Chapitre 2

La couche Contrôle : Les Servlets

1. Définition

- **Servlet**
  - Programme Java exécuté côté serveur Web
  - programme ”autonome” stocké dans un fichier **.class** sur le serveur web
  - Traitement de requêtes HTTP et génération de réponses dynamiques comparables aux CGI(Common Gateway Interface)
  - Permet d'étendre les fonctionnalités de base d'un serveur HTTP
  - exécutable avec tous les serveurs Web (Apache, IIS,…) ayant un moteur de servlet/JSP (e.g. **Tomcat**)

2. Rôles d'une Servlet

- Créer des pages HTML dynamiques, générer des images, …
- Effectuer des tâches de type CGI qui sont des traitements applicatifs coté serveur WEB
  - Manipulation d'une base de données
  - Gestion d'un système de surveillance, …

- Respecter les principes d'une architecture : écrire une application en Java dont l'interface utilisateur est dans le client
  - Applet (SWING)
  - Téléphone portable (WAP)
  - Navigateur (HTML)

3. Caractéristiques d'une Servlet

- **Portabilité**
  - Technologie indépendante de la plate-forme et du serveur
  - Un langage (Java) et plusieurs plate-forme (.NET plusieurs langages et une plate-forme)
- **Puissance**
  - Disponibilité de l'API de Java
  - Manipulation d'images, connectivité aux bases de données (JDBC), …
- **Efficacité**
  - Une Servlet est chargée une seule fois (CGI chargée puis déchargée après utilisation)
  - Une Servlet conserve son état (connexions à des bases de données)
- **Sûreté**
  - Typage fort de Java
  - Gestion des erreurs par exception

