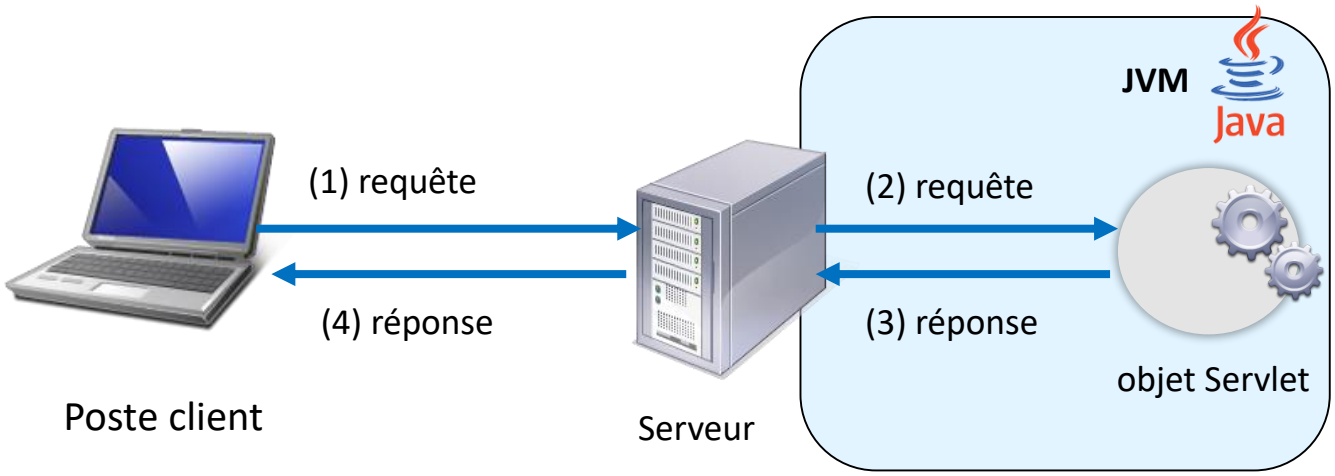
Applications Web Java Servlets

Philippe Genoud

Servlet ?

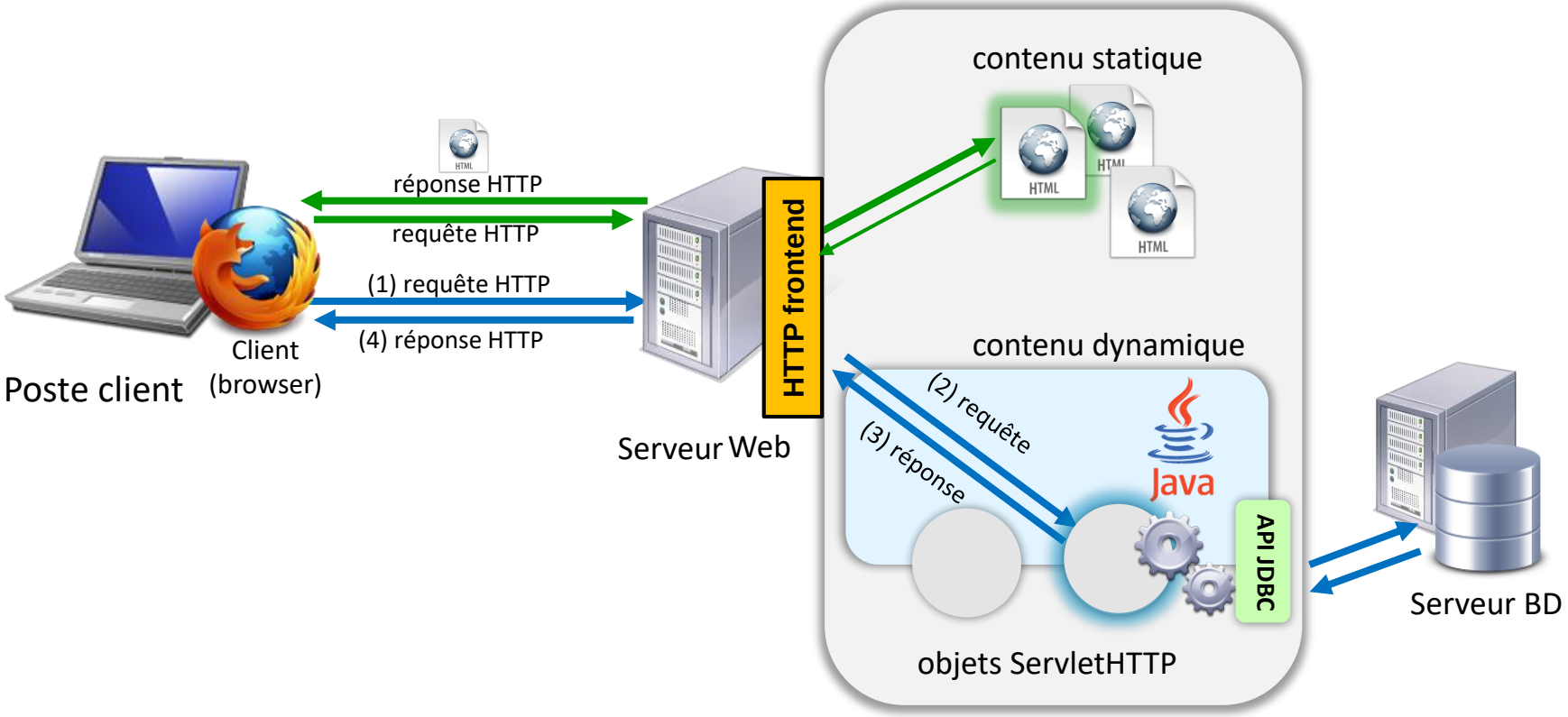
- Servlet (http://docs.oracle.com/javaee/7/tutorial/doc/servlets.htm)
  - *classe Java pour étendre les possibilités de serveurs hébergeant des applications accédées par un modèle de programmation de type requête-réponse*



  - *servlets peuvent potentiellement répondre à tout type de requête mais dans la réalité sont le plus souvent utilisées pour étendre des applications hébergées par un serveur web → requêtes HTTP*

Servlet ?

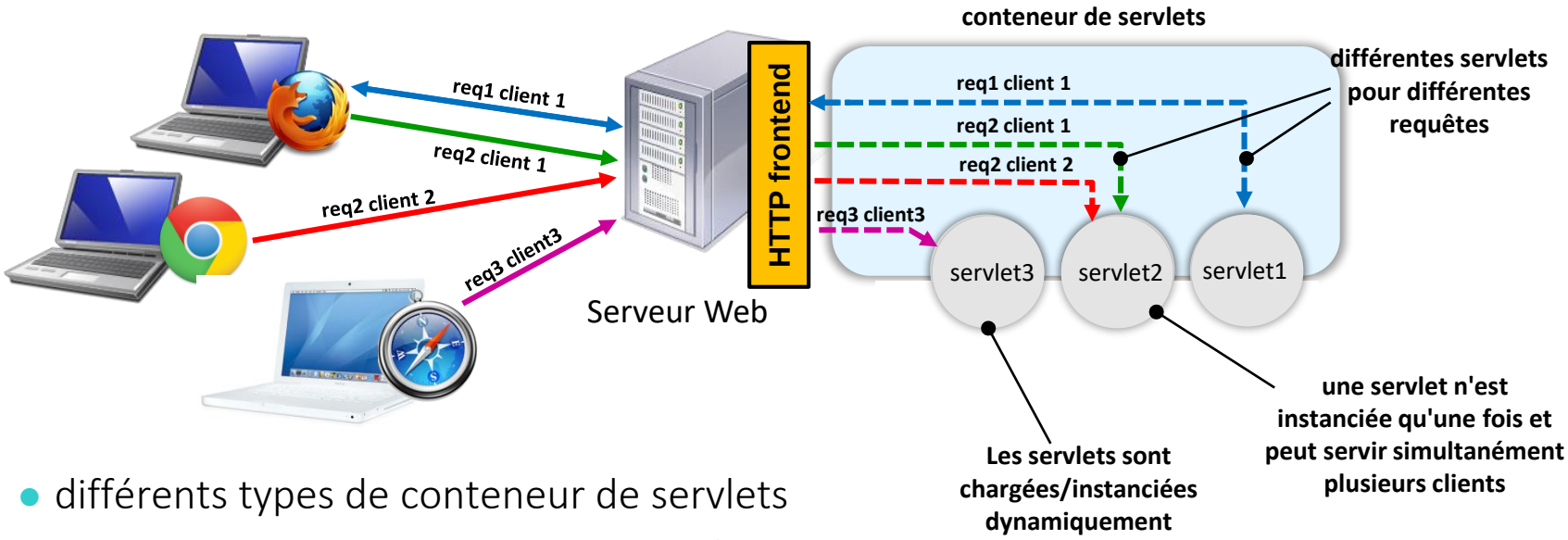
- technologie java propose des classes de servlets spécifiques pour le protocole HTTP



- 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 ...)*
- les servlets s'exécutent dans des serveurs dédiés (**servlets containers**)

Conteneurs de servlets

- 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 concurente dans des fils d'exécution (threads) différents*



