Institut Supérieur d'Informatique

DÉVELOPPEMENT D'APPLICATIONS WEB JAVA EE AVEC LES SERVLETS ET JSP

Fahem KEBAIR – kebairf@gmail.com

INTRODUCTION

- Expansion d'Internet, évolution des réseaux et diversité des supports informatiques (PC, tablettes, smartphones,…)
- Applications informatiques accessibles via un simple navigateur à travers le web ou un réseau Intranet, on parle donc d'**applications web** ou d'**applications intranet.**
- Ces applications exigent :
  - Traitement dynamique des demandes envoyées par les utilisateurs.
  - Performance et rapidité.
  - Sécurité.
  - Plateforme-indépendante.
  - Interopérabilité.

TECHNOLOGIES POUR LE DÉVELOPPEMENT WEB

Technologies Propriétaires

- *Plateformes* : Microsoft .NET
- *Langages* : C#, VB.net
- *Frameworks* : ASP .NET
- *Serveurs d'applications* : Websphere, Weblogic
- *SGBD* : Oracle, SQLServer, …

Technologies libres

- *Plateformes* : Java EE, JVM, Mono (version open source de Microsoft .NET)
- *Langages* : Java, PHP
- *Frameworks* : Struts 1 et 2, Spring, JSF 1 et 2, GWT, Symphony, Zend, …
- *Serveurs / Conteneurs web* : Apache HTTP, Tomcat, Jetty, …
- *Serveurs d'applications* : JBoss, Glassfish,
- *SGBD* : MySql, PostgreSql, …

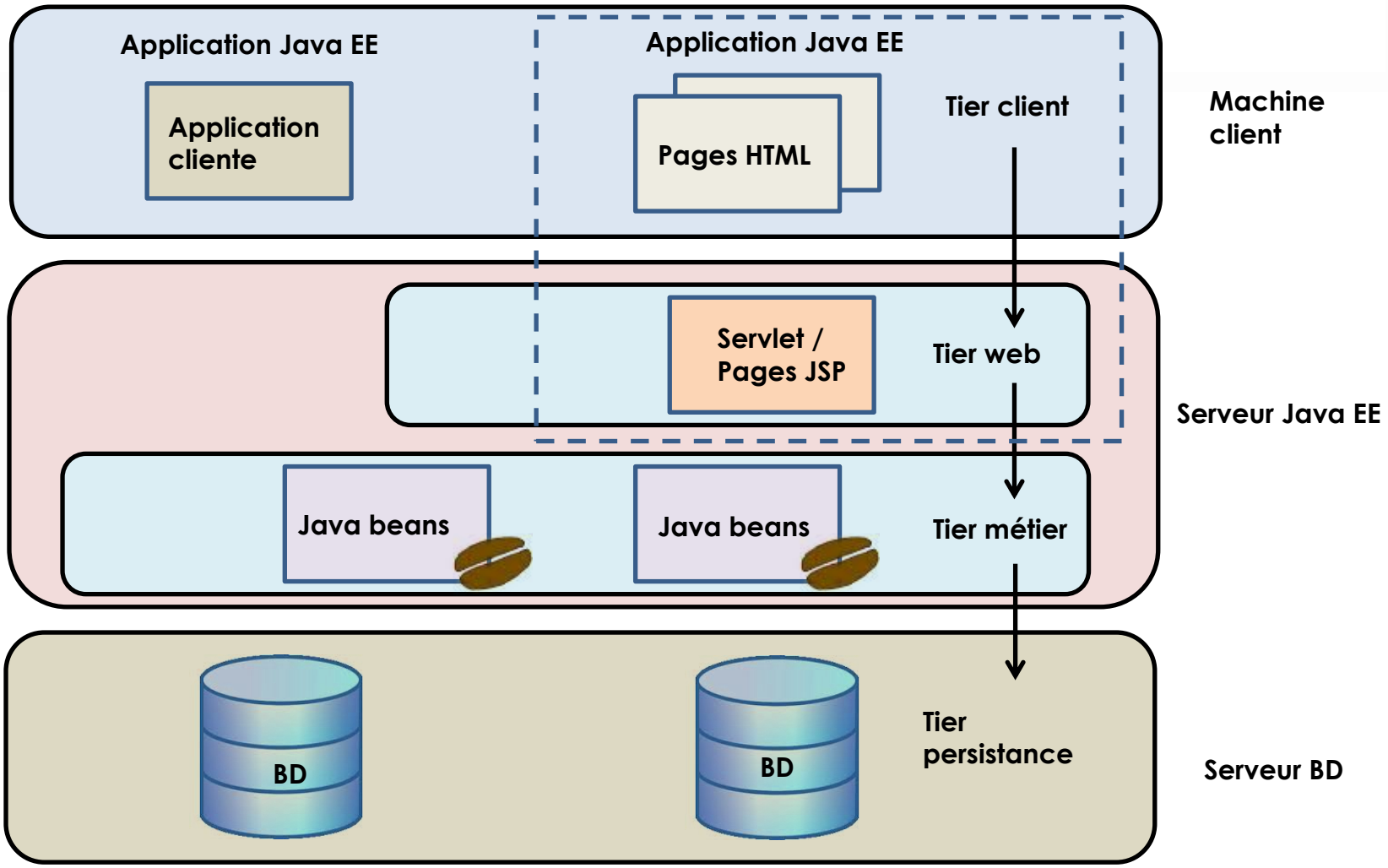
J2EE ET LE WEB DANS TOUT ÇA ?

