


 enhorse / java-interview Public[Code](#) [Issues 9](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#) master ▾

...

[java-interview](#) / jdbc.md

neokin Update jdbc.md ...

 History 3 contributors 285 lines (193 sloc) | 24.5 KB

...

[Вопросы для собеседования](#)

## JDBC

- [Что такое JDBC?](#)
- [В чем заключаются преимущества использования JDBC?](#)
- [Что из себя представляет JDBC URL?](#)
- [Из каких частей стоит JDBC?](#)
- [Перечислите основные классы и интерфейсы jdbc](#)
- [Опишите основные этапы работы с базой данных с использованием JDBC.](#)
- [Перечислите основные типы данных используемые в JDBC. Как они связаны с типами Java?](#)
- [Как зарегистрировать драйвер JDBC?](#)
- [Как установить соединение с базой данных?](#)
- [Какие уровни изоляции транзакций поддерживаются в JDBC?](#)
- [При помощи чего формируются запросы к базе данных?](#)
- [Чем отличается Statement от PreparedStatement?](#)
- [Как осуществляется запрос к базе данных и обработка результатов?](#)

- [Как вызвать хранимую процедуру?](#)
- [Как закрыть соединение с базой данных?](#)

## Что такое JDBC?

**JDBC, Java DataBase Connectivity** (соединение с базами данных на Java) — промышленный стандарт взаимодействия Java-приложений с различными СУБД. Реализован в виде пакета `java.sql`, входящего в состав Java SE.

JDBC основан на концепции драйверов, которые позволяют получать соединение с базой данных по специально описанному URL. При загрузке драйвер регистрирует себя в системе и в дальнейшем автоматически вызывается, когда программа требует URL, содержащий протокол, за который этот драйвер отвечает.

[к оглавлению](#)

## В чем заключаются преимущества использования JDBC?

Преимуществами JDBC считают:

- Лёгкость разработки: разработчик может не знать специфики базы данных, с которой работает;
- Код практически не меняется, если компания переходит на другую базу данных (количество изменений зависит исключительно от различий между диалектами SQL);
- Не нужно дополнительно устанавливать клиентскую программу;
- К любой базе данных можно подсоединиться через легко описываемый URL.

[к оглавлению](#)

## Что из себя представляет JDBC URL?

JDBC URL состоит из:

- `<protocol>`: (протокола) - всегда `jdbc:`.
- `<subprotocol>`: (подпротокола) - это имя драйвера или имя механизма соединения с базой данных. Подпротокол может поддерживаться одним или несколькими драйверами. Лежащий на поверхности пример подпротокола -

это "odbc", отведенный для URL, обозначающих имя источника данных ODBC. В случае необходимости использовать сервис имен (т.е. имя базы данных в JDBC URL не будет действительным именем базы данных), то подпротоколом может выступать сервис имен.

- `<subname>` (подимени) - это идентификатор базы данных. Значение подимени может меняться в зависимости от подпротокола, и может также иметь под-подимен с синтаксисом, определяемым разработчиком драйвера. Назначение подимени - это предоставление всей информации, необходимой для поиска базы данных. Например, если база данных находится в Интернет, то в состав подимени JDBC URL должен быть включен сетевой адрес, подчиняющийся следующим соглашениям: `//<hostname>:<port>/<subsubname>`.

Пример JDBC URL для подключения к MySQL базе данных «Test» расположенной по адресу localhost и ожидающей соединений по порту 3306:

```
jdbc:mysql://localhost:3306/Test
```

[к оглавлению](#)

## Из каких частей стоит JDBC?

JDBC состоит из двух частей:

- **JDBC API**, который содержит набор классов и интерфейсов, определяющих доступ к базам данных. Эти классы и методы объявлены в двух пакетах - `java.sql` и `javax.sql`;
- **JDBC-драйвер**, компонент, специфичный для каждой базы данных.

JDBC превращает вызовы уровня API в «родные» команды того или иного сервера баз данных.

[к оглавлению](#)

## Перечислите основные классы и интерфейсы JDBC.

- `java.sql.DriverManager` - позволяет загрузить и зарегистрировать необходимый JDBC-драйвер, а затем получить соединение с базой данных.
- `javax.sql.DataSource` - решает те же задачи, что и `DriverManager`, но более удобным и универсальным образом. Существуют также `javax.sql.ConnectionPoolDataSource` и `javax.sql.XADataSource` задача которых - обеспечение поддержки пула соединений.

