à un document
- par convention, tous les fichiers dans le répertoire `cgi-bin/` sont des scripts

CGI : fonctionnement

Etape 1 : client -> serveur

2 méthodes peuvent être employées : GET ou POST

- GET /cgi-bin/anniversaire.pl?mois=aout&jour=11 HTTP/1.0
les options à passer au script sont encodées après le ? et sont facultatives
- POST /cgi-bin/anniversaire.pl HTTP/1.0
Content-type: application/x-www-form-urlencoded
Content-length: 20
mois=aout&jour=11

dans ce cas (formulaire), c'est le navigateur qui construit cette requête

CGI : fonctionnement

Etape 2 : serveur -> script

les informations à passer aux scripts CGI par le serveur HTTP le sont

- soit par des variables d'environnement
- soit sur l'entrée standard du script

les principales variables d'environnement

- `CONTENT_TYPE`, `CONTENT_LENGTH`, ... (champs de l'en-tête du message)
- `SCRIPT_NAME` (ici `/cgi-bin/anniversaire.pl`)

les options d'appel de script (ici `mois=aout&jour=11`) sont transmises

- par la requête d'environnement `QUERY_STRING` pour la méthode `GET`
- sur l'entrée standard du script pour la méthode `POST`

Le script analyse ses variables (et l'entrée standard éventuellement), exécute son traitement et produit un résultat

CGI : fonctionnement

Etape 3 : script -> serveur

- la sortie standard du script est un document de la forme
 en-tête
 ligne blanche
 données
- 3 en-têtes possibles
 Content-type: ... (par ex text/html ou image/gif)
 Location: ... (par ex <http://www.truc.com>) : URL à transmettre au client
 Status: ... (par ex 403 Forbidden) : code (d'erreur) à renvoyer au client

Etape 4 : serveur -> client

Le document est envoyé au navigateur Web qui l'interprète.

CGI : exemple

Shell script sh

```
#!/bin/sh
echo "Content-type: text/html"
echo
echo "<!DOCTYPE HTML PUBLIC\"-//W3C/DTD HTML 3.2 Final/EN\">"
echo "<HTML>"
echo "<BODY>"
echo "Nous sommes le <B>`date`</B>"
echo "</BODY>"
echo "</HTML>"
```

CGI : exemple

Langage C

```
#include <stdio.h>
int main() {
    char c;
    int nb_args;
    printf("Content-type: text/html\n");
    printf("\n" );
    printf("<!DOCTYPE HTML PUBLIC\"-//W3C/DTD HTML 3.2 Final/EN\">\n");
    printf( "<HTML>\n" );
    printf( "<BODY>\n" );
    nb_args = 0;
    while ((c=getchar()) != EOF) if ( c == '&' ) nb_args++;
    if ( nb_args != 0 ) nb_args++;
    printf( "Il y a %d arguments a votre requete\n" , nb_args );
    printf( "</BODY>\n" );
    printf( "</HTML>\n" );
    return 0;
}
```

Servlets

Programme Java s'exécutant côté serveur Web

- servlet : programme "autonome" stocké dans un fichier `.class` sur le serveur
- jsp : programme source Java embarqué dans une page html

