



Java™

ISIMA

**Java EE : Présenter une application Web**  
HTML, CSS et Javascript !

# Le cours du jour

- Rappel des cours précédents
  - Les Servlets pour réaliser un service
  - Les Filters pour mutualiser des traitements entre les Servlets
- Nous allons voir comment
  - Présenter une application Web
  - Naviguer d'une page à une autre pour créer les enchainement d'écrans

# Préserver dans une application Web

- Présenter c'est :
  - Montrer l'information produite par le système
  - Permet d'accéder à d'autres informations
- Dans une application Web
  - Dans notre cas, l'information est produite par le serveur
    - Certains frameworks prennent le parti d'avoir des clients plus robustes (ex AngularJs) et un MVC complet en Javascript
  - la présentation à afficher est décrite par un langage textuel
    - C'est le navigateur qui interprète le contenu
    - Il génère les UI correspondant à la description « textuel »

# HTML

- Définition
- *Hyper Text Markup Language*
  - Un langage de description d'interface homme machine
  - Il est connu de tous les navigateurs
  - La dernière version est HTML 5
- Communément utilisé pour l'interface homme machine
  - Autres cas :
    - Flash/flex
    - Applet

# Contenu statique

- Publier une page HTML sur un serveur applicatif
- Une page HTML simple est un contenu statique
- Comment le serveur applicatif délivre ce type de ressource ?

# Téléchargement d'une page HTML

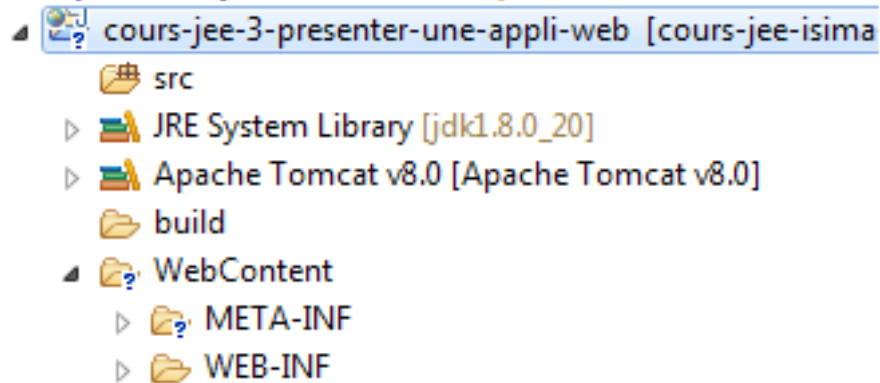
- Un schéma !

`http://aserver/cours-jee-3-presenter-une-appli-web/aPage.html`



# Créons aPage.html !

- A l'aide d'eclipse
- Nous avons un projet Web dynamique neuf



# WebContent

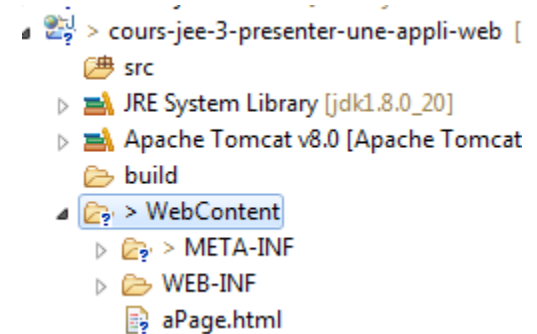
- Un répertoire rassemblant les fichiers à publier
- Le répertoire WebContent contient les ressources à publier
- Trois répertoires par défaut :
  - *WEB-INF* : pour stocker les infos qui ne doivent pas être accessible via HTTP ;
  - *META-INF* : des infos sur l'archive (le package de l'application)
  - *Classes* (caché dans la vue eclipse) contient les classes compilées



