



J2EE

Java Server Pages

J2EE: JSP et Java beans

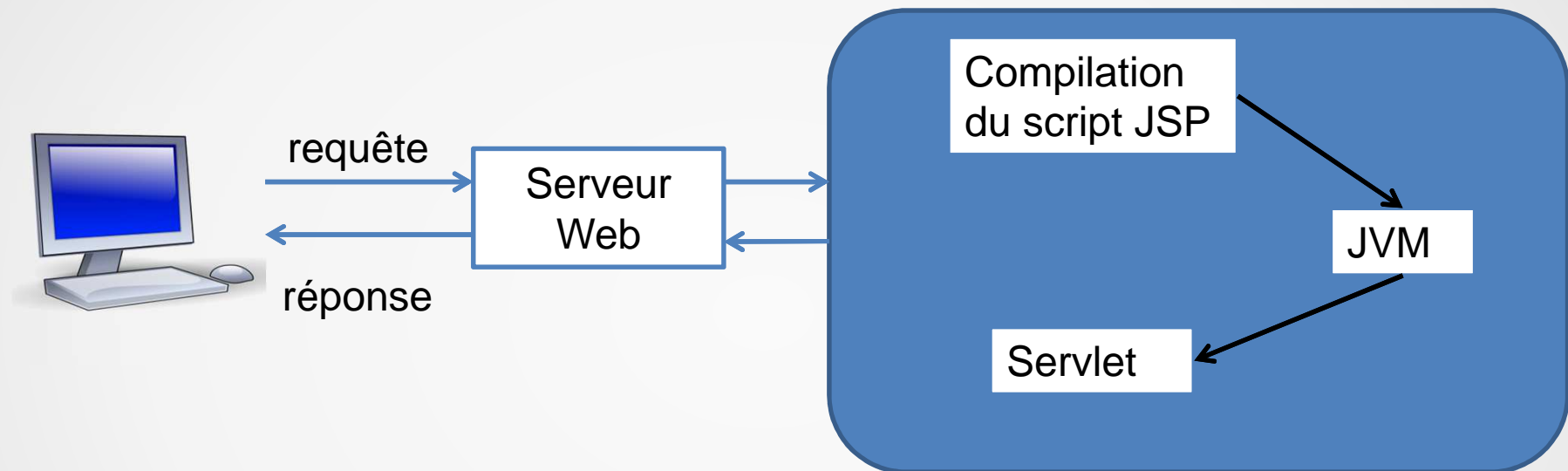
Java Server Pages

- Introduction
- Concepts fondamentaux
- Délégation et inclusion
- Partage de données
- Exemple
- Cycle de vie de la servlet générée

Introduction aux JSP

- Le langage JSP est un langage de scripts
- Les scriptlets JSP sont des instructions Java embarquées dans du code HTML, SVG, WML, XML entre les balises `<% %>`
- Chaque scriptlet est interprété par un moteur JSP qui différencie le code JSP du code HTML grâce à des balises spécifiques
- Le moteur JSP crée, compile, charge et exécute une servlet. C'est la servlet générée qui produit le code HTML,... de la page envoyée au client
- L'utilisation des JSP n'est pas limitée à la génération de code HTML. Le contenu généré peut être du code XML ou XHTML

Traduction puis compilation



- le script JSP est compilé en une servlet
- la servlet est compilée, chargée puis exécutée

Génération HTML

```
<HTML><BODY>
<H1>Affichage des prix HT et TTC</H1>
<% String param = request.getParameter("PHT");
    float prixHT = Float.parseFloat(param);
    float prixTTC= prixHT*(1+0.196F);
    out.print("prix HT =" + prixHT + "<BR>" );
    out.print("prix TTC =" + prixTTC + "<BR>" );
%>
</BODY> </HTML>
```

résultat = HTML
généré via
l'objet prédéfini
out du code
Java

fichier HTML envoyé coté client

fichier HTML
envoyé coté
client

```
<HTML><BODY>
<H1> Affichage des prix HT et TTC </H1>
prixHT = 42,50<BR>
prixTTC = 50,83<BR>
</BODY> </HTML>
```


Mécanismes mis en œuvre

Plusieurs zones `<% ... %>` peuvent cohabiter dans une même JSP

- Au premier chargement d'une JSP (ou après modification), le moteur JSP
 - rassemble tous les fragments `<% ... %>` de la JSP dans une classe
 - la compile en une servlet
 - la servlet est compilée et instanciée

