Outils de programmation Web

L'objectif de ce TP est de bien comprendre et maîtriser la technologie des servlets. La maîtrise de ces briques de base doit vous permettre de construire de larges applications Web. Aborder l'ensemble des subtilités liées à la technologie des servlets est relativement difficile. C'est pourquoi ce TP est relativement proche d'un tutoriel. Suivez pas à pas les consignes, utilisez les exemples fournis par TOMCAT et les annexes de ce document afin de bien comprendre les différents points abordés.

1 Rappel sur les Servlets

1.1 A quoi sert une Servlet ?

— créer des pages Web dynamiques ;
— effectuer des tâches de type `cgi` (traitement applicatif côté serveur Web) ;
— écrire une application en JAVA dont l'interface utilisateur est dans un navigateur.

1.2 Le cycle de vie d'une servlet

Le cycle de vie d'une servlet est assuré par le moteur de servlets, aussi appelé conteneur de servlets. Le serveur crée un pool de threads auxquels il va pouvoir affecter chaque requête. Lors de la première requête `HTTP` à destination de la servlet, celle-ci est chargée en mémoire par le moteur de servlet. Cette phase est appelée déploiement. La servlet est instanciée par le serveur. Une fois instanciée, la servlet est initialisée afin de pouvoir procéder à ses besoins d'initialisation (connexion à une base de donnée, ouverture d'une connexion réseau, ouverture d'un fichier, ...Rq : dans notre situation, nous établirons une connexion avec le serveur annuaire non pas à l'initialisation de la servlet `Login`, mais au début de de la méthode `doPost`). L'initialisation de la servlet se fait par l'appel de la méthode `init()` sur cette dernière par le conteneur. La servlet peut ensuite traiter les requêtes qu'elle reçoit. Lors de la première requête, le conteneur crée les objets `Request` et `Response` spécifiques à la requête. La méthode `doXXX()` est appelée à chaque requête `XXX` dans un nouveau thread. Les objets `Request` et `Response` lui sont passés en paramètre. Grâce à l'objet `Request`, la méthode `doXXX()` va pouvoir analyser les informations en provenance du client. Grâce à l'objet `Response`, la méthode `doXXX()` va fournir une réponse au client. La méthode `destroy()` est appelée lors du déchargement de la servlet, c'est-à-dire lorsqu'elle n'est plus requise par le serveur. La servlet est alors signalée au garbage collector (communément francisé en ramasse-miette).

2 Description du sujet de TP

Nous vous proposons, en vous basant sur le serveur d'annuaire réalisé lors du deuxième TP, de créer une application Web permettant de sélectionner un annuaire, consulter, modifier, supprimer les fiches de l'annuaire. Les fonctionnalités à implanter sont les suivantes :
— l'authentification : l'utilisateur devra s'authentifier avant de pouvoir accéder à l'application ;
— la consultation du contenu du serveur ;
— la sélection d'un annuaire ;
— la modification, l'ajout, et le retrait de personnes de l'annuaire.

Plusieurs pages Web seront nécessaires pour cette application :
— une première page doit permettre au client de s'authentifier ;
— une seconde page devra faire apparaître plusieurs informations :
  — un message précisant que la personne de login « `xxxx` » de la machine « `yyyy` » est maintenant connectée au serveur annuaire « `zzzz` » ;

— le contenu du serveur doit être consultable facilement. Ce contenu sera montré sous forme de liens, chaque lien correspondant à un annuaire. Le fait de cliquer sur un lien devra permettre de visualiser toutes les fiches de cet annuaire.
— un formulaire html permettant de modifier une fiche, d'ajouter une fiche ;
— un bouton `Logout` permettant de quitter l'application et donc de se déconnecter du serveur annuaire.

L'application se veut assez simple. Les servlets vont servir de relais entre le navigateur Web et le serveur annuaire. Ainsi, de façon générale, une servlet va transformer une requête http en une requête annuaire à destination du serveur annuaire, puis va traiter la réponse du serveur annuaire et renvoyer une réponse sous forme d'un formulaire html au navigateur Web.