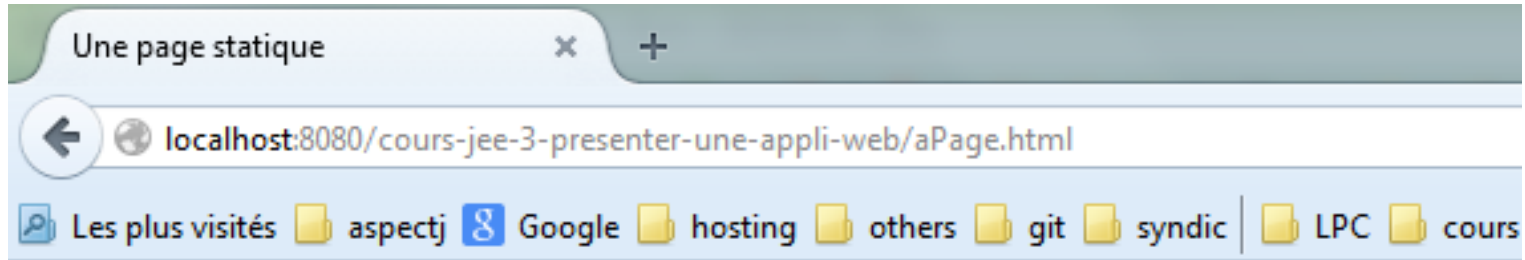
# aPage.html

- A la racine de WebContent, créons la page aPage.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Une page statique</title>
</head>
<body>
    <h1>Une page statique pour mon application</h1>
    <p>Un paragraphe html</p>
</body>
</html>
```



# Après publication de l'application



## Une page statique pour mon application

Un paragraphe html

# Les limites

- Le contenu statique est limité : il doit être écrit à l'avance sur le serveur
- Les données traitées par l'application ou dans une base ne peuvent être affichées
- Faible intérêt pour une application
- Une solution :
  - La génération de contenu dynamique

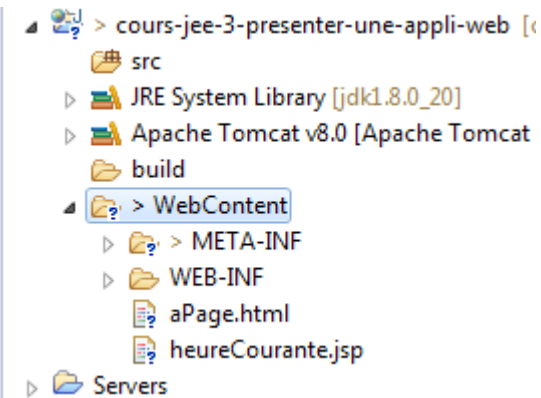
# Contenu dynamique

- Le contenu dynamique permet d'ajouter des informations du programme
- Intégrée à JEE, Java propose JSP - *Java Server Pages*
- Créons une JSP qui affiche l'heure courante

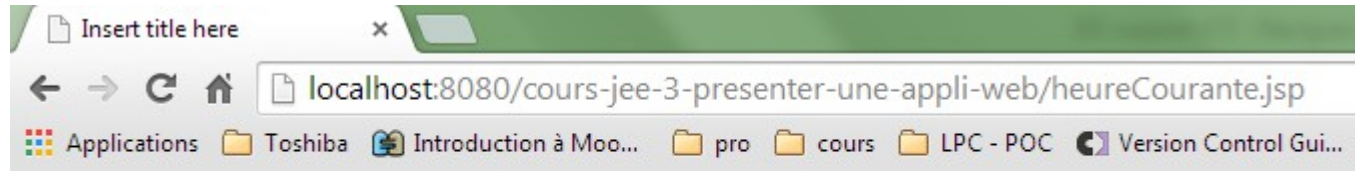
# heureCourante.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@page import="java.time.LocalDateTime"%>
<%@page import="java.time.format.DateTimeFormatter"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <%
        final DateTimeFormatter formatteur = DateTimeFormatter.ofPattern("HH:mm:ss 'le' dd/MM/yyyy");
        final String currentDateAsString = formatteur.format(LocalDateTime.now());
    %>
    <p>
        Au moment de la génération de cette page, il est
        <%=currentDateAsString%>
    </p>
</body>
</html>
```



# heureCourante.jsp sous Chrome



Au moment de la génération de cette page, il est 21:39:49 le 05/12/2014

# Décryptons notre page JSP !

- L'entête

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
```

Déclaration de jsp (pour le serveur applicatif)

```
<%@page import="java.time.LocalDateTime"%>
```

Import de la classe LocalDateTime de Java 8

```
<%@page import="java.time.format.DateTimeFormatter"%>
```

Import du formateur de date de Java 8

# Décryptons notre page JSP !

- Le formatage de la date

<%