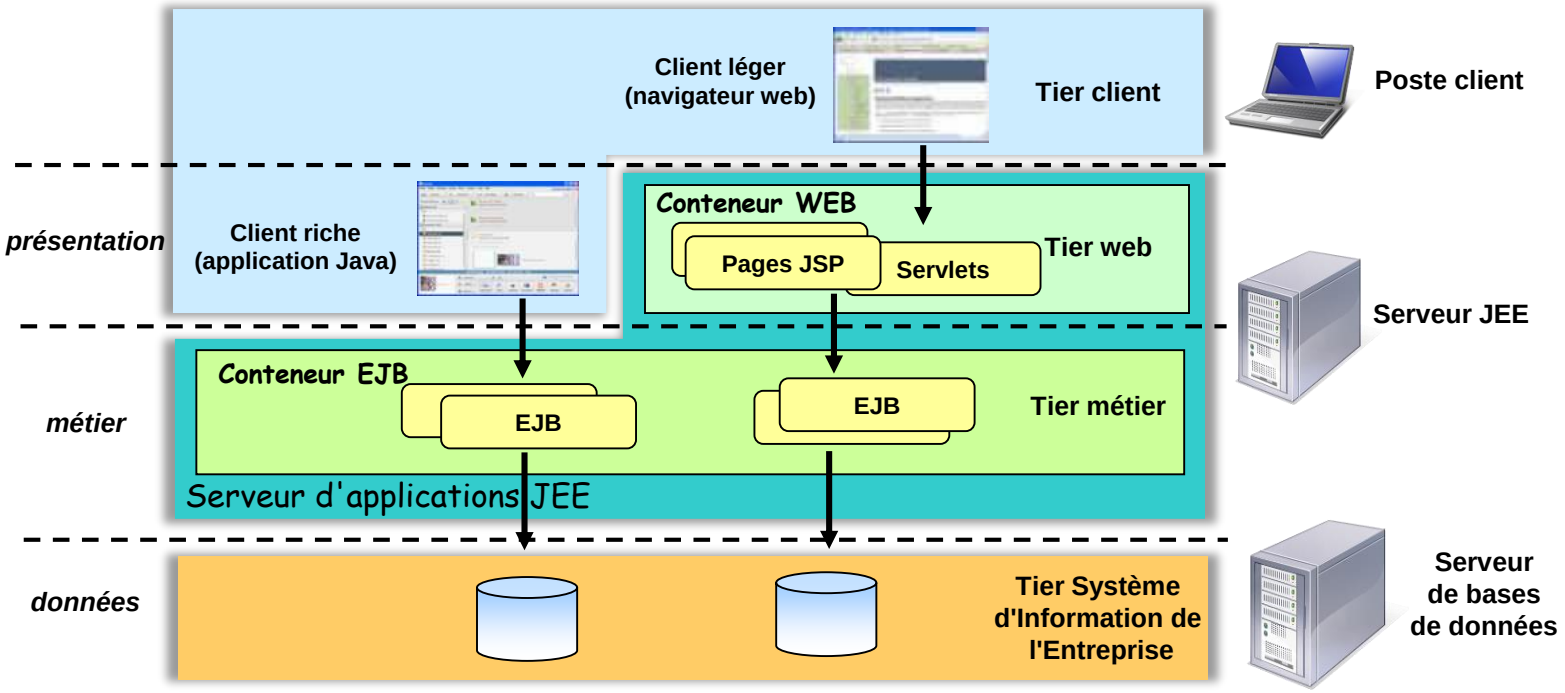
- différents types de conteneur de servlets
  - *conteneurs légers : Jetty, Resin, **Apache Tomcat***
  - *conteneurs JEE complets : Glassfish, JBoss, WebSphere….*
- Java assure la portabilité d'un conteneur à un autre

La plateforme JEE

- De manière plus générale les Servlets s'inscrivent dans la **plateforme JEE** (Java Entreprise Edition)
  - *spécifications (API) pour le développement d'applications réparties basées sur un ensemble de technologies Java* http://docs.oracle.com/javaee/7/api/
    - ***Servlets**, JSP(Java Server Pages), JSF (Java Server Faces) : pages web dynamiques*
    - *EJB (Entreprise Java Beans) : objets métiers*
    - *JDBC : API d'accès à des SGBD*
    - *JNDI (Java Naming an Directory Interface) : API pour nommage des objets*
    - *JTA (Java Transaction API) ; API pour gestion des transactions*
    - *JAAS (Java Authentication and Authorization Service)*
    - *Et de nombreuses autres…*
- S'appuie sur un modèle d'architecture multi-tiers (multi-couches)

Architecture des applications JEE

- Architecture multi-tiers



- Logique de l'application :
  - Composants web (Servlet, JSP,JFS)
  - Composants métiers (EJB Entreprise Java Beans)
- Services standards (cycle de vie des composants, multithreading, transactions, persistance…) pris en charge par les conteneurs Web et EJB du serveur d'application JEE

Situation actuelle de JEE

- Une spécification (Sun puis Oracle) :
  - *JEE 5 (2006), JEE 6 (2009), JEE 7 (2013)*
- Différentes implémentation de la plateforme
  - *Des implémentations commerciales*
    - *WebLogics server (Oracle),WebSphere (IBM), …*
  - *Ou open-source*
    - *GlassFish, JBoss, Geronimo (Apache), JOnAS…*
- Processus de certification
  - *TCK (Test Compatibility Kit) (~ 20000 tests)*
  - *http://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html*

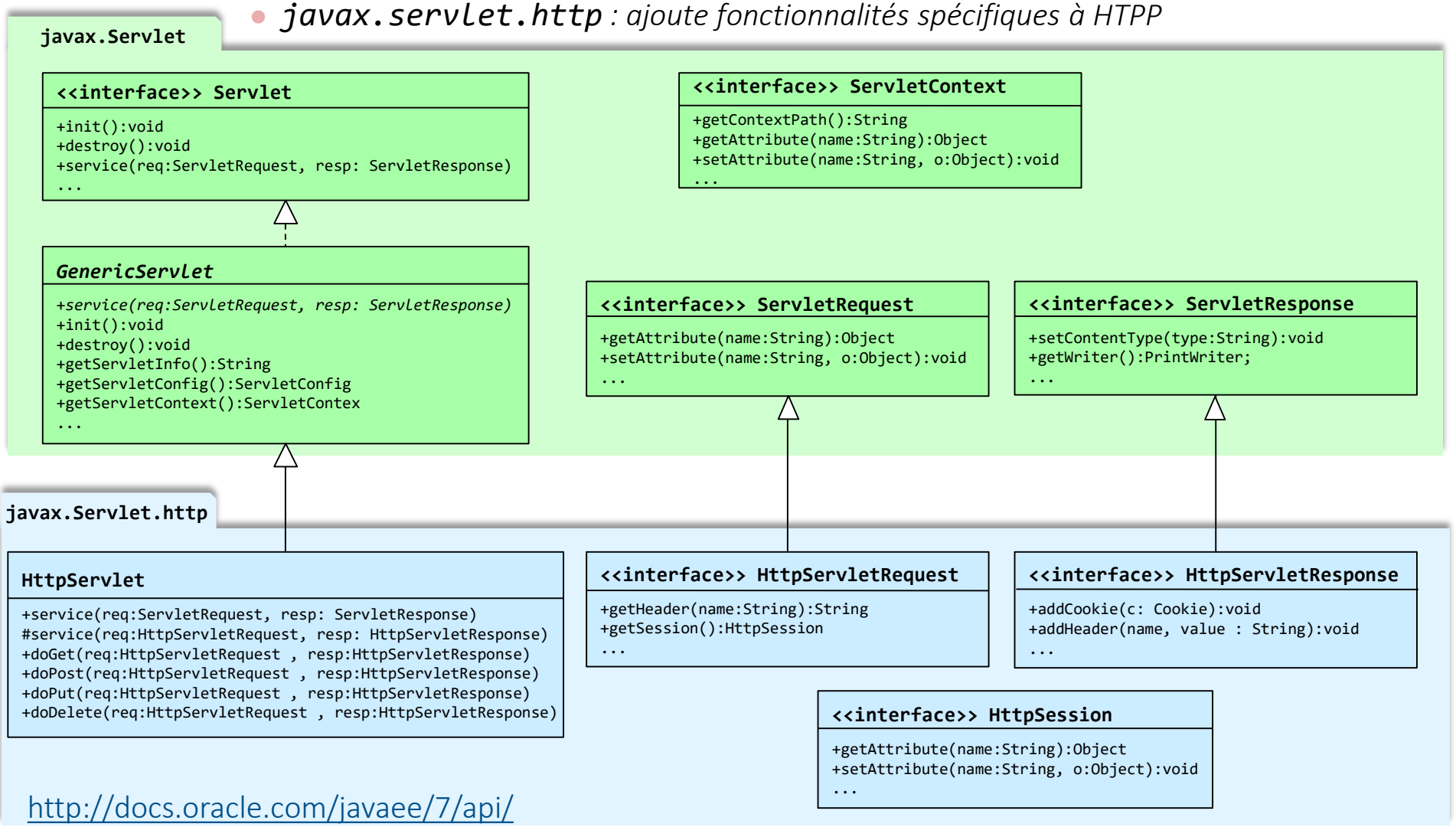
les versions de l'API Servlet

Version	Date de sortie	Plateforme
Servlet 4.0	Courant 2017	JavaEE 8 (fin 2017)
Servlet 3.1	Mai 2013	JavaEE 7
Servlet 3.0	Décembre 2009	JavaEE 6, JavaSE 6
Servlet 2.5	Septembre 2005	JavaEE 5, JavaSE 5
Servlet 2.4	Novembre 2003	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	Novembre 1998	--
Servlet 2.0	--	--
Servlet 1.0	Juin 1997	--