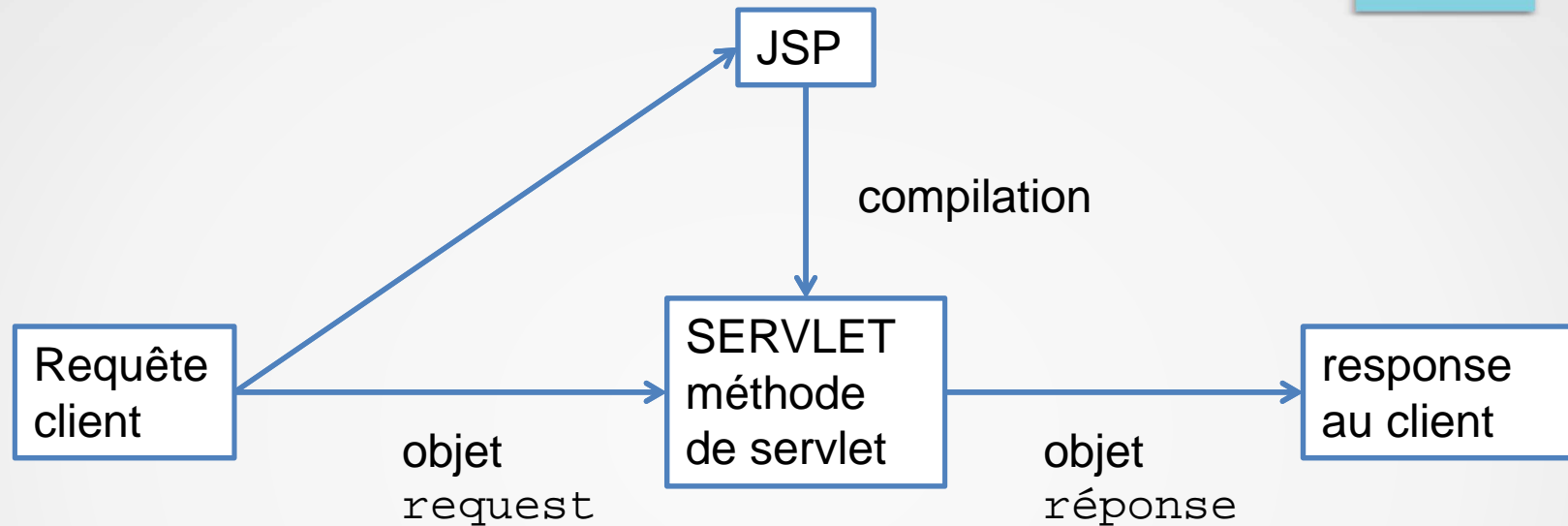
- Puis, ou lors des chargements suivants, le moteur JSP
 - exécute le code de la servlet dans un thread

=> délai d'attente lors de la 1ère invocation dû à la compilation



JSP = objet Java
présent dans le
moteur

Mécanismes mis en œuvre



Les objets `request` et `response` sont implicites. Pas de déclaration nécessaire dans la JSP

Balises de commentaires

2 manières de placer des commentaires dans une page JSP

```
<!-- mon commentaire -->
```

dans ce cas les commentaires sont transmis à la réponse. Ils sont donc visibles lorsqu'on visualise le code source de la page HTML.

ou

```
<%-- mon commentaire --%>
```

Le serveur JSP ne les prend pas en compte. Leur intérêt est uniquement de commenter le code JSP

4 types de balises

Les balises de directives spécifient les paramètres applicables à la page

`<%@ ... %>`

Les balises permettant de définir des variables globales à la page

`<%! ... %>`

Les balises de scriptlets permettant d'inclure du code java

`<% ... %>`

Les balises permettant d'évaluer une expression et l'affichage de sa valeur dans la page

`<%= ... %>`

Balises de scriptlets

Un scriptlet est une suite d'instructions Java contenues dans la balise `<% %>` et exécutées dès qu'un client invoque la page.

Les variables déclarées dans un scriptlet ont une portée locale et ne sont pas partagées;

Contrairement aux variables déclarées dans la balise `<%! %>` qui persistent d'une requête à l'autre

Exemple :

```
<% float[] pttc = {23.4,78.9,32.0};  
for(int i=0;i<pttc.length;i++){  
    out.println("<h2>Prix TTC</h2>" );  
    out.println(pttc[i]);  
}  
%>
```

Balises de déclarations

Les variables et méthodes qui deviendront membres de la servlet générée par le compilateur JSP sont déclarées par la balise :

`<%! déclarations %>`

Exemples :

```
<%! private int nbLettres(String nom) {  
    return nom.length();  
}  
int nb = 0;  
%>
```

Méthode d'instance
attachée à l'objet
correspondant à la JSP

variable d'instance initialisée à
l'instanciation de la JSP
- persiste entre 2 invocations
tant que la JSP ne change pas

Exemple

Note : une seule instance de JSP peut s'exécuter dans des threads différents pour des clients différents
=> prévoir un accès `synchronized`.

Attention !!

`<%! int cpt = 0; %>` différent de `<% int cpt = 0; %>`

variable **locale** à la JSP
(réinitialisée à chaque invocation de la JSP)

variable d'**instance** de la JSP
(**persiste**)

Balises d'expression

Une balise d'expression contient une expression Java

`<%= %>`

Exemple 1:

```
<input name="Nom" value= "<%= nom %>" type="text">
```

Cette expression est évaluée à l'exécution, sa valeur est envoyée à la page HTML réponse.