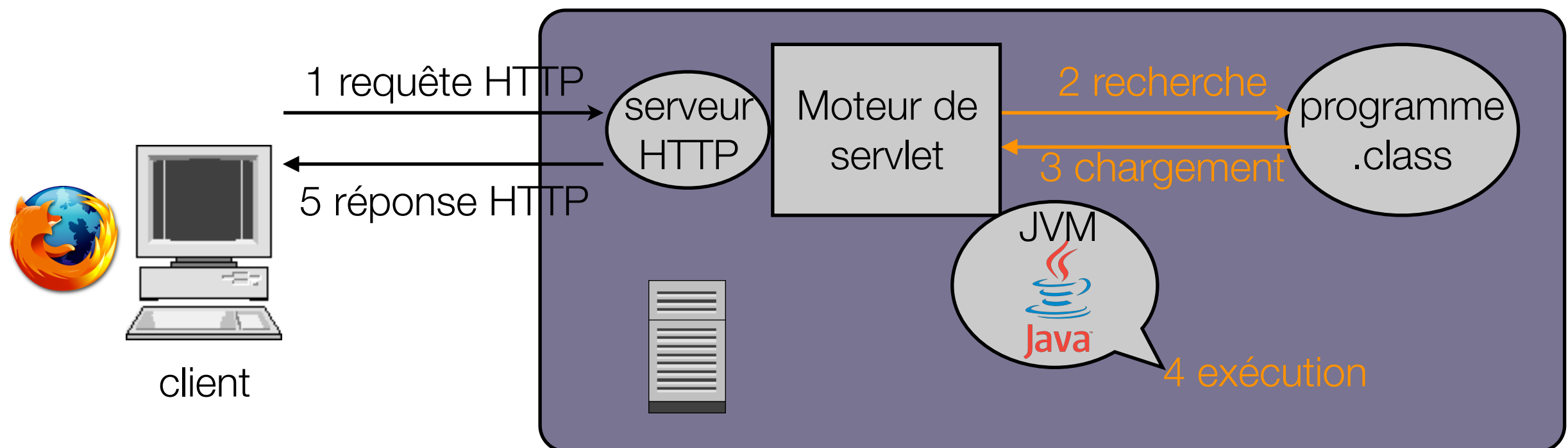
	côté client	côté serveur
<code>.class</code> autonome	applet	servlet
embarqué dans le html	JavaScript	JSP

Servlet et JSP

- exécutables 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

Servlets : principe

- les fichiers de bytecode (`.class`) sont stockés sur le serveur
- ils sont désignés par une URL : `http://www.lifl.fr/servlet/Prog`
- le chargement de l'URL provoque l'exécution de la servlet
 - les servlets étendent le comportement du serveur Web
 - elles sont exécutées par un "moteur" (ex. Tomcat)



Servlets : principe

Mozilla Firefox

Les plus visités ▾ À la une 📡 Débuter avec Firefox Lille - Toute l'actu...

file:///Users/roo...formulaire4.html +

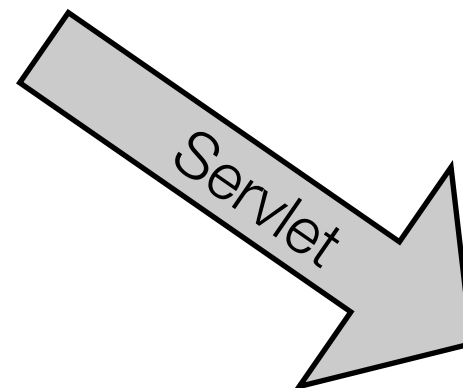
Nom Roos

Prénom JF

Envoi Remise à zéro

Terminé

Clic



Mozilla Firefox

Les plus visités ▾ À la une 📡 Débuter avec Firefox

Exemple de résultat

Bonjour JF Roos.

Terminé

Servlets : développement

- 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 une thread
- le code produit un résultat qui est envoyé au client
- en cas d'erreur dans le code Java de la servlet, le message est récupéré dans le navigateur

Servlets : développement

Utilisation des packages `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
renseignée automatiquement par le "moteur"

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

Servlets : développement

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 écrit le code HTML sur ce flux de sortie

Servlets : développement

premier exemple de servlet

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

imposé

```
public class HelloServlet extends HttpServlet {  
    public void service(ServletRequest request,  
                        ServletResponse response )  
        throws ServletException, IOException
```

imposé par l'API servlet

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

le résultat est en HTML

```
    PrintWriter out =response.getWriter();
```

recupère un flux pour
générer le résultat

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

```
}}
```

génération du code
HTML

Servlets : développement

premier exemple de servlet

- Compilation :
 - `HelloServlet.Class` installé dans l'arborescence de Tomcat
- Chargement via une URL de type `http://.../servlet/HelloServlet`
 - exécution de `HelloServlet.class`