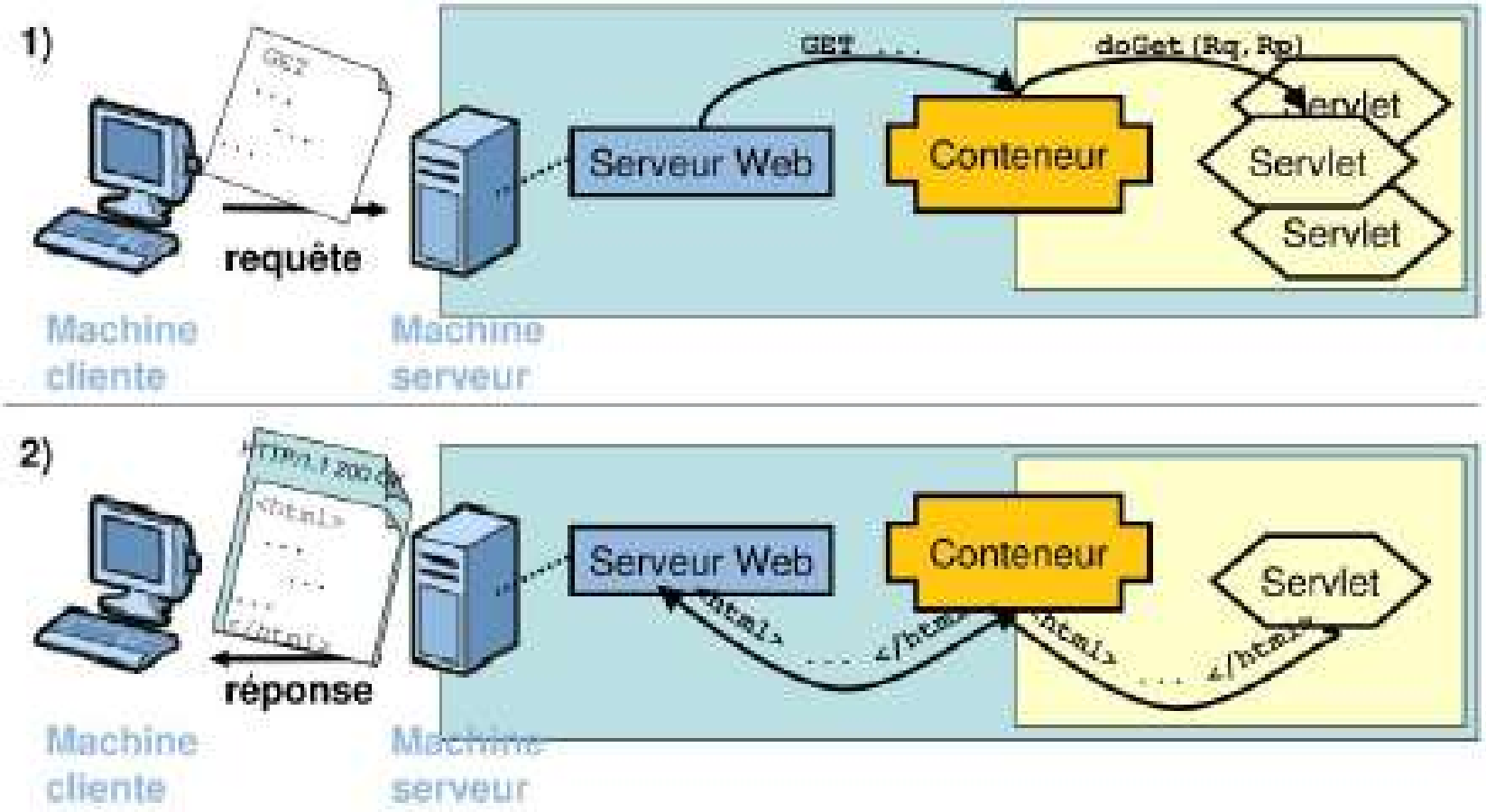
4. Conteneur de Servlet

- Un conteneur est un composant logiciel système qui contrôle d'autres composants, dits métier.
- Le conteneur de servlets est un serveur Web spécialisé et un environnement d'exécution propre aux servlets contenues, qui fournit l'API des classes et méthodes que les servlets euvent utiliser (**Package javax**) .
- **Tomcat** est un exemple de conteneur de servlets.
- Les servlets n'ont pas de méthode main(), elles sont **invoquées** et **contrôlées** par le conteneur Tomcat.
- Les requêtes ne sont pas adressées aux servlets mais au conteneur dans lequel elles sont déployées.
- Tous les conteneurs de servlets doivent supporter le protocole **HTTP** et peuvent aussi supporter le protocole **HTTPS**.

Rôles du conteneur de Servlets

Un conteneur fournit aux les Servlets

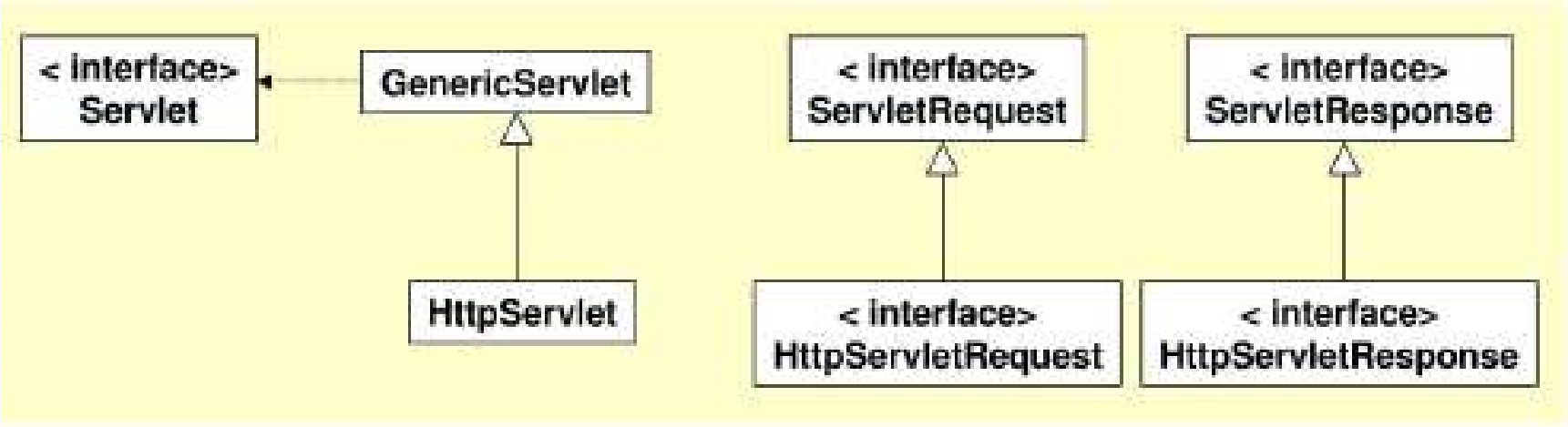
- Un support pour la communication : pas besoin de ServerSocket, Socket, Stream, …
- La gestion du cycle de vie des servlets
- Un support pour le Multithreading : création automatique des Threads pour traiter les requêtes parallèles.
- Un support pour la sécurité
- Un support pour les JSP