Concrètement, l'utilisateur saisit un nom qui sera un paramètre de la requête. La valeur sera null si aucune valeur n'est fournie

Exemple 2:

```
<%Date date = new java.util.Date();  
out.println( "date du jour : " + date );  
%>
```

est équivalent à

date du jour : `<%= new java.util.Date()%>`

la valeur retournée est
une String

Balises de directives

3 sortes de balises qui spécifient des directives de génération de page au compilateur :

- **page** permet de définir un certain nombre d'attributs qui s'appliqueront à la page
- **include** permet d'inclure un fichier texte ou autre ressource à la création de la servlet
- **taglib** permet de définir une bibliothèque de balises à utiliser

La directive page

Attributs qui s'appliqueront à la page

- l'attribut **extends** définit la classe parente de la servlet
`<%@page extends="unpackage.SuperClasse"%>`
- **import** définit les paquetages à importer
`<%@page import="java.util.Date"%>`
date du jour : `<%= new Date()%>`
- **langage** définit le langage de script utilisé dans la page
`<%@page langage="java"%>`
- **contentType** définit le type de contenu de la page générée
`<%@page contentType="text/plain" %>`
`<%@page contentType="text/html" %>`
- **errorPage** spécifie la page à afficher en cas d'erreur
`<%@page errorpage="500.jsp" %>`
isErrorPage vaut true si la page est une page d'erreur, false sinon
`<%@page isErrorPage="false" %>`

Directive page, attribut errorPage

Dans la JSP provoquant l'erreur :
on peut récupérer une exception déclenchée dans une scriptlet JSP.
son traitement pourra être assuré par une page d'erreur
la page d'erreur est spécifiée par la balise `page` et son attribut `errorPage`


```
<%@ page errorPage="erreur.jsp" %>
```

ou

```
<%@ page errorPage="erreur.jsp?code=xxx" %>
```

NOTE :

Une seule page d'erreur par JSP



transmission
d'information à la
page d'erreur

Directive page, attribut isErrorPage

Dans la page de traitement de l'erreur :

- on définit la page JSP comme une page d'erreur par

```
<%@ page isErrorPage=true %>
```

false indique une
page normale

- l'objet accompagnant l'exception est référencé par la variable implicite `exception`
On peut donc recueillir des informations sur l'erreur par les méthodes classiques de la classe `Exception`:

```
exception.getMessage()  
exception.printStackTrace()
```

Directive page, attribut errorPage: exemple (1/2)

```
<%@ page language="java" contentType="text/html" %>
<%@ page errorPage="/erreur.jsp" %>
<html><head><title>Page avec une erreur</title></head>
<body>
<% int var=90; var = var/0; %>
Division par 0 = <%= var %>
</body></html>
```

```
<%@ page language="java" contentType="text/html" %>
<%@ page isErrorPage="true" %>
<html><head><title>Page de gestion de l'erreur</title></head>
<body><h2><%=exception.getClass().getName()%><br>
l'exception déclenchée est : <%= exception.getMessage()%>
</body></html>
```

erreur.jsp

objet implicite

division.jsp

Directive page, attribut errorPage: exemple (2/2)



La directive include (1/3)

- **include** permet d'inclure dans la page un fichier texte ou autre ressource à la création de la servlet


```
<%@include  
file="/chemin/relatif/auContexte"%>
```

Recherche
relativement à
ServletContext



```
<%@include  
file="dansLe/repertoire/deLaPage"%>
```

Recherche dans le
même répertoire que la
JSP courante



- En général, le contenu du fichier est un fragment de page. Les variables créées par le fichier inclus sont dans la portée de la page

La directive `include` (2/3)

- Cette inclusion se fait au moment de la conversion, pendant la création de la servlet correspondante.
- Le contenu du fichier externe est inclus comme s'il était saisi directement dans la page JSP
- Les ressources à inclure doivent être contenues dans le contexte de l'application web

La directive include (3/3)

entete.html

```
<HTML><HEAD>  
<TITLE>Page de titre</TITLE>  
</HEAD><BODY>
```

corps.jsp

```
<%! String nom; %>  
<% nom="Meyer"; %>
```

```
<%@ include file="/entete.html" %>  
<%@ include file="/corps.jsp" %>  
Bonjour <%=nom %>  
<%@ include file="/piedpage.html" %>
```

piedpage.html

```
Ceci est le pied de page.  
</BODY>  
</HTML>
```

nom : variable déclarée dans le
fichier inclus corps.jsp

Les variables implicites (1/2)

- Un certain nombre d'objets sont toujours accessibles et utilisables à l'intérieur des pages JSP.

package javax.servlet

ServletRequest.**request** représente la requête utilisateur

ServletRequest.**response** représente la réponse fournie

ServletContext.**application** espace de données partagé entre toutes les servlets et JSP de la même application

