Шестая лекция
java for web
JDBC

Что такое JDBC

JDBC (Java DataBase Connectivity) - это прикладной программный интерфейс (далее API) для выполнения SQL-запросов. Он состоит из множества классов и интерфейсов, написанных на языке Java.

JDBC предоставляет стандартный API для разработчиков, использующих базы данных (далее БД). С помощью JDBC можно писать приложения на языке Java, использующие БД.

С помощью JDBC легко отсылать SQL-запросы почти ко всем реляционным БД. Другими словами, использование JDBC API избавляет от необходимости для каждой СУБД писать свое приложение.

Достаточно написать одну единственную программу, использующую JDBC API, и эта программа сможет отсылать SQL-запросы к требуемой БД. Кроме того, это проложение будет переносимо на различные платформы.

Структура JDBC

С точки зрения разработчика можно считать, что JDBC состоит из двух основных частей:

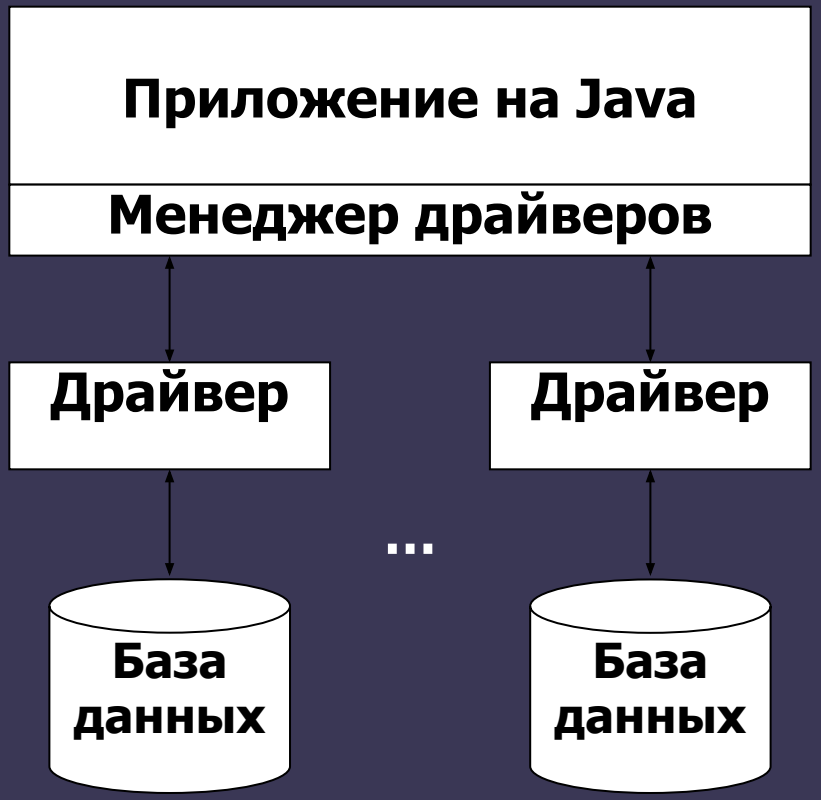
JDBC API, который содержит набор классов и интерфейсов, определяющих Java – ориентированный доступ к базам данных. Эти классы и методы объявлены в двух пакетах (package) java.sql и javax.sql

JDBC-драйвера, специфического для каждой базы данных (или других источников данных) JDBC превращает (тем или иным способом) вызовы уровня JDBC API в "родные" команды того или иного сервера баз данных.

Другими словами.

В задачу менеджера драйверов также входит присоединение Java-приложений к требуемому драйверу JDBC

Драйвер поддерживает обмен данными между приложением и базой данных



Основные классы и интерфейсы JDBC

java.sql.DriverManager - позволяет загрузить и зарегистрировать необходимый JDBC-драйвер, а затем получить соединение с базой данных.

java.sql.Connection - обеспечивает формирование запросов к источнику данных и управление транзакциями. Предусмотрены также интерфейсы javax.sql.PooledConnection (логическое соединение с БД из пула соединений) и javax.sql.XAConnection (логическое соединение с БД из пула, сопоставленное с внешней транзакцией).

java.sql.Statement , java.sql.PreparedStatement и java.sql.CallableStatenient - эти интерфейсы позволяют отправить запрос к источнику данных. Различные виды интерфейсов применяются в зависимости от того, используются в запросе параметры или нет и является ли запрос обращением к хранимой процедуре реляционной базы данных.

java.sql.ResultSet - объявляет методы, которые позволяют переме щаться по набору данных, возвращаемых оператором SELECT, и считывать значения отдельных полей в текущей записи

Основные типы данных JDBC

Так как SQL-типы и Java-типы данных не идентичны, то необходим какой-либо механизм передачи данных между Java-приложением и СУБД.
К счастью, программистам не обязательно отягощать себя именами типов SQL, используемых в таблицах БД.

JDBC поддерживает взаимное отображение между типами данных, характерных для использования SQL, и их Java-аналогами.