documentation API JavaEE7 : http://docs.oracle.com/javaee/7/api/

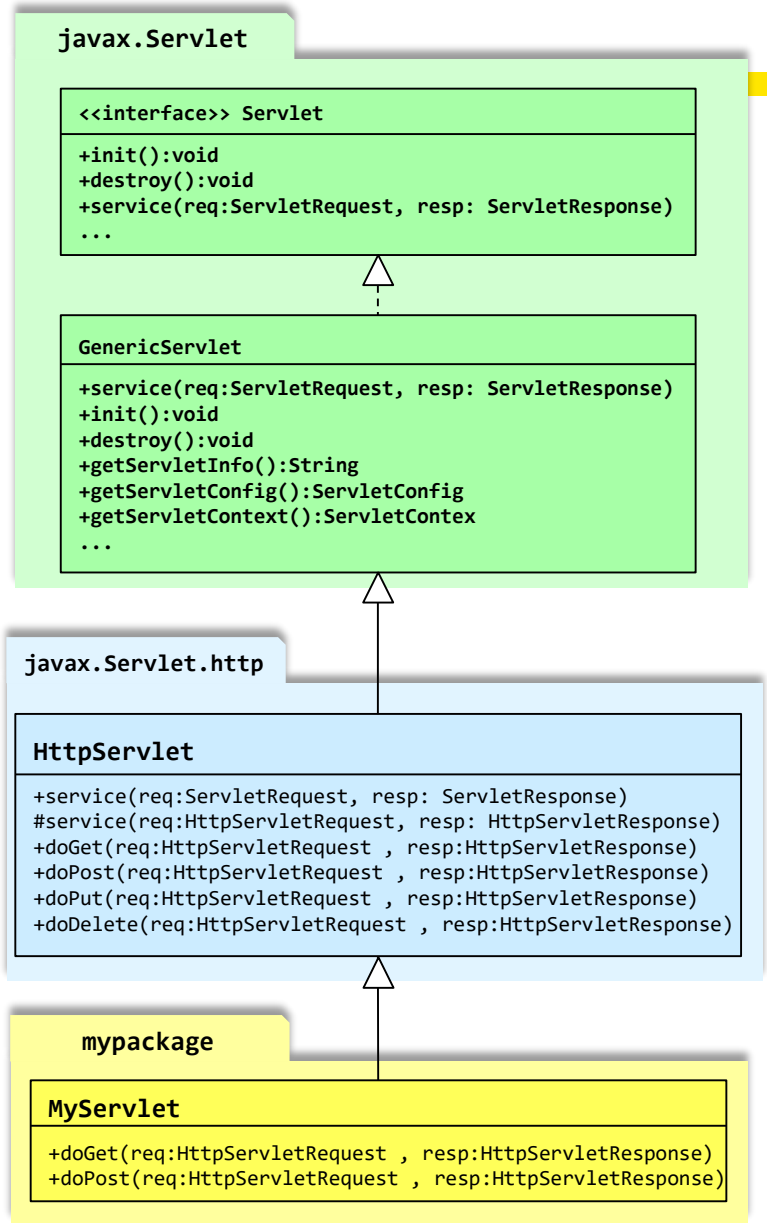
L'API servlets de JEE

- Deux packages :
  - `javax.servlet` : *support de servlets génériques indépendants du protocole*
  - `javax.servlet.http` : *ajoute fonctionnalités spécifiques à HTPP*



http://docs.oracle.com/javaee/7/api/

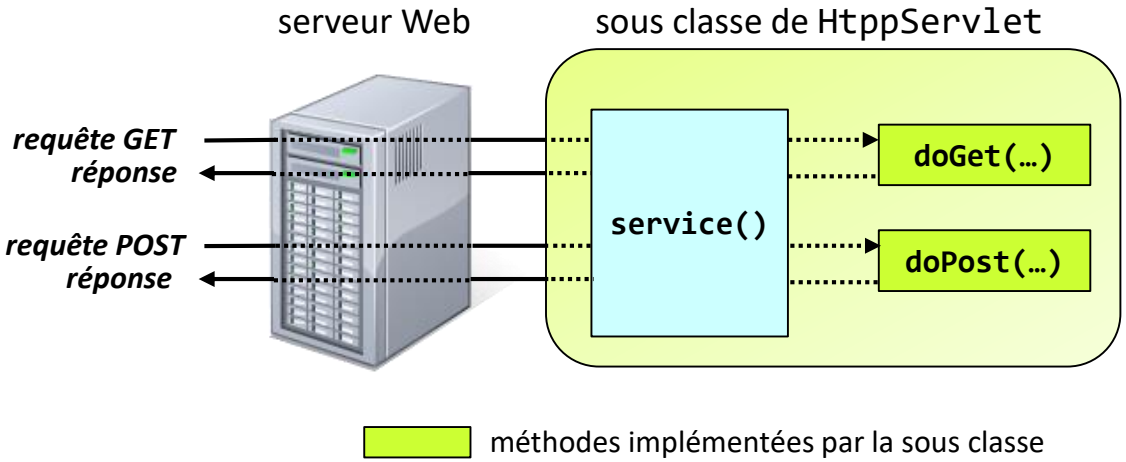
L'API servlets de JEE



- définir une servlet consiste à :
  - *sous classer* ***HttpServlet***
  - *redéfinir une ou plusieurs des méthodes* ***doXXX***

```
package mypackage;
import java.io.IOException;
import ...
...

public class MyServlet extends HttpServlet {
  /**
   * Handles the HTTP <code>GET</code> method.
   * @param request servlet request
   * @param response servlet response
   * @throws ServletException if a servlet-specific error occurs
   * @throws IOException if an I/O error occurs
   */
  @Override
  protected void doGet(HttpServletRequest request,
            HttpServletResponse response)
          throws ServletException, IOException {
      ...
  }

  @Override
  protected void doPost(HttpServletRequest request,
            HttpServletResponse response)
          throws ServletException, IOException {
      ...
  }
}
```

Cycle de vie d'une servlet



- 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()`

Première Servlet

```
package fr.im2ag.m2cci.servletsdemo.servlets;
import java.io.IOException;
import java.io.PrintWriter;
import java.utilDate;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

imports des différentes classes et interfaces nécessaires à l'écriture du code de la servlet.

avant JEE6 ces infos étaient dans un fichier de configuration: **web.xml**

```
@WebServlet(name = "HelloWorldServlet", urlPatterns = {"/HelloWorld"})
```

annotation définissant l'url associée à cette servlet (servlet Mapping)

```
public class HelloWorldServlet extends HttpServlet {
```

la servlet hérite de la classe `HttpServlet`

