Java intensif
Programmation Web

Serge Rosmorduc

2018-2021

Retour sur GET et POST

- Usage correct de GET et POST :
- GET : « bookmarkable », *répétable,* plus ou moins public : particulièrement adapté pour la *récupération* d'information ;
- POST : *a priori* non répétable, non bookmarkable : adapté pour la modification d'information.
- attention aux robots d'indexation : que ferait un robot sur une lien comme

```
<a href="supprimer&id=6">detruire</a>
```

Post et redirection

- Problème de répétition d'un POST lors d'un rechargement de page : modification effectuée deux fois ?
- Produit commandé deux fois au lieu d'une, par exemple...
- Solution : la page qui affiche le résultat de l'exécution du POST ne doit pas être celle qui l'a traité.
- Utilisation de `request.sendRedirect(...)`

Exemple

```
@WebServlet(name = "Supprimer", urlPatterns = {"/supprimer"})
public class SupprimerProduit extends HttpServlet {
    @Override
    protected void doPost(...) {
        int id= Integer.parseInt(req.getParameter("id"));
        BaseDeDonnee.getInstance().supprimer(id);
        resp.sendRedirect("messageSuppression&id=" + id);
    }

}
```

- On charge la page `/supprimer`, en passant en mode `POST` l'id de l'élément à supprimer (par exemple 144) ;
- on supprime l'élément 144 ;
- la servlet envoie une *redirection* : elle demande *au navigateur client* de charger (en mode `GET`) une nouvelle page, `/messageSuppression&id=144`, qui affichera un message confirmant la suppression.
- recharger cette page là se contentera d'afficher à nouveau le message.

Deuxième partie II

Beans

Beans

- Un bean est un objet d'une classe qui :
  - dispose d'un constructeur par défaut (on peut parfois s'en passer) ;
  - *utilise des accesseurs* ;
  - est sérialisable *(utile pour les beans sessions et application)* ;
- un bean a un *nom* et une *portée* ;
- les beans permettent de partager de l'information entre servlets (et jsps) ;
- la portée décrit la durée de vie du bean :
  - `request` : le bean est oublié à la fin de l'exécution de la requête (communication entre servlet et jsp) ;
  - `session` : le bean est lié à une session utilisateur avec un navigateur particulier. Il est oublié au bout d'un certain temps, ou à la fermeture du navigateur.
  - `application` : le bean a une durée de vie qui est celle de l'application elle-même en mémoire.

Beans Request

Permettent à une Servlet d'envoyer des données à une JSP.

Côté Servlet

```
1 String nomBean= ... ;
2 Object valeurBean= ... ;
3 request.setAttribute(nomBean, valeurBean);
```

Côté JSP

Déclaration optionnelle :

```
<jsp:useBean id="monBean" class="String" scope="request"/>
```

Accessible comme variable `monBean` dans le code java ou par l'*expression language* :

```
${monBean}
```

Beans Sessions

- Permet de conserver des données durant toute la durée de la connexion de l'utilisateur au site (termine lorsque le navigateur est fermé).
- les données de sessions sont typiquement conservées en mémoire ;
- utilisation : login d'utilisateur, panier à provisions, etc...
- conservation de l'état de l'application pour l'utilisateur courant.
- nécessite la création d'une session.

Beans Session, côté Servlet...

```
@WebServlet(name = "DemoCompteur", urlPatterns = {"/compteur"})
public class DemoCompteur extends HttpServlet {

    @Override
    protected void doGet(.........) {
        HttpSession session = req.getSession();
        Compteur cpt= (Compteur) session.getAttribute("compteur");
        if (cpt== null) {
            cpt= new Compteur();
            session.setAttribute("compteur", cpt);
        }
        cpt.incrementer();
        req.getRequestDispatcher("/WEB-INF/jsp/compteur.jsp")
              .forward(req, resp);
    }
}
```

- `getSession()` crée la session si elle n'existe pas ;
- Variante de `getSession` avec un booléen comme argument : crée la session uniquement si l'argument est à `true`.

Beans Session, le bean lui-même

```
public class Compteur implements Serializable{

    private int valeur = 0;

    public void incrementer() {
        valeur++;
    }

    public int getValeur() {
        return valeur;
    }
}
```

Beans Session, JSP

```
<!DOCTYPE html>
<html>
    <head>
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Compteur ${compteur.valeur} </h1>
    </body>
</html>
```

Beans Application

- Durée de vie : l'application ;
- Usage : pour éviter de charger des données trop fréquemment (catalogue de produits....) ;
- Utilisation a priori en multitâche : délicate ;
- utiliser avec précaution.

```
@WebServlet(name = "CApp", urlPatterns = {"/compteurApplication"})
public class CompteurApplication extends HttpServlet {

    @Override
    protected void doGet(.........) {
        Compteur cpt= (Compteur)
            req.getServletContext().getAttribute("cptApp");
        if (cpt== null) {
            req.getServletContext()
                    .setAttribute("cptApp", new Compteur());
        }
        ...
    }
}
```

Troisième partie III

Suppression du Java dans les JSP

Expression Language

- Dans les JSP, écrites entre `${....}`
- L'opérateur « a.b » a des effets variés selon le type de a.
  - si a est une Map, a.b ⇔ `a.get(b)`
  - si a est un objet, a.b ⇔ `a.getB()`
  - si a est un tableau a.b ⇔ `a[b]` (ex. `${tab.3}`

On peut utiliser « . » en cascade :

```
1  <p> rue ${facture.adresse.rue}....
```

Le langage d'expression est aussi utilisable pour des calculs :

```
1    prix TTC : ${facture.montant * 1.196}
```

Expression Language (suite)

Opérateurs

- `empty` : savoir si un bean est défini ou non. `${empty facture}` *Leftrightarrow* y-a-t-il un bean `facture` ?
- opérateurs arithmétiques :+, -, *, /, % ;
- opérateurs logiques : and, or, not ;
- comparaisons : eq (equal), ne (not equal), lt (less than), gt (greater than), ge (greater or equal), le (lesser or equal)

variables prédéfinies

param : valeur des paramètres. À utiliser pour les paramètres mono-valués ; par exemple `${param.nom}`

paramValues : valeur des paramètres. À utiliser pour les paramètres multi-valués (résultat de sélections multiples).

cookie : permet l'accès aux cookies.

