Développement web avancé – Java EE

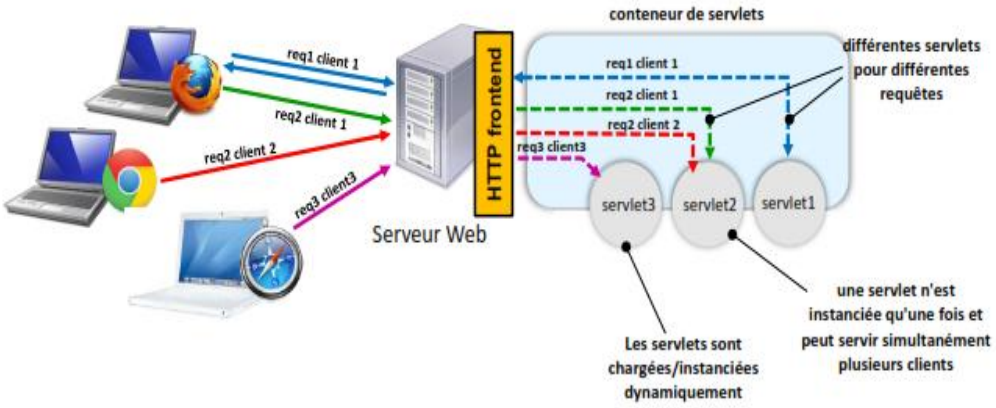
Génie Informatique – Semestre S6

Architecture Client serveur et Servlets (Suite)

1. Rappel

Les Servlets génèrent des pages web dynamiques (comme PHP, ASP, ASP.NET, ...). De manière plus générale n'importe quel contenu web dynamique (HTML, XML, JSON, etc …). Elles s'exécutent dans des serveurs dédiés : Servlets containers

Un conteneur de servlets gère les échanges avec le client (protocole HTTP), aiguille les requêtes vers les servlets, charge/décharge les servlets. Les servlets sont instanciées une seule fois, ensuite le traitement des requêtes s'effectue de manière concurrente dans des fils d'exécution (threads) différents (Voir la Figure ci-dessous).



Il existe différents types de conteneur de servlets

- conteneurs légers : Apache **Tomcat**, Jetty, Resin
- conteneurs Java EE complets : Glassfish, WildFly, anciennement JBoss (RedHat), WebSphere (IBM)….

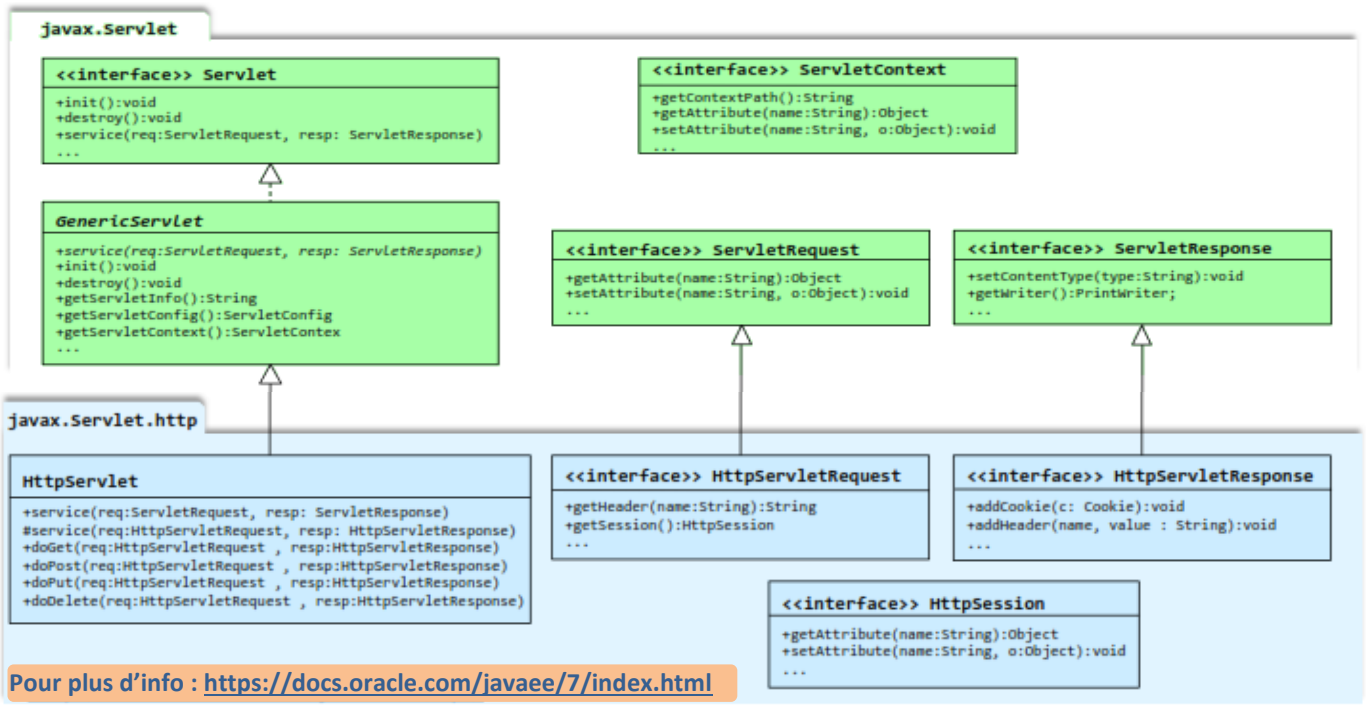
Java assure la portabilité d'un conteneur à un autre

Le tableau suivant donne une vue globale sur les différentes versions de l'API servlet.

Version	Date de sortie	Plateforme
Servlet 4.0	sept-17	JavaEE 8
Servlet 3.1	mai-13	JavaEE 7
Servlet 3.0	déc-09	JavaEE 6, JavaSE 6
Servlet 2.5	sept-05	JavaEE 5, JavaSE 5
Servlet 2.4	nov-03	J2EE 1.4, J2SE 1.3
Servlet 2.3	Aout 2001	J2EE 1.3, J2SE 1.2
Servlet 2.2	Aout 1999	J2EE 1.2, J2SE 1.2
Servlet 2.1	nov-98	--
Servlet 2.0	--	--
Servlet 1.0	juin-97	--

Documentation API Java EE7 : **http://docs.oracle.com/javaee/7/api/**

Documentation API Java EE8 : https://javaee.github.io/javaee-spec/javadocs/

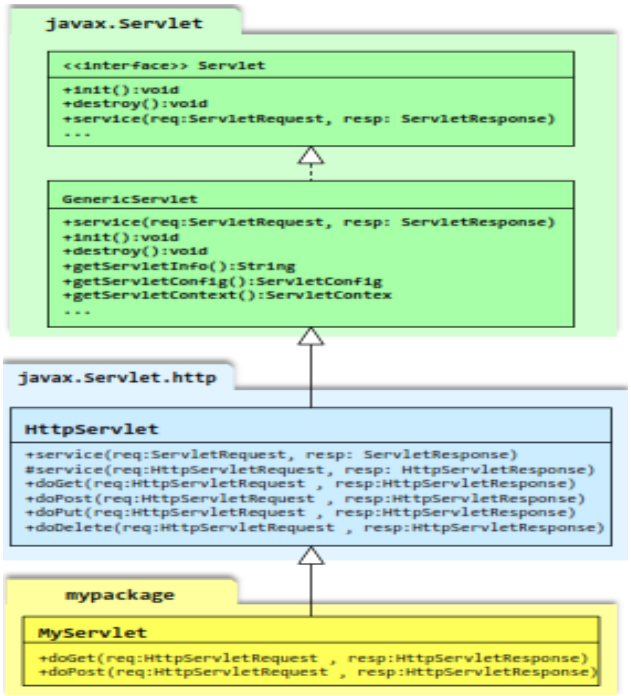


Il existe deux types de servlets

- Les **GenericServlet** qui ne pré-suppose pas d'un protocole
- Les **HttpServlet** qui repondent à des clients par le protocole HTTP

GenericServlet est une classe du paquetage **javax.servlet** tandis que HttpServlet est une classe du paquetage **javax.servlet.http.**

Définir une Servlet consiste à sous classer HttpServlet et redéfinir une ou plusieurs des méthodes doXXX



Cycle de vie géré par le serveur (container)

- Initialisation :
  - Chargement de la classe (au démarrage ou sur requête).

