



# Java EE

-

## Cours 3

Cours de 2<sup>e</sup> année ingénieur  
Spécialisation « Génie Informatique »



# Problème d'une Servlet : HTML dans Java

## 1. C'est laid!

```
out.println(docType);  
out.println("<html>");  
out.println("<head>\n<title>Clock</title>\n</head>");  
out.println("<body>\n"+  
            "<h1>Time on server</h1>\n"+  
            "<p>"+ today+"</p>\n"+  
            "</body>");  
out.println("</html>");
```

## 2. Les ingénieurs EISTI connaissent Java, mais les développeurs Web, NON !!

- Difficile de séparer les différentes tâches du développement (Web code vs. Business code)

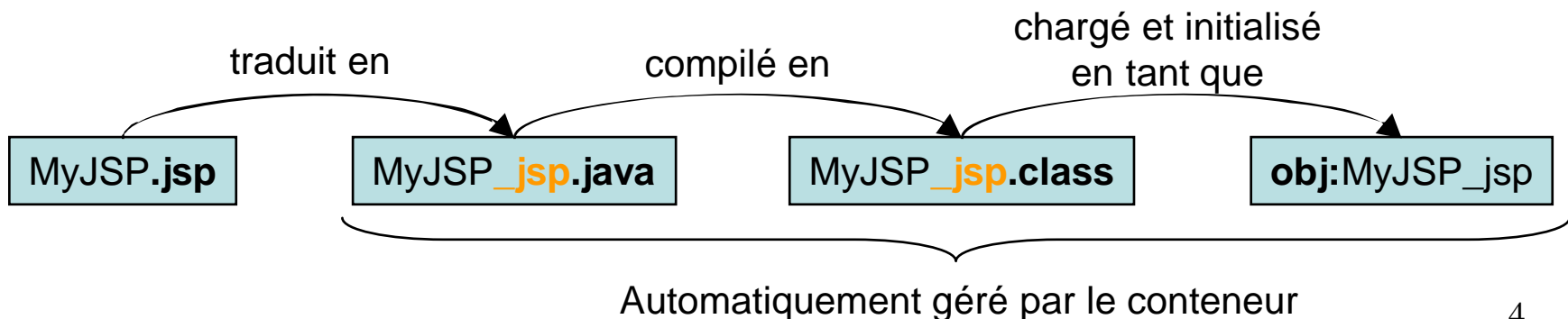
# Solution : Java dans HTML

- Une JSP est identique à une page HTML dans laquelle on peut ajouter du code Java (même principe qu'avec PHP)

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <title>Clock</title>
  </head>
  <body>
    <h1>Time on server</h1>
    <p> <%= new java.util.Date() %> </p>
  </body>
</html>
```

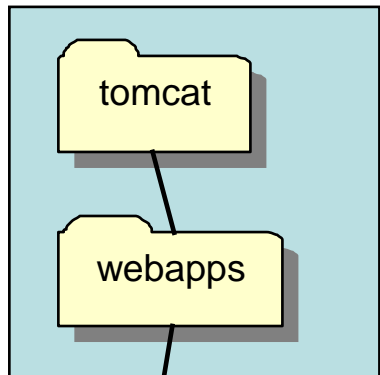
# JSP

- Les servlets facilitent le traitement avec java des requêtes et réponses HTTP, **mais** ils ne sont pas appropriés à l'écriture de code HTML
  - `out.println("<html><head><title>"+title+"</title>...");`
- Les JSP permettent d'intégrer du code java dans une page HTML
  - `<h1>Time on server</h1>`  
`<p><%= new java.util.Date() %></p>`
- Mais au final une JSP n'est qu'un servlet!

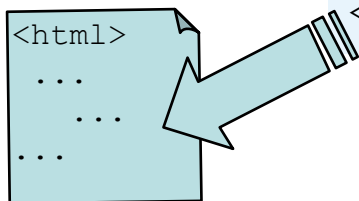


# Une JSP en 1 étape

- Ecrire une JSP Clock.jsp et la mettre dans AREL (le répertoire de notre application Web)



```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <title>Clock</title>
  </head>
  <body>
    <h1>Time on server</h1>
    <p> <%= new java.util.Date() %> </p>
  </body>
</html>
```



Clock.jsp

# Cycle de vie d'une JSP

		Request #1	Request #2		Request #3	Request #4		Request #5	Request #6
JSP page translated into servlet	Page first written	Yes	No	Server restarted	No	No	Page modified	Yes	No
Servlet compiled		Yes	No		No	No		Yes	No
Servlet instantiated and loaded into server's memory		Yes	No		Yes	No		Yes	No
init (or equivalent) called		Yes	No		Yes	No		Yes	No
doGet (or equivalent) called		Yes	Yes		Yes	Yes		Yes	Yes

# Correspondance JSP/Servlet

- JSP d'origine

