

Chapitre 2 : Gestion d'un pool de connexions SGBD par Tomcat

Rappel : En java , Il est possible se connecter à une base de données de différentes manières :

- 1) Connection via JDBC.ODBC (accès à une base access, via un lien ODBC au préalable défini dans le gestionnaire ODBC du système)
- 2) Connection via un pilote JDBC
- 3) Connection via JDBC-Datasource (sous Tomcat et base mySQL) (definition du Datasource dans le fichier server.xml et récupération du datasource via JNDI.)
- 4) Connection via DataSources (dans un serveur Jboss, Weblogic en paramétrant un DataSource, qui pointe sur une base de données et en faisant référence à l'utilisation de JNDI.

Ce chapitre vous présente le paramétrage et l'utilisation d'un pool de connexions SGBD avec Tomcat

1. Introduction

Il existe des outils de mapping objet (comme JDO, Hibernate) qui facilitent le stockage des objets dans une base de données. Toutefois ils sont plus complexes à mettre en place, et leur apprentissage demande de déjà connaître le fonctionnement de l'API JDBC. Les Servlets travaillent alors directement avec la base de données. Lors de l'accès à une base de données depuis une Servlet, l'établissement de la connexion peut prendre quelques secondes. Ce délai augmente le temps de réponse de votre Servlet.

La gestion d'un pool de connexions par Tomcat permet de réduire ce temps puisque les connexions à la base de données sont déjà établies et gérées par Tomcat qui fournit à la Servlet des objets DataSource. Les pools de connexions dans Tomcat utilisent les classes du pool de connexions DBCP (Database Connection Pools) de [Jakarta Commons](#). Toutefois il est possible d'utiliser n'importe quel autre pool qui implémente l'interface [javax.sql.DataSource](#). Dans ce chapitre, nous allons voir comment **enregistrer un pilote de base de données auprès de JNDI** pour le rendre accessible aux différentes applications hébergées par un serveur d'application Tomcat. **JNDI** est un service d'annuaire qui permet à un composant d'accéder à des objets gérés par son serveur d'application.

2. Configuration minimale

- le pilote JDBC mysql-connector-java.jar placé dans \$CATALINA_HOME/common/lib
- La configuration pour l'utilisation du pool de connexion se fait en deux endroits :
 - dans le fichier web.xml du contexte
 - dans le fichier server.xml du répertoire conf de Tomcat

2.1. Le fichier web.xml

Ce fichier est le descripteur de l'application web. On y déclare le nom JNDI (Java Naming and Directory Interface) sous lequel on cherchera l'objet DataSource. Par convention, ces noms suivent le sous-contexte jdbc (relatif au contexte de nommage standard java:comp/env qui est la racine de toutes les ressources fournies). Cette déclaration se fait après la section servlet-mapping.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>TutoPool</display-name>
  <servlet>
    <servlet-name>TutoPool</servlet-name>
    <servlet-class>tutorial.TutoPool</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>TutoPool</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
```

```

<welcome-file>index.html</welcome-file>
<welcome-file>index.htm</welcome-file>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>

    <resource-ref>
        <description>
            reference a la ressource BDD pour le pool
        </description>
        <res-ref-name>jdbc/TutoPool</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>

    <resource-ref>
        <description>
            reference a la ressource BDD pour le pool2
        </description>
        <res-ref-name>jdbc/TutoPool2</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>

</web-app>

```

2.2. Le fichier server.xml

Il s'agit ici de configurer la "resource factory" de Tomcat. Cette configuration peut aussi être faite dans le fichier META-INF\context.xml de l'archive war utilisée pour le déploiement de l'application web sur les versions 5.x de Tomcat. On a repris ici toute la partie relative au contexte de l'application web TutoPool. Cette section est à placée avec les autres déclarations de contexte entre les balises <Host> du fichier server.xml.

```

<Context path="/TutoPool"
    reloadable="true"
    docBase="\TutoPool" >
    <Resource
        name="jdbc/TutoPool"
        auth="Container"
        type="javax.sql.DataSource"/>
    <ResourceParams name="jdbc/TutoPool">
        <parameter>
            <name>username</name>
            <value>user</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>password</value>
        </parameter>
        <parameter>
            <name>driverClassName</name>
            <value>org.gjt.mm.mysql.Driver</value>
        </parameter>
        <parameter>
            <name>url</name>
            <value>jdbc:mysql://localhost:3306/base</value>
        </parameter>
    </ResourceParams>
</Context>

```

On retrouve dans la section concernant le Context, la déclaration de la ressource puis les paramètres minimums nécessaires à cette ressource : le nom d'utilisateur, le mot de passe, le driver et la chaîne de connexion à la base de données.

Le nombre de connexions

Trois paramètres permettent d'influencer la gestion du nombre de connexions dans le pool :

- **maxActive** : le nombre maximum de connexions qui peuvent être allouées depuis le pool. Sa valeur par défaut est 8, une valeur de 0 indique "sans limite". Il faut être sûr que la base de données autorisera ce nombre de connexions.
- **maxIdle** : le nombre maximum de connexions inactives qui peuvent être dans le pool. Sa valeur par défaut est de 8, une valeur de 0 indique "sans limite"
- **maxWait** : le temps d'attente maximum en millisecondes pour l'obtention d'une connexion. Passé ce délai, une exception est levée. Sa valeur par défaut est -1 qui correspond à un temps infini

Configuration 2 : Pour Tomcat 6

1) context.xml