- Instanciation d'un objet.
- Appel de la méthode init() (par exemple : ouverture de lien JDBC)

- Utilisation :
  - Appel de la méthode service()
  - Dans le cas d'une HttpServlet : appel vers doGet(), doPost().
- Destruction :
  - Peut être provoquée pour optimiser la mémoire
  - appel de la méthode destroy()

Les servlets : structure, mise en œuvre et méthodes invocation

1. Structure d'une servlet

Prenant cet exemple d'une servlet basique :

```
package sadiq.test.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
 //annotation  définissant  l'url associée à cette servlet (servlet Mapping)
 //ces infos étaient dans un fichier de configuration web.xml
@WebServlet(name = "Test2", urlPatterns = {"/Test2"})
public class Test2 extends HttpServlet {

 //redéfinition de la méthode doGet traitant les requêtes HTTP GET.
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

 //construction de la réponse à la requête en utilisant l'objet PrintWriter
//fournit par le paramètre HttpServletResponse.
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
          try {
          out.println("<html>");
          out.println("<head>");
          out.println("<title>Servlet HelloWorldServlet</title>");
          out.println("</head>");
          out.println("<body>");
          out.println("<h1>Hello World!!</h1>");
          out.println("<h2>Reponse envoyée le " + new Date() + "</h2>");
          out.println("</body>");
          out.println("</html>");
          } finally {
                out.close();
                }
}
}
```

2. Invocation d'une servlet

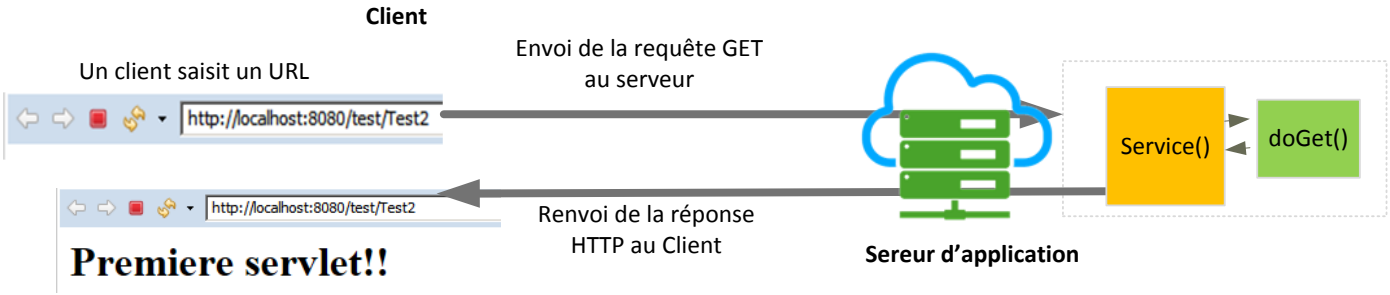
Pour accéder à une servlet on utilise un URL contentnat un contexte un un pattern

http://localhost:8084/ServletsDemos/ Test2

protocole	Serveur	contexte de la servlet(nom de l'application web sur le serveur*)	url pattern pour la servlet (défini par le servlet mapping **)

*** un serveur peut héberger plusieurs applications**

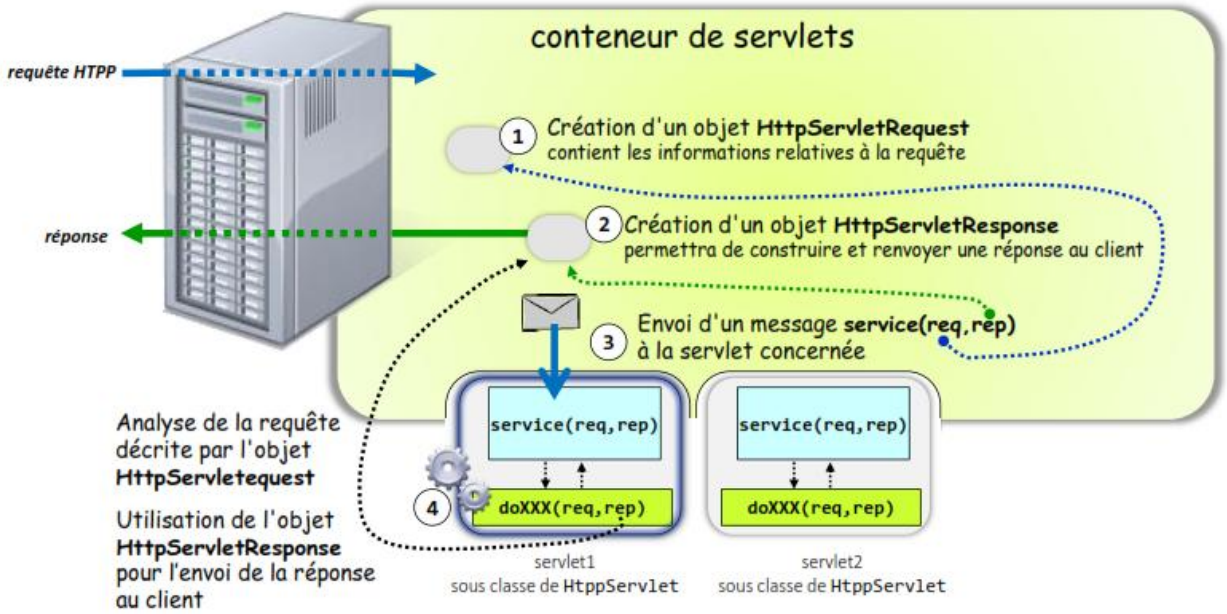
**** une application peut être composée de plusieurs servlets**



3. objets HttpServletRequest et HttpServletResponse

Les objets HttpServletRequest et HttpServletResponse doivent être de passés en paramètre des méthodes service() , doGet() , doPost() , doXXX()…

Ils sont instanciés par le conteneur de servlets, qui les transmet à la méthode service.



L'objet HttpServletRequest permet de gérer:

- Contexte de l'appel
- Paramètres de formulaires
- Cookies
- Headers
- . . .

