Cours n°9

Accès distant aux bases de données

Sommaire

INTRODUCTION

Bibliographie

George Reese, JDBC et Java Guide du programmeur, O'Reilly, 2001

Donald Balès, JDBC Pocket Reference , O'Reilly, 2003

Daniel Serain, Le middleware: Concepts et technologies , Masson, 1997

1.1 JAVA DATABASE CONNECTIVITY (JDBC)

Définition

API Java permettant la connexion avec les bases de données relationnelles (SGBDR)

Ensemble de classes et d'interfaces permettant l'utilisation sur le réseau d'un ou plusieurs SGBDR à partir d'un programme Java

Interface distante uniforme permettant un accès homogène aux SGBD

- simple à mettre en œuvre
- indépendant de la SGBD cible
- supportant les fonctionnalités de base du langage SQL-2
- possibilité de plusieurs interfaces simultanément

Portabilité sur de nombreux O.S. et sur de nombreuses SGBDR (Oracle, Informix, Sybase, Mysql, Postgres, ..)

Liberté totale vis a vis des constructeurs

API JDBC

Paquetage « java.sql »

10 interfaces définissant les objets nécessaires à la connexion à une base éloigné et à la création et exécution de requêtes SQL

Gestion des pilotes
Driver

Gestion de la connexion
Connection

Organisation des tables de données
DatabaseMetaData,ParameterMetaData, ResultSetMetaData

Gestion des requêtes
Statement, PreparedStatement, StatementCallableStatement

Résultats des requêtes
ResultSet

Mises à jour des tables de données
Savepoint

Pilote JDBC

Utilisation d'un composant logiciel spécifique à chaque base de données

Pilote (*driver*)

Conversion des requêtes JDBC dans le langage natif du SGBDR

Pilotes JDBC pour tous les principaux constructeurs

Oracle, Sybase, Informix, DB2, ...

4 Types de de pilote

Type I : Pont JDBC-ODBC
traduction en requête ODBC (windows), librairie en C (pas d'applet)

Type II : API natives du SGBD (non java)
traduction en requêtes spécifiques, librairie en C/C++ (pas d'applet)

Type III : API réseau générique en java (applet)

Type IV : API réseau du SGBD en java (applet)

Référence de tous les pilotes
www.javasoft.com/products/jdbc/drivers.html

Protocoles

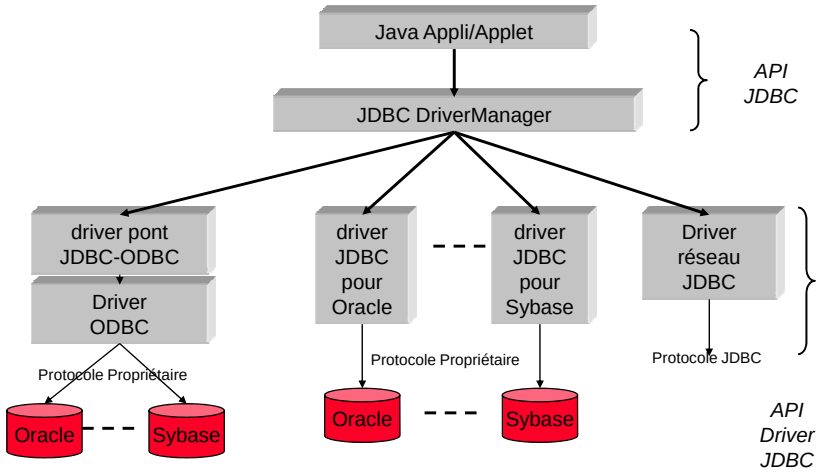


Schéma de mise en œuvre d'une requête

Chargement du(des) pilote JDBC pour le SGBDR

Class.forName(Nom de la classe contenant le pilote).newInstance();

Pilote de type IV pour la base de données Mysql "com.mysql.jdbc.Driver"

Etablissement de la connexion à la base de données

DriverManager.getConnection(..)