ServletConfig **config** contient les paramètres d'initialisation de la servlet

package javax.servlet.jsp

jspWriter.**out** représente le flot de sortie pour la réponse

PageContext.**pageContext** contient les attributs de la page

package javax.servlet.http

HttpSession.**session** contient les attributs attachés à la session

Les variables implicites (2/2)

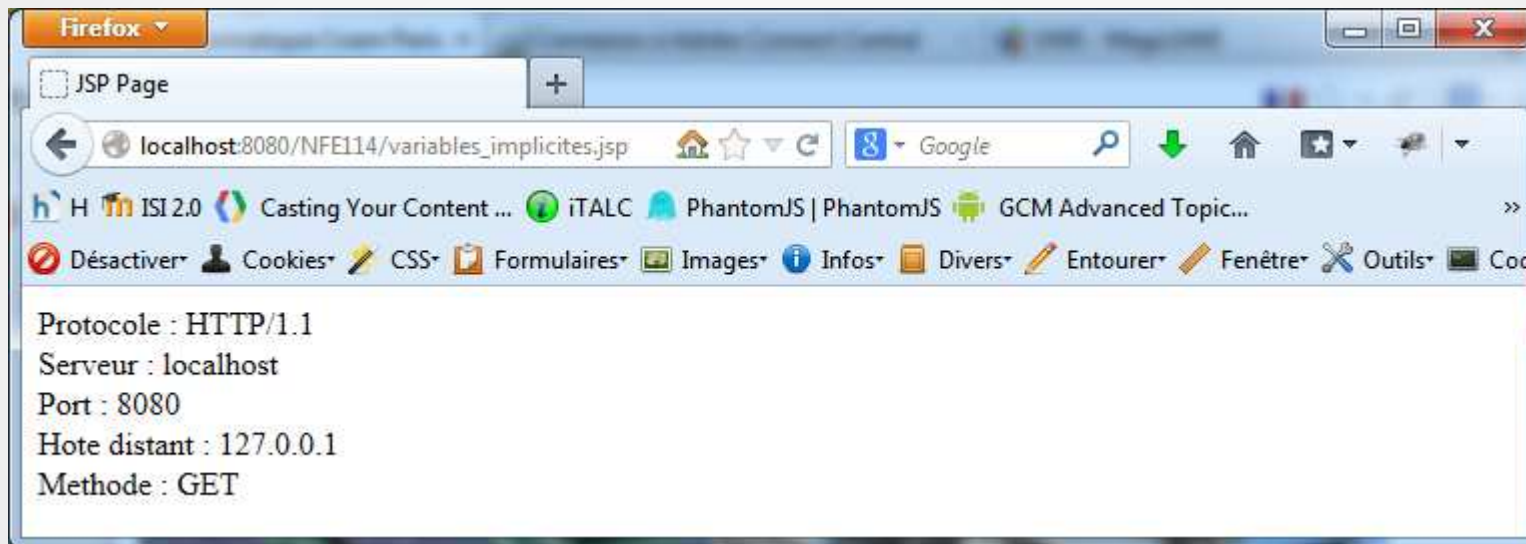
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page language="java" %>
<html><head><title>Informations du client</title></head>
<body bgcolor="white">
```

out et request:
variables implicites

```
Protocole : <%= request.getProtocol() %><br>
Serveur : <%= request.getServerName() %><br>
Port : <% out.println(request.getServerPort()); %><br>
Hote distant : <% out.println(request.getRemoteHost()); %><br>
Methode : <%= request.getMethod() %><br>
```

```
</body></html>
```


Les variables implicites (2/2)



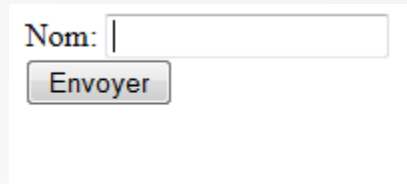
Cycle de vie de la servlet générée

Exemple : question.jsp