Servlets : développement

deuxième exemple de servlet

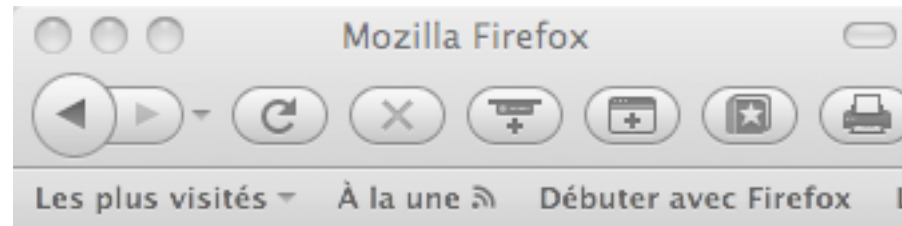
- Une servlet n'est instanciée qu'une seule fois :
persistance de ses données entre 2 invocations

```
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>«  ");  
    }  
}
```


Servlets : développement

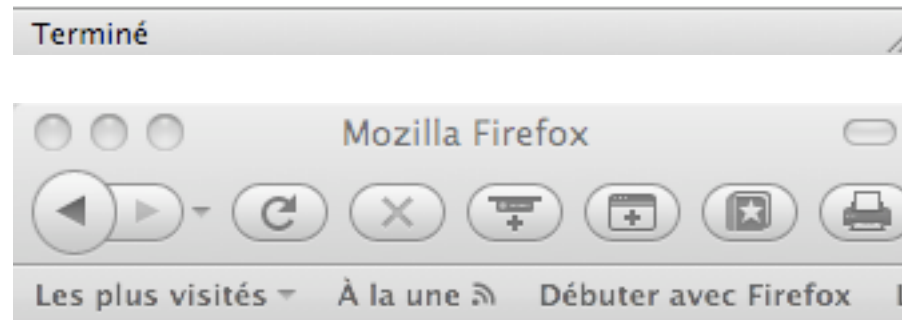
deuxième exemple de servlet

première invocation



0

deuxième invocation



1



Servlets : développement 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