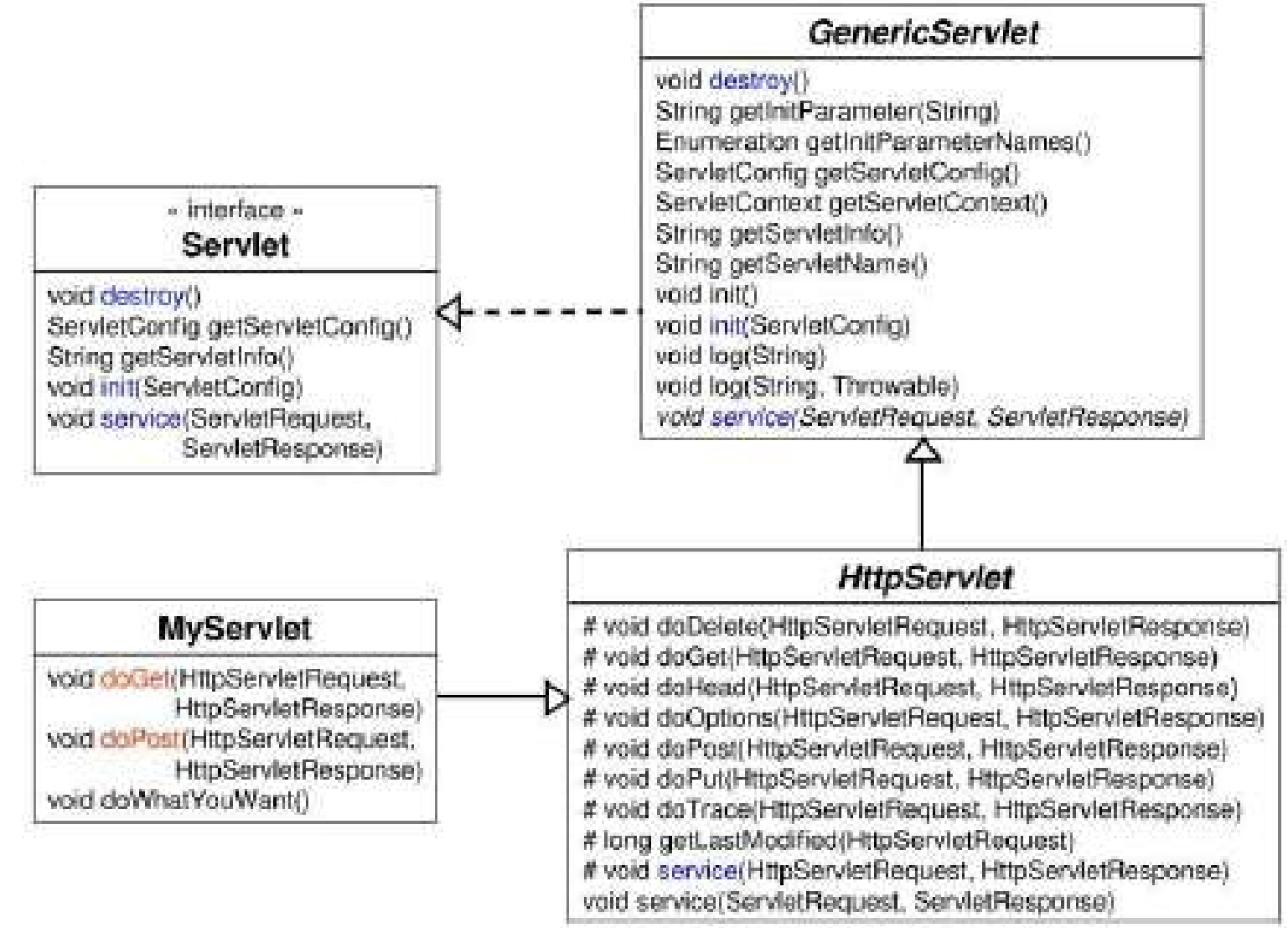

1)
GET ...
doGet (Rq, Rp)
Serveur Web
Conteneur
Servlet
Servlet
Servlet
requête
Machine cliente
Machine serveur
2)
Serveur Web
Conteneur
Servlet
réponse
Machine cliente
Machine serveur

5. L'API Servlet

- Les servlets sont conçues pour agir selon un modèle de **requête/réponse**.
- Tous les protocoles utilisant ce modèle peuvent être utilisés tels que http, ftp, etc .
- L'API **servlet** est une extension du jdk de base, elle est regroupée dans des packages préfixés par **javax**
- L'API servlet regroupe un ensemble de classes dans deux packages :
  - **javax.servlet** : contient les classes pour développer des serlvets génériques indépendantes d'un protocole
  - **javax.servlet.http** : contient les classes pour développer des servlets qui reposent sur le protocole http utilisé par les serveurs web.

API Servlet


< interface>
Servlet
GenericServlet
HttpServlet
< interface>
ServletRequest
< interface>
HttpServletRequest
< interface>
ServletResponse
< interface>
HttpServletResponse


« interface »
Servlet
void destroy()
ServletConfig getServletConfig()
String getServletInfo()
void init(ServletConfig)
void service(ServletRequest,
ServletResponse)
GenericServlet
void destroy()
String getInitParameter(String)
Enumeration getInitParameterNames()
ServletConfig getServletConfig()
ServletContext getServletContext()
String getServletInfo()
String getServletName()
void init()
void init(ServletConfig)
void log(String)
void log(String, Throwable)
void service(ServletRequest, ServletResponse)
MyServlet
void doGet(HttpServletRequest,
HttpServletResponse)
void doPost(HttpServletRequest,
HttpServletResponse)
void doWhatYouWant()
HttpServlet
void doDelete(HttpServletRequest, HttpServletResponse)
void doGet(HttpServletRequest, HttpServletResponse)
void doHead(HttpServletRequest, HttpServletResponse)
void doOptions(HttpServletRequest, HttpServletResponse)
void doPost(HttpServletRequest, HttpServletResponse)
void doPut(HttpServletRequest, HttpServletResponse)
void doTrace(HttpServletRequest, HttpServletResponse)
long getLastModified(HttpServletRequest)
void service(HttpServletRequest, HttpServletResponse)
void service(ServletRequest, ServletResponse)

6. Utilisation d'une Servlet

Une servlet est une classe Java qui implémente l'interface **javax.servlet.Servlet**

- soit directement,
- soit en dérivant d'une classe qui implémente déjà cette interface (comme **GenericServlet** ou **HttpServlet**)

- L'interface **javax.servlet.Servlet** possède **5** méthodes qui permettent au conteneur web de dialoguer avec la servlet
- Elle encapsule ainsi les méthodes nécessaires à la communication entre le conteneur et la servlet.

Interface Servlet

- Les méthodes offertes par l'interface **javax.servlet.Servlet** permettent de :
  - initialiser la servlet : **init()**
  - recevoir et répondre aux requêtes des clients : **service()**
  - détruire la servlet et ses ressources : **destroy()**

Interface Servlet

- Les méthodes **init()**, **service()** et **destroy()** assurent le cycle de vie de la servlet en étant respectivement appelées lors de la création de la servlet, lors de son appel pour le traitement d'une requête et lors de sa destruction.
- La méthode **init()** est appelée par le serveur juste après l'instanciation de la servlet.
- La méthode **service()** ne peut pas être invoquée tant que la méthode **init()** n'est pas terminée.
- La méthode **destroy()** est appelée juste avant que le serveur ne détruise la servlet : cela permet de libérer les ressources allouées dans la méthode **init()** tel qu'un fichier ou une connexion à une base de données.

