

Approfondissement JAVA : Applets, Servlets, JSP

Projet III
2000 / 2001
Alain Paoli

Plan

- Applet
- Servlet
 - Principe
 - Exemples
 - Cycle de vie
 - Session
 - API
 - Déploiement
- JSP
 - Principe
 - Syntaxe et Sémantique
 - Exemple

Applet

- **Application Java:**

- une entité indépendante qui s'exécute dans l'environnement de l'interpréteur (JVM)

- **Applet:**

- hérite du comportement de la classe **Applet** et de l'environnement du navigateur qui l'accueille
- gère des sollicitations et événements venant du clavier, de l'écran, de la souris, du réseau...
- Pour des raisons de sécurité l'applet ne peut pas :
 - lire/écrire des fichiers de la machine hôte
 - exécuter des programmes, charger des bibliothèques (shared library), créer (fork) des processus
 - communiquer avec d'autres machines, à part la machine origine de l'applet

Applet

- Dans la page HTML une balise <APPLET> est utilisée

```
<APPLET
CODEBASE = "." <!-- URL de MonApplet.class par rapport à cette page -->
CODE    = "exemple.MonApplet.class" <!-- nom de la classe -->
NAME    = "AppletTest"
WIDTH   = 400 <!-- largeur de la "fenêtre" d'applet dans la page -->
HEIGHT  = 300 <!-- hauteur de la "fenêtre" d'applet dans la page -->
ALIGN   = middle <!-- alignement de la "fenêtre" d'applet dans la page -->
<PARAM NAME="param1" VALUE="valeur1">
<!-- paramètre passé à l'applet et récupéré par la méthode getParameter() de l'Applet-->
```

- Lorsque cette balise est rencontrée par le navigateur, le programme Java présent sur le serveur est chargé dans le répertoire cache de l'ordinateur
- Ce code peut être présent localement sur le disque ou dans le répertoire de la page HTML.

Applet

- Le code peut se résumer à la surcharge de 4 méthodes traitant 4 événements:

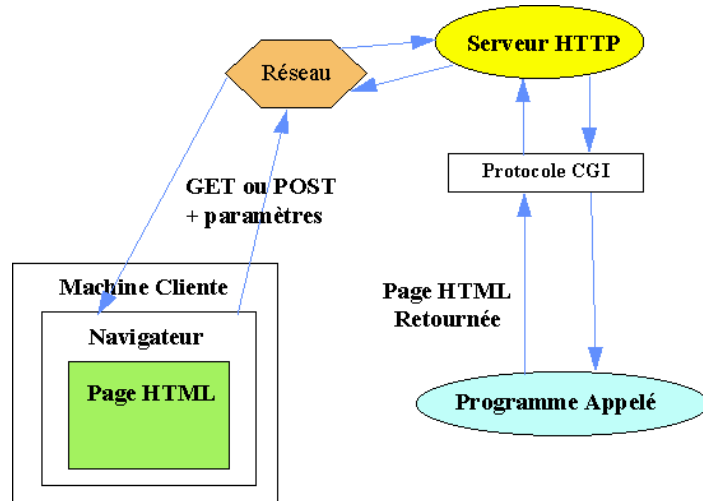
```
public class Simple extends java.applet.Applet {  
    ...  
    public void init() { }  
    public void start() { }  
    public void stop() { }  
    public void destroy() { }  
}
```

- Les méthodes `init()`, `start()`, `stop()`, `destroy()` sont appelées:
 - **init()**: au moment du (re-)chargement de l'applet
 - **start()**: après `init()`, démarre l'exécution de l'applet
 - **stop()**: quand on quitte le navigateur ou la page de l'applet (doit être surchargée si `start()` l'a été)
 - **destroy()**: nettoyage final

Servlets

Version 2.2
(version 2.3 au stade de
Proposed Final Draft)

