

# Distribution d'applications via le WEB

Documents actifs, applets  
& Scripts cotés serveur, Servlet

# Introduction à la notion de documents actifs

Produire des documents en fonction de différentes données.

## Objectif :

- personnaliser les documents en fonction des utilisateurs ;
- présenter des données extraites d'une requête dans une base de donnée ;
- Déclancher des opérations à distance et obtenir des résultats ;
- Afficher des comptes rendus de calculs, de mesure en temps réel.

## Source de données :

- documents construit en fonction des paramètres d'une requête ;
- documents construit en fonction d'une base de données ;
- documents construit en fonction d'un outil de mesure externe ;
- documents construit en fonction de résultats externes au système.

# Formulaires HTML

Objectif : permettre à l'utilisateur d'envoyer des informations vers le serveur.

Exemple :

- Chaîne de caractère,
- nombres,
- choix parmi une liste,
- cases à cocher,
- fichiers,
- choix d'un point sur une image...

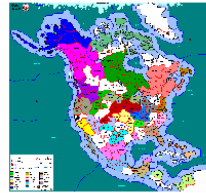
Transfert des informations saisies via le bouton « soumettre ».

2 méthodes de transfert :  
GET & POST

Champs de saisie de texte :

Choix parmi une liste :

Case à cocher :      Liste de choix :  
Case 1 : ☐      Choix 1- ☐  
Case 2 : ☒      Choix 2- ☐  
Case 3 : ☐      Choix 3- ☒



Bouton :

# Déclaration d'un formulaire en HTML

## Déclaration du formulaire :

```
<form name="form2" method="get" action="http://anURL">  
  Corps du formulaire  
</form>  
<form name="form1" method="post" action="http://anURL"  
  enctype="multipart/form-data">  
  Corps du formulaire  
</form>
```

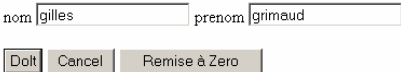
Method="get" : Les arguments fournis par l'utilisateur sont passer directement dans l'URL qui suit la commande GET (la taille des arguments doit rester modeste)

Method="post" : Les arguments fournis par l'utilisateur sont passer dans le corps de la requête, en utilisant par exemple le format MIME (cf. contenu d'un courrier électronique).

# Formatage des requêtes HTTP : via la commande GET

## Le formulaire en HTML :

```
<form name="f1" method="get" action="http://monsite.com/monprog.cgi">
  <p> nom   <input type="text" name="lenom"   value="un nom">
        prenom <input type="text" name="leprenom" value="un prenom"> </p>
  <p>
    <input type="submit" name="Submit" value="DoIt">
    <input type="submit" name="Cancel" value="Cancel">
    <input type="reset"  name="Reset"  value="Remise à Zero">
  </p>
</form>
```



## Requête envoyée au serveur sur click de soumettre :

```
GET /d?leprenom=gilles&lenom=grimaud&Submit=DoIt HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
```

# Règle de formatage d'une requête « GET »

Les arguments du formulaire commencent après le "?" Chaque argument est identifié par un couple : nomDArgument=ValueDeLArgument  
Le caractère "&" est utilisé pour séparer les arguments.

## – Pour les champs de saisie de texte :

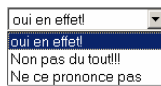
saisie de texte :

Code HTML : `<input type="text" name="arg1" value="unNom">`

Format dans la requête HTTP : `...?arg1=unNom&...`

## – Pour les listes de valeur :

Code HTML : `<select name="arg2">`



`<option value="OUI">oui OK!</option>`

`<option value="NON">Non Merci...</option>`

`<option value="BLANC">Blanc</option>`

`</select>`

Format dans la requête HTTP : `...&arg2=OUI&...`

Case 1 : ☐

Case 2 : ☒

Case 3 : ☐

## – Pour les cases à cocher :

Code HTML : `<input type="checkbox" name="Arg3" value="C1">`

Format dans la requête HTTP : `...&ChoixA=C1&...`

# Règle de formatage d'une requête « GET »

## – Pour les champs de saisie de texte :

Code HTML :

```
<input type="radio" name="Arg4" value="C1">
```

Format dans la requête HTTP :