```
<%  
String nom = request.getParameter("Nom");  
if(nom==null) nom=" ";  
%>  
  
<form action="" method="post">  
Nom: <input name="Nom" value="<%= nom %>" type="text"><br/>  
<input type="submit" value="Envoyer"><br/>  
<b><%= nom %></b>  
</form>
```

Cycle de vie de la servlet générée

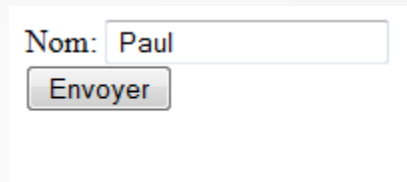
après chargement de la page



Nom:

Envoyer

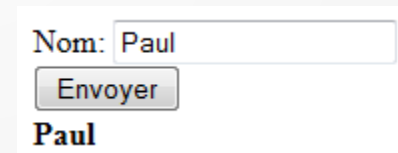
après saisie du nom



Nom:

Envoyer

après exécution de la servlet



Nom:

Envoyer

Paul

Cycle de vie de la servlet générée

Commentaires:

- La page JSP précédente est traduite en une servlet qui se charge de l'affichage en 1
- L'utilisateur saisit un nom dans le champ de texte en 2
- Il appuie sur le bouton envoyer qui transmet la requête avec comme paramètre le nom saisi. C'est une requête de type POST qui donne à la même servlet la responsabilité du traitement de la requête
`<form action=" " method="post">`
- La servlet `question_jsp.class` doit donc s'exécuter avec comme paramètre issu de la requête le nom Paul
- elle affiche de nouveau la page en remplaçant l'expression `<%= nom %>` par sa valeur

Cycle de vie de la servlet générée (1/2)

- Traduction du script JSP en servlet Java, le fichier `question.jsp` devient `question_jsp.java`
A ce niveau, les erreurs de syntaxe JSP sont détectées
- Compilation de la servlet `question_jsp.java` devient `question_jsp.class`
A ce niveau, les erreurs de syntaxe Java sont détectées
- Création et chargement d'un objet servlet
 - construction d'une instance de servlet
 - chargement en mémoire de cet objet

A chaque requête utilisateur, un thread est créé pour cette instance

Les variables d'instance sont accessibles et modifiables par tout utilisateur et pour toute requête

Pour des raisons de sécurité, il est important de créer des variables propres à chaque session pour éviter le partage des variables d'instance entre plusieurs utilisateurs

Cycle de vie de la servlet générée (2/2)

- Initialisation de la servlet
 - la méthode `_jspInit()` est exécutée une seule fois
 - elle a pour rôle d'initialiser les variables et de charger les valeurs d'environnement (`ServletConfig`)
- Appel à la méthode `_jspService()`
 - à chaque requête, un thread est créé qui fait appel à la méthode `_jspService()`
 - cette méthode détermine le type de requête (HTTP, ...) et le mode de transmission (GET ou POST)
 - les objets `request` et `response` sont créés
- Destruction de la servlet
 - la méthode `jspDestroy()` enregistre les données, libère les ressources utilisées par la servlet puis ferme les connexions aux BD
 - le programmeur peut redéfinir `jspDestroy()` et `_jspInit()`

Cycle de vie de la servlet générée

Traduction JSP->servlet

- les variables déclarées dans des balises `<%!...%>` sont traduites en variables d'instance
- idem pour les méthodes
- Les variables déclarées à l'intérieur d'une scriptlet sont traduites en variables locales de la méthode `_jspService()`
- Les expressions JSP `<%=...%>` sont placées dans un flux de sortie de type `out.println(...)`
- Les balises HTML sont placées dans un flux d'écriture de type `out.write(...)`
- Le code Java interne à une scriptlet est placé tel quel à l'intérieur de la méthode `_jspService()`

Java Beans

- Définition
- Persistance
- Propriétés
- Beans et JSP
- Exemple

Le modèle de composants Java

- Le modèle de composants Java (Java beans) permet de construire une application en assemblant des entités logicielles par programme (ou visuellement)
- Ce modèle de composants est décrit par un ensemble d'APIs
- Pour fonctionner, ce modèle repose sur un ensemble d'outils formant un environnement dans lequel les composants interagissent
- Ce modèle utilise des conteneurs qui fixent un cadre pour la coopération entre composants
- Dans l'architecture des applications web, les composants métiers sont implantés par des Java beans
- Modèle à composant alternatif : ActiveX de Microsoft

Définition

- Un bean correspond à une tâche bien définie (logique métier) comme la gestion d'une piscine, l'inventaire d'un magasin, le coût d'entretien d'une turbine, etc...
- Un bean est une classe Java à laquelle on associe des propriétés
- Chaque propriété a une ou plusieurs méthodes d'accès
- Le codage des méthodes suit certaines conventions de nommage

Persistence

- Un bean est un objet Java
- Comme tout objet, il possède un état
- Cet état doit être mémorisé pour toute utilisation ultérieure du bean
- Tout bean doit donc utiliser un mécanisme de persistance universel défini par leur conteneur
- A cette fin, ils se doivent d'implémenter l'interface [java.io.Serializable](#)

Exemple :

- composant logiciel chargé de surveiller la température d'une piscine
- la température min de déclenchement du chauffage est fixée à 18°
- si l'application s'arrête et reprend plus tard, il est souhaitable que ce seuil soit mémorisé

Construction d'un java bean

Les Java Beans sont des classes Java qui respectent les directives suivantes :

- un constructeur public sans argument
- les propriétés d'un Bean sont accessibles au travers de méthodes `getXXX` (lecture) et `setXXX` (écriture) portant le nom de la propriété

Méthodes de lecture des propriétés (pas de paramètre et le type de retour est celui de la propriété) :

```
type getNomDeLaPropriété()
```

Ou (cas d'une propriété booléenne)

```
boolean isNomDeLaPropriété()
```

Méthodes d'écriture des propriétés :

```
void setNomDeLaPropriété(type):
```

 un seul argument du type de la propriété et son type de retour est void

Un Java Beans implémente l'interface `java.io.Serializable` permettant la sauvegarde de l'état du Bean


Autres propriétés

Propriétés possédant des valeurs multiple :