3 L'authentification

Le but de cette partie est de permettre à un utilisateur de se connecter au serveur annuaire via un navigateur Web et d'obtenir en retour l'accès au serveur annuaire à travers la deuxième page web. En étudiant les exemples fournis par Tomcat (`HelloWorld`, `Request Info`, `Request Parameters`, `Sessions`), vous devrez construire la servlet `Login` qui :

1. lors de son initialisation récupère les paramètres d'initialisation de la servlet qui contiennent les informations concernant le serveur annuaire (`host` et `port`) ;
2. reçoit depuis un formulaire html grâce à la méthode `POST` le login et le mot de passe du client ;
3. ouvre une connexion avec le serveur annuaire (en utilisant `host` et `port`) ;
4. effectue l'authentification de l'utilisateur ;
5. crée une session et stocke dans cette session le login de l'utilisateur, le nom du serveur annuaire et la connexion vers l'annuaire ;
6. interroge le serveur annuaire afin de connaître son contenu ;
7. affiche un formulaire html qui correspond à la deuxième page web.

Avant de voir étape par étape comment mettre en œuvre la servlet `Login`, rappelons à quoi correspondent les objets suivants : `HttpSession`, `ServletConfig` et `ServletContext`. Une session est propre à un client et est accessible par toutes les servlets de l'application. La config est propre à une servlet. Le contexte est propre à une application et donc partagé par toutes les servlets de l'application. Dans la config de la servlet `Login` on ira chercher le `host` et le `port` correspondant au serveur annuaire (ce choix est discutable mais c'était pour illustrer qu'on pouvait initialiser une servlet dans le `web.xml`). La session va contenir la connexion une fois créée à l'appel de `Login`. Toutes les servlets en auront besoin pour interargir avec le serveur annuaire. On rajoute dans la session le login comme cela les servlet pourront recréer toute la deuxième page web et on place le `host` dans le contexte. ***Pour en savoir plus, question 5 et 6 de la FAQ***.

3.1 Récupération de données d'initialisation de la servlet

Créez une servlet nommée `Login.java` qui récupère les données d'initialisation de la servlet et qui affiche les valeurs des champs « `host` » et « `port` » sur une nouvelle page web.
Pour lancer votre servlet : `http://localhost:8080/tpcar/servlet/Login`
voir question 1 et 6 de la FAQ

3.2 Récupération de données depuis un formulaire

Dans la servlet `Login.java`, récupérez les données du formulaire `http://localhost:8080/tpcar/index.hml` (à créer) et affichez les valeurs des champs « `host` », « `port` », « `login` » et « `pass` » sur une nouvelle page web.
voir question 3 de la FAQ

3.3 Ouverture d'une connexion avec le serveur annuaire

Créez une méthode `connectAnnuaire()`. Cette méthode doit utiliser la partie cliente développée dans le TP2 afin d'envoyer une requête de connexion vers un serveur `annuaire`. Cette méthode récupérera

donc un objet `client` tel que vous l'avez conçu dans le TP2, l'important étant que cet objet contienne la connexion vers le serveur `annuaire`. Si une erreur survient, vous devrez envoyer un message approprié à l'utilisateur.

3.4 Création d'une session

Il s'agit de sauver dans une session le login de l'utilisateur, l'adresse du serveur annuaire et la connexion annuaire créée en 3.3. Les autres servlets pourront donc trouver les informations relatives à l'utilisateur dans la session.
voir question 5 de la FAQ
Remarque : on pourrait utiliser un cookie pour mettre en œuvre la gestion d'une session : créez une méthode `writeCookie()` qui prend en paramètre le numéro de session `SessionID` et qui crée un cookie avec comme nom `SessionCAR` et comme valeur l'`ID` de la session.
voir question 4 de la FAQ