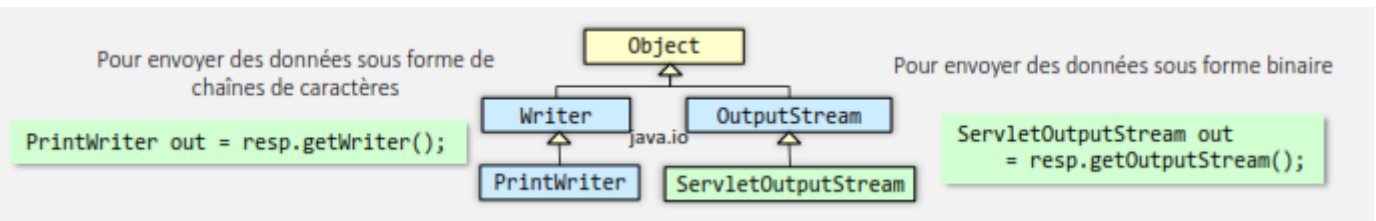
L'objet HttpServletResponse permet de gérer:

- Contrôle de la réponse
- Type de données
- Données
- Cookies
- Status
- . . .

Un pattern classique pour l'implémentation d'une méthode service (doXXX) d'une servlet est le suivant:

1. Extraire de l'information de la requête (paramètre request : HttpervletRequest)
2. Accéder éventuellement à des ressources externes
3. Produire une réponse sur la base des informations précédentes

a) Récupérer un flux de sortie pour l'écriture de la réponse (à l'aide du paramètre request : HttpervletRequest)



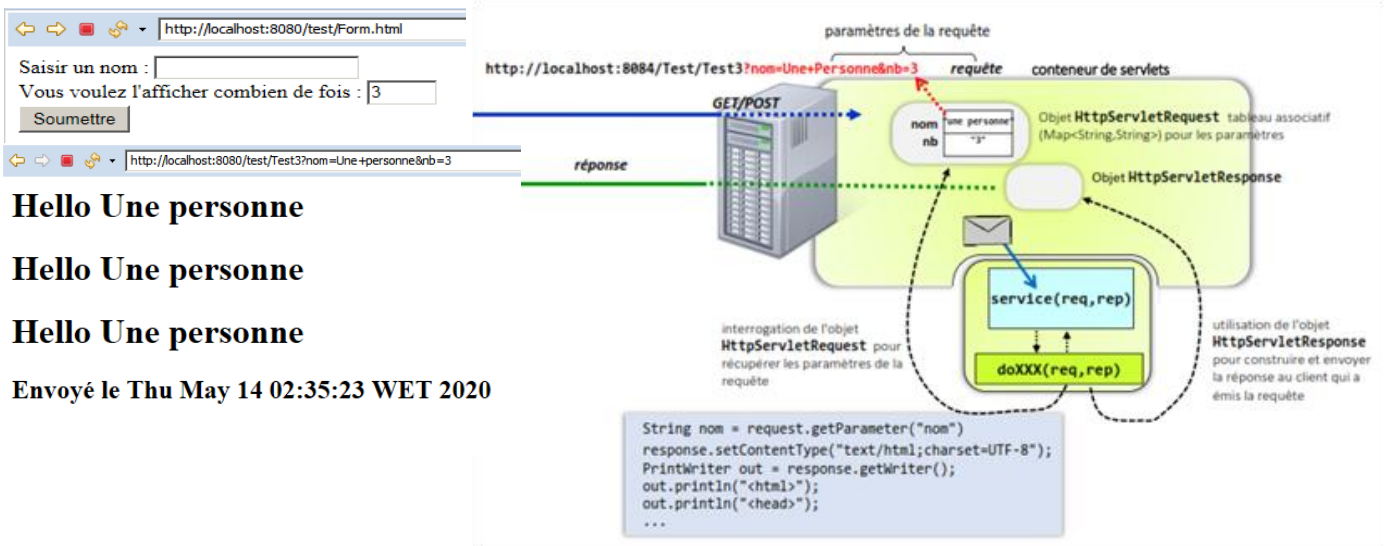
b) Définir les entêtes (HTPP headers) de la réponse

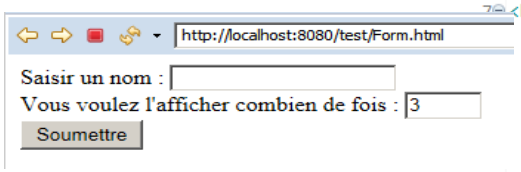
```
resp.setContentType("text/html");
```
indiquer le type du contenu
```
resp.setContentType("image/png");
resp.setContentType("application/pdf");
```
Types MIME (Multipurpose Internet Mail Extensions)
http://www.iana.org/assignments/media-types/

c) Ecrire le contenu (corps) de la réponse sur le flux de sortie

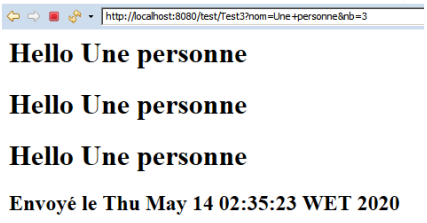
```
out.println("<p>blabla...</p>");
out.print(...)
```
directement avec l'objet out
```
byte[] tab = ...;
...
out.write(tab);
out.print(...);
```
en passant par API spécialisées
ex: iText (http://itextpdf.com/) pour pdf
http://www.ibm.com/developerworks/java/library/os-javapdf/index.html?ca=dat
```
Document document = new Document() ;
PdfWriter.getInstance(document,out) ;
```

En se basant sur ce principe nous allons créer une première Servelt pour la gérer la soumission d'un formulaire






```html
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Mon formulaire</title>
</head>
<body>
    <form action="Test3">
        Saisir un nom :
        <input type="text" name="nom" value="" size="20" /><br/>
            Vous voulez l'afficher combien de fois :
        <input type="text" name="nb" value="3" size="3" /><br/>
        <input type="submit" value="Soumettre" />
    </form>
</body>
</html>
```