```
<h1>Time on server</h1>
<p><%= new java.util.Date() %></p>
```

- Servlet généré par Tomcat

```
public final class Clock_jsp
    extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent{
    public void _jspService(HttpServletRequest request,
                            HttpServletResponse response)
        throws java.io.IOException, ServletException{

        response.setContentType("text/html");
        JspWriter out = response.getWriter();
        // ...
        out.write("<h1>Time on server</h1>\r\n");
        out.write("<p>");
        out.print( new java.util.Date() );
        out.write("</p>\r\n");
        // ...
    }
}
```

# Stratégie de conception: Limiter le code Java dans les JSP

- Deux options
  - Ecrire 25 lignes de code directement dans une JSP
  - Ecrire ces 25 lignes dans une classe Java à part et 1 ligne dans une JSP pour l'invoquer
- Pourquoi la 2<sup>e</sup> option est vraiment meilleure?
  - **Développement.** Ecriture de la classe dans un environnement Java et pas HTML
  - **Debugage.** S'il y a des erreurs, elles sont visible à la compilation
  - **Test.** L'accès à la classe Java facilite le test (ex: boucle de test de 100000 itérations sur un serveur...)
  - **Réutilisation.** Utilisation de la même classe dans différentes pages JSP



# Versions JSP

- Extensions de fichiers
  - .jsp, page JSP standard
  - .jspx, page JSP compatible XML
- Deux syntaxes
  - Standard (JSP 1.2)
  - XML (JSP 2.0)
- Depuis la 2.0 : Extensible à travers des librairies de tag (fichier **.tld**) importés ou personnels.

# Syntaxe de base

- Texte HTML
  - `<h1>Blah</h1>`
  - Passé au client. Réellement traduit en servlet par le code
    - `out.print("<h1>Blah</h1>");`
- Commentaires HTML
  - `<!-- Commentaire -->`
  - Pareil que les autres éléments HTML: passés au client
- Commentaires JSP
  - `<%-- Commentaires --%>`
  - Ne sont pas envoyés au client
- Echappement `<%`
  - Pour obtenir `<%` dans la sortie, utiliser `<%\%`

# Types des éléments de scripts

- Expressions
  - Format : `<%= expression %>`
  - Évaluée et insérée dans la sortie du servlet  
Se traduit par `out.print(expression)`
- Scriptlets
  - Format : `<% code %>`
  - Inséré tel quel dans la méthode `_jspService` du servlet
- Déclarations
  - Format : `<%! code %>`
  - Pour déclarer une variable ou bien une fonction / procédure.
- Include / Import / ...
  - Format : `<%@ directive ... %>`
  - Ex : `<%@ include file="unAutreFichier" %>`  
`<%@ page import="java.util.*" %>`  
`<%@ taglib prefix="myprefix" uri="taglib/mytag.tld" %>`

Expressions JSP:  
<%= valeur %>

# Expressions JSP

- Format
  - `<%= Expression Java %>`
- Résultat
  - Expression évaluée, convertie en String, et placée dans la page HTML à la place qu'elle occupe dans la JSP
  - L'expression est placée dans `_jspService` en paramètre du `out.print()`
- Exemples
  - Heurecourante: `<%= new java.util.Date() %>`
  - Hostname: `<%= request.getRemoteHost() %>`
- Syntaxe compatible XML (JSP 2.0)
  - `<jsp:expression> Java Expression </jsp:expression>`
  - On ne peut pas mixer les deux versions dans une même page. Il faut utiliser XML pour la page *entière* si on utilise `jsp:expression`.

# Correspondance JSP/Servlet

- JSP d'origine

```
<h1>Un nombre aléatoire</h1>
```

```
<%= Math.random() %>
```

- Code du servlet résultant de la traduction

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
    throws ServletException, IOException{
    response.setContentType("text/html");
    HttpSession session = request.getSession();
    JspWriter out = response.getWriter();
    out.println("<h1>Un nombre aléatoire</h1>");
    out.println(Math.random());
    ...
}
```

# Expressions JSP: Exemple

```
<body>
```

```
<h2>JSP Expressions</h2>
```

```
<ul>
```

```
<li>Current time: <%= new java.util.Date() %> </li>
```

```
<li>Server: <%= application.getServerInfo() %> </li>
```

```
<li>Session ID: <%= session.getId() %> </li>
```

