

Java intensif Programmation Web

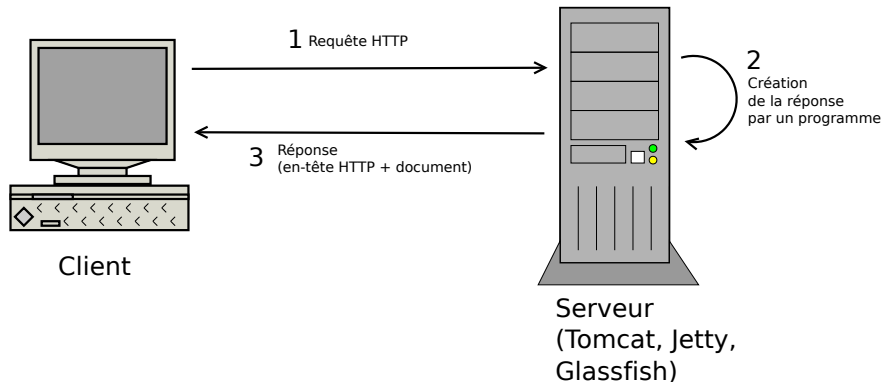
Serge Rosmorduc

2013-2021

Plan

- Rappel : L'architecture web : client, serveur, serveur applicatif ;
- Une application java ;
- notion de servlet ;
- limites des servlets ;
- notion de jsp ; expression language ; beans request ;
- combinaison jsp/servlets, forwarding ;
- architecture du traitement d'un formulaire (détaillé) ;

Rappels sur l'architecture Web

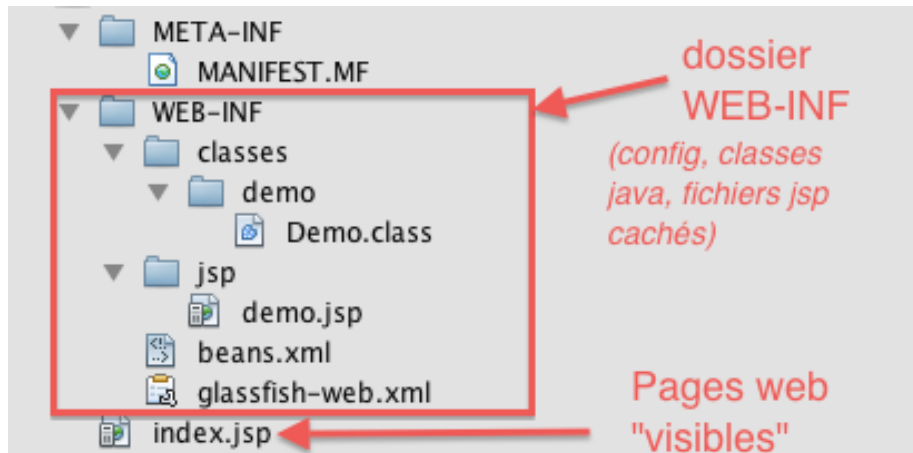


Pour HTML, formulaires... voir supports du cours NFA016 sur [deptinfo](http://deptinfo.org).

L'écosystème J2EE

- Serveurs applicatifs à géométrie variable ; nombreux modules.
 - Couche web : Servlets/JSP ; Wicket, JSF ;
 - + annuaire de services (JNDI) ;
 - + gestion de transactions (JTA) ; gestion de persistance (JTA) ;
 - + load balancing...
 - serveurs simples : Tomcat, Jetty...
 - serveurs complets : JBoss, Glassfish...
 - EJB3 vs. Spring : convergence des approches
 - Abandon de J2EE par Oracle : gestion par la fondation Eclipse ;
- Ce cours : on reste simple, Servlets, JSP, un peu d'archi.

