Langages Web 1

Programmation compilée coté serveur Servlets et JSP

Florent Nicart

Université de Rouen

2012–2013

Previously in LW1...

- Nous avons vu comment programmer des applications serveur avec le langage PHP,
- un langage de script, interprété, même s'il est parfois compilé à la volé (just-in-time compilation),
- qui permet d'incorporer des instructions au langage de description HTML,
- en utilisant une interface de communication avec le serveur HTTP : C.G.I.

Programmons avec Java :

- programmes compilés mais portables,
- la compilation apporte des vérifications (ex : typage fort).

Programmation coté serveur

Programmes compilés avec Java

- Java offre un modèle de programmation coté serveur et une architecture sensiblement différents.
- Il s'appuit sur JEE [1] : inversion de contrôle, conteneur d'exécution offrant des API de services, déploiements multiples, gestion de la montée en charge, ...
- Plusieurs approches :
  - Les **servlets** : de simples classes traitant des requêtes, elles sont la base des autres techniques.
  - **JSP** : une technique permettant d'écrire du code Java dans des pages web (à la PHP),
  - les **JavaServer Faces** : un framework agnostique au langage de présentation (mais souvent utilisé avec HTML).
- Dans tous les cas, un seul langage : **Java**.

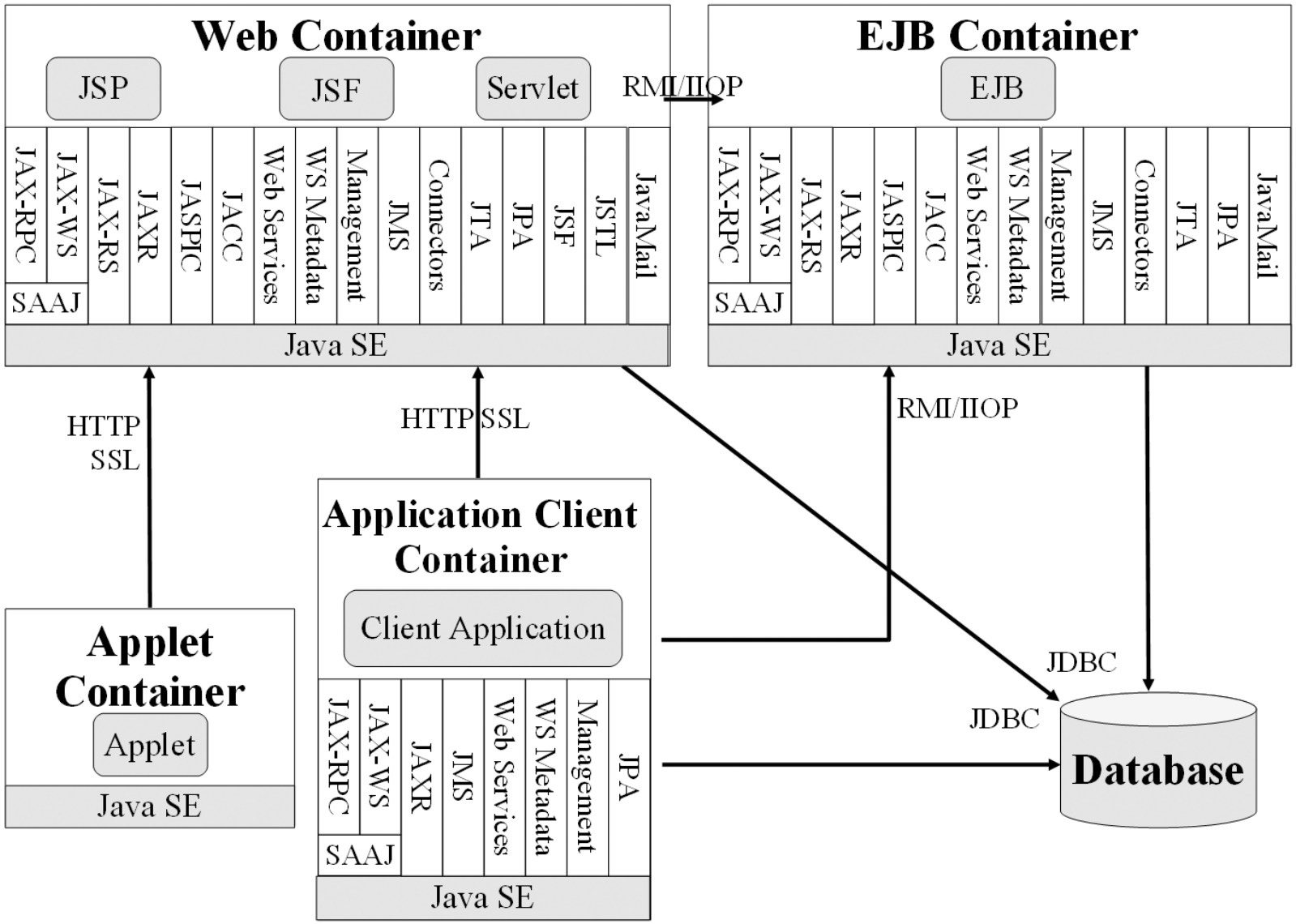
1. Java Enterprise Edition.

Le conteneur Web

- Un *conteneur web* est nécessaire pour déployer et exécuter une *servlet*.
- Il est responsable du cycle de vie de la servlet : création, destruction, connection aux ressources, etc.
- Le conteneur associe une *URL* à une servlet et lui transmet les requêtes en invoquant des méthodes de la classe servlet (un patron proxy est utilisé ici).
- Le conteneur est un environnement d'exécution composé d'une machine virtuelle et d'APIs qui déchargent le programmeur des tâches récurrentes.

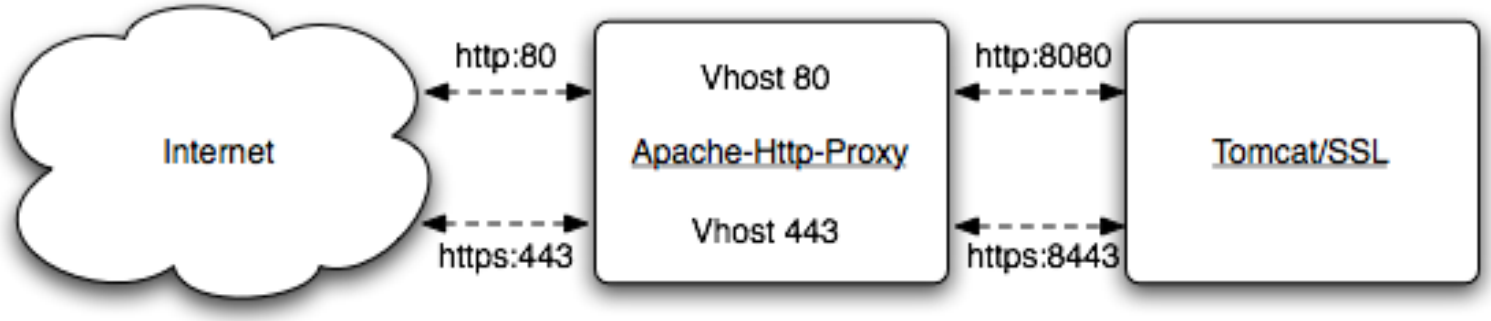
L'environnement JEE

Le conteneur Web



Une architecture différente

Dans le cas de Tomcat



- Tomcat [2] embarque son propre serveur HTTP,
- mais peut être conjugué à un serveur http classique en configurant ce dernier comme proxy, ou en utilisant un connecteur AJP (*Apache Jserv Protocol*).
- C.G.I. n'est pas utilisé, la gestion des processus est différente (voir *cycle de vie des servlets*).

