Работа с БД из Java-программ: JDBC

www.spro-club.ru

Необходимые условия

- Пакеты java.sql и javax.sql содержат классы и интерфейсы для работы с БД
- Для подключения к конкретной СУБД требуется специальная библиотека, называемая драйвером JDBC
- Написано большое количество драйверов для большинства СУБД как самими разработчиками, так и сторонними производителями

Загрузка драйвера

- Первый шаг для работы приложения с БД – это загрузка и регистрация необходимого класса драйвера (JAR-библиотеки)
- Скачать его можно с официального сайта СУБД или из других источников
- Затем библиотеку драйвера нужно «привязать» к проекту – указать путь к нему в параметре CLASSPATH

Основные компоненты для работы с БД

- Интерфейс Connection («соединение»):
  - в результате успешного подключения к БД создаётся объект, описывающий данное соединение
  - затем программе возвращается ссылка на этот объект как на «экземпляр» Connection
  - вся дальнейшая работа с БД ведётся с использованием этой ссылки на подключение

Основные компоненты для работы с БД

- Интерфейс Statement («выражение»):
  - объекты классов, реализующих этот интерфейс, используются для подготовки и выполнения SQL-запросов к БД
- Интерфейс ResultSet («результирующий набор»):
  - такие объекты содержат данные, извлечённые из БД в результате выполнения SELECT-запросов с помощью объектов Statement
  - представляет собой, по сути, коллекцию строк

Исключения при работе с БД

- Главный класс исключений – SQLException (наследник Exception)
- Выбрасывается (throws) в результате ошибок, связанных с некорректными SQL-запросами и т.п.

Алгоритм работы с БД

1. Регистрация драйвера JDBC.
2. Создание подключения (Connection).
3. Создание и подготовка объекта типа Statement (выражение, SQL-запрос)
4. Выполнение SQL-запроса, подготовленного с помощью Statement
5. Обработка результатов (например, с помощью ResultSet в случае запроса SELECT)
6. Закрытие подключения и освобождение использованных ресурсов

Регистрация драйвера

- Может осуществляться разными способами
- Например, с помощью класса java.sql. DriverManager
- Прямая загрузка класса драйвера:
  - требует явного описания типа, загружаемого драйвера в исходном коде приложения
  - требуется сначала импортировать соответствующий пакет, а потом загрузить драйвер:

```
import com.mysql.jdbc.Driver;
...
DriverManager.registerDriver(new com.mysql.jdbc.Driver());
```

  - недостаток: при изменении драйвера (смене СУБД) требуется изменение исходного кода программы

Загрузка драйвера по имени

- Более гибкий способ, широко распространён
- Может выбрасывать исключение ClassNotFound «Класс не найден»

```
String driverName = "com.mysql.jdbc.Driver”;
. . .
try {  Class.forName( driverName ).newInstance();
 } catch (ClassNotFoundException e) {
       System.err.println (
    “Ошибка: Драйвер " + driverName + " не найден");
       System.exit(0);
 }
```

Установка подключения

- класс DriverManager – объект управления JDBC, посредник между программой и драйвером
- класс-метод DriverManager.getConnection() устанавливает соединение с БД и возвращает ссылку типа Connection
- при вызове getConnection() происходит перебор всех загруженных в данный момент драйверов для использования подходящего

Установка подключения

- метод getConnection() может принимать различные параметры (имеет перегруженные версии):

```
public static Connection getConnection( String url )
    throws SQLException
```

- url – адрес БД (строка подключения), логин и пароль могут быть переданы прямо в URL

```
public static Connection getConnection( String url,
                                        String user,
                                        String password )
                throws SQLException
```

- передача логина и пароля через отдельные параметры

Настройка подключения

- Добавление свойств подключения в объект класса Properties и вызов метода getConnection

```
 Properties connInfo = new Properties();
        connInfo.put("user", user);
        connInfo.put("schema", scheme);
        connInfo.put("password", passwd);

con = DriverManager.getConnection(dbUrl, connInfo);
```