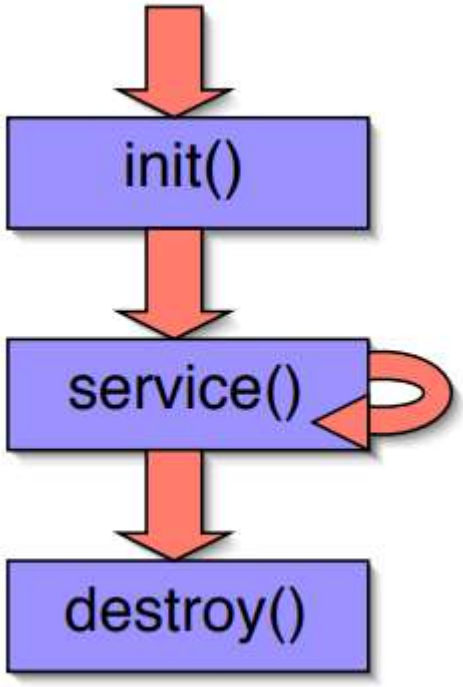
Interface Servlet

Méthode	Rôle
void service (ServletRequest req, ServletResponse res)	Cette méthode est exécutée par le conteneur lorsque la servlet est sollicitée : chaque requête du client déclenche une seule exécution de cette méthode. Cette méthode pouvant être exécutée par plusieurs threads, il faut prévoir un processus d'exclusion pour l'utilisation de certaines ressources.
void init(ServletConfig conf)	Initialisation de la servlet. Cette méthode est appelée une seule fois après l'instanciation de la servlet. Aucun traitement ne peut être effectué par la servlet tant que l'exécution de cette méthode n'est pas terminée.
ServletConfig getServletConfig()	Renvoie l'objet ServletConfig passé à la méthode init

Interface Servlet

Méthode	Rôle
void destroy()	Cette méthode est appelée lors de la destruction de la servlet. Elle permet de liberer proprement certaines ressources (fichiers, bases de données ...). C'est le serveur qui appelle cette méthode.
String getServletInfo()	Renvoie des informations sur la servlet.

Cycle de vie d'une servlet



- première requête de ce nom : chargement de la classe dans la JVM, instanciation, exécution de la méthode `init()`
- toute requête :
  - création des objets requêtes et réponses
  - création d'un thread. Exécution de la méthode `service()` ou d'une sous-méthode.
- fin de vie : appel à `destroy()` et destruction de l'objet.

Structure de base d'une servlet

- La servlet définie par un concepteur est une sous-classe de **HttpServlet** dans laquelle on surcharge les méthodes **doGet()** ou **doPost**, en fonction du mode d'envoi des données utilisé(par GET ou POST).
- Ces deux méthodes prennent deux arguments : **HttpServletRequest** et **HttpServletResponse**.
  - **HttpServletRequest** renferme des méthodes grâces auxquelles on détermine des informations sur les **données entrantes** (les données d'un formulaire, les en-têtes d'une requêtes http ou le nom de l'ordinateur du client).
  - **HttpServletResponse** permet de spécifier les **informations renvoyées**, comme le code d'état http, les en-têtes de la réponse (content-type, set-cookies,...).

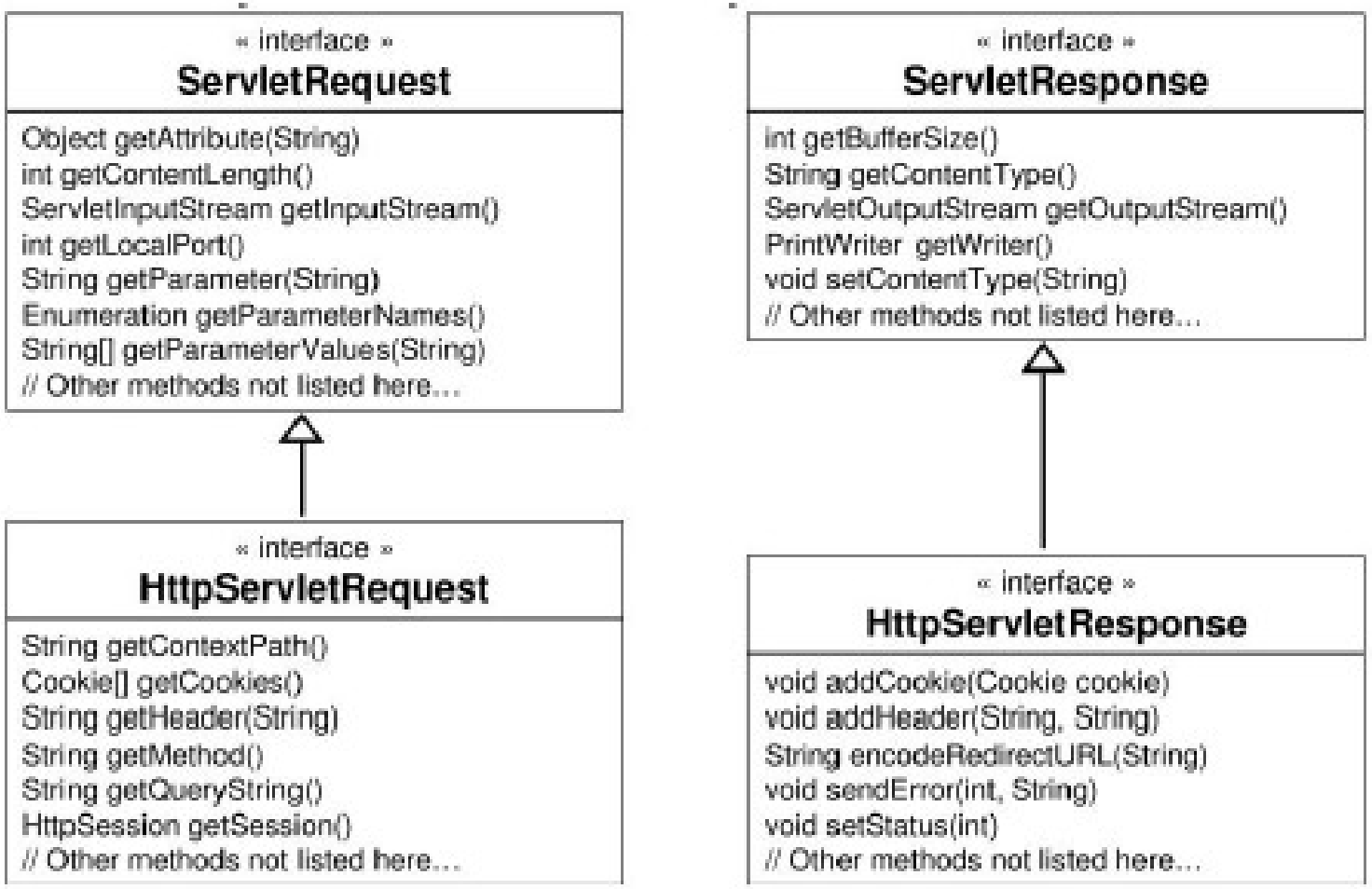
```
protected void service ( HttpServletRequest req ,
HttpServletResponse resp ) throws ServletException , java .io. IOException
```