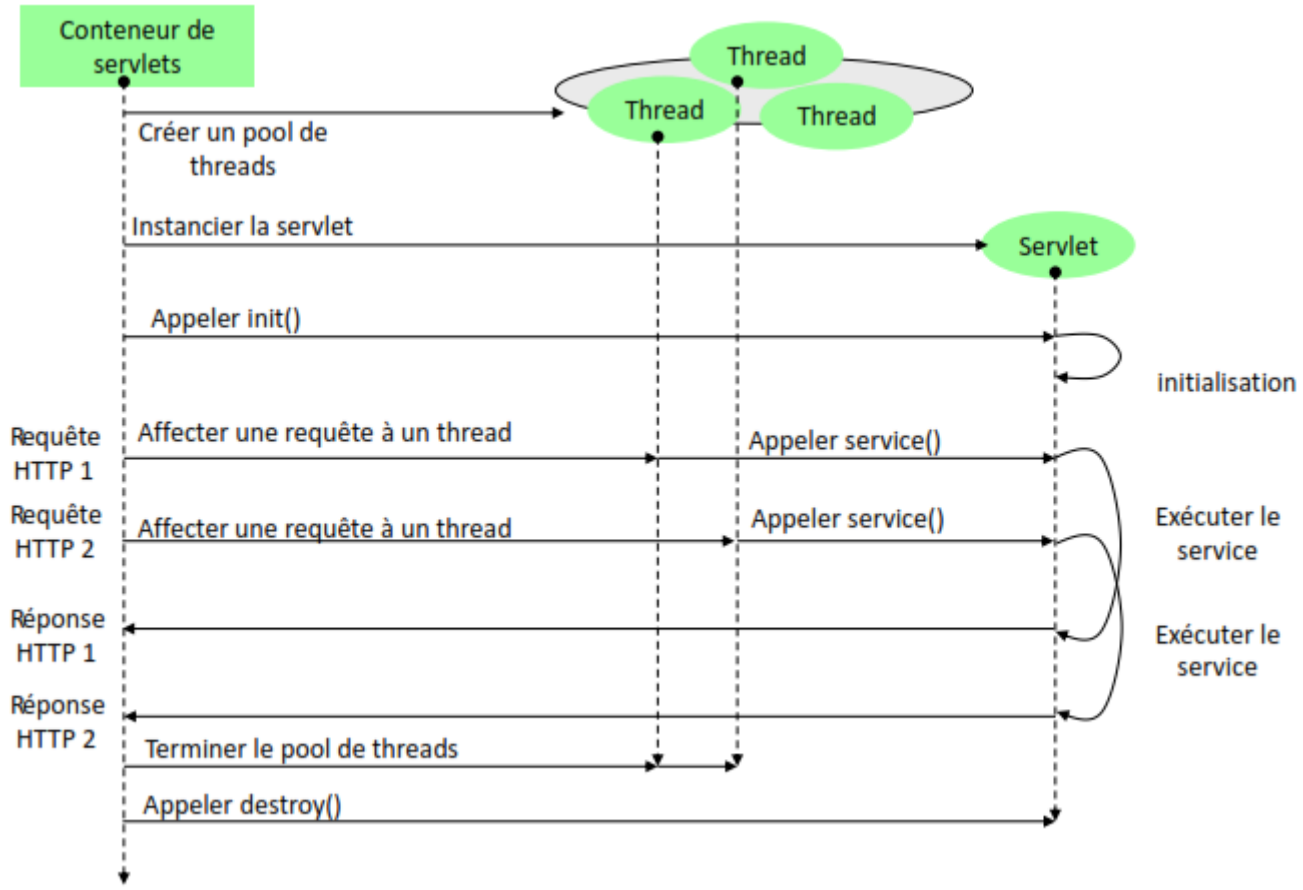



```java
package sadiq.test.servlets;
import java.io.IOException;

@WebServlet(name = "Test3", urlPatterns = {"/Test3"})
public class Test3 extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public Test3() {
        super();
        // TODO Auto-generated constructor stub
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
        String leNom = request.getParameter("nom");
        int nbRepet = Integer.parseInt(request.getParameter("nb"));
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet HelloWorldServlet</title>");
        out.println("</head>");
        out.println("<body>");
        for (int i = 0; i < nbRepet; i++) {
        out.println("<h1>Hello " + leNom + "</h1>");
        }
        out.println("<h2>Envoyé le " + new Date() + "</h1>");
        out.println("</body>");
        out.println("</html>");
        } finally {
        out.close();
        }
    }
```

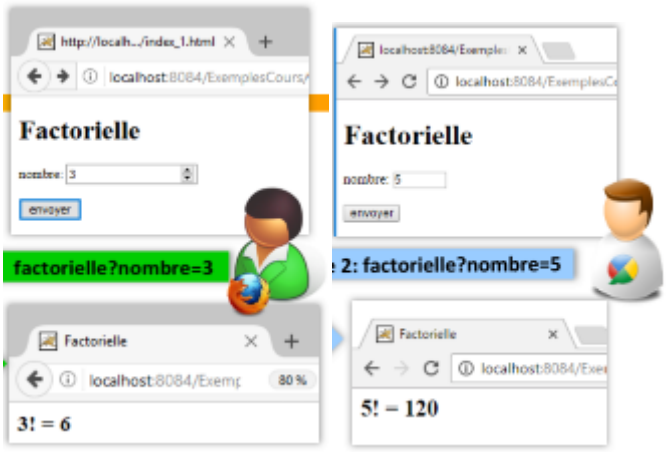
4. Execution concurrente des Servlets

Serveur web multi-threadé : Par défaut une servlet n'est instanciée qu'une seule fois. La même instance peut servir plusieurs requêtes simultanément et le conteneur crée un thread (processus léger) par requête pour pouvoir les traiter en parallèle.

Attention !! Les attributs de la servlet deviennent des ressources partagées, ils sont accédés de manière concurrente par chaque fil d'exécution




```
@WebServlet(name = "Factorielle", urlPatterns = {"/Factorielle"})
public class Factorielle extends HttpServlet {
protected int nombre;
protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
            response.setContentType("text/html;charset=UTF-8");
            PrintWriter out = response.getWriter();
            try {
                  out.println("<html>");
                  out.println("<head>");
                  out.println("<title>Factorielle</title>");
                  out.println("</head>");
                  out.println("<body>");
                  nombre = Integer.parseInt(request.getParameter("nombre"));
                  out.print("<h2>" + nombre + "!  =  ");
                  int fact = 1;
                  for (int i = 2; i <= nombre; i++) {
                  fact *= i;
                  }
                  out.println(fact + "</h2>");
                  out.println("</body>");
                  out.println("</html>");
            } finally {
                  out.close();
            }
      }

   }
```