Типы запросов

- Запросы на выборку данных (SQL-оператор Select) – возвращают результирующий набор данных (объект ResultSet)
- Запросы модификации данных (DML) (операторы Insert, Update, Delete) – возвращают целое число – количество обработанных строк

Выполнение запросов

- Требуется получить ссылку на объект типа Statement (выражение) из подключения (объекта Connection):

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

- SQL-запросы выполняются с помощью перегруженных методов **execute** класса Statement:

```
println("Результ "+stmt.execute("SELECT * FROM CUSTOMER"));
```

- Можно выполнить сразу несколько запросов с помощью int[] executeBatch()

Получение результата

- Для обработки результатов запроса на выборку надо получить ссылку на объект типа ResultSet (результирующий набор, курсор):

```
ResultSet rst = stmt.getResultSet();
или ResultSet rst = stmt.executeQuery("Select * from tab");
```

- Результирующий набор – это коллекция (таблица), имеющая несколько строк и столбцов
- Для последовательного перехода по строкам используются методы next(), previous(), absolute()
- Для извлечения данных используются методы

```
XXX getXXX(int index); //где XXX – тип, напр. getInt(…)
```

Навигация по курсору

- boolean next() – переход к следующей строке набора. Если записей больше нет, возвращает false
- boolean previous() – переход к предыдущей строке набора
- boolean absolute(int row) – абсолютное позиционирование курсора на заданную строку
- boolean relative(int rows) – относительное позиционирование курсора. rows – число строк, на которое нужно перейти вперёд или назад (может быть положительным или отрицательным)

Навигация по курсору

- int findColumn(String columnLabel) – возвращает номер столбца с указанным именем

Извлечение и запись данных

- существует множество методов вида:

```
XXX getXXX(int index); // index – номер столбца начиная с 1
XXX getXXX(String label) // label – имя столбца
XXX – тип данных, например getString(int col);
```

- Данные извлекаются из текущей строки
- Для записи значений в набор используется множество методов вида:

```
void updateXXX(int colIndex, XXX val)
например, updateByte(int columnIndex, byte x)
```

- После установки значения следует вызвать метод

```
void updateRow(); // обновление текущей строки в БД
```

Пример обновления и вставки

- Обновление значения в столбце:

```
rs.absolute(5); // moves the cursor to the fifth row of rs
rs.updateString("NAME", "AINSWORTH"); // updates the
      // NAME column of row 5 to be AINSWORTH
rs.updateRow(); // updates the row in the data source
```

- Вставка новой строки с 3-мя столбцами:

```
rs.moveToInsertRow(); // moves cursor to the insert row
rs.updateString(1, "AINSWORTH"); // updates the
         // first column of the insert row to be AINSWORTH
rs.updateInt(2,35); // updates the second column to be 35
rs.updateBoolean(3, true); // записывает в 3-й столбец true
rs.insertRow();
rs.moveToCurrentRow();
```

Пример навигации и чтения

```
// цикл по всем строкам результирующего набора
while (rst.next()) {
    System.out.printf("номер строки %3d:", rst.getRow());

    // получение значения столбцов по номеру и имени
столбца
    int id = rst.getInt(1);
    String nm = rst.getString("NAME");
    String city = rst.getString("CITY");

    System.out.printf("%10d | %30s | %10s\n",id,nm,city);
}
```

Получение метаданных

- Метаданные – информация о структуре результирующего набора – именах столбцов, типов их данных и т.п.
- Для получения метаданных используется метод getMetaData() результирующего набора (ResultSet), кот. возвращает ссылку на объект типа ResultSetMetaData

```
ResultSetMetaData meta=(ResultSetMetaData) rst.getMetaData();
```

- Эти данные могут быть полезны для настройки интерфейса пользователя (заголовки таблиц, ширина столбцов и т.п.)