- Java EE (Java Platform, Entreprise Edition) est une plateforme libre regroupant un ensemble de technologies pour le développement d'applications pour entreprises, reposant sur le langage Java.
- Java EE (avec ses technologies Servlet et JSP) et PHP sont les technologies les plus utilisées dans le développement web.
- Plusieurs frameworks de développement libres en Java ou PHP existent :
  - Java : Struts 1 et 2, JSF 1 et 2, Spring, GWT, Webwork, …
  - PHP : Symphoy, Zend, …

Alors Java/Java EE ou PHP ?

- Tout dépend :
  - *Java EE* : applications de grande taille pour entreprises (ex : système d'information bancaire,…), robustes (profitant de la puissance de Java), sécurisés et demandant des développeurs qualifiés.
  - *PHP* : applications personnelles (ex : sites web, blog, …) ou pour des petites et moyennes entreprises, rapide et facile à mettre en place, demandant des développeurs plus ou moins qualifiés.

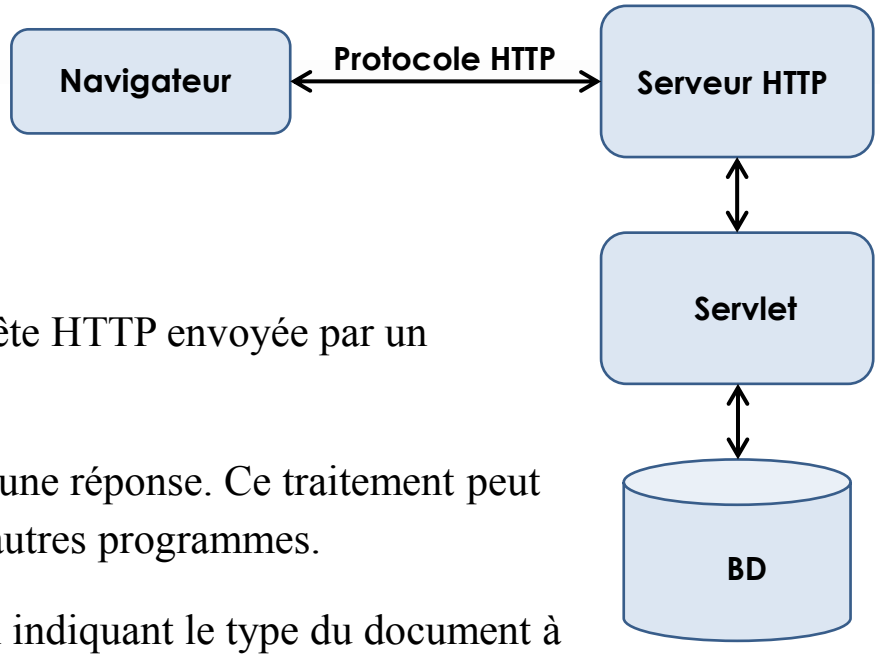
APERÇU SUR LA PLATEFORME JAVA EE



SERVLET
DÉFINITION

- C'est un programme en Java qui traite dynamiquement des requêtes HTTP et fourni en retour des réponses.
- Une servlet est définie par l'interface **javax.servlet.Servlet**. La bibliothèque de la servlet est inclue dans la bibliothèque Java EE SDK et dans le conteneur Tomcat.
- Elle s'exécute dans un conteneur web ou un serveur d'application.
- Elle permet de collecter des entrées client venant d'une page web, présenter des données à partir d'une BD et créer des pages web dynamiques.
- Elle a un objectif semblable à celui de la technologie Common Gateway Interface (CGI), sauf qu'elle offre beaucoup plus d'avantages :
  - Nettement plus performante.
  - S'exécutant dans la même application, sans avoir recours à créer un processus séparé pour traiter chaque requête client.
  - Plateforme-indépendante en tant que programme Java.
  - Sécurisée.
  - Profitant de la totalité des fonctionnalité de la librairie Java, ex : communication avec des applets, BD, autres logiciels via des sockets ou des mécanismes RMI,…

SERVLET
ARCHITECTURE



Fonctionnement

- Lecture des données à partir d'une requête HTTP envoyée par un navigateur client.
- Traitement des données et génération d'une réponse. Ce traitement peut invoquer une BD, un web service ou d'autres programmes.
- Envoi d'une réponse HTTP au client en indiquant le type du document à envoyer au client, ex : document (HTML, XML, binaire, Excel, …).
- Possibilité de remplir des cookies et d'exécuter d'autres actions de ce type.

SERVLET
CYCLE DE VIE

- Un processus qui commence depuis la création de la servlet jusqu'à sa destruction :
  - La servlet est initialisée par une méthode **init()**.
  - La servlet appelle la méthode **service()** pour traiter la requête client.
  - La servlet est détruite par la méthode **destroy()**.
  - Finalement, la servlet est mise à la ramasse miette par la JVM.