2. Conteneur Web de la fondation Apache.

Les servlets

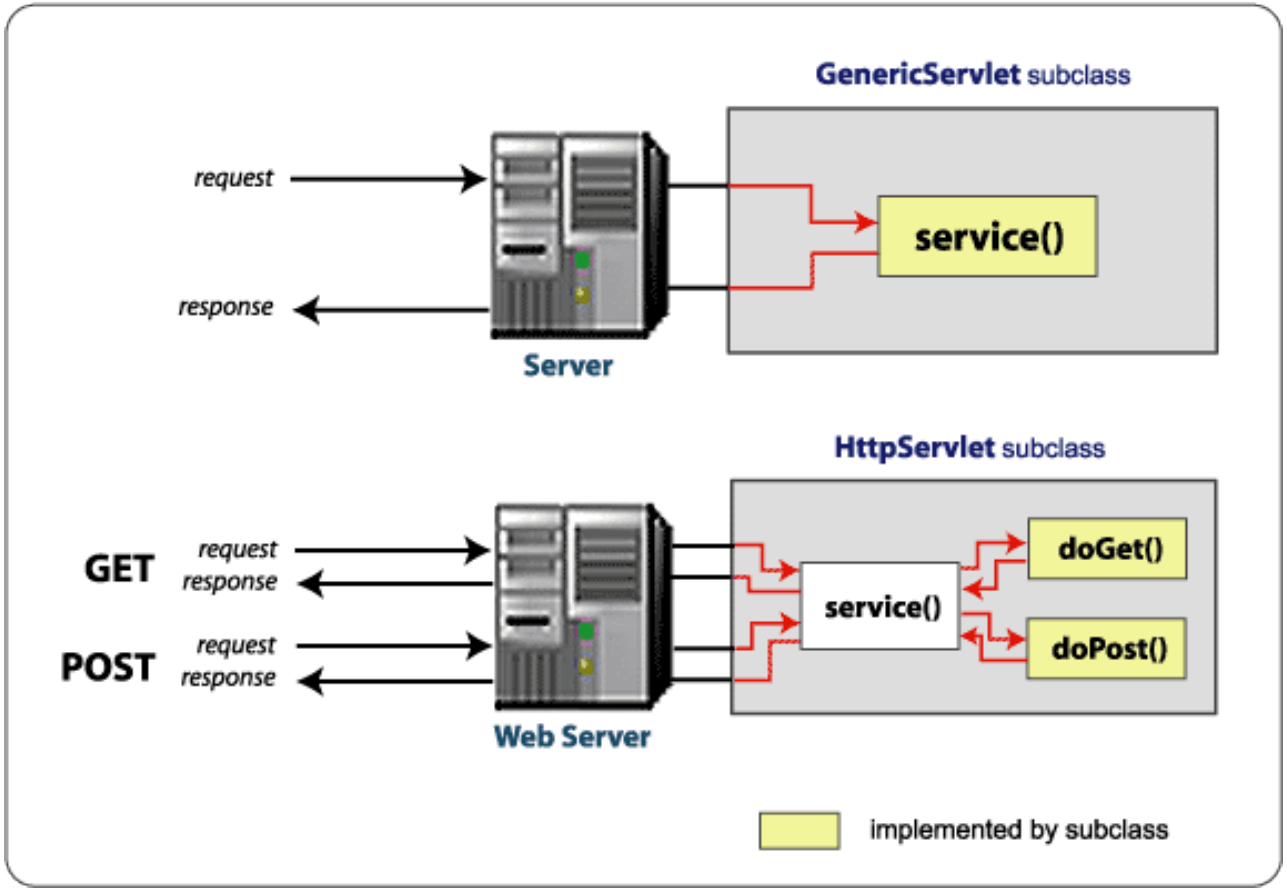
Définition (Servlet)

Une servlet *est une classe Java EE qui implémente des fonctionnalités serveur.*

- L'API servlet est défini dans le package `javax.servlet`
- et défini de quelle manière une servlet interagit avec le conteneur pour traiter une requête.
- Les servlets utilisent l'inversion de contrôle : elles sont incorporées comme composant du conteneur, c'est ce dernier qui constitue le programme principal.
- Leurs callbacks sont invoquées selon un modèle de programmation requête/réponse.

