7

Développement Web avec l'outillage Web Tools et les patterns

Ce chapitre présente l'outillage Web Tools pour le développement Web sans l'aide d'un framework particulier, en utilisant les bonnes pratiques de conception.

Nous commencerons par décrire l'outillage du projet Web Tools (WTP) et rappellerons les principes de base de l'architecture d'une application Web et de ses composants (IHM et couche d'accès aux données) ainsi que les étapes qui vont de son développement jusqu'à son déploiement.

Nous terminerons par le design et la mise en œuvre d'une portion d'application Web selon une approche fondée sur les modèles de conception ou pattern. Ce design et cette mise en œuvre s'effectueront en logique 3-tiers traditionnelle, c'est-à-dire sans l'utilisation d'un framework particulier et en utilisant des composants de type servlet et JSP, ainsi que les bibliothèques de balises JSTL (JavaServer Pages Standard Tag Library).

Enfin, nous ferons un tour d'horizon des fonctionnalités offertes par Web Tools pour simplifier le développement Web. Nous supposons acquises les bases du développement par servlets/JSP et JSTL, ainsi que les notions propres aux API JDBC, que nous nous contenterons de rappeler.

Principes de base de l'exécution d'une requête JDBC à partir d'un programme Java

Avant d'entrer dans le design et la mise en oeuvre du développement Web avec Web Tools, nous allons résumer les bases de l'interrogation SQL d'une base de données à partir d'un programme Java.

L'exécution d'une requête JDBC, implique trois concepts essentiels :

- une connexion base de données (objet Connection) ;

- une requête SQL (objet Statement) ;
- le résultat de la requête (objet ResultSet).

Pour utiliser ces objets, vous devez les importer dans votre application, comme le montre l'extrait de code suivant, dont le code complet est disponible sur la page Web dédiée à l'ouvrage :

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.SQLWarning;
import java.sql.Statement;
import java.sql.ResultSet;

public class FirstQuery {
    private static final String driver = "org.apache.derby.jdbc.EmbeddedDriver" ;
    private static final String url = "jdbc:derby:WebstockDB" ;
    private static final String qry =
        "SELECT articleId, nomArticle, articleCategorieId, fournisseurId,
        ➥description, poids FROM webstock.article" ;
```

Les classes Java et constantes nécessaires à l'interrogation de la base de données (URL d'accès à la base webstockdb et requêtes d'interrogation à la table webstock.article) sont importées explicitement. Chaque colonne est invoquée pour éviter les effets de bord en cas de modification du schéma de base de données sous-jacent.

Les opérations requises pour exécuter une requête à la base à partir d'un programme Java sont relativement simples, comme l'illustre l'extrait de code suivant, qui présente la méthode doQuery avec inclusion de la clause SQLException, qui permet de remonter l'exception vers le code appelant :

```
static void doQuery(Connection con) throws SQLException {

    SQLWarning swarn = null ;

    Statement stmt = con.createStatement() ;
    ResultSet rs = stmt.executeQuery(qry) ;

    while (rs.next()) {

        System.out.println("Numéro Article: " + rs.getString("articleId")) ;
        System.out.println("Nom Article:  " + rs.getString("nomArticle")) ;
        System.out.println("Catégorie Article:  "
        ➥+ rs.getString("articleCategorieId")) ;
System.out.println("Fournisseur Article:  " + rs.getString("fournisseurId")) ;
        System.out.println("Description: " + rs.getString("description") + '\n') ;
        System.out.println("Poids Article: " + rs.getString("poidsArticle") + '\n')


swarn = rs.getWarnings() ;

    if(swarn != null){
            printSQLWarning(swarn) ;
        }
    }
    rs.close() ;
    stmt.close() ;
}
```

Dans la méthode doQuery, vous créez d'abord un nouvel objet JDBC Statement en utilisant la méthode createStatement sur l'objet Connection. Vous utilisez ensuite la méthode executeQuery sur l'objet Statement créé pour envoyer la chaîne de requête à la base Apache Derby où celle-ci est exécutée. Vous accédez ensuite aux résultats de la requête dans le programme Java en utilisant l'implémentation resultset fournie par le package du driver JDBC embarqué dans Derby et en itérant ensuite sur cet objet (rs.next()).

Au sein de cette boucle, vous accédez aux six colonnes de l'enregistrement en utilisant la méthode getString appliquée à l'objet resultset. La méthode getString peut accéder aux colonnes de la table de deux façons : en utilisant le numéro de colonne original dans la requête, par exemple `getString(1)`, ou par `getString("articleId")`. Nous préconisons l'usage explicite des noms de colonnes pour éviter tout risque d'ambiguïté.

La méthode doQuery vérifie explicitement d'éventuels avertissements sur l'objet resultset après accès à chaque nouvel enregistrement effaçant le précédent éventuellement généré. Pour invoquer la méthode doQuery, il est nécessaire d'établir au préalable une connexion à la base de données et d'appeler celle-ci au sein d'un bloc `try…catch`, comme dans l'extrait de code suivant :

```
public static void main(String[] args) {
    Connection con = null ;

    try {
        Class.forName(driver) ;
        con = DriverManager.getConnection(url);

        SQLWarning swarn = con.getWarnings() ;

        if(swarn != null){
            printSQLWarning(swarn) ;
        }

        doQuery(con) ;