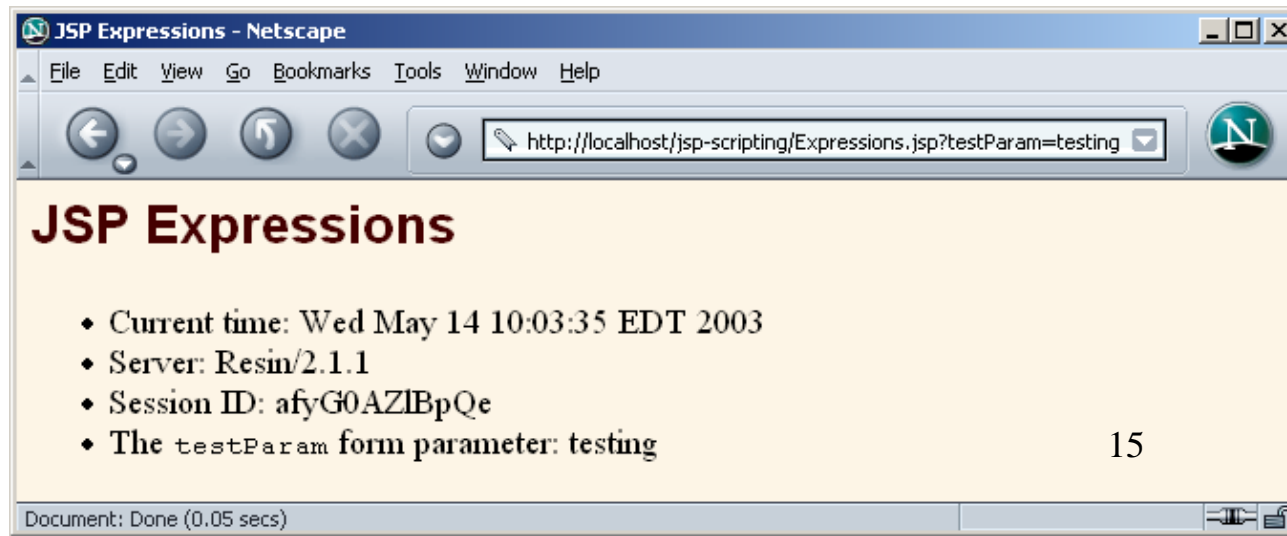
```
<li>The testParam form parameter:
```

```
<%= request.getParameter("testParam") %>
```

```
</ul>
```

```
</body>
```

```
</html>
```



# Variables prédéfinies

- request
  - Instance de `HttpServletRequest` (1<sup>e</sup> argument de `service/doGet`)
- response
  - Instance de `HttpServletResponse` (2<sup>e</sup> argument de `service/doGet`)
- out
  - Instance de `JspWriter` (une version bufferisée de `Writer`) utilisée pour envoyer des données sur la sortie vers le client
- session
  - Instance de `HttpSession` associée à la requête (sauf si désactivée avec l'attribut `session` de la directive de la page)
- application
  - Instance de `ServletContext` (pour partager des données) telle que obtenue via `getServletContext()`



# Servlets vs JSP:

## Lire 3 paramètres (Servlet)

```
public class ThreeParams extends HttpServlet{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException{
    ...
    out.println(docType+
        "<html>\n"+
        "<head><title>" +title+ "</title></head>\n"+
        "<body bgcolor=\"#FDF5E6\">\n"+
        "<h1 align=\"CENTER\">" + title + "</h1>\n"+
        "<ul>\n"+
        "<li><b>param1</b>:"
        + request.getParameter("param1")+"</li>\n"+
        "<li><b>param2</b>:"
        + request.getParameter("param2")+"</li>\n"+
        "<li><b>param3</b>:"
        + request.getParameter("param3")+"</li>\n"+
        "</ul>\n"+
        "</body></html>");
    }
}
```

