Développement Web Java coté serveur : JSP / Servlet / JSTL

Master TIIL & ILIADE 1ère année

Eric Cariou

Université de Bretagne Occidentale
UFR Sciences et Techniques – Département Informatique

Eric.Cariou@univ-brest.fr

Technologies Web coté serveur

- Présentation
  - Affichage document HTML + feuilles de styles
  - S'exécute dans un navigateur web : client léger
  - Interaction avec l'utilisateur via des formulaires (listes, entrées texte, boutons ...) et liens
- Applicatif
  - But : produire un contenu HTML
    - Selon les choix de l'utilisateur
    - Selon les données à afficher
  - Nécessite une génération dynamique des pages HTML
    - Nécessite d'exécuter du code qui va générer ces pages

Technologies Web dynamique

- Exécution de code : deux possibilités
  - Exécution coté serveur
    - Du code est exécuté coté serveur pour générer de manière dynamique la page HTML qui sera envoyée au client
    - On va s'intéresser à ce type d'exécution dans ce cours
  - Exécution coté client
    - Anciennes méthodes
      - Le client télécharge à partir du serveur du code qui s'exécute dans le navigateur du client
      - Exemple : applet Java, programme Java téléchargé puis exécuté dans le navigateur
      - Problèmes de sécurité car on exécute localement du code venant d'un élément distant
    - Nouvelles méthodes : code JavaScript exécuté coté client
      - Code récupéré avec la page Web, utilisation de bibliothèques (JQuery...)
      - Code qui peut modifier dynamiquement le contenu de la page courante : voir cours sur AJAX / WebSocket

Génération pages dynamiques

- Deux principes généraux de fonctionnement, coté serveur
  - Langages de scripts
    - Double type de contenu dans une page HTML
      - Partie statique : balises HTML standards avec contenu textuel
      - Partie dynamique : du code écrit dans un langage de script
        - Ce code génère du code HTML standard
    - La page entremêle les parties statiques et dynamiques
    - Quand un navigateur demande le contenu d'une page
      - La partie dynamique est exécutée et remplacée par le HTML généré
      - Le navigateur du client reçoit donc uniquement du code HTML
  - Exécution d'un programme
    - Un programme complet s'exécute et génère du contenu HTML
    - La page est entièrement dynamique
      - Plus précisément : les partie statiques existent mais elles sont intégrées dans le code, pas écrites directement en HTML standard
    - Le navigateur demande l'exécution d'un programme et récupère le code HTML généré
      - La demande d'exécution est transparente : utilise URL standard

Génération pages dynamiques

- Exemples de technologies
  - Langages de script
    - PHP
      - Langage interprété offrant l'avantage d'intégrer nativement les primitives de requêtes SQL sur des BDD
    - ASP et ASP .Net
      - Solutions Microsoft
    - JSP (Java Server Page)
      - Solution Oracle/Jakarta pour Java
  - Exécution de programmes
    - CGI : Common Gateway Interface
      - Historiquement, une des premières solutions
    - Servlet
      - Solution Oracle pour Java : exécution de code Java respectant certaines caractéristiques

Génération pages dynamiques

- Plateforme d'exécution
  - Dans tous les cas, il faut pouvoir exécuter du code et communiquer via HTTP avec le navigateur coté client
  - Deux composants pour génération de pages dynamiques
    - Serveur HTTP standard
    - Élément (conteneur) d'exécution des programmes ou des scripts
- Exemples de serveurs / plateformes
  - Apache Tomcat, Glassfish
    - JSP, Servlet
    - Logiciel libre
  - Microsoft IIS
    - ASP, ASP .Net
    - Propriétaire

Servlet

- Une servlet est un programme Java particulier
  - Classe qui hérite de HttpServlet
    - Equivalent d'une applet Java, mais coté serveur
  - Classe standard du point de vue de son contenu en terme d'attributs, méthodes ...
  - Mais exécution et interactions différentes d'un objet Java standard
    - Pas de main()
    - A la création de la servlet par le serveur d'exécution
      - Appel par le serveur d'exécution de la fonction init(ServletConfig)
      - On peut y faire les actions qu'on ferait dans un constructeur
    - Quand la servlet est détruite
      - Appel par le serveur d'exécution de destroy()
      - A redéfinir si on veut effectuer certaines actions à la suppression de la servlet

Servlet

- Navigateur web coté client
  - Envoie des requêtes HTTP au serveur
    - GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE
- Dans une servlet
  - A chaque requête correspond une méthode doXXX(...) héritée de la classe HttpServlet et qui sera appelée selon la requête client
    - On redéfinit ces méthodes pour y placer le code à exécuter par la servlet
    - En pratique, pas besoin de redéfinir toutes les méthodes (GET et POST minimum)
    - De manière plus générale (mais moins recommandée), la méthode service(...) reçoit toutes les requêtes et peut être redéfinie directement (par défaut elle redispatche aux doXXX(...))
  - Exemple pour GET
    - doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    - request : contient la description de la requête du client
    - response : à utiliser pour envoyer la réponse au client
    - Toutes les méthodes doXXX(...) ont la même signature

Paramètres des doXXX(...)

- HttpServletRequest request
  - Initialisé par l'appel du client (le navigateur)
  - Données / informations sur la requête envoyée par le client
  - On peut y récupérer notamment
    - Valeurs entrées pour un formulaire
    - Une session pour gérer un état dédié à chaque client
    - Cookies du client envoyés avec la requête
    - Informations sur l'URL utilisée pour l'appel de la servlet
    - Login de l'utilisateur s'il s'est identifié
    - Et de manière plus générale, tous les « headers » de l'appel
      - Identifiants du navigateur, type de requête (GET, PUT ...), type d'encodage supporté par le navigateur, ...

Paramètres des doXXX(...)