Création d'un descripteur de requêtes

Connexion.createStatement()

Exécution de la requête et traitement des données retournées

ResultSet Statement.executeQuery()

Fermeture des différents espaces

Statement.close() Connexion.close()

Classe DriverManager

Connexion à une base de données

static Connection **getConnection**(String url)

static Connection **getConnection**(String url, String user, String password)

static Connection **getConnection**(String url, Properties info)

Lancement d'une SQLException en cas d'erreur

URL de connexion

Accès à la base via un URL de la forme :

jdbc:<sous-protocole>:<nom-BD>;param=valeur, ... spécifiant

- l'utilisation de JDBC
- le driver ou le type de SGBDR
- l'identification de la base locale ou distante

avec des paramètres de configuration éventuels (nom utilisateur, mot de passe, …)

Exemples :

String url = **"jdbc:odbc:maBase"** ;

String url = **"jdbc:msql://leo.paris4.sorbonne.fr:3306:maBase"**;

testConnexion.java

Connexion à une base de données Mysql

```
try { Class.forName("com.mysql.jdbc.Driver").newInstance(); }
catch (Exception ex) {
      System.out.println("Erreur de chargement du pilote");
      System.exit(0);
      }

try {
      String url = "jdbc:mysql://172.20.76.1:3306/Ilgii1";
      Properties props = new Properties();
      props.setProperty("user","ilgii1");
      props.setProperty("password","milgii1");
      Connection connection=DriverManager.getConnection(url, props);
      System.out.println ("Connexion à la base de donnees");
}
catch (SQLException ex) {
      System.out.println("Erreur de connexion"+ex);
      System.exit(0);
}
```

```
Connexion à la base de donnees
```


Classe Connexion

Accès aux descripteurs de la base de données

DatabaseMetaData **getMetaData**()

Création d'un descripteur de requêtes

Statement **createStatement**()

descripteur de requête simple

PreparedStatement **prepareStatement**(String sql)

descripteur de requête précompilée

CallableStatement **prepareCall**(String sql)

descripteur de requête stockée dans le SBDBR

Gestion des mises à jour

void **commit**()

Validation d'un groupe de transaction, commit par défaut après chaque ordre SQL (passage en mode manuel avec la méthode setAutoCommit)

void **rollback**() void **rollback**(Savepoint savepoint)

Restauration de l'état de la base à un état précédent

testMetaData.java

Lecture des métadonnées d'une base de données

```
DatabaseMetaData meta=con.getMetaData();
System.out.println ("Lecture et Affichage des metadata ");
System.out.print(" Database: "+meta.getDatabaseProductName());
System.out.println(".Version "+meta.getDatabaseProductVersion());
System.out.println(" User Name: "+meta.getUserName());
ResultSet tables = meta.getTables(con.getCatalog(),null,"%",null);
// affichage des informations
while(tables.next()){
for(int i=0; i<tables.getMetaData().getColumnCount();i++){
      String nomColonne = tables.getMetaData().getColumnName(i+1);
      Object valeurColonne = tables.getObject(i+1);
      System.out.println(nomColonne+" = "+valeurColonne); } }
```

```
Database: MySQL.Version 5.0.21-community-nt
User Name: ilgii1@LILI
TABLE_CAT = Ilgii1
TABLE_NAME = personne
TABLE_TYPE = TABLE
```

Type de requêtes SQL

Classe Statement

Descripteur de requêtes interprétées à chaque exécution

Classe PreparedStatement

Précompilée et paramétrable

Préparée pour être exécutée plusieurs fois

Gérée par le programme Java

Classe CallableStatement

Procédure stockée dans le SGBDR et paramétrable

Ecriture dans le langage interne du SGBDR

SQL Server Transac-SQL

Gérée par le SGBDR

\+ Augmentation des performances

\- Langage propriétaire

Classe Statement (descripteur de requêtes SQL)