- La méthode service a généralement le comportement suivant :

```
public class SimpleServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
    HttpServletResponse res)
    throws ServletException, IOException
    {
        // Lire la requête du client à partir de l'objet Request
        // La traiter
        // Répondre en construisant l'objet Response
    }
}
```

La requête et la réponse

- L'interface **ServletRequest** définit plusieurs méthodes qui permettent d'obtenir des données sur la requête du client.
- Le paramètre **HttpServletRequest** (hérite de **ServletRequest**) encapsule la requête HTTP et fournit des méthodes pour accéder à toutes les informations nécessaires sur l'environnement du serveur.


« interface »
ServletRequest
Object getAttribute(String)
int getContentLength()
ServletInputStream getInputStream()
int getLocalPort()
String getParameter(String)
Enumeration getParameterNames()
String[] getParameterValues(String)
// Other methods not listed here…
« interface »
HttpServletRequest
String getContextPath()
Cookie[] getCookies()
String getHeader(String)
String getMethod()
String getQueryString()
HttpSession getSession()
// Other methods not listed here…
« interface »
ServletResponse
int getBufferSize()
String getContentType()
ServletOutputStream getOutputStream()
PrintWriter getWriter()
void setContentType(String)
// Other methods not listed here…
« interface »
HttpServletResponse
void addCookie(Cookie cookie)
void addHeader(String, String)
String encodeRedirectURL(String)
void sendError(int, String)
void setStatus(int)
// Other methods not listed here…

Objet Request

Les méthodes de la classe **Request** sont :

Méthode	Rôle
ServletInputStream getInputStream()	Permet d'obtenir un flux pour les données de la requête
BufferedReader getReader()	Idem

Objet Response

L'interface **ServletResponse** définit plusieurs méthodes qui permettent de fournir la réponse faite par la servlet suite à ces traitements.

L'objet **HttpServletResponse** (hérite de **ServletResponse**) est utilisé pour construire un message de réponse HTTP renvoyé au navigateur client. Il contient les méthodes nécessaires pour définir : le type de contenu, en-tête et code de retour. Il contient aussi un flot de sortie pour envoyer des données (HTML ou autre) au navigateur.

Objet Response

Méthode	Rôle
SetContentType	Permet de préciser le type MIME de la réponse
ServletOutputStream getOutputStream()	Permet d'obtenir un flux pour envoyer la réponse
PrintWriter getWriter()	Permet d'obtenir un flux pour envoyer la réponse

Exemple1 : Hello

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("Hello World");
}
}
```

Exemple2 : Hello via une page web

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloGet extends HttpServlet {
public void doGet(HttpServletRequest requete, HttpServletResponse reponse)
    throws ServletException, IOException {
      String prenom = requete.getParameter("prenom");
      reponse.setContentType("text/html");
      PrintWriter out= reponse.getWriter();
      out.println("<html>");
      out.println("<body>");
      out.println("<h1>Bonjour " + prenom + " ! </h1>");
      out.println("</body>");
      out.println("</html>");
}
}
```

8. Fonctionnalités d'une servlet

a. Session

- Le protocole HTTP est un protocole non connecté (on parle aussi de *protocole sans états*, en anglais *stateless protocol*), cela signifie que chaque requête est traitée indépendamment des autres et qu'aucun historique des différentes requêtes n'est conservé.
- Le serveur ne conserve pas les données d'un client, une fois la réponse envoyée.
- Ainsi le serveur web ne peut pas se "souvenir" de la requête précédente, ce qui peut poser problème dans certaines utilisations telles que le e-commerce, pour lequel le serveur doit mémoriser les achats de l'utilisateur sur les différentes pages.
- Le traitement des requêtes clientes est réalisé requête par requête

⇒ le web n'offre par défaut aucun suivi de session.