- HttpServletResponse response
  - A initialiser et utiliser par la servlet pour générer le résultat à envoyer au navigateur client
  - Définir le type MIME des données envoyées
    - Généralemet du code HTML
      - response.setContentType("text/html;charset=UTF-8");
    - Mais peut utiliser n'importe quel type MIME : image/gif, audio/mp3, application/pdf, ...
  - Récupérer le flux de sortie pour envoyer les données
    - PrintWriter getWriter()
      - Flux texte, typiquement pour du HTML
    - ServletOutputStream getOutputStream
      - Flux binaire pour des images, vidéos ...
  - Ajouter/envoyer des données coté client
    - Créé un nouveau cookie (ou modifier un existant)
    - Ajouter un header aux données envoyées

Caractéristiques d'une servlet

- Unicité d'une servlet
  - Une servlet n'est créée qu'en une seule instance
  - Si plusieurs clients accèdent à l'URL d'une même servlet
    - Appelle les méthodes sur cette instance unique
  - Une servlet possède des attributs donc un état
    - Cet état est permanent et conservé (tant que la servlet existe)
    - Il est accédé / modifié par tous les clients : état global
- Etat associé à un client
  - Temporaire : mode session
    - Pour chaque client, une session est créée
      - On peut lui associer des données via des couples « clé (chaine) / objet »
    - La session a une durée de vie configurable
    - Exemple typique d'utilisation d'une session
      - Panier d'un utilisateur sur un site de vente en ligne : conserve les produits choisis par l'utilisateur pendant son parcours sur le site
  - Permanent : cookies du navigateur

Exemple : servlet hello world

- Exemple basique de servlet
  - Génère une page qui affiche « hello word »
  - Gère et affiche un compteur qui est incrémenté à chaque accès à la servlet
- Note sur le code présenté
  - Le code qui suit a été créé sous Netbeans qui génère un squelette de code
    - Ce squelette contient une méthode principale processRequest() qui contiendra le code principal de la servlet
    - Les méthodes héritées doGet() et doPost() appellent directement cette méthode processRequest() avec les mêmes paramètres
- Une fois déployée sur le serveur, on accède à une servlet via une URL HTTP
  - La notre est contenue dans le fichier HelloWorld.java
  - URL sera de la forme : http://www.xxxxxx.fr/XXX/HelloWorld
    - Note : peut choisir une URL d'accès à la servlet différente du nom de la servlet Java

Exemple : servlet hello world

- Code de la servlet HelloWorld.java

```
public class HelloWorld extends HttpServlet {

    // compteur géré par la servlet
    protected int compteur = 0;

    protected void processRequest(
       HttpServletRequest request, HttpServletResponse response)
                                    throws ServletException, IOException {
       // out : flux de sortie mode texte qui contiendra le HTML
       response.setContentType("text/html;charset=UTF-8");
       PrintWriter out = response.getWriter();
       try {

         // on écrit ligne par ligne le code HTML à afficher chez le client
         out.println("<html>");
         out.println("<head>");
         out.println("<title>Servlet HelloWorld</title>");
         out.println("</head>");
         out.println("<body>");
         out.println("<h1>Servlet HelloWorld</h1>");
         // on insère ici la valeur du compteur (en l'incrémentant au passage)
         out.println("<p>Nombre d'appels : "+(++compteur)+"</p>");
         out.println("</body>");
         out.println("</html>");
       } finally {
           out.close();
    } }
```


Exemple : servlet hello world

- Exemple d'exécution

Après 4 chargements de la servlet
⇒ compteur passe à 5

Exemple : servlet hello world

- Code précédent
  - Le compteur est global : commun à tous les clients
- Variante pour gérer un compteur local à chaque client
  - Passer par les données qu'on peut associer à une session
  - Modification de processRequest()

```
try {
  // récupère la session associée au client
  HttpSession session = request.getSession(true);

  // récupère le compteur associé à la session
  Integer compteur = (Integer)session.getAttribute("compteur");
  // si valeur null : attribut « compteur » n'existe pas, ça signifie
  // que c'est la première exécution par le client, il faut créer le compteur
  if (compteur==null) {
    compteur = new Integer(1);
    session.setAttribute("compteur", compteur);
    // configure pour qu'une session dure 10 secondes max
    session.setMaxInactiveInterval(10);
  }
  (...)
  out.println("<p>Nombre d'appels : "+compteur+"</p>");
  (...)
  // incrémente le compteur
  session.setAttribute("compteur", new Integer(compteur.intValue()+1));
```


Exemple : passage de paramètres

- Autre exemple
  - Manipulation de rectangles avec 3 opérations
    - Création d'un rectangle
    - Décalage du rectangle courant (non présenté en détail pour gagner de la place)
    - Calcul de la surface du rectangle courant
- Implémentation
  - Une page HTML contient un formulaire pour entrer les paramètres et appeler la servlet avec ces paramètres
    - Note : dans l'exemple, c'est une page JSP mais c'est du HTML standard, aucun code Java n'est inclus dans la page
  - Servlet
    - Une servlet gère le rectangle et exécute les 3 opérations en fonction des paramètres venant du formulaire
    - On récupère ces paramètres via : request.getParameter(nomParam)

Exemple : servlet rectangle

- Code de la page HTML

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>JSP Page</title>
</head>
<body>
  <h1>Manipulation de rectangle avec une servlet</h1>

  <form action="RectServlet" method="post">
  <p> <table border="0">
      <tr> <td>X1 ou X décalage</td> <td><input name="x1"> </td> </tr>
      <tr> <td>Y1 ou Y décalage</td> <td><input name="y1"> </td> </tr>
      <tr> <td align="right">X2</td> <td><input name="x2"> </td> </tr>
      <tr> <td align="right">Y2</td> <td><input name="y2"> </td> </tr>
  </table></p>

  <p><input type="radio" name="action" value="creer" checked="checked" />
  Créer Rectangle<br />
  <input type="radio" name="action" value="decaler" />Décaler Rectangle<br />
  <input type="radio" name="action" value="surface" />Calcul Surface</p>