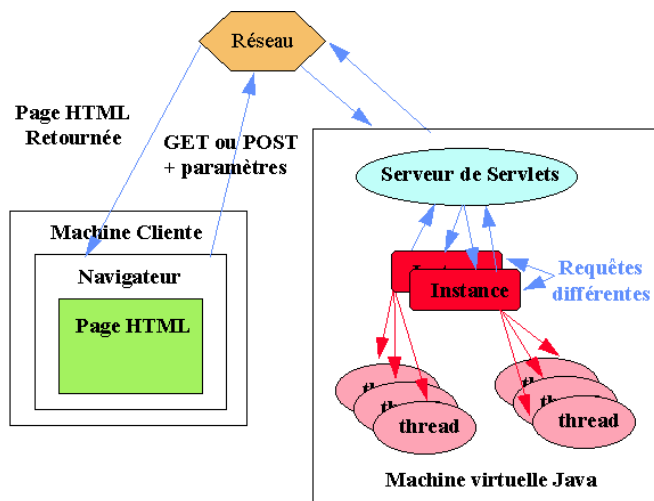
CGI



© Alain Paoli - 2000

7

Servlet



© Alain Paoli - 2000

8

Servlet - Principe

- Le programme Java spécifique qui reçoit les requêtes HTTP est un objet instance d'une classe munie d'une méthode main, et implémentant l'interface *javax.servlet.http.HttpServlet*.
- Les requêtes HTTP transmises spécifient le nom (ou classe du Servlet) appelé.
- Quand cet objet reçoit une requête HTTP:
 - S'il n'existe pas encore d'instance du Servlet correspondant à la requête HTTP, il en crée une.
 - Il appelle une méthode de cette instance, pour traiter la requête, à l'intérieur d'un nouveau processus léger (ou "thread") qui est créé pour l'occasion:
 - doGet (requête HTTP GET)
 - doPost (requête HTTP POST)

Servlet vs. CGI

- Performance
 - Processus:
 - Programme CGI: un nouveau processus est démarré pour chaque requête HTTP.
 - Servlet: chaque requête est traitée par un processus léger.
 - Requêtes simultanées pour la même URL:
 - Programme CGI: le code du programme est chargé N fois en mémoire.
 - Servlet: Le code de la classe n'est chargé qu'une fois à la création de l'instance unique du servlet.
 - Servlet: plus de possibilités d'optimisation (pool connexions, cache, ...)
- Puissance:
 - Partage d'information entre servlets.
 - Session utilisateur.
- Avantages liés à Java

Servlet - Idioms

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

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

        // Utiliser l'objet "request" pour lire les headers HTTP (e.x. cookies)
        // et les paramètres transmis par un formulaire HTML

        // Utiliser l'objet "response" pour écrire la ligne HTTP réponse avec ses headers
        // (e.x. spécifier le MIME type, passer des cookies).

        PrintWriter out = response.getWriter();
        // Utiliser l'objet "out" pour écrire la page réponse envoyée au navigateur
    }
}

```

© Alain Paoli - 2000

11

Servlet Exemple 1

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

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

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 " + "Transitional//EN">\n" +
            "<HTML>\n" +
            "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
            "<BODY>\n" +
            "<H1>Hello WWW</H1>\n" +
            "</BODY></HTML>");

    }
}

```

© Alain Paoli - 2000

12

Exemple - Formulaire

Formulaire Simple

Numéro Produit: 162
Quantité: 3
Prix unitaire: 12.5

Nom: Paoli
Prénom: Alain
Adresse: 252 rue St Martin - Paris

Carte de Crédit:
 Visa
 Master Card
 American Express

Numéro carte crédit: *****
OK

© Alain Paoli - Document : chargé

13

Réponse

Lecture des paramètres de la requête

Parameter Name	Parameter Value(s)
quantity	3
address	252 rue St Martin - Paris
itemNum	162
firstName	Paoli
cardType	Visa
price	12.5
lastName	Alain
cardNum	123456