# Lire 3 paramètres (Servlet) : Résultat





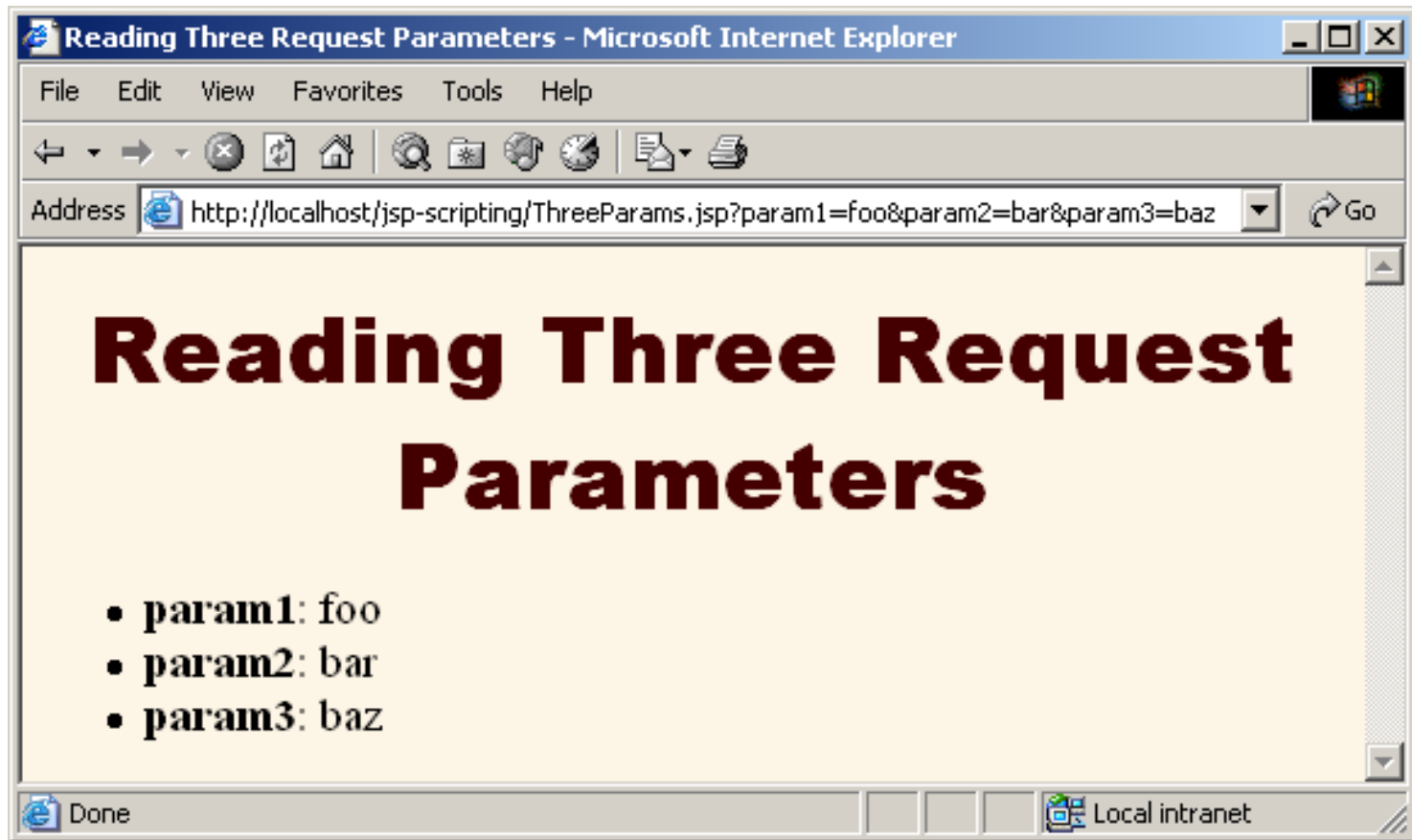
# Servlets vs JSP:

## Lire 3 paramètres (JSP)

```
<!DOCTYPE ...>
<html>
<head>
<title>Reading Three Request Parameters</title>
<link rel = STYLESHEET
      href = "JSP-Styles.css"
      type = "text/css">
</head>
<body>
<h1>Reading Three Request Parameters</h1>
<ul>
  <li><b>param1</b>:
      <%= request.getParameter("param1") %> </li>
  <li><b>param2</b>:
      <%= request.getParameter("param2") %> </li>
  <li><b>param3</b>:
      <%= request.getParameter("param3") %> </li>
</ul>
</body></html>
```

# Lire 3 paramètres (JSP) :

## Résultat



Scriptlets JSP:  
<% Code %>

# Scriptlets JSP

- Format
  - `<% Code Java %>`
- Résultat
  - Code inséré tel quel dans `_jspService()`
- Exemple
  - `<%  
String queryData = request.getQueryString();  
out.println("Attached GET data :" + queryData);  
%>`
  - `<% response.setContentType("text/plain"); %>`
- Syntaxe XML
  - `<jsp:scriptlet>Code Java</jsp:scriptlet>`

# Correspondance JSP/Servlet

- JSP d'origine

```
<h2>foo</h2>
```

```
<%= bar() %>
```

```
<% baz(); %>
```

- Code du servlet résultant de la traduction

```
public void _jspService(HttpServletRequest request,  
                        HttpServletResponse response)  
    throws ServletException, IOException{  
    response.setContentType("text/html");  
    HttpSession session = request.getSession();  
    JspWriter out = response.getWriter();  
    out.println("<h2>foo</h2>");  
    out.println(bar());  
    baz();  
    ...  
}
```

# Scriptlets JSP: Exemple

- On veut permettre à l'utilisateur de choisir la couleur de fond de la page HTML
  - Quel est le problème avec ce code?