  <p><input type="submit" value="Exécuter">
  <input type="reset" value="Remise à zéro></p>
  </form>
</body>
</html>
```


Exemple : servlet rectangle

- Liens entre page HTML et la servlet
  - Le formulaire de la page HTML contient
    - Une action associée qui est l'appel de la servlet RectServlet
    - 4 champs d'entrée de noms x1, y1, x2, y2
    - Une liste de boutons radio dont selon le choix, on aura le paramètre action égal à « creer », « decaler » ou « surface »
  - Dans le code de la servlet RectServlet.java
    - On récupére ces 5 paramètres dans request
    - Notamment le paramètre action qui précisera quelle opération on veut appeler
    - La servlet conservera le dernier rectangle créé (avec un premier rectangle défini par défaut) et les fonctions de décalage et de surface seront appelées sur ce rectangle
    - Pour simplifier, on met les fonctions métiers (manipulation de rectangles) directement dans le code de la classe de la Servlet

Exemple : servlet rectangle

- Code de la servlet ServRectangle.java

```
public class RectServlet extends HttpServlet {

   // rectange manipulé par la servlet
   protected Rectangle rectangle;

   public int calculSurface(Rectangle rect) {
       return ( (rect.x2 - rect.x1) * (rect.y2 - rect.y1));
   }

   public Rectangle decalerRectangle(Rectangle rect, int x, y){
      return new Rectangle(rect.x1 + x, rect.y1 + y,
                           rect.x2 + x, rect.y2 + y);
   }

   public void init(ServletConfig config)
                                    throws ServletException {
      // initialisation par défaut du rectangle
      rectangle = new Rectangle(10, 20, 30, 40);
   }
```


Exemple : servlet rectangle

- Code de la servlet ServRectangle.java (suite)

```
protected void processRequest(
   HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException {

  response.setContentType("text/html;charset=UTF-8");
  PrintWriter out = response.getWriter();
  try {
    // entête du document
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Test de Servlet : RectServlet</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Servlet RectServlet</h1>");

    // affichage du rectangle courant
    out.println("<p>Etat courant de mon rectangle:"+rectangle+"</p>");

    // récupère l'identifiant de l'action à réaliser
    String action = request.getParameter("action");

    // calcul de surface du rectangle courant
    if (action.equals("surface"))
       int surface = calculSurface(rectangle);
       out.println("<p>Surface : "+surface+"</p>");
```

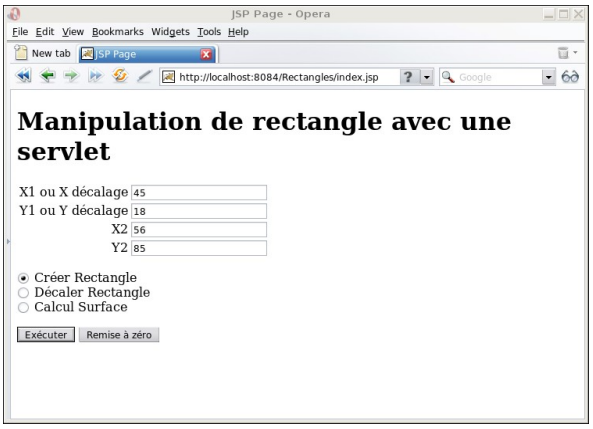

Exemple : servlet rectangle

- Code de la servlet ServRectangle.java (fin)

```
// création d'un rectangle
if (action.equals("creer")) {
  int x1 = Integer.parseInt(request.getParameter("x1"));
  int y1 = Integer.parseInt(request.getParameter("y1"));
  int x2 = Integer.parseInt(request.getParameter("x2"));
  int y2 = Integer.parseInt(request.getParameter("y2"));

  rectangle = new Rectangle(x1, y1, x2, y2);

  out.println("<p>Rectangle créé : "+ rectangle + "</p>");
}
catch (Exception e) {
  // en cas de problème, affiche un message
  out.println("<p><b>Erreur !!</b><br />");
  out.println(e.toString()+"</p>");
}

