Programmation d'Application Web avec Java

Tarak CHAARI

Tarak.chaari@isecs.rnu.tn

Version adaptée du cours de Mr Walid MAHDI (ISIMS)

Chapitre

1

Servlets java

(Le dessous cachés du JSP)

1- Qu'est ce qu'une Servlet ? (1/6)

Servlet : Server-side applet

Une Servelt est un composant Web du coté serveur Web :

- Qui permet d'étendre les possibilités d'un serveur Web.
  - Possibilité de générer du contenu dynamique en réponse à des requêtes clients.
    - Un peu comme les scripts CGI (Common Gateway Interface).
- C'est une classe Java exécutée sur un serveur multi-threadé (comme un serveur Web)
  - Elle est compilée sous forme de byte-code,
  - Elle est exécutée par une machine virtuelle Java (JVM).
- Elle est mise en œuvre est gérée par un conteneur Web (Tomcat par exemple).

2- Servlet vs. Applet (1/2)

Applets :

- Interface graphique utilisateur,
- Nécessite un browser adéquat,
- Traitements sur le client (Client lourd),
- Limites de services dues aux problèmes de sécurité.

Servlets

- pas d'interface graphique utilisateur (Langage HTML),
- pas de limitations de sécurité :
  - possibilité d'établissement de connexions avec d 'autres machines que le serveur (utilisation comme pont JDBC-ODBC),
  - possibilité d'appels systèmes (JDBC)
  - manipulation de ressources locales du serveur

2- Servlet vs. Applet (2/2)

Avantage Servlet:

- inhérents à Java :
  - JSDK ou JDK1.2 gratuit et portable
- par rapport aux Applets :
  - plus facile à développer,
  - meilleures performances,
  - client léger ,

Inconvénient Servlet :

- interface graphique utilisateur limitée à HTML.

3- Servlet vs. CGI (1/2)

Principe CGI :

- Un processus par requête est lancé sur le serveur.

Avantages CGI :

- Gratuit, pris en charge par tous les serveurs Web actuels,
- Peut être écrit dans n'importe quel langage (C, perl).

Inconvénients CGI :

- Manque d évolutivité (plusieurs processus créés),
- Serveur très sollicité si plusieurs requêtes en même moment,
- Assez lent et parfois difficile à développer.

3- Servlet vs. CGI (2/2)

Les Servlets sont portables, plus efficaces, plus pratiques et plus puissantes :

- indépendance des OS, c'est du Java !,
- indépendance des serveurs web (Apache, Microsoft IIS, WebStar, etc.),
- efficacité (connexion multi-threads avec les utilisateurs, un seul chargement, permanence en mémoire),
- super API (Application Programming Interface) pour gérer les formulaires HTML,
- dialogue possible avec des applets situées sur le client (utilisation d 'un protocole à objets distribués RMI),
- gestion des sessions,
- faible coût : kit de développement des Servlets gratuit (Apache/Tomcat reste la solution la plus efficace... 100% gratuite).

4- L'API Servlet (1/5)

- **L'API Servlet fournit un certain nombre de classes et d'interfaces permettant :**
  - le développement des Servlets,
  - leur déploiement,
  - et leur mise en œuvre au sein du conteneur Web.
- **L'API Servlet est contenue principalement dans deux packages :**
  - javax.servlet
  - javax.servlet.http

MapremièreServlet.java

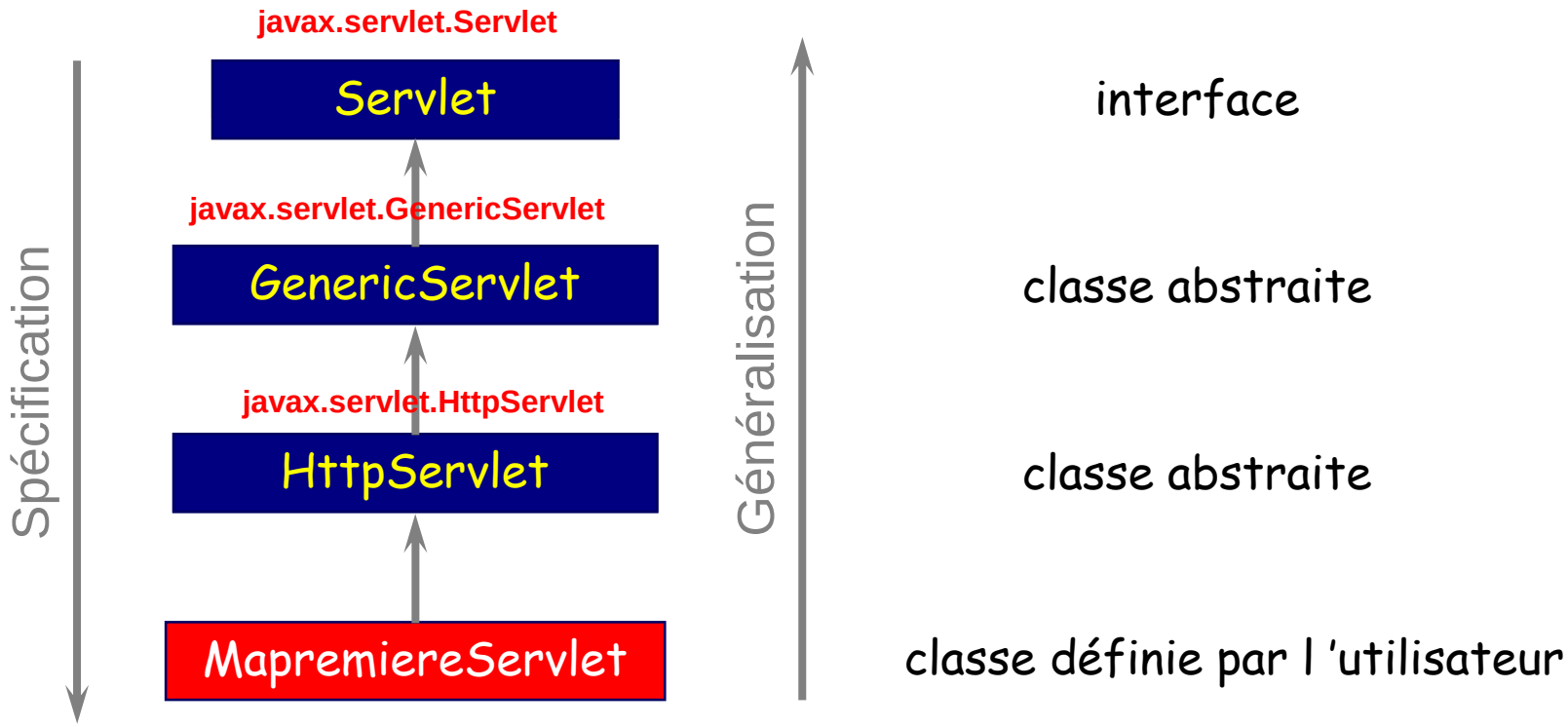
```
import javax.servlet.*
import javax.servlet.http.*
public class ……..{
……..
……..
……..
}
```

