JDBC dans une application Web

JDBC (Java DataBase Connectivity) est l'API standard pour interagir avec les bases données relationnelles en Java. Cette API peut être utilisée dans une application Web.

Accès à une DataSource

Dans un serveur d'application, l'utilisation du DriverManager JDBC est remplacée par celle de la DataSource. L'interface DataSource n'offre que deux méthodes :

```
// Attempts to establish a connection with the data source
Connection getConnection()

// Attempts to establish a connection with the data source
Connection getConnection(String username, String password)
```

Il n'est pas possible de spécifier l'URL de connexion à la base de données avec une DataSource. En revanche, il est possible de configurer dans le serveur une DataSource en lui associant des paramètres de configuration tels que l'URL et les identifiants de connexion. Cette DataSource est gérée comme une ressource par le serveur. Un serveur Java EE maintient l'ensemble de ses ressources sous la forme d'une arborescence. Comme pour une arborescence de fichiers sur un disque ou sur un réseau, il est possible d'accéder aux ressources du serveur à partir de leur nom. La différence avec les gestion de fichiers sur un disque ou sur un réseau est que les ressources d'un serveur sont des interfaces Java.

L'accès aux ressources du serveur se fait par programmation grâce à l'API **JNDI** (*Java Naming and Directory Interface*). JNDI est une API standard de Java permettant de se connecter à des annuaires (notamment les annuaires LDAP). Un serveur Java EE dispose de sa propre implémentation interne d'annuaire pour la gestion de ses ressources.

Les ressources telles que les *DataSources* sont donc stockées dans un annuaire interne et il est possible d'y accéder avec l'API JNDI. Les ressources sont rangées dans une arborescence (comme le sont les fichiers dans des répertoires). Une ressource est stockée dans l'arborescence dont le chemin commence par **java:/comp/env**.

Exemple de récupération d'une DataSource en utilisant l'API JNDI

```
// javax.naming.InitialContext désigne le contexte racine de l'annuaire.
// Un annuaire JDNI est constitué d'instances de javax.naming.Context
// (qui sont l'équivalent des répertoires dans un système de fichiers).
Context envContext = InitialContext.doLookup("java:/comp/env");

// On récupère la source de données dans le contexte java:/comp/env
DataSource dataSource = (DataSource) envContext.lookup("nomDeLaDataSource");
```

La classe InitialContext agit comme une classe *factory*. Elle permet de fabriquer un contexte qui représente une position dans l'arborescence des ressources JNDI. Le contexte JNDI **java:/comp/env** est un contexte particulier. Il désigne l'ensemble des composants Java EE disponibles dans l'environnement (env) du composant Java EE (comp) courant (dans notre cas l'application Web). la méthode lookup permet ensuite de récupérer une ressource à partir de son nom. Il faut effectuer un trans-typage (*cast*) vers le type réel de la ressource, pour le cas qui nous intéresse : une DataSource.

Injection d'une DataSource

Afin de simplifier l'accès à une DataSource, il est possible de l'injecter directement comme attribut d'un composant Java EE. Comme les Servlets sont des composants Java EE, il est possible d'ajouter l'annotation Resource sur un attribut de type DataSource :

Injection d'une DataSource dans une Servlet

```
import java.io.IOException;
import java.sql.Connection;

import javax.annotation.Resource;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

@WebServlet("/MyServlet")
public class MyServlet extends HttpServlet {

  @Resource(name = "nomDeLaDataSource")
  private DataSource dataSource;

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                        throws ServletException, IOException {

    try (Connection connection = dataSource.getConnection()) {
      // ...
    }

  }

}
```

L'annotation @Resource permet de spécifier le nom de la DataSource grâce à l'attribut *name*.