La méthode **init()**

- Appelée une seule fois lors du premier appel de la servlet (ou lancement du serveur dans certains cas).
- Effectue un traitement spécifique ou charger certaines données utiles pour le processus.
- Chaque requête d'un client crée un thread.

```
public void init() throws ServletException {
  // Initialization code...
}
```

SERVLET
CYCLE DE VIE

La méthode **service()**

- Méthode principale pour traiter les requêtes clients.
- Elle est appelée par le conteneur, qui appelle par la suite les méthodes doGet, doPost, doDelete, doPut, … selon le type de la requête.
- Les méthodes **doGet** et **doPost** sont les méthodes les plus souvent appelées et sont celles qui vont être surchargées par le développeur.
- La méthode service() prend deux paramètres :
  - ServletRequest : définit un objet contenant la requête client.
  - ServletResponse : définit un objet permettant à la servlet d'envoyer la réponse au client.

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

SERVLET
CYCLE DE VIE

La méthode **doGet()**

- Traite une requête HTTP de type GET.

```
public void doGet(ServletRequest request,
                  ServletResponse response)
      throws ServletException, IOException{
   // Servlet code
}
```

La méthode **doPost()**

- Traite une requête HTTP de type POST.

```
public void doPost(ServletRequest request,
                  ServletResponse response)
      throws ServletException, IOException{
   // Servlet code
}
```

SERVLET
CYCLE DE VIE

La méthode **destroy()**

- Appelée une seule fois à la fin du cycle de vie de la servlet.
- Achève certaines actions comme la fermeture des connections BD.

```
public void destroy(){
  // Finalization code
}
```

SERVLET
ENVIRONNEMENT

- **JRE et JDK de Java** : la machine virtuelle Java et le kit de développement Java pour le développement et l'exécution de toute application Java, qui sont libres et gratuits, http://www.oracle.com/technetwork/java/javase/downloads/index.html
- **IDE Eclipse pour les développeurs Java EE** : IDE libre et gratuit pour le développement de Java/Java EE, https://eclipse.org/downloads/
- **Tomcat** : conteneur de servlet libre et gratuit pour le déploiement des applications web Java EE, http://tomcat.apache.org/download-80.cgi

SERVLET
PREMIER EXEMPLE

Création d'un projet web dynamique avec Eclipse

- Il faut au préalable ajouter le serveur tomcat dans Eclipse : **Window**, **Preferences**, **Server**, **Runtime Environments**, **Add**, Choisir le plugin Apache qui correspond à la version téléchargée et spécifier le répertoire contenant ton serveur sur votre disque dur.
- Ouvrir Eclipse, puis **File**, **New**, **Dynamic Web Project**.
- Spécifier le nom du projet « helloworld », **Next**, **Next**, cocher **Generate web.xml**, **Finish**.

Ajout d'une servlet :

- Ajout d'une servlet permettant l'affichage d'un texte helloworld.
- Deux méthodes : en créant une classe (méthode qu'on adopte) ou une servlet

1. Ouvrir le projet, Java Resources, **src**, **New**, **Class**, Nommer la classe **HelloWorldServlet**.
2. Ouvrir le projet, Java Resources, **src**, **New**, **Servlet**, Nommer la classe **HelloWorldServlet**, vous pouvez éventuellement spécifier un package, par exemple **hellopackage**. **Next** (vous affiche le mapping de votre servlet /HelloWorldServlet), **Next** (vous pouvez cochez les méthodes que vous voulez générer comme doGet et doPost), **Finish**.

SERVLET
PREMIER EXEMPLE – ÉCRITURE DE LA SERVLET

- Une servlet est une classe java qui étend la classe **javax.servlet.HttpServlet.**

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

public class HelloWorldServlet extends HttpServlet {

  private String message;

  public void init() throws ServletException {
      message = "Hello World";
  }

  public void doGet(HttpServletRequest request,  HttpServletResponse
                response)throws ServletException, IOException  {

      // envoyer un contenu HTML à un client web
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<h1>" + message + "</h1>");
  }