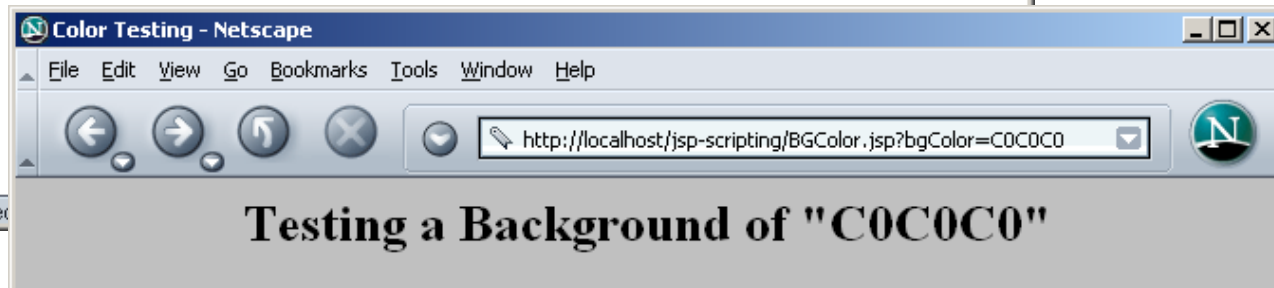
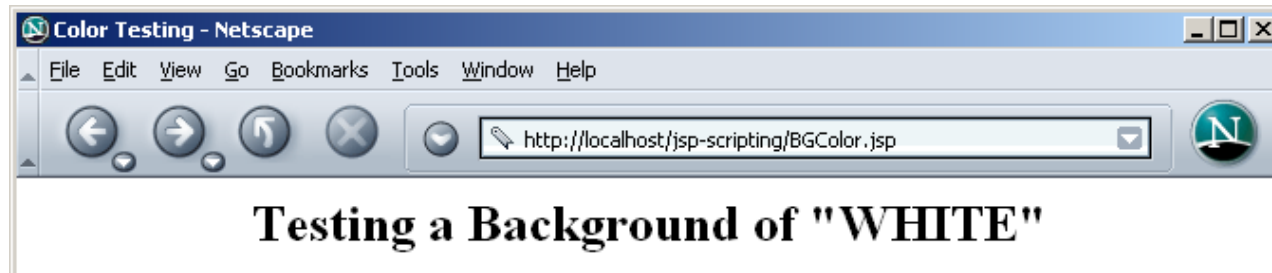
```
<body bgcolor = " <%= request.getParameter("bgColor") %> ">
```



# Scriptlets JSP: Exemple

```
<!DOCTYPE ...>
<html>
<head>
  <title>Color Testing</title>
</head>
<%
String bgColor = request.getParameter("bgColor");
if ((bgColor == null)|| (bgColor.trim().equals(""))){
  bgColor = "WHITE";
}
%>
<body bgcolor=" <%= bgColor %> ">
<h2 align="CENTER">Testing a Background of
"<%= bgColor %>".</h2>
</body></html>
```

# Scriptlets JSP: Résultat



# Utilisation des scriptlets pour créer des parties conditionnelles

- Postulat
  - Les scriptlets sont insérés tels quels dans le servlet
  - Pas besoin d'avoir des expressions Java complètes
  - Cependant, les expressions complètes sont la plupart du temps plus claires et faciles à maintenir
- Exemple
  - ```
<% if (Math.random() < 0.5) { %>
  Vous avez <b>gagné</b> !
<% } else { %>
  Vous avez <b>perdu</b>!
<%} %>
```
- Code du servlet résultant de la traduction
  - ```
if(Math.random() < 0.5) {
    out.println("Vous avez <b>gagné</b> !");
} else {
    out.println("Vous avez <b>perdu</b> !");
}
```

Déclarations JSP:  
<%! Code %>

# Déclarations JSP

- Format
  - `<%! Java Code %>`
- Résultat
  - Insérées telle quelle dans la définition de la classe du servlet, en dehors de toute méthode existante
- Exemples
  - `<%! private int someField = 5; %>`
  - `<%! private void someMethod(...) {...} %>`
- Remarque de conception
  - Les attributs sont clairement utiles. Pour les méthodes, il est la plupart du temps préférable de les définir dans une classe Java séparée
- Syntaxe XML
  - `<jsp:declaration> Code Java </jsp:declaration>`

# Correspondance JSP/Servlet

- JSP d'origine

```
<h1>Some Heading</h1>
```

```
<%!
```

```
    private String randomHeading(){  
        return("<h2>" + Math.random() + "</h2>");  
    }  
}
```

```
%>
```

```
<%= randomHeading() %>
```

- (Alternative : créer *randomHeading()* en méthode statique dans une classe Java séparée)

# Correspondance JSP/Servlet

- Code du servlet résultant de la traduction

```
public class MyJSP_jsp implements HttpJspPage{
    private String randomHeading(){
        return("<h2>" + Math.random() + "</h2>");
    }

    public void _jspService(HttpServletRequest request,
                            HttpServletResponse response)
        throws ServletException, IOException{
        response.setContentType("text/html");
        HttpSession session = request.getSession();
        JspWriter out = response.getWriter();
        out.println("<h1>Some Heading</h1>");
        out.println(randomHeading());
        ...
    }
}
```