```
  /**
   * Handles the HTTP <code>GET</code> method.
   * @param request servlet request
   * @param response servlet response
   * @throws ServletException if a servlet-specific error occurs
   * @throws IOException if an I/O error occurs
   */
  @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>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();
    }
  }
}
```

redéfinition de la méthode doGet traitant les requêtes HTTP GET.

construction de la réponse à la requête en utilisant l'objet `HttpServletResponse`.

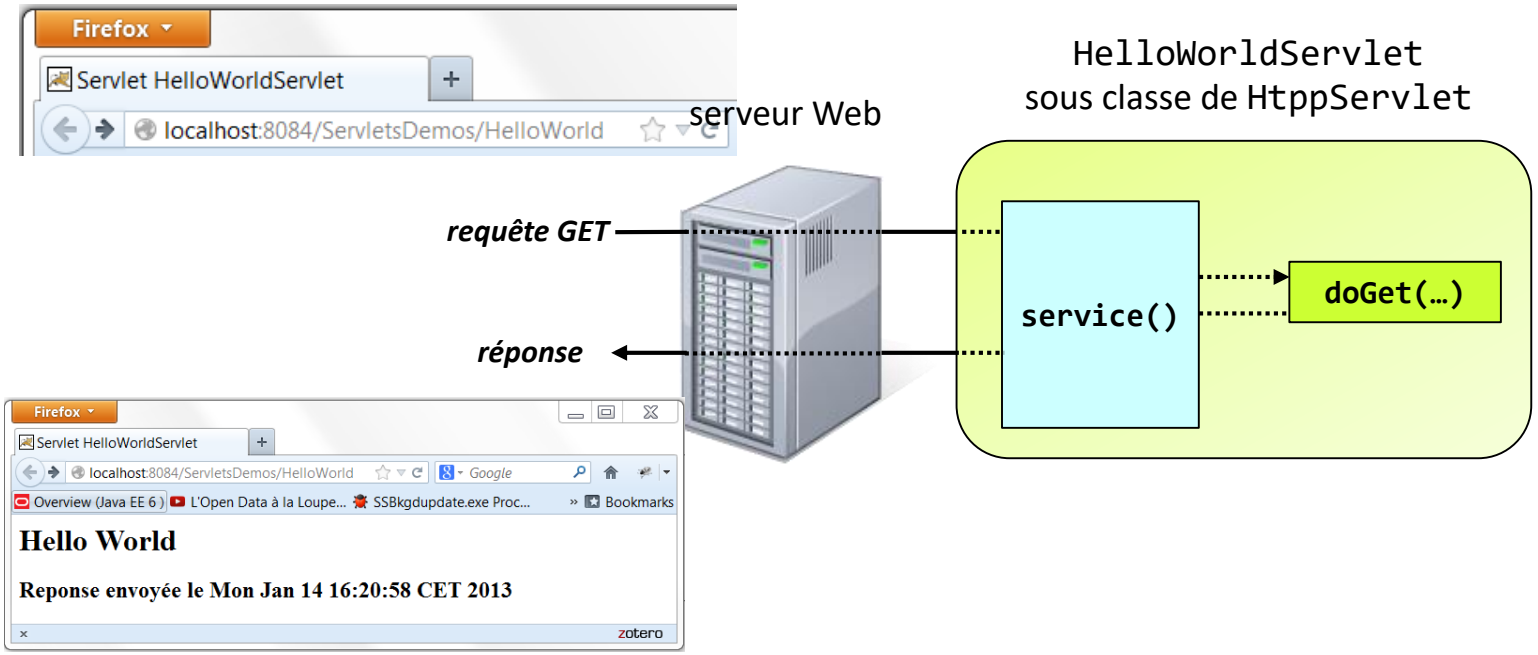
Invoquer la servlet

`http://localhost:8084/ServletsDemos/HelloWorld`

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

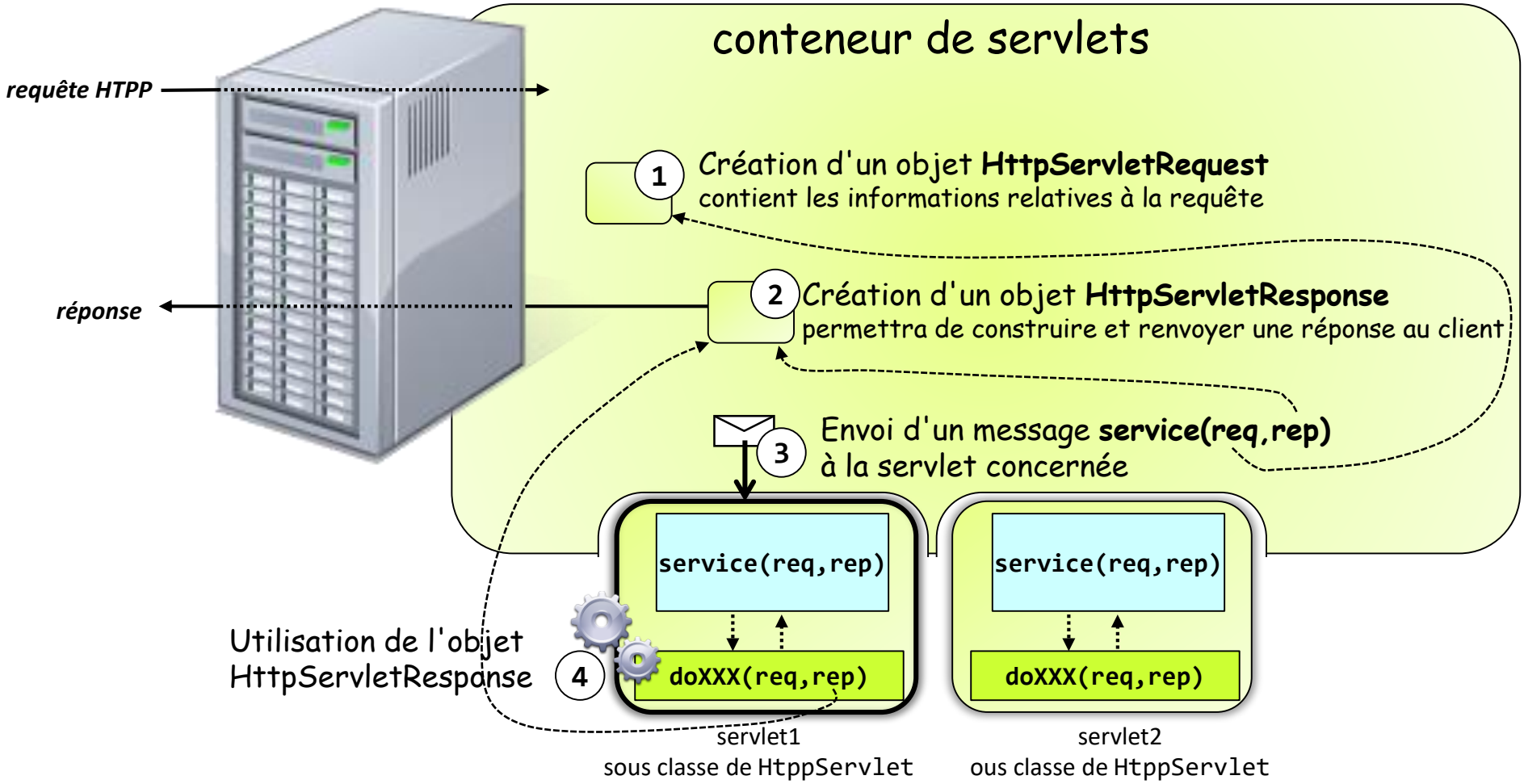
* un serveur peut héberger plusieurs applications

** une application peut être composée de plusieurs servlets



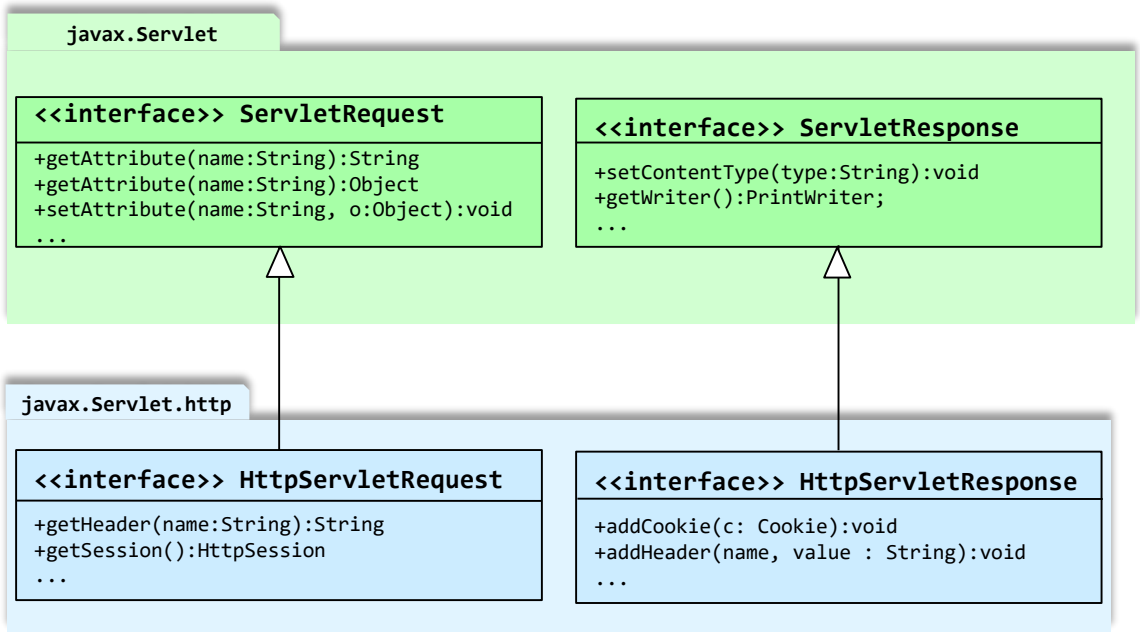
objets HttpServletRequest et HttpServletResponse

- passés en paramètre des méthodes `service()` , doGet() , doPost() , doXXX()…
- instanciés par le conteneur de servlets, qui les transmet à la méthode `service`.



objets HttpServletRequest et HttpServletResponse

- HttpServletRequest :
  - *Contexte de l'appel*
  - *Paramètres de formulaires*
  - *Cookies*
  - *Headers*
  - *. . .*
- HttpServletResponse
  - *Contrôle de la réponse*
  - *Type de données*
  - *Données*
  - *Cookies*
  - *Status*
  - *. . .*

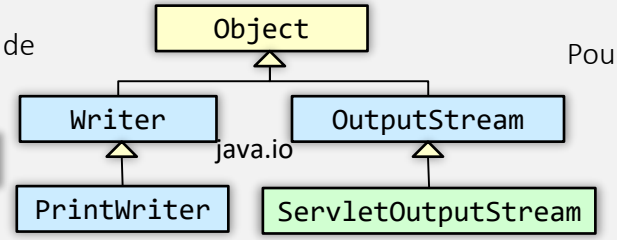


utilisation des objets Request et Response

- 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*
  2. *Accéder éventuellement à des ressources externes*
  3. *Produire une réponse sur la base des informations précédentes*
     a. *Récupérer un flux de sortie (OutputStream) pour l'écriture de la réponse*

Pour envoyer des données sous forme de chaînes de caractères

```
PrintWriter out = resp.getWriter();
```



Pour envoyer des données sous forme binaire

```
ServletOutputStream out
    = resp.getOutputStream();
```

     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

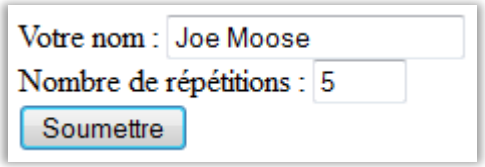
```
byte[] tab = ...;
...
out.write(tab);
out.print(...);
```

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) ;
```