- `java.sql.Connection` - обеспечивает формирование запросов к источнику данных и управление транзакциями. Также предусмотрены интерфейсы `javax.sql.PooledConnection` и `javax.sql.XAConnection`.
- `java.sql.Statement`, `java.sql.PreparedStatement` и `java.sql.CallableStatement` - эти интерфейсы позволяют отправить запрос к источнику данных.
- `java.sql.ResultSet` - объявляет методы, которые позволяют перемещаться по набору данных и считывать значения отдельных полей в текущей записи.
- `java.sql.ResultSetMetaData` - позволяет получить информацию о структуре набора данных.
- `java.sql.DatabaseMetaData` - позволяет получить информацию о структуре источника данных.

[к оглавлению](#)

## Перечислите основные типы данных используемые в JDBC. Как они связаны с типами Java?

JDBC Type	Java Object Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	Boolean
TINYINT	Integer
SMALLINT	Integer
INTEGER	Integer
BIGINT	Long
REAL	Float
FLOAT	Double

JDBC Type	Java Object Type
DOUBLE	Double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
CLOB	Clob
BLOB	Blob
ARRAY	Array
STRUCT	Struct
REF	Ref
DISTINCT	сопоставление базового типа
JAVA_OBJECT	базовый класс Java

[к оглавлению](#)

## Опишите основные этапы работы с базой данных при использовании JDBC.

- Регистрация драйверов;
- Установление соединения с базой данных;
- Создание запроса(ов) к базе данных;
- Выполнение запроса(ов) к базе данных;
- Обработка результата(ов);
- Закрытие соединения с базой данных.

[к оглавлению](#)

## Как зарегистрировать драйвер JDBC?

Регистрацию драйвера можно осуществить несколькими способами:

- `java.sql.DriverManager.registerDriver(%объект класса драйвера%)` .
- `Class.forName(«полное имя класса драйвера»).newInstance()` .
- `Class.forName(«полное имя класса драйвера») ;`

[к оглавлению](#)

## Как установить соединение с базой данных?

Для установки соединения с базой данных используется статический вызов

```
java.sql.DriverManager.getConnection(...)
```

В качестве параметра может передаваться:

- URL базы данных

```
static Connection getConnection(String url)
```

- URL базы данных и набор свойств для инициализации

```
static Connection getConnection(String url, Properties info)
```

- URL базы данных, имя пользователя и пароль

```
static Connection getConnection(String url, String user, String password)
```

В результате вызова будет установлено соединение с базой данных и создан объект класса `java.sql.Connection` - своеобразная «сессия», внутри контекста которой и будет происходить дальнейшая работа с базой данных.

[к оглавлению](#)

## Какие уровни изоляции транзакций поддерживаются в JDBC?

**Уровень изолированности транзакций** — значение, определяющее уровень, при котором в транзакции допускаются несогласованные данные, то есть степень изолированности одной транзакции от другой. Более высокий уровень изолированности повышает точность данных, но при этом может снижаться количество параллельно выполняемых транзакций. С другой стороны, более низкий уровень изолированности позволяет выполнять больше параллельных транзакций, но снижает точность данных.

Во время использования транзакций, для обеспечения целостности данных, СУБД использует блокировки, чтобы заблокировать доступ других обращений к данным, участвующим в транзакции. Такие блокировки необходимы, чтобы предотвратить:

- *«грязное» чтение (dirty read)* — чтение данных, добавленных или изменённых транзакцией, которая впоследствии не подтвердится (откатится);
- *неповторяющееся чтение (non-repeatable read)* — при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными;
- *фантомное чтение (phantom reads)* — ситуация, когда при повторном чтении в рамках одной транзакции одна и та же выборка даёт разные множества строк.