4- L'API Servlet (2/5)

- **Le package javax.servlet :**
  - Contient les classes pour un support des Servlets génériques et indépendant du protocole.
- **Le package javax.servlet.http :**
  - Contient des extensions des classes du package javax.servlet.
    - Ces extensions consistent à ajouter des fonctionnalités spécifiques au protocole HTTP.
- **Le nom du package le plus haut « javax » au lieu du « java » plus familier, indique que l'API Servlet est une extension standard.**

4- L'API Servlet (3/5)

Chaque Servlet utilisateur doit implémenter l'interface javax.servlet.Servlet soit directement soit par l'extension de la classe spéciale javax.servlet.GenericServlet ou javax.servlet.http.HttpServlet.



4- L'API Servlet (4/5)

- **L'interface javax.servlet.Servlet possède les méthodes :**
  - init( ) :
    - pour initialiser la Servlet.
  - Service ( ) :
    - pour recevoir et répondre aux requêtes des clients.
  - destroy()
    - détruire la servlet et ses ressources.

- **Ces 3 méthodes sont tous héritées donc par une Servlet utilisateur.**

4- L'API Servlet (5/5)

- Une **Servlet** utilisateur peut implémenter l'interface **javax.servlet.Servlet** directement

```
public class MapremiereServlet implements Servlet {
....
```

- Une Servlet utilisateur indépendante du protocole devait être une sous classe de **GenericServlet**

```
public class MapremiereServlet extends GenericServlet {
....
```

- Une Servlet **Http** devait être une sous classe de **HttpServlet.**

```
public class MapremiereServlet extends HttpServlet {
....
```

5- Modèle de programmations (1/4)

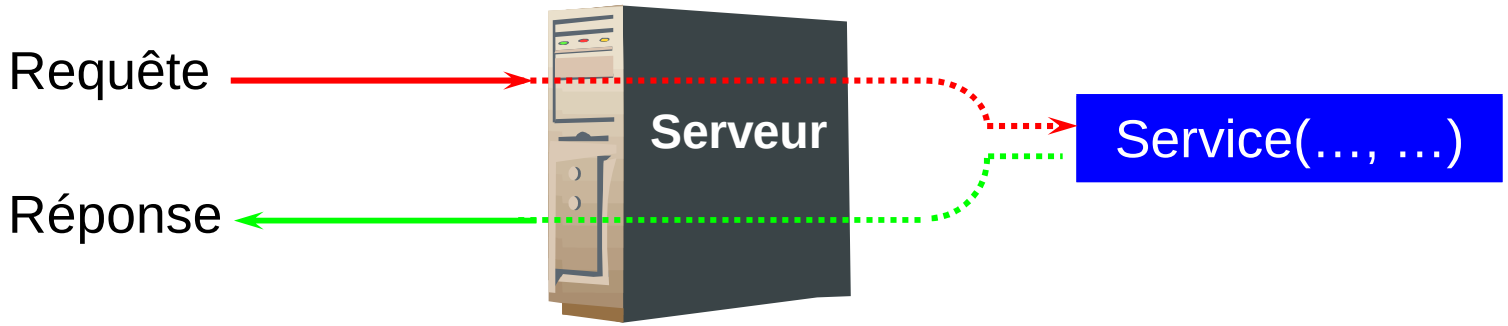
Une Servlet suit un modèle de programmation requête-service-réponse.

- A la place d'une méthode main( ), elle possède une méthode service( ) qui sera invoquée automatiquement à chaque fois que la Servlet reçoit une requête par le serveur.