// fin du document : lien pour retour en arrière
out.println("<p><a href=\""+request.getContextPath()+"\">
                                                  Retour</a></p>");
out.println("</body>");
out.println("</html>");
out.close();
}
```

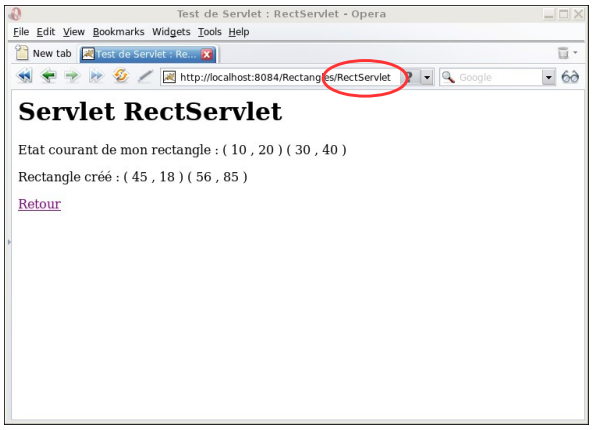

Exemple : servlet rectangle

- Affichage de la page HTML
  - On remplit les 4 champs et sélectionne « créer »



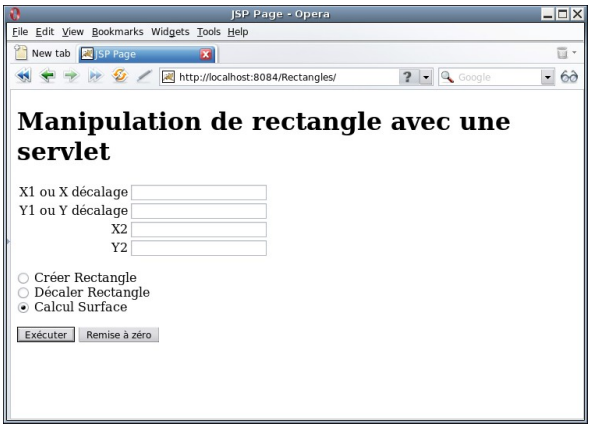
Exemple : servlet rectangle

- Après clic sur « exécuter »
  - Se retrouve « sur » la servlet qui crée alors un rectangle



Exemple : servlet rectangle

- Sélection de retour pour revenir à la page HTML puis choix de « calcul surface »



Exemple : servlet rectangle

- Après clic sur exécuter
  - Exécute la servlet avec l'opération de calcul surface
  - On voit qu'elle s'applique sur le rectangle qu'on vient de créer et qui a donc été conservé

JSP : insertion de code Java

- Attributs implicites (non déclarés) utilisables dans le code Java
  - Similaires à ce qu'on aurait dans une servlet
    - out : flux de sortie texte dans lequel on écrit le code HTML généré par les parties de code Java
    - request : paramètres de l'appel de la JSP / Servlet
    - response : pour renvoyer la réponse au client
    - session : la session spécifique au client courant
    - config : configuration de la JSP
  - Spécifique au fonctionnement des JSP
    - page : l'instance de la JSP / Servlet (équivalent du this)
    - exception : l'exception qui a été levée en cas de problème
    - application : données communes à toutes les JSP du serveur
    - pageContext : contexte de la page (gestion d'attributs ...)

JSP : insertion de code Java

- Balise spéciale <%@ page .... %>
  - Informations générales sur la JSP
    - A utiliser si les valeurs par défaut de la page sont à modifier
  - Import de classes Java
    - <%@ page import = java.util.Vector %>
  - Type MIME de la page
    - Par défaut du text/html
    - <%@ page contentType = "image/png" %>
  - Gestion des erreurs
    - Exécuter une JSP en cas d'erreur rencontrée
      - <%@ page errorPage="erreur.jsp" %>
    - Préciser qu'une JSP est ou pas une page d'erreur
      - <%@ page isErrorPage="true" %>

Java Server Page : JSP

- Pages mixtes contenant à la fois
  - Du code HTML statique
  - Du code Java qui est exécuté dynamiquement et génère du code HTML
  - Les parties statiques et Java sont entremêlées pour former une page JSP
  - Le navigateur client récupère une page HTML standard contenant
    - Les parties statiques
    - Les parties en Java remplacées par le code HTML qu'elles ont générées
  - En pratique
    - Le serveur d'exécution de la JSP la traduit d'abord en une servlet complète équivalente et c'est cette servlet qui est exécutée

JSP : insertion de code Java

- Balises spéciales pour insérer du code Java
  - Trois balises <% ... %>, <%! ... %> et <%= ... %>
  - Peut insérer plusieurs de ces balises dans une même page JSP
- Point important
  - JSP compilée en Servlet donc unicité de la JSP également
- <%! ... %>
  - Déclaration d'attributs et de méthodes
  - Ils seront globaux à tous les appels de la JSP pour tous les clients
- <% ... %>
  - Scriplet : suite d'instructions Java qui sera exécutée à chaque appel de la page JSP
  - Si on y déclare des attributs, ils sont locaux à chaque appel
- <%= expr %>
  - Evalue le expr et affiche le résultat dans le code HTML
  - Equivalent de <% out.print(expr) %>

Exemple : JSP hello world

- Exemple basique de page JSP
  - Code statique HTML : affiche « Hello World ! »
  - Code dynamique et code Java
    - Gestion d'un compteur local et d'un compteur global d'appel
    - Affichage de ces 2 compteurs (via mélange code statique et dynamique)
- Implémentation
  - Trois insertions de code Java pour gérer les compteurs
    1. <% ...%> : déclaration du compteur local
    2. <%! ... %> : déclaration du compteur global et d'une méthode d'incrémentation de ce compteur
    3. <% ... %> : incrémentation des 2 compteurs
  - Puis affiche via des <%= ... %> les valeurs des compteurs

Exemple : JSP hello world

- Code de la page JSP

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

  <body>
    <h1>Hello World!</h1>

    <%
      int nbLocal = 0;
    %>

    <%!
      int nbGlobal = 0;
      void incNbGlobal() { nbGlobal++; }
    %>

    <%
      nbLocal++;
      incNbGlobal();
    %>
     <p>Compteur local : <%= nbLocal %> <br />
     Compteur global : <%= nbGlobal %></p>
  </body>
</html>
```

Légende couleur :

- *Code exécuté localement*
- *Code déclaré globalement*
- *Evaluation de la valeur des attributs*
- *Code HTML statique*

Exemple : JSP hello world

- Exemple d'exécution

Après 4 chargements de la servlet
⇒ compteur global passe à 5
mais le local est systématiquement à 1

Exemple : JSP hello world

- Commentaires sur le code présenté
  - Dans dernière balise <% ... %>
    - Aurait pu remplacer « incGlobalNb(); » par directement « globalNb++; »
    - Les attributs et méthodes déclarées en global sont tous accessibles directement
  - Ne peut par contre pas déclarer une méthode pour incrémenter le compteur local
    - void incNbLocal { nbLocal++; }
    - Ne peut pas déclarer de méthode dans une balise <%! ... %>
      - Ne contient qu'une suite d'instructions à exécuter
    - Ne peut pas déclarer cette méthode dans la balise <% ... %>
      - L'attribut nbLocal est local et n'est pas connu dans le code global