При программировании с использованием JDBC API программисты могут использовать эти JDBC-типы для обращения к базовым типам SQL незаботясь о том, какие имена типов данных использовались при создании БД.

В таблице приведены соответствия между основными типами.

JDBC Type	Java Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
BOOLEAN	Boolean
INTEGER	Int
BIGINT	Long
…	…

Соединение

Регистрация драйверов осуществляется классом DriverManager. Он содержит информацию о всех зарегистрированных драйверах.

Метод getConnection на основании параметра URL находит java.sql.Driver соответствующей базы данных и вызывает у него метод connect.

Задача класса DriverManager - обеспечить поиск нужного JDBC-драйвера среди всех доступных при поступлении запроса клиента, который содержит URL нужной базы данных.

Во всех примерах подключения к базе данных вы обязательно встретите эти строки:

```
Class.forName(driverClass);

Connection connection = DriverManager.getConnection(url, user, password) ;
```

Вызов Class.forName загружает класс , инициализирует, и осуществляет регистрацию драйвера в DriverManager.

Пример для MySQL:

```
Class.forName("com.mysql.jdbc.Driver");

Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/stunet3?user=ИМЯ&password=
GПАРОЛЬ&characterEncoding=UTF-8");
```

Выполнение SQL-команд

В **JDBC** есть три класса для посылки SQL-запросов в БД и три метода в интерфейсе **Connection** создают экземпляры этих классов.

Statement - создается методом **createStatement**. Объект Statement используется при простых SQL-запросах.

PreparedStatement - создается методом **prepareStatement**. Объект PreparedStatement используется в SQL-запросах с одним или более входными параметрами (IN parameters). PreparedStatement содержит группу методов, устанавливающих значения входных параметров, которые отсылаются в БД при выполнении запроса. Экземпляры класса PreparedStatement расширяют (наследуются от) Statement и, таким образом, включают методы Statement. Объект PreparedStatement потенциально может быть более эффективным, чем Statement, так как он прекомпилируется и сохраняется для будущего использования.

CallableStatement - создается методом prepareCall. Объекты CallableStatement используются для выполнения т.н. хранимых процедур - именованных групп SQL-запросов, наподобие вызова подпрограммы. Объект CallableStatement наследует методы обработки входных (IN) параметров из PreparedStatement, а также добавляет методы для обработки выходных (OUT) и входных-выходных (INOUT) параметров.

Другими словами: Интерфейс **Statement** предоставляет базовые методы для выполнения запросов и извлечения результатов. Интерфейс **PreparedStatement** добавляет методы управления входными (IN) параметрами; **CallableStatement** добавляет методы для манипуляции выходними (OUT) параметрами.

Создание объектов Statement

Как только соединение с определенной БД установлено, оно может использоваться для выполнения SQL-запросов. Объект Statement создается методом Connection.createStatement, как показано ниже:

```
try {
Class.forName("com.mysql.jdbc.Driver");
conn = DriverManager.getConnection("url");
        loadAllStudents = conn.createStatement();
        List<Student> studentsArray = new LinkedList<Student>();
           rs = loadAllStudents.executeQuery("SELECT * FROM students");
           while (rs.next()) {
              Student newStudent = new Student();
              newStudent.setIdStudent(rs.getInt("id"));
              newStudent.setNameStudent(rs.getString("name"));
              newStudent.setSurnameStudent(rs.getString("surname"));
              studentsArray.add(newStudent);
          }
      } catch (Exception e) {
         e.printStackTrace();
      }
```

Методы выполнения SQL-команд

Интерфейс **Statement** предоставляет три различных метода выполнения SQL-выражений: executeQuery, executeUpdate и execute, в зависимости от SQL-запроса.

executeQuery() - для запросов, результатом которых является один единственный набор значений, таких как запросов SELECT. Метод возвращает набор данных, полученный из базы

executeUpdate() - для выполнения операторов INSERT, UPDATE или DELETE, а также для операторов DDL (Data Definition Language). Метод возвращает целое число, показывающее, сколько строк данных было модифицировано

execute() – исполняет SQL-команды, которые могут возвращать различные результаты. Например, может использоваться для операции CREATE TABLE, и т.д.

Все методы выполнения SQL-запросов закрывают предыдущий набор результатов (result set) у данного объекта Statement. Это означает, что перед тем как выполнять следующий запрос над тем же объектом Statement, надо завершить обработку результатов предыдущего (ResultSet).