Choix 2- ☐  
Choix 3- ☒

```
...?arg4=C1&...
```

## – Pour un point sur une image :

Code HTML :

```
<input type="image" border="0" name="arg5"
width="200" height="184" src="monImage.gif">
```

Format dans la requête HTTP :

```
...&arg5.x=110&arg5.y=13&...
```

## – Pour les boutons (autre que reset) :

Code HTML :

```
<input type="submit" name="arg6" value="DoIt">
```

Format dans la requête HTTP :

```
...&arg6=DoIt
```

## Règle de formatage d'une requête « POST »

⇒ Limitation de la commande GET (doit pouvoir être contenu dans une URL) inadapté aux requêtes volumineuses (exemple : transfert d'un fichier)

Déclarer un formulaire selon la méthode «post» :

```
<form name="f1" method="POST"
      action="http://monsite.com/monprog.cgi"
      enctype="multipart/form-data">
```

Contenu du formulaire équivalent à celui d'un formulaire GET sauf qu'il accepte les fichiers :

```
<input type="file" name="unFichier" maxlength="60">
```





# Formatage des requêtes HTTP : via la commande POST

```
POST /bidon.cgi HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Content-Type: multipart/form-data; boundary=-----7d29cf5045a
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Content-Length: 1073
```

```
-----7d29cf5045a
Content-Disposition: form-data; name="arg1"
Nom de l'utilisateur
-----7d29cf5045a
Content-Disposition: form-data; name="arg2"
OUI
-----7d29cf5045a
Content-Disposition: form-data; name="arg3"
C2
-----7d29cf5045a
Content-Disposition: form-data; name="arg4"
3
```

Champs de saisie de texte :

Choix parmi une liste :

Case à cocher :

Case 1 : ☐

Case 2 : ☒

Case 3 : ☐

Liste de choix :

Choix 1- ☐

Choix 2- ☐

Choix 3- ☒

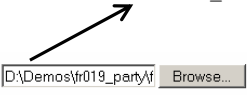
# Formatage des requêtes HTTP : via la commande POST

```
-----7d29cf5045a
Content-Disposition: form-data; name="arg5"; filename="C:\Demos\fr019_party\file
_id.diz"
Content-Type: text/plain


.farbrausch consumer consulting
fr-019: poem to a horse - party version
.64k demo production
.mekkasymposium 2002
.code      chaos .graphics    fiver2
.music     rp    .synth      kb
.packer    ryg
.wait for the final - www.farb-rausch.com
-----7d29cf5045a
Content-Disposition: form-data; name="imageField.x"

108
-----7d29cf5045a
Content-Disposition: form-data; name="imageField.y"

135
-----7d29cf5045a--
```



Fichier indiqué



# Traitement de la requête sur le Serveur

Déclancher une action, un calcul, ...

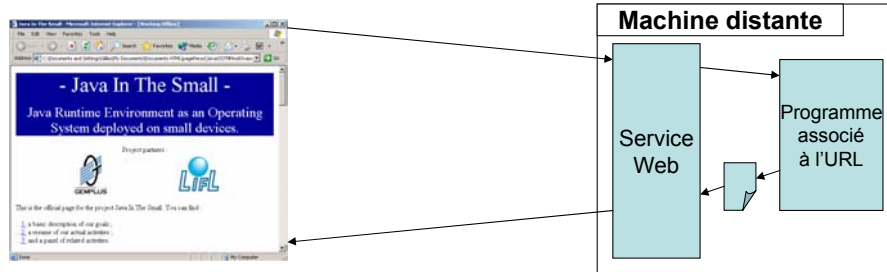
Exécuter un programme

- CGI, ISAPI ;
- Servlets ;
- Documents contenant des scripts exécuter par le serveur.

