JDBC

(Java Database Connectivity)

Préparé par

M.G. BELKASMI

JDBC

- JDBC est une API Java permettant d'accéder aux bases de données à l'aide du langage Java via des requêtes SQL.
- Cette API permet d'atteindre de manière quasi-transparente des bases Sybase, Oracle, Informix, ... avec le même programme Java JDBC.

JDBC

- En fait cette API est une spécification de ce que doit implémenter un constructeur de BD pour que celle ci soit interrogeable par JDBC.
- De ce fait dans la programmation JDBC on utilise essentiellement des références d'interface (Connection, Statement, ResultSet, ...).

JDBC

- Les constructeurs de BD se sont charger de fournir des classes qui implémentent les interfaces précitées qui permettent de soumettre des requêtes SQL et de récupérer le résultat.
- Par exemple Oracle fournit une classe qui, lorsqu'on écrit :

```
Statement stmt = conn.createStatement();
```

retourne un objet concret de la classe

```
OracleStatement (implements Statement )
```

qui est repéré par la référence stmt de l'interface Statement.

Pilotes (Drivers) JDBC

- L'ensemble des classes qui implémentent les interfaces spécifiées par JDBC pour un gestionnaire de bases de données particulier est appelé un **pilote JDBC.**
- Les protocoles d'accès aux BD étant propriétaires il y a donc plusieurs drivers pour atteindre diverses BD.

Pilotes (Drivers) JDBC

- l'interface **Driver**, décrit ce que doit faire tout objet d'une classe qui implémente l'essai de connexion à une base de données.
- Un tel objet doit obligatoirement s'enregistrer auprès du **DriverManager** et retourner en cas de succès un objet d'une classe qui implémente l'interface **Connection**.

Pilotes (Drivers) JDBC

On distingue deux familles de pilotes :

- Ceux dits **natifs** qui utilisent une partie écrite dans du code spécifique non Java (souvent en langage C) et appelé par ces implantations.
  - Ces pilotes sont rapides mais doivent être présent sur le poste client car ne peuvent pas être téléchargés par le ClassLoader de Java (ce ne sont pas des classes Java mais plutôt des librairies dynamiques).
  - Ils ne peuvent donc pas être utilisés par des applets dans des browsers classiques.
- Ceux dits **100% Java** qui interrogent le gestionnaire de base de données avec du code uniquement écrits en Java.
  - Ces pilotes peuvent alors être utilisés par des applets dans des browsers classiques.

Pilotes (Drivers) JDBC

Plus précisément il y a 4 types de pilotes:

- pilote de classe 1 : pilote jdbc:odbc
- pilote de classe 2 : pilote jdbc:protocole spécifique et utilisant des méthodes natives.
- pilote de classe 3 : pilote écrit en Java jdbc vers un middleware qui fait l'interface avec la BD
- pilote de classe 4 : pilote écrit en Java jdbc:protocole de la BD. Accède directement à l'interface réseau de la BD.

programme JDBC

Un code JDBC est de la forme :

- recherche et chargement du driver approprié à la BD.
- établissement de la connexion à la base de données.
- construction de la requête SQL
- envoi de cette requête et récupération des réponses
- parcours des réponses.