Exécution de requêtes SQL

boolean **execute**(String sql)

Envoi d'une requête SQL ne renvoyant pas de résultat

int **executeUpdate**(String sql)

Envoi d'une requête SQL de type INSERT, UPDATE, or DELETE

ResultSet **executeQuery**(String sql)

Envoi d'une requête SQL renvoyant un résultat (type SELECT, …).

int[] **executeBatch**()

Envoi d'une liste de Requêtes SQL ne renvoyant pas de résultat

void **addBatch**(String sql)	Ajout d'une requête SQL à la liste
ResultSet **getResultSet**()	Accès au résultat de la dernière requête
void **close**()	Fermeture du descripteur

Correspondance des types

Type JDBC	Type Java	Accesseurs
CHAR VARCHAR	String	getString()
FLOAT DOUBLE	double	getDouble()
REAL	float	getFloat()
BIT	boolean	getBoolean()
BIGINT	long	getLong()
SMALLINT	short	getShort()
TINYINT	byte	getByte()
INTEGER	int	getInt()
NUMERIC DECIMAL	java.math.BigDecimal	getBigDecimal()
DATE	java.sql.Date	getDate()
TIMESTAMP	java.sql.TimeStamp	getTimeStamp()
TIME	java.sql.Time	getTime()

Création et destruction d'une table de données

testCréationTable.java

```
// création du descripteur de requête
Statement req = con.createStatement();
// création de la table de données personne
String tb1 = "personne (Nom CHAR (20), Masculin BIT, Age TINYINT);";
req.execute("create table " + tb1);
// création de la table de données vente
String tb2 = "vente (Montant NUMERIC, date DATE);";
req.execute("create table " + tb2);
// destruction de la table de données vente
req.execute("drop table vente");
```

```
Chargement du pilote
Connexion à la base données
Création de la table personne (Nom CHAR (20), Masculin BIT, Age
TINYINT);
Création de la table vente (Montant NUMERIC, date DATE);
Destruction de la table vente (Montant NUMERIC, date DATE);
Fermeture de connexion à la base données
```

Classe PreparedStatement (requête SQL précompilée)

Envoi d'une requête sans paramètres à la base de données

Plus rapide qu'un descripteur de requête classique (Statement)

une seule analyse par le SGBDR

Arguments dynamiques spécifiés par un « ? »

Connection.**prepareStatement**(« insert ? ? »)

Spécification des paramètres

Positionnement des paramètres par les 26 méthodes **setXXX()**

Exemple : void **setByte**(int parameterIndex, byte x)

au moins 2 arguments : numéro relatif de l'argument dans la requête, valeur à positionner

Envoi de la requête complétée

Mêmes méthodes que pour la classe Statement (sans paramètres)

testInsert.java

Insertion d'un élément avec une requête précompilée

```
String[] nom = {"Chirac", "Villepin", "Sarkozy", "Alliot-Marie"};
boolean[] masculin = {true, true, true, false};
byte[] age = {70, 65, 55, 50};

// création de la requête précompilée
preq = con.prepareStatement("insert into personne values (?, ?, ?)");
for (int i = 0;i < nom.length;i++) {
      preq.setString(1, nom[i]);
      preq.setBoolean(2, masculin[i]);
      preq.setByte(3, age[i]);

      // envoi de la requête
      preq.executeUpdate();
}
preq.close();
```

```
Chargement du pilote
Connexion à la base données
```


Classe ResultSet

Méthodes d'accès à un tuple

boolean next() retourne false si dernier tuple lu, true sinon
avancement du curseur sur le tuple suivant

boolean previous() retourne false si premier tuple lu, true sinon
avancement du curseur sur le tuple précédent

boolean absolute(int row), **boolean first**(), **boolean last**(),
Accès direct à un tuple

boolean relative(int row)

Accès relatif à un tuple par rapport à la position courante

Accès aux champs

xxx **getXXX**(int) et xxx **getXXX**(String) pour les valeurs des champs

Exemple : byte **getByte**(int columnIndex), byte **getByte**(String columnName)