Concept de session

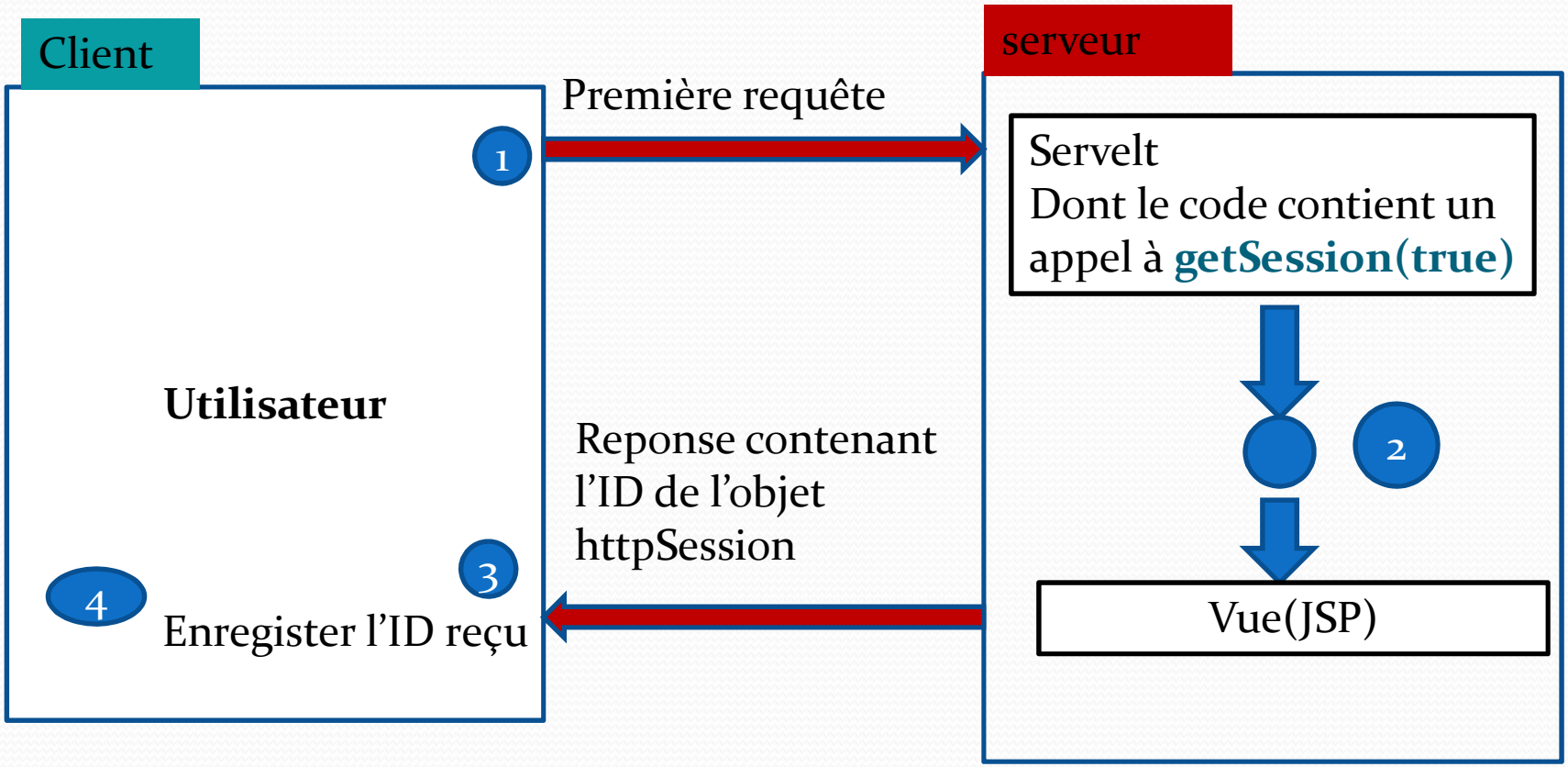
- Permettre au serveur de mémoriser les informations relatives au client.