Exemple : JSP rectangle

- Même exemple de gestion des rectangles
  - On a besoin d'une seule page JSP au total
  - Elle fera l'affichage du formulaire avec HTML standard et contiendra le code Java des opérations (en vert)
  - Elle s'auto-appellera pour l'exécution de l'opération choisie
- Code la page rectangle.jsp

```
<head>
    <meta http-equiv="Content-Type"
                                content="text/html; charset=UTF-8">
    <title>Rectangle</title>
</head>
<body>
    <%@page import="rect.*" %>
    <h1>Manipulation de rectanges avec du JSP</h1>
```

la classe Rectangle est définie dans un package rect

Exemple : JSP rectangle

- Code la page rectangle.jsp (suite)

```
<%!
   Rectangle rectangle = new Rectangle(10, 20, 30, 40);

   public int calculSurface(Rectangle rect) {
      return ( (rect.x2 - rect.x1) * (rect.y2 - rect.y1));
   }

   public Rectangle decalerRectangle(Rectangle rect, int x, int y){
       return new Rectangle(rect.x1 + x, rect.y1 + y,
                                    rect.x2 + x, rect.y2 + y);
   }
%>

<form action="rectangle.jsp" method="post">
<p> <table border="0">
    <tr> <td>X1 ou X décalage</td> <td><input name="x1"> </td> </tr>
    <tr> <td>Y1 ou Y décalage</td> <td><input name="y1"> </td> </tr>
    <tr> <td align="right">X2</td> <td><input name="x2"> </td> </tr>
    <tr> <td align="right">Y2</td> <td><input name="y2"> </td> </tr>
</table></p>

<p><input type="radio" name="action" value="creer" checked="checked" />
Créer Rectangle<br />
<input type="radio" name="action" value="decaler" />Décaler Rectangle<br />
<input type="radio" name="action" value="surface" />Calcul Surface</p>
```


Exemple : JSP rectangle

- Code la page rectangle.jsp (suite)

```
<p><input type="submit" value="Exécuter">
<input type="reset" value="Remise à zéro"></p>
</form>

<p>Rectangle courant : <%= rectangle.toString() %> </p>

<%
try {
    String action = request.getParameter("action");

    // cas du premier affichage de la page : pas de paramètre
    // il n'y a pas d'actions à exécuter
    if (action == null) return;

    // création d'un rectangle
    if (action.equals("creer")) {
       int x1 = Integer.parseInt(request.getParameter("x1"));
       int y1 = Integer.parseInt(request.getParameter("y1"));
       int x2 = Integer.parseInt(request.getParameter("x2"));
       int y2 = Integer.parseInt(request.getParameter("y2"));

      rectangle = new Rectangle(x1, y1, x2, y2);
      out.println("<p>Rectangle créé : "+rectangle + "</p>");
  }
```

Exemple : JSP rectangle

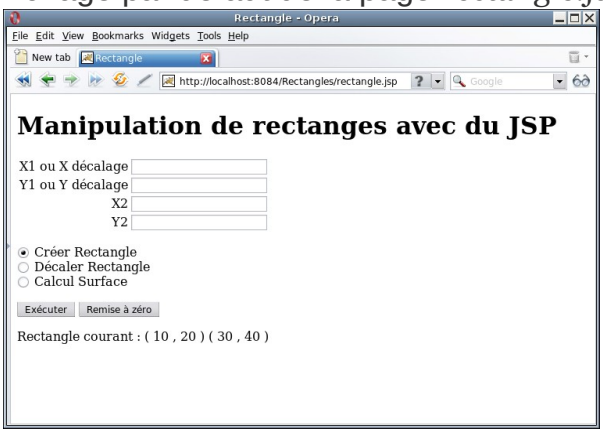
- Code la page rectangle.jsp (suite)

```
      // calcul de surface du rectangle courant
      if (action.equals("surface")) {
         int surface = calculSurface(rectangle);
         out.println("<p>Surface : "+surface+"</p>");
      }

   } // try
   catch (Exception e) {
     out.println("<p><b>Erreur !!</b><br />");
     out.println(e.toString()+"</p>");
   }
%>
</body>
</html>
```

- On utilise ici les attributs implicites
  - request : pour récupérer les paramètres passés avec getParameter()
  - out : flux de sortie pour générer du code HTML
- Deux façons de générer du code HTML en dynamique
  - Utilisation du out comme avec une servlet
  - Utilisation d'une balise <%= ... %> dans du code HTML

Exemple : JSP rectangle

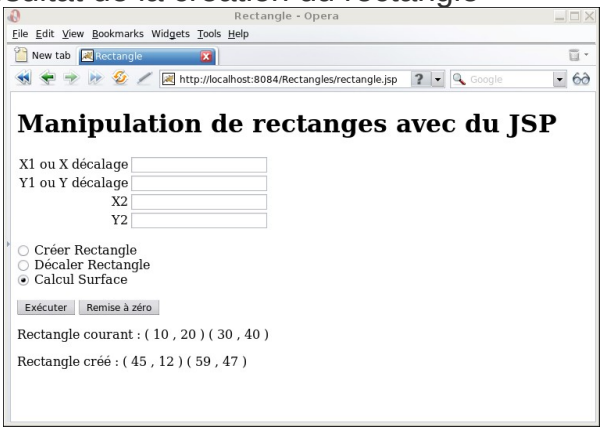
- Affichage par défaut de la page rectangle.jsp



- On va remplir les 4 champs avec valeurs 45, 12, 59, 47 et lancer l'exécution de « créer rectangle »

Exemple : JSP rectangle

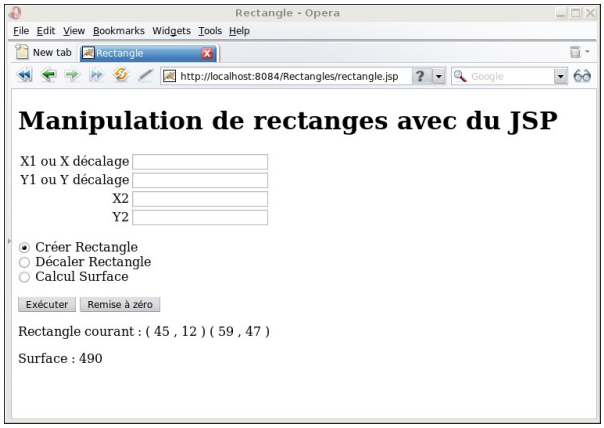
- Résultat de la création du rectangle



- On exécute maintenant à partir de cette page « calcul surface »