programme JDBC

Syntaxe

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection conX = DriverManager.getConnection(...);
Statement stmt = conX.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c ... FROM
    …  WHERE ...");
while (rs.next()) {
// traitement
}
```

- On utilise le package **java.sql.** La plupart des méthodes lèvent l'exception **java.sql.SQLException**

programme JDBC

Chargement du pilote

- On commence un programme JDBC en chargeant dans le programme, le pilote approprié pour la BD.
- Comme le programme peut interroger divers types de BD il peut avoir plusieurs pilotes.
- C'est au moment de la connexion que sera choisi le bon pilote par le **DriverManager**

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

programme JDBC

La connexion

- On ouvre une connexion avec une des méthodes **DriverManager.getConnection**(...) qui retourne un objet d'une classe qui implémente l'interface **Connection**.
- Ces méthodes contiennent comme premier argument une "URL JDBC".
- Elles recherchent le pilote adapté pour gérer la connexion avec la base repérée par cette URL.

programme JDBC

La connexion

- La syntaxe d'une URL JDBC est :

  jdbc:<*sous-protocole*>:<*baseID*>

- Le second argument est le protocole sous jacent dans lequel le pilote traduit. Le troisième argument est un identificateur de base de données *propre au sous-protocole.*
- *Exp:*
  - jdbc:odbc:@ orsay:1751:test1
  - jdbc:oracle:thin:@oraserv:1521:oradb

programme JDBC

La connexion

- Les arguments suivants de **getConnection**(...) sont des informations nécessaires à l'ouverture de la connexion : login/mot de passe.

Connection conX =
DriverManager.getConnection(URLjdbc,
"dbUser1", "pwuser1");

programme JDBC

La connexion

- La classe **DriverManager** est une classe qui ne contient que des méthodes statiques.
- Elle fournit des méthodes qui sont des utilitaires pour gérer l'accès aux bases de données par Java et les différents drivers JDBC à l'intérieur d'un programme Java.
- Finalement on ne créée ni ne récupère d'objet de cette classe.

programme JDBC

Les requêtes

- Syntaxiquement en Java on utilise :
  - **executeQuery**(...) si la requête est une requête SELECT
  - **executeUpdate**(...) si a requête est une requête SQL LDD ou LMD.

programme JDBC

Les requêtes

- On utilise l'une des trois interfaces **Statement**, **PreparedStatement** ou **CallableStatement**.
  - Les instructions PreparedStatement sont précompilées.
  - On utilise CallableStatement pour lancer une procédure du SGBD.
- Avec un Statement on a :

```
Statement smt = conX.createStatement();
ResultSet rs = smt.executeQuery( "SELECT * FROM
    Livres" );
```

programme JDBC

Récupération des résultats

- Les résultats des requêtes SELECT sont mis dans un **ResultSet**.
- Cet objet récupéré modélise le résultat qui peut être vu comme une table.
- Un pointeur géré par Java JDBC permet de parcourir tout le ResultSet et est initialisé : il est automatiquement positionné avant la première ligne de résultat.

programme JDBC

Récupération des résultats

- On parcourt tout le ResultSet pour avoir l'ensemble des réponses à la requête.
- La boucle de parcours est :

```
while( rs.next() ) {
// traitement
}
```

- Les colonnes demandées par la requête sont numérotées à partir de 1

programme JDBC

Récupération des résultats

- Rq :
  - La numérotation est relative à l'ordre des champs de la requête et non pas l'ordre des champs de la table interrogée.
  - Les colonnes sont typées en type SQL. Pour récupérer une valeur de colonne il faut donc indiquer le numéro de la colonne ou son nom et faire la conversion de type approprié

programme JDBC

Récupération des résultats

- Par exemple :

```
Statement stmt = ...;
ResultSet rs = stmt.executeQuery(
    SELECT nom, prenom, age, date FROM LaTable");
…
String leNom = rs.getString( 1 ) ;
String lePrenom = rs.getString( 2 );
int lage = rs.getInt ( 3 );
Date laDate = rs.getDate( 4);
```

programme JDBC

Récupération des résultats

- On peut aussi désigner les colonnes par leur nom (c'est moins rapide mais plus lisible) et réécrire :

```
ResultSet rs = stmt.executeQuery("SELECT nom,
    prenom, age, date FROM LaTable");
…
String leNom = rs.getString( "nom") ;
String lePrenom = rs.getString( "prenom" );
int lage = rs.getInt ( "age" );
Date laDate = rs.getDate( "date ");
```

programme JDBC

Récupération des résultats

- Certaines correspondances entre type SQL et type Java :

SQL type	Java Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

programme JDBC : Fonctionnement

- Après l'instruction

Connection **conX** =
DriveManager.getConnection(URLjdbc,

"dbUser1", "pwuser1");

un pilote à la base de données a été trouvé par le programme (et la classe DriverManager) parmi les nombreux pilotes qui ont pu être chargés à la suite d'instructions

Class.forName("piloteToBD");

programme JDBC : Fonctionnement

- Ce pilote a été capable de faire une connexion à la base et de retourner un objet d'une classe (oracle ou autre) qui implémente l'interface **Connection**.
- On peut alors repéré cet objet par une référence d'interface **Connection** (polymorphisme !!).
- Voila pourquoi le code que nous écrivons est universel et permet d'accéder à tout type de base.

programme JDBC : Fonctionnement

- Par la suite cet objet repéré par **conX** va retourner, à la suite de l'instruction :

Statement smt = conX.createStatement();

un objet d'une classe qui implémente l'interface Statement et qui est propre à la base (oracle ou autre), peu importe l'essentiel et qu'on puisse alors repérer cet objet par une référence d'interface Statement

programme JDBC : Fonctionnement

- Enfin l'instruction

ResultSet rs = smt.executeQuery("SELECT * FROM Livres");

demande le lancement de la méthode executeQuery(...) sur un objet d'une classe qui implémente Statement et qui est propre à la base et cela fonctionne et est universel (pour la troisième fois ;-)).

JDBC : PreparedStatement

- Lors de l'envoi d'une requête pour exécution 4 étapes doivent être faites :
  - analyse de la requête
  - compilation de la requête
  - optimisation de la requête
  - exécution de la requête
- ceci même si cette requête est la même que la précédente !! Or les 3 premières étapes ont déjà été effectuées dans ce cas.

JDBC : PreparedStatement

- Les BD définissent la notion de requête préparée, requête où les 3 premieres étapes ne sont effectuées qu'une seule fois.
- JDBC propose l'interface **PreparedStatement** pour modéliser cette notion.
- Cette interface dérive de l'interface **Statement**.
- PreparedStatement permet de construire des requêtes paramétrées (un des arguments est une variable) (impossible avec Statement).

JDBC : PreparedStatement

- Même sans paramêtres, les PreparedStatement ne s'utilisent pas comme des Statement :