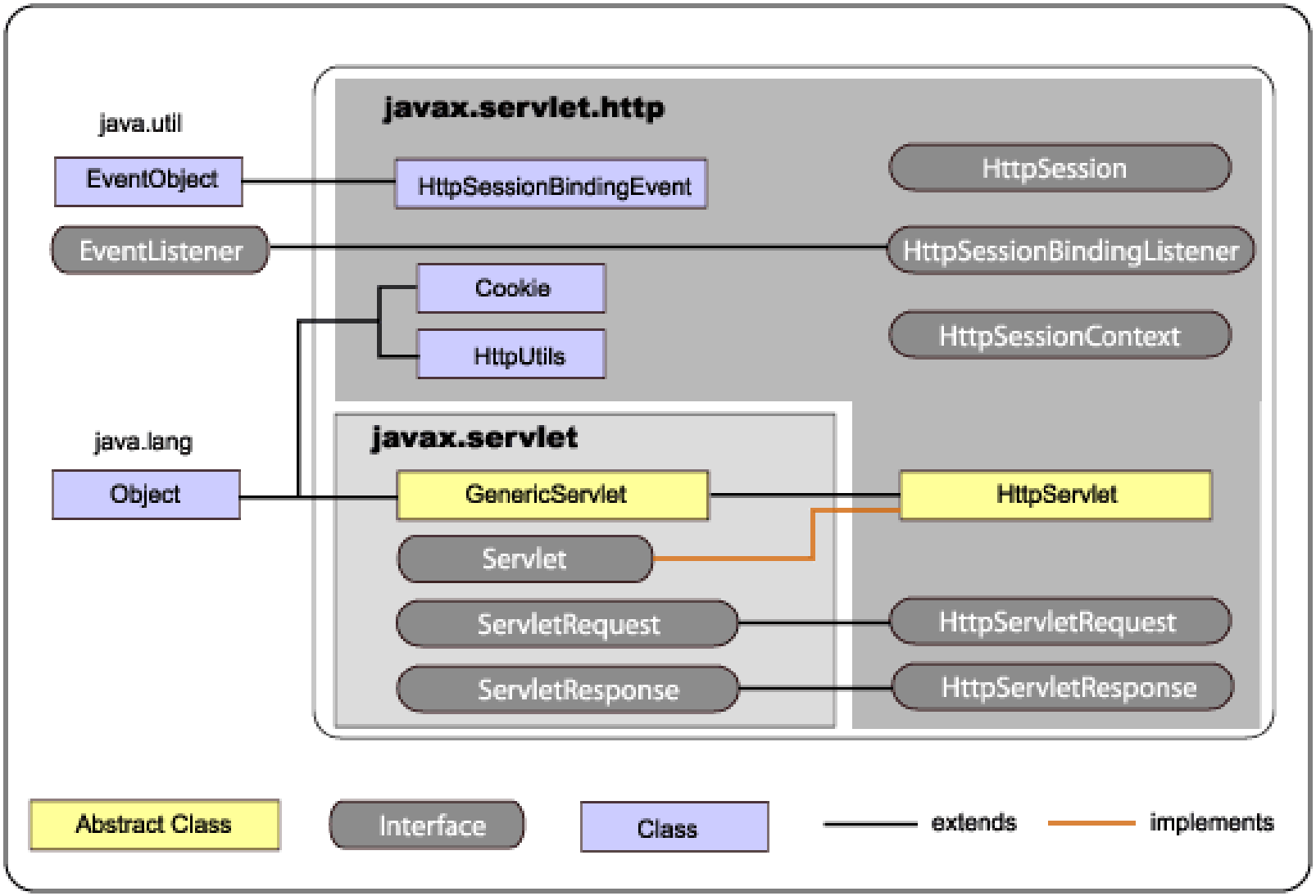
Protocoles

- Les servlets sont indépendantes du protocole client/serveur mais sont souvent utilisées avec HTTP :

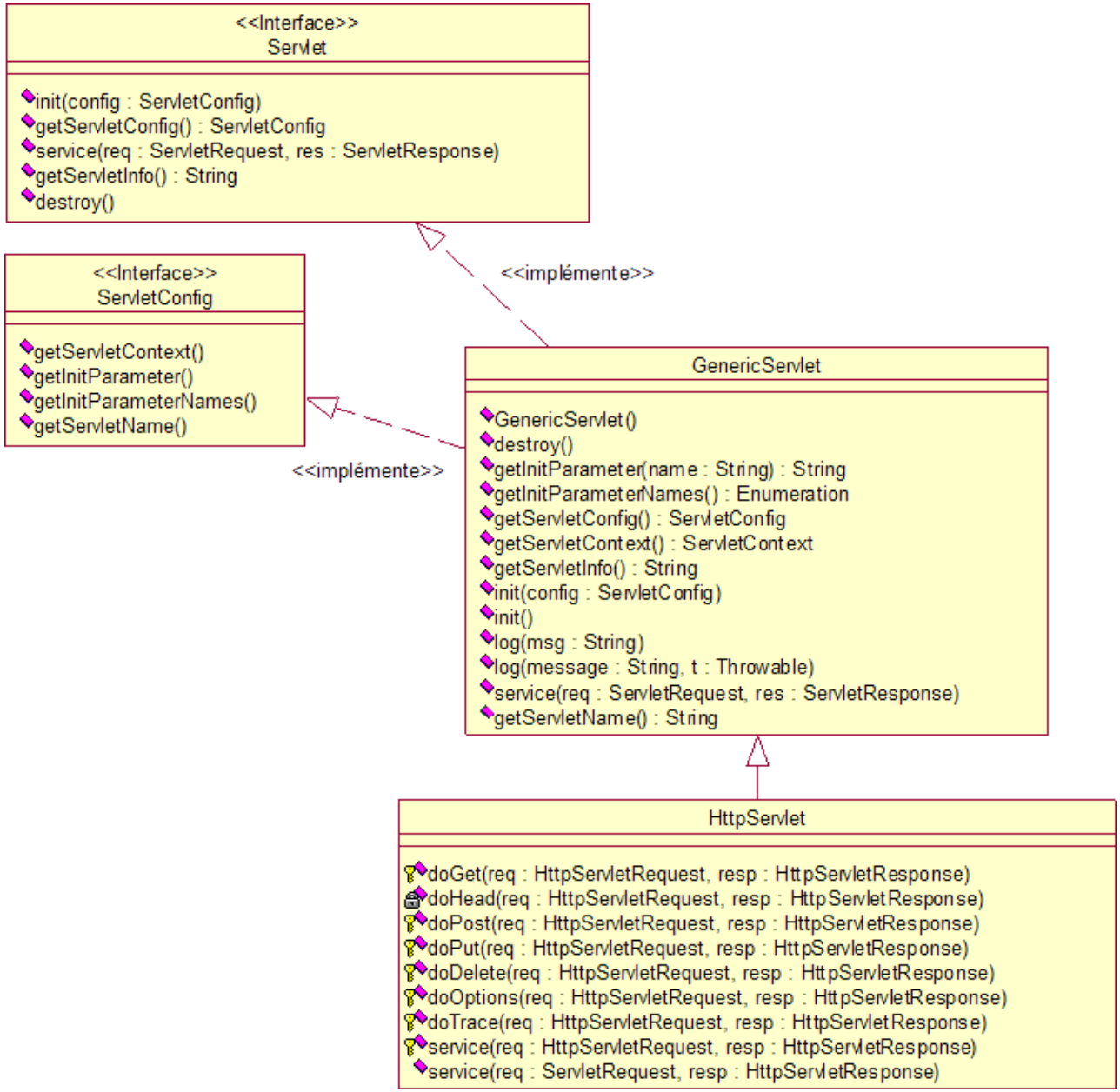


L'API servlet

- L'API servlet et sa variante web :



Les classes de servlet



L'interface `Servlet`

- `void destroy()` : called by the servlet container to indicate to a servlet that the servlet is being taken out of service.
- `ServletConfig getServletConfig()` : returns a ServletConfig object, which contains initialization and startup parameters for this servlet.
- `String getServletInfo()` : returns information about the servlet, such as author, version, and copyright.
- `void init(ServletConfig config)` : called by the servlet container to indicate to a servlet that the servlet is being placed into service.
- `void service(ServletRequest req, ServletResponse res)` : called by the servlet container to allow the servlet to respond to a request.

L'interface `ServletConfig`

- `String getInitParameter(String name)` : gets the value of the initialization parameter with the given name.
- `Enumeration<String> getInitParameterNames()` : returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.
- `ServletContext getServletContext()` : returns a reference to the ServletContext in which the caller is executing.
- `String getServletName()` : returns the name of this servlet instance.

