Конспект лекции
JDBC и работа с БД

Цель и задачи лекции

Цель – изучить принципы работы с JDBC.

Задачи:

1. Понять принципы ACID
2. Понять принцип выбора данных из БД с использование JDBC
3. Описать основные методы для работы с базой данных

План занятия

1. Связь с базами данных через JDBC
2. Драйверы JDBC
3. Использование файла ресурсов
4. Подготовленные запросы и хранимые процедуры
5. Транзакции

Связь с базами данных через JDBC

В основном информация хранится не в файлах, а в базах данных. Приложение должно уметь связываться с базой данных для получения из нее информации или для помещения информации в базу данных. Дело здесь осложняется тем, что СУБД (системы управления базами данных) сильно отличаются друг от друга и совершенно по-разному управляют базами данных. Каждая СУБД предоставляет собственный набор функций для доступа к базам данных, и приходится для каждой СУБД писать свое приложение.

Но что делать при работе по сети, когда неизвестно, какая СУБД управляет базой на сервере?

Выход был найден корпорацией Microsoft, создавшей набор интерфейсов ODBC (Open Database Connectivity) для связи с базами данных, оформленных как прототипы функций языка C. Эти прототипы одинаковы для любой СУБД, они просто описывают набор действий с таблицами базы данных. В приложение, обращающееся к базе данных, записываются вызовы функций ODBC. Для каждой системы управления базами данных разрабатывается так называемый драйвер ODBC, реализующий эти функции для конкретной СУБД. Драйвер просматривает приложение, находит обращения к базе данных, передает их СУБД, получает от нее результаты и подставляет их в приложение.

Идея оказалась очень удачной, и использование ODBC для работы с базами данных стало общепринятым. Компания Sun подхватила эту идею и разработала набор интерфейсов и классов,

названный JDBC, предназначенный для работы с базами данных. Эти интерфейсы и классы составили пакет java.sql, а также пакет javax.sql и его подпакеты, входящие в Java SE.

Драйверы JDBC

JDBC (Java DataBase Connectivity) – стандартный прикладной интерфейс (API) языка Java для организации взаимодействия между приложением и СУБД.

Кроме классов с методами доступа к базам данных для каждой СУБД необходим драйвер JDBC — промежуточная программа, реализующая интерфейсы JDBC методами данной СУБД. Драйверы JDBC могут быть написаны разработчиками СУБД или независимыми фирмами. В настоящее время написано несколько сотен драйверов JDBC для разных СУБД под разные их версии и платформы.

Это взаимодействие осуществляется с помощью драйверов JDBC, обеспечивающих реализацию общих интерфейсов для конкретных СУБД и конкретных протоколов. В JDBC определяются четыре типа драйверов:

Существуют четыре типа драйверов JDBC:

- драйвер, реализующий методы JDBC вызовами функций ODBC. Это так называемый мост (bridge) JDBC—ODBC. Непосредственную связь с базой при этом осуществляет драйвер ODBC, который должен быть установлен на той машине, на которой работает программа;
- драйвер, реализующий методы JDBC вызовами функций API самой СУБД. В этом случае на машине должен быть установлен клиент СУБД;
- драйвер, реализующий методы JDBC вызовами функций сетевого протокола, независимого от СУБД, например HTTP. Этот протокол должен быть, затем, реализован средствами СУБД;
- драйвер, реализующий методы JDBC вызовами функций сетевого протокола СУБД.

Использование JDBC

Порядок работы с БД из Java:

1. Подключение библиотеки с классом-драйвером базы данных. Дополнительно требуется подключить к проекту библиотеку, содержащую драйвер, поместив ее предварительно в папку /lib приложения.

Например, mysql-connector-java-[номер версии]-bin.jar для СУБД MySQL, ojdbc[номер версии].jar для СУБД Oracle.

2. Установка соединения с БД. Для установки соединения с БД вызывается статический метод getConnection() класса java.sql.DriverManager. В качестве параметров методу передаются URL базы данных, логин пользователя БД и пароль доступа. Загрузка класса драйвера базы данных при отсутствии ссылки на экземпляр этого класса в JDBC 4.1 происходит автоматически при установке соединения экземпляром DriverManager. Метод возвращает объект Connection. URL

базы данных, состоящий из типа и адреса физического расположения БД, может создаваться в виде отдельной строки или извлекаться из файла ресурсов. Соответственно:

```
Connection cn = DriverManager.getConnection("jdbc:mysql://localhost:3306/testphones",

"root", "pass");

Connection cn =
DriverManager.getConnection("jdbc:oracle:thin:@//localhost:1521:testphones",

"system", "pass");
```

В результате будет возвращен объект Connection и будет одно установленное соединение с БД с именем testphones. Класс DriverManager предоставляет средства для управления набором драйверов баз данных. С помощью метода getDrivers() можно получить список всех доступных драйверов.