}
```

SERVLET
PREMIER EXEMPLE – LE FICHIER WEB.XML

Structure de base d'un fichier web.xml

- Le fichier web.xml, situé dans le répertoire WebContent/WEB-INF est un fichier XML de déploiement qui décrit les classes, les ressources et la configuration d'une application web dans le serveur Tomcat et la manière avec laquelle une servlet peut être atteinte à travers une URL.
- <servlet> : déclare la servlet, comporte les éléments :
  - <servlet-name> : nom de la servlet
  - <servlet-class> : classe définissant la servlet
- <servlet-mapping> : mappe une URL à une servlet, comporte les éléments :
  - <servlet-name> : nom de la servlet définie auparavant
  - <url-pattern> : modèle de l'URL appelant la servlet. Le modèle peut utiliser un asterisk (*) au début ou à la fin du modèle pour indiquer 0 ou plusieurs caractères.

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="3.0">
    <servlet>
        <servlet-name>helloworld</servlet-name>
        <servlet-class>hellopackage.HelloWorldServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>helloworld</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

La servlet est accessible de cette manière :
http://hote:8080/nom_contexte
non_contexte : nom du WAR déployé

SERVLET
PREMIER EXEMPLE – DÉPLOIEMENT DE L'APPLICATION

Le conteneur web Tomcat

- Apache Tomcat est un conteneur web libre et gratuit permettant l'exécution d'applications à base de servlets et de pages JSP.
- Tomcat fonctionne par défaut sur le port 8080 (**http://localhost:8080**), il peut être aussi utilisé en tant que serveur HTTP, sauf qu'il n'est pas aussi performant que le serveur HTTP Apache. Les deux peuvent fonctionner ensemble en ajoutant un connecteur.

Administration de Tomcat

- Tomcat possède une interface d'administration accessible via : http://localhost:8080/manager/html
- Les utilisateurs de Tomcat se trouvent dans le fichier conf/tomcat-users.xml.
- Un administrateur doit avoir le rôle manager-gui, un utilisateur est défini ainsi :

```
<role rolename="manager-gui"/>
<user username="toto" password="secret" roles="manager-gui" >
```

SERVLET
PREMIER EXEMPLE – DÉPLOIEMENT DE L'APPLICATION

- Deux façons pour déployer l'application dans le serveur Tomcat :
  - *Déploiement à l'intérieur d'eclipse* : en phase de développement.
  - *Déploiement à l'intérieur Tomcat* : en phase de production.

Déploiement à l'intérieur d'eclipse

- Ajouter un serveur pour le déploiement dans eclipse : **clic droit**, **New**, **Server**, **choisir Apache Tomcat**, **Next**, **Ajouter l'application**, **Finish**.
- Démarrer le serveur : **clic droit**, **start**.

Déploiement à l'intérieur de Tomcat

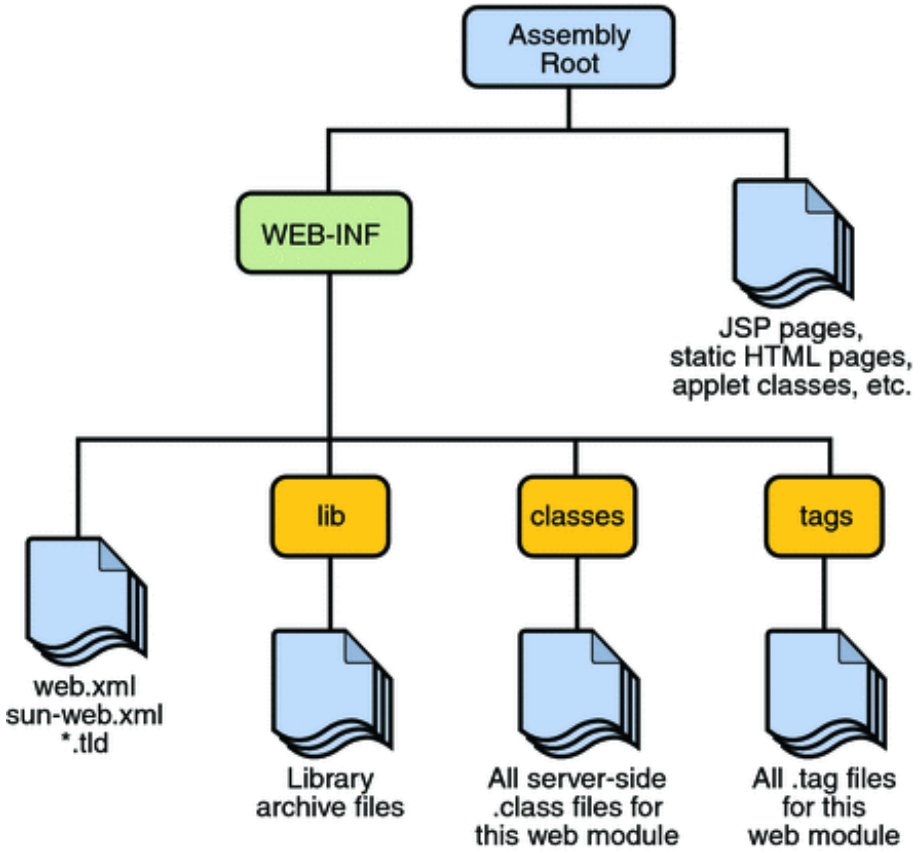
- Exporter le projet sous forme d'une archive WAR : **clic droit** sur le projet, **export**, spécifier le nom et la destination. Mettre le .war doit dans le dossier webappas du répertoire de Tomcat.
- Démarrer Tomcat : lancer le script startup.bat (windows) ou startup.sh (linux) dans le répertoire bin de Tomcat.
- Le .war peut être aussi déployé avec le gestionnaire de Tomcat : **http://localhost:8080/manager/html**

SERVLET
L'ARCHIVE WAR

- L'archive WAR (Web Application Archive) est appelé **module de web**, qui est la plus petite unité de déploiement dans Java EE. Cette unité représente l'application web.
- Le module web contient des ressources web : servlets et pages jsp compilés et des fichiers statiques tels que des fichiers des scripts JS, CSS, des librairies JAR, ….

La structure d'un module web est composée d'une racine root dans lequel se trouvent les pages JSP et les fichiers statiques et d'un répertoire WEB-INF contenant les repertoires suivants :

- web.xml : le descripteur de déploiement de l'application
- Des fichiers descripteurs de la librairie de tags
- classes : les fichiers compilés .class des servlets, des classes Java et des Java Beans
- tags : des fichiers de tag qui sont des implémentations des librairies des tag, permettant de définir des composants web
- lib : les archives JAR des librairies appelées par les classes



SERVLET
LES FORMULAIRES

- Deux méthodes HTTP utilisées par le navigateur permettant l'envoi des données au serveur : GET et POST.

Méthode GET

- Envoi les données de l'utilisateur en les rattachant à la requête de la page. La page et les données sont séparées par le caractère « ? », comme suit:
  `http://www.test.com/hello?key1=value1&key2=value2`
- Méthode par défaut utilisée dans les formulaires qui produit une longue chaine de caractères visible dans l'emplacement de l'URL du navigateur. Il est donc formellement interdit d'utiliser GET si on envoi un mot de passe ou d'autres informations critiques.
- Limité en taille de sa requête avec seulement 1024 caractères.

Méthode POST

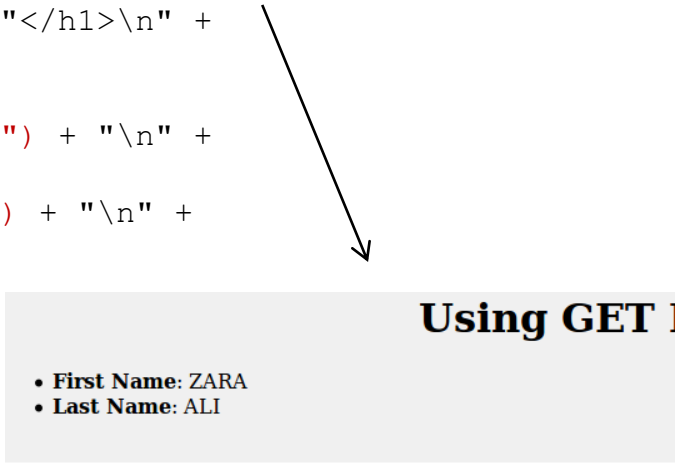
- Plus générale et efficace pour passer des informations au serveur.
- Fonctionne comme GET, sauf que les données sont envoyées en tant que message séparé qui peut être parsé et traité côté serveur.
- Utile pour envoyer des données critiques et n'est pas limitée en taille.

SERVLET
LES FORMULAIRES – EXEMPLE AVEC GET

- url : http://localhost:8080/helloform?first-name=ZARA&last_name=ALI
- Utilisation de la méthode **getParameter("paramName")** pour récupérer les données envoyés. La servlet HelloForm.java est la suivante :