    } catch (SQLException se) {
        printSQLException(se) ;
    }
```

Design de l'application Web avec Web Tools

Reprenez le projet webstock ainsi que la base webstock sous Derby, que vous avez commencé à construire au cours des chapitres précédents.

Vous allez développer le cas d'utilisation relatif à la connexion et à l'identification de l'utilisateur en utilisant l'outillage Web Tools. Pour ce faire, vous développerez les ressources suivantes :

- construction d'une page de login d'accès à l'application Web Login.jsp ;
- construction d'une servlet de traitement LoginServlet chargée de vérifier les informations de connexion à la base webstock ;
- connexion à la base webstock *via* le driver JDBC Derby et accès à la table WebStock-Access ;

- en cas de succès lors de l'authentification, transfert du contrôle à une autre servlet de traitement, LoginSuccess, *via* l'objet RequestDispatcher ;
- déploiement/test et débogage sous Web Tools de l'application Web déployée.

Ces ressources seront ensuite déployées sur un serveur JBoss 4.0 ou JBoss 4.2, dont la configuration avec WTP a été abordée au chapitre 5.

Création de la page d'accueil

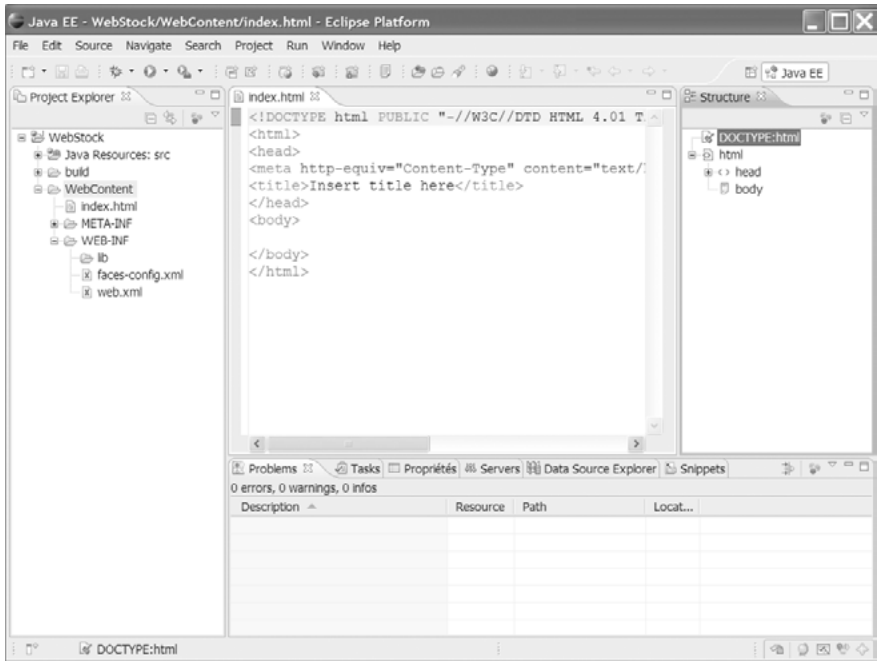
Vous allez construire la page d'accueil standard index.html de l'application Web. Cela vous permettra de faire connaissance avec l'éditeur HTML intégré de Web Tools et d'avoir un aperçu de ses possibilités :

1. Sélectionnez le projet webstock créé au cours des chapitres précédents puis le dossier WebContent.
2. Dans l'assistant de création de page HTML, cliquez sur New puis HTML.
3. Dans le champ File Name, saisissez index, puis cliquez sur Next.
4. Sélectionnez le template proposé par défaut (4.0.1 Transitional), puis cliquez sur Finish.

   Vous devez obtenir le résultat illustré à la figure 7.1 (notez la présence de l'éditeur à syntaxe coloré pour le contenu HTML et du volet Structure pour la navigation au sein des balises HTML).

Figure 7.1

Éditeur HTML de Web Tools



5. Remplacez le contenu situé entre les balises <body> et </body> par le contenu suivant :

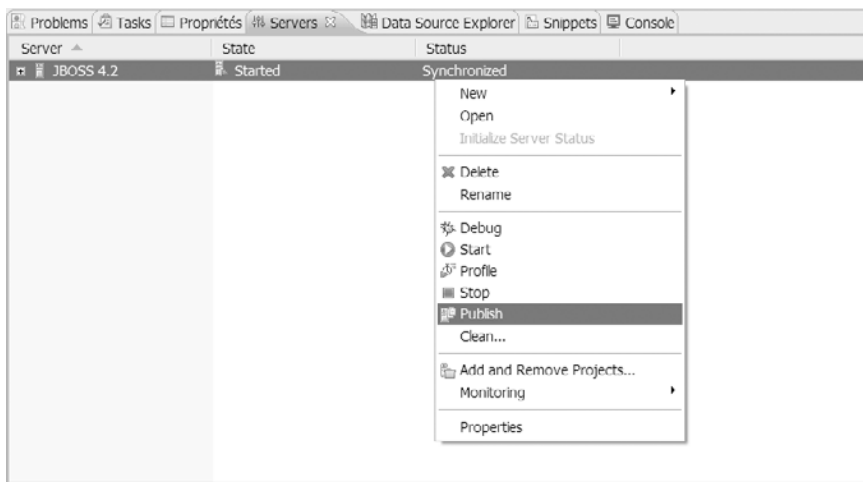
```
<body>
<p align="center"> </p>
```

```
<p align="center"><strong><font size="6"
    style="BACKGROUND-COLOR: #999999"><IMG alt="" src="WebStore.gif"
    border="0">  WebStock</font></strong></p>

<p align="center"> </p>

<p align="center"> </p>

<p align="center"><a title="Login"
    href="http://localhost:8080/LoginServlet/Login.jsp"><font
    style="BACKGROUND-COLOR: #ffffff"><strong><font size="4">
Bienvenue dans le</font></strong> <strong><font size="4">Système</font>
➥</strong><strong><font
    size="4">WebStore</font></strong></font></a></p>

<p align="center"> </p>

<p align="center"> </p>
<hr>
<br>
<br>

<p align="center"><STRONG>(c) Eyrolles <font
    style="BACKGROUND-COLOR: #ffffff">2005</font></STRONG></p>