Наборы данных

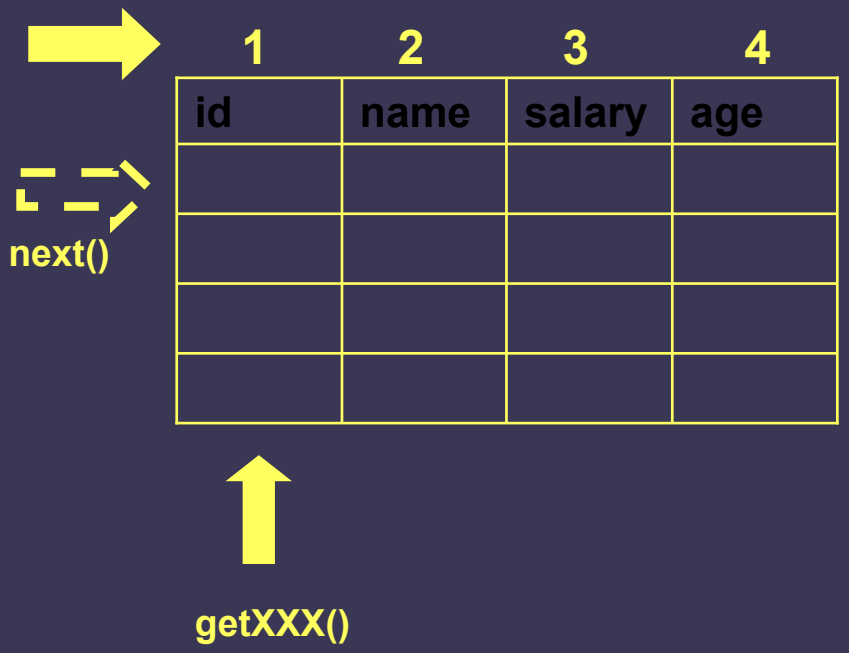
Метод executeQuery() возвращает объект с интерфейсом ResultSet, который хранит в себе результат запроса к базе данных.

В наборе данных есть курсор, который может указывать на одну из строк таблицы, эта строка называется текущей.

Курсор перемещается по строкам при помощи метода next().

Сразу после получения набора данных его курсор находится перед первой строкой. Чтобы сделать первую строку текущей надо вызвать метод **next()**

Поля текущей записи (колонки таблицы) доступны программе при помощи методов интерфейса ResultSet: getInt() getFloat(), getString(), getDate() и им подобных.



Интерфейс PreparedStatement

Экземпляры PreparedStatement сохраняют скомпилированные SQL-выражения.

Особенностью SQL-выражений в PreparedStatement является то, что они могут иметь параметры

- Параметризованное выражение содержит знаки вопроса в своем тексте.
  - Например: “SELECT * FROM students WHERE id=?”
- Перед выполнением запроса значение каждого вопросительного знака явно устанавливается методами setXxx()
  - Например: ps.setInt(1, 30);

Использование PreparedStatement приводит к более быстрому выполнению запросов при их многократном вызове с различными параметрами.

Пример использования PreparedStatement

```
public Student getAllStudentId(int id) {
rs = null;
Student student = new Student();
try {
        getStudById.setInt(1, id);
        rs = getStudById.executeQuery();
        if (rs.next()) {
          student.setIdStudent(rs.getInt("id"));
          student.setNameStudent(rs.getString("name"));
          student.setSurnameStudent(rs.getString("surname"));
          student.setDate(rs.getDate("date"));
          student.setGroupe(rs.getString("groupe"));
        }
} catch (SQLException e) {
e.printStackTrace();
}
        return student;
}
```

Интерфейс CallableStatement

Интерфейс CallableStatement используется, когда необходимо обратиться к хранимой процедуре.

Работа с интерфейсом CallableStatement усложняется несколькими обстоятельствами:

- Хранимые процедуры сильно отличаются (с точки зрения синтаксиса их вызова) для различных серверов баз данных
- Хранимые процедуры отличаются друг от друга тем, какие результаты они могут возвращать и как именно выполняется возврат.
  - Например, для некоторых серверов (точнее, их JDBC-драйверов) хранимые процедуры могут возвращать только out- (и inout-) параметры,
  - Для других серверов процедура возвращает out- и inout-параметры как поля в наборе записей (типа ResultSet).
  - Третьи поддерживают и тот, и другой подход

При работе с интерфейсом CallableStatement (который является производным от интерфейса PreparedStatement) широко используется escape-синтаксис, и программист задает команду обращения к процедуре либо с помощью вызова метода executeQuery(), либо с помощью executeUpdate().

В любом случае при работе с конкретным JDBC-драйвером нужно изучить, каким образом этот драйвер обеспечивает взаимодействие с хранимыми процедурами

Пример вызова хранимой процедуры.

Эта хранимая процедура имеет единственный входной аргумент типа integer, а в качестве выходного результата тоже возвращает целое число

Синтаксис вызова хранимой процедуры и способ трактовки out-аргументов являются специфическими для используемого JDBC-драйвера

```
CallableStatement pstmt = connection.prepareCall ("{call getUniqueValue (?)}");
// Настройка параметров и обращение к процедуре
pstmt.setInt (1, 1);
ResultSet rs =pstmt.executeQuery();
```

Работа с пулом соединений

При создании нового коннекта к БД может потребоваться значительное время, особенно если БД удаленная. Чтобы на каждое обращение к базе не открывать новое соединение используется пул соединений

Применение пулов соединений позволяет повысить производительность приложения за счет переиспользования объектов-соединений с БД.

В настоящее время большинство драйверов для работы с БД поддерживают работу с пулами соединений.

Существует несколько реализаций пулов подключений к БД. Можно, сделать и свою.