```
public class HelloForm extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
           String title = "Using GET Method to Read Form Data";
      String docType = "<!doctype>\n";
      out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body bgcolor=\"#f0f0f0\">\n" +
                "<h1 align=\"center\">" + title + "</h1>\n" +
                "<ul>\n" +
                "  <li><b>First Name</b>: "
                + request.getParameter("first_name") + "\n" +
                "  <li><b>Last Name</b>: "
                + request.getParameter("last_name") + "\n" +
                "</ul>\n" +
                "</body></html>");
  }
}
```



SERVLET

LES FORMULAIRES – EXEMPLE AVEC GET AVEC UN FORMULAIRE

- Définir un fichier hello.html :

```
<html>
<body>
<form action="HelloForm" method="GET">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

- La page hello.html est accessible via : http://localhost:8080/non_contexte/hello.html
- Un click sur le bouton submit va envoyer une requête au serveur qui sera captée par la servlet. Le résultat est identique à celui de l'exemple précédent.
- L'attribut *action* doit correspondre au nom de la servlet appelée (celui du mapping définit dans web.xml)

SERVLET
LES FORMULAIRES – EXEMPLE AVEC POST

- Une servlet traitant à la fois les méthodes GET et POST :

```
public class HelloForm extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Using GET Method to Read Form Data";
      String docType =
      "<!doctype !>\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<ul>\n" +
        "  <li><b>First Name</b>: "
        + request.getParameter("first_name") + "\n" +
        "  <li><b>Last Name</b>: "
        + request.getParameter("last_name") + "\n" +
        "</ul>\n" +
        "</body></html>");
  }
public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
  }
}
```

Requête HTTP avec une méthode POST

```
<html>
<body>
<form action="HelloForm"
          method="POST">
First Name: <input type="text"
name="first_name">
<br />
Last Name: <input type="text"
name="last_name" />
<input type="submit"
          value="Submit" />
</form>
</body>
</html>
```

Une requête envoyée à partir de ce formulaire sera captée par la méthode **doPost** de la servlet. Le résultat est identique à celui de l'exemple précédent.