```
<Context>

<!-- Default set of monitored resources -->
<WatchedResource>WEB-INF/web.xml</WatchedResource>

  <Resource name="jdbc/TutoPool"
    auth="Container"
    type="javax.sql.DataSource"
    username="root"
    password=""
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/base1"
    maxActive="8"
    maxIdle="4"
    maxWait= "1000" />

  <Resource name="jdbc/TutoPool2"
    auth="Container"
    type="javax.sql.DataSource"
    username="root"
    password=""
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/base2"
    maxActive="8"
    maxIdle="4"/>

  <Resource name="jdbc/postgres"
    auth="Container"
    type="javax.sql.DataSource"
    username="postgres"
    password="postgres"
    driverClassName="org.postgresql.Driver"
    url="jdbc:postgresql://localhost:5432/base3"
    maxActive="8"
    maxIdle="4"
    maxWait=-1 />

</Context>
```

Configuration 3 : Server.xml

Ajouter ce qui suit entre le tag `</Context>` et le tag `</Host>` qui termine la definition *localhost*

```
<Context docBase="appliTest" path="/appliTest" reloadable="true" crossContext="true">

<Logger
  className="org.apache.catalina.logger.FileLogger"      prefix="localhost_DBTest_log."
  suffix=".txt"      timestamp="true"/>

<Resource
  name="jdbc/maBase"
  auth="Container"
  type="javax.sql.DataSource"/>

<ResourceParams name="jdbc/maBase">
  <parameter>
    <name>factory</name>
    <value>          org.apache.commons.dbcp.BasicDataSourceFactory
    </value>
  </parameter>

<!--
Nombre maximum de connexions à la base que supportera pool.
Il faut être sûr d'avoir configuré mysqld max_connections
assez largement pour subvenir aux besoins de connexion à la base.

Pour avoir un nombre illimité de connexions mettre 0.
-->

  <parameter>
    <name>maxActive</name>
    <value>100</value>
  </parameter>

<!--
Nombre maximum de connexions inactives vers la base de données que le pool conservera.

Pour un nombre illimités de connexions mettre -1.

Pour plus d'information voir la documentation de DBCP et la configuration du paramètre
minEvictableIdleTimeMillis
-->

  <parameter>
    <name>maxIdle</name>
    <value>30</value>
  </parameter>

<!--
Temp maximun pour qu'une connexion soit active en ms, ici j'ai mis 2 secondes.
Une exception est levée si ce timeout est dépassé.
Pour un temps d'attente infini mettre -1
-->

  <parameter>
    <name>maxWait</name>
    <value>20000</value>
  </parameter>
```

```

<!--
login et mot de passe pour se connecter à la base Mysql
-->

<parameter>
  <name>username</name>
  <value>root</value>
</parameter>

<parameter>
  <name>password</name>
  <value></value>
</parameter>

<!-- Classe du connecteur JDBC officiel pour Mysql
-->

<parameter>
  <name>driverClassName</name>
  <value>com.mysql.jdbc.Driver</value>
</parameter>

<!-- L'url JDBC pour la connexion à la base Mysql.
-->

<parameter>
  <name>url</name>
  <value>jdbc:mysql://localhost/baseMysql</value>
</parameter>

</ResourceParams>

</Context>

```

3. Le code de la Servlet

Une utilisation typique des pools de connexions se fait en récupérant l'objet DataSource dans l'init de la Servlet. Les connexions sont ensuite obtenues par la méthode getConnection(). Il ne faut pas oublier de libérer ces connexions après utilisation.

Cette Servlet d'exemple se contente d'afficher à l'écran le contenu de la table table1.

Exemple Servlet 1 :

```

package tutorial;
import java.io.*;
import java.sql.*;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.sql.*;
public class TutoPool extends HttpServlet {
    private DataSource ds; //la source de données
    protected void doGet(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");

```