```
final DateTimeFormatter formatteur =  
DateTimeFormatter.ofPattern("HH:mm:ss 'le' dd/MM/yyyy");  
final String currentDateAsString =  
formatteur.format(LocalDate.now());
```

%>

Le caractère <% %> déclare que nous allons exécuter un code en Java. Ici un formatage de date



# Décryptons notre page JSP !

- L'affichage de la date

<p>

Au moment de la génération de cette page, il est

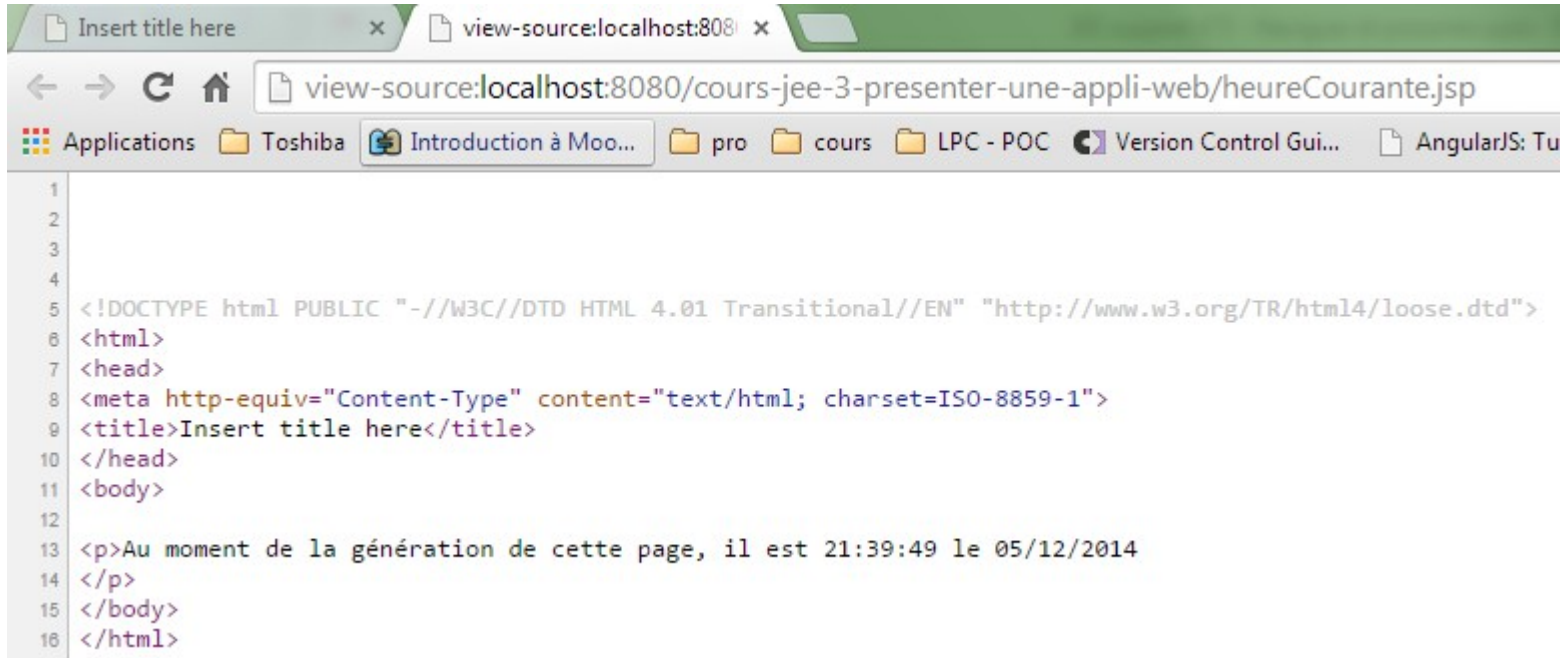
<%=currentDateAsString%>

</p>

- <%= signifie que l'on souhaite afficher la valeur de la variable ou le retour de la fonction - ici currentDateAsString

# Que reste-t-il coté client ?

- Un view source sous chrome !



```
1
2
3
4
5 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
6 <html>
7 <head>
8 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
9 <title>Insert title here</title>
10 </head>
11 <body>
12
13 <p>Au moment de la génération de cette page, il est 21:39:49 le 05/12/2014
14 </p>
15 </body>
16 </html>
```

# Que conclure ?

- Une page JSP permet d'afficher du contenu dynamique
- Le code JSP est interprété au niveau du serveur
  - Le client (navigateur) ne voit jamais les balises
- Le JSP est en fait une Servlet que le moteur JEE compile.
  - C'est pourquoi le 1er lancement est plus long !
  - Il est d'ailleurs possible au niveau du « work » de voir ce que donne la classe compilée

# Lien avec la Servlet

- La JSP doit se cantonner à la génération de l'affichage
  - Le code de « traitement » diminue la lisibilité du HTML
  - Principe de séparation des préoccupations
- La solution :
  - Créer une servlet qui effectue le traitement et transmet les informations à la JSP !

# AfficherHeureCourante