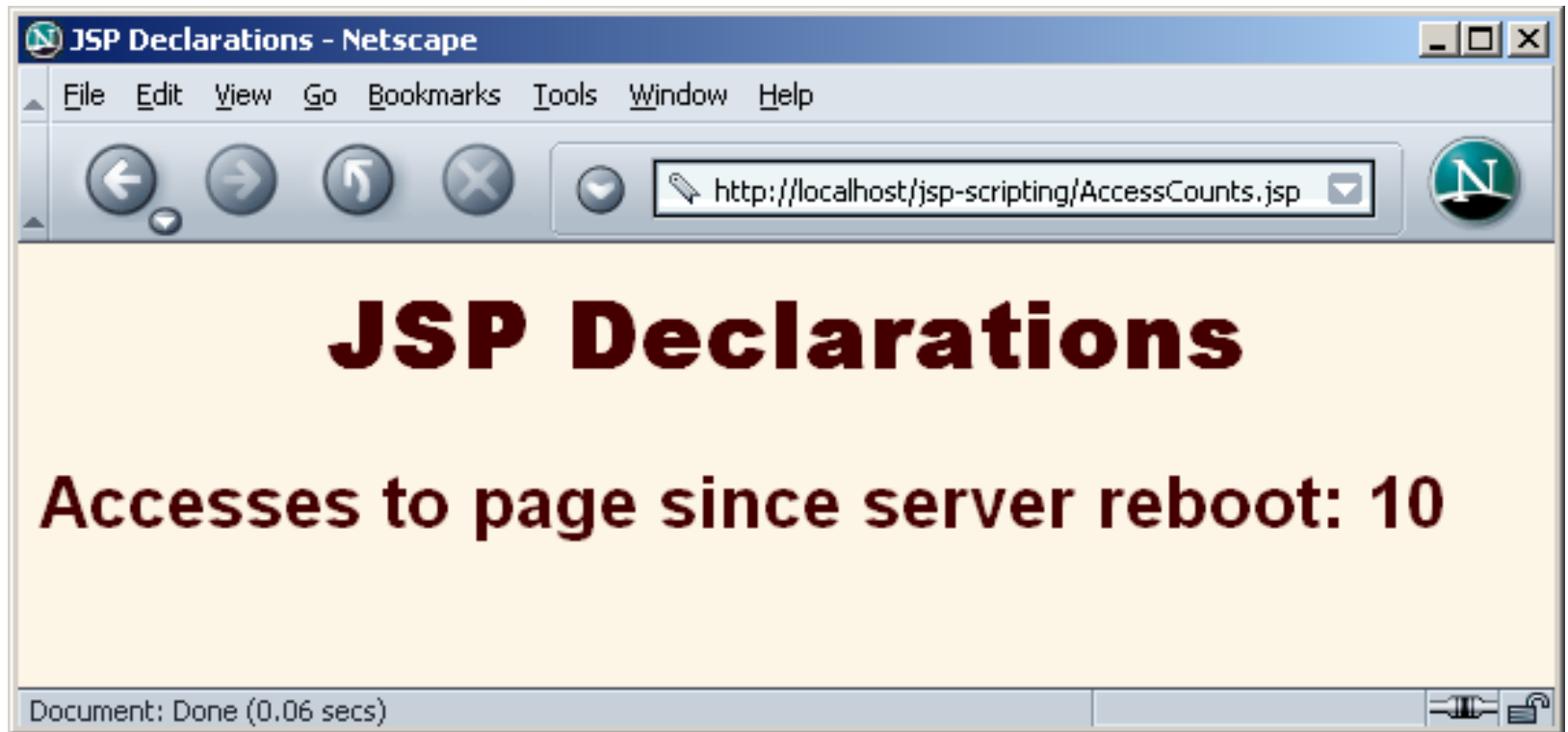


# Déclarations JSP: Exemple

```
<!DOCTYPE ...>
<html>
<head>
<title>JSP Declarations</title>
<link rel=STYLE SHEET
      href="JSP-Styles.css"
      type="text/css">
</head>
<body>
<h1>JSP Declarations</h1>
<%! private int accessCount = 0; %>
<h2>Accesses to page since server reboot:
<%= ++accessCount %> </h2>
</body></html>
```



# Déclarations JSP: Résultats



# Déclarations JSP: les méthodes `jspInit` et `jspDestroy`

- Les pages JSP, comme les servlets normaux, ont parfois besoin d'utiliser `init` et `destroy`
- Problème : le servlet qui résulte de la traduction de la page JSP peut déjà utiliser `init` et `destroy`
  - Les redéfinir causerait des problèmes
  - Ainsi, il n'est pas permis d'utiliser des déclarations JSP pour définir `init` ou `destroy`
- Solution : utiliser **`jspInit`** et **`jspDestroy`**
  - Le servlet auto-généré garantit que ces méthodes seront appelées depuis `init` et `destroy`, mais par défaut l'implémentation de `jspInit` et `jspDestroy` est vide (on peut donc les redéfinir)