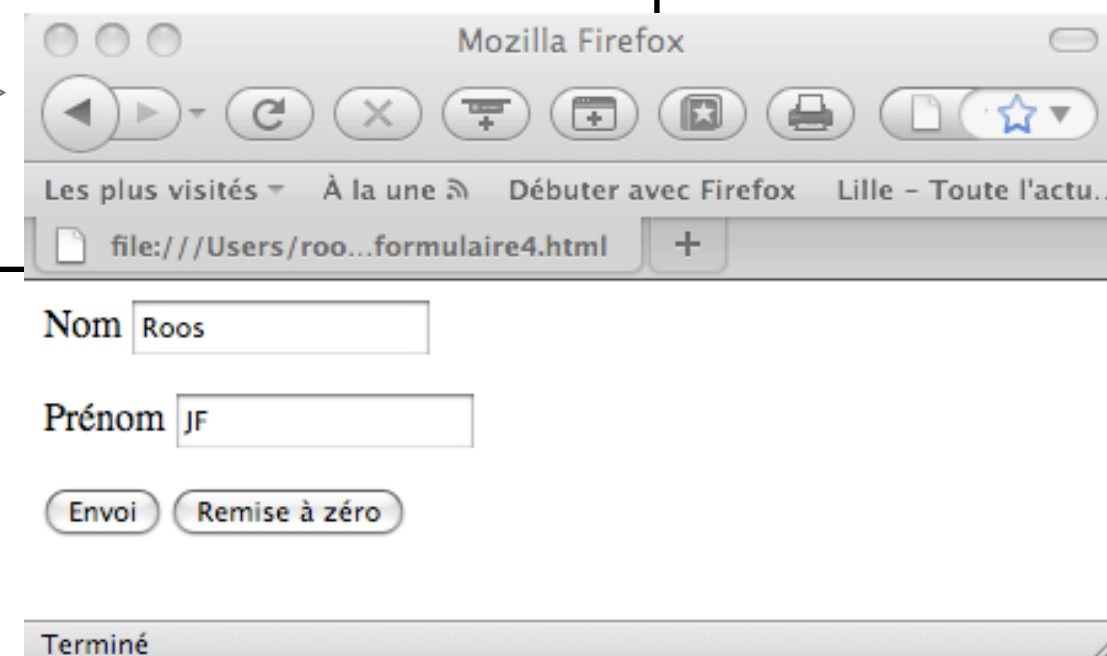
```
<HTML> <BODY>

<FORM ACTION="url de la servlet" METHOD=POST>

Nom <INPUT NAME="nom" SIZE=15> <P>
Prénom <INPUT NAME="prenom" SIZE=15> <P>

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

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



Servlets : développement

récupération des données d'un formulaire

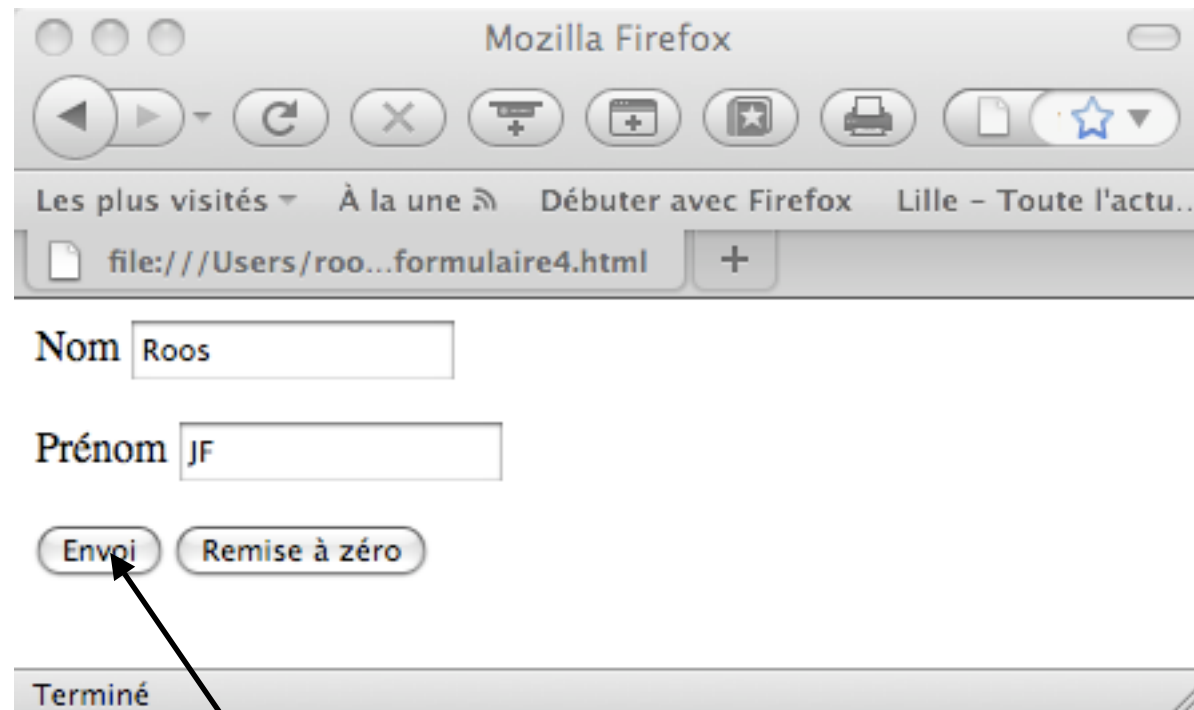
```
public class FormulaireServlet extends HttpServlet {

