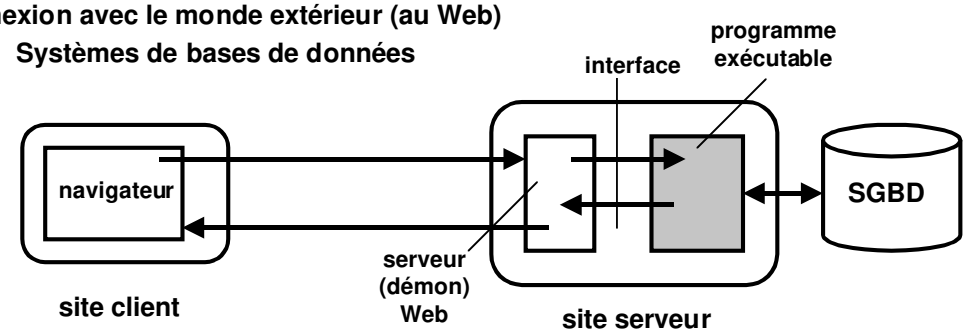
Le Web comme support d'exécution d'applications

Au delà de sa fonction d'accès à l'information, le Web peut servir de support à l'exécution d'applications réparties.

- **Intérêt du Web comme support d'applications**
  - Une interface familière, intuitive et présente partout (le navigateur, les liens)
  - Des outils de base
    - Espace universel de désignation (les URI)
    - Protocole universel de transfert d'information (HTTP)
    - Gestion d'information sous un format standard (HTML), ou extensible (XML)
  - Le Web comme un "système d'exploitation" primitif pour applications réparties ?
- **Problèmes à résoudre**
  - Où et comment sont exécutés les programmes ?
    - sur le site serveur : scripts CGI, *servlets*
    - sur le site client : scripts dans extension du navigateur (*plugin*), *applets*
  - Comment est assurée la sécurité ?
    - problème majeur, non entièrement résolu
      - protection des sites
      - chiffrement de l'information
      - restriction sur les conditions d'exécution

Exécution de programmes sur un serveur Web

- **Intérêt**
  - Exécution de programmes interactifs
    - Formulaires (inscriptions, enquêtes, etc.)
    - Requêtes (moteur de recherche, etc.)
  - Connexion avec le monde extérieur (au Web)
    - Systèmes de bases de données



- **Mécanismes**
  - Script CGI *(Common Gateway Interface)*
    - le plus ancien
    - langages multiples, interface commune standard
  - *Servlets*
    - programmes Java activés sur le serveur, interface spécifique Java

CGI : principes

Consulter le standard de facto (NCSA) : http://hoohoo.ncsa.uiuc.edu/cgi/

- **Interface commune pour l'exécution de programmes sur le site d'un serveur Web**
  - Multi-langages : scripts (Perl, Tcl, *shell* Unix, etc.), langages compilés (C, C++, etc.)
  - Localisation des programmes dans des répertoires spécifiques
    - /cgi-bin (exécutables), /cgi-src (sources des programmes compilés)
  - Conventions communes pour le passage de paramètres via HTTP
- **Exemples de conventions** (présentation simplifiée)
  - Récupération des paramètres d'entrée envoyés depuis le navigateur
    - par la commande HTTP GET
      - dans variable $QUERY_STRING, tout ce qui se trouve derrière le "?" dans l'URI transmise
    - par la commande HTTP POST
      - dans flot d'entrée standard `stdin`, encodage selon standard (~MIME)
  - Composition des résultats pour envoi au navigateur
      - dans un document (HTML ou autre, selon standard MIME
      - sous forme d'une URI (pointant vers le résultat)