La classe abstraite `GenericServlet`

Méthodes ajoutées :

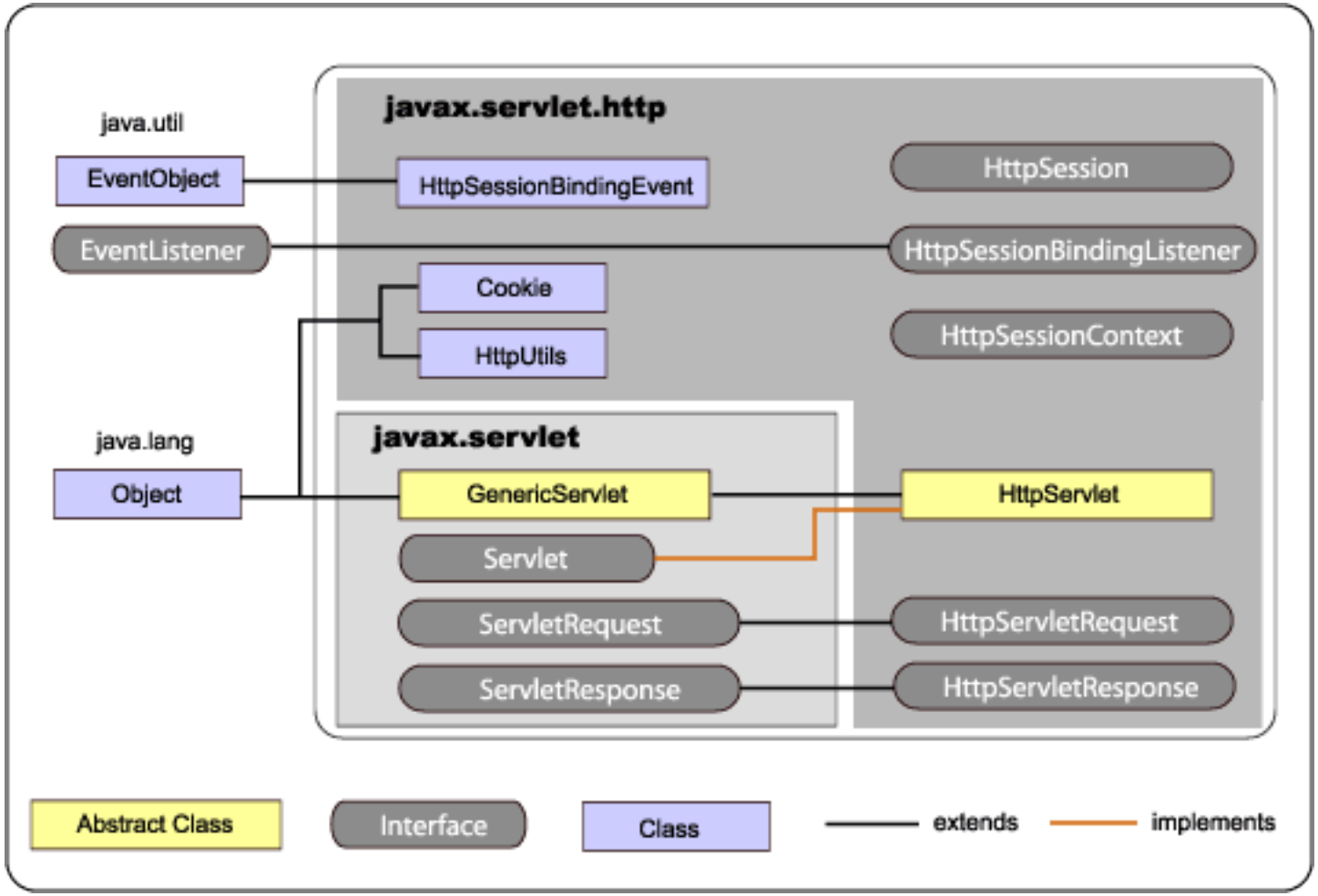
- `String getInitParameter(String name)` : the named initialization parameter, or null if the parameter does not exist.
- `Enumeration<String> getInitParameterNames()` : the names of the servlet's initialization parameters.
- `ServletContext getServletContext()` : a reference to the ServletContext in which this servlet is running.
- `String getServletInfo()` : information about the servlet, such as author, version, and copyright.
- `String getServletName()` : returns the name of this servlet instance.
- `void log(String msg)` : writes the specified message to a servlet log file.

Interfaces `ServletRequest` et `ServletResponse`

- `ServletRequest` permet d'accéder aux informations de la requête. Javadoc : docs.oracle.com/javaee/6/api/javax/servlet/ServletRequest.html
- `ServletResponse` permet de générer la réponse à la requête. Javadoc : docs.oracle.com/javaee/6/api/javax/servlet/ServletResponse.html

Les composants web

- La variante web redéfini les composants génériques et en ajoute d'autres :



La classe abstraite `HttpServlet`

- `void do`ϕ`(HttpServletRequest req, HttpServletResponse resp)` : appelée par le serveur (à travers la méthode `service()` pour traiter une requête de type $\phi \in \{$*Delete*, *Get*, *Head*, *Options*, *Post*, *Put*, *Trace*$\}$
- `long getLastModified(HttpServletRequest req)` : durée écoulée depuis la dernière modification de l'objet `HttpServletRequest` en millisecondes depuis minuit le 1er January 1, 1970 GMT.
- `void service(HttpServletRequest req, HttpServletResponse resp)` : reçoit les requêtes HTTP et les distribue sur les méthodes ϕ.

Interfaces

`HttpServletRequest`

Quelques méthodes (voir la javadoc) :

- `Cookie[] getCookies()` : les cookies envoyés par le client avec la requête.
- `String getHeader(String name)` : la valeur du champs d'en-tête HTTP spécifié.
- `String getMethod()` : le nom de la méthode utilisée (GET, POST, ou PUT).
- `HttpSession getSession()` : la session associée à la requête.
- `HttpSession getSession(boolean create)` : idem avec création.
- `isRequestedSessionIdFromCookie()`, `isRequestedSessionIdFromURL()`, `isRequestedSessionIdValid()` : tests sur les identifiants de session.

Interfaces

`HttpServletResponse`

Quelques méthodes (voir la javadoc) :

- `void addCookie(Cookie cookie)` : ajoute un cookie à la réponse.
- `void addHeader(String name, String value)` : ajoute un champs à l'en-tête de la réponse.
- `void sendRedirect(String location)` : envoie une réponse de redirection au client.
- `void setStatus(int sc)` : définit le code de status de la réponse.
- `PrintWriter getWriter()` : renvoie un PrintWriter pour générer la réponse.
- `ServletOutputStream getOutputStream()` : flux de sortie approprié aux données binaires.
- `void setContentType(String type)` : définit le type MIME de la réponse.
- ...