# Déclarations JSP et variables prédéfinies

- Problème
  - Les variables prédéfinies (request, response, out, session, etc.) sont *locales* à la méthode `_jspService`.  
Ainsi, elles ne sont pas disponibles pour les méthodes définies par des déclarations JSP et les méthodes des classes externes.
  - Que peut-on faire?
- Solution : les passer en paramètres. Ex:

```
<%! private void someMethod(HttpSessions) {  
    doSomethingWith(s);  
} %>  
<% someMethod( session ); %>
```
- Rq1: les méthodes statiques ne résolvent pas le problème
  - Il faut également les passer en paramètres
- Rq2: `println` de `JSPWriter` lance une `IOException`
  - Utiliser “throws `IOException`” pour les méthodes qui utilisent `println`

# Pages JSP 2.0 (avec la syntaxe XML)

# Pourquoi deux versions?

- La syntaxe classique des JSP n'est pas compatible XML
  - `<%= ... %>`, `<% ... %>`, `<%! ... %>` ne sont pas permis en XML
  - HTML4 n'est pas compatible XML non plus
  - Donc au final, on ne peut pas utiliser des éditeurs XML
- Pourquoi voudrait-on utiliser du code JSP dans des environnements XML?
  - Pour produire du xhtml en particulier
  - Pour produire des documents XML en général
    - Il est possible d'utiliser la syntaxe classique pour créer des documents XML, mais il est plus simple de travailler en XML dès le début
      - pour Web services
      - pour les applications Ajax
- Il y a donc une seconde syntaxe qui suit les règles de XML

# Syntaxe XML pour générer des fichiers XHTML (somefile.jspx)

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<html xmlns:jsp="http://java.sun.com/JSP/Page">
```

```
<jsp:output
```

```
  doctype-root-element="html"
```

```
  doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
  doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-  
transitional.dtd" />
```

```
<jsp:directive.page contentType = "text/html"/>
```

```
<head><title>Some Title</title></head>
```

```
<body bgcolor="#fdf5e6">
```

```
  <!-- body -->
```

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

Le namespace jsp est nécessaire pour utiliser les commandes jsp:quelquechose. On peut utiliser d'autres namespaces pour se servir de bibliothèques de tags JSP

Construit la ligne du DOCTYPE

Pour les pages JSP en syntaxe XML, le *contentType* par défaut est text/xml

Contenu xhtml + commandes JSP utilisant la syntaxe jsp:quelquechose + bibliothèques de tags JSP



# Syntaxe XML pour générer des fichiers XML (somefile.jspx)

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<your-root-element xmlns:jsp="http://java.sun.com/JSP/Page">
```

```
  <your-tag1>foo</your-tag1>
```

```
  <your-tag2>bar</your-tag2>
```

```
</your-root-element>
```

- Utilisation

- Quand on communique avec un client qui attend du vrai XML

- Ajax
- Web services
- Custom clients

# Exemple de page HTML4: syntaxe classique (sample.jsp)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD ...">
<html>
<head><title>Sample (Classic Syntax)</title></head>
<body bgcolor="#FDF5E6">
<center>
<h1>Sample (Classic Syntax)</h1>

<h2>Num1: <%= Math.random()*10 %></h2>
<%= double num2 = Math.random()*100; %>
<h2>Num2: <%= num2 %></h2>
<%=! private double num3 = Math.random()*1000; %>
<h2>Num3: <%= num3 %></h2>