Organisation d'une application Web

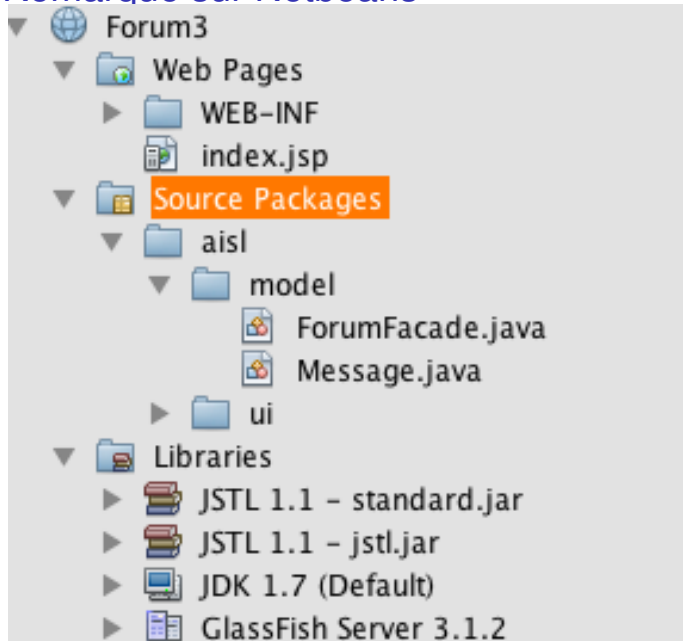


Organisation d'une application Web

L'application est typiquement compressée dans un fichier `.war`
la racine de l'application contient :

- des ressources directement accessibles (pages web, images, jsp) ;
- un dossier nommé `WEB-INF` qui contient :
 - ▶ un fichier de configuration `web.xml` ;
 - ▶ un dossier `classes` contenant des classes java (`.class`) ;
 - ▶ un dossier `lib` contenant des bibliothèques java (`.jar`) ;
 - ▶ ...
- `WEB-INF` n'est pas visible depuis l'extérieur : on y range souvent des ressources qu'on veut cacher.

Remarque sur Netbeans



Rappel sur la méthode GET

Les données sont passées dans l'URL :

`http://www.google.com/search?q=cnam&sourceid=chrome`

- **protocole et serveur** `http://www.google.com`
- **page** : `search`
- **variable(s)** :
 - ▶ `q` qui vaut « `cnam` » ;
 - ▶ `sourceid` qui vaut « `chrome` ».

Une requête en mode get est simple à construire sur un navigateur web.

Rappel sur la méthode POST

- Autre système : les données de la requête sont envoyées dans le corps de la requête http.
- elles ne sont pas visibles.

Notion de Servlet

- classe java qui étend `HttpServlet` ;
- associée à une URL ;
- fournit deux méthodes : `doGet ()` et `doPost ()` ;
- ces méthodes sont supposées récupérer les requêtes et y répondre.

Remarque technique (java 8 vs java 11)

- ancienne version (oracle) : `packages javax.servlet;`
- nouvelle version (eclipse) : `packages jakarta.servlet;`
- les tps sont mis à jour pour `jakarta`; en entreprise, il y a des chances de rencontrer du code `javax`.

Hello World en Servlet

```
@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
                          HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<body>");
            out.println("Bonjour " + request.getParameter("nom"));
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }
}
```

Servlets

- Annotation :

```
@WebServlet(name = "HelloServlet",  
            urlPatterns = "/hello")
```

Dis que la servlet est associée à l'URL « /hello »

- objet `request` : représente la requête. Permet de récupérer les données de formulaire ;
- méthodes utiles :
 - ▶ `String getParameter(String)` : retourne la valeur d'un paramètre
 - ▶ `String[] getParameterValues(String)` : retourne les valeurs d'un paramètre (pour les listes, etc...)
- objet réponse : ici, permet de fixer le type et le codage de la réponse, ainsi que son contenu.

Limitation des servlets

- Génération de HTML très lourde ;
- Le HTML est mélangé au java :
 - ▶ mauvaise séparation affichage/traitements ;
 - ▶ pas adapté à une séparation des tâches

Les JSP

En gros, HTML avec des vrais morceaux de java dedans.