    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>");
    }
}
```

Servlets : développement récupération des données d'un formulaire



Mozilla Firefox

Les plus visités ▾ À la une 📶 Débuter avec Firefox Lille - Toute l'actu...

file:///Users/roo...formulaire4.html +

Nom Roos

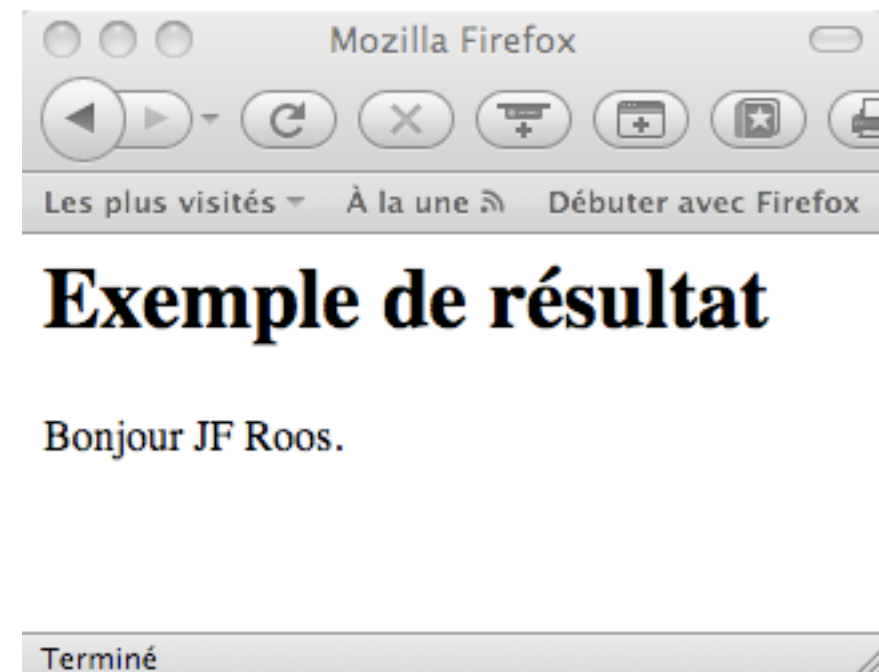
Prénom JF

Envoi Remise à zéro

Terminé

Clic

Servlet



Mozilla Firefox

Les plus visités ▾ À la une 📶 Débuter avec Firefox

Exemple de résultat

Bonjour JF Roos.

Terminé

Servlets : développement

différenciation des méthodes HTTP

- `service()` traite toutes les requêtes HTTP
- possibilité de différencier les traitements en fonction de la commande HTTP
POST, GET, ...
`doGet()` `doHead()` `doPost()` `doPut()` `doDelete()` `doTrace()`
- les méthodes `doXXX()` ont le même profil/fonctionnement que `service()`

```
public class FormulaireServlet extends HttpServlet {  
  
    public void doGet( HttpServletRequest req,HttpServletResponse resp)  
                    throws ServletException, IOException {  
        // ... le traitement lorsque la servlet est invoquée avec GET }  
  
    public void doPost( HttpServletRequest req,HttpServletResponse resp)  
                    throws ServletException, IOException { /* idem POST */ }  
    // ... éventuellement autres méthodes doXXX  
}
```

Servlets : développement

cycle de vie d'une servlet

Une servlet peut définir les méthodes `init()` et `destroy()`.

- `void init(ServletConfig conf)`
méthode appelée par le moteur au démarrage de la servlet
 - peut contenir le code d'initialisation de la servlet
 - \approx constructeur pour la servlet
 - méthode appelée par le moteur lors de l'installation de la servlet
- `void destroy()`
méthode appelée lors de la destruction de la servlet
 - lors de l'arrêt du moteur
 - ou lors du déchargement de la servlet
 - peut-être appelée pour arrêter la servlet

Servlets : développement

types de contenu générés par une servlet

- 80% du temps, HTML
- mais peut être n'importe quel type de contenu : GIF, PDF, DOC, ...
- le type MIME du contenu est précisé par `resp.setContentType("...")`
quelques types MIME courants
 - text/html
 - image/gif
 - video/mpeg
 - audio/mp3
 - application/pdf
 - application/octet-stream : un fichier binaire quelconque

Servlets : développement

servlet retournant le contenu d'un fichier binaire