La classe `Cookie`

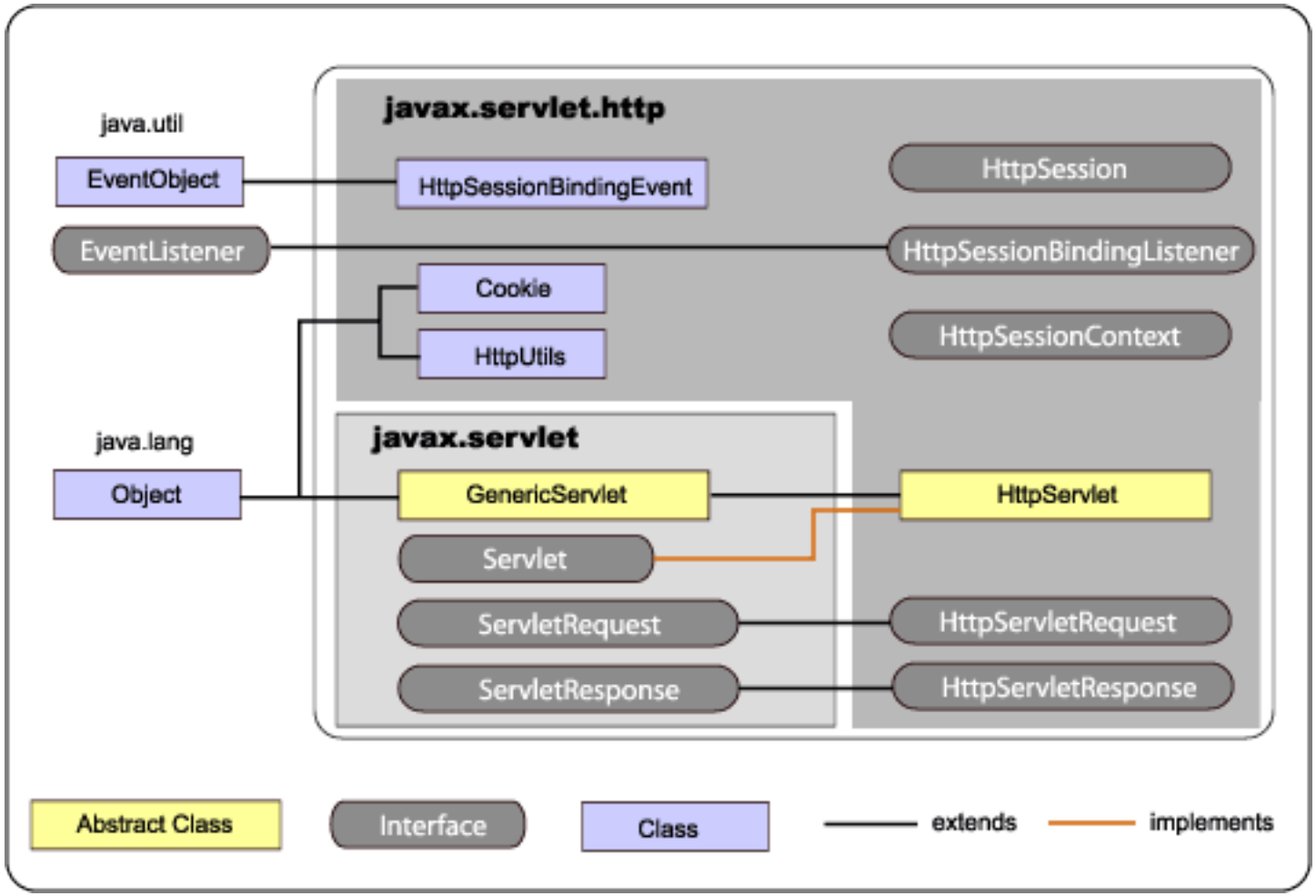
- `String getDomain(),void setDomain(String domain)` : lit/définit le domaine du cookie.
- `boolean getSecure(),void setSecure(boolean flag)` : transmission sur protocole sécurisé seulement.
- `int getMaxAge() ,void setMaxAge(int expiry)` : age maximum en secondes.
- `String getName()` : le nom du cookie.
- `String getValue(),void setValue(String newValue)` : lit/définit la valeur du cookie.
- ...

L'interface `HttpSession`

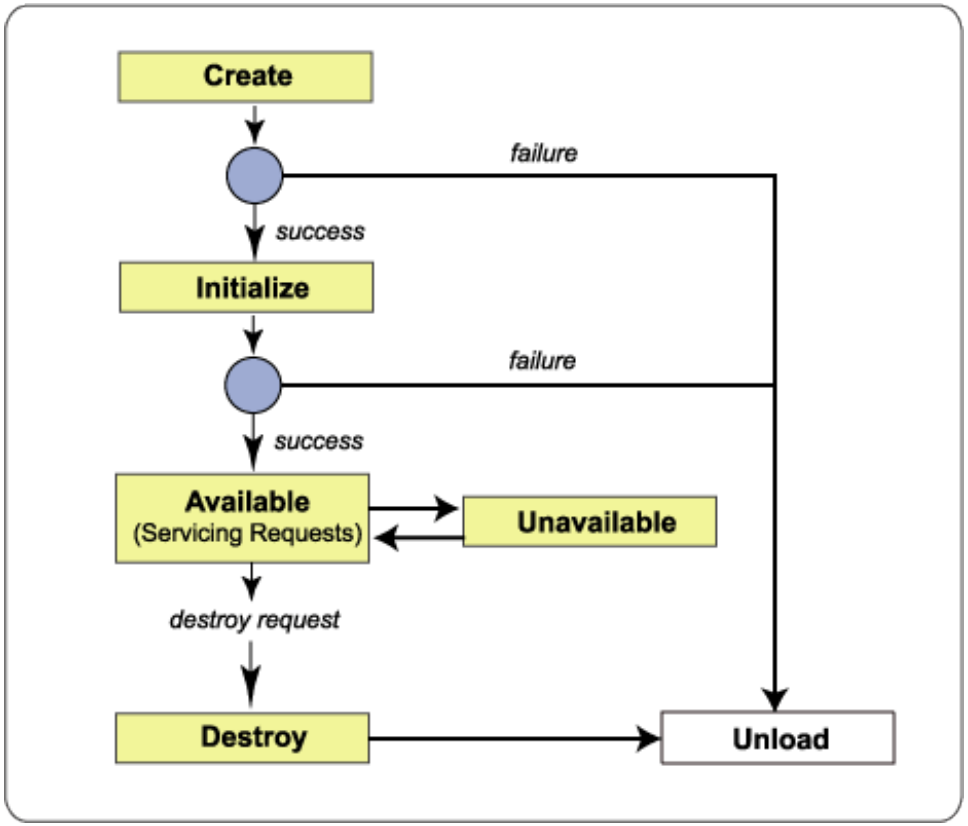
- `Object getAttribute(String name)` : valeur de l'attribut *name* ou null.
- `long getCreationTime()` : date de création de la session en millisecondes depuis le 1er Janvier 1970.
- `String getId()` : retourne l'identifiant de session.
- `boolean isNew()` : indique si la session vient d'être crée (i.e. pas liée au client).
- `void invalidate()` : invalide la session et désassocie les objets.
- `void setAttribute(String name, Object value)` : lie un objet à une variable de session.
- `void setMaxInactiveInterval(int interval)` : temps d'inactivité du client avant invalidation de la session.
- ...

Les composants web

- Note, `HttpSessionContext` est déprécié, les listeners de bindingEvents servent à notifier les changement d'association d'un objet avec une session.