```

        PrintWriter out = response.getWriter();
        out.println("<html><head></head>");
        out.println("<body>");
        Connection con=null;
        Statement s=null;
        ResultSet rs=null;
        try {
            //récupération de la Connection depuis le DataSource
            con = ds.getConnection();
            s = con.createStatement();
            rs = s.executeQuery("SELECT * FROM table1");
            while (rs.next()) {
                out.println(rs.getString(1) + "    ");
                out.println(rs.getString(2) + "<br/>");
            }
        } catch (SQLException e) {
            response.sendError(500, "Exception sur l'accès à la BDD " + e);
        } finally {
            if (rs != null)
            {
                try {
                    rs.close();
                } catch (SQLException e) {}
                rs = null;
            }
            if (s != null) {
                try {
                    s.close();
                } catch (SQLException e) {}
                s = null;
            }
            if (con != null) {
                try {
                    con.close();
                } catch (SQLException e) {}
                con = null;
            }
        }
        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    public void init() throws ServletException {
        try {
            //récupération de la source de donnée
            Context initCtx = new InitialContext();
            ds = (DataSource) initCtx.lookup("java:comp/env/jdbc/TutoPool");
            // ds = (DataSource) initCtx.lookup("java:comp/env/jdbc/postgres");

        } catch (Exception e) {
            throw new UnavailableException(e.getMessage());
        }
    }
}

```

Le résultat est visible par l'appel dans un navigateur de <http://localhost:8080/TutoPool/> (8080 étant le port qui est configuré par défaut lors de l'installation de Tomcat).



L'appel à la méthode `Connection.close()` ne ferme pas vraiment la connexion avec la base de données mais la libère pour qu'elle retourne dans le pool.

Exemple 2 JSP:

```

<%@ 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="javax.naming.Context"%>
<%@page import="javax.naming.InitialContext"%>
<%@page import="javax.sql.DataSource"%>
<%@page import="java.sql.Connection"%>
<%@page import="java.sql.ResultSet"%>

<html>
<head>
    <title>Le nombre de clients</title>
</head>
<body>
<%
    String jndiName = "java:comp/env/jdbc/TutoPool";
    Context ctx = new InitialContext();
    DataSource ds = (DataSource) ctx.lookup(jndiName);
    Connection conn = ds.getConnection();
    String sql = "SELECT count(*) FROM Client";
    ResultSet rs = conn.createStatement().executeQuery(sql);
    rs.next();
    int nbLigne = rs.getInt(1);
%>

Nombre : <%=nbLigne %>
</body>
</html>

```

Accès à une base de données Mysql sans utiliser JNDI

Il est recommandé de créer l'objet Connection par DataSource implementation class, com.mysql.jdbc.jdbc2.optional.MysqlDataSource. Voici un simple programme permettant de créer une connection à une base de données en utilisant le DataSource class sans utiliser JNDI services:

```

/**
 * MySqlDataSource.java
 * Copyright (c) 2007 by Dr. Herong Yang. All rights reserved.
 */
import java.sql.*;
import javax.sql.*;
public class MySqlDataSource {
    public static void main(String [] args) {
        Connection con = null;
        try {

// Setting up the DataSource object
        com.mysql.jdbc.jdbc2.optional.MysqlDataSource ds
            = new com.mysql.jdbc.jdbc2.optional.MysqlDataSource();
        ds.setServerName("localhost");
        ds.setPortNumber(3306);
        ds.setDatabaseName("Base");
        ds.setUser("Herong");
        ds.setPassword("TopSecret");

// Getting a connection object
        con = ds.getConnection();

```

```
// Getting database info
DatabaseMetaData meta = con.getMetaData();
System.out.println("Server name: "
    + meta.getDatabaseProductName());
System.out.println("Server version: "
    + meta.getDatabaseProductVersion());

// Closing the connection
con.close();
} catch (Exception e) {
    System.err.println("Exception: "+e.getMessage());
}
}
```

```
C:\>javac -cp .;\local\lib\mysql-connector-java-5.0.7-bin.jar
    MySqlDataSource.java
```

```
C:\>java -cp .;\local\lib\mysql-connector-java-5.0.7-bin.jar
    MySqlDataSource
```