Servlets et formulaires

formulaire HTML

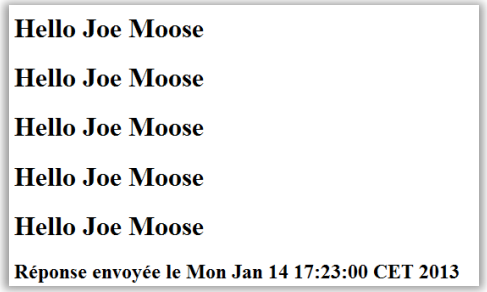


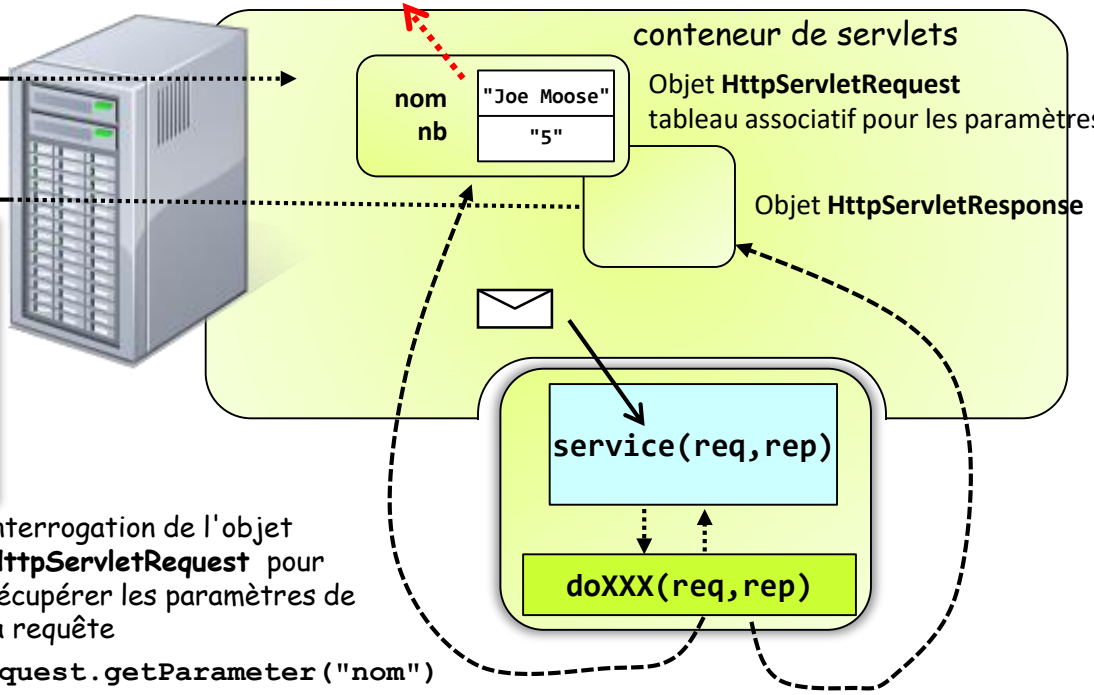
paramètres de la requête

`http://localhost:8084/ServletsDemos/HelloWorld2?nom=Joe+Moose&nb=5`

requête GET/POST

réponse





interrogation de l'objet **HttpServletRequest** pour récupérer les paramètres de la requête

```
request.getParameter("nom")
```

```
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
out.println("<html>");
out.println("<head>");
...
```

utilisation de l'objet **HttpServletResponse** pour construire et envoyer la réponse au client qui a émis la requête

Servlets et formulaires

```
<!DOCTYPE html>
<html>
  <head>
    <title>Formulaire Hello</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <form action="HelloWorld2">
      Votre nom : <input type="text" name="nom" value="" size="20" /><br/>
      Nombre de répétitions : <input type="text" name="nb" value="5" size="3" /><br/>
      <input type="submit" value="Soumettre" />
    </form>
  </body>
</html>
```

Votre nom : Joe Moose
Nombre de répétitions : 5
Soumettre

```
@WebServlet(name = "HelloWorld2Servlet", urlPatterns = {"/HelloWorld2"})
public class HelloWorld2Servlet 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 {
      int nbRepet = Integer.parseInt(request.getParameter("nb"));
      String name = request.getParameter("nom");
      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 " + name + "</h1>");
      }
      out.println("<h2>Réponse envoyée le " + new Date() + "</h1>");
      out.println("</body>");
      out.println("</html>");
    } finally {
      out.close();
    }
  }
}
```

getParameter() **retourne des String**

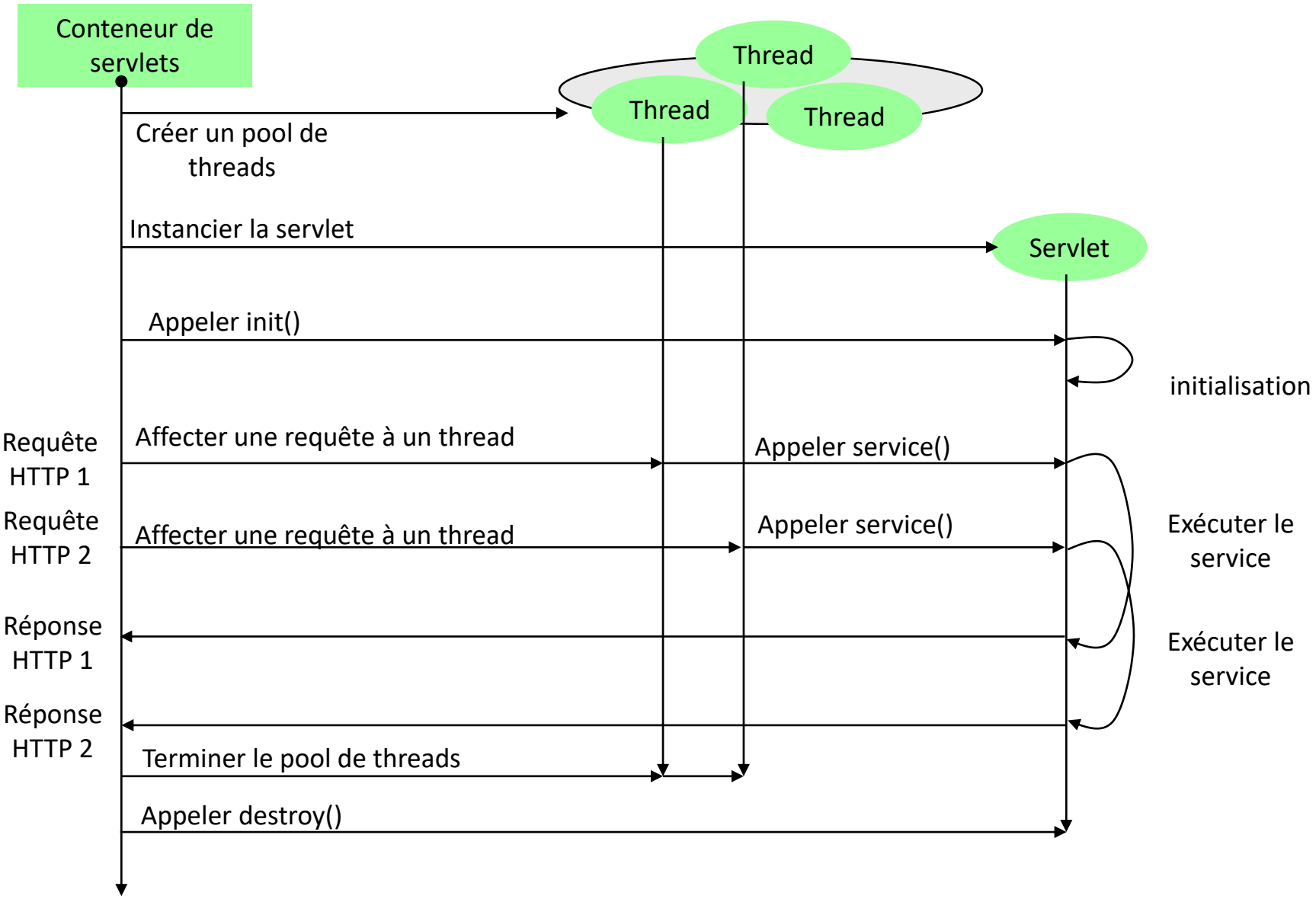
Hello Joe Moose
Hello Joe Moose
Hello Joe Moose
Hello Joe Moose
Hello Joe Moose
Réponse envoyée le Mon Jan 14 17:23:00 CET 2013

Exécution 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*
    - *le conteneur crée un thread (processus léger) par requête pour pouvoir les traiter en parallèle*

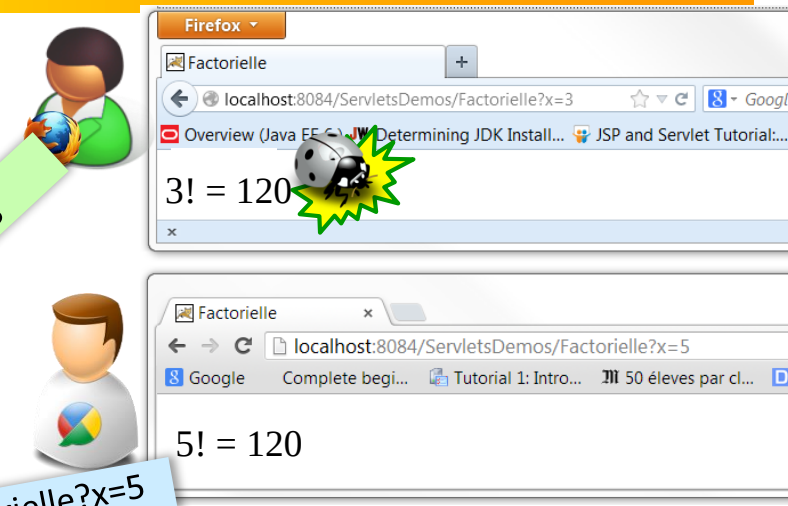
- Attention !! Les attributs de la classe deviennent des ressources partagées
  - *ils sont accédés de manière concurrente par chaque fil d'exécution*

Execution des Servlets



Execution des servlets

```
@WebServlet(name = "Factorielle", urlPatterns = {"/Factorielle"})
public class Factorielle extends HttpServlet {

  protected int nombre;

  protected int factorielle(int x) {
    int res = 1;
    try {
      this.nombre = x;
      Thread.sleep(1000);
      res = 1;
      for (int i = 1; i <= nombre; i++) {
        res *= i;
      }
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    return res;
  }

  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>");
        int x = Integer.parseInt(request.getParameter("x"));
        int fact = factorielle(x);
        out.println(x + "!  =  " + fact);
        out.println("</body>");
        out.println("</html>");
      } finally {
        out.close();
      }
  }
```



Exécution des servlets

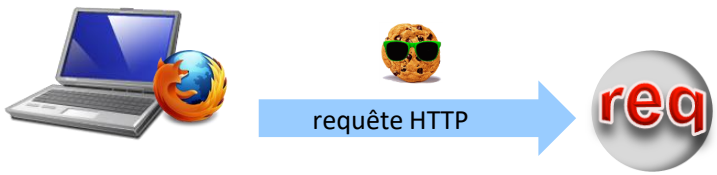
- Solutions
  - *Sections critiques :*
    - *Utilisation du mot clé* ***synchronized*** *(blocs synchronisés ne pouvant être exécutés que par un seul thread à la fois)*
    - *http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html.*
  - *Modèle d'exécution : 1 instance par thread*
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

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.