```
@WebServlet("/AfficherHeureCourante")
public class AfficherHeureCouranteServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        final DateTimeFormatter formatteur = DateTimeFormatter.ofPattern("HH:mm:ss
'le' dd/MM/yyyy");
        final String currentDateAsString = formatteur.format(LocalDate.now());
        request.setAttribute("currentDateAsString", currentDateAsString);
        request.getRequestDispatcher("/heureCourante.jsp").forward(request,
response);
    }
}
```

# Détail du code source

- Les instructions marquantes

- ```
request.setAttribute("currentDateAsString",  
currentDateAsString);
```

- Utilisation de la request pour sauver la date. Il permet de transmettre l'information à la JSP

- ```
request.getRequestDispatcher("/heureCourante.jsp").forward(request,  
response);
```

- Forward de la servlet vers la jsp heure courante

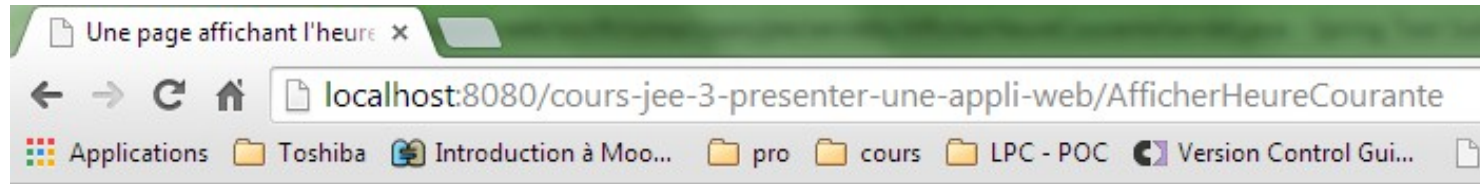
- Les informations de la requête sont transmises

# Et heureCourante.jsp ?

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Une page affichant l'heure courante</title>
</head>
<body>
    <p>
        Au moment de la génération de cette page, il est
        <%=request.getAttribute("currentDateAsString")%>
    </p>
</body>
</html>
```