```
public TypePropriété[] getNomPropriété();  
public TypePropriété getNomPropriété(int index);  
public void setNomPropriété(TypePropriété[] valeur);  
public void setNomPropriété (int index, TypePropriété[] valeur);
```

Java bean : exemple

```
public class Personne implements Serializable{
    private String nom;
    private String prenom;
    private int age;
    public Personne(){
    public Personne(String nom, String prenom, int age) {
        this.nom = nom;this.prenom = prenom;this.age = age;}
    public int getAge() {return age;}
    public String getNom() {return nom;}
    public void setAge(int age) {this.age = age;}
    public void setNom(String nom) { this.nom = nom;}
    public void setPrenom(String prenom){this.prenom = prenom;}
    public String getPrenom() {return prenom;}
}
```



Beans et JSP

- Un bean est inclus dans une page JSP par la balise :
`<jsp:useBean>`
- Après exécution, un bean rend son résultat à la page JSP ou à la servlet
- L'utilisation de beans évite l'emploi de la méthode `getRequestParameter(...)`
- La valeur du paramètre d'une requête peut être utilisée directement pour renseigner la propriété d'un bean
- Un serveur peut gérer la portée d'un bean en le liant soit à une page, une requête, une session ou une application

Portée d'un bean (1/2)

- Portée Page
 - Portée par défaut des beans
 - Tout objet de cette portée disparaît dès que la page est générée
- Portée Request
 - Même durée de vie que la requête
 - Si la requête est forwardée (jsp:forward), le bean reste utilisable dans la JSP forwardée
- Portée Session
 - Le bean est une donnée associée à l'utilisateur pendant sa visite sur le site

Portée d'un bean (2/2)

```
<jsp:useBean id = "nomDuBean"  
    scope = "page|request|session|application"  
    class = "nomPackage.nomClasse"  
    type = "typeDuBean">  
</jsp:useBean>
```

- `class = "nomClasse"` on suppose ici que la classe `nomClasse` se trouve dans le package `nomPackage` de l'application
- `class = "fr.cnam.eclipse.nomPackage.nomClasse"`

On indique ici le nom absolu de la classe

- `type` est utilisé pour le transtypage. Il indique soit une classe mère soit une interface de la classe

Propriétés d'un bean

* => pour transmettre toutes les valeurs de la requête dans les propriétés du bean

```
<jsp:setproperty name="nomDuBean" property="*" />
```

- Si une propriété spécifique (`unePropriete`) doit être modifiée

```
<jsp:setproperty name="nomDuBean"  
    property="unePropriete" />
```

- On utilise ici un paramètre de requête dont le nom est différent de celui de la propriété du bean

```
<jsp:setproperty name="nomDuBean" property="unePropriete"  
param=nomduParametre/>
```

- La propriété prend directement la valeur constante

```
<jsp:setproperty name="nomDuBean" property="unePropriete"  
value="constante" />
```

Exemple 1 :

Affichage des propriétés d'un bean (1/4)

- La page JSP qui suit affiche le texte : `Bonjour` suivi de la valeur du paramètre `nom` de la requête.
- La classe `HelloBean` est stockée dans le répertoire `entites`.
- La propriété `nom` du bean `HelloBean` est fixée par la balise `setProperty`
- La propriété `prenom` du bean a une valeur par défaut
- L'exécution de la page JSP provoque l'affichage du texte `bonjour` suivi du contenu des 2 propriétés du bean `HelloBean`
- Intérêt :
 - pas de code Java dans la page JSP
 - modularité dans la construction de l'application

Exemple 1 : Affichage des propriétés d'un bean (2/4)

```
package entites;
import java.io.Serializable;
public class HelloBean implements Serializable{
    private String nom = " à tous";
    private String prenom = " ";
    public void setNom( String nom ){
        this.nom=nom;
    }
    public String getNom(){
        return nom;
    }
    public void setPrenom( String prenom ){
        this.prenom=prenom;
    }
    public String getPrenom(){
        return prenom;
    }
}
```

Exemple 1 : Affichage des propriétés d'un bean (3/4)