- classe `javax.servlet.http.Cookie`

Cookie
+Cookie(String name, String value) +getName():String +getMaxAge():int +getValue():String +getDomain():String ... +setValue(String v):void +setMaxAge(int expiry):void +setdomain(String domain):void ...

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



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

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

Sessions

- 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*
- **Session :**
  - *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`*

<<interface>> HttpSession
+getAttribute(name:String):Object +getAttributeNames():Enumeration<String> +getId():int +invalidate():void +isNew():boolean +setAttribute(name:String, o:Object):void +setMaxInactiveInterval(interval:int):void

utilisation de l'objet HttpSession

- Objet HttpSession
  - *pas de classe d'implémentation dans l'API JavaEE (`HttpSession` est une interface)*
  - *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*
  - *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)*
- 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)*

utilisation de l'objet HttpSession

- 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()`
        recupère 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

utilisation de l'objet HttpSession

- Sauvegarde d'objets :
  - *Association clé ⇔ instance :* `void setAttribute(String key, Object value)`
    - *exemple :*

```
HttpSession session = request.getSession();
String name= "Joe Moose";
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");
```

contexte de l'application

- application constituée généralement de plusieurs ressources
  - *servlets, pages JSP (Java Server Pages), pages HTML, images …*
- ces éléments sont regroupés par le serveur dans un conteneur : le **contexte de l'application**
  - *servlets, pages JSP*
  - *informations de configuration (issues du fichier de déploiement web.xml ou des annotations des servlets)*
  - *informations déposées par les servlets ou JSP et partagées par tous*
    - *tous les composants y on accès indépendamment des sessions utilisateur ("variables globales")*
    - *attributs du contexte de l'application*
  - *représenté par un objet instance de l'interface* `http.servlet.ServletContext`

contexte de l'application

- obtention de l'objet `ServletContext`
  - *méthode* `ServletContext getServletContext()` *héritée de* `HttpServlet`

- méthodes de l'objet `ServletContext`
  - *gestion des attributs , même principe que pour la session*
    - `void setAttribute(String key, Object value)`
      déposer un attribut
    - `Object getAttribute(String key)`
      récupérer un attribut
  - *accès aux paramètres de configuration de l'application*
    - `String getInitParameter(String name)`
    - *paramètres déclarés dans le fichier de déploiement (web.xml)*
      - *exemple*

```
<context-param>
      <param-name>admin-mail</param-name>
      <param-value>Joe.Moose@yukon.org</param-value>
</context-param>
```

Utilisation d'autres ressources

- Le traitement d'une requête HTTP par une servlet peut nécessiter l'utilisation d'autres ressources
  - *inclusion d'une page HTML statique*
  - *redirection vers une autre servlet ou page JSP*
- Réalisé à l'aide d'un objet `javax.servlet.RequestDispacher`

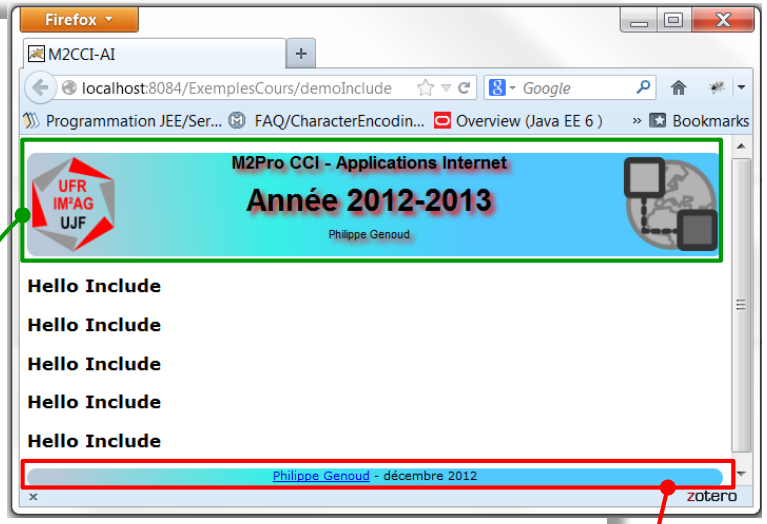
<<interface>> RequestDispatcher
`+ forward(servletRequest req, ServletResponse rep) : void` `+ include(servletRequest req, ServletResponse rep) : void`

- Obtention d'un objet ResquestDispacher
  - *via un objet `javax.servlet.ServletContext`*
  - *via un objet `javax.servlet.http.HttpServletRequest`*
  - *`RequestDispatcher getRequestDispatcher(java.lang.String path)`*
    - ***`path`*** *chaîne de caractères précisant le chemin de la ressource vers laquelle la requête doit être transférée (doit commencer par un '/' et est interprété comme relatif au contexte de l'application)*

RequestDispatcher : include

- souvent utilisé pour insérer dans réponse des fragments de code HTML

```
package fr.im2ag.m2cci.servletsdemo.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import …..

@WebServlet(name = "HelloInclude", urlPatterns = {"/demoInclude"})
public class HelloInclude 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 {
      // inclusion 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(request, response);
    } finally {
      out.close();
    }
  }
}
```

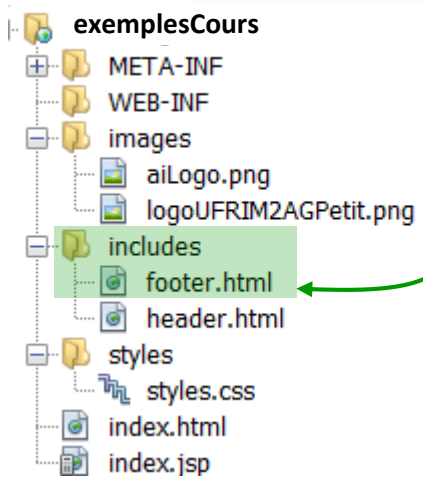


RequestDispatcher : include

localhost:8084/ExemplesCours/demoInclude

```
RequestDispatcher dispatcher =
      getServletContext().getRequestDispatcher("/includes/header.html");
dispatcher.include(request, response);
...
getServletContext().getRequestDispatcher("/includes/footer.html").include(request, response);
```

chemin relatif au contexte de l'application



header.html

```
<html>
<head>
   <title>M2CCI-AI</title>
   <meta charset="utf-8" />
   <link href="./styles/styles.css" rel="stylesheet" type="text/css" />
</head>
<body>
   <header>
      <a href="http://ufrima.imag.fr/">
         <img src="./images/logoUFRIM2AGPetit.png" class="imageAGauche"/>
      </a>
      <a href="http://www.ujf-grenoble.fr">
         <img src="./images/aiLogo.png" class="imageADroite"/>
      </a>
      <h3>M2Pro CCI - Applications Internet</h3>
      <h1 class="titreTP">Ann&eacute;e 2012-2013</h1>
      <p class="auteur">Philippe Genoud</p>
   </header>
```

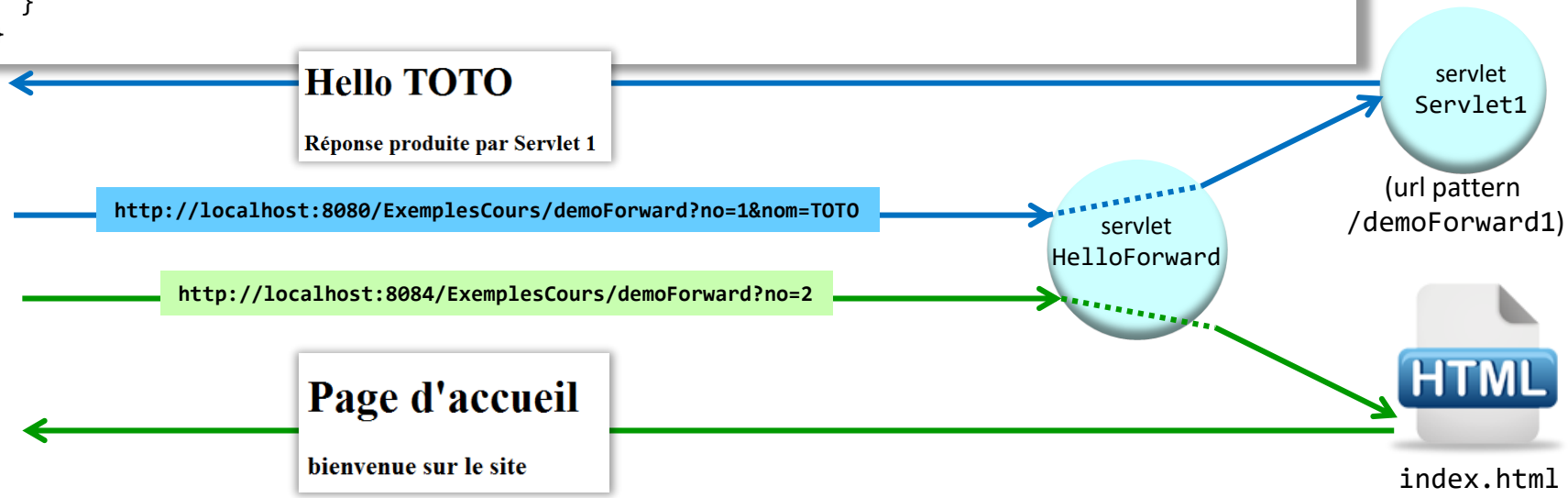
footer.html

```
    <footer>
        <a href="mailto:Philippe.Genoud@imag.fr">Philippe Genoud<a> - d&eacute;cembre 2012
    </footer>
</body>
</html>
```

RequestDispatcher : forward

```
package fr.im2ag.m2cci.servletsdemo.servlets;

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);
    }
  }
}
```

- redirige vers une autre ressource pour produire la réponse.