4 Annexe : Faq sur les Servlets

Question 1

Comment envoyer du texte au navigateur ?
La réponse d'une Servlet au navigateur se fait via un flot de sortie. Pour ce faire, il faut récupérer le flot de sortie contenu dans l'objet `HttpServletResponse` passé en paramètre dans les méthodes `doXXX()`. L'objet réponse nous permet de créer un objet `PrintWriter`. Il faut alors commencer par définir le type de la réponse (par exemple : `"text/html"`) Ensuite nous extrayons le flot de sortie, objet `PrinterWriter`, en utilisant la méthode `getWriter` de la réponse. Une fois le flot de sortie récupéré, l'envoi de donnée se fait par l'appel de la méthode `println()` sur l'objet `PrintWriter`. L'envoi effectif du texte se fait au moment de la fermeture de l'objet `PrintWriter`.

```
public void doXXX( HttpServletRequest req, HttpServletResponse res)
Throws ServletException, IOException
{
  ...
  ...
  res.setContentType("text/html"); // définition du type de la sortie
  PrinterWriter out = res.getWriter(); //ouverture du flot de sortie
  out.println("<HTML>"); //préparation de l'envoi de données
  out.println("<HEAD>");
  out.println("<TITLE>Titre de la page !!!</TITLE>");
  out.println("</HEAD>");
  out.println("<BODY>");
  ...
  ...
  out.println("</BODY>");
  out.println("</HTML>");
  out.close(); //fermeture du flot de sortie et envoi des données
}
```

Voir l'exemple `HelloWorld` et la Javadoc sur les interfaces : `javax.servlet.http.HttpServletResponse` et `javax.servlet.ServletResponse`.

Question 2

Comment récupérer des informations sur le client ? Dans de nombreux cas, il est très important de récupérer un certain nombre d'informations sur le client du service. L'objet `HttpServletRequest` passé en paramètre des méthodes `doXXX()` permet de récupérer un certain nombre d'informations sur l'émetteur de la requête. Par exemple la méthode `getRemoteHost` retourne le nom complet du client qui a émis la requête.

```
public void doXXX( HttpServletRequest req, HttpServletResponse res)
Throws ServletException, IOException
{
  ...
  String NomDuClient = req.getRemoteHost();
  ...
}
```

Question 3

Comment récupérer des données depuis un formulaire HTML ?

Rappelons le fonctionnement d'un formulaire html.

```
<html>
  <head>
    <title>Un exemple simple de formulaire</title>
  </head>
  <body>

    <H1>Exemple simple de formulaire utilisant une Servlet</H1>
    <P> Ceci est un formulaire simple qui illustre l'utilisation des Servlets pour traiter
    les informations saisies dans les zones de dialogue. </P>

    <HR>
    <form method="post" action="examples/servlet/TraitementFormulaire">

      <H2>Questionnaire</H2>
      <P> Quel est votre prénom ? <input name="prenom"/> </P>
      <P> Quel est votre nom ? <input name="nom"/> </P>

      <P> Quelle est votre couleur favorite ?
      <select name="couleur">
        <option selected>blanc</option selected>
        <option>jaune</option>
        <option>orange</option>
        <option>rouge</option>
        <option>vert</option>
        <option>bleu</option>
        <option>violet</option>
        <option>noir</option>
      </select> </P>

      <P> Aimez vous l'informatique ?
      <input type="radio" name="choix" value="oui" checked> Oui ou
      <input type="radio" name="choix" value="non"> Non </P>

      <P> Veuillez expliquer brièvement ci-dessous le choix que vous venez de faire :
      <input name="message" size=60,5> </P>

      <P> Cliquez sur <input type="submit" value="Valider">
      pour soumettre votre requête., sinon
      <input type="reset" value="Annuler"> </P>
    </form>
  </body>
</html>
```

Dans ce formulaire, quand l'utilisateur clique sur `Valider`, la servlet `TraitementFormulaire` est exécutée. L'objet `HttpServletRequest` passé en paramètre des méthodes `doXXX()` permet de récupérer un certain nombre d'informations sur la requête. Entre autres, pour collecter les données entrées par l'utilisateur dans la page, nous avons recours à la méthode `getParameter(String key)`. Le paramètre `key` indique le nom du paramètre dont nous souhaitons extraire une valeur. Dans l'exemple ci-dessus `key` pourra prendre comme valeur (`"prenom`, `nom`, `couleur`, `choix` ou `message"`) afin d'obtenir les données fournies par l'utilisateur dans le formulaire. Toutefois, si plusieurs valeurs ont été associées à la même clé et que l'on désire récupérer toutes ces valeurs, il est nécessaire d'utiliser la méthode getParameterValues(String key) qui retourne l'ensemble de ces valeurs.