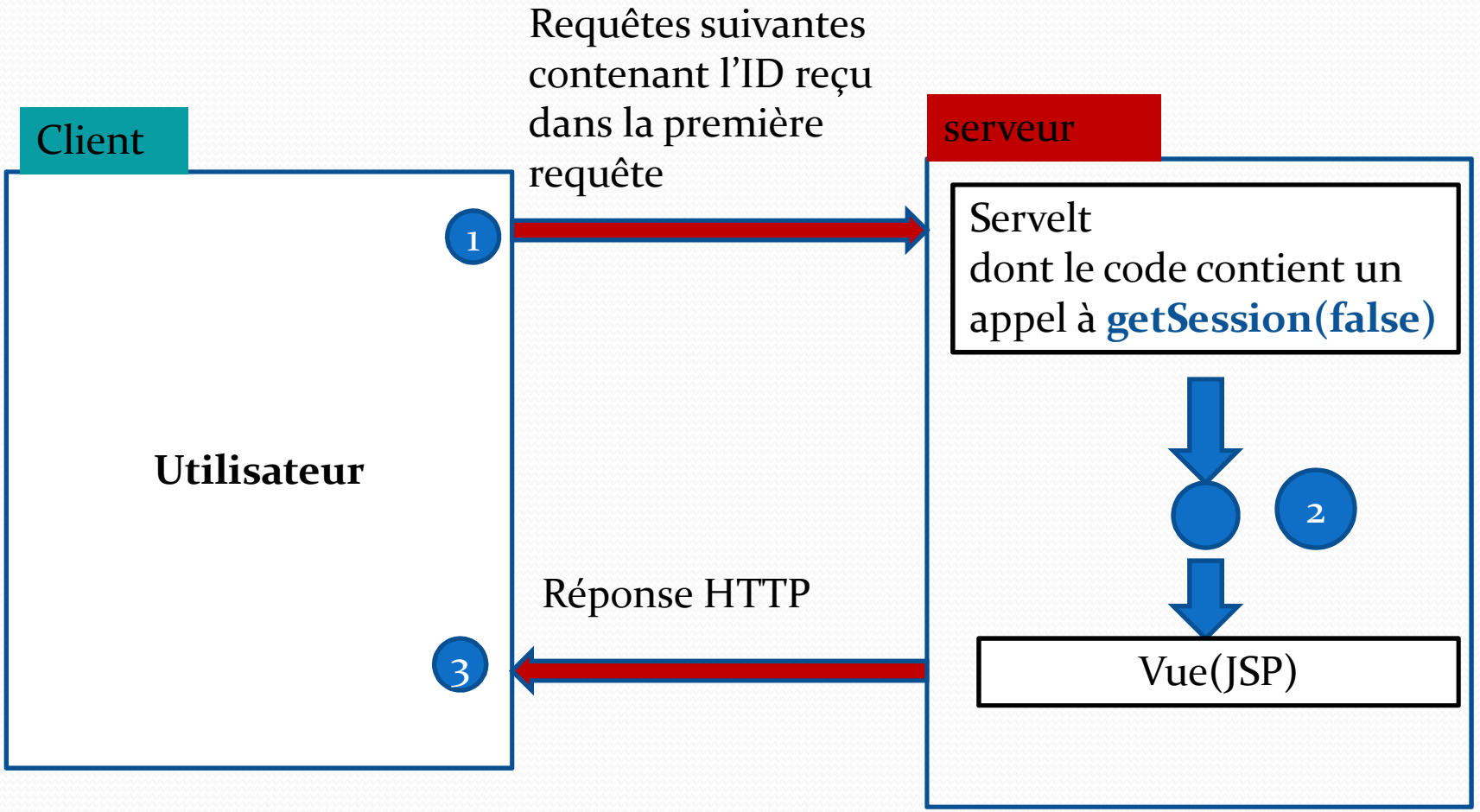
Session

- Représente un espace mémoire alloué pour chaque utilisateur, pour sauvegarder des informations .
- Le contenu d'une session est conservé jusqu'à ce que l'utilisateur **ferme son navigateur**, reste **inactif** trop longtemps, ou il **se déconnecte**.
- Cette fonctionnalité est supportée par les servlets à l'aide de l'objet **HttpSession** obtenu à l'aide de request.getSession()

  HttpSession session = request.**getSession()**;
- L'identifiant de l'objet session est échangé à chaque fois entre le navigateur du client et le serveur web(voir schéma suivant)

Création et échange d'un objet Session




Requêtes suivantes contenant l'ID reçu dans la première requête
Client
serveur
Servelt
dont le code contient un appel à getSession(false)
1
Utilisateur
2
Réponse HTTP
3
Vue(JSP)

Session

L'objet **session** propose les méthodes suivantes :

void **setAttribute**(String name, Object value)

ajoute un couple (name, value) pour cette session

Object **getAttribute**(String name)

retourne

- l'objet associé à la clé name s'il le trouve
- ou null s'il ne le trouve pas

Exemple : suivi d'1 session par un Compteur

```
public class HttpSessionServlet extends HttpServlet {
protected void doGet ( HttpServletRequest req , HttpServletResponse res)
throws ServletException , IOException {
    res.setContentType (" text / plain ");
    PrintWriter out = res.getWriter ();
    HttpSession session = req.getSession();
    Integer count = ( Integer ) session.getAttribute (" count ");
    if ( count == null )
        count = new Integer (1);
    else
        count = new Integer ( count.intValue() + 1);
     session.setAttribute (" count ", count );
     out. println (" Vous avez visité cette page " + count + " fois .");
    }}
```

b. Collaboration entre Servlets

- Les Servlets s'exécutant dans le **même serveur peuvent** dialoguer entre elles
- Deux principaux styles de collaboration
  - **Partage d'information** : un état ou une ressource.

    Exemple : un magasin en ligne pourrait partager les informations sur le stock des produits ou une connexion à une base de données
  - **Partage du contrôle** : une requête.

    Réception d'une requête par une Servlet et laisser à l'autre Servlet une partie ou toute la responsabilité du traitement

b. Collaboration entre Servlet: Partage d'information

- La collaboration entre les ressources d'une application web est réalisé à travers les objets implicites :
  - **L'objet Request :** pour le partage d'information dans une même requête;
  - **L'objet Session :** pour le partage d'information entre les requêtes d'un même client
  - **L'objet ServletContext :** L'utilisation de **ServletContext** permet aux applications web de disposer de son propre conteneur d'informations unique, et donc partager des informations entre différents clients de l'application.