```
<%-- hello.jsp --%>
```

```
<jsp:useBean id="hello" class="entites.HelloBean">
```

```
<jsp:setProperty name="hello" property="prenom"
    value="Albert" />
```

```
</jsp:useBean>
```

valeur à donner à la propriété

```
<HTML>
```

```
<HEAD><TITLE>Bonjour</TITLE></HEAD>
```

```
<BODY>
```

```
<H2>
```

```
Bonjour <jsp:getProperty name="hello" property="nom" /><br/>
```

```
<jsp:getProperty name="hello" property="prenom" />
```

```
</H2>
```

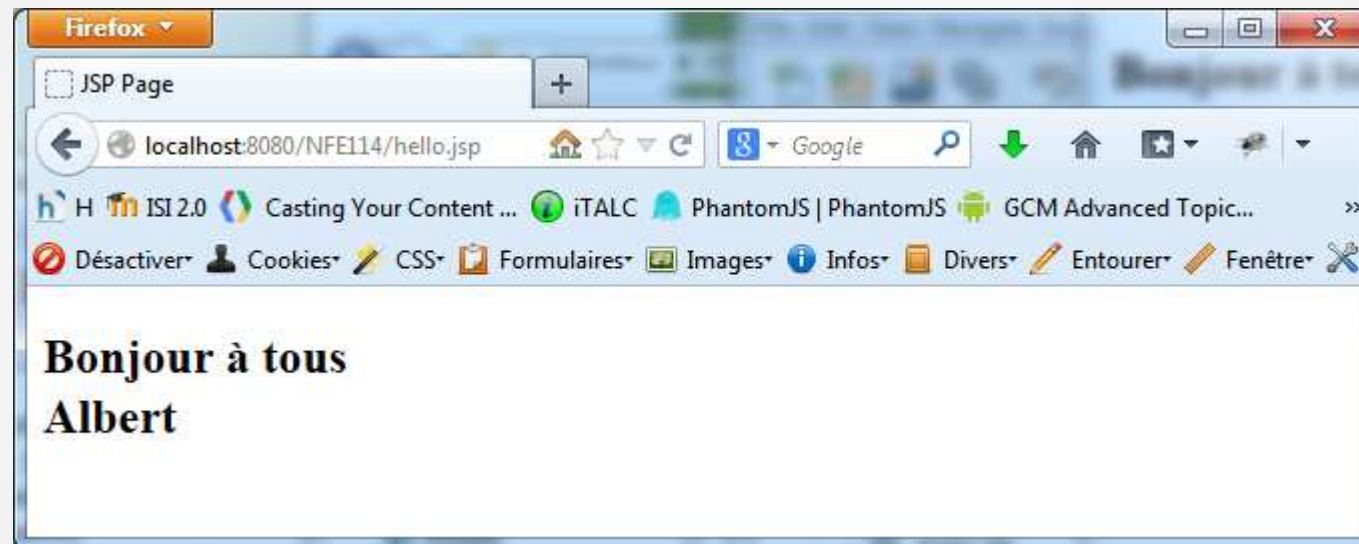
```
</BODY>
```

```
</HTML>
```

référence du bean

nom de la propriété

Exemple 1 : Affichage des propriétés d'un bean (4/4)



Exemple 2 (1/2)

Saisie, modification d'une propriété, appel d'une méthode du bean (1/2)

```
package fr.cnam.beans;
import java.io.Serializable;
public class Disque implements Serializable{
    public Disque() {}
    private double rayon;
    public double surface(){
        return Math.PI*rayon*rayon;
    }
    public double getRayon() {
        return rayon;
    }
    public void setRayon(double rayon) {
        this.rayon = rayon;
    }
}
```

Exemple 2 (2/2)

```
<jsp:useBean id="disque" class="fr.cnam.beans.Disque" />
<jsp:setProperty name="disque" property="rayon" />
```

```
<form action="" method="get" />
    Rayon = <input type="text" name="rayon" />
    <input type="submit" />
</form><br />
```

```
Surface = <%= disque.surface() %>
```

Rayon = Envoyer

Surface = 0.0

Rayon = 12.6 Envoyer

Surface = 0.0

Rayon = Envoyer

Surface = 997.5184993678309

Exemple 3 (1/2)

Saisie, modification d'une propriété, affichage d'une nouvelle page

```
package fr.cnam.beans;
import java.io.Serializable;
public class User implements Serializable{
    private String nom;
    private String prenom;
    public void setNom( String nom ){
        this.nom = nom;
    }
    public String getNom(){ return nom; }
    public void setPrenom( String prenom ){
        this.prenom = prenom;
    }
    public String getPrenom(){ return prenom; }
}
```

Exemple 3 (2/2)

```
<h3>Formulaire</h3>
<form action="user.jsp" method="get" >
Votre nom : <input type="text" name="nom" />
<input type="submit" value="valider" />
</form>
```

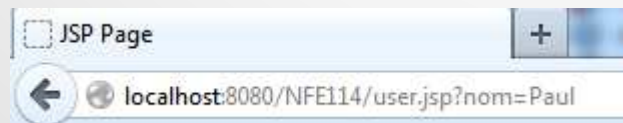
Formulaire

Votre nom :

Formulaire

Votre nom :