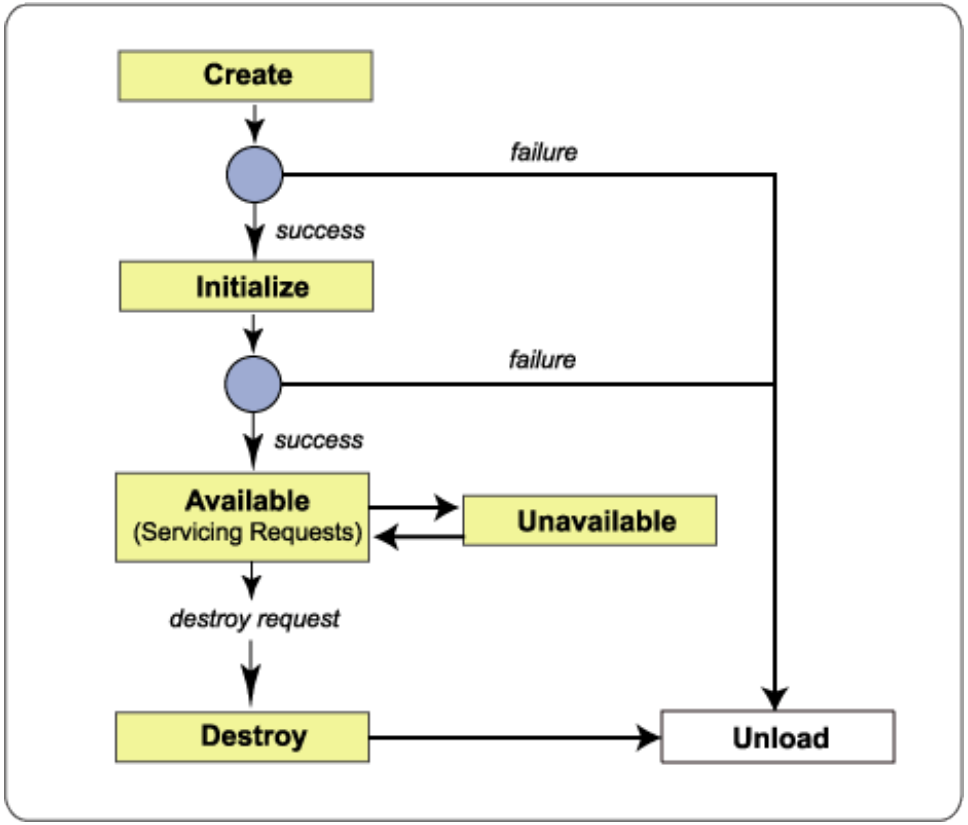


Cycle de vie des servlets



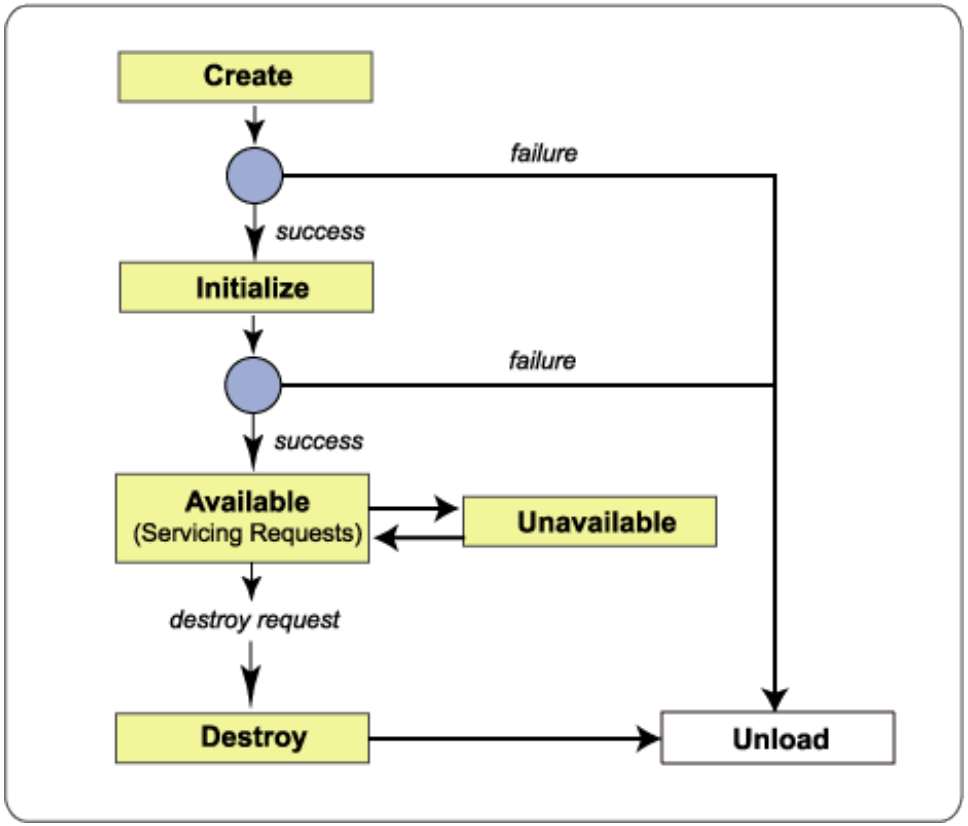
- Le conteneur instancie la servlet avec le constructeur par défaut puis appelle sa méthode `init()` qui l'initialise avant sa mise en service.

Cycle de vie des servlets



- Disponible : la servlet traite des requêtes dans des threads séparés. Le conteneur appelle la méthode `service()` qui appelle la méthode adéquat.

Cycle de vie des servlets



- Destroy : le conteneur décide de ne plus utiliser la servlet, il appelle la méthode `destroy()` avant de détruire la servlet.

Avantages/différences avec C.G.I.

- En C.G.I, chaque requête est traitée par un nouveau processus créé par le démon HTTP. La création est coûteuse et, pour des script courts, le serveur peut passer plus de temps à créer des processus qu'à les exécuter.
- Les servlets traitent les requêtes dans des threads distincts à l'intérieur du processus du serveur.
- Le serveur constitue souvent un *pool* de threads prêts avant l'arrivée d'un requête.
- L'instance unique de la servlet permet de mutaliser les ressources (difficile à partir de processus séparés).

Hello servlet

```
1  import java.io.*;
2  import javax.servlet.*;
3  import javax.servlet.http.*;
4
5  public class TestServlet extends HttpServlet {
6    public void doGet(HttpServletRequest request,
         HttpServletResponse response) throws ServletException,
          IOException {
7      response.setContentType("text/html");
8      PrintWriter out = response.getWriter();
9
10     String docType = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
           HTML 4.0 " + "Transitional//EN\">\n";
11
12     out.println(docType + "<HTML>\n" +
13     "<HEAD><TITLE>Servlet Test</TITLE></HEAD>\n" +
14     "<BODY BGCOLOR=\"#FDF5E6\">\n" +
15     "<H1>Servlet Test</H1>\n" +
16     "URL you used: " + request.getRequestURL() + "\n" +
17     "</BODY></HTML>");
18   }
19 }
```