Retourner un document, produit par l'exécution de  
la requête.

- Une page HTML ;
- Une image ;
- Un renvoi vers une autre page...

# Production de documents



- Documents actifs CGI

```
#include <stdio.h>
int getNextCount();
int main(int argc, char **argv)
{ printf("<html>\r\n");
  printf(" <head> <title> mon titre </titre> </head>\n\r");
  printf(" <body> <h1> Hello World </h1> \n\r");
  printf(" <p> - %d</p> </body>\n\r",getNextCount());
  printf("</html>\n\r");
  return 1;}
```

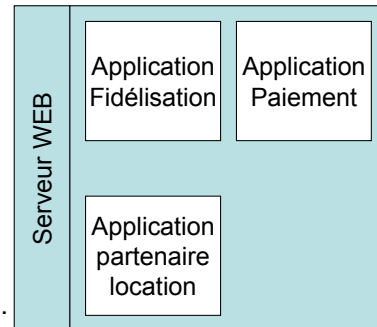
## Intégrer les applications cotés serveur dans un cadre de travail

- Dépasser le schéma applicatif :  
Un serveur Web pour une application

Intégrer les applications cotés serveur dans le Serveur Web lui-même :

Pour améliorer les performances du serveur :  
Par exemple « DLL ISAPI sous IIS »

Pour faciliter le déploiement / administration et la maintenance d'applications WEB :  
Exemple : JSP & Servlets sous TomCat Apache.



# Production de documents via des Servlets

## Déclaration d'une Servlet dans le fichier de mapping web.xml :

```
<web-app>
  <servlet>
    <servlet-name>MaServlet1</servlet-name>
    <servlet-class>coreservlets.entryClass</servlet-class>
    <init-param>
      <param-name>Message</param-name>
      <param-value>Bienvenu en ce mois de Mai</param-value>
    </init-param>
    <init-param>
      <param-name>compteur</param-name>
      <param-value>134</param-value>
    </init-param>
  </servlet>
</web-app>
```

# Production de documents via des Servlets

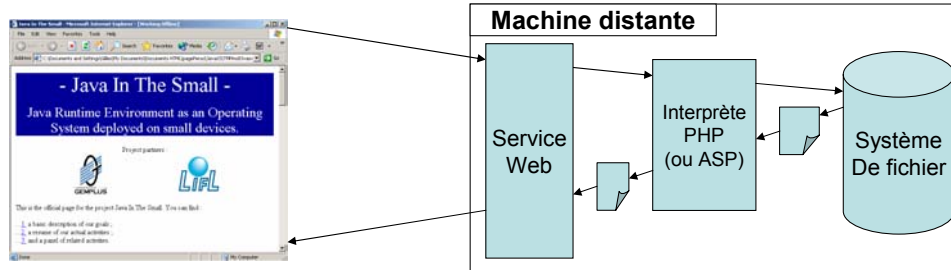
```
Public class EntryClass extends HttpServlet {
    private String message = "pas de message";
    private int compteur = 1;
    public void init() throws ServletException {
        ServletConfig cfg = this.getServletConfig();
        String s = cfg.getInitParameter("message");
        if (s!=null) message = s;
        s = cfg.getInitParameter("compteur");
        try { compteur = Integer.parseInt(s); } catch (NumberFormatException e) {}
    }
    public void doGet(HttpServletRequest req,HttpServletResponse res) {
        response.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        pw.println("<html> <head> <title> EntryClass document </title> </head>");
        pw.println("<body> <h1> The EntryClass message </h1> <p>");
        for(i=0;i<compteur;i++)
            pw.println(message);
        pw.println("</p> </body>");
    }
}
```