ResultSetMetaData getMetaData() pour les types de données

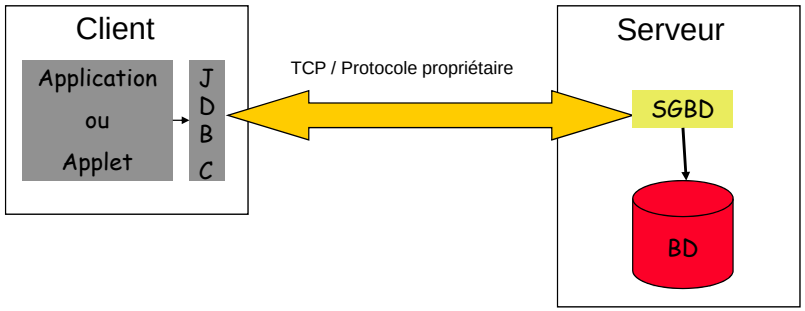
testConsultation.java

Consultation d'une base de données

```
Statement req = con.createStatement();
ResultSet res = req.executeQuery("SELECT Nom, Age FROM personne ;");
while (res.next()) {
      String nom = res.getString("Nom");
      byte age = res.getByte("Age");
      System.out.println("L'age de "+nom+" est : " + age);
}
req.close();
```

```
Chargement du pilote
Connexion à la base données
L'age de Chirac est : 70
L'age de Villepin est : 65
L'age de Sarkozy est : 55
L'age de Alliot-Marie est : 50
Fermeture de connexion à la base données
```

Architecture 2-tiers



Architecture 2-tiers

Principe

Page web contenant une applet (ou programme Jaca) envoyant des requêtes au SGBDR à l'aide de l'API JDBC

Avantages

Très simple à mettre en œuvre

Bon choix pour des applications clientes peu évoluées, à livrer rapidement et n'exigeant que peu de maintenance

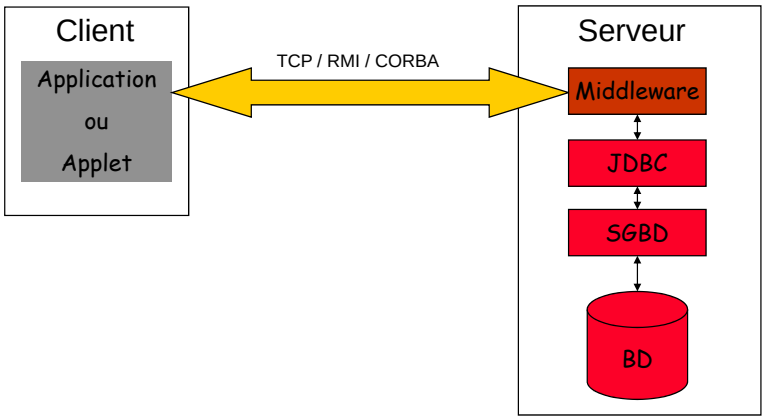
Inconvénients

Dépendance forte entre le programme client (applet ou programme java) et la structure du SGBDR

Modification de tous les programmes clients en cas de changement

Tout le traitement du côté client (pas de gestion de la concurrence)

Architecture 3-tiers



Architecture 3-tiers

Principes

Pas de liaison directe entre le programme client et le SGBDR
Echange par un programme tiers : Serveur *middleware*
réception des demandes du programme client
transmission sous forme de requêtes au SGBDR
Programme tiers souvent écrit en Java sous forme de servlet

Avantages

Ajout d'un niveau de sécurité et gestion de la concurrence
Possibilité de plusieurs méthodes de transmission entre le client et le middleware
sockets, RMI Java, CORBA, …
Plus de nécessité d'un serveur web et du SGBDR sur la même machine
duplication de bases de données

Inconvénients

Solution professionnelle plus difficile à mettre en oeuvre