```
public class FichierServlet extends HttpServlet {
    public void service( HttpServletRequest req,
                        HttpServletResponse resp )
                        throws ServletException, IOException {
        resp.setContentType("application/octet-stream");
        resp.setHeader( // facultatif
            "Content-Disposition", // fournit le nom du fichier
            "attachment;filename=monfichier.ext"); // au navigateur
        OutputStream os = resp.getOutputStream();
        File f = new File("monfichier.ext");
        byte [] content = new byte[f.length()];
        FileInputStream fis = new FileInputStream(f);
        fis.read(content);
        fis.close();
        os.write(content);
    }
}
```


Servlets : 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`

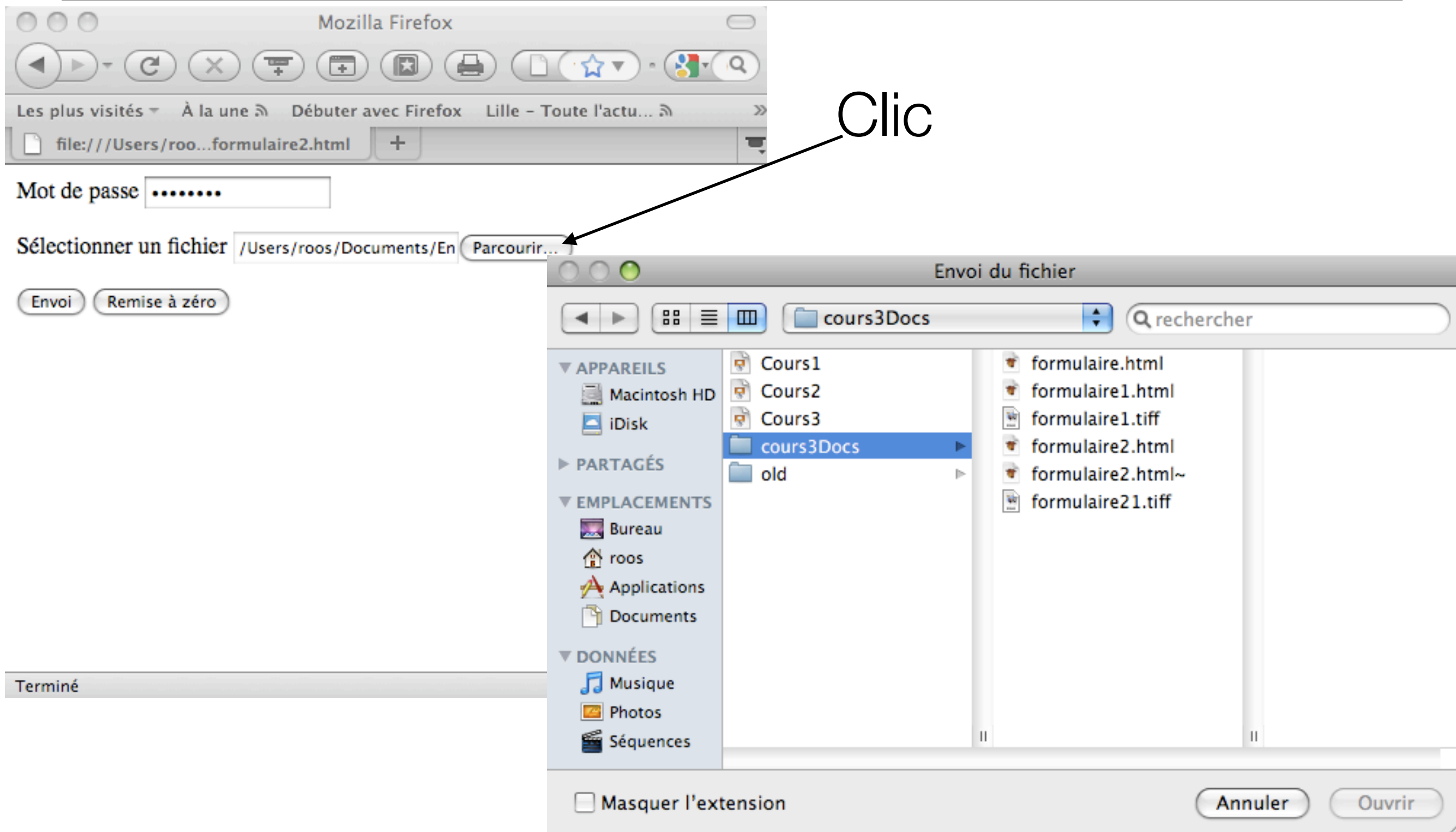
- `HttpSession session = request.getSession(true)`
retourne la session courante pour cet utilisateur ou une nouvelle session
- `HttpSession session = request.getSession(false)`
retourne la session courante pour cet utilisateur ou `null`

Servlets : suivi de session

méthodes d'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)`

Servlets : upload récupération de fichier à partir d'un formulaire



Servlets : upload

définition des formulaires avec upload