```
Server name: MySQL
Server version: 5.0.45-community-nt
```

4. Compléments d'informations

La déclaration du DataSource peut aussi se faire par l'intermédiaire de la console d'administration de Tomcat.

The screenshot shows the Tomcat Web Server Administration Tool interface. On the left is a tree view with the following structure:

- Tomcat Server
 - Service (Catalina)
 - Resources
 - Data Sources** (selected)
 - Mail Sessions
 - Environment Entries
 - User Databases
 - User Definition
 - Users
 - Groups
 - Roles

The main panel is titled "Data Sources" and contains a table with the following properties and values:

Property	Value
JNDI Name:	<input type="text" value="jdbc/TutoPool"/>
Data Source URL:	<input type="text" value="jdbc:mysql://localhost:3306/base"/>
JDBC Driver Class:	<input type="text" value="org.gjt.mm.mysql.Driver"/>
User Name:	<input type="text" value="user"/>
Password:	<input type="password" value="xxxxxxx"/>
Max. Active Connections:	<input type="text" value="8"/>
Max. Idle Connections:	<input type="text" value="8"/>
Max. Wait for Connection:	<input type="text" value="10000"/>
Validation Query:	<input type="text" value="SELECT 1"/>

At the top right of the main panel is a "Commit Changes" button. At the bottom right is a "Save" button.

Comme c'est une GlobalNamingResource qui est ajoutée au server.xml, elle n'est pas limitée à un seul Context et il faut déclarer un lien pour chaque Context qui peut y accéder. Cette solution permet d'éviter la section <resource-ref> du fichier web.xml et simplifie l'écriture du server.xml à :

```
<Context path="/TutoPool"
    reloadable="true"
    docBase="\TutoPool" >
    <ResourceLink
        name="jdbc/TutoPool"
        global="jdbc/TutoPool"
        type="javax.sql.DataSource"/>
</Context>
```

5. Accès à une base de données PostgreSql via une Datasource et JNDI

5.1. Aperçu

L'API JDBC fournit une interface client et serveur pour la gestion de pools de connexions. L'interface client est `javax.sql.DataSource`, qui est ce que le code de l'application utilisera typiquement pour récupérer une connexion à la base de données depuis ce pool. L'interface du serveur est `javax.sql.ConnectionPoolDataSource`. C'est la méthode utilisée par la plupart des serveurs d'application pour se créer une interface avec le pilote JDBC de PostgreSQL.

5.2. Serveurs d'application : `ConnectionPoolDataSource`

PostgreSQL inclut une implémentation de `ConnectionPoolDataSource` pour JDBC 2 et une pour JDBC 3, comme montré dans [Tableau 1](#).

Tableau 1. Implémentations de `ConnectionPoolDataSource`

JDBC	Classe d'implémentation
2	<code>org.postgresql.jdbc2.optional.ConnectionPool</code>
3	<code>org.postgresql.jdbc3.Jdbc3ConnectionPool</code>

Les deux implémentations utilisent le même schéma de configuration. JDBC requiert qu'un `ConnectionPoolDataSource` soit configuré via des propriétés JavaBean, décrites dans [Tableau 2](#), si bien qu'il y a des méthodes get et set pour chacune des propriétés.

Tableau 2. Propriétés de configuration de `ConnectionPoolDataSource`

Propriété	Type	Description
<code>serverName</code>	String	Nom d'hôte du serveur de bases de données PostgreSQL
<code>databaseName</code>	String	Nom de la base de données PostgreSQL
<code>portNumber</code>	int	Port TCP sur lequel le serveur de bases de données PostgreSQL écoute (ou 0 pour utiliser le port par défaut)
<code>User</code>	String	Utilisateur utilisé pour réaliser les connexions à la base de données
<code>Password</code>	String	Mot de passe utilisé pour réaliser les connexions à la base de données
<code>defaultAutoCommit</code>	boolean	Les connexions doivent-elles activer la validation automatique (autocommit) lorsqu'elles sont fournies au demandeur. Par défaut à false pour désactiver la validation automatique.

5.3. Applications : `DataSource`

PostgreSQL inclut deux versions de `DataSource` pour JDBC 2 et deux pour JDBC 3, comme le montre [Tableau 3](#). Les implémentations ne ferment pas les connexions lorsque le client appelle la méthode `close` mais renvoie à la place les connexions dans un ensemble de connexions disponibles pour les autres clients. Ceci évite une surcharge pour l'ouverture et la fermeture des connexions et permet à un grand nombre de clients de partager un petit nombre de connexions à la base de données.

L'implémentation des sources de données en pool fournie ici ne dispose que de fonctionnalités limitées. Entre autres choses, les connexions ne sont jamais fermées jusqu'à ce que le pool soit lui-même fermé ; il n'existe pas de moyens de diminuer le pool. De même, les connexions demandées par les utilisateurs autres que celui de l'utilisateur configuré par défaut ne sont pas gérées dans ce pool. Beaucoup de serveurs d'application fournissent des fonctionnalités plus avancées et utilisent à la place l'implémentation `ConnectionPoolDataSource`.

Tableau 3. Implémentations de `DataSource`

JDBC	Gestion en ensemble	Classe d'implémentation
2	Non	<code>org.postgresql.jdbc2.optional.SimpleDataSource</code>
2	Oui	<code>org.postgresql.jdbc2.optional.PoolingDataSource</code>
3	Non	<code>org.postgresql.jdbc3.Jdbc3SimpleDataSource</code>
3	Oui	<code>org.postgresql.jdbc3.Jdbc3PoolingDataSource</code>

Toutes les implémentations utilisent le même schéma de configuration. JDBC impose qu'une `DataSource` soit configurée via des propriétés JavaBean, affichées dans [Tableau 4](#), si bien qu'il existe des méthodes get et set pour chacune de ces propriétés.

Tableau 4. Propriétés de configuration de DataSource

Propriété	Type	Description
serverName	String	Nom d'hôte du serveur de bases de données PostgreSQL
databaseName	String	Nom de la base de données PostgreSQL
portNumber	int	Port TCP sur lequel le serveur de bases de données PostgreSQL est en écoute (ou 0 pour utiliser le port par défaut)
User	String	Utilisateur utilisé pour réaliser les connexions à la base de données
Password	String	Mot de passe utilisé pour réaliser les connexions à la base de données

Les implémentations de gestion de pools requièrent quelques propriétés supplémentaires de configuration, qui sont affichées dans [Tableau 5](#).

Tableau 5. Propriétés supplémentaires de configuration des ensembles de DataSource

Propriété	Type	Description
dataSourceName	String	Chaque DataSource de l'ensemble doit avoir un nom unique.
initialConnections	int	Le nombre de connexions à créer sur la base de données à l'initialisation de l'ensemble.
maxConnections	int	Le nombre maximum de connexions ouvertes sur la base de données. Quand plus de connexions sont demandées, l'appelant est bloqué jusqu'à ce qu'une connexion soit disponible dans le pool.

[Exemple 1](#) montre un exemple de code d'application typique utilisant un DataSource de l'ensemble.

Exemple 1. Exemple de code pour DataSource

Le code pour initialiser un DataSource de l'ensemble pourrait ressembler à ceci :