Avertissement

Pour avoir accès à l'annotation Resource, vous devez ajouter une dépendance dans votre projet. Pour un projet Maven, il suffit d'ajouter dans le fichier `pom.xml` la dépendance suivante :

```
<dependency>
  <groupId>javax.annotation</groupId>
  <artifactId>javax.annotation-api</artifactId>
  <version>1.3.2</version>
  <scope>provided</scope>
</dependency>
```

Déclaration de la DataSource dans le fichier web.xml

Le fichier de déploiement `web.xml` doit déclarer la DataSource comme une ressource de l'application. Cela va permettre au serveur d'application de permettre à l'application de se connecter à la base de données associée. Pour cela, on utilise l'élément `<resource-ref>` dans le fichier `web.xml` :

Déclaration de la DataSource dans le fichier web.xml

```
<resource-ref>
  <res-ref-name>nomDeLaDataSource</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
</resource-ref>
```

Mais comment le serveur d'application fait-il pour lier une DataSource avec une connexion vers une base de données ? Malheureusement, il n'existe pas de standard et chaque serveur d'application dispose de sa procédure. Nous allons voir dans la section suivante comment créer une DataSource spécifiquement pour Tomcat.

Déclaration d'une DataSource dans Tomcat

Tomcat n'est pas à proprement parler un serveur d'application, il s'agit juste d'un conteneur de Servlet. Néanmoins, il peut déployer des applications Web Java et il supporte l'annotation @Resource ainsi que la configuration d'une DataSource dans le serveur ou dans l'application.

Une connexion JDBC est réalisée à travers un pilote. Pour déclarer une DataSource vers une base de données MySQL, par exemple, nous devons installer le pilote MySQL dans le serveur. Pour cela, il vous faut télécharger le pilote et le placer dans le répertoire `lib` situé dans le répertoire d'installation du serveur.

Une fois, le pilote ajouté, il est possible de déclarer la DataSource dans le fichier `conf/server.xml`. Si vous utilisez Tomcat dans Eclipse, alors vous devez disposer d'un projet `Servers` qui a été créé automatiquement dans votre espace de travail. Dedans, se trouvent

toutes les configurations des serveurs que vous avez créés. Pour Tomcat 9, le nom par défaut est `Tomcat v9.0 Server at localhost-config`. Dans ce répertoire, se trouve le fichier `server.xml` qu'il va falloir modifier.

> **Note**
>
> L'intégration de Tomcat dans Eclipse crée automatiquement une copie de la configuration originale du serveur pour chacune des instances de serveur créées dans Eclipse. Ainsi, il est possible de déclarer plusieurs serveurs Tomcat dans Eclipse qui possèdent chacun leur configuration spécifique.

Dans le fichier `server.xml`, vous devez ajouter avant la fin de l'élément `<Host />`, les informations de déploiement de votre application.

> **Avertissement**
>
> Si vous utilisez le serveur Tomcat dans Eclipse, la balise `<Context/>` est automatiquement ajoutée par Eclipse lorsque vous ajoutez votre application dans le serveur. Attention, cette balise sera aussi supprimée si vous supprimez l'application du serveur.

Exemple de balise de contexte de déploiement dans le fichier server.xml

```
<Context docBase="nomAppli"
         path="/nomAppli"
         reloadable="true"
         source="org.eclipse.jst.jee.server:jdbc">
  <Resource name="[nomDataSource]"
            auth="Container"
            type="javax.sql.DataSource"
            maxTotal="100"
            maxIdle="30"
            maxWaitMillis="10000"
            username="[USERNAME]"
            password="[PASSWORD]"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://[HOST]:3306/[NOM BASE]" />
</Context>
```

Si vous ne voulez pas modifier la configuration du serveur, il est possible d'ajouter un fichier `src/main/webapp/META-INF/context.xml` dans votre projet Maven et de déclarer à l'intérieur l'élément `<Resource />` :

Exemple de fichier context.xml dans l'application

```
<Context>

  <Resource name="[nomDataSource]"
            auth="Container"
            type="javax.sql.DataSource"
            maxTotal="100"
            maxIdle="30"
            maxWaitMillis="10000"
            username="[USERNAME]"
            password="[PASSWORD]"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://[HOST]:3306/[NOM BASE]" />

</Context>
```

Note

Pour une présentation de la déclaration des *data sources* pour différents SGBDR, reportez-vous à la documentation de Tomcat.

Dans sa gestion des *data sources*, Tomcat inclut la gestion d'un *pool* de connexions en s'appuyant sur la bibliothèque Apache DBCP. L'ensemble des paramètres de configuration du *pool* de connexions est disponibles dans la documentation de DBCP. Ces paramètres sont utilisables comme attributs de l'élément `<Resource />`.