SERVLET
LES FORMULAIRES – LECTURE DE TOUS LES PARAMÈTRES

- Récupérer tous les paramètres de la requête avec la méthode **getPrametersNames()**, qui retourne un énumérateur contenant tous les paramètres et leurs valeurs.

```
Enumeration paramNames = request.getPrameterNames();
While(paramNames.hasMoreElements())
{
      // Récupérer un paramètre
      String paramName = (String)paramNames.nextElement();

      // Récupérer toutes les valeurs du paramètre
      String[] paramValues = request.getParameterValues(paramName);

      // Lire une seule valeur
      if (paramValues.length == 1)
         String paramValue = paramValues[0];
      else if (paramValues.length == 0)
         // pas de valeurs
      else // plusieurs valeurs, cas d'un checkbox par exemple
         for (int i=0; i<paramValues.length; i++)
            String paramValue = paramValues[i];
}
```

SERVLET
LES ATTRIBUTS

- Un attribut dans une servlet est un objet qui peut être édité, lu ou supprimé à partir de l'une des portées suivantes :
  - portée d'une requête
  - portée d'une session
  - portée d'une application
- Les informations peuvent ainsi être passés d'une servlet à une autre en utilisant les attributs. Ceci est similaire au passage d'un objet d'une classe à une autre.

Gestion d'un attribut à l'interieur d'une servlet : Accès à une application

```
// Servlet qui édite l'attribut dans l'application
public class ServletEditor extends HttpServlet {
public void doGet(HttpServletRequest req,HttpServletResponse res) {

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

   ServletContext context = getServletContext();
   context.setAttribute("company","IBM");

   out.println("Ce lien nous mène vers une deuxième servlet");
   out.println("<a href='servlet2'>visit</a>");
}}
```

Dans cet exemple on édite un attribut dans la portée de l'application et on récupère sa valeur dans une autre servlet
ServletContext : le context de la servlet
setAttribute(nam, value) : éditer un attribut
getAttribute(name) : récupérer un attribut

SERVLET
LES ATTRIBUTS

Gestion d'un attribut à l'intérieur d'une servlet : : Accès à une application (suite)

```
// Servlet qui lit l'attribut à partir de l'application
public class ServletReader extends HttpServlet {
public void doGet(HttpServletRequest req,HttpServletResponse res) {

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

   ServletContext context = getServletContext();
   String n = (String)context.getAttribute("company");

   out.println(The company is" + n);
}}
```

Dans cet exemple on édite un attribut dans la portée de l'application et récupérer sa valeur d'une servlet à une autre
ServletContext : le context de la servlet
setAttribute(nam, value) : éditer un attribut
getAttribute(name) : récupérer un attribut

SERVLET
LES ATTRIBUTS

Gestion d'un attribut à l'intérieur d'une servlet : Accès à une session

```
// Servlet qui gère un attribut à l'intérieur d'une session
public class ServletEditor extends HttpServlet {
public void doGet(HttpServletRequest req,HttpServletResponse res) {

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

   // récupération d'une session
   HttpSession session = req.getSession();
   // création et édition d'un attribut
   Personne p = new Personne("toto");
   session.setAttribute("personne",p);
   // lecture d'un attribut
   Personne personne = (Personne)session.getAttribute("personne");

}}
```

Dans cet exemple on édite un attribut dans la portée de la session et le récupérer
HttpSession: la session de la requête
setAttribute(name, value) : éditer un attribut
getAttribute(name) : récupérer un attribut

SERVLET
REDIRECTION DES REQUÊTES

Deux façons pour rediriger une requête

- Transférer la même requête vers une autre servlet. Cette méthode ne fonctionne que côte serveur, elle est complètement opaque pour le navigateur.
- Envoyer une nouvelle requête vers une autre servlet ou une page web. Cette méthode peut fonctionner côté client ou côté serveur, elle est complètement transparente pour le navigateur. Il s'agit de définir une nouvelle URL, donc exécuter une nouvelle requête.

Redirection d'une requête avec *forward()*

- *forward(ServletRequest request, ServletResponse response)* est une méthode de l'interface *RequestDispatcher*.
- Elle permet de rediriger **une même requête** d'une servlet à une autre au sein du même serveur.
- Exemple :

```
public class HelloForm extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
  request.getRequestDispatcher("servlethandler").forward(request,response);
}
```

Ici, c'est une autre servlet ServletHandler (mappée par /servlethandler dans le web.xml) qui traitera les données de la requête et affichera le résultat.