Balises spécifiques

`<% ... %>` : introduit des *instructions* java ; `<%= ... %>` : introduit des *expressions* java, dont le résultat est *affiché*. ;

Objets disponibles

- `out` : permet d'écrire (`JspWriter`)
- `request` : objet requête ;
- `response` : objet réponse ;
- `session` : objet représentant la session ;
- `application` : représente l'application web.

JSP, exemple

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>JSP</title>
  </head>
  <body>
    <p>Bonjour <%= request.getParameter("nom") %></p>

    <p>
      <%
        for (int i= 0; i < 100; i++) {
          out.print("-");
        }
      %>
    </p>
  </body>
</html>
```


Utilisation propre de tout ça

- La **servlet** :

- ▶ c'est du java (chic !)
- ▶ reçoit la requête ;
- ▶ délègue le traitement à des classes « métier » (parce que le web c'est l'enfer) ;
- ▶ délègue l'affichage à une JSP

- La JSP :

- ▶ mélange du HTML et du code ;
- ▶ **ne calcule rien du tout.**
- ▶ **se contente d'afficher les données transmises par la servlet.**

- avantages : éléments développables séparément, *testables* séparément.

En pratique...

Transmettre la main à une JSP depuis une servlet

Exemple : affichage du nom d'une personne capitalisé.

On utilise un `RequestDispatcher` dans la servlet....

```
public class Hello2 extends HttpServlet {
    protected void doGet(HttpServletRequest request,
                          HttpServletResponse response)
        throws ServletException, IOException {
        // Récupération des paramètres
        String nom = request.getParameter("nom");
        // Traitement : une classe métier...
        String nomCap= NomHelper.capitaliser(nom);
        // On transmet les données
        // à afficher comme "beans request"
        request.setAttribute("nomCapitalise", nomCap);
        // On crée le dispatcher
        RequestDispatcher r =
            request.getRequestDispatcher("/WEB-INF/jsp/hello2.jsp");
        // On passe la main à la servlet
        r.forward(request, response);
    }
}
```

Commentaires

- La capitalisation n'est pas si simple (nom null, nom de taille 0 ou 1)...
- on la sort de la servlet, qui doit être le moins intelligente possible
→ création d'une classe `NomHelper`
- la méthode est statique, mais c'est juste parce que l'exemple est simple ;
- séparée de la servlet, elle peut être testée seule ;
- on calcule les données à afficher (ici `nomCap`), et on les transmet à la jsp comme *beans request* à l'aide de la méthode

`request.setAttribute(ident, valeur)`

- ▶ **ident** est un identifiant pour désigner l'attribut ;
- ▶ **valeur** est un objet quelconque (pas forcément une `String`)

RequestDispatcher

Sert à demander à une JSP de terminer le traitement de la requête.

Utilisation

```
String chemin= "/WEB-INF/jsp/hello2.jsp";  
RequestDispatcher r =  
    request.getRequestDispatcher(chemin);  
r.forward(request, response);
```

- chemin : page qui traite l'affichage. Très souvent une JSP, placée dans WEB-INF.
- r.forward(request, response) expédie la requête courante à cette nouvelle page.

L'expression language (première approche)

- Dans la JSP, on peut écrire :

```
1 <%= request.getAttribute("nomCapitalise") %>
```

- Mais c'est long et peu lisible.
- de plus, on souhaite éliminer le code java des JSP : *la JSP ne fait que de l'affichage.*
- Un langage spécial est disponible : `${nomCapitalise}` affiche la valeur du bean.

Propriétés des beans

- Quand un bean est un *objet* avec des accesseurs... ;
- par exemple un bean `personne` de classe `Personne` avec les accesseurs `getNom()` et `getPrenom()` ;
- alors `${personne.nom}` affiche le nom de la personne en appelant `personne.getNom()`

hello2.jsp