Question 4

Comment se servir des Cookies ?

Introduction aux cookies

Les cookies représentent un moyen simple de stocker temporairement des informations chez un client, afin de les récupérer ultérieurement. Concrètement il s'agit de fichiers texte stockés sur le disque dur du client après réception d'une réponse `HTTP` contenant des champs appropriés dans l'en-tête. Les cookies font partie des spécifications du protocole `HTTP`. Les requêtes et réponses `HTTP` contiennent des en-têtes permettant d'envoyer des informations particulières de façon bilatérale. Un de ces en-têtes est réservé à l'écriture de fichiers sur le disque : les cookies. L'en-tête `HTTP` réservé à l'utilisation des cookies s'appelle `Set-Cookie`, il s'agit d'une simple ligne de texte de la forme :

`Set-Cookie : NOM=VALEUR; domain=NOM_DE_DOMAINE; expires=DATE`

Il s'agit donc d'une chaîne de caractères commençant par `Set-Cookie :` suivie par des paires clés-valeur sous la forme `CLE=VALEUR` et séparées par des virgules. L'API servlet de Java propose un objet permettant de gérer de façon quasi-transparente l'usage des cookies, il s'agit de l'objet `Cookie`.

L'objet `Cookie`
La classe `javax.servlet.http.Cookie` permet de créer un objet `Cookie` encapsulant toutes les opérations nécessaires à la manipulation des cookies. Ainsi, le constructeur de la classe `Cookie` crée un cookie avec un nom et une valeur initiaux passés en paramètre. Il est toutefois possible de modifier la valeur de ce cookie ultérieurement grâce à sa méthode `setValue()`.

Envoi du cookie
L'envoi du cookie vers le navigateur du client se fait grâce à la méthode `addCookie()` de l'objet `HttpServletResponse` : `void AddCookie(Cookie cookie)`. Etant donnée que les cookies sont stockés dans les en-têtes `HTTP`, et que celles-ci doivent être les premières informations envoyées, la création du cookie doit se faire avant tout envoi de données au navigateur (le cookie doit être créé avant toute écriture sur le flot de sortie de la servlet).

```
Cookie MonCookie = new Cookie("nom", "valeur");
response.addCookie(MonCookie);
```

Récupération des cookies du client
Pour récupérer les cookies provenant de la requête du client, il suffit d'utiliser la méthode `getCookies()` de l'objet `HttpServletRequest` : `Cookie[] getCookies()`.Cette méthode retourne un tableau contenant l'ensemble des cookies présents chez le client. Il est ainsi possible de parcourir le tableau afin de retrouver un cookie spécifique grâce à la méthode `getName()` de l'objet `Cookie`.

Récupération de la valeur d'un cookie
La récupération de la valeur d'un cookie se fait grâce à la méthode `getValue()` de l'objet `Cookie`

```
String Valeur = Cookie.getValue();
```

Question 5

Comment se servir des Sessions Java ?

Le protocole `HTTP` est un protocole non connecté (on parle aussi de protocole sans états, en anglais stateless protocol), cela signifie que chaque requête est traitée indépendamment des autres et qu'aucun historique des différentes requêtes n'est conservé. Ainsi le serveur web ne peut pas se « souvenir » de la requête précédente, ce qui est dommageable dans des utilisations telles que le e-commerce, pour lequel le serveur doit mémoriser les achats de l'utilisateur sur les différentes pages. Il s'agit donc de maintenir la cohésion entre l'utilisateur et la requête, c'est-à-dire reconnaître les requêtes provenant du même utilisateur, associer un profil à l'utilisateur, connaître les paramètres de l'application (nombre de produits vendus, ...). On appelle ce mécanisme de gestion des états le « suivi de session » (en anglais session tracking).