© Alain Paoli - 2000 Document : chargé

14

Source HTML

```
<HTML>
<HEAD><TITLE>Formulaire Simple</TITLE></HEAD>
<BODY>
<H1 ALIGN="CENTER">Formulaire Simple</H1>
<FORM action="http://localhost:8080/servlet/exemple.ShowParameters" method="POST">
  Numéro Produit: <INPUT TYPE="TEXT" NAME="itemNum"><BR>
  Quantité: <INPUT TYPE="TEXT" NAME="quantity"><BR>
  Prix unitaire: <INPUT TYPE="TEXT" NAME="price" VALUE="F"><BR>
  <HR>
  Nom: <INPUT TYPE="TEXT" NAME="firstName"><BR>
  Prénom: <INPUT TYPE="TEXT" NAME="lastName"><BR>
  Adresse: <TEXTAREA NAME="address" ROWS=3 COLS=40></TEXTAREA><BR>
  Carte de Crédit:<BR>
  <INPUT TYPE="RADIO" NAME="cardType" VALUE="Visa">Visa<BR>
  <INPUT TYPE="RADIO" NAME="cardType" VALUE="Master Card">Master Card<BR>
  <INPUT TYPE="RADIO" NAME="cardType" VALUE="Amex">American Express<BR>
  Numéro carte crédit: <INPUT TYPE="PASSWORD" NAME="cardNum"><BR>
  <INPUT TYPE="SUBMIT" VALUE="OK">
</FORM>
</BODY>
</HTML>
```

© Alain Paoli - 2000

15

```
package exemple;
import javax.servlet.*; import javax.servlet.http.*; import java.io.*; import java.util.*;

public class ShowParameters extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>\n" + ..... "<TR>\n" + "<TH>Parameter Name<TH>Parameter Value(s)");
    Enumeration paramNames = request.getParameterNames(); // récupère les noms de paramètres
    while(paramNames.hasMoreElements()) {
      String paramName = (String) paramNames.nextElement(); //paramName contient un nom de paramètre
      out.println("<TR><TD>" + paramName + "\n<TD>");
      String[] paramValues = request.getParameterValues(paramName); // récupère la liste des valeurs de paramName
      if (paramValues.length == 1) { // si mono-valué
        String paramValue = paramValues[0];
        if (paramValue.length() == 0) // si pas de valeurs
          out.print("<I>No Value</I>");
        else out.print(paramValue);
      } else { // cas multi-valué
        out.println("<UL>");
        for(int i=0; i<paramValues.length; i++) { // parcourt la liste des valeurs de ce paramètre
          out.println("<LI>" + paramValues[i]);
        }
        out.println("</UL>");
      }
    }
    out.println("</TABLE>\n</BODY></HTML>");
  }
}
```

© Alain Paoli - 2000

16

Servlet - Cycle de Vie

- Initialisation
 - appel de la méthode *init()*. Peut être surchargée pour, par ex, se connecter à une BDR.
- Traitement requêtes:
 - appel de la méthode *service()* qui dispatche vers *doGet()* ou *doPost()* suivant le type de requête HTTP reçue.
- Destruction
 - appel de la méthode *destroy()*. Peut être surchargée pour, par ex, se déconnecter d'une BDR.

API - Session

- Session: par cookie temporaire ou paramètre requête
 - Un contexte est maintenu pour un utilisateur
 - `HttpSession session = request.getSession(true);`
`// récupère la session courante. En crée une si besoin est`
 - `session.setAttribute("o1", objet1);` // stocke un objet
 - `MaClasse o = session.getAttribute("o1");` // retrouve l'objet
 - `Enumeration enumAttNames = session.getAttributeNames();`
 - Méthodes:
 - `getCreationTime(): long` → Date de création de la session
 - `getId(): String` → Identifiant de la session
 - `getLastAccessedTime() : long` → Date dernière requête
 - `get/setMaxInactiveInterval(int):` nb de millisecondes d'inactivité
 - `isNew(): boolean` → vrai si la session vient d'être créée

API: requête / réponse

- **HttpServletRequest:**
 - `getProtocol`: HTTP/1.0
 - `getServerName`: company.com
 - `getMethod`: GET
 - `getPathInfo`: exemple.ShowParameters
 - `getPathTranslated`: c:\mesServlets\exemple\ShowParameters.html
 - `getQueryString`: les paramètres (avec leur valeurs) de la requête
 - `getServletPath`: /servlet/exemple/exemple.ShowParameters
- **HttpServletResponse:**
 - `addCookie(Cookie cookie)`: ajoute un cookie
 - `sendRedirect(String location)`: redirige la requête
 - `setContentType(type)`: spécifier MIME type réponse 'text/html'
 - `getWriter()`: java.io.PrintWriter