<p><br>
<br>
</p>
</body>
```

6. Copiez la figure WebStore.gif dans le répertoire WebContent.
7. Faites une publication du contenu des ressources vers le serveur JBoss préalablement configuré en actionnant l'onglet Servers, puis sélectionnez le serveur JBoss dans le menu contextuel (voir figure 7.2).

Figure 7.2

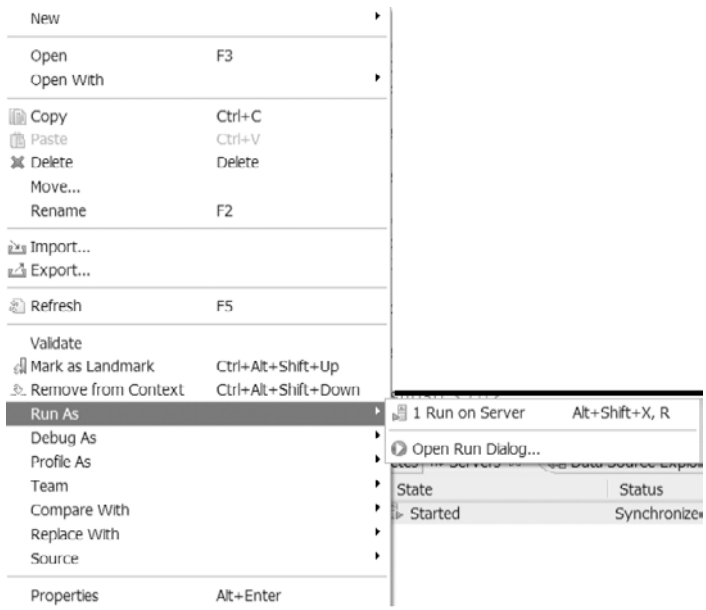
Assistant de publication Web Tools (1/2)



8. Après publication avec succès des ressources figurant dans l'onglet Console, cliquez sur l'option du menu contextuel Start pour lancer le serveur JBoss.
9. Sélectionnez la page index.html dans la vue explorateur de projet, et exécutez votre projet *via* l'option Run As, comme illustré à la figure 7.3.

Figure 7.3

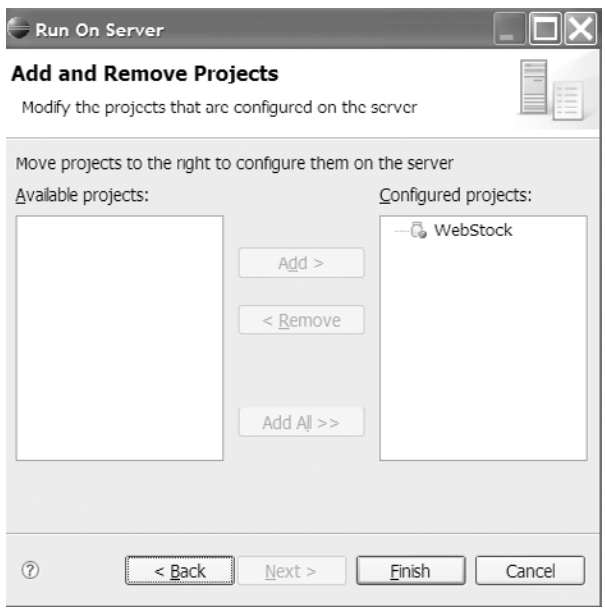
Assistant de publication Web Tools (2/2)



10. L'assistant de sélection de définition d'un serveur s'ouvre. Sélectionnez le serveur que vous avez préalablement défini (ici JBoss 4.2), puis cliquez sur Next (voir figure 7.4).

Figure 7.4

Assistant de définition de serveur



11. Votre projet en cours de définition s'affiche dans la colonne Configured projects. Cliquez sur Finish. Votre page d'accueil s'affiche comme illustré à la figure 7.5.

Figure 7.5

Exécution après déploiement de la page d'accueil de l'application webstock

Création des pages Login.jsp et LoginSuccess.jsp

Vous allez créer vos premières pages JSP en vous aidant de l'assistant de création de pages JSP de Web Tools :

1. Sélectionnez le projet webstock créé aux chapitres précédents puis le dossier WebContent destiné à contenir vos pages JSP, et sélectionnez New puis JSP.
2. Cliquez sur Next.
3. Dans le champ File name, saisissez comme nom de la page JSP Login. Cliquez sur Next.
4. Choisissez le premier modèle de template (JSP with html Markup), puis cliquez sur Finish. La page JSP est automatiquement générée dans l'éditeur Web Tools.
5. Remplacez le contenu de la page générée par le contenu suivant :

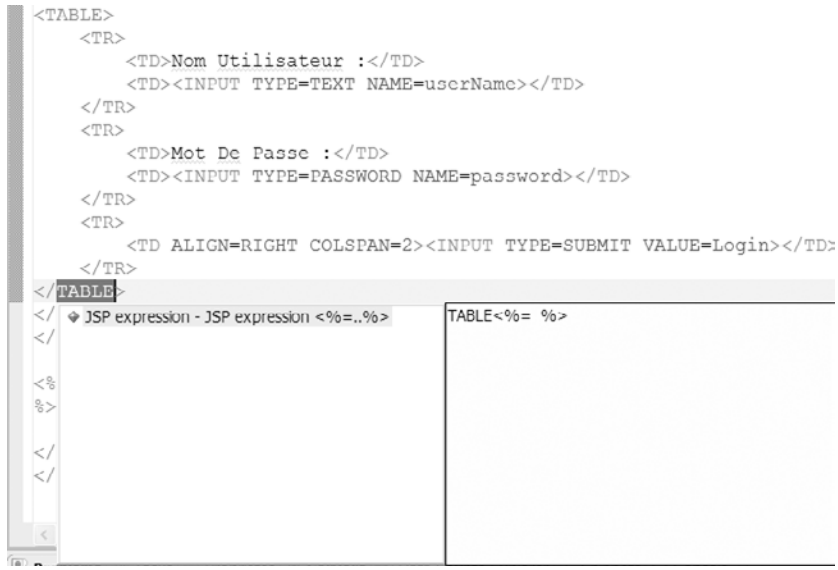
```
<%@ page import="java.util.Calendar"%>

<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":
➥"+request.getServerPort()+path+"/";
%>
<HTML>
<HEAD>
<TITLE>Page d'Identification</TITLE>
```

```
</HEAD>
<BODY>
<CENTER><BR>
<BR>
<H2>Login Page</H2>
<BR>
<BR>
Saisissez votre nom et votre mot de passe<BR>
<BR>
<FORM METHOD=POST ACTION="/WebStock/LoginServletTask">
<TABLE>
    <TR>
        <TD>Nom Utilisateur :</TD>
        <TD><INPUT TYPE=TEXT NAME=userName></TD>
    </TR>
    <TR>
        <TD>Mot De Passe :</TD>
        <TD><INPUT TYPE=PASSWORD NAME=password></TD>
    </TR>
    <TR>
        <TD ALIGN=RIGHT COLSPAN=2><INPUT TYPE=SUBMIT VALUE=Login></TD>
    </TR>
</TABLE>
</FORM>
</CENTER>