Lecture de formulaire

```
1  import javax.servlet.*;
2  import javax.servlet.http.*;
3  import java.util.*;
4  public class UserInfo extends HttpServlet {
5    public void doPost(HttpServletRequest request,
6            HttpServletResponse response)
7      throws ServletException, IOException {
8      response.setContentType("text/html");
9      PrintWriter out = response.getWriter();
10     out.println("<HTML>\n<BODY>\n" +
11       "<H1>Recapitulatif␣des␣informations</H1>\n"
12     "<UL>\n" +
13     "␣␣<LI>Nom:␣"
14         + request.getParameter("Nom") + "\n" +
15         "␣␣<LI>Prenom:␣"
16         + request.getParameter("Prenom") + "\n" +
17         "␣␣<LI>Age:␣"
18         + request.getParameter("Age") + "\n" +
19         "</UL>\n" +
20     "</BODY></HTML>");
21   }
22 }
```

Exemple de session

```
1  import javax.servlet.*;
2  import javax.servlet.http.*;
3  import java.util.*;
4  public class Caddie extends HttpServlet {
5    public void doGet (HttpServletRequest request,
         HttpServletResponse response)
6      throws ServletException, IOException    {
7      // Recupere la session
8      HttpSession session = request.getSession(true);
9      // Recupere l'age de l'utilisateur
10     Age = (int)session.getAttribute("Age");
11
12     if (Age != null) {
13       // ... faire quelque chose
14       out = response.getWriter();
15     }
16     else {
17       Age = new Integer(...);
18       session.setAttribute("Age", Age);
19       // ... faire quelque chose d'autre
20       out = response.getWriter();
21     }
22   }
23 }
```

Variables d'application

La variable *compteur* est partagée par toutes les instances :

```
1  import javax.servlet.*;
2  import javax.servlet.http.*;
3  import java.util.*;
4
5  public class Counter extends HttpServlet {
6    public void doGet (HttpServletRequest request,
         HttpServletResponse response)
7      throws ServletException, IOException    {
8      // Recupere le context
9      ServletContext application=this.getServletContext();
10
11     Integer counter=(Integer)application.getAttribute("
           compteur");
12     if (counter==null) counter=new Integer(1);
13     else counter++;
14
15     application.setAttribute("compteur", counter);
16     ...
17   }
18 }
```

TODO XXX

- directives
- taglib
- ...

Les servlets

Définition (JavaServer Pages)

JavaServer Pages *est une technique de programmation qui permet d'inclure du code Java dans un texte HTML pour rendre son contenu dynamique.*

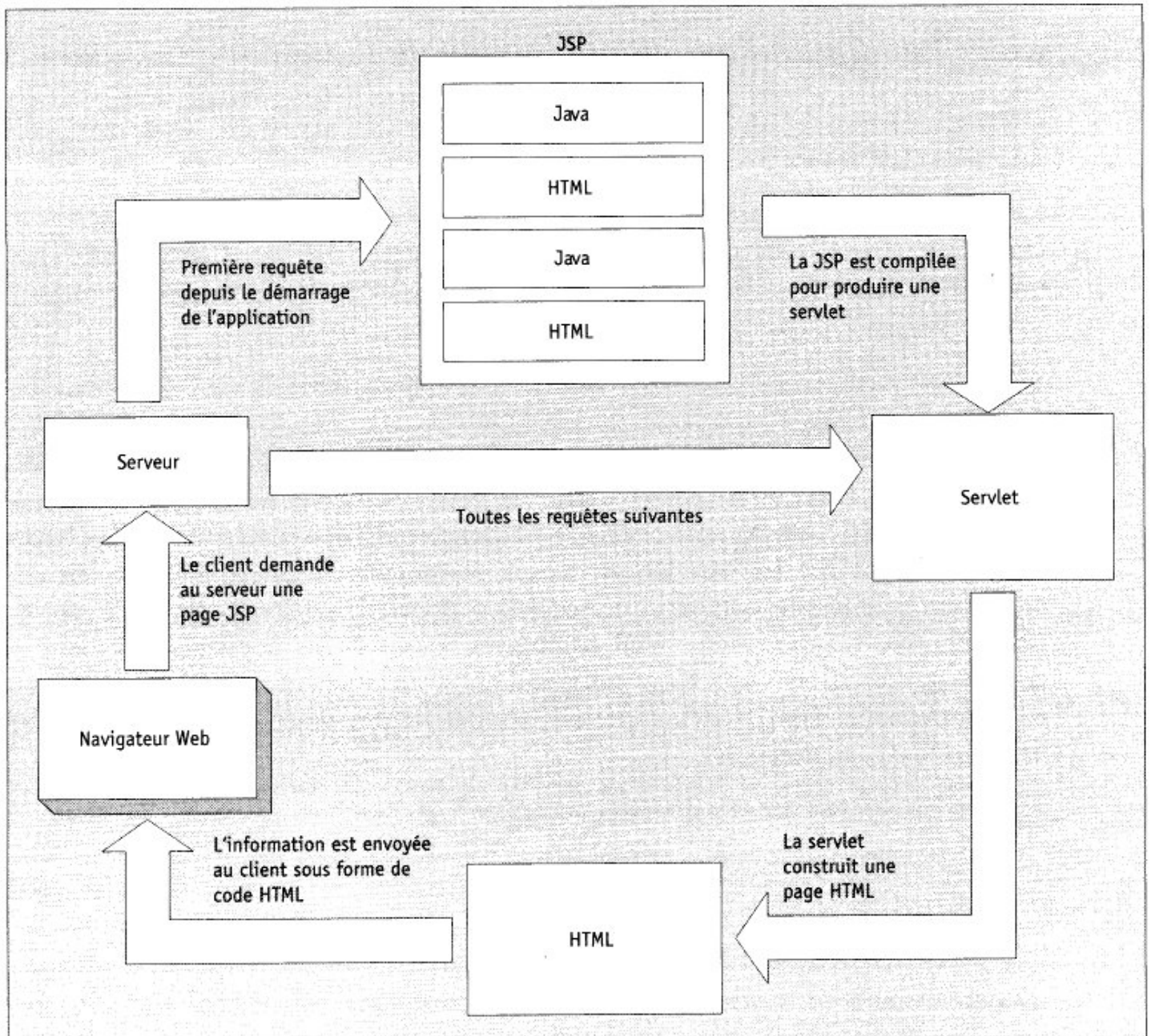
- Le code est extrait et utilisé pour générer automatiquement une classe dérivée `HttpServlet`,
- en fonction des directives, le code est ajouté comme attribut de la classe, inséré dans la méthode de traitemetn de la requête, etc ...
- Le code fait donc un usage aux objets implicites des servlets, ex : les objets `HttpServletRequest` et `HttpServletResponse`.