Descripteur de déploiement

- Le descripteur de déploiement définit tous les paramètres d'une *application Web*.
 - Une application Web est une collection de servlets, pages html, images, classes Java, etc.. qui peut être déployée dans des environnements différents. Une application Web est définie par une URL racine (ex: *www.mycorp.com/catalog*). Toutes les requêtes qui commencent par ce préfixe seront dirigée vers cette collection de servlet (ServletContext).
 - Le descripteur est défini en XML (voir spécification).
 - Répertoires: Le répertoire racine de l'application (ex: *catalog*) contient:
 - les pages JSP, images etc...
 - un répertoire appelé 'Web-inf' qui contient:
 - le descripteur de déploiement (fichier 'web.xml')
 - un répertoire appelé 'classes' contenant les packages Java applicatifs

Java Server Page

(JSP version 1.1)

Principe

- **But: Séparer les parties statiques des parties dynamiques d'une page**
 - pouvoir utiliser les outils standard pour la partie statique HTML
- **Inclusion de code Java dans pages HTML**
 - le code Java est inclus entre des tags '`<%`' ... '`%>`'
 - l'extension de la page est '.jsp'
 - le code HTML et le code Java sont compilés dans un servlet dont le source est généré dynamiquement

Syntaxe

- Quatre sortes de tags
 - Expression:
 - Ce sont des expressions Java retournant une String qui est placée dans la page réponse
 - Éléments de scripts (*Scriptlet*):
 - code Java inclut dans le servlet.
 - Déclarations
 - déclaration de variables ou de méthodes
 - Directives
 - contrôle la structure globale du servlet
 - Actions
 - déclaration des classes Java utilisées (import), contrôle du comportement du moteur JSP

Expression JSP

- Sert à inclure une valeur dans le flot de sortie (page réponse)
- Syntaxe:
 - `<%= expression Java %>`
- Exemple:
 - `<p>Aujourd'hui: <%= new java.util.Date() %> </p>`
- L'expression Java est évaluée, convertie en une string, et insérée dans la page. Cette évaluation est faite lors du traitement de la requête HTTP.
- **Attention:** Une expression ne comporte pas de ';' final

Variables pré-définies JSP

- Des variables pré-définies peuvent être utilisées dans les expressions et les scriptlets:
 - *request* (HttpServletRequest)
 - *response* (HttpServletResponse)
 - *out* (PrintWriter) utilisé pour écrire dans la page réponse
 - *session* (HttpSession) la session (si elle existe) associée à la requête
 - *application* (ServletContext) identique à *getServletConfig().getContext()*.
 - *config* (ServletConfig)
 - *page* (this)
 - *pageContext*. Accès aux classes JSP spécifiques (*javax.servlet.jsp*)
- Exemple:
 - `<p>Le nom de votre machine est: <%= request.getRemoteHost() %>.</p>`

Scriptlet JSP

- Permet d'insérer n'importe quel code Java
 - Syntaxe: `<% code Java %>`
- Exemple: *Les variables déclarées sont locales.* *Attention: les instructions se terminent normalement par un ';'.*

```
<%  
String queryData = request.getQueryString();  
out.println("Attached GET data: " + queryData);  
%>
```
- Le code d'un scriptlet est inséré *tel quel* dans le corps de la méthode *service()* du servlet généré, et n'importe quel code HTML statique avant ou après un scriptlet est converti en instructions d'écriture de la page réponse. En conséquence, les scriptlet peuvent constituer blocks 'incomplets', ces blocks incluant du code HTML statique en dehors du scriptlet.

Scriptlet JSP

- Exemple:

```
<% if (Math.random() < 0.5) { %>
Vous avez <B>gagné</B>!
<% } else { %>
Vous avez <B>perdu</B>!
<% } %>
```

- Sera compilé en quelque chose comme:

```
if (Math.random() < 0.5) {
out.println("Vous avez <B>gagné</B>!");
} else {
out.println(" Vous avez <B>perdu</B>!");
}
```

Déclaration JSP

- Permet de définir des méthodes ou des attributs au niveau du servlet (en dehors de la méthode *service()*).
 - La valeur d'un attribut sera partagée par tous les threads appliqués à la méthode *service()*, et accessible de n'importe quel scriptlet de la page JSP
 - La méthode sera accessible de n'importe quel scriptlet de la page JSP.