Les méthodes traditionnelles de suivi de session
Il existe une méthode, pouvant se décliner en plusieurs variantes, permettant d'assurer le suivi de session. Elle consiste à stocker les informations concernant l'utilisateur sur le serveur, et de les relier à chaque requête grâce à un identifiant, généralement appelé id, présent dans la requête et sur le serveur. Les méthodes les plus directes associées au partage des données de sessions sont la réécriture d'URL, les champs de formulaires cachés ou l'envoi de cookies. Concrètement, ces techniques écrivent toutes les deux des données dans le code HTML qu'elles envoient au navigateur, afin de forcer celui-ci à les intégrer dans les requêtes suivantes. Cependant, lors du développement de servlets, il est possible d'utiliser une autre méthode, les objets `HttpSession`. Les informations concernant l'utilisateur sont alors stockées sur le serveur dans un objet persistant, nommé `HttpSession`. Il permet de stocker les informations saisies au fur et à mesure par l'utilisateur et de les relier à chaque requête grâce à l'identifiant passé en paramètre. Nous ne nous intéressons ici qu'à cette dernière méthode.

L'objet `HttpSession`
L'objet `HttpSession` permet de mémoriser les données de l'utilisateur, grâce à une structure similaire à une table de hachage, permettant de relier chaque id de session à l'ensemble des informations relatives à l'utilisateur. Ainsi en utilisant un mécanisme tel que les cookies, permettant d'associer une requête à un id, et l'objet `HttpSession`, permettant de relier des informations relatives à l'utilisateur à un id, il est possible d'associer facilement une requête aux informations de session ! L'objet `HttpSession` s'obtient grâce à la méthode `getSession()` de l'objet `HttpServletReqest`.

Gérer les sessions
La gestion des sessions se fait de la manière suivante :
— Obtenir l'ID de session
  — Si GET : en regardant dans la requête
  — Si POST : en regardant dans les en-têtes HTTP
  — Sinon dans les cookies
— Vérifier si une session est associée à l'ID
— Si la session existe, obtenir les informations
— Sinon Générer un ID de Session
— Si le navigateur du client accepte les cookies, ajouter un cookie contenant l'ID de session
— Sinon ajouter l'ID de session dans l'URL
— Enregistrer la session avec l'ID nouvellement créé

Obtenir une session
La méthode `getSession()` de l'objet `HttpServletRequest` permet de retourner la session relative à l'utilisateur (l'identification est faite de façon transparente par cookies ou réécriture d'URL) :
`HttpSession getSession(boolean create);`
L'argument `create` permet de créer une session relative à la requête lorsqu'il prend la valeur `true`. Etant donné que les cookies sont stockés dans les en-têtes HTTP, et que celles-ci doivent être les premières informations envoyées, la méthode `getSession()` doit être appelée avant tout envoi de données au navigateur (la méthode doit être invoquée avant toute écriture sur le flot de sortie de la servlet). L'exemple suivant récupère la session associée à la requête de l'utilisateur et la crée si elle n'existe pas déjà :

```
public class Caddie extends HttpServlet {

  public void doGet (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Recupere la session HttpSession
    session = request.getSession(true);
    ...

    // Ecrit la reponse
    out = response.getWriter();
    ...
  }
}
```

Obtenir des informations d'une session
Pour obtenir une valeur précédemment stockée dans l'objet `HttpSession`, il suffit d'utiliser la méthode `getAttribute()` de l'objet `HttpSession` (celle-ci remplace dans la version 2.2 de l'API servlet la méthode `getValue()` qui était utilisée dans les versions 2.1 et inférieures). La méthode `getAttribute()` retourne une instance d'`Object`, il faut donc effectuer un surtypage pour obtenir un type élémentaire de données. Si l'attribut passé en paramètre n'existe pas, la méthode retourne la valeur `null`.

```
public class Caddie extends HttpServlet {
  public void doGet (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // Recupere la session HttpSession
    session = request.getSession(true);
    // Recupere l'age de l'utilisateur
    age = (Integer)session.getAttribute("Age");
    if (age != null) {
      // ... faire quelque chose puis ecrit la reponse
      out = response.getWriter();
    } else {
      age = new Integer(...);
      //... faire quelque chose d'autre puis ecrit la reponse
      out = response.getWriter();
    }
  }
}
```