Un premier exemple

hello.jsp

```
1  <%@ page contentType="text/html; charset=utf-8" %>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "
       http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3  <html>
4    <head>
5      <title>Ma premiere page JSP</title>
6    </head>
7  <body>
8    <% String prenom=request.getParameter("prenom"); %>
9    <h1>Bonjour <%=(prenom!=null && prenom.length()!=0)?prenom
         :"bel(le) inconnu(e)"%></h1>
10   <% if (prenom!=null && prenom.equals("le monde")) { %>
11   <h2>Bien joue !!!!</h2>
12   <% } %>
13
14   <form action="hello.jsp" method="post">
15     <label>Prenom : </label><input type="text" name="prenom"
           size="30">
16     <input type="submit" value="envoyer">
17   </form>
18 </body>
19 </html>
```

Compilation à la volée



Un deuxième exemple

example.jsp

```
1  <%@ page errorPage="myerror.jsp" %>
2  <%@ page import="com.foo.bar" %>
3
4  <html>
5  <head>
6  <%! int serverInstanceVariable = 1;%>
7
8  <% int localStackBasedVariable = 1; %>
9  <table>
10 <tr><td><%= toStringOrBlank( "expanded inline data " + 1 )
       %></td></tr>
```

Résultat de la compilation :

```
1  package jsp_servlet;
2   import java.util.*;
3   import java.io.*;
4   import javax.servlet.*;
5   import javax.servlet.http.*;
6   import javax.servlet.jsp.*;
7   import javax.servlet.jsp.tagext.*;
8  ...
```

Un deuxième exemple

Servlet correspondant à example.jsp

```
1  import com.foo.bar; //  <%@ page import="com.foo.bar" %>
2  import ...
3
4  class _myservlet implements javax.servlet.Servlet, javax.
       servlet.jsp.HttpJspPage {
5     // result of <%! int serverInstanceVariable = 1;%>
6     int serverInstanceVariable = 1;
7     ...
8     public void _jspService( javax.servlet.http.
          HttpServletRequest request,
9     javax.servlet.http.HttpServletResponse response )
10    throws javax.servlet.ServletException,
11    java.io.IOException {
12        javax.servlet.ServletConfig config = ...; // Get the
              servlet config
13        Object page = this;
14        PageContext pageContext = ...;              // Get the
              page context for this request
15        javax.servlet.jsp.JspWriter out = pageContext.getOut
              ();
16        HttpSession session = request.getSession( true );
```

Un deuxième exemple

Servlet correspondant à example.jsp

```
1       ...
2       try {
3             out.print( "<html>\r\n" );
4             out.print( "<head>\r\n" );
5             ...
6             // From <% int localStackBasedVariable = 1; %>
7             int localStackBasedVariable = 1;
8             ...
9             out.print( "<table>\r\n" );
10            out.print( "␣<tr><td>" );
11            // From <%= toStringOrBlank( "expanded␣inline␣
                  data␣" + 1 ) %>
12            out.print( toStringOrBlank( "expanded␣inline␣
                  data␣" + 1 ) );
13            out.print( "␣</td></tr>\r\n" );
14            ...
15        } catch ( Exception _exception ) {
16            // Clean up and redirect to error page in <%@
                  page errorPage="myerror.jsp" %>
17        }
18    }
19 }
```

Les éléments JSP

Les directives permettent d'incorporer du code Java dans un flux statique HTML. En voici quelques unes :

- `<%@ ... %>` : directives globales à la page,
- `<%! ... %>` : déclarations (variables, méthodes),
- `<% ... %>` : scriptlet (= script : tests, itération, ...),
- `<%= ... %>` : récupération d'une valeur d'expression,
- `<jsp: include ... />` : inclusion à l'exécution,
- `<jsp:forward ... />` : délégation à un autre composant,
- `<jsp:useBean ... />` : utilisation d'un java bean,
- ...

Les directives

Placées entre les symboles `<%@` et `%>`

- `<%@ include file="unAutreFichier" %>` : inclut une autre JSP,
- `<%@ taglib prefix="myprefix" uri="taglib/my` indique qu'une bibliothèque de balises doit être utilisée. Un préfixe doit être utilisé (namespace) et l'URI de la description de la bibliothèque donné.

La directive *page*

import	Importe un paquetage (package) Java. Cette directive résulte en une instruction import dans la servlet.
contentType	Définit le type de contenu de la page générée. Par exemple, ce peut être text/html pour du HTML.
errorPage	Indique la page à afficher si une exception se produit pendant le traitement de la requête HTTP.
isErrorPage	Si cette valeur est à true, la page est une page d'erreur.
isThreadSafe	Indique si la servlet résultante est protégée pour les threads (fils d'exécution).

La directive *page*

Exemple avec errorPage

```
1 <%@ page contentType="text/html; charset=UTF-8" %>
2 <%@ page errorPage="erreur.html" %>
3 <!DOCTYPE html PUBLIC ...>
4 <html>
5   <head><title>Ma premiere page jsp</title></head>
6   <body><h1>1/0 = <%=1/0 %></h1></body>
7 </html>
```

Résultat :

```
1 <!DOCTYPE html PUBLIC ...>
2 <html>
3 <head>
4   <meta http-equiv="Content-Type" content="text/html;
        charset=UTF-8" />
5   <title>Page d'erreur</title>
6 </head>
7 <body><h1>Il doit y avoir une erreur dans la JSP...</h1></
    body>
8 </html>
```

Objets implicites

La servlet compilée déclare les objets suivants (entre autres) :

- http ://-java.sun.com/products/jsp/syntax/2.0/syntaxref20.html
- **request** : *httpServletRequest* : correspondant à la requête,
- **response** : *httpServletResponse* : correspondant à la réponse,
- **session** : permet de gérer une session,
- **out** : le flot de sortie de la réponse,
- **application** : Servlet application ; contient, entre autres, la méthode,
- **log()** : pour écrire dans le fichier log,
- **pageContext** : gestion du contexte et des attributs de la JSP,

Programmation MVC

Et le modèle MVC dans tout ça ?

Programmation MVC

- Les *JSP* produisent des servlets générant du code de présentation (HTML). Elles seront donc clairement réservée à la partie *vue*,
- Un contrôleur a pour but de recevoir les requêtes, il sera donc implémenté sous forme d'une servlet simple.
- La partie modèle sera implémentée de façon classique en se reposant sur les sessions pour l'état de l'application et sur JPA (par exemple) pour la persistance des données en base.

Question : comment connecter le contrôleur et les vues.

Relation contrôleur–vues

- Une approche simple consiste et encapsuler les vues dans des classes maisons et invoquer leurs méthodes produisant le code de présentation de la vue correspondante.
- *inconvénient :* on se prive de toutes les facilités de JSP, moins flexible.
- Autre approche : déléguer l'affichage de la vue à une JSP,
- *problème :* une JSP est une servlet, qui prend en paramètre un objet requête et un objet réponse.

Solution : le transfert de requête.

Un premier exemple

hello.jsp

```
1  import java.io.IOException;
2  import javax.servlet.*;
3  import javax.servlet.http.*;
4  public class servletToJsp extends HttpServlet {
5    public void doGet(HttpServletRequest request, HttpServletResponse response){
6      // Parametres a destination de la JSP :
7      request.setAttribute("servletName", "servletToJsp");
8
9      try {
10       getServletConfig().getServletContext().getRequestDispatcher(
11         "/JSP/Demo.hello.jsp").forward(request,response);
12
13     } catch (ServletException e) {
14       ...
15     }
16   }
17
18 }
```

Quelques références

- Référence servlet : http ://docs.oracle.com/javaee/6/api/javax/servlet//package-summary.html
- Référence JSP 2.0 : http ://-java.sun.com/products/jsp/syntax/2.0/syntaxref20.html