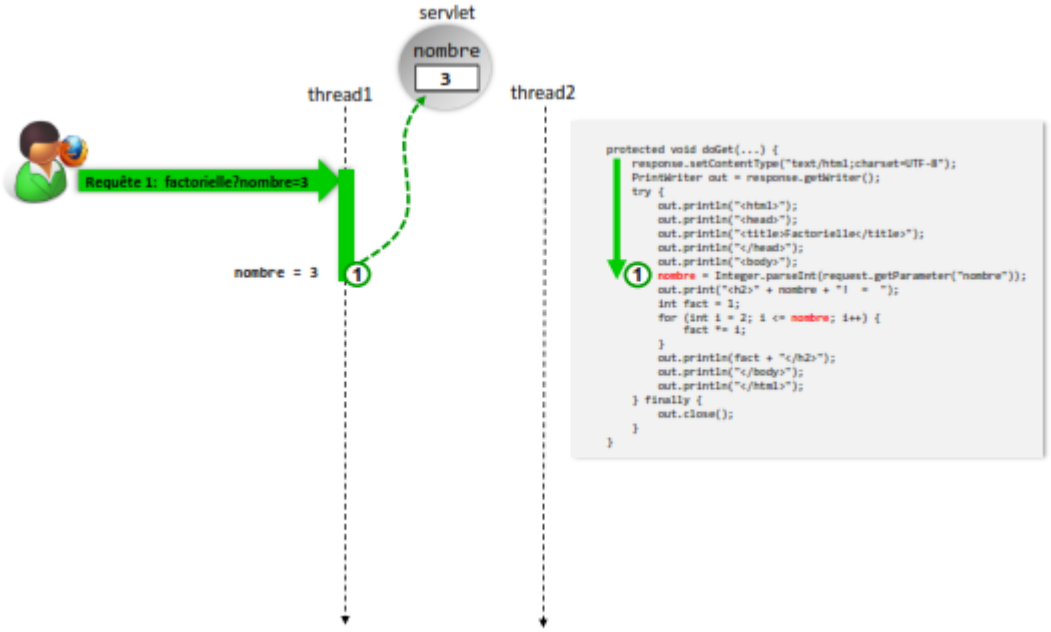


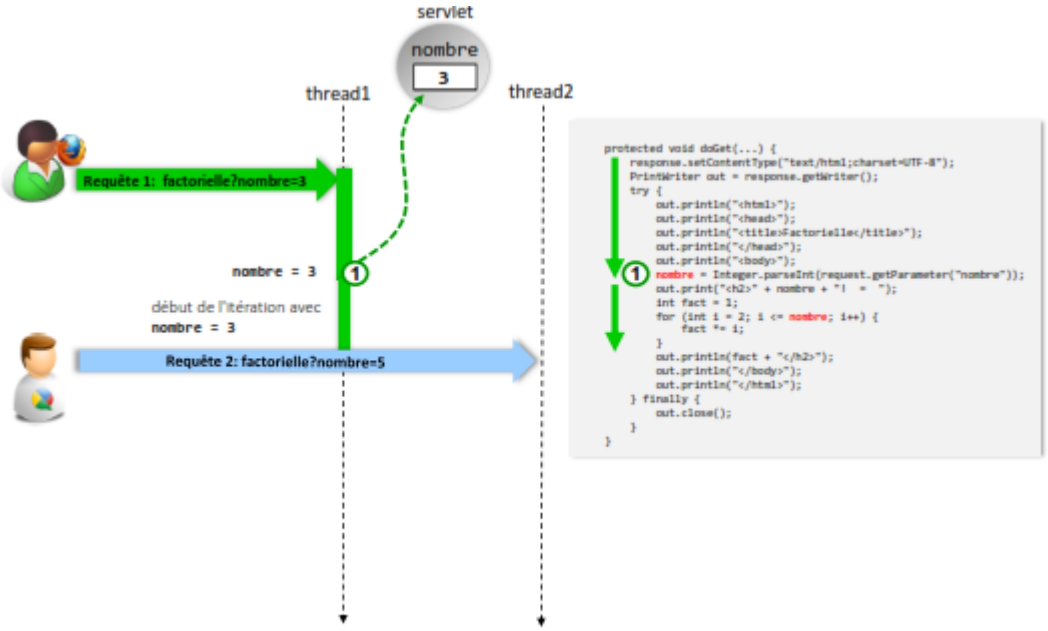

```
@WebServlet(name = "Factorielle", urlPatterns = {"/Factorielle"})
public class Factorielle extends HttpServlet {
protected int nombre;
protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
            response.setContentType("text/html;charset=UTF-8");
            PrintWriter out = response.getWriter();
            try {
                  out.println("<html>");
                  out.println("<head>");
                  out.println("<title>Factorielle</title>");
                  out.println("</head>");
                  out.println("<body>");
                  nombre = Integer.parseInt(request.getParameter("nombre"));
                  out.print("<h2>" + nombre + "!  =  ");
                  int fact = 1;
                  for (int i = 2; i <= nombre; i++) {
                  fact *= i;
                  }
                  out.println(fact + "</h2>");
                  out.println("</body>");
                  out.println("</html>");
            } finally {
                  out.close();
            }
      }

   }
```

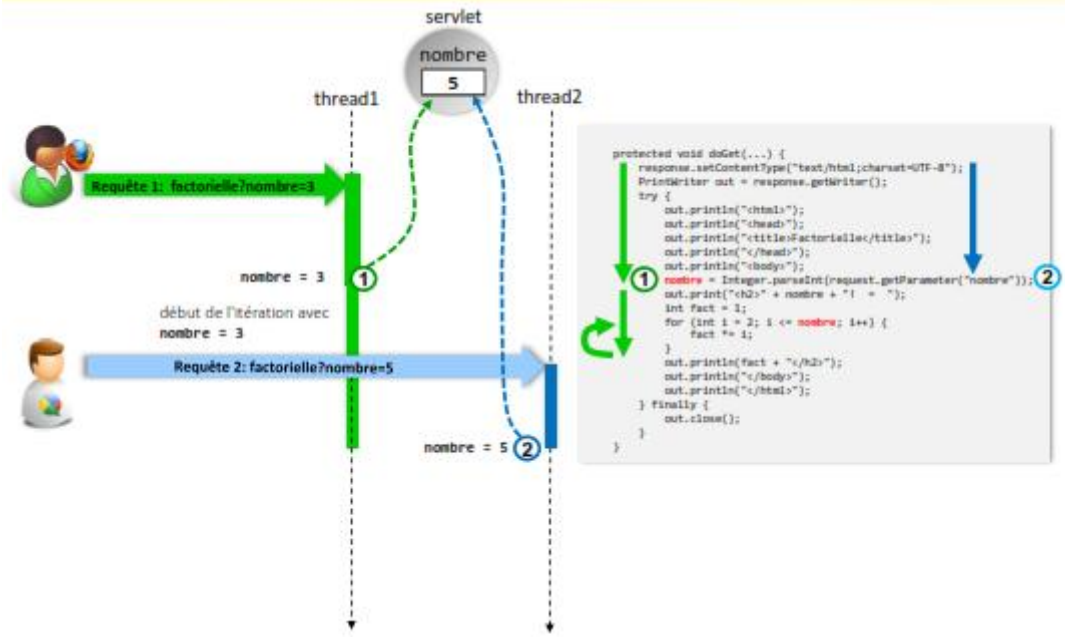
Source du problème

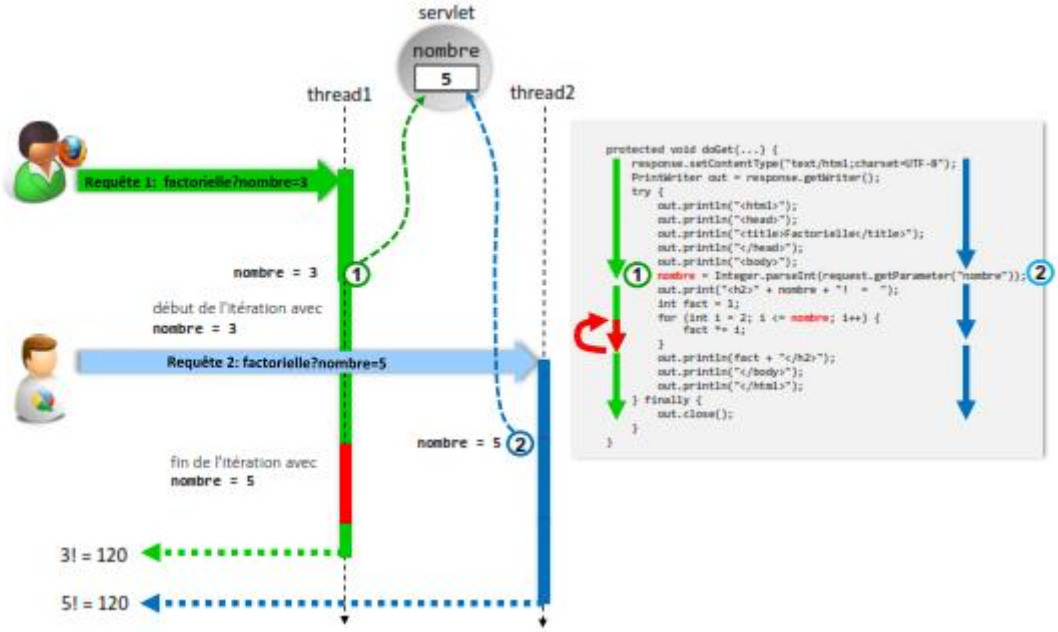
La servlet n'est instanciée qu'une seule fois,