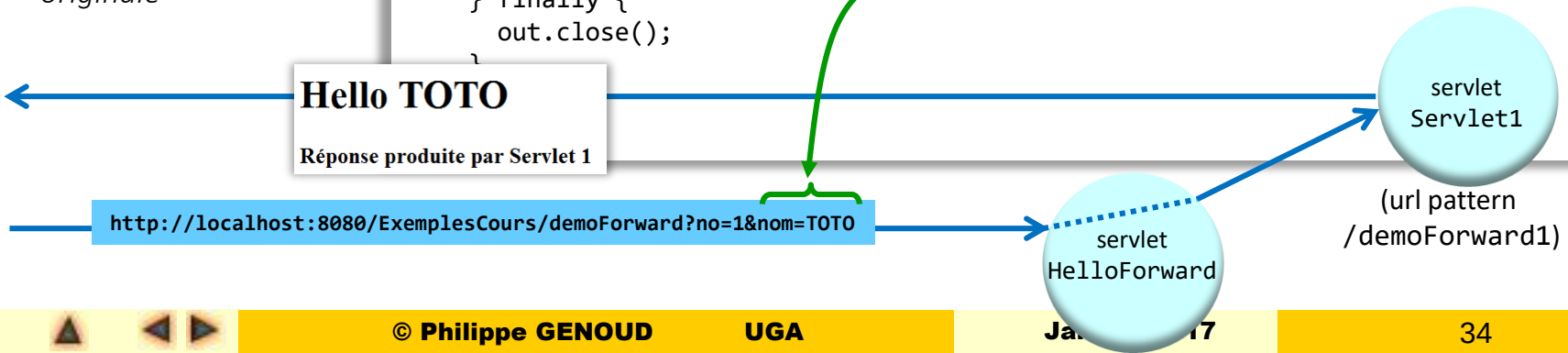


RequestDispatcher : forward

```
@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().ge
    }
  }
```

```
...
@WebServlet(name = "Servlet1", urlPatterns = {"/demoForward1"})
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 1</h3>");
      out.println("</body>");
      out.println("</html>");
    } finally {
      out.close();
```

- les objets **request** et **response** transmis à la ressource vers laquelle la requête est redirigée sont ceux de la servlet initiale
  - *paramètres de **request** sont ceux de la requête originale*



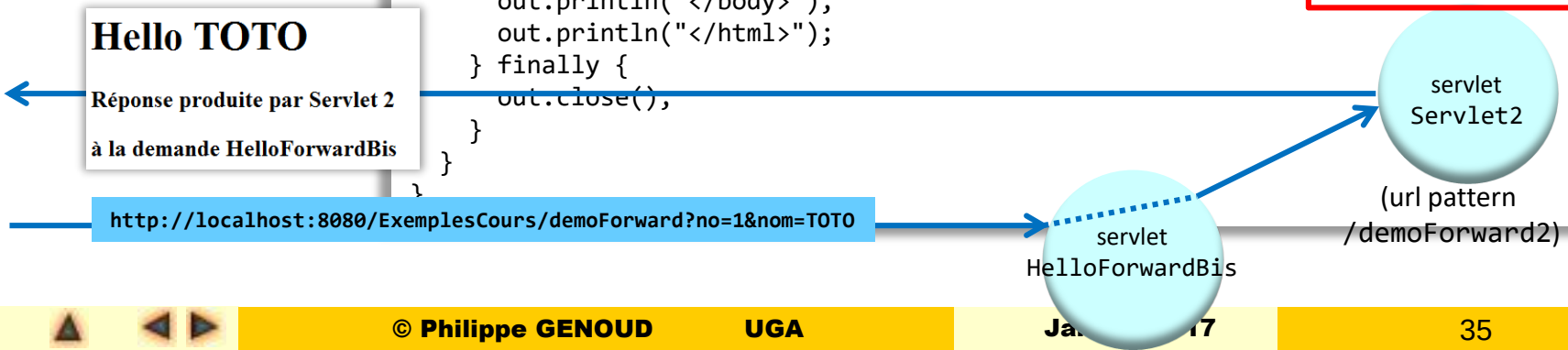
RequestDispatcher : forward

```
@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().ge
    }
  }
```

```
...
@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();
    }
  }
}
```

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

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



RequestDispatcher : forward

- 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>");
        out.println("</html>");
    1   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>");
        out.println("</html>");
      }
} finally {
      out.close();
}
```

2 *si on ne force pas le vidage du buffer par `reponse.flushBuffer()`*

3

Filtres

- Introduits à partir de la version 2.3 de l'API des servlets les filtres permettent d'intercepter les requêtes et réponses des servlets
  - *possibilité d'ajouter un traitement*
    - *avant que la requête ne soit reçue par une servlet*
    - *avant que la réponse ne soit transmise au client*

- Nombreuses utilisations des filtres :
  - *authentificiation*
  - *redirection*
  - *modification des entêtes*
  - *cryptage des données*
  - *contrôle des accès effectués par les clients*
  - *journalisation des accès…*

- Possibilité d'exécuter plusieurs filtre successivement sur le trajet d'une requête ou d'une réponse HTTP

Pour toutes les JSP et Servlets de l'application ne faire le traitement que si une session est présente et que l'objet client est présent

```
HttpSession session = request.getSession(false);

if (session == null || session.getAttribute("client") == null) {
        request.getRequestDispatcher(
                "/accueil.jsp").forward(request, response);
        return;
}
```

Mettre en œuvre ce traitement pour toutes les jsp
et les servlets `TransfertServlet` et `LogoutServlet`

Pour mettre en place ce traitement facilement utiliser un **filtre de servlets**

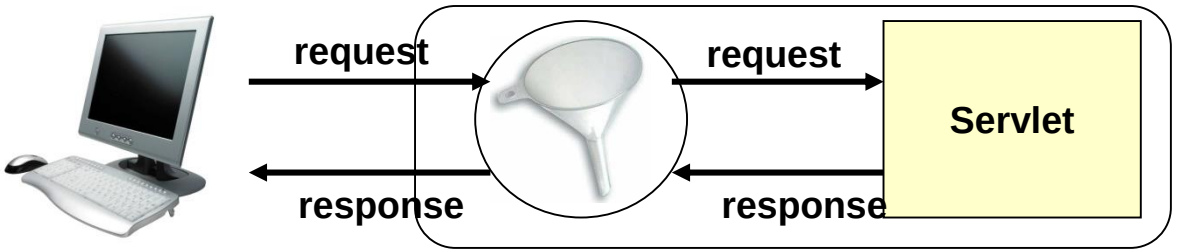
Introduits à partir de la version 2.3 de l'API des servlets les filtres permettent d'intercepter les requêtes et réponses des servlets

Nombreuses utilisations des filtres :

- authentificiation
- redirection
- modification des entêtes
- cryptage des données
- …

Un filtre : classe Java qui implémente l'interface `javax.servlet.Filter`

Method Summary	
void	**destroy**() Called by the web container to indicate to a filter that it is being taken out of service.
void	**doFilter**(ServletRequest request, ServletResponse response, FilterChain chain) The doFilter method of the Filter is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.
void	**init**(FilterConfig filterConfig) Called by the web container to indicate to a filter that it is being placed into service.




```
public class SimpleFilter implements Filter {

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {

      HttpServletRequest httpRequest = (HttpServletRequest) request;
      System.out.println("Dans SimpleFilter  filtrage de la requete " + httpRequest.getRequestURL());

      chain.doFilter (request, response);

      System.out.println ("Dans SimpleFilter filtrage de la réponse ...");
    }

    ...
}
```

Appel au prochain objet (un filtre ou la servlet) dans la chaine de filtrage

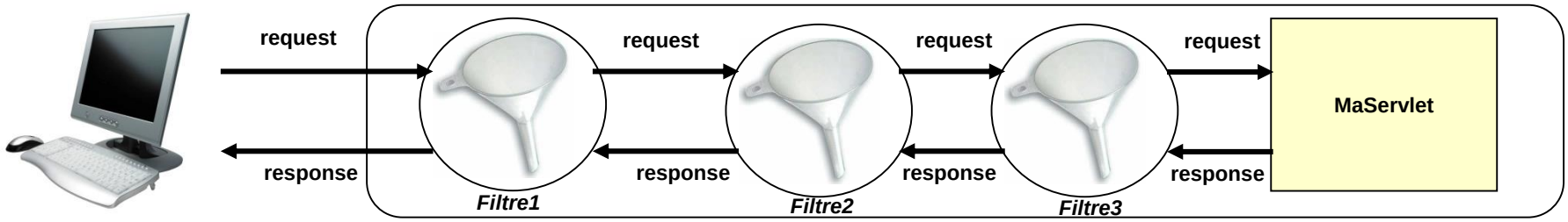
avec JavaEE 6 possibilité d'utiliser les annotations

Le déploiement des filtres s'effectue au travers du fichier web.xml

```
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" ...>

    <!- Definition des filtres présents dans l'application Web -->
    <filter>
        <filter-name>SimpleFilter</filter-name>
        <filter-class>bima.filters.SimpleFilter</filter-class>
    </filter>
    ...