```
PreparedStatement pSmt = conX.prepareStatement ( "SELECT* FROM Livres" );
ResultSet rs = pSmt.executeQuery();
```

- la requête est décrite au moment de la "construction" pas lors de l'exécution qui est lancée par executeQuery() sans argument.

JDBC : PreparedStatement

Requêtes paramétrées

- On utilise donc les PreparedStatement et on écrit

des requêtes de la forme :

SELECT nom FROM Personnes WHERE age > ?
AND adresse = ?

- Puis on utilise les méthodes

set*Type(numéroDeLArgument, valeur)*

pour positionner chacun des arguments.

- Les numéros commencent à 1 dans l'ordre d'apparition

JDBC : PreparedStatement

Requêtes paramétrées

- On a donc un code comme :

```
PreparedStatement pSmt = conX.prepareStatement(
        "SELECT nom FROM Personnes
        WHERE age > ? AND adresse = ?" );
pSmt .setInt(1, 22);
pSmt .setString(2, "Oujda");
ResultSet rs = pSmt.executeQuery();
```

JDBC : CallableStatement

- Certaines requêtes pour une base de données sont si courantes qu'elles sont intégrées dans la base de données.
- On les appelle des procédures stockées et elles sont modélisées en JDBC par l'interface **CallableStatement** qui dérive de PreparedStatement.
  - Elle peut donc avoir des paramétres.

JDBC : CallableStatement

- On lance l'éxécution d'une procédure stockée à l'aide de la syntaxe :

  {call nom_Procedure[(?, ?, ...)]}

- Pour une procédure retournant un résultat on a la syntaxe :

  {? = call nom_Procedure[(?, ?, ...)]}

- La syntaxe pour une procédure stockée sans paramêtres est :

  {call nom_Procedure}

JDBC : CallableStatement

- Exp : Procédure

```
CallableStatement cStmt =
conX.prepareCall("{call setVille(?, ?) }");
cStmt.setString(1, "Ahmed");
cStmt.setString(2, "Rabat");
cSmt.executeUpdate();
```

JDBC : CallableStatement

- Exp : Fonction

```
CallableStatement cs= uneConnection.prepareCall(
                              "{?=call some_sal(?)}" );
   cs.registerOutParameter(1, Types.NUMERIC);
   cs.setInt(2, 30);
   cs.executeUpdate();
   some = cs.getDouble(1);
```

JDBC : les MetaData

- On peut avoir des informations sur la BD elle même. Ces informations sont appelées des métadatas.
- On obtient un objet de la classe **DatabaseMetaData** à partir de la connexion par :

```
DatabaseMetaData dmd = conX.getMetaData();
```

- À partir de là, cette classe fournit beaucoup de renseignements sur la base par exemple :

```
String nomSGBD = dmd.getDatabaseProductName();
```

JDBC : les MetaData

- De même on peut avoir des métadatas sur un ResultSet :

```
//Affichage des colonnes de la table
rs = smt.executeQuery( "SELECT * FROM Livres " );
ResultSetMetaData rsmd = rs.getMetaData();
int numCols = rsmd.getColumnCount();
for ( int i = 1; i <= numCols; i++ )
System.out.println( "\t" + i + " : " +
rsmd.getColumnLabel( i ) );
```