- Le conteneur utilise des threads différents pour répondre à chaque requête
- `nombre` est accédée de manière concurrente


servlet
nombre
3
thread1
thread2
Requête 1: factorielle?nombre=3
nombre = 3
1
protected void doGet(...) {
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {
out.println("<html>");
out.println("<head>");
out.println("<title>Factorielle</title>");
out.println("</head>");
out.println("<body>");
nombre = Integer.parseInt(request.getParameter("nombre"));
out.print("<h2>" + nombre + "! = ");
int fact = 1;
for (int i = 2; i <= nombre; i++) {
fact *= i;
}
out.println(fact + "</h2>");
out.println("</body>");
out.println("</html>");
} finally {
out.close();
}
}


servlet
nombre
3
thread1
thread2
Requête 1: factorielle?nombre=3
nombre = 3
1
début de l'itération avec
nombre = 3
Requête 2: factorielle?nombre=5
protected void doGet(...) {
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {
out.println("<html>");
out.println("<head>");
out.println("<title>Factorielle</title>");
out.println("</head>");
out.println("<body>");
nombre = Integer.parseInt(request.getParameter("nombre"));
out.print("<h2>" + nombre + "! = ");
int fact = 1;
for (int i = 2; i <= nombre; i++) {
fact *= i;
}
out.println(fact + "</h2>");
out.println("</body>");
out.println("</html>");
} finally {
out.close();
}
}


servlet
nombre
5
thread1
thread2
Requête 1: factorielle?nombre=3
nombre = 3
1
début de l'itération avec
nombre = 3
Requête 2: factorielle?nombre=5
nombre = 5
2
protected void doGet(...) {
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {
out.println("<html>");
out.println("<head>");
out.println("<title>Factorielle</title>");
out.println("</head>");
out.println("<body>");
nombre = Integer.parseInt(request.getParameter("nombre"));
out.print("<h2>" + nombre + "! = ");
int fact = 1;
for (int i = 2; i <= nombre; i++) {
fact *= i;
}
out.println(fact + "</h2>");
out.println("</body>");
out.println("</html>");
} finally {
out.close();
}
}



Solutions

- Sections critiques :
  - Utilisation du mot clé synchronized (blocs synchronisés ne pouvant être exécutés que par un seul thread à la fois)
  - http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html.
- Modèle d'exécution : 1 instance par thread
  - Implémenter l'interface SingleThreadModel
  - déprécié depuis la version 2.4 des servlets

Utilisation d'autres ressources

Le traitement d'une requête HTTP par une Servlet peut nécessiter l'utilisation d'autres ressources comme par exemple :

- l'inclusion d'une page HTML statique
- la redirection vers une autre Servlet ou page JSP

Cela peut être assuré à l'aide d'un objet javax.servlet.RequestDispacher

```
<<interface>> RequestDispatcher
+ forward(servletRequest req, ServletResponse rep) : void
+ include(servletRequest req, ServletResponse rep) : void
```

Un objet ResquestDispacher est utilisable en passant via d'autres objets :

- via un objet javax.servlet.ServletContext
- via un objet javax.servlet.http.HttpServletRequest
- RequestDispatcher getRequestDispatcher(java.lang.String path)

Ici, path est une chaîne de caractères précisant le chemin de la ressource vers laquelle la requête doit être transférée. Sa valeur doit commencer par un '/' et est interprété comme relatif au contexte de l'application.

5. RequestDispatcher : include

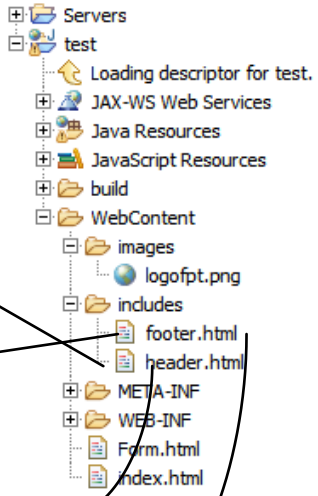
Exemple: insérer dans la réponse des fragments de code HTML

```
package fr.im2ag.m2cci.servletsdemo.servlets;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
...
// Exemple: insérer dans la réponse des fragments de code HTML
@WebServlet(name = "InclusionTest", urlPatterns = {"/TestInclude"})
public class InclusionTest extends HttpServlet {
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {
// La partie de l'en tête
RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/includes/header.html");
dispatcher.include(request, response);
// contenu généré par la servlet
for (int i = 0; i < 5 ; i++) {
out.println("<h3>Hello Include</h3>");
}
// inclusion du pied de page
getServletContext().getRequestDispatcher("/includes/footer.html").include(reques
t, response);
} finally {
out.close();
}
}
}
```

Dans l'exemple on utilise les chemins relatifs au contexte de l'application

```html
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Mon Header</title>
</head>
<body>
    <header>
        <a href="www.fpt.ac.ma"><img src="./images/logofpt.png"/></a>
        <h1>Header de la FPT : Test d'inclusion</h1>
    </header>
</body>
</html>
```

```html
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Mon Footer</title>
</head>
<body>
    <footer> Test d'inclusion d'un footer - Copyright : CEFT</footer>
</body>
</html>
```



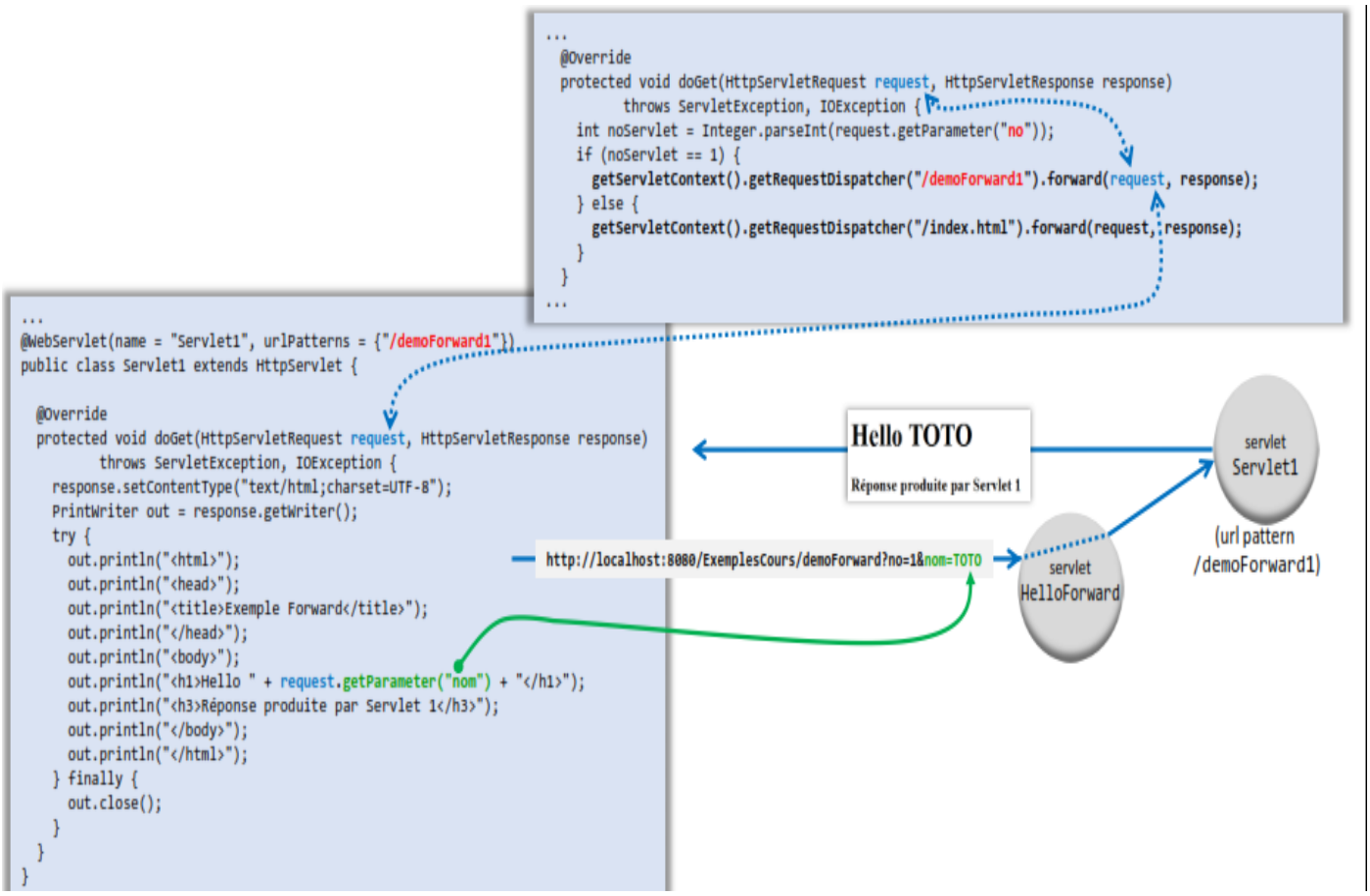

```java
// La partie de l'en tête
RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/includes/header.html");
dispatcher.include(request, response);
// contenu généré par la servlet
for (int i = 0; i < 5 ; i++) {
out.println("<h3>Hello Include</h3>");
}
// inclusion du pied de page
getServletContext().getRequestDispatcher("/includes/footer.html").include(reques
t, response);
```