CGI : problèmes

- **Le principal problème de l'utilisation de CGI est la sécurité**
  - CGI permet (potentiellement) à un client de faire exécuter un programme quelconque sur un serveur en lui passant des paramètres quelconques
  - Dangers
    - pénétration dans le système du serveur (via exécution de scripts Unix)
    - extraction d'informations confidentielles
- **Mesures pour améliorer la sécurité**
  - Les répertoires `/cgi-bin` et `/cgi-src` doivent être protégés (accès réservé à l'administrateur)
  - Il faut éviter d'y placer des commandes qui exécutent leurs paramètres comme un programme - cela revient à laisser le client écrire son propre programme
    - Exemples de telles commandes: `system` (dans *shell* Unix), `eval` (dans Tcl, Perl, etc.)
  - Il faut filtrer les paramètres des programmes (filtre spécifique à chaque programme)
    - Attention aux paramètres qui provoquent une exception dans le programme (type illégal, valeur hors limites, chaîne trop longue, etc.)
  - Le processus qui exécute le programme CGI doit avoir des droits limités
    - Il ne doit en particulier pas s'exécuter avec les droits (`uid`) de `root`, même s'il est lancé par `root`

Servlets (1)

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

- **Une alternative à CGI, utilisant le langage Java**
  - **La classe `Servlet` (et ses extensions) fournissent des outils pour traiter les requêtes HTTP et pour construire les réponses (documents HTML)**
    - **`doGet` et `doPost` : traitent les opérations GET et POST de HTTP**
    - **`ServletRequest` : objet permettant de récupérer les paramètres fournis par le client**
    - **`ServletResponse` : objet permettant de préparer une réponse HTML au client**
  - **Autres outils**
    - **synchronisation entre clients multiples d'un *Servlet***
    - **communication entre servlets**
    - **communication avec d'autres sites**
- **CGI versus Servlets**
  - **Avantages des *Servlets* : sécurité meilleure (JVM) ; plus grande efficacité d'exécution (*threads* au lieu de processus lourds)**
  - **Avantage de CGI : pas de restriction sur le langage : possibilité de développement rapide avec langages interprétés, en particulier prototypage rapide**
  - **Utilisation préférable des *Servlets* pour application importante nécessitant connexions externes (accès à un SGBD)**

Servlets (2)

Exemple très simple de *servlet*

Répond à l'opération GET en renvoyant un message prédéfini dans une page HTML

Objets définis:
HttpServletRequest représente la requête du client (n'est pas utilisée)

HttpServletResponse représente la réponse construite par le serveur et renvoyée au client