Méthodes de servletContext

- Une Servlet retrouve le ServletContext de son application web par un appel à **getServletContext()**
- L'objet ServletContext propose les méthodes suivantes :
  - void **setAttribute**(String name, Object o) : lie un objet sous le nom indiqué
  - Object **getAttribute**(String name) : retrouve l'objet sous le nom indiqué
  - Enumeration **getAttributeNames**() : retourne l'ensemble des noms de tous les attributs liés
  - void **removeAttribute**(String name) : supprime l'objet lié sous le nom indiqué

Exemple : Partage de la spécialité de pizza à préparer entre tous les locaux de vente

```
public class PizzasAdmin extends HttpServlet {
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
      res.setContentType("text/plain");
      PrintWriter out = res.getWriter();
      ServletContext context = this.getServletContext();
      context.setAttribute("Specialite", "PiZZa Quatre Fromages");
      context.setAttribute("Date", new Date());
      out.println("La pizza du jour a été définie.");
}
}
```

Création de deux attributs

Exemple : Servlets qui vendent des pizzas et partagent une spécialité du jour

```
public class PizzasClient extends HttpServlet {
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
      res.setContentType("text/plain");
      PrintWriter out = res.getWriter();
      ServletContext context = this.getServletContext();
      String pizza_spec = (String)context.getAttribute("Specialite");
      Date day = (Date)  context.getAttribute("Date");
      DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM);
      String today = df.format(day);
      out.println("Aujourd'hui (" + today + "), notre specialite est : " +
         pizza_spec);
}}
```

Lecture des attributs

b. Collaboration entre servlets: Partage de contrôle

Les Servlets peuvent partager ou distribuer le contrôle de la requête

- Deux types de distribution
  - Distribution par délégation : une Servlet peut renvoyer une requête entière
  - Distribution par inclusion : une Servlet peut inclure du contenu généré
- Les avantages sont
  - La délégation de compétences
  - Une meilleure abstraction et une plus grande souplesse
  - Architecture logicielle MVC (Servlet = contrôle et JSP = présentation)

b. Collaboration entre servlets: Partage de contrôle

- Le support de la délégation de requête est obtenu par l'interface **RequestDispatcher**
- Une Servlet obtient une instance sur la **requête**
- La méthode

getRequestDispatcher(String path)

retourne une instance de type **RequestDispatcher** par rapport à un composant

- Un composant peut-être de tout type : Servlet, JSP, fichier statique, …
- path est un chemin relatif ou absolu ne pouvant pas sortir du contexte

b. Collaboration entre servlets : distribution par délégation

- La méthode forward(…) de l'interface RequestDispatcher renvoie une requête d'une Servlet à une autre ressource sur le serveur

  void **forward**(ServletRequest req, ServletResponse res) : redirection de requête

```
RequestDispatcher dispatch =
req.getRequestDispatcher("/index.html");
dispatch.forward(req,res);
```

Transmission des paramètres

- Lors du partage ou passage de contrôle :
  - Il y'a transmission des objets requête (**ServletRequest**) **&** réponse **(ServletResponse)**
  - Possibilité de transmettre des informations lors du renvoi :
    - en attachant une chaîne d'interrogation (au travers de l'URL)
    - en utilisant les attributs de requête via la méthode setAttribute(…)

Partage du contrôle : distribution par délégation

Exemple : distribuer un renvoi de Emetteur à Récepteur

```
public class EmetteurServlet extends HttpServlet{
   protected void doGet(HttpServletRequest req, Httpservlet
   Response res) throws ServletException, IOException{
   req.setAttribute("mot1", "au revoir ");
   RequestDispatcher   dispatch =
   req.getRequestDispatcher("/recepteur?mot2="bonjour ");
   Dispatch.forward(req, res);
   // ne fait rien
}}
```

Transmission d'informations par attributs

Le chemin est absolu par rapport au contexte de l'application web

Transmission d'informations par chaine d'interrogation

Exemple : distribuer un renvoi de Emetteur à Récepteur

```
public class RecepteurServlet extends HttpServlet{
   protected void doGet(HttpServletRequest req,
   Httpservlet Response res) throws
   ServletException, IOException{
         res.setContentType(« text/plain »);
         PrintWriter   out = res.getWriter();
         out.println(req.getParameter(”mot2”));
         out.println(req.getAttribute(”mot1”)
}}
```

Affichage des informations stockées dans la requête

Partage du contrôle : sendRedirect

- La méthode sendRedirect() est définie dans l'interface HttpServletResponse
- Elle permet de rediriger la réponse a une autre ressource(servlet, jsp ou fichier html).
- Elle accepte une URL absolue ou relative
- Différence entre sendRedirect et forward

forward	sendRedirect
Exécuté coté serveur	Coté client
Utilise les objets Request et Response	Elle envoie une nouvelle requete

Partage du contrôle : distribution par inclusion

- La méthode include(…) de l'interface **RequestDispatcher** inclut le contenu d'une ressource dans la réponse courante

```
RequestDispatcher dispatch =
req.getRequestDispatcher("/index.html");
Dispatch.include(req,res);
```

- La différence avec une distribution par renvoi est :
  - la Servlet appelante garde le contrôle de la réponse
  - elle peut inclure du contenu avant et après le contenu inclus
  - Possibilité de transmettre des informations lors de l'inclusion
    - en attachant une chaîne d'interrogation (au travers de l'URL)
    - en utilisant les attributs de requête via la méthode setAttribute(…)