# Sous Chrome

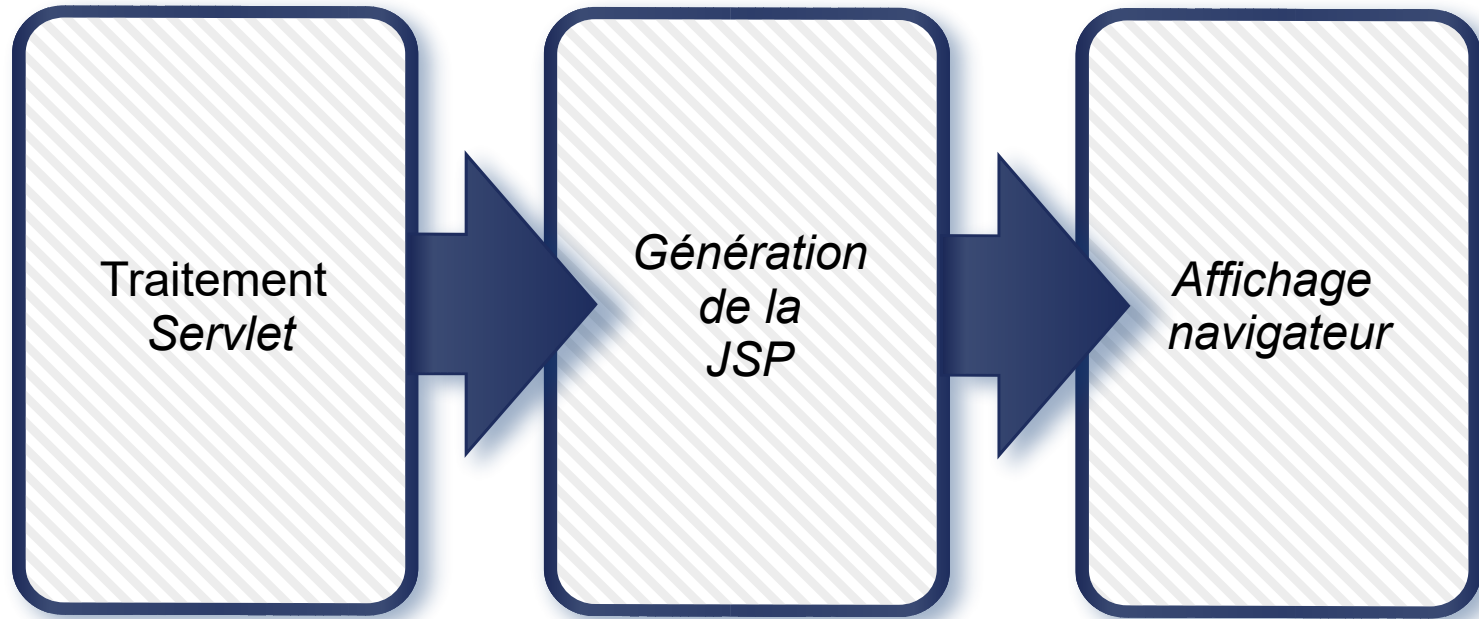
- Seul l'url d'appel change



Au moment de la génération de cette page, il est 22:03:20 le 05/12/2014



# Structure de l'affichage



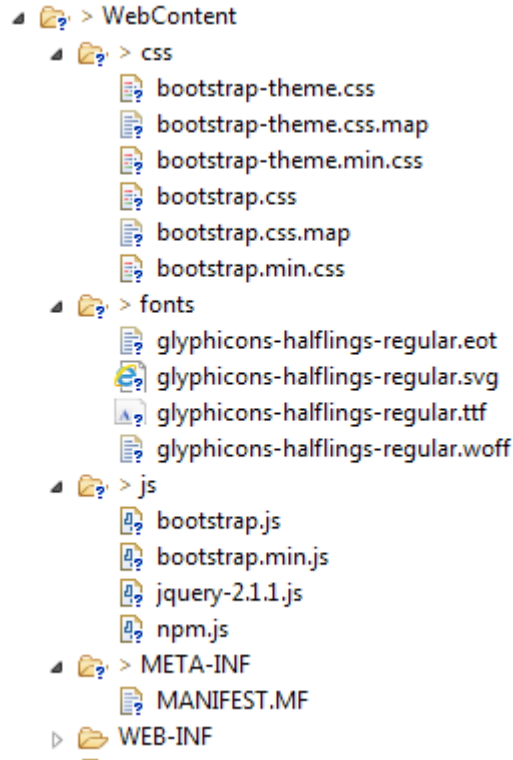
# Et le style ?

- La présentation est une combinaison HTML/CSS pour la forme
- Javascript pour le dynamisme
- Frameworks de présentation :
  - Bootstrap Twitter
  - ExtJs
  - JQuery UI
  - ...

# Améliorons la présentation

- Bootstrap twitter
- Téléchargeons Bootstrap Twitter v3.3.6
  - <http://getbootstrap.com/getting-started/>
- Téléchargeons JQuery dont dépend Bootstrap (2.2.0)
  - <http://jquery.com/>
- Vous pouvez également utiliser bower pour faire votre installation (dépend de node.js)
  - <http://bower.io/>, <http://nodejs.org/>

# Notre workspace eclipse



- Organisation standard d'un projet
- Création de deux répertoires
  - **CSS** : ou nous mettrons toutes nos feuilles de styles
  - **JS** : le répertoire de stockage des javascript
- Fichiers en double
  - Une version lisible pour le dev et debug
  - Une version min compressée pour la production

# Importons les css/js dans heureCourante.jsp

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Une page affichant l'heure courante</title>

<script type="text/javascript"
    src="{pageContext.request.contextPath}/js/jquery-2.2.0.js"></script>
<script type="text/javascript"
    src="{pageContext.request.contextPath}/js/bootstrap.min.js"></script>
<link rel="stylesheet" type="text/css"
    href="{pageContext.request.contextPath}/css/bootstrap-theme.min.css">
<link rel="stylesheet" type="text/css"
    href="{pageContext.request.contextPath}/css/bootstrap.min.css">
</head>
```

# Les points à souligner !

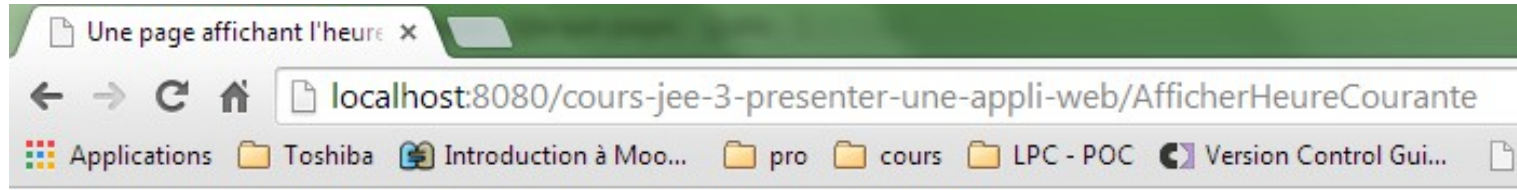
- `${pageContext.request.contextPath}`
  - Instruction en expression language (EL)
  - Interprété coté serveur
    - Contient le contexte applicatif (cours-jee-3-presenter-une-appli-web)
  - Permet d'accéder à des variables plus élégamment qu'avec le `<%= %>`
- `<script type="text/javascript" src="JSPATH"></script>`
  - Import du Javascript
- `<link rel="stylesheet" type="text/css" href="CSSPATH">`
  - Import du CSS

# Mettons à jour le body avec une EL