SERVLET
REDIRECTION DES REQUÊTES ET NAVIGATION

Redirection d'une requête avec *forward() (suite)*

```
public class ServletHandler extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Using GET Method to Read Form Data";
      String docType =
      "<!doctype !>\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<ul>\n" +
        "  <li><b>First Name</b>: "
        + request.getParameter("first_name") + "\n" +
        "  <li><b>Last Name</b>: "
        + request.getParameter("last_name") + "\n" +
        "</ul>\n" +
        "</body></html>");
  }
```

SERVLET
REDIRECTION DES REQUÊTES ET NAVIGATION

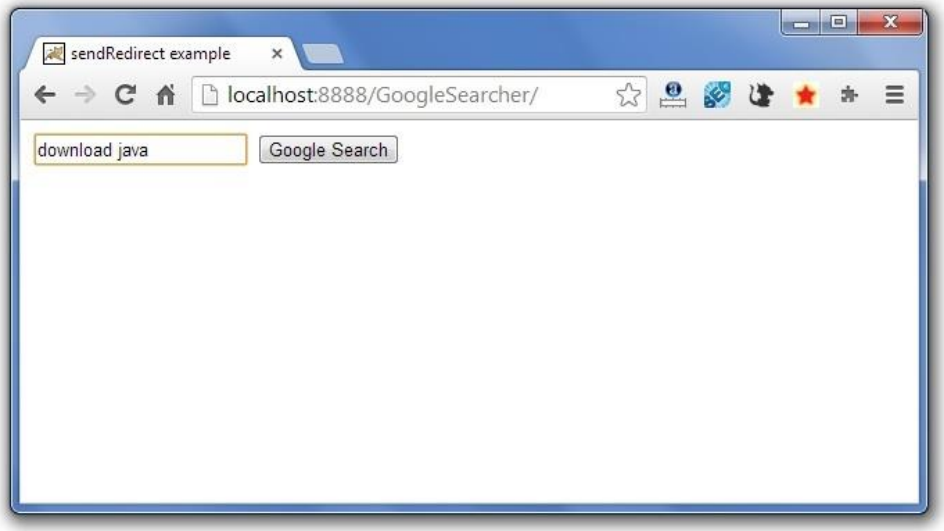
Redirection d'une requête avec *senRedirect()*

- *sendRedirect(String url)* est une méthode de l'interface *HttpServletResponse*.
- Elle permet de rediriger la requête client vers une autre ressource sur le même serveur ou dans un autre. Il s'agit d'envoyer une nouvelle requête avec une adresse visible via le navigateur.
- Exemple :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>sendRedirect example</title>
</head>
<body>

<form action="MySearcher">
<input type="text" name="name">
<input type="submit" value="Google Search">
</form>

</body>
</html>
```



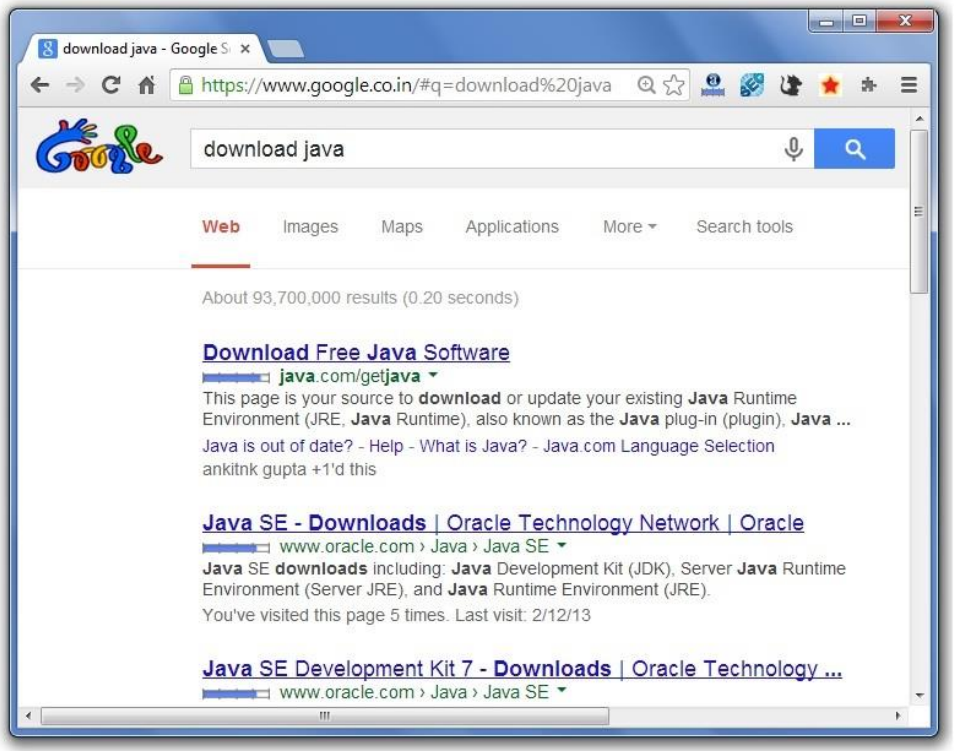
Cette page va envoyer une requête à la servlet « MySearcher » contenant un mot à chercher. La requête se charge à son tour d'exécuter une nouvelle requête qui invoque google afin de chercher le mot correspondant.

SERVLET
REDIRECTION DES REQUÊTES ET NAVIGATION

Redirection d'une requête avec *senRedirect() (suite)*

```
public class MySearcher extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {

        String name=request.getParameter("name");
         response.sendRedirect("https://www.google.co.in/#q="+name);
    }
}
```



JAVA SERVER PAGES
DÉFINITION

Définition

- Java Server Pages (JSP) est une technologie de programmation côté serveur permettant le développement d'applications web dynamiques.
- JSP permet le développement des pages web qui supporte l'integration du code java dans une page HTML grâce à des balises JSP, dont la plupart commencent par <% et se clôturent par %>.

Avantages de JSP

- JSP est plus performant que la technologie CGI (Common Gateway Interface) en intégrant des éléments dynamiques dans des pages HTML au lieu de fichiers séparés.
- JSP sont souvent compilés avant son traitement par le serveur, alors que la CGI/Perl nécessite un interpréteur chaque fois que la page est requise.
- JSP, tout comme les servlets, ont accès aux APIs de Java y compris JDBC, JNDI, EJB, JAXB, …
- JSP est portable et peut fonctionner sur tous les OS

Compilation

- Une page JSP est parsée et convertie en une servlet, pour le premier appel de la page ou lorsqu'elle a été modifiée, qui sera par la suite compilée.

JAVA SERVER PAGES
SYNTAXE

Scriplète

- Une scriplète peut contenir toutes les instructions, variables, méthodes ou expressions Java. Elle est entourée par <% et %>.

Exemple :

```
<html>
<head><title>Hello World</title></head>
<body>
Hello World!<br/>
<%
out.println("Your IP address is " + request.getRemoteAddr());
%>
</body>
</html>
```

Déclaration

- Une déclaration déclare une ou plusieurs variables ou méthodes.

Exemple :

```
<%! int i=0; %>
<%! Circle a = new Circle(2,0);%>
```

JAVA SERVER PAGES
SYNTAXE

Expression

- Une expression JSP contient une valeur d'expression de langage qui est converti en un String et inséré dans le flux de données retourné au client.
- *Syntaxe* : `<%= expression %>`

Exemple :

```
<html>
<head><title>A Comment Test</title></head>
<body>
<p>
   Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
</body>
</html>
```

Commentaire

- Texte ignoré par le conteneur

Exemple :

```
<% this content will not be visible in the page source %>
```

JAVA SERVER PAGES
SYNTAXE

Directive

- Une directive JSP permet de préciser des informations globales sur la page JSP.
- *Syntaxe* : `<%@ directive attribute="value" %>`
- Il existe trois types de directives :
  - `<%@ page …%>` : définit des options qui s'appliquent à la toute la JSP.

  Exemple : `<%@ page import="java.util.*"%>`
  - `<%@include ...%>` : inclut un fichier (Fragment de code JSP, HTML ou Java) dans le code source JSP. Cet élément est utile pour insérer un élément commun à plusieurs pages tels qu'un entête ou un bas de page.

  Exemple : `<%@include file="bonjour.html"%>`
  - `<%@taglib ...%> : déclare l'utilisation d'une bibliothèque de tags personnalisés.`

JAVA SERVER PAGES
SYNTAXE – FLUX DE CONTRÔLE

Conditions

```
<%! int day = 3; %>
<html>
<head><title>IF...ELSE Example</title></head>
<body>
<% if (day == 1 | day == 7) { %>
      <p> Today is weekend</p>
<% } else { %>
      <p> Today is not weekend</p>
<% } %>
</body>
</html>
```

Boucles

```
<%! int fontSize; %>
<html>
<head><title>FOR LOOP Example</title></head>
<body>
<%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
   <font color="green" size="<%= fontSize %>">
    JSP Tutorial
   </font><br />
<%}%>
</body>
</html>
```

JAVA SERVER PAGES
EXEMPLE JSP / SERVLET

- Dans ce exemple, une page JSP contenant un formulaire va demander à une servlet d'afficher la couleur saisie.

Page JSP

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
  <form action="HelloServlet">
     Please enter a color <br>
     <input type="text" name="color"size="20px">
     <input type="submit" value="submit">
  </form>
</body>
</html>
```

Le nom de l'action permet d'indiquer la servlet à exécuter, ceci devra correspondre au mapping défini dans le fichier web.xml.
L'url aura donc cette forme :
http://localhost:8080/helloapp/HelloServlet?color=blue

JAVA SERVER PAGES
EXEMPLE JSP / SERVLET

Servlet

```
public class HelloForm extends HttpServlet {

  public void doGet(HttpServletRequest request,
       HttpServletResponse response)throws ServletException, IOException{

          String color= request.getParameter("color");
          PrintWriter out = response.getWriter();
          out.println ("<!DOCTYPE>\n"
          + "<html> \n"
           + "<head> \n"
          + "<title> My first jsp </title> \n"
          + "</head> \n"
           + "<body> \n"
          + "<font size=\"12px\" color=\"" + color + "\">"
          + "Hello World"
         + "</font> \n"
          + "</body> \n"
          + "</html>" );
  }
}
```

JAVA SERVER PAGES
EXEMPLE JSP / SERVLET

Fichier de déploiement web.xml

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="3.0">
    <welcome-file-list>
       <welcome-file> hello.jsp </welcome-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>helloform</servlet-name>
        <servlet-class>HelloForm</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>helloform</servlet-name>
        <url-pattern>/HelloServlet</url-pattern>
    </servlet-mapping>
</web-app>
```