- La méthode service(objet1, objet2) accepte deux paramètres et permet de recevoir et de répondre aux requêtes des clients :
  - Un objet1: javax.servlet.ServletRequest
    - contient les informations nécessaires pour une communication du client vers le serveur (Requête).
  - Un objet2: javax.servlet.ServletResponse
    - contient les informations nécessaires pour une communication du serveur vers le client (Réponse).

5- Modèle de programmations (2/4)

Modèle de programmation requête-service-réponse
pour une Servlet Générique



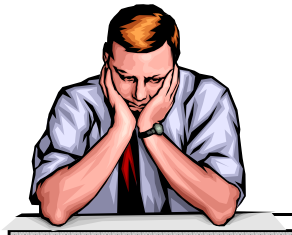
- **Une Servlet générique doit surcharger sa méthode service(ServletRequest obj1, ServletResponse obj2).**

5- Modèle de programmations (3/4)

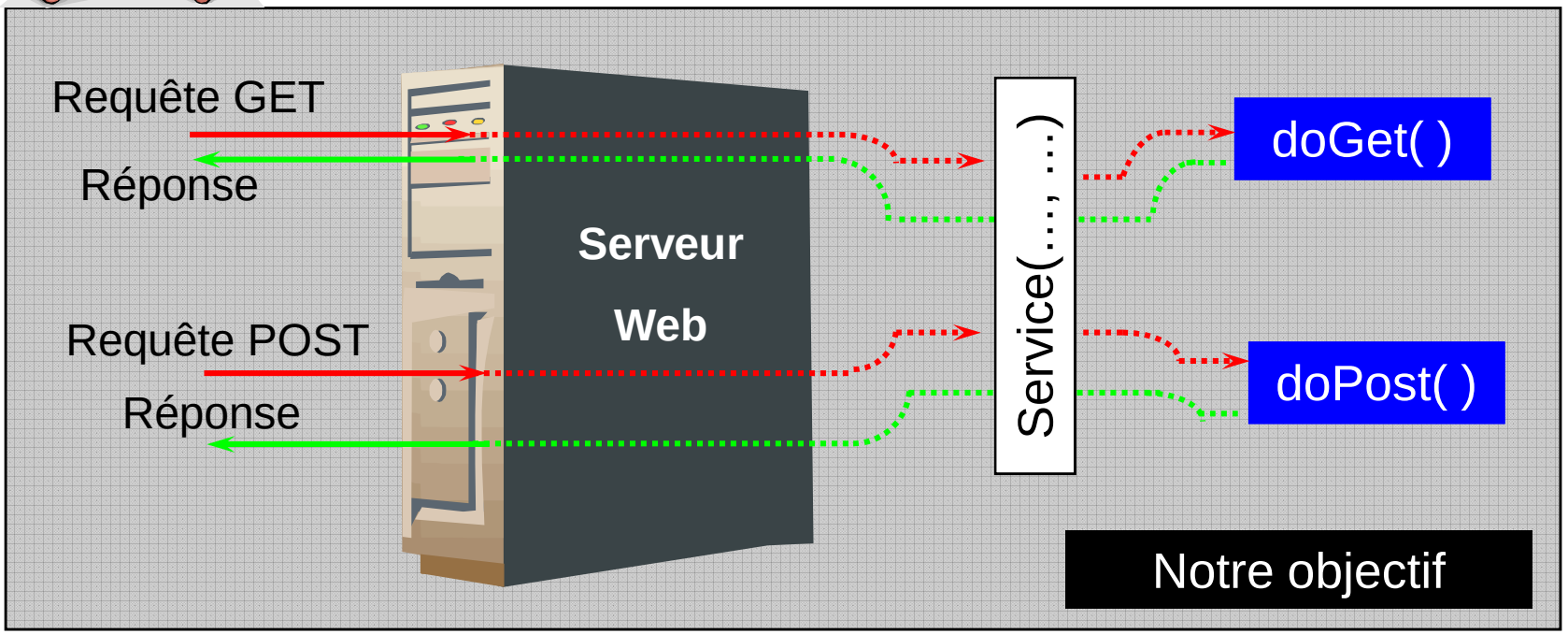
Une Servlet Http ne surcharge pas la méthode service().

- Cette méthode service() de la classe mère est remplacée avantageusement par 2 méthodes ayant la même signature :
  - doGet() : pour traiter des requêtes Http de type GET
  - doPost() : pour traiter des requêtes Http de type POST
- Une Servlet Http doit obligatoirement contenir l'une ou l'autre de ces 2 méthodes.
- La méthode service() de HttpServlet n'est pas surchargée mais elle prend en charge l'appel automatique de la bonne méthode doXXX() en fonction du type de requêtes.

