Java
Conexão com Banco de Dados

Alex Coelho

Introdução

- funcionalidade primordial em qualquer sistema;
- “é a habilidade para comunicar-se com um repositório de dados”.
- A Linguagem Java possui uma *Application Programming Interface* (API) que possibilita o acesso a BDs;
- um modelo de conexão uniforme;
- API é a *Java DataBase Connectivity* (JDBC)

JDBC

- O que é a JDBC?
  - consiste em uma biblioteca;
  - implementada em Java;
  - disponibiliza classes e interfaces para o acesso ao banco de dados;
- Para cada banco de dados existe uma implementação: Drives, no caso interfaces que devem ser implementadas;
- São interfaces porque levam em considerações particularidades;

Tipos de JDBC

- JDBC-ODBC:
  - Também conhecido como Ponto JDBC-ODBC;
  - restrito à plataforma *Windows*;
  - utiliza ODBC para conectar-se com o banco de dados;
  - converte métodos JDBC em chamadas às funções do ODBC;
  - geralmente é usado quando não há um *driver* “puro-java”;

Tipos de JDBC

- Driver API-Nativo:
  - traduzir as chamadas realizadas com o JDBC para realizadas com a API cliente do banco de dados utilizado;
  - Funciona +- como a JDBC-ODBC;
  - pode ser que sejam necessários que outras soluções;

Tipos JDBC

- Driver de Protocolo de Rede:
  - Converte a chamada por meio do JDBC para um protocolo de rede;
  - Independe do banco de dados que está sendo utilizado;
  - devido ao fato de se utilizar de protocolos que não se prendem a um determinado banco de dados;
  - modelo de conexão mais flexível e abrangente;

Tipos JDBC

- Driver nativo:
  - converte as chamadas JDBC diretamente no protocolo do banco de dados;
  - é implementado na linguagem Java;
  - normalmente independe de plataforma;
  - escrito pelos próprios desenvolvedores;
  - muito flexível;
  - tipo mais recomendado para ser usado e mais utilizado no mercado;

Pacote java.sql

- fornece a API para acesso e processamento de dados;
- geralmente acessa uma base de dados relacional;
- principais classes e interfaces:
  - DriverManager, responsável por criar uma conexão com o banco de dados;
  - Connection, classe responsável por manter uma conexão aberta com o banco;
  - Statement, gerencia e executa instruções SQL;
  - PreparedStatement, gerencia e executa instruções SQL, permitindo também a passagem de parâmetros em uma instrução;
  - ResultSet, responsável por receber os dados obtidos em uma pesquisa ao banco.

DriverManager

- responsável pelo gerenciamento de *drivers* JDBC;
- estabelece conexões a bancos de dados;

```
// Carregando um driver em tempo de execução
Class.forName("org.gjt.mm.mysql.Driver");

// Tentando estabelecer conexão com o Banco de Dados
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/aula?
                                                autoReconnect=true", "conta","senha");
```

- retorna uma implementação para a interface *Connection;*

Connection

- representa a conexão com o banco de dados;
- proporcionar informações sobre as tabelas do banco através de transações;

- métodos desta interface freqüentemente utilizados (SUN, 2007):
  - *commit()*, executa todas as alterações feitas com o banco de dados pela atual transação.
  - *rollback()*, desfaz qualquer alteração feita com o banco de dados pela atual transação.
  - *close()*, libera o recurso que estava sendo utilizado pelo objeto.
- Se preocupa em como o banco irá se comportar;

Statement

- Implementação de uma Interface que fornece métodos para executar uma instrução SQL;
- Não aceita a passagem de parâmetros;

- principais métodos da Interface *Statement* são (SUN, 2007):
  - *executeUpdate()*, executa instruções SQL do tipo: *INSERT*, *UPDATE* e *DELETE*;
  - *execute()*, executa instruções SQL de busca de dados do tipo *SELECT*;
  - *close()*, libera o recurso que estava sendo utilizado pelo objeto.

```
// Instanciando o objeto statement (stmt)
Statement stmt = conn.createStatement();

// Executando uma instrução SQL.
stmt.executeUpdate("INSERT INTO ALUNO VALUES (1, 'Pedro da Silva')");
```

PreparedStatement

- A interface PreparedStatement possui todos os recursos da interface Statement;
- acrescentando a utilização de parâmetros em uma instrução SQL;

- métodos da interface PreparedStatement são (SUN, 2007):
  - *execute()*, consolida a instrução SQL informada;
  - *setDate()*, método utilizado para atribuir um valor do tipo Data;
  - *setInt()*, utilizado para atribuir valores do tipo inteiro;
  - *setString()*, método utilizado para atribuir valores do tipo Alfa Numéricos.