```
<HTML> <BODY>

<FORM ACTION="url de la servlet" METHOD=POST
      ENCTYPE="multipart/form-data">

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

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

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

Servlets : upload encodage fichiers joints

```
-----7d225420d803c8
Content-Disposition: form-data; name="fichier";
filename="..."
Content-Type: image/gif
GIF89a& ... contenu binaire du fichier ...
-----7d225420d803c8--
```

- séparateur déterminé aléatoirement à chaque upload par le navigateur
- précisé dans les en-têtes HTTP de la requête

```
Content-Type: multipart/form-data;
boundary=-----7d225420d803c8
```

- format défini par la RFC 1867 de l'IETF
- voir <http://www.ietf.org/rfc/rfc1867.txt>

Servlets : upload

récupération de fichier à partir d'un formulaire

- récupération du flux binaire, programmer le décodage :(
- **librairie existante** Commons FileUpload du projet Apache
<http://commons.apache.org/fileupload/>
- exemple d'utilisation (version 1.2.1)

```
FileItemFactory factory = new DiskFileItemFactory();
ServletFileUpload upload = new ServletFileUpload(factory);

List /* FileItem */ items = upload.parseRequest(request);

for( Iterator i = items.iterator() ; i.hasNext() ; ) {
    FileItem fi = i.next();
    File monFichier = new File("...");
    fi.write(monFichier);
}
```

Servlets : chaînage des servlets

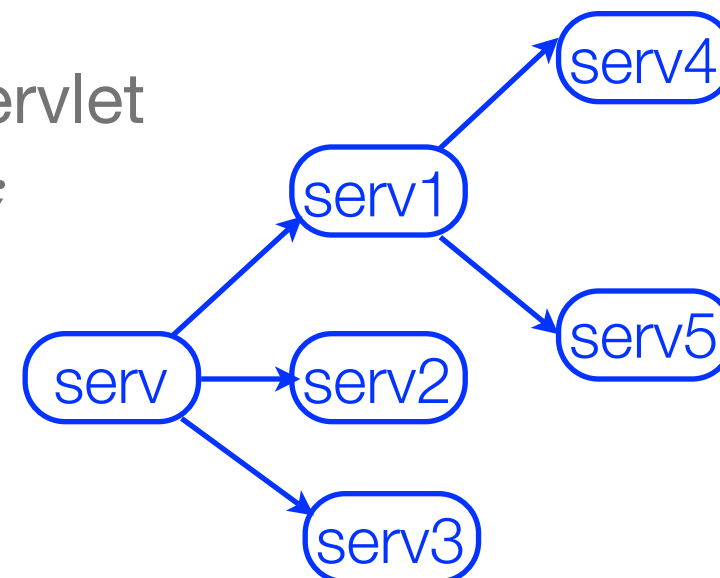
- agrégation des résultats fournis par plusieurs servlets
 - meilleure modularité
 - meilleure réutilisation

Utilisation d'un `RequestDispatcher` obtenu via un objet `request`

```
RequestDispatcher rd= request.getRequestDispatcher("serv1");
```

- inclusion du résultat d'une autre servlet
`rd.include(request, response);`

- délégation du traitement à une autre servlet
`rd.forward(request, response);`



URL

Servlets : 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 `SingleThreadModel`

```
public class SynchroServlet extends HttpServlet
    implements SingleThreadModel {

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


Servlets : partage de données entre servlets

Notion de contexte d'exécution :

ensemble de couples (name,value) partagés par toutes les servlets instanciées, donc partage de données entre tous les clients

```
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

Servlets : déploiement

archivage, format de fichier `.war`

Problématique : comment diffuser une application à base de servlets ?

- souvent plusieurs servlet (fichiers `.class`)
- des ressources additionnelles (`.gif`, `.jpeg`, `.html`, `.xml`, ...)