```
<p class="lead">Cette page a été générée à $  
{currentDateAsString}</p>
```

- Permet de retrouver la date en remplaçant l'appel à `request.getAttribute()` par une instruction plus simple

# Nouvel aspect !



Au moment de la génération de cette page, il est 22:30:11 le 05/12/2014



# Quelques points à savoir sur les EL

- Quand on interroge une classe
- EL permet d'interroger une map
  - `${map.key}` et l'équivalent de `map.get(key)`
- Il permet également d'interroger des propriétés de bean
  - Exemple un Bean x avec une méthode `getTitre()` devient avec EL `${x.titre}`
  - Idem avec « is » par exemple `isTitre()` devient `${x.titre}`.
  - Attention cela peut générer des bugs si les deux méthodes `getTitre` et `isTitre` existent dans le même objet

# Qu'allons nous faire ?

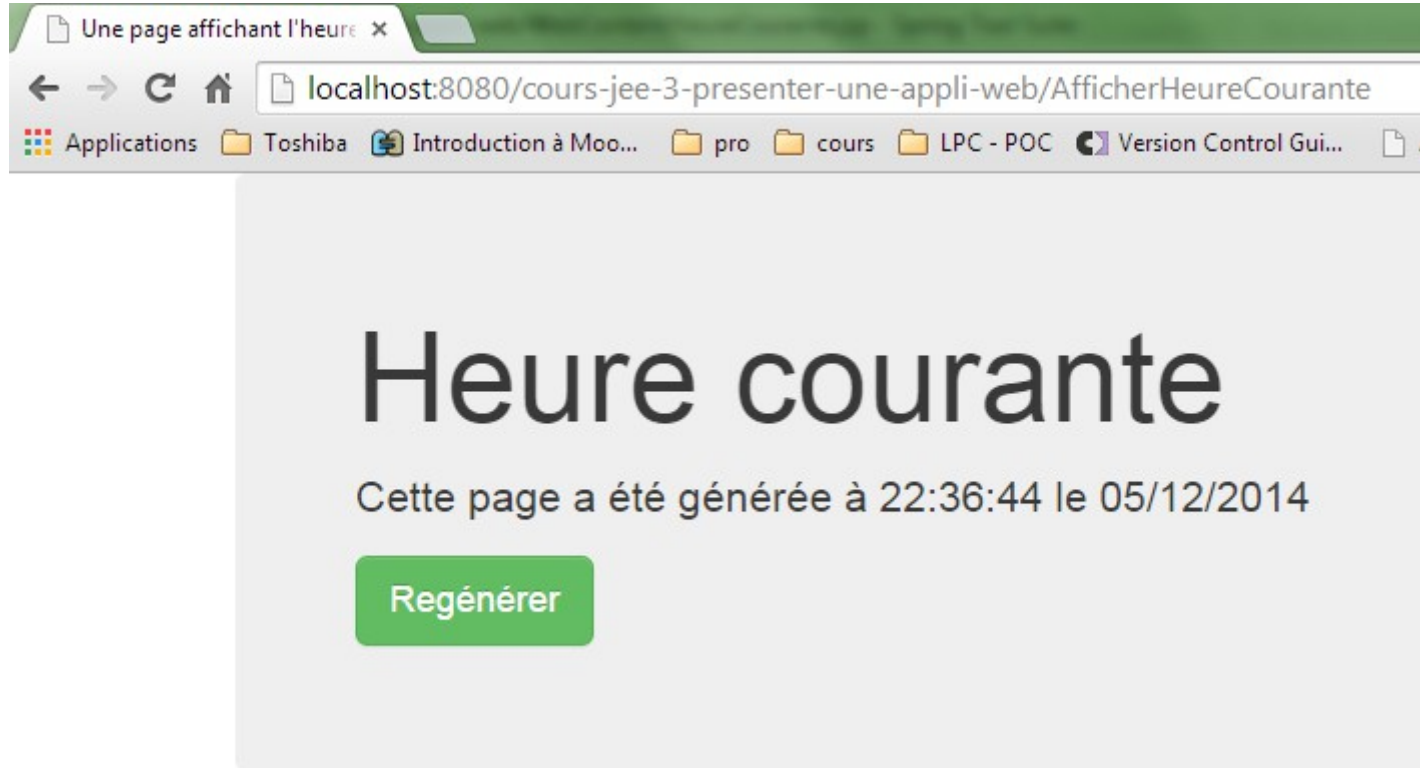
- Améliorons ce texte
- Mettons le en avant !
- Ajoutons un bouton pour rafraichir la page !
  - Utilisons un hyper lien pour le faire
    - La balise `<a>` est la base pour permettre de passer d'une page à une autre
    - Historiquement c'est la base de la révolution internet : les articles étaient liées !
    - <http://fr.wikipedia.org/wiki/Hypertexte>

# Affichons l'heure dans un Jumbotron

- Un composant de bootstrap pour mettre en avant une info !