5- Modèle de programmations (4/4)



Modèle de programmation requête-service-réponse
pour une Servlet Http



6- Cycle de vie d'une Servlet HTTP

Instanciation et chargement

Initialisation
invocation de la méthode init()

Requêtes clients

En service
invocation de la méthode service()

Arrêt du serveur

Destruction
invocation de la méthode destory()

puis finalize()

Classe de Servlet

7- Modèles d'implémentation d'une Servlet Http (1/3)

Hello.java

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

public class Hello extends HttpServlet{
public void doGet(HttpServletRequest req,  HttpServletResponse res)
    throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML>");
    out.println("<HEAD> <TITLE>Hello</TITLE> </HEAD>" );
    out.println("<BODY>");
    out.println("<H1>Hello </H1>" );
    out.println("<BODY> </HTML>");
    }
}
```

7- Modèles d'implémentation d'une Servlet Http (2/3)

- **Sans les imports nécessaires le compilateur javac ne peut pas compiler la classe Hello.java.**
  - L'exception ServletException est définie dans le paquetage javax.servlet.*;
  - L'exception IOException est définie dans le paquetage javax.io;
  - Les objets HttpServletRequest, HttpServletResponse utilisés comme paramètres des méthodes doGet() ou doPost sont définies dans le paquetage javax.servlet.http.*;
- **ServletException, IOException sont des déclaration d'exceptions levées mais non traitées.**

7- Modèles d'implémentation d'une Servlet Http (3/3)

- **L'instruction res.setContentType("text/html") initialise l'objet res qui est de type HttpServletResponse comme étant une réponse de type « texte/html », le type MIME standard pour le contenu de pages HTML.**
  - Exemples de types MIM image/gif, image/jpeg, text/html, text/plain, text/*, */*.
- **L'objet PrintWriter permet à une Servlet Http de construire la page HTML destinée à l'utilisateur.**
  - L'instruction « printWriter out = res.getWriter() » permet de retrouver un flux de sortie « out » (un printWriter) à travers l'objet res pour envoyer le message « Hello » au client.

8- Invocation d'une Servlet à partir d'un navigateur Web (1/7)

- **Invoquer une Servlet c'est utiliser tout d'abord un conteneur Web (exp : JSWDK) pour sa mise en œuvre.**

- **Déployer la Servlet au sein d'un serveur Web (supportant l'exécution des servlets).**
  - [Optionnel selon le container web] Déclarer l'ajout de la Servlet dans le conteneur Web.
  - Copier le code compilé de la Servlet dans le répertoire d'hébergement de la servlet (exp: **webpages\WEB-INF\servlets**)

- **Deux possibilités d'invocation d'une Servlet :**
  - invocation de la méthode doGet(…),
  - invocation de la méthode doPost(..).

8- Invocation d'une Servlet à partir d'un navigateur Web (2/7)

1ère Invocation de la méthode doGet(...) :

- Saisie de l'URL de la Servlet dans la barre d'adresse du navigateur.
- http://<hôte>:<port>/servlet/<servlet>
- http://localhost:8080/servlet/Hello

8- Invocation d'une Servlet à partir d'un navigateur Web (3/7)

2ième Invocation de la méthode doGet(...):

- Clique sur un lien hypertexte qui pointe sur l'URL de la Servlet.

Index.html

```
<HTML>
<HEAD>
  <TITLE>
   Test de la servlet Heloo par clic sur lien
  </TITLE>
 </HEAD>
 <BODY>
  <P>
  <A href="/servlet/Hello">
   Cliquer pour tester la Servlet Hello
   </A>
   </P>
</BODY>
</HTML>
```

8- Invocation d'une Servlet à partir d'un navigateur Web (4/7)

Invocation de la méthode doPost(…) :

- La méthode doPost() d'une Servlet est invoquée principalement lors de l'envoie des données saisies dans un formulaire HTML (par un clic sur un bouton de type submit).

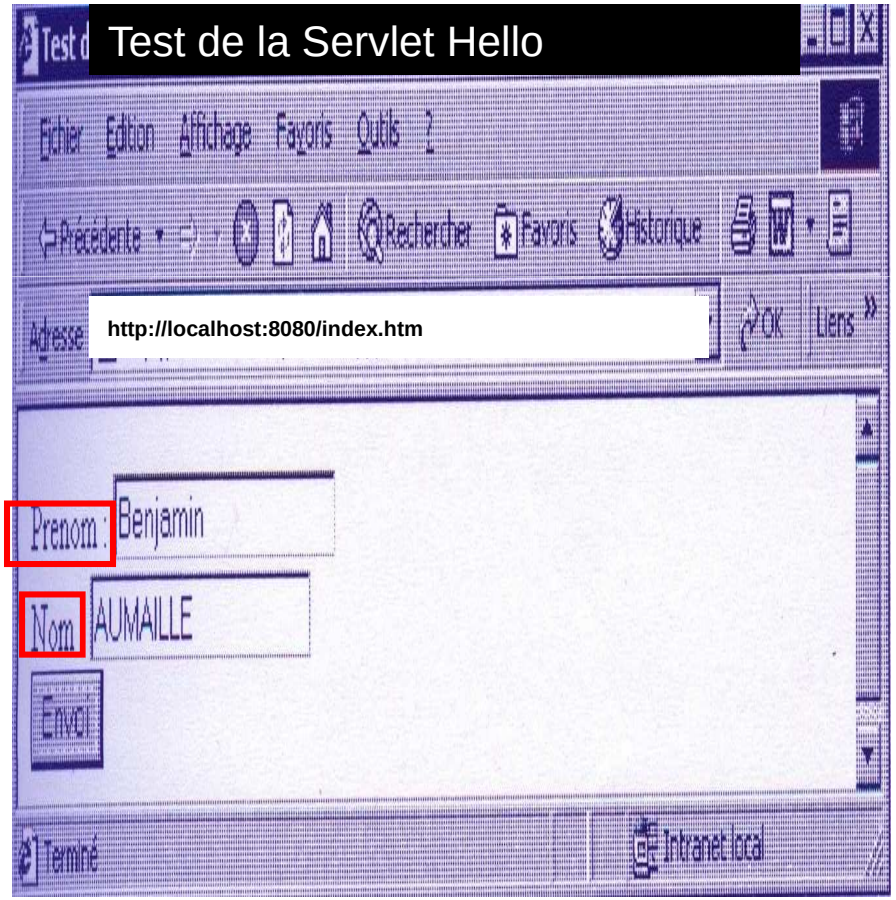
- Exemple de méthode doPost() qui retourne une chaîne de caractères concaténée avec les valeurs des paramètres transmis par le client.

8- Invocation d'une Servlet à partir d'un navigateur Web (5/7)

Invocation de la méthode doPost(...) :

Index.html