```
Jdbc3PoolingDataSource source = new Jdbc3PoolingDataSource();
source.setDataSourceName("Une source de données");
source.setServerName("localhost");
source.setDatabaseName("test");
source.setUser("utilisateur_test");
source.setPassword("motdepasstest");
source.setMaxConnections(10);
```

Puis le code utilisant une connexion du pool pourrait ressembler à ceci. Notez qu'il est indispensable que les connexions soient fermées. Sinon, l'ensemble << manquera >> de connexions et finira par bloquer tous les clients.

```
Connection con = NULL;
try {
    con = source.getConnection(); // utilisez la connexion
} catch (SQLException e) {
    // trace de l'erreur
} finally {
    if (con != NULL) {
        try { con.close(); } catch (SQLException e) {}
    }
}
```

5.4. Sources de données et JNDI

Toutes les implémentations de `ConnectionPoolDataSource` et de `DataSource` peuvent être enregistrées dans JNDI. Dans le cas d'une implémentation sans pool de connexions, une nouvelle instance est créée à chaque fois que l'objet est récupéré à partir de JNDI avec les mêmes paramètres que ceux de l'instance qui avaient été stockés. Pour les

implémentations de pool, la même instance est récupérée aussi longtemps qu'elle sera disponible (c'est-à-dire pas une JVM différent récupérant l'ensemble à partir de JNDI), sinon une nouvelle instance est créée avec les mêmes paramètres.

Dans l'environnement du serveur d'application, habituellement, l'instance de DataSource du serveur d'applications est stockée dans JNDI au lieu de l'implémentation ConnectionPoolDataSource de PostgreSQL.

Dans un environnement applicatif, l'application pourrait stocker le DataSource dans JNDI pour qu'elle n'ait pas besoin de faire une référence au DataSource disponible pour tous les composants de l'application qui pourraient en avoir besoin. Un exemple de ceci est montré dans [Exemple 2](#).

Exemple 2. Exemple de code pour un DataSource JNDI

Le code de l'application pour initialiser un DataSource de l'ensemble et pour l'ajouter à JNDI pourrait ressembler à ceci :

```
Jdbc3PoolingDataSource source = new Jdbc3PoolingDataSource();
source.setDataSourceName("A Data Source");
source.setServerName("localhost");
source.setDatabaseName("test");
source.setUser("testuser");
source.setPassword("testpassword");
source.setMaxConnections(10);
new InitialContext().rebind("DataSource", source);