```
<body>
  <div class="container">
    <div class="jumbotron">
      <h1>Heure courante</h1>
      <p class="lead">Cette page a été générée à ${currentDateAsString}</p>
      <p>
        <a role="button"
          href="${pageContext.request.contextPath}/AfficherHeureCourante"
          _class="btn btn-lg btn-success">Regénérer</a>
      </p>
    </div>
  </div>
</body>
```

# Le résultat à l'écran

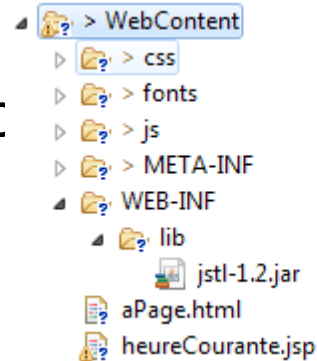


# Un dernière notion utile

- La JSTL
- La Java Server Tag Library
- Permet d'exécuter des instructions sous forme XML
- Permet notamment de faire des boucles et des conditions de façon élégantes
- Mettons ça en place pour notre exemple « heure courante »

# Importer la JSTL sous tomcat

- La JSTL fait l'objet d'une JSR mais n'est pas fournit directement avec le container applicatif
- Téléchargeons la librairie ici :
  - <http://central.maven.org/maven2/javax/servlet/jstl/1.2/jstl-1.2.jar>
- Vous pouvez également la prendre dans l'exemple du cours
- Copier le fichier dans WEB-INF/lib



# La JSTL

- Comment l'importer ?

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

- Il existe plusieurs namespaces mais les autres correspondent à des opérations complexes (opérations sur des chaînes par exemple) qu'il est préférable de réaliser dans des fonctions Java

# Cachons le bouton de rafraichissement

- Nous allons voir comment fonctionne les conditions
- Ajoutons un paramètre showButton sur AfficherHeureCourante
  - Il est transmis en GET soit AfficherHeureCourante?showButton=true
  - Quand il est à true ou absent nous afficherons le bouton
  - Lorsqu'il est à false nous le masquerons



# Le tag c:if

- Utilisons le tag c:if pour signifier qui contient un test. Ce test prend la forme d'EL
- Dans heureCourante.jsp :