```
<HTML>
<HEAD>
   <TITLE> Test de la servlet Hello </TITLE>
</HEAD>

<BODY>
<FORM action = "/servlet/Hello" method ="post">
 <P>
 Prenom : <INPUT type ="text" name="prenom">
 <BR>

 Nom : <INPUT type ="text" name="nom">
 <BR>

<INPUT type ="submit" value="Valider">
 </P>

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



8- Invocation d'une Servlet à partir d'un navigateur Web (6/7)

Invocation de la méthode doPost(...) :

Hello.java

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

 public class Hello extends HttpServlet{
 public void doPost(HttpServletRequest req,  HttpServletResponse res)
 throws ServletException, IOException {

 String prenom = req.getParametre("prenom");
 String nom = req.getParametre("nom");

 res.setContentType("text/html");
 PrintWriter out = res.getWriter();
  out.println("<HTML><BODY>");
  out.println("<H1>Bonjour" +prenom+" "+ nom+"."+ "</H1>" );
  out.println("</HTML><BODY> ");
   }
 }
```

8- Invocation d'une Servlet à partir d'un navigateur Web (7/7)

Invocation de la méthode doPost(...) :

9- Paramètres de l'application Web (1/9)

- **De la même manière que pour chaque Servlet d'une application Web, il est possible de déclarer des paramètres globaux (informations de configuration) pour toute l'application Web.**

- **Ces paramètres peuvent être utiles pour déclarer des informations susceptible d'être utilisées par plusieurs Servlets de l'application Web:**
  - Nom et e-mail de l'administrateur, qui peuvent être utilisés pour générer une page d'erreur à un client.
  - Nom d'hôte ou adresse IP de machines distantes, qui peuvent être utiles pour l'accès à des ressources distantes
  - Nom de la base de données, nom du pilote JDBC à utiliser, nom d'utilisateur et mot de passe pour établir la connexion,
  - Etc,

9- Paramètres de l'application Web (2/9)

- **Les informations de configuration d'une application Web sont représentées par un objet de type javax.servlet.ServletContext.**
  - Chaque Servlet d'une même application Web a donc accès à ces informations.

- **L'objet javax.servlet.ServletContext propose des méthodes permettant de travailler principalement avec deux catégories de données :**
  - Créer, lire et supprimer des attributs de façon logicielle, permettant le partage de ressources entre les Servlets d'une même application Web.

13- Paramètres de l'application Web (4/9)

Méthodes de l'interface javax.servlet.ServletContext dédiées à la récupération des paramètres globaux d'initialisation:

- public String getInitParameter(String nom) :
  - Récupérer une chaîne de caractères contenant la valeur d'un paramètre nommé nom ou la valeur null si le paramètre n'existe pas.

- public java.util.Enumeration getInitParameterNames():
  - Récupérer sous la forme d'un objet de type java.util.Enumeration l'ensemble des noms des paramètres déclarés pour la Servlet.

13- Paramètres de l'application Web (6/9)

Exemple de manipulation :

ErreurServlet.java

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

        public class ErreurServlet extends HttpServlet{

        public void doGet(HttpServletRequest req,  HttpServletResponse res) throws ServletException, IOException {

        ServletContext application=getServletContext();
        String nom= application.getInitParameter ("nomAdmin");
        String email= application.getInitParameter (" emailAdmin");

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

        out.println("<HTML><BODY>");
         out.println("<H1>Erreur de l'application</H1>" );
         out.println("<BR><H4>Veuillez contacter <B>" + nom + "</B>" );
         out.println("<A href =\"mailto:"+ email + "\" >"+email+"</A>" );
         out.println("<H4></BODY></HTML> ");
        }
        }
```

13- Paramètres de l'application Web (8/9)

Méthodes de l'interface javax.servlet.ServletContext dédiées à la gestion logicielle des attributs du contexte d'application:

- public String setAttribute(String nom, Object objet) :
  - Créer un attribut dans le contexte de l'application Web. Si le nom de l'attribut existe déjà, la valeur existante est remplacée par la nouvelle.
- public Object getAttribute(String nom) :
  - Récupérer la valeur d'un attribut dont le nom est passé en paramètre, ou la valeur **null** si l'attribut n'existe pas.
- public java.util.Enumeration getAttributeNames():
  - Récupérer sous la forme d'un objet de type java.util.Enumeration le nom de tous les attributs stockées dans l'application Web.
- public void removeAttribute(String nom):
  - Supprimer un attribut du contexte de l'application Web, dont le nom est passé en paramètre.