До появления JDBC 4.0 объект драйвера СУБД нужно было создавать явно с помощью вызова соответственно:

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

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

или зарегистрировать драйвер

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

DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
```

В большинстве случаев в этом нет необходимости, так как экземпляр драйвера загружается автоматически при попытке получения соединения с БД.

3. Создание объекта для передачи запросов.

После создания объекта Connection и установки соединения можно начинать работу с БД с помощью операторов SQL. Для выполнения запросов применяется объект Statement, создаваемый вызовом метода createStatement() класса Connection.

```
Statement st = cn.createStatement();
```

Объект класса Statement используется для выполнения SQL-запроса без его

предварительной подготовки. Могут применяться также объекты классов

PreparedStatement и CallableStatement для выполнения подготовленных запросов и хранимых процедур.

4. Выполнение запроса.

Созданные объекты можно использовать для выполнения запроса SQL, передавая его в один из методов execute(String sql), executeBatch(), executeQuery(String sql) или executeUpdate(String sql). Результаты выполнения запроса помещаются в объект ResultSet:

```
/* выборка всех данных таблицы phonebook */

ResultSet rs = st.executeQuery("SELECT * FROM phonebook");
```

Для добавления, удаления или изменения информации в таблице запрос помещается в метод executeUpdate().

5. Обработка результатов выполнения запроса производится методами интерфейса ResultSet, где самыми распространенными являются next(), first(), previous(), last() для навигации по строками таблицы результатов и группа методов по доступу к информации вида getString(int pos), а также аналогичные методы, начинающиеся с getТип(int pos) (getInt(int pos), getFloat(int pos) и др.) и updateТип().

Среди них следует выделить методы getClob(int pos) и getBlob(int pos), позволяющие извлекать из полей таблицы специфические объекты (Character Large Object, Binary Large Object), которые могут быть, например, графическими или архивными файлами. Эффективным способом извлечения значения поля из таблицы ответа является обращение к этому полю по его имени в строке результатов методами типа int getInt(String columnLabel), String getString(String columnLabel),ObjectgetObject(StringcolumnLabel)и подобными им. Интерфейс располагает большим числом методов по доступу к таблице результатов, поэтому рекомендуется изучить его достаточно тщательно.

При первом вызове метода next() указатель перемещается на таблицу результатов выборки в позицию первой строки таблицы ответа. Когда строки закончатся, метод возвратит значение false.

6. Закрытие соединения, statement

```
st.close(); // закрывает также и ResultSet

cn.close();
```

После того, как база больше не нужна, соединение закрывается. Для того, чтобы правильно пользоваться приведенными методами, программисту требуется знать типы полей БД. В распределенных системах это знание предполагается изначально. В Java 7 для объектов-ресурсов, требующих закрытия, реализована технология try with resources.

Использование файла ресурсов

Параметры соединения можно задавать несколькими способами: с помощью прямой передачи значений в коде класса, а также с помощью файлов properties или xml. Окончательный выбор производится в зависимости от конфигурации проекта.

Чтение параметров соединения с базой данных и получение соединения следует вынести в отдельный класс. Класс ConnectorDB использует файл ресурсов database.properties, в котором хранятся, как правило, параметры подключения к БД, такие, как логин и пароль доступа. Например:

```
db.driver = com.mysql.jdbc.Driver
```

```
db.user = root

db.password = pass

db.poolsize = 32

db.url = jdbc:mysql://localhost:3306/testphones

db.useUnicode = true

db.encoding = UTF-8
```

В данном случае класс для подключения к БД будет выглядеть следующим образом:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.ResourceBundle;

public class ConnectorDB {
    public static Connection getConnection() throws SQLException {
        ResourceBundle resource = ResourceBundle.getBundle("database");
        String url = resource.getString("db.url");
        String user = resource.getString("db.user");
        String pass = resource.getString("db.password");
        return DriverManager.getConnection(url, user, pass);
    }
}
```

В таком случае получение соединения с БД сведется к вызову

```
Connection cn = ConnectorDB.getConnection();
```

Подготовленные запросы

Для представления запросов существует еще два типа объектов PreparedStatement и CallableStatement. Объекты первого типа используются при выполнении часто повторяющихся запросов SQL. Такой оператор предварительно готовится и хранится в объекте, что ускоряет обмен информацией с базой данных при многократном выполнении однотипных запросов. Второй интерфейс используется для выполнения хранимых процедур, созданных средствами самой СУБД.

При использовании PreparedStatement невозможен sql injection attacks. То есть если существует возможность передачи в запрос информации в виде строки, то следует использовать для выполнения такого запроса объект PreparedStatement.