- Syntaxe:

– `<%! Code Java %>`

- Exemples:

```
<%! private int accessCount = 0; %> <!-- déclaration d'attribut -->
<p>Nombre d'accès à cette page : <%= ++accessCount %></p> <!-- utilisation -->

<%! public String hello() { return "hello"; } %> <!-- déclaration de méthode -->
```

Directive JSP

- Une directive affecte la structure du servlet.
- Syntaxe:
 - `<%@ directive attribut="valeur" %>`
- Les deux principales directives sont *page* et *include*:
 - **page**: import de classes, paramétrisation du servlet généré
 - **include**: insertion d'un fichier lors de la traduction de la page JSP en servlet.
 - **taglib**: déclaration d'un tag défini par le programmeur (non abordé dans cette présentation)

Attributs de la Directive Page

- **import**="*package.class*" ou **import**="*package.class1* ,..., *package.classN*". Exemple:
`<% @ page import="java.util.*" %>`
- **contentType**="*MIME-Type*" ou **contentType**="*MIME-Type*; charset=*Character-Set*". Spécifie le type MIME de la sortie ("text/html" par défaut). Exemple: `<% @ page contentType="text/plain" %>` a le même effet que le scriptlet:
`<% response.setContentType("text/plain"); %>`
- **session**="true|false". Une valeur 'true' (par défaut) spécifie que la variable prédéfinie session (HttpSession) sera liée à la session courante si elle existe, et sinon qu'une nouvelle session sera créée. Une valeur 'false' spécifie qu'aucune session sera utilisée, et qu'un accès à la variable session se traduira par une erreur.
- **info**="*message*". Définit la chaîne retournée par la méthode `getServletInfo()`.
- **errorPage**="*url*". Spécifie la page JSP qui sera retournée si n'importe quelle exception, non récupérée dans la page, est levée.
- **isErrorPage**="true|false". Indique si cette page est une page d'erreur pour une autre page JSP. Par défaut vaut 'false'.

Directive Include

- **Syntaxe:**
 - `<%@ include file="relative url" %>`
- L'URL est interprétée relativement à la page contenant cette directive.
- Exemple

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<BODY>
<%@ include file="/navbar.html" %> <!-- inclut une barre de navigation -->
<!-- Partie spécifique de la page ... -->
</BODY>
</HTML>
```
- Le fichier `'/navbar.html'` est inclus au moment de la compilation du servlet.

Actions

- Utilise une syntaxe XML pour modifier dynamiquement le comportement du moteur JSP.
- *jsp:include* - Inclut un fichier lors de l'exécution de la requête.
- *jsp:useBean* - Retrouve ou instancie un Bean.
- *jsp:setProperty* - Positionne les propriétés d'un Bean.
- *jsp:getProperty* - Insère la valeur d'une propriété d'un Bean dans la sortie.
- *jsp:forward* - Dirige la requête vers une autre page.
- *jsp:plugin* - Génère du code spécifique pour le navigateur permettant d'insérer un applet (tag OBJECT) ou le plug-in de la JVM (tag EMBED).

Action *jsp:include*

- Syntaxe:

- `<jsp:include page="relative URL" flush="true" />`

- Cette action insère le fichier quand la requête est traitée.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
```

```
<HTML>
```

```
<BODY>
```

```
<CENTER>Quoi de neuf ? </CENTER> <P>Sommaire des nouvelles les plus récentes:
```

```
<OL>
```

```
<LI> <jsp:include page="news/Item1.html" flush="true"/>
```

```
<LI> <jsp:include page="news/Item2.html" flush="true"/>
```

```
<LI> <jsp:include page="news/Item3.html" flush="true"/>
```

```
<LI> <jsp:include page="news/Item4.html" flush="true"/>
```

```
</OL>
```

```
</BODY>
```

```
</HTML>
```

© Alain Paoli - 2000

33

Include

- Directive

- Une directive *include* traite le fichier inclut comme un objet statique: le résultat de la compilation du fichier inclut (qui peut être une page JSP ou HTML) est inséré dans le résultat de la compilation de la page JSP qui le contient.