<% out.println("Date courante: " + Calendar.getInstance().getTime());
%>

</BODY>
</HTML>
```

6. Pour apprécier les fonctionnalités de complétion de code de Web Tools, positionnez votre curseur sur le début de la balise recherchée, puis pressez Ctrl+Espace (voir figure 7.6).

Figure 7.6

Outil de complétion de code de Web Tools



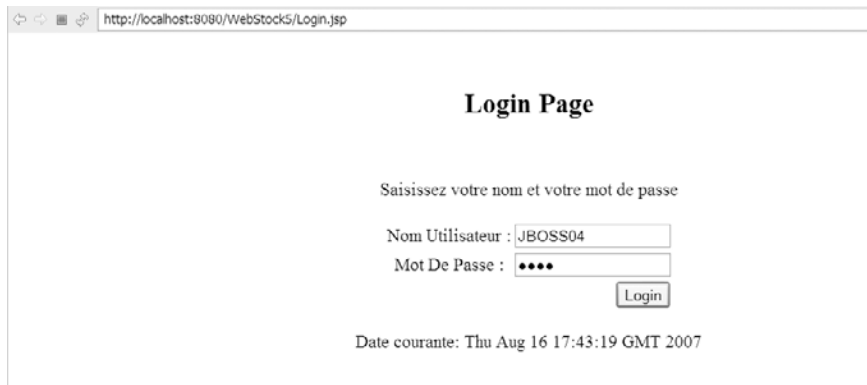
Éditeur de JSP

Vous avez la possibilité de personnaliser l'éditeur de JSP Web Tools (coloration des commentaires, des éléments de scripting, etc.) *via* le menu Preferences d'Eclipse en sélectionnant Web And XML puis le sous-menu JSP Files.

7. Sauvegardez votre page (Ctrl+S), et déployez-la sur le serveur JBoss comme lors des étapes précédentes. Votre page doit ressembler à celle illustrée à la figure 7.7.

Figure 7.7

Page d'authentification de webstock



8. Sous le même dossier, créez votre seconde page JSP LoginSuccess.jsp, qui, en cas de succès de l'identification, routera l'utilisateur vers cette page :

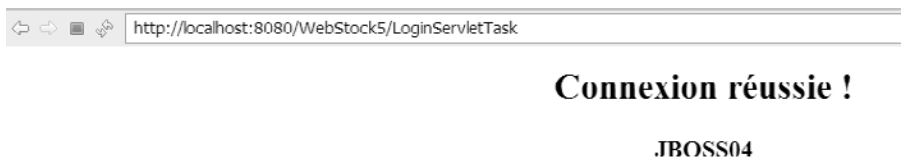
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
➥"http://www.w3.org/TR/html4/loose.dtd">
<%@ page import="java.util.Calendar"%>

<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":
➥"+request.getServerPort()+path+"/";
%>
<HTML>
<HEAD>
<TITLE>Page d'Identification</TITLE>
</HEAD>
<BODY>
<p>
<CENTER>

<H2>Connexion réussie ! </H2>

</CENTER>
<p align="center"><STRONG><%= request.getParameter("userName") %><font size="5"
    style="BACKGROUND-COLOR: #ffffff"></font></STRONG></p>
```

```
<p><br>
<br>

</BODY>
</HTML>
```

Le résultat de la page générée lors d'une connexion doit s'afficher comme illustré à la figure 7.8.

Figure 7.8

Page Login-Success.jsp après réussite d'une connexion



Création de la servlet LoginServletTask

La ressource servlet LoginServletTask est chargée de la validation du formulaire.

Pour la créer, procédez comme suit :

1. Créez un package com.webstock.chap07 (*via* les options New et Package après sélection du dossier Java Resources).
2. Sélectionnez le package dans la vue explorateur de projets, puis cliquez à partir du menu contextuel sur New et Other, et sélectionnez l'assistant Web et Servlet. Cliquez sur Next.
3. Dans le champ Class name, entrez LoginServletTask, puis cliquez sur Finish. Le code de la servlet avec les méthodes doGet/doPost génériques est automatiquement généré.
4. Saisissez le code suivant dans cette servlet :

```
import java.sql.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * @author djafaka
 *
 * TODO Pour changer le modèle de ce commentaire de type généré, allez à :
 * Fenêtre - Préférences - Java - Style de code - Modèles de code
```

```
 */
public class LoginServletTask extends HttpServlet {

/**
 * Constructor of the object.
 */
public LoginServletTask() {
    super();
}

/**
 * Destruction of the servlet. <br>
 */
public void destroy() {
    super.destroy(); // Just puts "destroy" string in log
    // Put your code here
}

/**
 * The doGet method of the servlet. <br>
 *
 * This method is called when a form has its tag value method equals to get.
 *
 * @param request the request send by the client to the server
 * @param response the response send by the server to the client
 * @throws ServletException if an error occurred
 * @throws IOException if an error occurred
 */
public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        sendLoginForm(response, false);
      }
private void sendLoginForm(HttpServletResponse response,
        boolean withErrorMessage)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Login</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<CENTER>");

        if (withErrorMessage)
          out.println("Login failed. Please try again.<BR>");

        out.println("<BR>");
        out.println("<BR><H2>Login Page</H2>");
        out.println("<BR>");
        out.println("<BR>Please enter your user name and password.");
        out.println("<BR>");
```

```
        out.println("<BR><FORM METHOD=POST>");
        out.println("<TABLE>");
        out.println("<TR>");
        out.println("<TD>User Name:</TD>");
        out.println("<TD><INPUT TYPE=TEXT NAME=userName></TD>");
        out.println("</TR>");
        out.println("<TR>");
        out.println("<TD>Password:</TD>");
        out.println("<TD><INPUT TYPE=PASSWORD NAME=password></TD>");
        out.println("</TR>");
        out.println("<TR>");
        out.println("<TD ALIGN=RIGHT COLSPAN=2>");
        out.println("<INPUT TYPE=SUBMIT VALUE=Login></TD>");
        out.println("</TR>");
        out.println("</TABLE>");
        out.println("</FORM>");
        out.println("</CENTER>");
        out.println("</BODY>");
        out.println("</HTML>");
      }


/**
 * The doPost method of the servlet. <br>
 *
 * This method is called when a form has its tag value method equals to post.
 *
 * @param request the request send by the client to the server
 * @param response the response send by the server to the client
 * @throws ServletException if an error occurred
 * @throws IOException if an error occurred
 */
 public void doPost(HttpServletRequest request,
         HttpServletResponse response)
         throws ServletException, IOException {

         String userName = request.getParameter("userName");
         String password = request.getParameter("password");
         System.out.println("doPost"+userName);
         if (login(userName, password)) {
           RequestDispatcher rd =
             request.getRequestDispatcher("LoginSuccess.jsp");
           rd.forward(request, response);
         }
         else {
           sendLoginForm(response, true);
         }
       }
 boolean login(String userName, String password) {
    try {
      Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
      Connection con =
        DriverManager.getConnection("jdbc:derby ","derby","");
      System.out.println("got connection");

      Statement s = con.createStatement();
      String sql = "SELECT USERID FROM webstock.webstockaccess" +
```

```
        " WHERE NomUser='" + userName + "'" +
        " AND MotDePasse='" + password + "'";

      ResultSet rs = s.executeQuery(sql);
      if (rs.next()) {
        rs.close();
        s.close();
        con.close();
        return true;
      }
      rs.close();
      s.close();
      con.close();
    }
    catch (ClassNotFoundException e) {
      System.out.println(e.toString());
    }
    catch (SQLException e) {
      System.out.println(e.toString());
    }
    catch (Exception e) {
      System.out.println(e.toString());
    }
    return false;
  }


/**
 * Initialization of the servlet. <br>
 *
 * @throws ServletException if an error occure
 */
public void init() throws ServletException {
    // Put your code here
}

}
```