Solution

- monde Java : archive `.jar` pour la diffusion de programmes
- fichier `.war` = `.jar` pour les servlets
diffusion d'un seul fichier prêt à l'emploi
- fichiers `.war` se manipulent (création, extraction, ...) avec la commande `jar`
- ex. :
 - `jar cf app.war index.html WEB-INF/classes/*` **création**
 - `jar tf app.war` **affiche le contenu**
 - `jar xf app.war` **extraction**

Servlets : déploiement

descripteur de déploiement web.xml

Chaque archive `.war` doit être accompagnée d'un fichier `web.xml` décrivant les servlets incluses dans l'archive

- 2 balises principales : `<servlet>` et `<servlet-mapping>`

```
<web-app>
  <servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>mypackage>HelloServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/version/beta/Hello</url-pattern>
  </servlet-mapping>
</web-app>
```

Servlets : déploiement

descripteur de déploiement `web.xml`

- une balise `<servlet>` par servlet
 - un nom et une classe par servlet
 - le fichier `.class` de la servlet doit être stocké dans `WEB-INF/classes`
 - éventuellement sous-répertoires correspondant aux packages
 - ex. : `WEB-INF/classes/mypackage/HelloServlet.class`
- une balise `<servlet-mapping>` par servlet
 - un nom correspondant à une balise `<servlet>` existante
 - une URL relative permettant d'accéder à la servlet
- plusieurs autres balises peuvent être utilisées
voir `http://java.sun.com/j2ee/dtds/web-app_2_2.dtd`

Servlets : déploiement

descripteur de déploiement `web.xml`

Paramètres d'initialisation

- possibilité d'inclure des paramètres d'initialisation de la servlet dans `web.xml`
avantage : peuvent être changés sans avoir à recompiler

```
<servlet>
  <servlet-name>HelloServlet</servlet-name>
  <servlet-class>mypackage.HelloServlet</servlet-class>
  <init-param>
    <param-name>nom</param-name>
    <param-value>valeur</param-value>
  </init-param>
</servlet>
```

- dans le code de la servlet (par ex. méthode `init`) sur l'objet `ServletConfig`
`String valeur= config.getInitParameter("nom");`

Servlets : déploiement

installation d'une archive .war dans Tomcat

dans le répertoire `<tomcat_root>/webapps`

`webapps`

`| -> myapp.war`

`| -> myapp`

`| -> index.html`

`| -> WEB-INF`

`| -> web.xml`

`| -> classes`

`| -> mypackage`

`| -> HelloServlet.class`

- URL pour accéder à la servlet

`http://machine.com:8080/myapp/version/beta/Hello`

dépend de la balise `<url-pattern>` fournie dans `web.xml`

Servlets : moteurs

Moteur de servlet (servlet engine)

parfois aussi appelé conteneur de servlet

- logiciel servant à exécuter des servlets
- les servlets ne sont pas des programmes autonomes (pas de `main`)
doivent être pris en charge par un moteur pour être exécutées
- **Tomcat** <http://tomcat.apache.org>
- **Jetty** <http://jetty.mortbay.com>
- **Resin** <http://www.caucho.com>

Servlets : moteurs

Tomcat



- le plus connu, le plus utilisé
- logiciel écrit 100% en Java
- inclut un serveur HTTP
- par défaut port écoute sur le port 8080
- il peut s'utiliser
 - en mode standalone: joue le rôle du serveur HTTP + moteur servlet
 - couplé avec le serveur Web Apache
- diffusé par le consortium Apache
- peut exécuter aussi des JSP
- souvent inclus dans d'autres logiciels
- ex. : serveur Java EE (JBoss, JOnAS, ...)

Servlets : conclusion

les servlets permettent de développer des applications Web en Java

résumé des fonctionnalités

- traitement des données fournies par les utilisateurs
- gestion de session
- gestion de cookies
- format d'archivage `.war`

Comparaison

- avec une applet : servlet s'exécute côté serveur
- avec JSP : côté serveur aussi mais
 - servlet : classe Java, facilité d'écriture traitement ++ / HTML --
 - JSP : fichier HTML, facilité d'écriture traitement - / HTML ++