Exemple

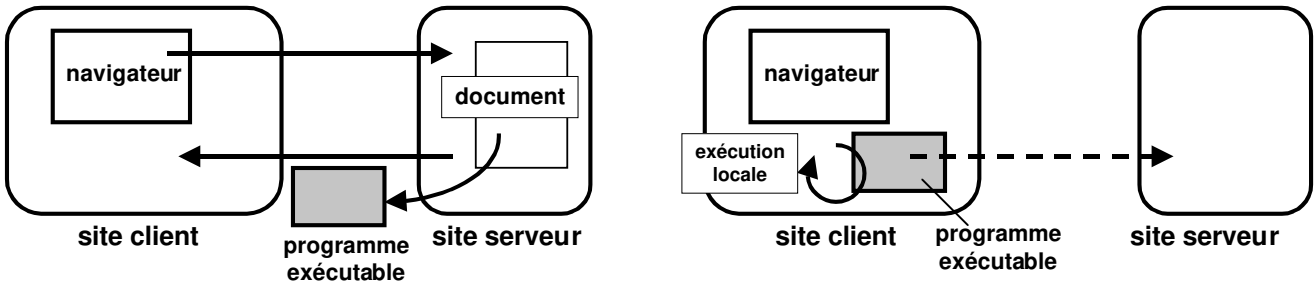
```
public class SimpleServlet extends HttpServlet
 {
     /**
      * Repondre à l'opération HTTP GET en  renvoyant
      * une page web simple.
      * Surcharge la méthode doGet de la classe HttpServlet
      */
      public void doGet (HttpServletRequest request,
          HttpServletResponse response)
          throws ServletException, IOException
        {
          PrintWriter         out;
          String    title = " Message de SimpleServlet ";

               // fixer le type de la réponse
               response.setContentType("text/html");
               // écrire le texte de la éeponse
               out = response.getWriter();

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

Exécution de programmes sur un client Web

- **Intérêt**
  - Décharger le serveur, en utilisant la capacité de traitement locale
  - Rendre l'exécution indépendante de la charge du réseau (en particulier pour affichage, animation)
  - Réaction rapide pour les programmes interactifs



- **Mécanismes** (nécessitent extension du navigateur)
  - Scripts
    - Tcl, Perl, JavaScript
  - *Applet* Java

Applets

Consulter http://java.sun.com/docs/books/tutorial/applet/

- **Définition**
  - Une *applet* est un programme Java inclus dans une page HTML. Quand la page est chargée par un client, l'*applet* est exécutée sur la machine virtuelle Java incluse dans le navigateur
- **Forme**

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

  Le programme (bytecodes) est dans MyApplet.class, dans le même répertoire que le document HTML qui contient l'*applet*

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

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

- **Restrictions**
  - Les capacités d'une *applet* sont limitées pour des raisons de sécurité. Actions interdites
    - charger des bibliothèques (importer du code nouveau)
    - accéder aux fichiers locaux sur le site du client (possible sous certaines conditions dans JDK 1.2)
    - ouvrir une connexion Internet, sauf vers le serveur d'où elle provient
    - lancer un nouveau processus sur le site du client
    - lire certaines caractéristiques du système d'exploitation local

Applets

Exemple d'*applet*

Cet exemple montre une *applet* qui réagit aux principaux événements de son cycle de vie (premier chargement de la page HTML support, démarrage, changement de page, destruction) en affichant un message approprié dans une fenêtre.

Les méthodes `init`, `start`, etc. sont définies dans la classe `Applet` et surchargées dans cet exemple. Elles sont automatiquement appelées par le navigateur ou l'`AppletViewer` (inspecteur d'*applets*) lors des événements correspondants

```
import java.applet.Applet;
import java.awt.Graphics;

              public class Simple extends Applet {

    StringBuffer buffer;

    public void init() {
        buffer = new StringBuffer();
        addItem("initializing... ");
    }

    public void start() {
        addItem("starting... ");
    }

    public void stop() {
        addItem("stopping... ");
    }

    public void destroy() {
        addItem("preparing for unloading...");
    }

    void addItem(String newWord) {
        System.out.println(newWord);
        buffer.append(newWord);
        repaint();
    }

    public void paint(Graphics g) {
        //Draw a Rectangle around the applet's display area.
        g.drawRect(0, 0, size().width - 1, size().height - 1);

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

extrait de http://java.sun.com/docs/books/tutorial/applet/

Javascript

Consulter http://developer.netscape.com/tech/javascript/

■ Qu'est-ce que JavaScript ?

- **Langage de script (interprété) destiné à être exécuté sur un site client Web**
  - **il y a aussi une utilisation possible côté serveur**
- **Un programme JavaScript est inclus dans une page HTML (comme une *applet*)**
  <script language="JavaScript"> … texte du programme …</script>
- **L'interprète JavaScript est inclus dans le navigateur (JavaScript est issu de Netscape)**
- **Malgré son nom, JavaScript ≠ Java. JavaScript a un modèle à objets rudimentaire**

■ Usages de JavaScript

- **Interactions locales sur le site client (évite la consultation du serveur)**
  - **modifier dynamiquement le contenu d'une page HTML**
  - **afficher sélectivement des images**
  - **afficher des boîtes de dialogue, contrôler la validité des données d'un formulaire**
  - **gérer un historique des documents visités**

■ JavaScript versus applets Java

- **Domaine plus restreint pour JavaScript (formulaires, images, affichage local)**
- **Langage de script vs langage compilé (développement plus rapide, interactif)**
- **Sécurité moindre pour JavaScript (pas d'équivalent de la JVM)**