La servlet LoginServletTask essaie de faire correspondre les valeurs saisies dans le formulaire précédant le login et le mot de passe avec celles des colonnes NomUser et MotDePasse stockées dans la table WebStockAccess.

Lorsque l'utilisateur saisit dans son navigateur l'URL http://localhost:8080/WebStock/login.jsp et qu'il clique sur le bouton Login, la méthode doGet est invoquée. Celle-ci appelle la méthode privée sendLoginForm, qui envoie la page HTML à l'utilisateur pour se connecter.

Cette dernière méthode possède deux arguments : un objet HttpServletResponse, que la méthode peut utiliser pour envoyer le résultat au navigateur, et un type booléen, withErrorMessage. Ce booléen est un drapeau qui indique si le message d'erreur doit être envoyé avec le formulaire. Ce message d'erreur informe l'utilisateur que le précédent login a échoué.

Lorsque la servlet est appelée pour la première fois, aucun message n'est envoyé. Cela explique que la valeur false soit transmise en argument à la méthode sendLoginForm (voir l'extrait de code précédant la méthode doGet).

Après l'envoi du formulaire à la servlet *via* l'attribut ACTION="/webstock/LoginServletTask" (voir code de la JSP login.jsp précédente) et la transmission des paramètres à la servlet, la méthode login(userName, password) est appelée *via* la méthode doPost de la servlet. Cette méthode retourne true ou false en fonction du résultat de la requête dans la base webstock par le biais de la table WebStockAccess.

En cas de succès, la requête est transmise à une JSP LoginSuccess pour un traitement associé à une connexion réussie au système webstock. En cas d'échec, la méthode doPost appelle à nouveau la méthode sendLoginFom avec un message d'erreur sendLoginForm (response, true).

Gestion des commandes avec Web Tools et les patterns

Vous allez maintenant construire les composantes de la partie gestion des commandes de l'application webstock à l'aide des assistants du projet Web Tools.

Au chapitre suivant, vous utiliserez un framework adapté au design d'applications 3-tiers, qui facilitera l'interaction entre les couches de présentation, la logique métier et la persistance des données avec l'aide des patterns Commande et Singleton.

Cette partie simplifiée de l'implémentation de l'application webstock concerne deux tables du modèle de données, les tables `Commande` et `Client`, et est constituée des cas d'utilisation suivants :

- Les clients doivent être enregistrés sur le site pour placer leurs commandes.
- Les clients peuvent commander des articles.
- Les clients peuvent voir leur commande.
- Les administrateurs peuvent afficher la liste des clients enregistrés.

Le système sera implémenté en utilisant un modèle de programmation fondé sur les servlets et le design pattern Commande.

La figure 7.9 décrit le design de cette partie de l'application.

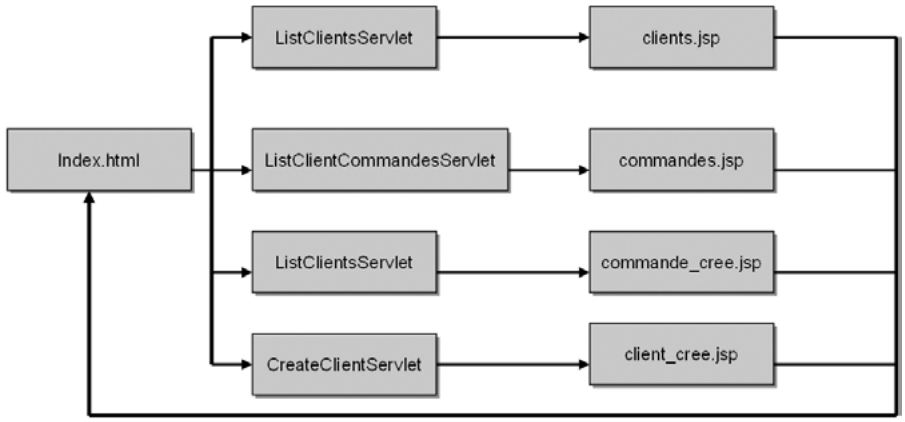


Figure 7.9

Cinématique de la gestion des commandes

Design de la partie métier

Avant toute chose, vous devez concevoir les classes métier qui supporteront le développement de l'application Web. La figure 7.10 rappelle la portion du domaine métier de l'étude webstock concernée et les relations de dépendance associées.

À la partie III de l'ouvrage, vous reviendrez plus amplement sur ces éléments du modèle lors de l'étude du mapping objet-relationnel avec l'API JPA ainsi que sur les beans entité POJO.