6. RequestDispatcher : forward

Exemple : redirige vers une autre ressource pour produire la réponse

```java
...
import java.io.IOException;
...
@WebServlet(name = "HelloForward", urlPatterns = {"/demoForward"})
public class HelloForward extends HttpServlet {
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
       int noServlet = Integer.parseInt(request.getParameter("no"));
              if (noServlet == 1) {
                     getServletContext().getRequestDispatcher("/demoForward1").forward(request,
response);
              } else {

       getServletContext().getRequestDispatcher("/index.html").forward(request, response);
                           }
       }
}
```

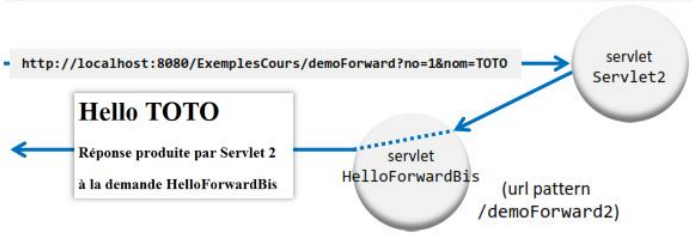
Les objets `request` et `response` de la servlet initiale sont transmis à la ressource vers laquelle la requête est redirigée



Les paramètres de `request` sont ceux de la requête originale.
La ressource appelante peut transmettre des informations supplémentaires via les attributs de la requête. Cette ressource peut transmettre des informations supplémentaires via les attributs de la requête.

```
@Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
    int noServlet = Integer.parseInt(request.getParameter("no"));
    if (noServlet == 1) {
      request.setAttribute("demandeur","HelloForwardBis");
      getServletContext().getRequestDispatcher("/demoForward2").forward(request, response);
    } else {
      getServletContext().getRequestDispatcher("/index.html").forward(request, response);
    }
  }
```

- la ressource appelante peut transmettre des informations supplémentaires via les **attributs** de la requête




```
...
@WebServlet(name = "Servlet2", urlPatterns = {"/demoForward2"})
public class Servlet1 extends HttpServlet {

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
      out.println("<html>");
      out.println("<head>");
      out.println("<title>Exemple Forward</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h1>Hello " + request.getParameter("nom") + "</h1>");
      out.println("<h3>Réponse produite par Servlet 2</h3>");
      String origine = (String) request.getAttribute("demandeur");
      out.println("<h3>à la demande " + origine + "</h3>");
      out.println("</body>");
      out.println("</html>");
    } finally {
      out.close();
    }
  }
}
```

Les paramètres sont des **String**
Les attributs peuvent être des **objets quelconques**

```
public void setAttribute(String name, Object value)

public Object getAttribute(String name)
```

Forward comporte plusieurs contraintes :

1. La servlet d'origine ne doit pas avoir déjà expédié des informations vers le client sinon une exception **`java.lang.IllegalStateException`** est lancée
2. Si des informations sont déjà présentes dans le buffer de la réponse mais n'ont pas encore été envoyées, elles sont effacées lorsque le contrôle est transmis à la deuxième ressource
3. Lorsque la deuxième ressource a terminé le traitement de la requête, l'objet réponse n'est plus utilisable par la première pour produire du contenu

```
int noServlet = Integer.parseInt(request.getParameter("no"));
PrintWriter out = response.getWriter();
try {
if (noServlet == 1) {
response.setContentType("text/html;charset=UTF-8");
      out.println("<h3>Cet en tête n'apparaît pas</h3>");
      out.println("</body>");                    2
1     out.println("</html>");
      out.flushBuffer();
getServletContext().getRequestDispatcher("/demoForward1").forward(request, response);
} else {
getServletContext().getRequestDispatcher("/demoForward3").forward(request, response);
out.println("<h3>Ce texte n'apparait pas</h3>");
out.println("</body>");                          3
out.println("</html>");
      }
} finally {
out.close();
    }
```

Cookies

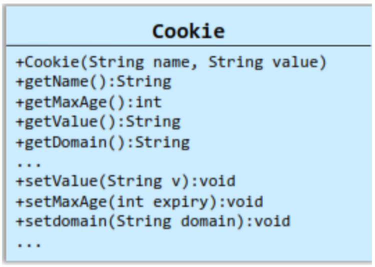
1. Introduction

Cookie (témoin de connexion) : petit élément d'information envoyé depuis un serveur vers un client HTTP et que ce dernier retourne lors de chaque interrogation du même serveur HTTP sous certaines conditions. Les cookies sont stockée localement sur le poste client (Source : wikipedia).

2. La classe Cookie

- classe `javax.servlet.http.Cookie`

Une fois le cookie créé, son nom ne peut plus être changé (pas de méthode setName())



L'utilisation des cookies est simple. Les figures ci-dessous illustres les méthodes utilisées pour récupérer ou déposer une cookie.

- récupérer un cookie
  - *via l'objet* `HttpServletRequest`
  - `Cookie[] getCookies()`

- déposer un cookie
  - *via l'objet* `HttpServletResponse`
  - `void addCookie(Cookie c)`

3. Utilisation et exemple

3.1. Mode d'utilisation

Le constructeur d'une cookie prends en paramètres son nom et la valeur a stocker.
Le reste des méthodes (les plus importantes) sont listées ci-dessous.