- Action

- Une action *include* traite le fichier comme un objet dynamique: la requête reçue par la page qui le contient est transmise au fichier inclut, qui la traite, et le résultat est inclut dans la page résultat (mécanisme similaire à celui de l'appel d'un sous programme).
 - La valeur du paramètre '*page*' peut être calculée dynamiquement.
 - Une action *include* peut être insérée dans un *scriptlet* (conditionnée par un *if*, par exemple).

© Alain Paoli - 2000

34

Action forward

- Syntaxe:
 - `<jsp:forward page="relative URL" />`
- Cette action termine l'exécution de la requête pour la page courante, le buffer de sortie est vidé, et la requête est transmise à la page dont l'URL est spécifié par le paramètre 'page'.
 - La valeur du paramètre 'page' peut être calculée dynamiquement.
 - Une action *include* peut être insérée dans un *scriptlet* (conditionnée par un *if*, par exemple).
- Exemple:

```
<% String whereTo = "/templates/"+someValue; %>
<jsp:forward page='<%= whereTo %>' />
```

jsp:useBean

- Syntaxe:
 - `<jsp:useBean id="user" class="appli.User" scope="session"/>`
- Cette action associe un Bean, instance de la classe spécifiée par 'class', avec une portée spécifiée par 'scope', avec la référence spécifiée par 'id'.
- Si une instance associée avec la même référence n'est pas trouvée dans la portée, une nouvelle instance est créée.
- Portée: quatre valeurs possibles
 - *application*: bean visible de toutes les pages, pour toutes les sessions
 - *session*: bean visible de toutes les pages pour une session considérée
 - *request*: bean visible de toutes les pages participant à une requête
 - *page*: bean visible dans la page considérée

Jsp:setProperty

- Syntaxe: deux variantes
 - (A) `<jsp:setProperty name="user" property="*" />`
 - (B) `<jsp:setProperty name="user" property="logon" param="nomUser" />`
- Les deux variantes positionnent une ou des valeurs de propriétés du Bean dont la référence est spécifiée par '*name*', en utilisant l'inspection:
 - **Variante (A):** Positionne toutes les propriétés du Bean avec les valeurs passées en paramètres dans la requête, de telle façon qu'un paramètre de nom '*prop*' positionnera la valeur d'une propriété de nom '*property*' (i.e. ce Bean est muni d'un '*setter*' *setProp()*), en faisant au préalable des *conversions implicites*.
 - **Variante (B):** Positionne une propriété du Bean avec la ou les valeurs passées dans la requête pour un paramètre de telle façon que le paramètre de nom '*param*' positionnera la valeur de la propriété de nom '*property*' (i.e. ce Bean est muni d'un '*setter*' *setProp()*), en faisant au préalable des *conversions implicites*.

Accesseurs d'un Bean

- Propriété mono-valuée autre que *boolean*:
 - **public** *type* *getPropname()*
 - **public void** *setPropname(type x)*
- Propriété de type *boolean*:
 - **public boolean** *isPropname()*
 - **public void** *setPropname(boolean x)*
- Propriété multi-valuée
 - **public type[]** *getPropname()*
 - **public void** *setPropname(type[] x)*
- Dans le cas d'un paramètre multi-valué dans la requête, le Bean doit être muni d'accesseurs pour une propriété multi-valuée.

Conversions implicites

- Toutes les valeurs des paramètres insérés dans la requête sont vues comme des String dans la page JSP compilée, et converties vers le type de la propriété cible du Bean, suivant les règles indiquées ci-dessous.

Type Propriété du Bean	Conversion
boolean ou Boolean	java.lang.Boolean.valueOf(String)
byte ou Byte	java.lang.Byte.valueOf(String)
char ou Character	java.lang.Character.valueOf(String)
double ou Double	java.lang.Double.valueOf(String)
int ou Integer	java.lang.Integer.valueOf(String)
float ou Float	java.lang.Float.valueOf(String)
long ou Long	java.lang.Long.valueOf(String)

Exemple

- Gérer un formulaire d'identification
 - L'utilisateur doit saisir son logon et son mot de passe, puis cliquer sur 'OK'
 - Si pas de logon ou de mot de passe saisi: message d'erreur affiché dans la page réponse.
 - Si utilisateur inconnu (logon ou mot de passe non enregistré): message d'erreur affiché dans la page réponse.
 - Si erreur, on affiche à nouveau le formulaire avec valeurs saisies et message d'erreur
 - Sinon, on affiche la page d'accueil.
 - Architecture
 - Une page JSP affichant le formulaire + message d'erreur
 - Un Bean User + Bean Controleur du dialogue