Javascript : exemples

Exemple 1 : alternance entre images multiples

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

Dans cet exemple, une image est affichée (et sert d'ancre pour un lien hypertexte). Quand le curseur de la souris entre ou sort du cadre de l'image, une image différente est affichée (on alterne entre 2 images)

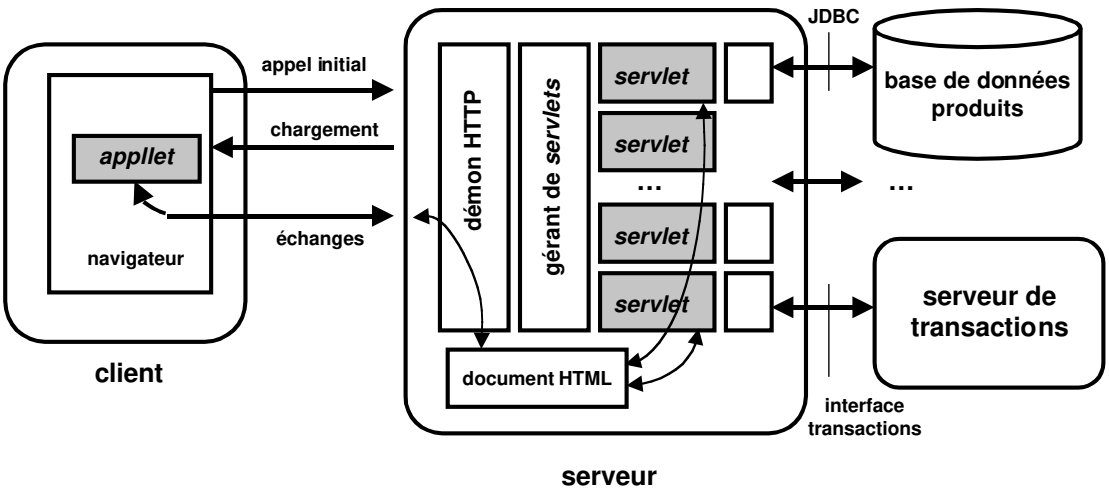
Exemple 2 : une horloge simple (affiche heure courante)
(extrait de :
http://www.multimania.com/dliard/Sciences/Informatique/Langages/Scripts/Javascript)

```
<HTML>
  <HEAD>
    <SCRIPT LANGUAGE="javascript">
      <!--
      function clock(){
        var date=new Date();
        string=" "+date.getHours()+' : '+
               date.getMinutes()+' : '+
               date.getSeconds();
        document.forms[0].elements[0].value=string;
        setTimeout('clock()',150);
      }
      //-->
    </SCRIPT>
  </HEAD>
  <BODY onLoad="clock();">
    ...
    <FORM>
      Il est très exactement :
      <INPUT Type="text" Value="hh : mm : nn">
    </FORM>
    ...
  </BODY>
</HTMLd
```

Le type Date et les fonctions getHours, etc. sont prédéfinis

Schéma (très simplifié) d'une application sur le Web

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



Web - aspects système : caches Web

- **Fonctions générales d'un cache (rappel)**
  - Introduire un niveau intermédiaire, d'accès rapide, entre le lieu de stockage d'une information et celui de son utilisation.
  - Objectifs :
    - réduire le temps moyen d'accès, en conservant les informations les plus utilisées
    - réduire le trafic entre les niveaux de stockage (pour le web : trafic sur l'Internet)
  - Hypothèse de travail (souvent vérifiée) : localité d'accès (réutilisation des informations)
- **Le web se prête bien à l'usage de caches**
  - Les informations changent relativement peu souvent
  - On peut travailler sur des regroupements de demandes (à plusieurs niveaux)
    - cache individuel sur disque
    - cache local pour un département, une entreprise, etc.
    - cache régional pour un ensemble de réseaux
- **Problèmes à résoudre**
  - Choix des informations à conserver
  - Politique d'élimination des informations quand le cache est plein
  - Rafraîchissement des informations supposées périmées
  - Coopérations entre caches