    <!- association des filtres à des servlets ou des urls -->
    <filter-mapping>
        <filter-name>SimpleFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

   <!-- Définition des servlets -->
   <servlet>
        <servlet-name>LoginServlet</servlet-name>
        <servlet-class>bima.controler.LoginServlet</servlet-class>
   </servlet>
   ...
   <!- servlets mappings -->
   ...
</web-app>
```

Filtres

Plusieurs filtres peuvent être définis et appliqués à la même URL.
Chaînage effectué par appel à `chain.doFilter (request, response);`




```
public class Filtre1Class implements Filter {

  public void doFilter(ServletRequest request,
                  ServletResponse response, FilterChain chain)
          throws IOException, ServletException {

    System.out.println("Dans Filtre1 filtrage de la requete" );
    chain.doFilter (request, response);
```

```
public class Filtre2Class implements Filter {

  public void doFilter(ServletRequest request,
                  ServletResponse response, FilterChain chain)
          throws IOException, ServletException {

    System.out.println("Dans Filtre2 filtrage de la requete" );
    chain.doFilter (request, response);
```

```
public class Filtre3Class implements Filter {

  public void doFilter(ServletRequest request,
                  ServletResponse response, FilterChain chain)
          throws IOException, ServletException {

    System.out.println("Dans Filtre3 filtrage de la requete" );
    chain.doFilter (request, response);
    System.out.println ("Dans Filtre3 f
  }

    ...
}
```

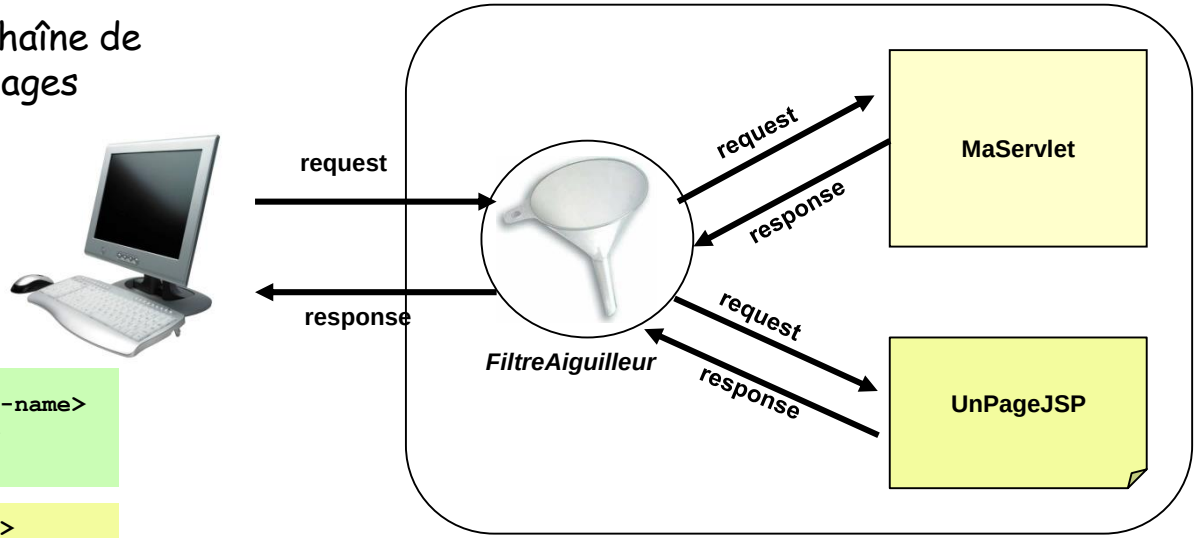
```
Dans Filtre1 filtrage de la requete
Dans Filtre2 filtrage de la requete
Dans Filtre3 filtrage de la requete
Dans Filtre3 filtrage de la réponse
Dans Filtre2 filtrage de la réponse
Dans Filtre1 filtrage de la réponse
```

```
<!- Definition des filtres présents -->
<filter>
    <filter-name>Filtre1</filter-name>
    <filter-class>Filtre1Class</filter-class>
</filter>
<filter>
    <filter-name>Filtre3</filter-name>
    <filter-class>Filtre3Class</filter-class>
</filter>
<filter>
    <filter-name>Filtre2</filter-name>
    <filter-class>Filtre2Class</filter-class>
</filter>
```

```
<!- association des filtres à des urls -->
<filter-mapping>
    <filter-name>Filtre1</filter-name>
    <url-pattern>/uneURL</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>Filtre2</filter-name>
    <url-pattern>/uneURL</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>Filtre3</filter-name>
    <url-pattern>/uneURL</url-pattern>
</filter-mapping>
```

L'ordre d'exécution est défini par l'ordre des mappings dans le descripteur de déploiement

Un filtre peut "courcircuiter" la chaîne de filtrage pour effectuer des aiguillages




```
<!- Definition des filtres présents -->
<filter>
    <filter-name>FiltreAiguilleur</filter-name>
    <filter-class>UnFiltre</filter-class>
</filter>
```

```
<!- association des filtres à des urls -->
<filter-mapping>
    <filter-name>FiltreAiguilleur</filter-name>
    <url-pattern>/maServlet</url-pattern>
</filter-mapping>
```

```
public class UnFiltre implements Filter {

  public void doFilter(ServletRequest request,
                  ServletResponse response, FilterChain chain)
          throws IOException, ServletException {

    if (condition) {
       RequestDispatcher rd = request.getRequestDispatcher("/UnePageJSP.jsp");
       rd.dispatch(request,respeonse);
    }
    else
       chain.doFilter (request, response);
  }

    ...
}
```

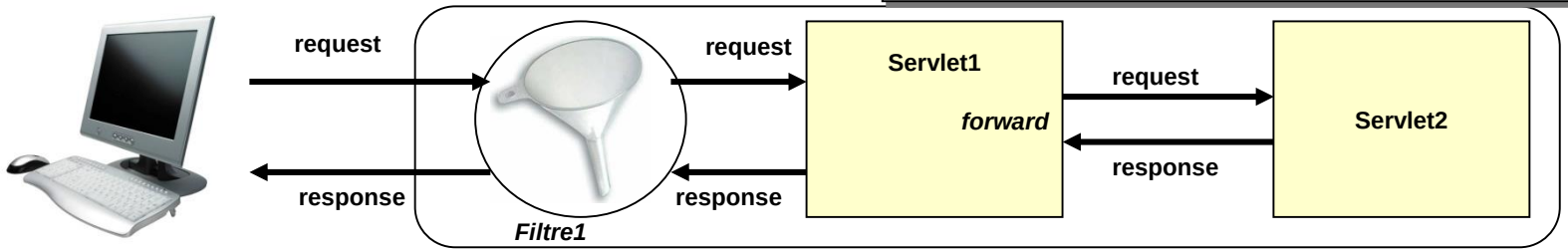
Par défaut le filtre n'est appliqué qu'aux requêtes issues directement du client mais pas aux requêtes forwardées

```
public class Servlet1 extends HttpServlet {

   protected void doGet(HttpServletRequest request, HttpServletResponse response
                                                  throws ServletException, IOExceptio
      System.out.println("Dans servlet 1");
      request.getRequestDispatcher("/Servlet2").forward(request,response);
   }

   …
}
```

```
public class Servlet2 extends HttpServlet {

   protected void doGet(HttpServletRequest request, HttpServletResponse response
                                                  throws ServletException, IOExceptio
      System.out.println("Dans servlet 2");
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      out.println("<html>");
      out.println("<head>");
      out.println("<title>Servlet Servlet2</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h1>Servlet Servlet2 at " + request.getContextPath () + "</h1>");
      out.println("</body>");
      out.println("</html>");
      out.close();
   }

   …
}
```

```
Dans Filtre1  filtrage de la requete http://localhost:8090/Test/Servlet1
Dans servlet 1
Dans servlet 2
Dans Filtre1  filtrage de réponse
```

```
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" ...>

   <!– Definition des filtres présents dans l'application Web -->
   <filter>
      <filter-name>Filtre1</filter-name>
      <filter-class>filters.Filter1</filter-class>
   </filter>

   <filter>
      <filter-name>Filtre2</filter-name>
      <filter-class>filters.Filter2</filter-class>
   </filter>

   <!– association des filtres à des servlets ou des urls -->
   <filter-mapping>
      <filter-name>Filtre1</filter-name>
      <url-pattern>/Servlet1</url-pattern>
   </filter-mapping>

   <filter-mapping>
      <filter-name>Filtre2</filter-name>
      <url-pattern>/Servlet2</url-pattern>
   </filter-mapping>

   <!–– Définition des servlets -->
   <servlet>
      <servlet-name>Servlet1</servlet-name>
      <servlet-class>servlet.Servlet1</servlet-class>
   </servlet>
   …
   <!– servlets mappings -->
   …
</web-app>
```



Filtres

Depuis version 2.4 des servlets possibilité de contrôler dans le fichier de déploiement Dans quel "contexte de dispatching" les filtres doivent être invoqués

Filtre appliqué :
- au requêtes issues du client
- au requêtes issues d'un forward

```
<dispatcher>REQUEST</dispatcher>
<dispatcher>FORWARD</dispatcher>
```

```
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" ...>
    <!– Definition des filtres présents dans l'application Web -->
    <filter>
       <filter-name>Filtre1</filter-name>
       <filter-class>filters.Filter1</filter-class>
    </filter>
    <filter>
       <filter-name>Filtre2</filter-name>
       <filter-class>filters.Filter2</filter-class>
    </filter>
    <!– association des filtres à des servlets ou des urls -->
    <filter-mapping>
       <filter-name>Filtre1</filter-name>
       <url-pattern>/Servlet1</url-pattern>
    </filter-mapping>
    <filter-mapping>
       <filter-name>Filtre2</filter-name>
       <url-pattern>/Servlet2</url-pattern>
    </filter-mapping>
    <!–– Définition des servlets -->
    <servlet>
       <servlet-name>Servlet1</servlet-name>
       <servlet-class>servlet.Servlet1</servlet-class>
    </servlet>
    …
    <!– servlets mappings -->
    ...
</web-app>
```

```
Dans Filtre1  filtrage de la requete http://localhost:8090/Test/Servlet1
Dans servlet 1
Dans Filtre2  filtrage de la requete http://localhost:8090/Test/Servlet2
Dans servlet 2
Dans Filtre2   filtrage de la réponse
Dans Filtre1  filtrage de réponse
```