```
<h3>Bienvenue!</h3>
<jsp:useBean id="obj" class="fr.cnam.beans.User" />
<jsp:setProperty name="obj" property="nom" />
Bienvenue, <jsp:getProperty name="obj" property="nom" />
```



Bienvenue!

Bienvenue, Paul

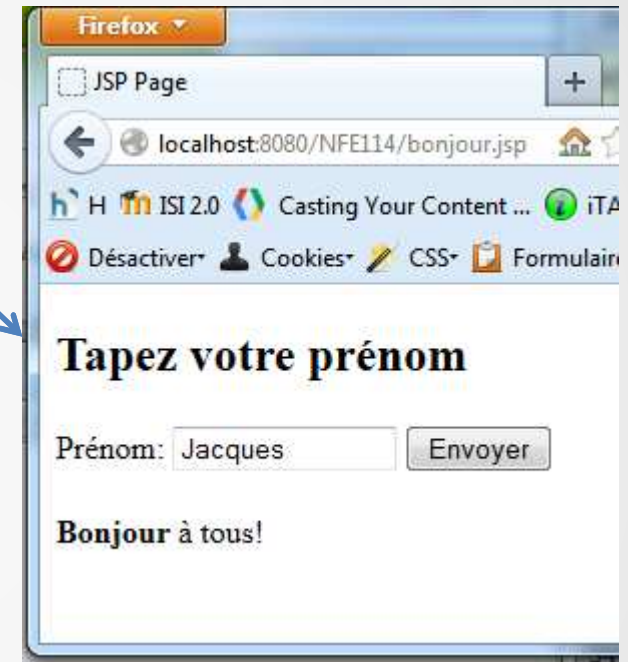
Exemple 4

Saisie et réaffichage de la page

Un prénom est saisi à travers un formulaire



Page affichée en réponse



Exemple 4

Saisie et réaffichage de la page

```
package entites;
import java.io.Serializable;

public class HelloBean implements Serializable{
    private String nom;
    private String prenom;
    public void setNom( String nom ){
        this.nom = nom;
    }
    public String getNom(){ return nom; }
    public void setPrenom( String prenom ){
        this.prenom = prenom;
    }
    public String getPrenom(){ return prenom; }
}
```

Exemple 4

Saisie et réaffichage de la page

bonjour.jsp:

initialisation

```
<jsp:useBean id="hello" class="entites.HelloBean">
<jsp:setProperty name="hello" property="prenom" value=" " />
<jsp:setProperty name="hello" property="nom" value=" à tous!" />
</jsp:useBean>

<jsp:setProperty name="hello" property="prenom" />

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
    <title>Bonjour</title>
  </head>
  <body>
```

Exemple 4

Saisie et réaffichage de la page

```
<h2>Tapez votre prénom</h2>
<form action="bonjour.jsp" method="post">
Prénom:
<input name="prenom" value="" type="text" size="15">
<input type="submit" value="Envoyer">
</form>

<br/><b>Bonjour</b>
<jsp:getProperty name="hello" property="nom" /><br/>
<jsp:getProperty name="hello" property="prenom" />

</body>
</html>
```

Exemple 4

Fonctionnement (1/2)

Initialisation



après le premier appel
à `bonjour.jsp`, `nom` et
`prenom` sont initialisés

L'objet `hello` est créé et initialisé avec les valeurs par défaut comprises entre les balises

```
<jsp:useBean id="hello"
```

```
</jsp:useBean>
```

Exemple 4

Fonctionnement (2/2)

Après validation du formulaire, la nouvelle valeur de la propriété **prenom** est transmise à l'objet **hello** par la balise jsp :

```
<jsp:setProperty name="hello" property="prenom" />
```

La page **bonjour.jsp** est de nouveau invoquée et la valeur des propriétés **nom** et **prenom** est affichée

Tapez votre prénom

Prénom:

Bonjour à tous!
Jacques

```
<b>Bonjour</b>
```

```
<jsp:getProperty name="hello" property="nom" /><br />
```


```
<jsp:getProperty name="hello" property="prenom" />
```


La directive taglib

L'API 1.1 de Java Server Pages permet aux développeurs de créer une bibliothèque de balises qui peuvent être mélangées aux balises jsp standards ainsi qu'aux balises html.

Le moteur jsp traite ces balises comme des balises standards pour créer du code java compilé ensuite en servlets.

```
<%@taglib uri="taglib/maBalise.tld"  
    prefix="unPrefixe"%>
```



Fichier xml spécifiant
l'ensemble des attributs de
chaque balise

La directive taglib

Pour définir une balise ("Tag ")

- Définir un gestionnaire de balises ("Handler ")
 - Objet invoqué par le container web pour évaluer le tag pendant l'exécution de la JSP qui le référence
 - Objet qui implante une interface **Tag** ou **BodyTag** ou dérive d'une classe de base **TagSupport** ou **BodyTagSupport**
 - Et définit les méthodes **doStartTag()** et **doEndTag()**
- Le déclarer dans un TLD (" tag library descriptor "), document XML qui décrit une bibliothèque de balises

La directive taglib

Exemple

Le tag : <tt:simple />

Serait implanter par le gestionnaire :

```
public SimpleTag extends TagSupport {
    public int doStartTag() throws JspException {
        try {
            pageContext.getOut().print("Hello.");
        } catch (Exception ex) {
            throw new JspTagException("SimpleTag: " +
                ex.getMessage());
        }
        return SKIP_BODY; // ce tag n'a pas de corps
    }
    public int doEndTag() {
        return EVAL_PAGE; // si le reste de la page
        // doit être évalué
    }
}
```