Stocker des informations dans une session
Le stockage d'informations dans la session est similaire à la lecture. Il suffit d'utiliser la méthode `setAttribute()` en lui fournissant comme paramètres la clé et la valeur associée. L'exemple suivant stocke dans la session la page d'appel de la servlet (une page ayant fait un lien) :

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Caddie extends HttpServlet {

  public void doGet (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Recupere la session
    HttpSession session = request.getSession(true);


    // la page d'appel
    session.setAttribute("referer",request.getHeader("Referer"));
  }
}
```

Invalider une session
D'une manière générale, il est préférable de laisser une session se terminer seule, par expiration. Cependant, dans certains cas, il peut être utile de supprimer manuellement une session si l'utilisateur achète le contenu du caddie par exemple. Pour supprimer une session, il suffit de faire appel à la méthode `invalidate()` de l'objet `HttpSession`.

```
public class Caddie extends HttpServlet {

  public void doGet (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Recupere la session
    HttpSession session = request.getSession(true);

    session.invalidate();
  }
}
```

Question 6

Comment utiliser le contexte d'une servlet ? Quelle est la différence entre la configuration d'une servlet et le contexte d'une servlet ?

Définition d'un contexte
Un contexte constitue pour chaque servlet d'une même application une vue sur le fonctionnement de cette application. Une application web peut être composée de : servlets, JSP, classes utilitaires, documents statiques (pages html, images, sons, etc.), beans, applications clients, méta informations décrivant la structure de l'application. Dans le code source d'une servlet, un contexte est représenté par un objet de type `ServletContext` qui peut être obtenu par l'intermédiaire d'un objet de type `ServletConfig`, par exemple de la façon suivante :

```
  public void init(ServletConfig config) {
    ServletContext contexte = config.getServletContext();
    /* suite du code */
  }
```

Grâce à ce contexte, il est possible d'accéder à chacune des ressources de l'application web correspondant au contexte. Il faut noter qu'à une application correspond un et un seul contexte, et qu'à un contexte correspond une et une seule application. On peut donc en déduire que chaque contexte est propre à une application et qu'il n'est pas possible de partager des ressources entre applications différentes. Par exemple la servlet `ServletLogin` de l'application de commerce en ligne ne peut pas

accéder aux instances de servlets de l'application de configuration du serveur web. Les ressources qu'il est possible de partager sont :

— des documents statiques : vous pouvez accéder à n'importe quel document statique d'un contexte grâce à la méthode `getRessource` ou à `getRessourceAsStream`. Le chemin passé en paramètre est interprété de façon relative à la racine de l'application correspondant au contexte en cours.
— des instances de servlets : en utilisant la méthode `getServlet`.
— des paramètres d'initialisation pour toute l'application : en utilisant la méthode `getInitParameter`, il est possible de connaître la valeur d'un paramètre d'initialisation si on possède son nom. Pour connaître tous les paramètres d'initialisations définis, il faut utiliser la méthode `getInitParameterNames`.
— des attributs d'instance de servlets : il est ainsi possible de partager des données au sein d'une même application. Pour cela, utilisez les méthodes `setAttribute` et `getAttribute`, en fournissant à chacune de ces méthodes le nom que devra avoir l'attribut au sein de l'application. Pour obtenir le nom de tous les attributs partagés, vous devez utiliser la méthode `getAttributeNames` et pour retirer un attribut, `removeAttribute`.

Vous devriez maintenant comprendre ce qu'est un contexte. C'est une abstraction supplémentaire qui permet de matérialiser les relations privilégiées que connaissent les modules d'une même application et le fait que chaque application est différente. Décrivons maintenant comment mettre en place plusieurs contextes différents avec Tomcat.

Définition d'une configuration

Une configuration d'une serlvet correspond à un ensemble d'informations qui est propre à la servlet et qui ne doit pas être accessible par les autres servlets de l'application. Dans le code source d'une servlet, une configuration est représentée par un objet de type `ServletConfig`. Les élements d'initialisation de cette configuration (voir fichier `web.xml`) sont accessibles de la façon suivante :