JSTL

- Java Standard Tags Library ;
- jeu de tags pour remplacer la plupart des constructions java dans les JSP ;
- travaille de très près avec l'expression language.
- Utilisation : mettre la ligne

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

  vers le début du fichier jsp.
- éventuellement ajouter la bibliothèque `JSTL 1.1` à votre projet.

Comparez

Sans JSTL

```
<% for (int i=0; i < facture.getLignes().size(); i++) {
        siteMarchand.dto.LigneFacture ligne=
                (siteMarchand.dto.LigneFacture)facture.getLignes().get(i);
%>
  <li><%= ligne.getArticle().getDesignation() %>,
       qté : <%= ligne.getQuantite() %>
<%
        }
%>
```

Avec JSTL

```
<c:forEach var="ligne" items="${facture.lignes}">
        <li> ${ligne.article.designation}, qté : ${ligne.quantite}</li>
</c:forEach>
```

Boucles en JSTL

- boucle `forEach`.
- Une variable dont le nom est donné par « `var=` » prend une série de valeurs.

```
Parcours d'une liste à partir de l'élément numéro 4 :

<ul>
<c:forEach var="e" items="${promotion}" begin="4">
        <li> ${e} </li>
</c:forEach>
</ul>

Parcours d'une map :
<ul>
<c:forEach var="a" items="${param}">
        <li> ${a.key} : ${a.value}</li>
</c:forEach>
</ul>

Numérique : affiche 0 3 6 9 12
<c:forEach var="i" begin="0" end="12" step="3">
        ${i}
</c:forEach>
```

Test en JSTL

If (sans else)

```
<!-- Afficher si le bean "catalogue" est présent -->
<c:if test="${not empty catalogue}">
        Voici le catalogue :
        ...
</c:if>
```

choose/when/otherwise

Test complet :

```
<c:choose>
  <c:when test="${securite eq 'administrateur'}"> ..... </c:when>
  <c:when test="${securite eq 'utilisateur'}"> ..... </c:when>
  <c:when test="${securite eq 'invite'}"> ..... </c:when>
  <c:otherwise> Désolé </c:otherwise>
</c:choose>
```

Protection contre le XSS

- Pourquoi : éviter les injections de Javascript.
- le code

```
1  <h1> Message  </h1> <p> ${message}</p>
```

est dangereux :

- un utilisateur a entré comme message :

```
1  <SCRIPT language="Javascript">
2          document.location.href="http://sitePirate.com"
3  </SCRIPT>
```

- le javascript est chargé, exécuté, et le lecteur envoyé du le site pirate : XSS (Cross Site Scripting).
- solution : transformer les < et > en caractères normaux.

Protection contre le XSS

- Solution : la balise `<c:out value="...">`
- le code devient

```
1  <h1> Message </h1> <p> <c:out value="${message}"/></p>
```

- sortie :

```
1     <h1> Message </h1>
2     &lt;SCRIPT language=&#034;Javascript&#034;&gt;
3             document.location.href=&#034;http://sitePirate.com&#034; ;
4     &lt;/SCRIPT&gt;
```

Très laid à l'affichage, mais inoffensif.

Constuction d'URL

- pour écrire des liens dans la page ;
- les problèmes :
  - comment ajouter le chemin de l'application en début de lien ?
  - comment coder correctement les arguments en mode GET ?

La solution : `c:url`
Exemple simple, dans l'application `demos` :

```
<a href="<c:url value='/list'/>">liste</a>
```

va engendrer le code html :

```
<a href="/demos/list">liste</a>
```

Au lieu de devoir écrire :

```
<a href="<%= request.getContextPath()%>/list">liste</a>
```

ça reste peu lisible...

c :url

On peut copier l'URL dans une variable, qui sera reprise en utilisant l'expression language :

```
<c:url var="urlListe" value='/list'/>
<a href="${urlListe}">liste</a>
```

On peut donner des paramètres à l'URL (mode GET) :

```
    <c:url var="urlPage" value='/page'>
        <c:param name="id" value="${numPage}"/>
    </c:url>

<a href="${urlPage}">page suivante</a>
```

Ce qui donne

```
<a href="/demos/page?id=3">page suivante</a>
```

Les données sont correctement encodées.

Gestion d'utilisateur et identification

Principes :

- on stocke les informations sur l'utilisateur connecté quelque part (en session, par exemple) ;
- si un visiteur essaie de réaliser une tâche interdite, on l'envoie sur le formulaire de login ;
- lors du login, si le couple login/password est correct, on place l'objet *utilisateur* en session ;
- pour vérifier la connexion, on regarde s'il y a un objet *utilisateur* dans la session ;
- déconnexion : on peut utiliser
  `session.removeAttribute("utilisateur");`

Il y a d'autres possibilités. On peut aussi utiliser le système d'authentification du protocole HTTP.