# Cycle de vie d'une Servlet

- **Init**
  - Exécuter une fois lorsque le serveur web charge la servlet dans son espace de travail. Elle devrait être redéfini par la sous-classe de `Servlet` pour initialiser l'application.
- **Service**
  - Cette méthode est appelée (dans une nouvelle thread) pour chaque réception d'une requête HTTP. Elle décode les requêtes et redistribue les messages vers les méthodes « `doGet` », « `doPost` », ... (ne pas surcharger cette méthode sauf pour décoder des requêtes très particulières.)
- **`doGet`, `doPost`, ...**
  - Il faut surcharger ces méthodes pour définir les réponse du document actif aux requêtes http transmises par le client.
- **Destroy**
  - Appelé lorsque le serveur « retire » la servlet de son environnement d'exécution. Il faut surcharger cette méthode pour « libérer » les ressources monopolisées par la servlet...



# Plus simple et plus facile à produire : PHP, ASP, JSP...



- Extrait de code PHP

```
<html>
  <head> <title> MonTitre </title> </head>
  <body> <h1> Hello World </h1>
  <p> - <?= getNextCount() ?> (Ou <? echo getNextCount(); ?>)
  </body>
</html>
```

## Problème liés aux génération de documents coté serveur :

### Surcharge de calcul :

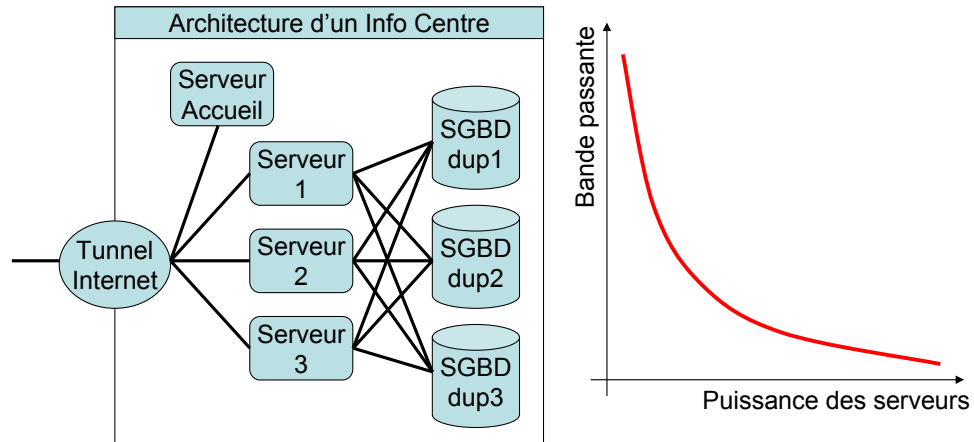
- création de processus CGI ;
- traitement de la requête...

### Surcharge de la bande passante :

- Réception de requêtes invalides ;
- Transfert de données non compactes...

⇒ Répartir la charge de travail.

# Quand le serveur devient un info centre



# Répartir la charge de travail

Des millions de clients pour quelques serveurs :

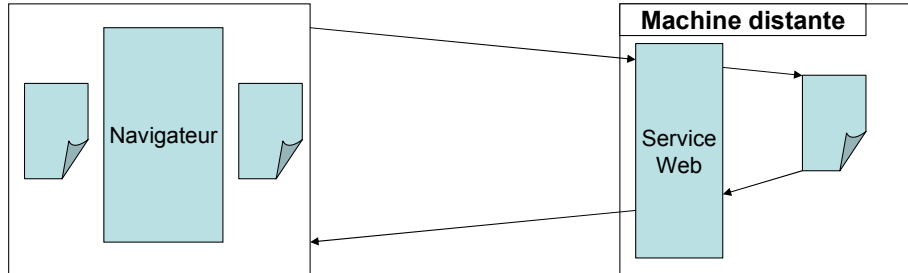
- ⇒ minimiser le nombre et la taille des requêtes ;
- ⇒ éviter la transmission de requêtes infondées ;
- ⇒ réduire le nombre de connexions simultanées au(x) serveur(s) ;
- ⇒ Transférer les données sous formes compactes plutôt que complètement formatées (exemple : des listes de valeurs plutôt que des images GIF ou JPG)

Solution :

- ⇒ Déployer des applications sur les clients.