```
public void init(ServletConfig config) {
  String host = config.getInitParameter("DirectoryServer");
  /* suite du code */
}
```

Il est possible de rajouter des informations dans la configuration une fois que la servlet est lancée. Il faut alors utiliser les méthodes `setAttribute` et `getAttribute`.

Configuration avec Tomcat

A chaque contexte correspond une arborescence dans le système de fichiers qui contient les ressources accédées lors des requêtes vers le moteur de servlets. Cette arborescence est identique pour chaque contexte. Voici comment se décompose la structure des répertoires :

— la racine : elle fait office de répertoire racine pour les ressources qui font partie du contexte. Par exemple l'URL `http://www.monserveur.fr/commerce/index.html` fait référence au fichier `index.html` de ce répertoire racine. Vous pouvez créer les répertoires que vous désirez ou déposer les fichiers que vous voulez dans ce répertoire, comme par exemple un répertoire `images/` qui sera accessible via l'URL `http:// www.monserveur.fr/commerce/images/`.
— le répertoire `WEB-INF` : situé à la racine, il contient un fichier `web.xml` qui est le descripteur de déploiement du contexte. Il contient tous les paramètres de configuration utilisés par le contexte.
— le répertoire `WEB-INF/classes/` : c'est le répertoire dans lequel vous déposez les classes de vos servlets et des classes personnalisées utilisées par celles-ci. Le chargeur de classe de l'application vient chercher les classes à charger dans ce répertoire.
— le répertoire `WEB-INF/lib/` : il permet de déposer les archives au format jar (Java ARchive Files) qui contiennent des servlets, des beans ou des classes utilitaires utilisées par l'application. Le chargeur de classe de l'application recherche aussi dans ces archives pour trouver les classes dont il a besoin.

La configuration du contexte d'une application

Le fichier `web.xml` est écrit en `XML`. Les informations données par le fichier de configuration sont :

— les paramètres d'initialisation du contexte.
— la configuration de la session.
— les définitions des servlets et des JSPs.

— les correspondances entre servlets et entre JSPs.
— les correspondances entre types MIME.
— la liste des fichiers de bienvenue.
— les pages d'erreur.
— la sécurité.

Nous n'aborderons pas ici tous les aspects de la configuration, mais sachez qu'il est possible de régler tous ces paramètres au sein d'un contexte. Nous ne regardons ici qu'un exemple de fichier de déploiement simple d'une application WEB contenant trois Servlets et contenant des paramètres d'initialisation.

```
<webapp xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                            http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
  <!- nom de l application ->
  <display-name>
    CAR-WebAnnuaire -- TP CAR première partie
  </display-name>

  <!- paramètre disponibles dans le contexte de l application ->
  <context-param>
    <param-name>nomannuaire</param-name>
    <param-value>annuairelabo</param-value>
  </context-param>

  <!- définition de la servlet de login ->
  <servlet>
    <servlet-name>login</servlet-name>
    <servlet-class>Login.class</servlet-class>
    <!- paramètres d initialisation de la servlet ->
    <init-param>
      <pram-name>host</param-name>
      <param-value>localhost</param-value>
    </init-param>
    <init-param>
      <param-name>port</param-name>
      <param-value>1110</param-value>
    </init-param>
  </servlet>
  <!- fin de la définition de la servlet ->

  <!- définition de la servlet de traitement ->
  <servlet>
    <servlet-name>ProcessAjouter</servlet-name>
    <servlet-class>ProcessAjouter.class</servlet-class>
  </servlet>
  <!- fin de la définition de la servlet ->

  <!- fichiers par défaut pour une requête concernant une URL n en spécifiant pas ->
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
  <welcome-file-list>

  <!- définition des pages d erreur ->
  <error-page>
    <!- pour une erreur 404, c est le fichier 404.html qui est transmis au client ->
    <error-code>404</error-code>
    <location>/404.html</location>
  </error-page>

<!- fin de la configuration du contexte ->
</web-app>
```