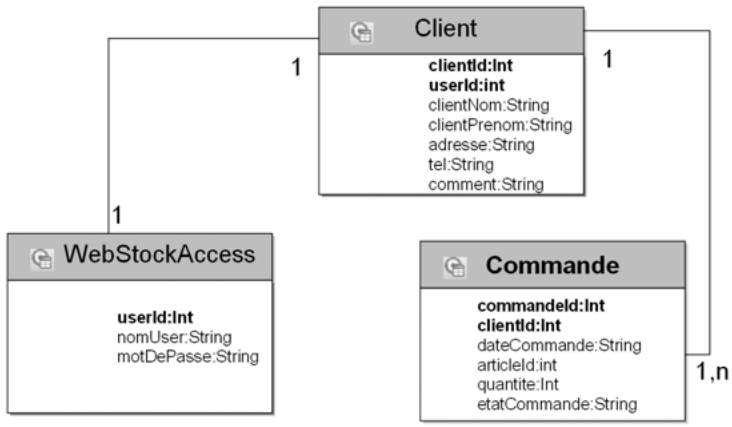


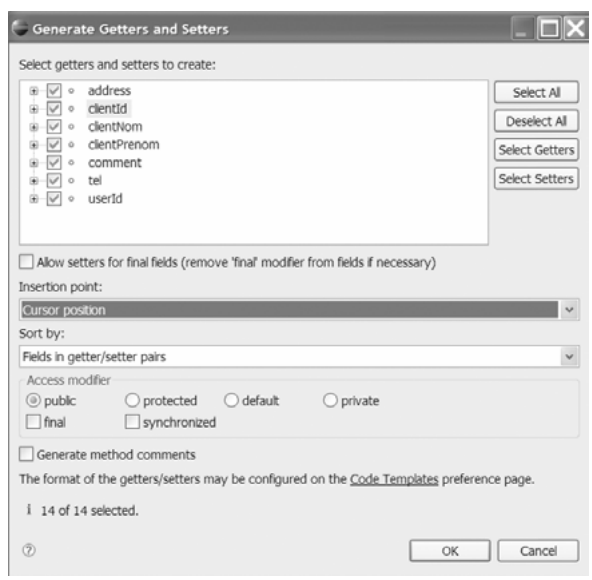
Figure 7.10

Design du modèle de données de la gestion des commandes

Vous utiliserez l'assistant de génération de code fourni par Eclipse pour la génération de code automatique des getters et setters à partir des champs de la classe concernée : par exemple la classe `Commande` (options de menu contextuel Source et Generate Getters and Setters) illustrée à la figure 7.11.

Figure 7.11

Assistant de génération de getters/-setters Eclipse



Faites de même avec les deux autres classes du modèle considéré, que vous regrouperez pour des raisons de lisibilité et de bonne pratique logicielle dans un package com.webstock.chap07.domain que vous aurez préalablement créé à l'aide de l'assistant de création de package sous Java Resources.

Création des classes d'accès aux données selon les design patterns Commande et Singleton

Vous allez créer des classes d'accès aux données en utilisant le design pattern Commande, qui permet aux programmes et aux objets GUI d'être complètement séparés des actions qu'ils initient.

Lorsque l'interface utilisateur reçoit une commande, elle demande à l'objet Commande de s'occuper de toutes les actions associées, sachant que la règle fondamentale de ce pattern est de ne pas se soucier des tâches qui seront exécutées.

L'interface du pattern Commande pour la gestion des données est la suivante :

```
package com.webstock.chap07.command;

import java.sql.Connection;
import java.sql.SQLException;

public interface DatabaseCommand {
public Object executeDatabaseOperation(Connection conn) throws SQLException ;
}
```

Les classes CRUD (Create/Read/Update/Delete) suivantes implémentent le pattern Commande représenté par l'interface DatabaseCommande pour effectuer les opérations dans la base et insérer les enregistrements dans les tables `Commande` et `Client` :

```
public class CreateClient implements DatabaseCommand
public class CreateCommande implements DatabaseCommand
public class ListClients implements DatabaseCommand
public class ListCommandesClients implements DatabaseCommand
```

Pour permettre l'exécution de vos classes `Commande`, vous devez créer une classe permettant d'accéder à la source de données, obtenir une connexion SQL et exécuter une commande d'accès aux données particulières.

Pour ce faire, vous vous aiderez d'un second pattern incontournable, le pattern Singleton, qui appellera la classe CommandExecution, garantissant l'instanciation à un seul objet :

```
private static CommandExecutor myOnlyInstance = null;

public static CommandExecution getInstance() throws NamingException {
    if (myOnlyInstance == null) {
        myOnlyInstance = new CommandExecution();
    }
    return myOnlyInstance;
}
```

L'exécution proprement dite d'une classe particulière `Commande` s'effectue par le biais de l'invocation suivante :

```
Object o = CommandExecution.getInstance().executeDatabaseCommand
("Instance particulière d'un objet commande d'accès aux données")
```

Pour invoquer l'objet `Commande` d'affichage de la liste des clients, vous aurez, par exemple :

```
…
try {

        ArrayList<Client> list = (ArrayList<Client>)CommandExecution.getInstance()
        ➥.executeDatabaseCommand(new command.ListClients());
        request.setAttribute("clients", list);
        RequestDispatcher rd = getServletContext().getRequestDispatcher("/clients.jsp");
        rd.forward(request, response);
    } catch (Exception e) {
        throw new ServletException(e);
    }
}
```

L'accès à la source de données s'effectue par un lookup du contexte JNDI, comme le montre l'implémentation de la méthode `getDataSource()` suivante, la référence à cette ressource étant stockée dans le fichier de source de données associé (voir configuration de la source de données du serveur JBoss) :

```
public DataSource getDataSource() throws NamingException {

    if (ds == null) {

        InitialContext ctx = new InitialContext();
        Context envCtx = (Context) ctx.lookup("java:comp/env");
        ds = (DataSource) envCtx.lookup("jdbc/WebStockDB");

    }

    return ds;
}
```

L'exécution d'une commande d'accès aux données particulières utilise la méthode `executeDatabaseCommand()` :

```
public Object executeDatabaseCommand(DatabaseCommand c) throws Exception {

    Connection conn = null;
    try {
        conn = getConnection();
        Object o = c.executeDatabaseOperation(conn);
        return o;
    } catch (SQLException e) {
        throw e;
    } catch (NamingException ne) {
        throw ne;
    } finally {
        if (conn != null) conn.close();
    }
}
```

À présent que les bases de votre conception Java ont étés implémentés, vous pouvez passer à la mise en œuvre en utilisant l'outillage Web Tools.

Développement des classes CRUD pour les objets métier *Client* et *Commande*

À présent que l'ossature pour l'invocation des méthodes métier a été développée pour les beans métier `Commande` et `Client`, passez aux opérations CRUD proprement dites sur ces classes.