13- Paramètres de l'application Web (9/9)

Exemple de manipulation :

```
……..

        Employé emp1 = new Employé (" Walid ", "MAHDI ");
        Employé emp2 = new Employé ("toto ", "titi ");
        Employé emp3 = new Employé (" tata ", "tatou ");
        ……..

        javax.servlet.ServletContext contextApp = getServletContext();

        contextApp.setAttribute(" Employé1 ", emp1);
        contextApp.setAttribute(" Employé2 ", emp2);
        contextApp.setAttribute(" Employé3 ", emp3);
        ……
```

```
…

        javax.servlet.ServletContext contextApp = getServletContext();
        Java.util.Enumeration nomAttributs = contextApp.getAttributeNames( );

        while ( nomAttributs.hasMoreElements() ) {
           String nom = (String) nomAttributs.nextElement();
           Employé e = (Employé) contextApp.getAttribute(nom);
           …….

           contextApp.removeAttribute(nom);
        ……
```

14- Interfaces ServletRequest et HttpServletRequest (1/7)

Méthodes de Récupération d'informations sur l'URL de la requête

- public String getScheme( ) :
  - Retourne le nom du protocole utilisé par le client pour émettre sa requête. Par exemple : http, ftp, etc.
- public String getContextPath( ) :
  - Retourne sous la forme d'une chaîne de caractères commençant par un /, la portion de l'URL de la requête correspondant au nom du contexte de l'application Web . Par exemple : **/MaWebApp.**
- public String getMethod():
  - Retourne le nom de la méthode HTTP(GET, POST, etc) utilisée par le client pour émettre sa requête.

14- Interfaces ServletRequest et HttpServletRequest (2/7)

Méthodes de récupération d'informations sur l'URL de la requête (suite)

- public String getRequestURL( ) :
  - Retourne l'URL que le client a utilisée pour émettre sa requête. L'URL retournée contient le nom du protocole, le nom du serveur, le numéro de port et le chemin d'invocation de la ressource web, mais pas les paramètres de la chaîne de requête. Par exemple : **http://localhost:8080/servlet/Hello**.

- public String getServletPath( ) :
  - Retourne la partie de l'URL qui invoque la Servlet/JSP, composée du chemin et du nom ou de l'alias de la Servlet/JSP. Par exemple : **/servlet/Hello**.

14- Interfaces ServletRequest et HttpServletRequest (3/7)

Méthodes de récupération d'informations sur le client

- public String getRemoteAddr( ) :
  - Retourne l'adresse IP du client qui a émis la requête. Par exemple : **127.0.0.1**
- public String getRemoteHost( ) :
  - Retourne le nom complet du client qui a émis la requête. Par exemple : **127.0.0.1**
- public String getRemoteUser( ) :
  - Retourne le nom de l'utilisateur qui a envoyé la requête si celui s'est authentifié au préalable, sinon retourne la valeur **null**.

14- Interfaces ServletRequest et HttpServletRequest (4/7)

Méthodes de récupération d'informations sur le serveur

- public String getServerName( ) :
  - Retourne le nom d'hôte du serveur qui a reçu la requête. Par exemple : **loclahost**

- public String getServerPort( ) :
  - Retourne le numéro de port d'écoute du serveur qui a reçu la requête. Par exemple : **8080**

14- Interfaces ServletRequest et HttpServletRequest (5/7)

Méthodes de récupération d'informations dans l'en-tête HTTP

- public String getHeader(String nom) :
  - Retourne la valeur de l'entête nommé, passé en paramètre ou la valeur **null** si l'entête n'existe pas. Le nom de l'entête est sensible à la casse. Par exemple : **getHeader("Accept-Language")** retourne **fr**.
- public java.util.Enumeration getHeaders(String nom) :
  - Retourne sous la forme d'un objet de type java.util.Enumeration l'ensemble des valeurs de l'en-tête de la requête spécifié en paramètre.
- public java.util.Enumeration getHeaderNames() :
  - Retourne sous la forme d'un objet de type java.util.Enumeration l'ensemble des noms des en-têtes contenus dans la requête.

14- Interfaces ServletRequest et HttpServletRequest (6/7)