Exemple : JSP rectangle

- Résultat du calcul de surface



Exécution concurrente

- Modèle par défaut d'exécution
  - Servlet / page JSP est unique
  - Serveur HTTP peut avoir plusieurs requêtes d'accès à la servlet/ JSP par plusieurs clients à la fois
    - Exécution multi-threadée par le serveur HTTP
    - Plusieurs threads peuvent donc exécuter en même temps du code de la même servlet/JSP
- S'assurer qu'un seul thread à la fois exécute une servlet/JSP
  - Servlet
    - Implémentation de l'interface (vide) SingleThreadModel
    - public class MaServlet extends HttpServlet implements SingleThreadModel
  - JSP
    - Passe par les paramètres généraux de la page : isThreadSafe
    - <%@ page isThreadSafe="true" %>
  - Pour une gestion plus fine et plus précise
    - Marquer des méthodes ou des parties de code de la servlet/JSP avec synchronized

Communication inter-éléments

- Chaque servlet / JSP possède un état global ou un état local pour chaque client
  - Peut aussi vouloir avoir un état global à toutes les pages JSP ou les servlets gérées par le serveur HTTP
- Attribut implicite application dans JSP
  - Permet de définir des attributs comme pour une session, mais communs à toutes les pages JSP gérées par le serveur HTTP
  - Exemple : code Java pour gérer un compteur commun à toutes les pages JSP

```
<%
    Integer compteur = (Integer)application.getAttribute("compteur");
    if (compteur==null) {
       // on crée le compteur qui n'existait pas
       compteur = new Integer(0);
       application.setAttribute("compteur", compteur);
       out.println("<p>Création du compteur ...</p>");
    }
    application.setAttribute("compteur", ++compteur);
    out.println("<p>Compteur global  : "+compteur+"</p>");
%>
```

Communication inter-éléments

- Etat commun global au serveur HTTP
  - Autre technique : définir une classe accessible (dans même package) par toutes les servlets / JSP
    - Y définir des attributs ou méthodes statiques
    - Exemple pour le compteur
      - public class Compteur {

```
protected static int compteur = 0;

public synchronized static int nextValue() {
   return ++compteur;
  }
}
```

    - On accède par exemple au compteur global dans une servlet ou une page JSP
      - out.println(" Nombre d'appels : "+Compteur.nextValue()+" ");

Relation entre servlets/JSP

- Dans une servlet ou une page JSP, possibilité
  - D'appeler une autre servlet ou page JSP pour déléguer le traitement de la requête ou inclure un résultat supplémentaire dans celui de la servlet/page JSP courante
  - Servlet
    - Récupére le « request dispatcher »
      - RequestDispatcher rd = request.getRequestDispatcher("maServlet");
    - Fait suivre le traitement de la requête ou inclus un traitement réalisé par la servlet « maServlet »
      - rd.forward(request, response);
        rd.include(request, response);
  - Page JSP
    - Inclusion du résultat d'une autre page
      - <jsp:include page="maPage.jsp"></jsp:include>
    - Faire suivre le traitement de la requête à une autre page JSP
      - <jsp:forward page="maPage.jsp"></jsp:forward>
    - Dans les 2 cas, peut passer des paramètres à la page appelée via inclusion
      - <jsp:param name="compteur '' value="12" />

Relation entre éléments

- En cas d'erreur dans une page JSP, peut aussi exécuter une page JSP particulière
  - Dans la page, insérer une redirection en cas d'erreur
    - <%@ page errorPage="erreur.jsp" %>
    - En cas d'exception levée par une instruction Java, la page erreur.jsp est alors exécutée
  - Dans erreur.jsp
    - Préciser que l'on est une page d'erreur
      - <%@ page isErrorPage="true" %>
    - Peut alors récupérer l'exception levée et par exemple l'afficher
      - Exception levée : <%= exception %>
      - L'attribut implicite exception est une exception Java classique, on peut appeler dessus des méthodes comme getMessage(), getCause(), printStackTrace(), ...

Retour sur l'exemple des rectangles

- Dans une JSP, une part importante du code est dédiée à simplement afficher le contenu d'objets
  - Même avec des <%= ... %> ça peut être relativement lourd à réaliser
- Exemple
  - Reprenons notre manipulation de rectangles
    - La servlet RectServlet gère le rectangle courant et exécute les actions associés
    - Une page HTML affiche un formulaire pour rentrer les différents champs
    - Si l'utilisateur veut modifier une des 4 valeurs de coordonnée (y1 par exemple) du rectangle courant, il doit réentrer les 4 valeurs
    - On pourrait remplir par défaut les 4 champs avec le contenu du rectangle courant

Retour sur l'exemple des rectangles

- Modifie RectServlet.java pour ajouter le rectangle en tant qu'attribut de la session
  - protected void processRequest(...) {

```
HttpSession session = request.getSession(true);

// premier appel de la servlet, on crée l'attribut rectangle
if (session.getAttribute("rectangle") == null)
      session.setAttribute("rectangle", rectangle);

if (action.equals("creer")) {
       // récupère les paramètres x1, x2, y1, y2 dans la requête
       (...)
       rectangle = new Rectangle(x1, y1, x2, y2);
       out.println("<p>Rectangle créé : "+rectangle + "</p>");
       // met à jour l'attribut de la session avec le nouveau rectangle
        session.setAttribute("rectangle", rectangle);
}
(...)
```


Retour sur l'exemple des rectangles

- Transforme la page HTML associée en page JSP pour pouvoir récupérer le rectangle associé à la session
  - <%@page import = "rect.Rectangle" %>
    <% Rectangle rect =(Rectangle) session.getAttribute("rectangle"); %>

```
<h1>Manipulation de rectangle avec une servlet</h1>
<form action="RectServlet" method="post">
<p> <table border="0">
   <tr>
      <td>X1</td>
      <td><input name="x1" value="<%=rect.getX1() %>"></td>
   </tr>
   (...)
```

  - On récupére l'attribut « rectangle » associé à la session
  - On s'en sert pour mettre une valeur par défaut dans les champs d'entrée des 4 valeurs de coordonnées