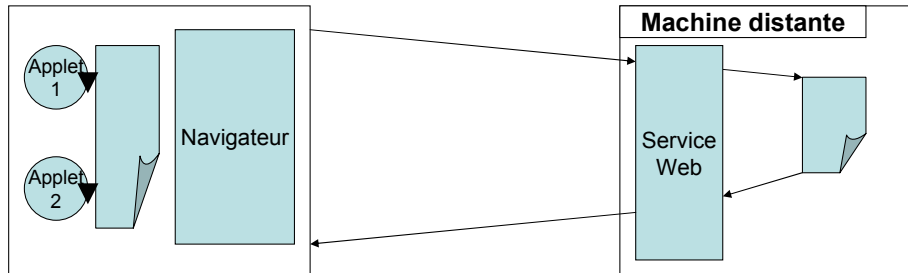
Problème d'hétérogénéité des plateformes, des environnements,  
...

# JavaScript est les documents réactifs sur le navigateur client



```
...  
<SCRIPT language=JavaScript>  
<!--  
window.onerror = new Function("return false;");  
window.onload = init;  
  
var eActiveButton = null;  
function init(){  
    tblSiteToolbar.onselectstart = new Function("return false;");  
}  
//-->  
</SCRIPT>  
...
```

# Java et les applets intégrées aux documents HTML



```
...  
<applet  
  codebase="." code="Clock2.class"  
  width=170 height=150  
  alt="alternative message" >  
  <param name=bgcolor value="000000">  
  <param name=unusedp value="Un Texte">  
</applet>  
...
```

## Définir une applet Java

- Hériter de `Java.applet.Applet`  
qui est une sorte de `java.awt.panel`.
- Redéfinir la méthode `void init()`  
appelé lorsque l'applet viens d'être chargés.
- Redéfinir la méthode `void start()`  
appelé à chaque (re)démarrage de l'applet.
- Redéfinir la méthode `void stop()`  
appelé lorsque le document contenant l'applet est quitté.
- Redéfinir la méthode `void destroy()`  
appelé lorsque l'applet est retirée du navigateur.
- Redéfinir la méthode `String [][] getParameterInfo()`

# Exemple d'Applet

```
public class Examples extends Applet implements Runnable {
    Thread timer;
    public void init() {
        try {setBackground(new Color(Integer.parseInt(getParameter("bgcolor"),16)));}
        catch (Exception E) { }
    }
    public void paint(Graphics g) { ... }
    public void start() { timer = new Thread(this); timer.start(); }
    public void stop() { timer = null; }
    public void run() { Thread me = Thread.currentThread();
        while (timer == me) {
            try { Thread.currentThread().sleep(100);}
            catch (InterruptedException e) {}
            repaint();}
    }
    public String getAppletInfo() { return "Title: An Applet \n By Me!"; }
    public String[][] getParameterInfo() {
        return {{ "bgcolor", "hexadecimal RGB number", "The background color.
        Default is the color of your browser."}};
    }
}
```



## Interaction Applet / Navigateur et Applet / Navigateur / Applet

Les applets peuvent interagir (dans un cadre limité par les problèmes de sécurité) avec le navigateur sur lequel elles sont déployées :

```
public void init() {  
    AppletContext aC = this.getAppletContext();  
    ac.showDocument(new URL("une URL"), "target");  
    ac.showStatus("a status text");  
}
```

Elles peuvent interagir entre elles via le navigateur :

```
Applet a = ac.getApplet("leNomDeLApplet");
```

# Bénéfice de la répartition des tâches

Répartition réduite à :

- IHM coté client
- Données coté serveur

Problème de génie logiciel :

- Ou est la place des composants métiers