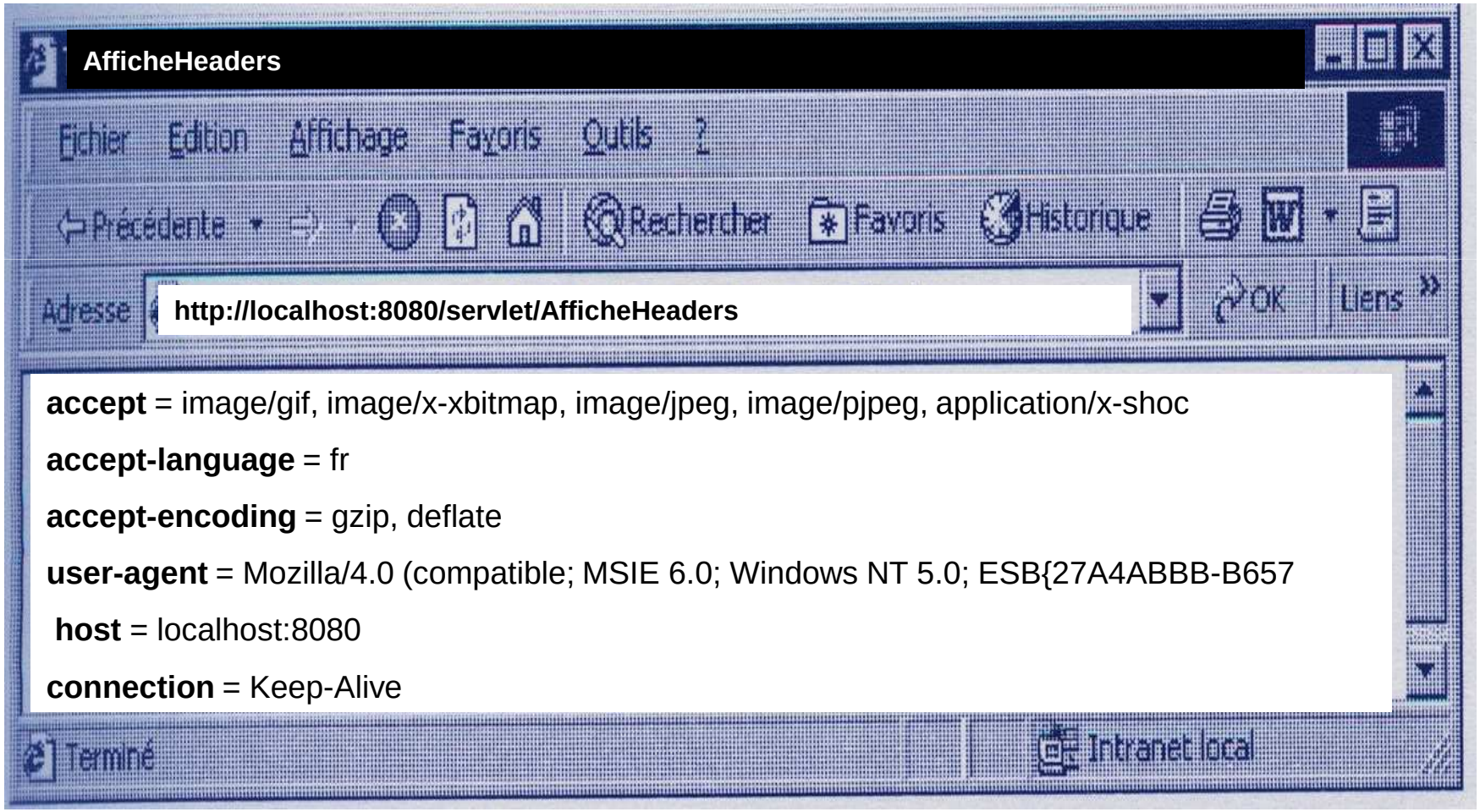
Exemple de manipulation :

AfficheHeaders.java

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

public class AfficheHeaders extends HttpServlet{

public void doGet(HttpServletRequest req,  HttpServletResponse res) throws ServletException, IOException {
 PrintWriter out = res.getWriter();
 res.setContentType("Text/plain");
Enumeration enum = req.getHeaderNames( );

 while (enum.hasMoreElements() ) {
                String headerNom = (String) enum.nextElement();
                out.println(headerNom+" = "+ req.getHeader(headerNom));
        }