Puis, le code pour utiliser la connexion de l'ensemble pourrait ressembler à ceci :
Connection con = NULL;
try {
    DataSource source = (DataSource)new InitialContext().lookup("DataSource");
    con = source.getConnection();
    // utilisez la connexion
} catch (SQLException e) {
    // tracez l'erreur
} catch (NamingException e) {
    // La source de données n'a pas été trouvée dans JNDI
} finally {
    if (con != NULL) {
        try { con.close(); } catch (SQLException e) {}
    }
}
```

Nota : Mettre dans le classpath le pilote jar de PostgreSql

6. Mise en place d'une authentification via un formulaire sous Tomcat

Cet article explique comment mettre en place une méthode d'authentification ou un *Realm* (entité vérifiant les droits des utilisateurs) s'appuyant basée sur une *DataSource* s'effectuant via un formulaire.

I. Création de la base de données

Dans la mesure où notre *Realm* s'appuie sur une *DataSource*, les utilisateurs et leurs mots de passe seront stockés dans une base de données.:

```
CREATE TABLE users (
    name VARCHAR(32),
    password VARCHAR(32)
);
CREATE TABLE roles (
    name VARCHAR(32),
    role VARCHAR(32)
);
```

Il est conseillé de mettre le nom d'utilisateur en clé primaire ou d'avoir un index unique dessus.

Une fois la base en place, il est nécessaire de créer l'entrée JNDI qui permettra d'y accéder.

II. Création de la DataSource

II.A. Récupération des drivers JDBC

Il s'agit de classes Java, la plupart du temps compactées au format JAR. Pour que Tomcat les prenne en compte, il suffit de les placer dans le répertoire **<CATALINA_HOME>/common/lib**.

II.B. Déclaration de la DataSource


La création de la source de données se fait au choix via l'application d'administration de Tomcat ou à la main, en éditant les fichiers XML appropriés. La configuration via l'application admin étant assez simple nous ne la décrivons pas, nous nous intéresserons à la description de la définition manuelle.


Dans le cadre de notre exemple, les balises indiquées ci-dessous sont placées au sein des balises **Context** de l'application qui utilisera l'authentification. Cette balise peut se trouver au sein d'un fichier de configuration XML indépendant (habituellement situé dans le répertoire **<CATALINA_HOME>/conf/Catalina/localhost/nomAppli.xml**). Si vous utilisez l'environnement de développement *Web Tools Platform* d'Eclipse, la définition du contexte se trouve au sein du fichier **server.xml** du serveur utilisé (si vous venez de déployer l'application sur le serveur, la balise **Context** est vide, n'oubliez pas de supprimer le

La définition de la *DataSource* se fait en deux fois dans la balise **<Context ... /> </Context>** du **server.xml** .. Tout d'abord, il faut indiquer son type et l'alias JNDI utilisé pour y faire référence ; ensuite, via une seconde balise, on indique les paramètres de la *DataSource*.

Commençons donc par définir notre source de données :

```
<Resource name="jdbc/authen" type="javax.sql.DataSource" />
```

 Si vous désirez accéder à la *DataSource* manuellement (typiquement si la même base de données est utilisée pour l'authentification et pour le fonctionnement de l'application proprement dite), vous utiliserez un code de la forme **DataSource ds = (DataSource) new InitialContext().lookup("java:comp/env/jdbc/authen");**.

 Si vous désirez partager la définition de la *DataSource* entre plusieurs applications, vous pouvez la déclarer au sein de la balise **GlobalNamingResources** qui est elle-même située au sein de la balise **Server** du fichier **server.xml**.

Une fois la *DataSource* déclarée, il faut lui passer les paramètres nécessaires pour établir la connexion à la base de données. Ceci se fait au moyen de la balise **ResourceParams** qui se trouve au même niveau que la balise **Ressource** qu'elle configure. Voici un exemple de configuration de la *DataSource* pour notre exemple, à vous de l'adapter à votre base de données.

```
<ResourceParams name="jdbc/authen">
  <parameter>
    <name>username</name>
    <value>authen</value>
  </parameter>
  <parameter>
    <name>password</name>
    <value>authen</value>
  </parameter>
  <parameter>
    <name>url</name>
    <value>jdbc:postgresql:authen</value>
  </parameter>
  <parameter>
    <name>driverClassName</name>
    <value>org.postgresql.Driver</value>
  </parameter>
</parameter>
```

```
<name>maxIdle</name>
<value>2</value>
</parameter>
<parameter>
<name>maxWait</name>
<value>5000</value>
</parameter>
<parameter>
<name>maxActive</name>
<value>4</value>
</parameter>
</ResourceParams>
```

La *DataSource* est maintenant prête à être utilisée, vous pouvez la tester en essayant d'y accéder par le biais du code suivant :