Retour sur l'exemple des rectangles

- Le code présenté semble pertinent ... mais ne marche pas
  - Au premier chargement de la JSP, aucun appel n'a encore été fait sur la Servlet : pas d'attribut « rectangle » associé à la session
    - L'attribut rect vaut donc null et <%= rect.getX1() %> lève une exception
  - Il faut gérer explicitement le cas où l'attribut n'est pas encore défini

```
<%@page import = "rect.Rectangle" %>
<% Rectangle rect = (Rectangle) session.getAttribute("rectangle");
     String x1 = "", x2= "", y1 = "", y2 = "";
     if (rect != null) {
         x1 = new Integer(rect.getX1()).toString();
         (...)
  } %>
<h1>Manipulation de rectangle avec une servlet</h1>
<form action="RectServlet" method="post">
<p> <table border="0">
   <tr>
      <td>X1</td>
      <td><input name="x1" value="<%= x1 %>"></td>
(...)
```


Retour sur l'exemple précédent

- Autre solution pour récupérer le rectangle courant
  - Expression UEL (Unified Expression Language)
  - Simplifie l'accès aux attributs (dans le sens des attributs associés à une session)
    - La page JSP devient tout simplement

```
<h1>Manipulation de rectangle avec une servlet</h1>
<form action="RectServlet" method="post">
<p> <table border="0">
  <tr>
    <td>X1</td>
    <td><input name="x1" value=" ${sessionScope.rectangle.x1}"></td>
(...)
```

    - Le champ d'entrée X1 est initialisé par le contenu de la propriété x1 de l'attribut rectangle associé à la session
    - Si l'attribut rectangle n'existe pas, ne génère pas d'erreur, retourne simplement une chaîne vide
      - Plus besoin de vérifier explicitement que l'attribut existe

UEL & JSTL

- Unified Expression Language (UEL)
  - Langage d'expression permettant simplement dans une page JSP
    - De récupérer des attributs associés à des contextes
      - Attributs de la session, de l'application, du header ou de la requête HTTP ...
    - D'écrire des expressions de calcul ou de test
      - Addition, multiplication, modulo ...
    - Comparaisons de valeurs, opérateurs logiques (OU, ET ...)
- Java Server Pages Standard Tag Library (JSTL)
  - Ensembles de balises offrant des fonctionnalités pour manipuler des fichiers XML, accès à des BDDs, l'internationalisation ...
  - Ensemble « core » permet notamment de manipuler simplement des ensembles de données, faire des tests et d'associer des balises HTML (ou autres) à ces traitements ...
- En combinant UEL et JSTL
  - Facilite grandement dans une page JSP l'affichage de données retournées par la logique métier

UEL : attributs, propriétés

- Attribut
  - Instance d'une classe Java respectant quelques contraintes structurelles : Java Beans (identique aux POJO)
    - Définit des propriétés
      - Un attribut (de la classe) avec un getter et un setter associé
    - Définit un constructeur sans paramètre
  - On peut ensuite accéder aux propriétés des attributs via une notation pointée à partir d'un certain contexte
    - Exemple : sessionScope.rectangle.x1
      - Contexte : sessionScope
      - Attribut : rectangle
      - Propriété : x1
    - Revient à exécuter de manière réflexive
      - ((Rectangle)sessionScope.getAttribute("rectangle")).getX1()
      - On voit bien l'obligation d'avoir un getter/setter associé à chaque propriété pour pouvoir y accèder
  - Si l'attribut ou une de ses propriétés n'est pas définie (valeur null) génère une chaîne vide

UEL : contextes et objets implicites

- Objets implicites, dont certains similaires aux objets implicites JSP
  - pageContext, param, paramValues, header, headerValues, cookie, initParam
  - Contextes (« scope »), à partir desquels on peut associer des attributs
    - pageScope : la page JSP
    - requestScope : la requête HTTP
    - sessionScope : la session associée avec le navigateur client
    - applicationScope : contexte global à toutes les pages JSP du serveur

UEL : tableaux et map

- Accède aux données d'un tableau (array, list ...) ou d'une map via une notation indexée
  - att[index]
    - Avec index qui est soit la clé pour une map, soit l'index pour un tableau
    - Peut aussi être le contenu d'une variable
    - Pour un tableau, peut utiliser un index sous forme d'entier ou de chaîne
- Exemple : une servlet exécute le code suivant

```
Rectangle rect1 = new Rectangle(10,10,20,20);
Rectangle rect2 = new Rectangle(30, 30, 50, 50);

HashMap<String, Rectangle> mapRect = new HashMap<String, Rectangle>();
Rectangle[] arrayRect = new Rectangle[2];
int index = 1;

mapRect.put("premier",rect1);
mapRect.put("second", rect2);
arrayRect[0] = rect1;
arrayRect[1] = rect2;

session.setAttribute("mapRect", mapRect);
session.setAttribute("arrayRect", arrayRect);
session.setAttribute("index", index);
```

UEL : tableaux et map

- La servlet forwarde la requête à la page JSP
  -
    Map -> "premier" : ${sessionScope.mapRect["premier"].x1}
    Map -> 'second' : ${sessionScope.mapRect['second'].x1}
    Map -> "troisieme" : ${sessionScope.mapRect["troisieme"].x1}
    Array -> 0 : ${sessionScope.arrayRect[0].x1}
    Array -> '1' : ${sessionScope.arrayRect['1'].x1}
    Array -> 2 : ${sessionScope.arrayRect[2].x1}
    Array -> index : ${sessionScope.arrayRect[sessionScope.index].x1}

- Affichage résultant
  - Map -> "premier" : 10
    Map -> 'second' : 30
    Map -> "troisieme" :
    Array -> 0 : 10
    Array -> '1' : 30
    Array -> 2 :
    Array -> index : 30
  - Si des éléments dans la map n'existent pas ou si on dépasse l'index max du tableau, renvoie une chaîne vide