Vous verrez les opérations de création/affichage des enregistrements associés à la classe `Client`, qui vaut aussi pour l'objet métier `Commande` (le code source complet est disponible sur la page Web dédiée à l'ouvrage).

Voici le code de création et d'affichage associé à la classe métier `Client` :

```
package com.webstock.chap07.command;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.webstock.chap07.domain.Client;

public class CreateClient implements DatabaseCommand {

private Client clt;

public CreateClient(Client c) {
    this.clt = c;
}

public Object executeDatabaseOperation(Connection conn) throws SQLException {

    PreparedStatement stmt = conn.prepareStatement("INSERT INTO CLIENT (CLIENTID, USERID,
    ➥CLIENTNOM, CLIENTPRENOM, ADDRESSE,TEL,COMMENT) VALUES ( ?, ?, ?, ?, ?, ?, ?)");
    stmt.setInt(1, clt.getClientId());
    stmt.setString (2, clt.getUserId());
    stmt.setString (3, clt.getClientNom());
    stmt.setString (4, clt.getClientPrenom());
    stmt.setString (5, clt.getAdresse());
    stmt.setString (6, clt.getTel());
    stmt.setString (7, clt.getComment());
    int lignes_maj = sta.executeUpdate();
    stmt.close();
    return new Integer(lignes_maj);
}

}


package com.webstock.chap07.command;

import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;
import java.util.ArrayList;
import com.webstock.chap07.domain.Client;

/**
 * Affiche les clients existants dans la base Webstock
 */
```

```
public class ListClients implements DatabaseCommand {

public Object executeDatabaseOperation(Connection conn) throws SQLException {


    ArrayList<Client> list = new ArrayList<Client>();
    Statement sta = conn.createStatement();
    ResultSet rs = sta.executeQuery("SELECT CLIENTID, CLIENTNOM, CLIENTPRENOM,
    ➥ADRESSE, TEL, COMMENT FROM CLIENT");
    while(rs.next()) {
        Client clt = new Client();
        clt.setClientId(rs.getInt(1));
        clt.setClientNom(rs.getString(2));
        clt.setClientPrenom(rs.getString(3));
        clt.setAdresse(rs.getString(4));
    clt.setTel(rs.getString(5));
        clt.setComment(rs.getString(6));

        list.add(clt);
    }

    rs.close();
    sta.close();

    return list;


}

}
```

Création des composants servlets et JSP avec les assistants Web Tools

Selon le modèle classique MVC (modèle, vue, contrôleur), vous avez implémenté à la section précédente la couche modèle constituée d'objets du domaine et des classes `Commande`. La couche contrôleur sera implémentée sous la forme d'un composant servlet et la vue sous la forme de pages JSP.

Web Tools fournit un ensemble complet d'assistants et d'éditeurs pour faciliter la gestion des servlets et des JSP. Nous supposons créé un projet Web de type Web dynamique, appellé WebStock.

1. Créez un package sous le répertoire src créé par défaut, et appelez-le com.webstock.chap07.servlet
2. Lancez l'assistant de création de servlet en sélectionnant File, New, Other et Web-Servlet, puis saisissez ListClientsServlet comme nom de servlet.
3. Cliquez sur Next, et spécifiez une description optionnelle pour la servlet, ainsi que des paramètres d'initialisation éventuels lors du chargement de la servlet (méthode `init()` de l'objet ServletConfig) et le mapping de l'URL (/ListClientsServlet par défaut).
4. Cliquez sur Next.

5. À cette étape, l'assistant propose un certain nombre d'options concernant les interfaces que la servlet peut implémenter (par défaut, l'interface javax.servlet.Servlet) ainsi que les méthodes qui sont automatiquement générées lors de la création de la servlet. Cochez la méthode doGet comme illustré à la figure 7.12.

Figure 7.12

Assistant de création de servlet et de définition des méthodes

6. Cliquez sur Finish pour générer la servlet. Le code suivant de la servlet est généré (notez que le fichier descripteur web.xml de l'application Web est automatiquement mis à jour) :

```
<display-name>WebStock</display-name>
 <servlet>
   <description>Affichage des clients</description>
   <display-name>ListClientsServlet</display-name>
   <servlet-name>ListClientsServlet</servlet-name>
  <servlet-class>
  com.webstock.chap06.servlet.ListClientsServlet
  </servlet-class>
 </servlet>
 <servlet-mapping>
   <servlet-name>ListClientsServlet</servlet-name>
   <url-pattern>/ListClientsServlet</url-pattern>
 </servlet-mapping>
</web-app>
```

Code des servlets de support à la gestion des objets *Commande* et *Client*

Terminez par le code associé aux servlets de gestion des clients, respectivement ListClientsServlet pour l'affichage des clients et CreateClientServlet pour la création de

l'instance client (le code complet pour la gestion des commandes, ListCommandesServlet et CreateCommandeServlet est disponible sur la page Web dédiée à l'ouvrage) :

```
package com.webstock.chap07.servlet ;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.RequestDispatcher;
import java.util.ArrayList;

import com.webstock.chap07.domain.*;
import com.webstock.chap07.command.*;

/**
 * Classe d'implementation pour la servlet ListClients
 *
 */
 public class ListClientsServlet extends javax.servlet.http.HttpServlet {
    /* (non-Java-doc)
 * @see javax.servlet.http.HttpServlet#HttpServlet()
 */

/* (non-Java-doc)
 * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest request,
 ➥HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response)
➥throws ServletException, IOException {

    try {

        ArrayList<Client> list = (ArrayList<Client>)CommandExecution.getInstance()
        ➥.executeDatabaseCommand(new com.webstock.chap07.command.ListClients());
        request.setAttribute("clients", list);
        RequestDispatcher rd = getServletContext().getRequestDispatcher("/clients.jsp");
        rd.forward(request, response);

    } catch (Exception e) {
        throw new ServletException(e);
    }
}

/* (non-Javadoc)
 * @see javax.servlet.GenericServlet#init()
 */
public void init() throws ServletException {
    // TODO Auto-generated method stub
    super.init();

    try {
        CommandExecution.getInstance();
    } catch (Exception e) {
        throw new ServletException(e);
    }
}
}
```