```
DataSource ds= (DataSource) new InitialContext().lookup("java:comp/env/jdbc/authen");
```

Placez ces instructions dans un fichier JSP, lancez Tomcat et essayez d'y accéder, si une erreur survient ("Le nom xxx n'est pas lié à ce contexte") c'est que vous avez mal paramétré la *DataSource* (vérifiez qu'elle est bien placée au bon niveau dans le fichier XML).



Sous Tomcat 5.5, la déclaration de la *DataSource* est plus simple car elle s'effectue en une seule balise :

```
<Resource name="jdbc/authen" auth="Container" type="javax.sql.DataSource" username="authen"
password="authen" driverClassName="org.postgresql.Driver" url="jdbc:postgresql:authen"
maxActive="8" maxIdle="4"/>
```

III. Création du Realm

Un *Realm* est un dispositif servant à identifier les utilisateurs. Il permet de faire l'association login/mot de passe afin de déterminer si l'utilisateur est correctement authentifié ou non. Pour chaque utilisateur, le *Realm* connaît la liste des rôles associés. Les rôles sont les responsabilités attribuées à un utilisateur donné. La protection des ressources se fait par rôle, c'est-à-dire que l'on indique le rôle dont doit disposer un utilisateur pour accéder à la ressource. L'alimentation du *Realm* (ajout d'utilisateurs et des rôles correspondants) est à la charge du développeur.

Un *Realm* peut exploiter des données stockées sous différentes formes : annuaire LDAP accessible via JNDI, fichier XML (par exemple le fichier **tomcat-users.xml** qui sert à configurer l'accès aux applications admin et manager de Tomcat) ou encore, comme ici, une *DataSource* JNDI. Cette liste n'est pas exhaustive, par ailleurs, vous pouvez créer vos propres *Realms* en implémentant l'interface **org.apache.catalina.Realm**.

Un *Realm* peut se déclarer au niveau *Engine* (partagé par toutes les applications de tous les hôtes virtuels), au niveau *Host* (partagé par toutes les applications de l'hôte virtuel) ou au niveau *Context* (valable pour l'application dans lequel il est défini). Un *Realm* défini à un niveau donné masque ceux de niveau supérieur. Dans le cas où il utilise des ressources (comme pour notre *Realm*), il faut que la ressource soit visible au niveau de la déclaration du *Realm*.

En ce qui concerne les *DataSources* JNDI, il existe déjà un *Realm*, **org.apache.catalina.realm.DataSourceRealm** qui fait tout ce dont nous avons besoin. Il suffit donc de le configurer.

La configuration du *Realm* est relativement simple, il suffit de lui indiquer la *DataSource* à utiliser ainsi que le schéma de la base. Vous pouvez choisir de stocker les mots de passe sous une forme cryptée, dans ce cas il faut indiquer l'algorithme à utiliser (n'oubliez pas dans ce cas de crypter le mot de passe avec le même algorithme avant de l'envoyer en base).

La configuration minimale d'un *Realm* s'appuyant sur une *DataSource* est la suivante :