Bean User

```
package demo.security;

public class User {
    private String logon = "";
    private String pass = "";

    public User() { } // constructeur
    public String getLogon() { return logon; }
    public String getPass() { return pass; }
    public void setLogon(String logon) { this.logon = logon; }
    public void setPass(String pass) { this.pass = pass; }
    public boolean isRegistered() {
        // si (logon != null) et (pass != null) et (logon et pass correspondent à un
        // utilisateur enregistré), alors retourner 'true' sinon retourner 'false'
    }
}
```

Page JSP

```
<jsp:useBean id="user" class="demo.security.User" scope="session"/>
<jsp:setProperty name="user" property="*" />
<jsp:useBean id="controler" class="demo.IdentificationControler" scope="page"/>
<% controler.process(request, response); %> <!-- appel du contrôleur - ->
<html>
<head><LINK REL=STYLESHEET TYPE="text/css" HREF="dt.css"></head>
<body>
<form method="post" action="identification.jsp?state=control">
<p>Votre email
    <input type="text" name="logon" size="40" value="<%= user.getLogon() %>">
<p>Mot de passe
    <input type="password" name="pass" maxlength="10" size="10" value="<%= user.getPass()
    %>">
<p><input type="submit" name="submit" value="Entrée">
</form>
<p><%= controler.getErrorMessage() %>
</body>
</html>
```

```

package demo;
import demo.security.*; import javax.servlet.http.*;
public class IdentificationController {
    private String errorMessage = "";
    public IdentificationController() { }
    public void process(HttpServletRequest request, HttpServletResponse response) {
        try {
            if (request.getParameter("state") == null) { // état: initialisation
                return;
            }
            String logon = request.getParameter("logon"); String pass = request.getParameter("pass");
            if (logon.equals("") || pass.equals("")) {
                errorMessage = "<b>Erreur: vous n'avez pas saisi votre identifiant ou votre mot de passe</b>";
                return;
            }
            User user = (User) request.getSession().getAttribute("user");
            if (!user.isRegistered()) {
                errorMessage = "<b>Erreur: vous n'êtes pas enregistré comme utilisateur</b>";
                return;
            }
            response.sendRedirect("accueil.jsp"); //sendRedirect() doit être fait avant écriture du tag <html>
        } // end try
        catch(IOException e) { errorMessage = "<b>Erreur: Fichier " + nextJSP + " introuvable.</b>"; }
    }
}

```

© Alain Paoli - 2000

43

Exécution

- (1) Affichage du formulaire pour la première fois / session:
 - Création du bean *User* (logon et pass initialisé à "")
 - Création d'une instance du contrôleur (sa portée est "page")
 - attribut *errorMessage* initialisé à ""
 - sa méthode *process* ne fait rien.
- (2) Saisie du logon et du pass et envoi
 - Récupération du bean *User* (logon et pass contiennent les valeurs saisies)
 - Création d'une instance du contrôleur (sa portée est "page")
 - attribut *errorMessage* initialisé à ""
 - il y a un paramètre "state" avec la valeur "control" : la méthode *process* vérifie les valeurs saisie
 - Si OK: redirige la requête vers "accueil.jsp"
 - Sinon positionne un message d'erreur dans *errorMessage* . Le message sera affiché par l'exécution du scriptlet : `<%= controleur.getErrorMessage() %>`

© Alain Paoli - 2000

44

Bibliographie

- Cours
 - <http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/> (Servlet + JSP)
 - <http://java.sun.com/products/jsp/docs.html> (débutant JSP)
 - <http://java.sun.com/products/jsp/taglibraries.html>
- Ressources:
 - <http://www.servlets.com/>
 - <http://www.interpasnet.com/JSS/>
 - <http://jakarta.apache.org/>
- Spécifications JSP 1.1 et 1.2 (Proposed final Draft)
 - <http://java.sun.com/products/jsp/>
- Implémentations
 - <http://jakarta.apache.org/builds/tomcat/release/v3.1/bin/jakarta-tomcat.zip>