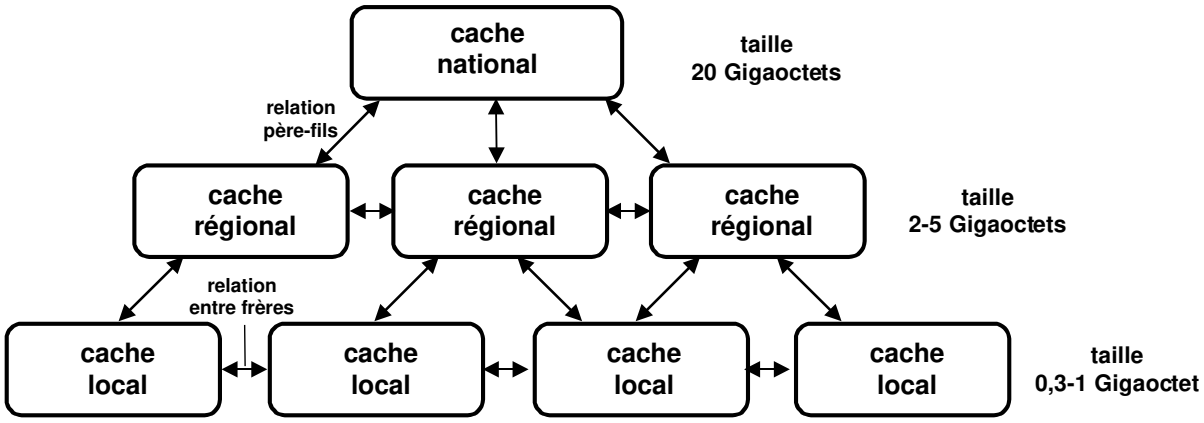
Un cache populaire (gratuit) : Squid. Voir http://squid.nlanr.net/Squid/

Problèmes des caches web

- **Politique de remplacement** (quels documents éliminer quand on a besoin de place)
  - FIFO (dans l'ordre des arrivées) : simple à réaliser, peu intéressant
  - SIZE : éliminer le document le plus gros (pour gagner de la place) : gain à court terme, mais risque de perte si le document éliminé était très demandé
  - LRU (*Least Recently Used*) : fondé sur l'hypothèse de localité, souvent utilisé
- **Cohérence** (comment garantir que les documents du cache sont à jour)
  - Invalidation : le serveur prévient le cache quand l'original est modifié
    - idéal, mais grosse charge de gestion pour le serveur (doit garder trace des copies)
  - TTL *(Time To Live)* : durée de vie limitée ; élimination ou rappel serveur à l'expiration
  - Autre solution : durée de vie proportionnelle à l'âge du document
- **Coopération entre caches**
  - Hiérarchie : tout cache a un "parent", auquel il transmet la requête s'il ne peut la résoudre
    - Le parent fait de même (ou contacte le serveur s'il n'a pas de parent), puis répond au fils
  - Entre égaux : un cache transmet la requête aux autres caches "frères" et au serveur ; il prend la première réponse qui arrive
  - Le mode de coopération entre deux caches n'est pas fixé a priori et peut dépendre de la nature des requêtes

Exemple de hiérarchie de caches Web

- **Projet de caches du réseau Renater**



Rendement espéré : local 25%, régional 20%, national 15%

Voir : http://www.serveurs-nationaux.jussieu.fr/cache/

Résumé de la séance 12

- **World Wide Web (suite)**
  - **Le Web comme support d'applications**
    - **exécution sur le site client ou serveur**
    - **scripts, applets, servlets**
    - **structure d'une application**
    - **quelques aspects de la sécurité**
  - **Aspects système du Web**
    - **caches, coopération entre caches**

Fin du cours

Examen le vendredi 5 mai 2000, 14h-17h, salles F018-F320 UFR IMA
(date et lieu seront confirmés)

Seront autorisés :
les notes personnelles
les documents distribués pendant le cours
les documents distribués pendant les TP