  }
}
```

14- Interfaces ServletRequest et HttpServletRequest (7/7)

Exemple de manipulation (suite):



15- Interfaces ServletResponse et HttpServletResponse (1/7)

Méthodes de déclaration du type du contenu et de la taille de la réponse

- public void setContentType(String type) :
  - Spécifier le type MIMIE de contenu du corps de la réponse HTTP. Par exemple text/html pour du HTML, text/plain pour du texte brut, application/pdf pour un document Adobe pdf …

- public void setContentLength(int taille) :
  - Spécifier la taille du contenu de la réponse HTTP. Autrement dit définir l'en-tête HTTP Content-Length.

15- Interfaces ServletResponse et HttpServletResponse (2/7)

Méthodes de renseignement des informations dans l'en-tête HTTP

- public void setHeader(String nom, String Valeur) :
  - Initialiser un en-tête dans la réponse HTTP, avec le nom et la valeur spécifiés en paramètres. Si l'en-tête existe déjà, la nouvelle valeur remplace l'ancienne
- public void addHeader(String nom, String Valeur) :
  - Ajouter un en-tête dans la réponse HTTP, avec le nom et la valeur spécifiés en paramètres. Cette méthode permet à un en-tête d'avoir plusieurs valeurs.
- public boolean containsHeader(String nom) :
  - Retourne un booléen indiquant si un entête existe ou non.

15- Interfaces ServletResponse et HttpServletResponse (3/7)

Méthodes d'envoi d'erreurs et d'états HTTP

- public void sendError(int sc) throws java.io.IOException
- public void sendError(int sc,String message) throws java.io.IOException
  - Envoyer un code d'erreur HTTP au client. Par exemple **SC-NOT-FOUND(404)** ou **SC-SERVICE-UNAVAILABLE(503)**.

- public void sendStatus(int sc) :
  - Appliquer un code d'état à la réponse HTTP quand il n'y a pas d'erreur, comme par exemple **SC-OK(200)** ou **SC-CONTINUE(100)**.

15- Interfaces ServletResponse et HttpServletResponse (4/7)

Méthodes de redirection d'URL

- public void sendRedirect(String url) throws java.io.IOException
  - Envoyer au navigateur du client un ordre de redirection sur une autre ressources Web, qui peut être de la même application Web ou nom.
  - L'URL de la ressources Web passée en paramètre peut être relative ou absolue.
  - Exemple d'URL relative :
    - res.sendRedirect("/MaWebApp/indentification.html")
  - Exemple d'URL absolue :
    - res.sendRedirect("http://www.tunisie.com");

15- Interfaces ServletResponse et HttpServletResponse (5/7)

Méthodes pull client

- Le pull client est similaire à la redirection, avec une différence principale : le navigateur affiche le contenu de la première page et attends un certain temps avant de retrouver et afficher le contenu de la page suivante.

- Utilités :
  - Le contenu de la première page peut expliquer au client que la page demandée a été déplacée avant que la page suivante ne soit automatiquement chargée.
  - Les pages peuvent être retrouvées en séquence, rendant ainsi possible une animation de mouvements de âges lent.

15- Interfaces ServletResponse et HttpServletResponse (6/7)

Méthodes pull client (suite)

- L'information de pull client est envoyée au client via l'en-tête HTTP Refresh.
- La valeur de cet en-tête indique le nombre de secondes pendant lesquelles la page doit être affichée avant d'aller chercher la prochaine et elle peut aussi inclure l'URL indiquant où aller la chercher.
- res.SetHeader("Refresh", "3");
  - Indique au client de recharger la même Servlet après avoir affiché son contenu courant pendant trois secondes
- res.SetHeader("Refresh", "3;URL=http://www.tunisie.com");
  - Indique au client d'afficher la page d'accueil Tunisie après trois secondes.

15- Interfaces ServletResponse et HttpServletResponse (7/7)

Exemple de manipulation : Mise à jour de l'heure courante

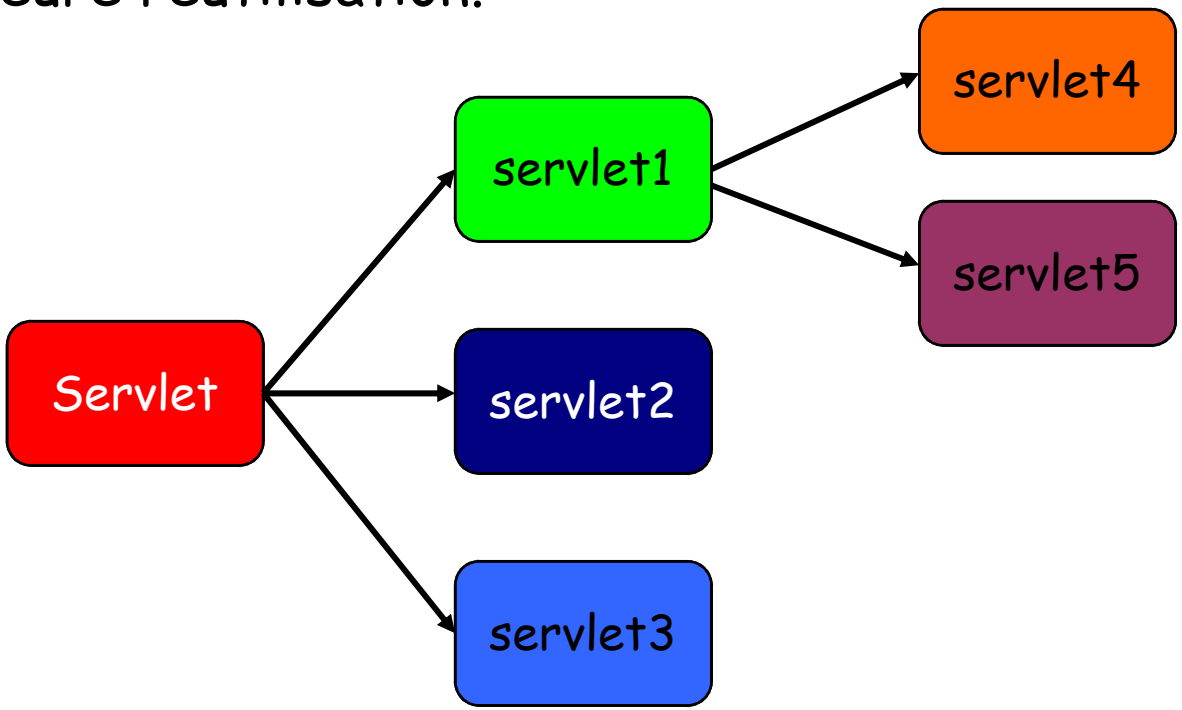
ClientPull.java

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

public class ClientPull extends HttpServlet{

public void doGet(HttpServletRequest req,  HttpServletResponse res) throws ServletException, IOException {
res.setContentType("Text/plain");
PrintWriter out = res.getWriter();

res.setHeader ("Refresh", "60");

out.println(new Date().toString());

  }
}
```

17- Collaboration entre Servlets : L'Interface RequestDispatcher (1/2)

Agrégation de résultats fournis par des Servlets :

- meilleure modularité,
- meilleure réutilisation.



17- Collaboration entre Servlets : L'Interface RequestDispatcher (1/2)

Obtention d'un RequestDispatcher :

- dans la méthode de traitement de requête de Servlet

```
.....
RequestDispatcher rd;
rd = getServeletContext().getRequestDispatcher("/servlet/MaServlet");
if(rd==null) res.sendError(404);
      .....
```

- Redirection d'une requête
  - dans méthode de traitement de requête, demande à une autre Servlet de répondre au client

```
rd.forward(req, res);
```