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

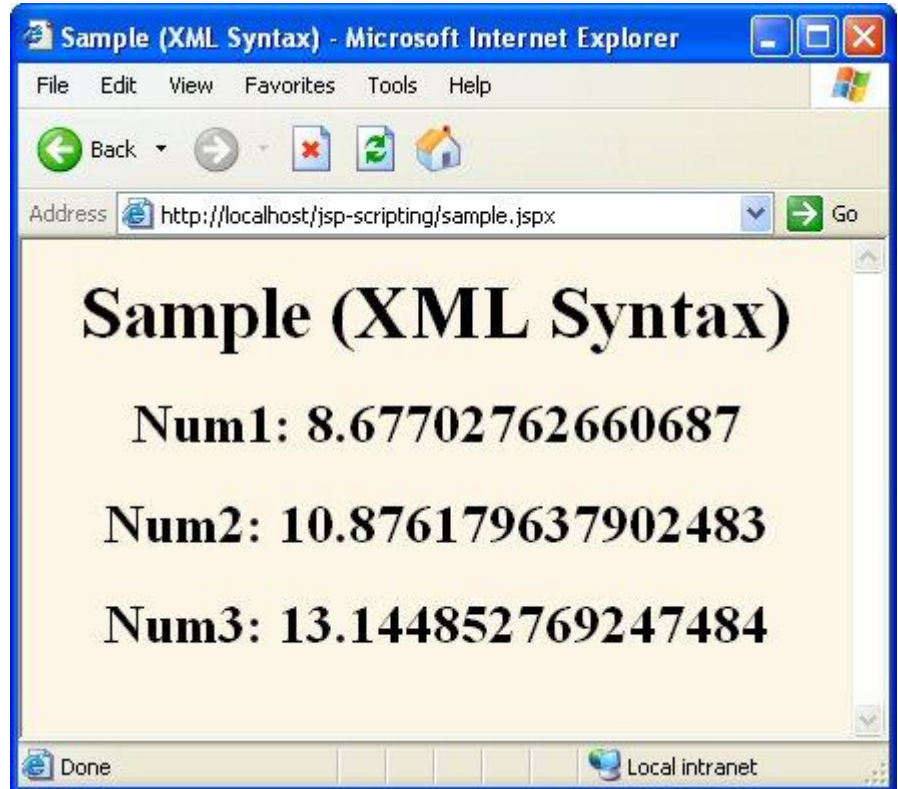
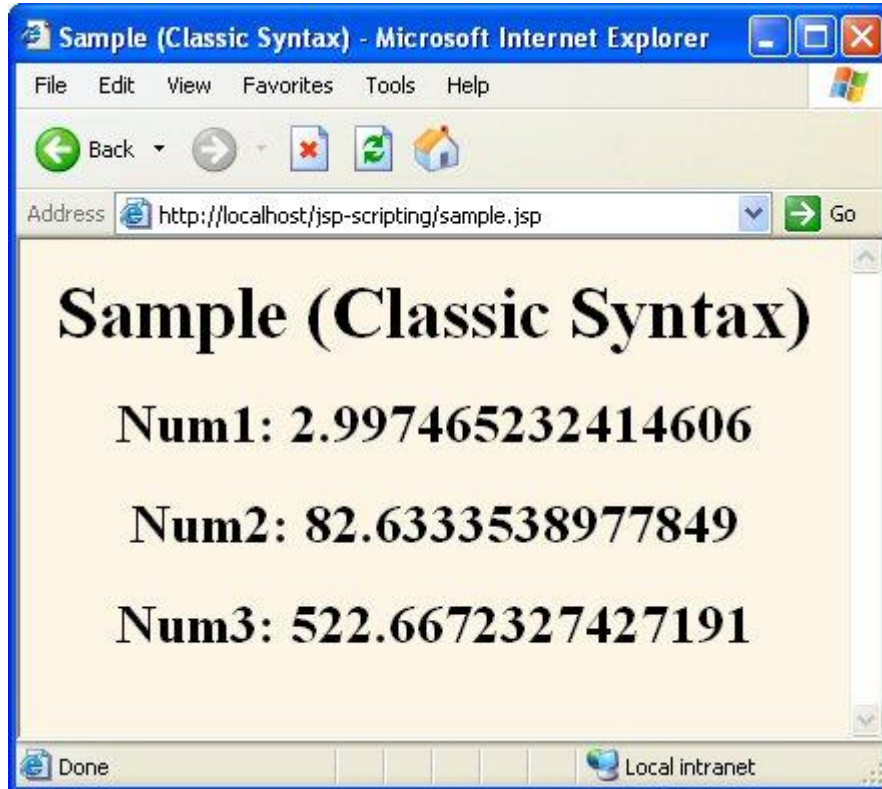




# Exemple de page XHTML: syntaxe XML (sample.jspx)

```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns:jsp="http://java.sun.com/JSP/Page">
<jsp:output
  doctype-root-element="html"
  doctype-public="-//W3C//DTD..."
  doctype-system="http://www.w3.org...dtd" />
<jsp:directive.page contentType="text/html"/>
<head><title>Sample (XML Syntax)</title></head>
<body bgcolor="#fdf5e6">
<div align="center">
<h1>Sample (XML Syntax)</h1>
<h2>Num1: <jsp:expression>Math.random()*10</jsp:expression></h2>
<jsp:scriptlet>
double num2 = Math.random()*100;
</jsp:scriptlet>
<h2>Num2: <jsp:expression>num2</jsp:expression></h2>
<jsp:declaration>
private double num3 = Math.random()*1000;
</jsp:declaration>
<h2>Num3: <jsp:expression>num3</jsp:expression></h2>
</div></body></html>
```

# Exemples de pages : Résultat



# Document XML généré avec une syntaxe XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<some-root-element
  xmlns:jsp="http://java.sun.com/JSP/Page">
  <some-element-1>Text</some-element-1>
  <some-element-2>
    Number:
    <jsp:expression>Math.random()*10</jsp:expression>
  </some-element-2>
</some-root-element>
```