```
<Realm className="org.apache.catalina.realm.DataSourceRealm"
```

```

dataSourceName="jdbc/authen"
userTable="users"
userRoleTable="roles"
userNameCol="name"
userCredCol="password"
roleNameCol="role"
/>

```

Cette configuration fonctionne dans le cas où la *DataSource* est déclarée dans la zone **GlobalNamingResources**. Dans le cas où la *DataSource* est déclarée au niveau du *Context*, il faut ajouter l'attribut **localDataSource="true"** au *Realm*. Si vos mots de passe sont stockés sous forme cryptée, indiquez l'algorithme utilisé par le biais de l'attribut **digest**.

Votre *Realm* est maintenant prêt, il ne reste plus qu'à mettre en place la protection des pages.

IV. Mise en place de la protection

IV.A. Définition des pages à protéger

Dans un premier temps, nous utiliserons une authentification classique (*via* la boîte de dialogue du navigateur) pour vérifier que le *Realm* est correctement configuré.

A partir de maintenant, sauf indication contraire, les balises se placent dans le fichier **web.xml** de l'application à protéger.

Les pages à protéger et les rôles nécessaires pour y accéder sont déclarés dans des balises **security-constraint**. Une balise **display-name** permet de définir le nom de la contrainte pour sa gestion par des outils tiers. Les pages se déclarent au moyen de la balise **web-resource-collection** qui contient une balise **web-resource-name** et une ou plusieurs balises **url-pattern** qui indiquent le motif des URL protégées (avec ou sans utilisation de jokers). La *web-resource-collection* est suivie par une balise **auth-constraint** qui donne les rôles à avoir et, éventuellement, les méthodes HTTP nécessitant une authentification (si aucune n'est précisée, elles en nécessitent toutes une). Notez que l'utilisateur doit disposer d'un des rôles spécifiés pour pouvoir accéder à la ressource. À la suite de la balise **security-constraint**, une balise **login-config** définit la façon dont s'effectue la connexion. Une balise **realm-name** indique le nom de la zone protégée (ce que vous voulez) et une balise **auth-method** définit le type de connexion :

- **BASIC** boîte de dialogue "classique" ;
- **DIGEST** le mot de passe est crypté avant l'envoi par le navigateur, cette méthode n'étant pas supportée par tous les navigateurs, elle n'est pas très utilisée en pratique. De plus, elle n'est pas supportée par tous les *Realms* ;
- **FORM** l'authentification est effectuée via un formulaire créé par le développeur, nous entrerons dans les détails plus loin dans cet article ;
- **CLIENT-CERT** authentification SSL basée sur un certificat client.

Les rôles utilisés au sein de la balise **auth-constraint** doivent être déclarés au préalable par le biais de balises **security-role** (une par rôle).

Dans notre cas, nous utiliserons tout d'abord une authentification **BASIC** afin de pouvoir tester au plus tôt la configuration du *Realm* mise en place sans créer de formulaire.

La configuration a donc l'aspect suivant :

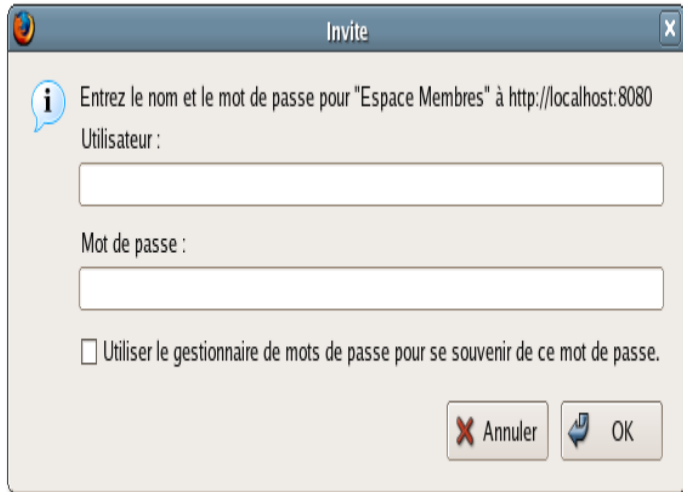
```

<security-constraint>
  <display-name>Test d'authentification tomcat</display-name>
  <!-- Liste des pages protégées -->
  <web-resource-collection>
    <web-resource-name>Page sécurisée</web-resource-name>
    <url-pattern>/admin/*</url-pattern>
  </web-resource-collection>
  <!-- Rôles des utilisateurs ayant le droit d'y accéder -->
  <auth-constraint>
    <role-name>admin</role-name>
  </auth-constraint>
</security-constraint>
<login-config>
  <!-- Type d'authentification -->
  <auth-method>BASIC</auth-method>
  <realm-name>Espace Membres</realm-name>
</login-config>

```

```
<!-- Rôles utilisés dans l'application -->
<security-role>
  <description>Administrateur</description>
  <role-name>admin</role-name>
</security-role>
```

Placez des pages dans le répertoire protégé et tentez d'y accéder, vous devriez avoir la boîte de dialogue classique :



Invite d'identification

Saisissez un nom et un mot de passe présent dans la base, normalement vous devriez accéder à la ressource protégée.

V. Mise en place du formulaire

La mise en place du formulaire est relativement simple. Tout d'abord, bien sûr, il faut le créer. Il doit répondre à des critères au niveau des noms de champs et de l'action :

- Le champ correspondant au login doit s'appeler **j_username** ;
- Le champ du mot de passe doit s'appeler **j_password** ;
- L'action du formulaire doit être **j_security_check**.

Le formulaire résultant a donc l'aspect suivant :

```
<form action="j_security_check" method="post">
<input type="text" name="j_username"/>
<input type="password" name="j_password"/>
<input type="submit" value="Connexion"/>
</form>
```

Une fois le formulaire prêt, vous devez aussi créer une page d'erreur qui sera référencée dans le fichier de configuration.

Notez que la page de login et la page d'erreur peuvent toutes deux se trouver dans le répertoire protégé, cela ne pose pas de problème.

La balise **login-config** doit maintenant être modifiée pour avoir la structure suivante :

```
<login-config>
<auth-method>FORM</auth-method>
<realm-name>Espace membres</realm-name>
<form-login-config>
  <form-login-page>/admin/login.jsp</form-login-page>
  <form-error-page>/admin/error_login.jsp</form-error-page>
</form-login-config>
</login-config>
```

Si vous tentez désormais (après avoir relancé Tomcat) d'accéder à une des ressources protégées sans être identifié, vous devez passer par la page de login avant d'atteindre réellement la ressource.