Методы класса ResultSetMetaData

- int getColumnCount() – число столбцов в наборе
- String getColumnName(int column) – имя указанного столбца (нумерация с 1)
- String getColumnTypeName(int column) – наименование типа указанного столбца
- int getColumnType(int column) – возвращает тип столбца (целое) - одну из констант класса java.sql.Types
- String getTableName(int column) – имя таблицы, которой принадлежит указанный столбец набора (актуально при запросах к нескольким таблицам)

Методы класса ResultSetMetaData

- boolean isAutoIncrement(int column) – является ли указанный столбец автоинкрементным (счётчиком, увеличивающимся автоматически)
- int isNullable(int column) – может ли столбец содержать пустые значения (NULL)
- int getColumnDisplaySize(int column) – максимальная ширина столбца в символах
- Все методы могут выбрасывать исключение **SQLException**

Пример

```
// информация о столбцах ResultSet-а
System.out.println("\n\nResultSet metadata info:");
ResultSetMetaData meta = (ResultSetMetaData)
rst.getMetaData();
int n = meta.getColumnCount();
for (int i=1;i<=n;i++)  {
    System.out.printf("%d = %s <%s>",
            i,
            meta.getColumnName(i),
            meta.getColumnTypeName(i));
    if (meta.isNullable(i)==meta.columnNoNulls)
System.out.print(" NOT NULL");
    if (meta.isAutoIncrement(i)) System.out.print(" AUTO");
    System.out.println("");
}
```

Подготовленные выражения

- Служат для повышения эффективности многократного выполнения однотипных запросов (с разными параметрами)
- Для работы с ними используется объект класса PreparedStatement

Пример

```
PreparedStatement pst=null;
pst = conn.prepareStatement("select * from Customer where
City like ?");
if (pst != null) {
    // 1-е использование
    pst.setString(1, "%New%");
    rst = pst.executeQuery();
    while (rst.next()) {
     System.out.printf("%10d| %30s| %10s\n", rst.getInt(1),
       rst.getString("NAME"),
rst.getString("CITY"));
    }
    // 2-е использование
    pst.setString(1, "%San%");
    rst = pst.executeQuery();
    while (rst.next()) {
     System.out.printf("%10d| %30s| %10s\n",rst.getInt(1),
        rst.getString("NAME"), rst.getString("CITY"));
    }
}
```

Транзакции

- Транзакция – это механизм выполнения составных запросов (операций), позволяющий сохранить изменения **только** в случае **успешного** выполнения **всех** операций, входящих в набор
- Пример: перевод денег с одного счёта на другой состоит из 2х операций (запросов Update) – списание со счёта отправителя и зачисление на счёт получателя
- Работают по принципу «всё или ничего»
- Являются неотъемлемой частью профессиональной работы с БД, особенно в крупных и ответственных проектах

Автоматическая фиксация изменений

- По умолчанию в JDBC принята автоматическая фиксация каждого изменения
- Можно изменить это поведение с помощью метода:
- conn.setAutoCommit(false);
- В этом случае для сохранения изменений требуется явным образом вызывать метод commit(), а для их отмены – метод rollback() интерфейса Connection
- Такое поведение позволяет управлять транзакциями

Пакетное выполнение

- Класс PreparedStatement содержит методы для пакетного выполнения DML-запросов (на изменение данных)
- Это актуально при отключении авто-обновления
- Для этого после подготовки выражения (prepareStatement) вместо выполнения вызывается метод addBatch()
- Для выполнения пакета запросов вызывается метод executeBatch(), кот.возвращает массив целых чисел – кол-во обработанных строк по каждому запросу

Типы результирующих наборов

- При создании выражения (statement) можно указать тип результирующего набора:
- stmt = con.createStatement(resultSetType, resultSetConcurrency)
- где resultSetType – тип результрующего набора:
  - ResultSet.TYPE_FORWARD_ONLY
  - TYPE_SCROLL_INSENSITIVE
  - TYPE_SCROLL_SENSITIVE
- resultSetConcurrency – возможность изменения набора:
  - ResultSet.CONCUR_READ_ONLY – работа с набором только в режиме чтения данных
  - CONCUR_UPDATABLE – возможность изменения набора (вставки и обновления записей)