PreparedStatement

```
// Instanciando o objeto preparedStatement (pstmt)
PreparedStatement pstmt =
        conn.prepareStatement("UPDATE ALUNO SET NOME = ?");
// Setando o valor ao parâmetro
pstmt.setString(1, “MARIA RITA”);
```

ResultSet

- Esta interface permite o recebimento e gerenciamento do conjunto de dados resultante de uma consulta SQL;
- métodos capazes de acessar os dados;
- métodos desta interface freqüentemente utilizados (SUN, 2007):
  - *next()*, move o cursor para a próxima linha de dados, já que o conjunto de dados retornados pela consulta SQL é armazenado como em uma lista.
  - *close()*, libera o recurso que estava sendo utilizado pelo objeto.
  - *getString(String columnName)*, recupera o valor da coluna informada como parâmetro, da linha atual do conjunto de dados recebidos pelo objeto ResultSet.

ResultSet

```
//Recebendo o conjunto de dados da consulta SQL
ResultSet rs = stmt.executeQuery("SELECT id, nome FROM ALUNO");

// Se houver resultados, posiciona-se o cursor na próxima linha de
    dados
while (rs.next()) {
    // Recuperando os dados retornados pela consulta SQL
    int id = rs.getInt("id");
    String nome = rs.getString("nome");
}
```

- métodos como o *getInt()*, *getString()* para recuperar os valores;

Exemplo

```
import java.awt.Frame;
import java.awt.List;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
    public class Aula {
            private Connection conn;
            private PreparedStatement pstmt;
            private Statement stmt;
            private ResultSet rs;
```

Exemplo

```
public void open() {
    try {
        Class.forName("org.gjt.mm.mysql.Driver");
        conn = DriverManager.getConnection(
                "jdbc:mysql://localhost/aula04?autoReconnect=true",
                "root", "");
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Exemplo

```
public void close() {
    try {
        if (stmt != null)
                stmt.close();
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Exemplo

```
public void inserir(){
    try {
        // Abrindo a conexão com o banco
        open();
        // Instanciando o objeto statement (stmt)
        stmt = conn.createStatement();
        // Executando uma instrução SQL.
        stmt.executeUpdate(
                "INSERT INTO ALUNO VALUES (1, 'Pedro da Silva')");
        // Fechando a conexão com o banco
        close();
    } catch (SQLException e) {
        // Fechando a conexão com o banco
        close();
        e.printStackTrace();
    }
}
```

Exemplo

```
public void alterar(){
    try {
        // Abrindo a conexão com o banco
        open();
        // Instanciando o objeto preparedStatement (pstmt)
        pstmt = conn.prepareStatement("UPDATE ALUNO SET NOME = ?
        Where id = 1");
        // Setando o valor ao parâmetro
        pstmt.setString(1, "MARIA RITA");
        // Fechando a conexão com o banco
        pstmt.execute();
        close();
    } catch (SQLException e) {
        // Fechando a conexão com o banco
        close();
        e.printStackTrace();
    }
}
```

Exemplo

```
public ResultSet buscarAlunos(){
    try {
            open();
            stmt = conn.createStatement();
            rs = stmt.executeQuery("SELECT id, nome FROM ALUNO");
            return rs;
    } catch (SQLException e) {
            close();
            e.printStackTrace();
    }
    return null;
}
```

Exemplo

```
public void imprimirAlunos() throws SQLException{
    JTextArea output = new JTextArea(20,30);
    Banco aula = new Banco();
    aula.rs = aula.buscarAlunos();

    while (aula.rs.next()){
        output.setText("Id: " + aula.rs.getInt("id") + " - " + "Nome: " +
                        aula.rs.getString("nome")+"\n");
    }

    JFrame janela = new JFrame("Janela");
    janela.setLocation(300, 300);
    janela.setSize(300, 300);

    janela.add(new JScrollPane(output));
    janela.setVisible(true);
}
```

Exemplo

```
public static void main(String[] args ){
    Aula aula = new Aula();
    aula.inserir();
    aula.alterar();
    aula.rs = aula.buscarAlunos();
    try {
        aula.imprimirAlunos();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Referências

DEITEL, Harvey M. **Java:** como programar. 6 ed. São Paulo: Pearson Prentice Hall, 2005.

THOMPSON, Marco Aurélio. **Java 2 & banco de dados**. São Paulo: Érica, 2002.

SUN, MicroSystem. Java Technology. Disponivel em http://java.sun.com. Acessado em 12/10/2007.