JSTL : core

- Utilisation du JSTL core dans une page JSP, insérer la balise dans page JSP
  - <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
  - Au déploiement, ne pas oublier le lien vers la librairie (.jar) externe implémentant cet ensemble de balises
- Affichage d'une valeur
  - <c:out value="*valeur*" />
- Test d'une valeur
  - <c:if test="*condition*"> ... </c:if>
- Choix conditionnels
  - <c:choose>
    <c:when test="*condition1*"> ... </c:when>
    <c:when test="*condition2*"> ... </c:when>
    ...
    <c:otherwise> ... </c:otherwise>
    </c:choose>

Servlet vs JSP

- Servlet
  - C'est du java standard qui est exécuté
  - Les lignes « out.println() » servent à générer le code HTML qui sera affiché
  - Problème : une bonne partie de ce code HTML est statique
    - Entête, affichage du titre ...
    - Un peu lourd à générer de la sorte
- Page JSP
  - Permet de « mélanger » du code HTML standard avec du Java standard
  - Meilleure découpage des différentes parties qu'avec des servlets
  - Moins lourd à programmer qu'une servlet
    - Simplification des affichages des données, surtout si combiné avec UEL et JSTL
    - Mais moins structuré du point de vue du code Java

UEL : opérateurs

- Opérateurs logiques
  - && (ou and), || (ou or), ! (ou not)
- Opérateurs de comparaison
  - == (ou eq), != (ou ne), > (ou gt) ...
- Opérateurs de calcul
  - +, -, *, / (ou div), % (ou mod)
  - Types de nombre disponibles : entier et réel
- Divers
  - empty : renvoie vrai si valeur « vide »
    - Egale à null, chaine vide, un tableau vide ou une map vide
    - ? : choix conditionnel
      - test ? choixVrai : choixFaux

JSTL : core

- Parcours d'une liste / tableau
  - <c:forEach items="*laListe*" var="*eltCourant*"> ...
    </c:forEach>
- Autres tags : <c:forTokens>,<c:set>, <c:remove>, <c:catch>, <c:import>, <c:url>, <c:redirect>, <c:param>
- Exemple
  - <c:if test="${empty sessionScope.arrayRect[2].x1}">
    Pas d'élément d'index 2 dans le tableau

    </c:if>
    Surface des rectangles :
    <c:forEach items="${sessionScope.arrayRect}" var="rect">
    -> <c:out value="${(rect.x2 - rect.x1) * (rect.y2 - rect.y1)}"/>
    </c:forEach>
  - Affichage résultant
    - Pas d'élément d'index 2 dans le tableau

      Surface des rectangles :
      -> 100
      -> 400

Servlet et JSP

- Servlet / JSP
  - Technologie coté serveur permettant donc la génération dynamique de page HTML
  - Intérêt est de profiter de la puissance d'un langage objet
- En pratique
  - Le code embarqué dans les Servlet / JSP ne doit pas intégrer le code métier ou l'accès aux BDD
    - On s'appuie sur d'autres éléments (composants EJB par exemple)
  - Pour un serveur applicatif Web, découper son rôle en
    - Logique de présentation
      - Manière de présenter et de générer les pages aux clients
    - Logique métier
      - Appelée par la logique applicative pour générer les pages

Exemple final

- Base de données
  - Celle vue dans le cours sur JDBC/JPA sur les sports et les sportifs
- Combinaison de JPA, servlets, pages JSP avec UEL et JSTL pour afficher les sportifs avec leurs disciplines et sports pratiqués
  - Page HTML via un lien envoie une requête à la servlet SportServlet
  - Servlet SportServlet.java (d'URL « /Sports »)
    - Récupère via une requête JPQL l'ensemble des sports
    - Ajoute cet ensemble à la requête HTTP reçue par la servlet
    - Puis fait suivre (forward) la requête HTTP à la page afficherSports.jsp
  - Page afficherSports.jsp
    - Récupère l'ensemble des sports dans la requête HTTP
    - Via UEL et JSTL, met en page leur affichage
  - La servlet ne génère aucun code HTML, c'est le rôle de la page JSP à qui on a forwardé la requête

Exemple final

- Lien dans la page HTML
  - Afficher la liste des sportifs
- Code de SportServlet.java
  - public List<Sportif> getListeSportifs() throws Exception {
    // requête JPQL pour récupérer les sportifs dans la BDD
    EntityManager em = emf.createEntityManager();
    Query requete = em.createQuery("SELECT s FROM Sportif s");
    return requete.getResultList();
    }

    protected void processRequest(... request, ... response) {
    String operation = request.getParameter("operation");
    if (operation.equals("listeSportif")) {
    // récupère la liste des sports et l'associe à la requête HTTP
    request.setAttribute("sportifs", getListeSportifs());
    // forwarde la requête à la page JSP
    getServletConfig().getServletContext().getRequestDispatcher(
    "/afficheSportifs.jsp").forward(request,response);
    } (...)

Exemple final

- Page JSP afficheSportifs.jsp

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<h2>Liste des sportifs et de ce qu'ils font</h2>
    <c:forEach items="${requestScope.sportifs}" var="sp">

      <h3>${sp.nom}</h3>
      <p><b>Adresse : </b>${sp.adresse.rue} - ${sp.adresse.codePostal} ${sp.adresse.ville}</p>

      <c:choose>
        <c:when test="${empty sp.disciplines}">
          <p><i>Ne pratique aucune discpline sportive</i></p>
        </c:when>

        <c:otherwise>
          <p><b>Liste des disciplines pratiquées : </b></p>
          <ul>
            <c:forEach items="${sp.disciplines}" var="disc">
              <li>${disc.intitule}</li>
            </c:forEach>
          </ul>
          <p><b>Liste des sports : </b> </p>
          <ul>
            <c:forEach items="${sp.sports}" var="sport">
              <li>${sport.intitule}</li>
            </c:forEach>
          </ul>
        </c:otherwise>
      </c:choose>
    </c:forEach>
```


Exemple final