Для подготовки SQL-запроса, в котором отсутствуют конкретные параметры, используется метод prepareStatement(String sql) интерфейса Connection, возвращающий объект PreparedStatement.

```
String sql = "INSERT INTO phonebook(idphonebook, lastname, phone) VALUES(?, ?, ?)";

PreparedStatement ps = cn.prepareStatement(sql);
```

Установка входных значений конкретных параметров этого объекта производится с помощью методов setString(int index, String x), setInt(int index, int x) и подобных им, после чего и осуществляется непосредственное выполнение запроса методами int executeUpdate(), ResultSet executeQuery().

```
public class DataBaseHelper {
    private final static String SQL_INSERT =
            "INSERT INTO phonebook(idphonebook, lastname, phone ) VALUES(?,?,?)";
    private Connection connect;
    public DataBaseHelper() throws SQLException {
        connect = ConnectorDB.getConnection();
    }
    public PreparedStatement getPreparedStatement(){
        PreparedStatement ps = null;
        try {
            ps = connect.prepareStatement(SQL_INSERT);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return ps;
    }
    public boolean insertAbonent(PreparedStatement ps, Abonent ab) {
        boolean flag = false;
        try {
            ps.setInt(1, ab.getId());
            ps.setString(2, ab.getName());
            ps.setInt(3, ab.getPhone());
            ps.executeUpdate();
            flag = true;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return flag;
    }
    public void closeStatement(PreparedStatement ps) {
        if (ps != null) {
            try {
                ps.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Транзакции

При проектировании распределенных систем часто возникают ситуации, когда сбой в системе или какой-либо ее периферийной части может привести к потере информации или к финансовым потерям. Простейшим примером может служить пример с перечислением денег с одного счета на другой. Если сбой произошел в тот момент, когда операция снятия денег с одного счета уже произведена, а операция зачисления на другой счет еще не произведена, то система, допускающая такие ситуации, должна быть признана не отвечающей требованиям заказчика. Или должны выполняться обе операции, или не выполняться вовсе. Такие две операции трактуют как одну и называют транзакцией.

Транзакция, или деловая операция, определяется как единица работы, обладающая свойствами ACID:

- Атомарность — две или более операций выполняются все или не выполняется ни одна. Успешно завершенные транзакции фиксируются, в случае неудачного завершения происходит откат всей транзакции.
- Согласованность — при возникновении сбоя система возвращается в состояние до начала неудавшейся транзакции. Если транзакция завершается успешно, то проверка согласованности удостоверяется в успешном завершении всех операций транзакции.
- Изолированность — во время выполнения транзакции все объекты-сущности, участвующие в ней, должны быть синхронизированы.
- Долговечность — все изменения, произведенные с данными во время транзакции, сохраняются, например, в базе данных. Это позволяет восстанавливать систему.

Для фиксации результатов работы SQL-операторов, логически выполняемых в рамках некоторой транзакции, используется SQL-оператор COMMIT.

В API JDBC эта операция выполняется по умолчанию после каждого вызова методов executeQuery() и executeUpdate(). Если же необходимо сгруппировать запросы и только после этого выполнить операцию COMMIT, сначала вызывается метод setAutoCommit(boolean param) интерфейса Connection с параметром false, в результате выполнения которого текущее соединение с БД переходит в режим неавтоматического подтверждения операций. После этого выполнение любого запроса на изменение информации в таблицах базы данных не приведет к необратимым последствиям, пока операция COMMIT не будет выполнена непосредственно. Подтверждает выполнение SQL-запросов метод commit() интерфейса Connection, в результате действия которого все изменения таблицы производятся как одно логическое действие. Если же транзакция не выполнена, то методом rollback() отменяются действия всех запросов SQL, начиная от последнего вызова commit(). В следующем примере информация добавляется в таблицу в режиме действия транзакции, подтвердить или отменить действия которой можно, снимая или добавляя комментарий в строках вызова методов commit() и rollback().

Литература и ссылки

1. Спецификация языка Java https://docs.oracle.com/javase/specs/jls/se8/html/index.html
2. Блинов И.Н., Романчик В.С., Java. Методы программирования. Минск, 2013.
3. Шилдт Г. Java 8. Полное руководство. 9-е издание. — М.: Вильямс, 2012. — 1377 с.
4. Эккель Б. Философия Java. 4-е полное изд. — СПб.: Питер, 2015. — 1168 с.
5. Эванс Бенджамин, Вербург Мартин. Java. Новое поколение разработки, 2014.

Вопросы для самоконтроля

1. В чем смысл транзакций?
2. Для чего нужен драйвер JDBC?
3. Опишите процесс выбора данных из БД.