```
<c:if test="$ {param.showButton ne false}">
  <p>
    <a role="button" href="$
{pageContext.request.contextPath}/AfficherHeureCourante"
class="btn btn-lg btn-success">Regénérer</a>
  </p>
</c:if>
```

# Décryptons

- C:if test=« » permet de définir l'expression à évaluer  
*`\${param.showButton ne false}`* littéralement  
param.showButton not equals to false
- Dans une EL il est possible d'utiliser des opérateurs que vous pouvez voir ici <http://docs.oracle.com/javaee/6/tutorial/doc/bnaik.html>
- Exemples : *==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le*

# Pour faire un else

- Limitation du tag if
- Le tag if ne permet pas de faire un else
- Pour pouvoir faire un else il faut utiliser le tag c:choose
- Reprenons notre exemple précédent et ajoutons un message lorsque le bouton est caché pour le préciser

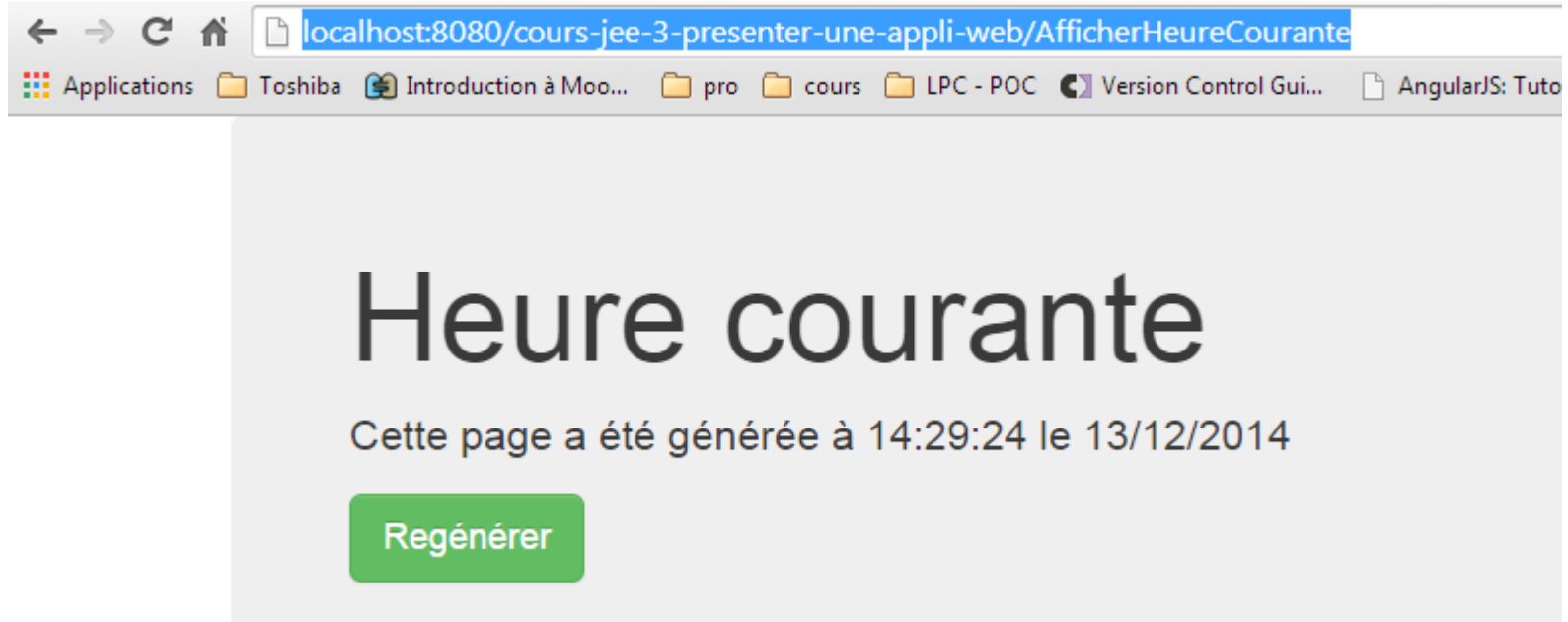
# Notre nouveau code

```
<c:choose>
  <c:when test="{param.showButton ne false}">
    <p>
      <a role="button"
        href="{pageContext.request.contextPath}/AfficherHeureCourante"
        class="btn btn-lg btn-success">Regénérer</a>
    </p>
  </c:when>
  <c:otherwise>
    Le bouton de rafraichissement est caché
  </c:otherwise>
</c:choose>
```

# Le résultat à l'écran avec showButton « false »



# Le resultat à l'écran sans showButton



# D'autres tags intéressants

```
<c:forEach items="{collection}" var="value_">${value}</c:forEach>  
<c:import url=""></c:import>
```

- La boucle foreach remplaçant du for
- Le tag c:import pour importer du contenu

# Récapitulons

- Les points importants de la génération d'une IHM
- Les serveurs JEE permettent
  - De distribuer des pages statiques
    - Pas de traitement coté serveur
    - Exemple aPage.html
  - Distribuer des pages dynamiques (*JSP*)
    - Intégrer du contenu « Java » dans les pages
    - Exemple : heureCourante.jsp
    - C'est en réalité une Servlet d'un type particulier



# Récapitulons

- Les points importants de la génération d'une IHM
- Servlet et JSP ont des rôles distincts
  - La servlet concentre les traitements
  - La JSP effectue l'affichage
  - Les méthodes forward() et sendRedirect() permettent de gérer l'enchaînement entre les deux
- EL est un langage élégant
  - Permet d'accéder à des informations du contexte Web
  - `${}`

# Récapitulons

- Les points importants de la génération d'une IHM
- La présentation peut-être améliorée par des framework
  - Léger à mettre en place
  - Efficace pour créer une belle IHM
  - Exemple twitter bootstrap
- Les tags JSTL permettent de faire des opérations simples dans les JSP tout en étant mieux intégrés dans la JSP
  - Bonne pratique : ne jamais mettre de code Java dans une JSP