```
<!DOCTYPE html>
<html>
  <head>
    <title>JSP Page</title>
  </head>
  <body>
    Bonjour ${nomCapitalise}
  </body>
</html>
```

Récapitulation

- la servlet reçoit une requête, extrait les arguments, et décide quoi faire ;
- elle délègue le traitement effectif à des classes métiers, qui ne connaissent rien du web ;
- puis elle range les valeurs à afficher dans des beans request,
- enfin, elle passe la main à une JSP.

Deuxième partie II

Du traitement des formulaires

Formulaire, exemple

- Application qui
 - ▶ saisit le nom et le prénom d'une personne ;
 - ▶ vérifie que les champs sont remplis ;
 - ▶ calcule le texte de la salutation ;
 - ▶ l'affiche avec une jsp ;
 - ▶ réaffiche le formulaire avec des messages d'erreur *et les données erronées* en cas de problème.
- C'est **la même servlet qui affiche le formulaire et le traite.**
- deux jsp : le formulaire et le résultat.
- souvent, `doGet ()` est appelé pour le premier affichage, et `doPost` pour les traitements.

Servlet

```
public class Saluer extends HttpServlet {
    protected void doGet(...) {
        // A priori, premier appel...
        req.getRequestDispatcher("/WEB-INF/jsp/saluerForm.jsp")
            .forward(req, resp);
    }

    protected void doPost(...) {
        String jsp;
        String nom= req.getParameter("nom");
        String prenom= req.getParameter("prenom");
        if (nom == null || prenom == null ||
            "".equals(nom) || "".equals(prenom)) {
            req.setAttribute("message",
                "les_champs_doivent_être_remplis");
            jsp= "/WEB-INF/jsp/saluerForm.jsp";
        } else {
            req.setAttribute("nom", nom.toUpperCase());
            req.setAttribute("prenom", prenom);
            jsp= "/WEB-INF/jsp/saluer.jsp";
        }
        req.getRequestDispatcher(jsp).forward(req, resp);
    }
}
```

JSP formulaire

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <title>Saluer</title>
  </head>
  <body>
    <p style="color:red">${message}</p>
    <form method="POST">
      <p>nom :
        <input type="text" name="nom" value="${param.nom}"/></p>
      <p>prenom :
        <input type="text" name="prenom" value="${param.prenom}"/></p>
      <p><input type="submit"/></p>
    </form>
  </body>
</html>
```

- Noter que quand `${message}` est absent, ça ne pose pas de problème...
- `${param.nom}` : valeur du *paramètre* nom (et non de l'attribut) : valeur provenant du formulaire qu'on traite.

Amélioration : gestion des erreurs par tableau

```
protected void doPost(...) {
    String jsp;
    HashMap<String,String> erreurs= new HashMap<String,String>();
    String nom= req.getParameter("nom");
    String prenom= req.getParameter("prenom");

    if (nom == null || "".equals(nom)) {
        erreurs.put("nom", "ce_champ_doit_être_rempli");
    }
    if (prenom == null || "".equals(prenom)) {
        erreurs.put("prenom", "ce_champ_doit_être_rempli");
    }

    if (! erreurs.isEmpty()) {
        req.setAttribute("erreurs", erreurs);
        jsp= "/WEB-INF/jsp/saluerForm1.jsp";
    } else {
        req.setAttribute("nom", nom.toUpperCase());
        req.setAttribute("prenom", prenom);
        jsp= "/WEB-INF/jsp/saluer.jsp";
    }
    req.getRequestDispatcher(jsp).forward(req, resp);
}
```

Nouvelle version de la JSP

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Saluer</title>
    <style>
      .erreur {color: red;}
    </style>
  </head>
  <body>
    <form method="POST">
      <p>nom : <input type="text" name="nom" value="${param.nom}"/>
        <span class="erreur"> ${erreurs.nom}</span>
      </p>
      <p>prenom : <input type="text" name="prenom"
        value="${param.prenom}"/>
        <span class="erreur"> ${erreurs.prenom}</span>
      </p>
      <p><input type="submit"/></p>
    </form>
  </body>
</html>
```