Constructeur :

```
public Cookie(String name,String value)
```

Envoi d'un cookie à un client par une servlet :

```
public void HttpServletResponse.addCookie(Cookie cookie)
```

Récupération des cookies. Il est impossible de récupérer un cookie connaissant son nom. On ne peut que récupérer un tableau de tous les cookies envoyés par le navigateur :

```
public Cookie[] HttpServletRequest.getCookies()
```

Modification de la valeur d'un cookie :

```
public void setValue(String newValue)
```

Exemple d'utilisation d'un cookie :

```
Cookie unCookie = new Cookie("nom","martin");
res.addCookie(unCookie);
```

- Récupération d'un cookie (`req` est l'objet requête)

```
Cookie[] cookies = req.getCookies();
if (cookies != null)
   for ( int i=0;i<cookies.length;i++ ){
       if (cookies[i].getName().equals("nom")){
             String valeur = cookies[i].getValue();
             break;
       }
   }
}
```

Un cookie est défini par son nom auquel est associée une valeur. Il est possible d'y ajouter certains attributs :

`void setDomain(String pattern)`
- précise le domaine pour lequel le cookie est valide
- par défaut, le cookie est retourné seulement au serveur qui l'a sauvé

`void setMaxAge(int expiry)`
- spécifie l'age maximum du cookie en secondes
- expiry<0 => le cookie expire avec le navigateur - expiry=0 => le cookie est détruit immédiatement

`void setPath(String uri)`
- définit le chemin dans le domaine pour lequel le cookie est valide
- par défaut, le cookie est valide pour la page qui le définit et toutes les pagesdu répertoire du dessous
- si uri="/" , le cookie est valide pour toutes les pages du serveur

`void setSecure(boolean flag)`
- f lag=t rue => le cookie doit être envoyé via un canal sécurisé (SSL)

`void setComment(String comment)`
- définit un commentaire qui décrit le rôle du cookie

3.2. Exemple d'utilisation d'un cookie

L'exemple présente une page qui permet de saisir un mot de passe. Lo mot de passe est enregistré dans un cookie

Ainsi chaque fois que la page est demandée (après rechargement), le dernier mot saisi est affiché.

De même si une requête à cette page est réalisée à travers une autre instance du navigateur

Après déconnexion, la valeur du cookie est réinitialisée

3.2.1. Fichier Question.java

```
package servlets;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.*;

@WebServlet(name = "Question", urlPatterns = {"/Question"})
  public class Question extends HttpServlet{public void doGet
        (HttpServletRequest request,HttpServletResponse response) throws IOException, ServletException{
    response.setContentType("text/html");
```

```
        PrintWriter out = response.getWriter();
        String raz=request.getParameter("raz");
        if(raz!=null) {
             Cookie[] cookies = request.getCookies();
             if (cookies != null) {
                  for (int i=0;i<cookies.length;i++){
                       if(cookies[i].getName().equals("cookieMot")) {
                          cookies[i].setValue(" ");
                                  response.addCookie(cookies[i]);}
            }String mot=null;
        out.println("<html><head><title>Question</title></head>"+ " <body bgcolor=yellow text=blue>"+"<center>"
        if(mot==null       || mot.equals(" "))     (
        out.println("<h2> Saisissez un mot mot= </h2>");
        }else{
        out.println("<h2>Dernier mot saisi"+mot +"</h2>");
        }
        out.println("<form action='reponse' method='post' >"+ "<hr><table><tr>"+
        "<td>Votre mot</td>"+
        "<td><input name='Mot' value="+mot+" \""); out.println("type='text' size='20'></tr></table><table>"+
    "<tr>"+
        "<td><input type='submit'value='Envoyer'></td>"+
        "<td><input type='reset' value='Rétablir'></td>"+ "</tr></table>"+
        "</form>"+
        "</center>"+"</body>"+"</html>");
        }
        }
```

3.2.2. Fichier Reponse.java

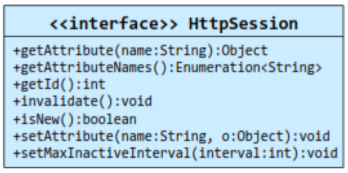
```
package servlets;
import javado.I0Exception;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.*;
@WebServlet(name = "Reponse", urlPatterns = {"/Reponse"})
public class Reponse extends HttpServlet { public void doPost
(HttpServletRequest request,HttpServletResponse response)
throws I0Exception, ServietException { response.setContentType("text/html");
PrintWriter out = response.getWriter();
String mot=request.getParameter("Mot");
Cookie cookieMot=new Cookie("cookieMot",mot);
cookieMot.setMaxAge(600); response.addCookie(cookieMot);
Cookie[] cookies = request.getCookies();
if (cookies != null)
          for (int i=0;i<cookies.length;i++){ if(cookies[i].getName().equals("cookieMot"))
                mot=cookies[i).getValue();
out.println( "<html>"+ "<head>"+ "<title>Bienvenue</title>"+ "</head>"+
"<body bgcolor=green text=white>"+ "<center>"+
"<h2> Vous avez saisi :"+ mot+"</h2>"+ "<FORM action='question' method='get'>"
+"<INPUT type='submit' value='déconnexion'>"+ "<INPUT type='hidden' name='raz' value='l'>"+
"</FORM>"+ "</center>"+ "</body>"+ "</html>");
```

Sessions

1. Introduction

De nombreuses applications nécessitent de relier entre elles une série de requêtes issues d'un même client
ex : application de E-commerce doit sauvegarder l'état du panier du client
La Session est une période de temps correspondant à navigation continue d'un utilisateur sur un site
HTTP étant sans état (Stateless) c'est de la responsabilité des applications web de conserver un tel état, i.e de gérer les sessions :

- Identifier l'instant où un nouvel utilisateur accède à une des pages du site
- Conserver des informations sur l'utilisateur jusqu'à ce qu'il quitte le site
- Cookies ≃ variables permettant de contrôler l'exécution de l'application Web. Mais …
    - stockage côté client
    - Possibilité de modifier les variables côté client
    - DANGEREUX
- Les données liées aux sessions doivent rester côté serveur
    - seul l'identifiant de session transmis sous forme de cookie
- en Java utilisation de l'interface : *javax.servlet.http.HttpSession*



2. Objet HttpSession

Pour la gestion des sessions il n'y a as de classe d'implémentation dans l'API JavaEE (HttpSession est une interface)
La classe d'implémentation dépend du conteneur web JEE (TomCat, Jetty, ….)

- *instanciée par le conteneur*
- *dans le code des servlets on n'utilise que l'interface*

L'objet HttpSession est associé à l'objet HttpServletRequest qui permet de le récupérer ou de le créer

- *logique : car identifiant de session transmis dans requête http (cookie ou paramètre de requête)*

2.1. Fonctionnalités de base

2.1.1. Obtenir une session

HttpSession session = request.getSession() ;

- *Récupération de la session associée à la requête*
- *Création d'une nouvelle session si elle n'existe pas*

HttpSession session = request.getSession(boolean create) ;

- *si une session est associée à la requête récupération de celle-ci*
- *sinon (il n'existe pas de session)*
  – si create == true, création d'une nouvelle session et récupération de celle-ci
  – si create == false null

boolean isNew()

- *true si le serveur a crée une nouvelle session (et que l'identifiant n'a pas encore été transmis au client)*

2.1.2. Fin de session

l'application met volontairement fin à la session : void invalidate()

- destruction de la session et de tous les éléments qu'elle contient

Le serveur le fait automatiquement après une certaine durée d'inutilisation

- configuré dans le fichier de déploiement de l'application (web.xml)
  – exemple

```
<session-config>
<session-timeout>10</session-timeout>
</session_config>
```

- *configuré par l'application*
  – int getMaxInactiveInterval()
  Récupère la durée de vie (en seconde) de la session si elle est inutilisée
  – void setMaxInactiveInterval(int interval)
  Fixe la durée de vie (en seconde) de la session si elle est inutilisée

2.1.3. Sauvegarde d'objets

Association clé ⇔ instance : void setAttribute(String key, Object value)

- *exemple :*

```
HttpSession session = request.getSession();
String name= "Ahmad Ali";
 session.setAttribute("nom", name);
session.setAttribute("couleur", new Color(222,114,14));
```

Extraction d'objets :

récupération d'un objet à partir de sa clé : Object getAttribute(String key)

Exemple :

```
HttpSession session = request.getSession();
String theName = (String) session.getAttribute("nom");
Color c = (Color) session.getAttribute("couleur");
```