Comme vous pouvez le remarquer sur ce dernier code, l'attribut requête "clients" est positionné à un objet générique ArrayList et l'objet RequestDispatcher est utilisé pour renvoyer la réponse à la page JSP clients.jsp. Enfin, dans la méthode `init()` de la servlet ListClientsServlet, une instance de la commande CommandExecution est ajoutée afin de récupérer la source de données (DerbyDS) et la « cacher » pour les besoins futurs de gestion des connexions à la base WebStockDB.

```
package com.webstock.chap07.servlet;

import java.io.IOException;
import java.sql.Timestamp;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.RequestDispatcher;

import com.webstock.chap07.command.CommandExecution;
import com.webstock.chap07.command.DatabaseCommand;
import com.webstock.chap07.command.CreateClient;
import com.webstock.chap07.domain.Client;

/**
 * Servlet implementation class for Servlet: CreateCustomerServlet
 *
 */
 public class CreateClientServlet extends javax.servlet.http.HttpServlet implements
 ➥javax.servlet.Servlet {

/* (non-Java-doc)
 * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest request,
 ➥HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response)
➥throws ServletException, IOException {

    try {

        // creation du client
        String prenom = request.getParameter("prenom");
        String nom = request.getParameter("nom");
        int userid = request.getParameter("userid");
        String tel = request.getParameter ("tel");
        String adresse = request.getParameter ("adresse");
        String comment = request.getParameter("comment");
        int clientId = Math.abs((int)System.currentTimeMillis());
        Client c = new Client();
        c.setClientId(clientId);
        c.setPrenom (prenom);
        c.setNom(nom);
        c.setUserId(userid);
        c.setAdresse(adresse);
        c.setTel(tel);
        c.setComment(comment);

        DatabaseCommand command = new CreateClient(c);
```

```
        int lignes = (Integer)CommandExecution.getInstance()
        ➥.executeDatabaseCommand(command);
        RequestDispatcher rd = getServletContext().getRequestDispatcher
        ➥("/client_cree.jsp");
        rd.forward(request, response);

    } catch (Exception e) {
        throw new ServletException(e);
    }

}

/* (non-Javadoc)
 * @see javax.servlet.GenericServlet#init()
 */
public void init() throws ServletException {
    // initialize servlet
    super.init();

    try {
        CommandExecution.getInstance();
    } catch (Exception e) {
        throw new ServletException(e);
    }

}
}
```

Le code de la page clients.jsp créé par l'assistant de création de page JSP de Web Tools conclut la gestion de l'affichage des clients :

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
➥pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Liste des clients<title>
</head><body>

<jsp:useBean id="clients" type="java.util.ArrayList<domain.Client>" scope="request"/>

<b>Liste des clients enregistrés dans la base WebStock:</b><br>
<table border="1">
<tr>
<th>Id</th>
<th>Nom</th>
<th>Prenom</th>
<th>Telephone</th>
<th>Commentaire</th>
</tr>
<% for(com.webstock.chap07.domain.Client c : clients) { %>
<tr>
<td><%= c.getClientId() %></td>
<td><%= c.getClientNom() %></td>
<td><%= c.getClientPrenom() %></td>
<td><%= c.getTel() %></td>
```

```
<td><%= c.getAdresse %></td>
<td><%= c.getComment() %></td>

<td><a href="/WebStock/ListCommandes?clientId=<%= c.getClientId() %>">Commandes</a>
➥</td>
<% } %>
</body>
</html>
```

Remarquez dans ce dernier extrait l'utilisation de certaines fonctions de J2SE 5.0, comme l'utilisation des génériques et de la boucle for.

Configuration de la source de données du serveur JBoss

Avant de démarrer le serveur JBoss, vous devez ajuster sa configuration pour qu'il fonctionne avec le SGBD Derby, l'ajustement de la configuration devant être effectué dans le répertoire de configuration du serveur (JBOSS_HOME\conf, soit, dans notre configuration particulière, C:\jboss-4.2.0.GA\server\default\conf).

Vous devez également mettre à jour la référence à la source de données pour qu'elle fonctionne avec la base Derby, JBoss étant par défaut configuré avec la base de données HypersonicDB.

Bonne pratique

Une bonne pratique pour ajuster les fichiers de configuration et gérer ses configurations de manière pratique consiste à copier ces derniers dans un répertoire et à les ajuster en fonction de vos besoins. JBoss peut être démarré avec le nom du répertoire de configuration (sans le chemin complet) comme premier paramètre et l'option –c (run –c maconfig, par exemple).

1. Copiez le driver JDBC du SGBD Derby (derby.jar), dans le répertoire lib de la configuration.
2. Créez un fichier derby-ds.xml dans le répertoire deploy avec la configuration suivante :

```
<datasources>
   <local-tx-datasource>
      <jndi-name>DerbyDS</jndi-name>
      <connection-url>
jdbc:derby:${jboss.server.data.dir}${/}derby${/}WebStockDB;create=true
      </connection-url>
      <!-- The driver class -->
      <driver-class>org.apache.derby.jdbc.EmbeddedDriver</driver-class>
      <user-name>derby</user-name>
      <password></password>
      <min-pool-size>5</min-pool-size>

      <max-pool-size>20</max-pool-size>
      <idle-timeout-minutes>5</idle-timeout-minutes>
      <track-statements/>
      <depends>jboss:service=Derby</depends>
   </local-tx-datasource>
</datasources>
```

Déploiement de l'application sur le serveur JBoss

À ce stade, votre application peut être publiée et déployée sur votre serveur JBoss par l'assistant de publication de Web Tools.

Pointez votre navigateur sur l'URL http://localhost:8080/Webstock/index.html. Si tout se passe bien, une page d'accueil vous propose un certain nombre d'options, comme l'affichage des clients enregistrés dans la base webstock illustré à la figure 7.13.

Figure 7.13

Page d'affichage de la liste des clients

Liste des clients enregistrés dans la base WebStock:

Id	Nom	Prenom	Adresse	Telephone	Commentaire	Commande
CL001	ALPHONSO	PIERRE	2 Rue des Amandiers NANTERRE	0134567889	Client régulier	CLOO1
CL002	DUPONT	François	2 Rue des PCs PARIS	0145234455	Client occasionnel	CLO02

accueil

En résumé

Vous avez vu dans ce chapitre que la mise en œuvre de l'outillage Web Tools pour un développement sans framework particulier exige une certaine rigueur dans la construction de l'ossature sous-tendant le développement Web.

Le chapitre suivant se penche sur Seam, un puissant framework destiné à compléter et faciliter l'outillage à la disposition du développeur JBoss.