Уровни изоляции транзакций определены в виде констант интерфейса

`java.sql.Connection` :

- `TRANSACTION_NONE` — драйвер не поддерживает транзакции;
- `TRANSACTION_READ_UNCOMMITTED` — позволяет транзакциям видеть несохраненные изменения данных: разрешает грязное, не проверяющееся и фантомное чтения;
- `TRANSACTION_READ_COMMITTED` — любое изменение, сделанное в транзакции, не видно вне неё, пока она не сохранена: предотвращает грязное чтение, но разрешает не проверяющееся и фантомное;
- `TRANSACTION_REPEATABLE_READ` — запрещает грязное и не проверяющееся, фантомное чтение разрешено;
- `TRANSACTION_SERIALIZABLE` — грязное, не проверяющееся и фантомное чтения запрещены.

**NB!** Сервер базы данных может не поддерживать все уровни изоляции. Интерфейс `java.sql.DatabaseMetaData` предоставляет информацию об уровнях изолированности транзакций, которые поддерживаются данной СУБД.

Уровень изоляции транзакции используемый СУБД можно задать с помощью метода `setTransactionIsolation()` объекта `java.sql.Connection`. Получить информацию о применяемом уровне изоляции поможет метод `getTransactionIsolation()`.

[к оглавлению](#)

## При помощи чего формируются запросы к базе данных?

Для выполнения запросов к базе данных в Java используются три интерфейса:

- `java.sql.Statement` - для операторов SQL без параметров;
- `java.sql.PreparedStatement` - для операторов SQL с параметрами и часто выполняемых операторов;
- `java.sql.CallableStatement` - для исполнения хранимых в базе процедур.

Объекты-носители интерфейсов создаются при помощи методов объекта `java.sql.Connection`:

- `java.sql.createStatement()` возвращает объект *Statement*;
- `java.sql.prepareStatement()` возвращает объект *PreparedStatement*;
- `java.sql.prepareCall()` возвращает объект *CallableStatement*;

[к оглавлению](#)

## Чем отличается Statement от PreparedStatement?

- **Statement** используется для простых случаев запроса без параметров.
- **PreparedStatement** предварительно компилирует запрос, который может содержать входные параметры и выполняться несколько раз с разным набором этих параметров.



Перед выполнением СУБД разбирает каждый запрос, оптимизирует его и создает «план» (query plan) его выполнения. Если один и тот же запрос выполняется несколько раз, то СУБД в состоянии кэшировать план его выполнения и не производить этапов разборки и оптимизации повторно. Благодаря этому запрос выполняется быстрее.

Суммируя: **PreparedStatement** выгодно отличается от **Statement** тем, что при повторном использовании с одним или несколькими наборами параметров позволяет получить преимущества заранее прекомпилированного и кэшированного запроса, помогая при этом избежать **SQL Injection**.

[к оглавлению](#)

## Как осуществляется запрос к базе данных и обработка результатов?

Выполнение запросов осуществляется при помощи вызова методов объекта, реализующего интерфейс `java.sql.Statement` :

- **executeQuery()** - для запросов, результатом которых является один набор значений, например запросов `SELECT` . Результатом выполнения является объект класса `java.sql.ResultSet` ;
- **executeUpdate()** - для выполнения операторов `INSERT` , `UPDATE` или `DELETE` , а также для операторов *DDL (Data Definition Language)*. Метод возвращает целое число, показывающее, сколько записей было модифицировано;
- **execute()** – исполняет SQL-команды, которые могут возвращать различные результаты. Например, может использоваться для операции `CREATE TABLE` . Возвращает `true` , если первый результат содержит *ResultSet* и `false` , если первый результат - это количество модифицированных записей или результат отсутствует. Чтобы получить первый результат необходимо вызвать метод `getResultSet()` или `getUpdateCount()` . Остальные результаты доступны через вызов `getMoreResults()` , который при необходимости может быть произведён многократно.

Объект с интерфейсом `java.sql.ResultSet` хранит в себе результат запроса к базе данных - некий набор данных, внутри которого есть курсор, указывающий на один из элементов набора данных - текущую запись.

Используя курсор можно перемещаться по набору данных при помощи метода `next()` .

**NB!** Сразу после получения набора данных его курсор находится перед первой записью и чтобы сделать её текущей необходимо вызвать метод `next()` .

Содержание полей текущей записи доступно через вызовы методов `getInt()` , `getFloat()` , `getString()` , `getDate()` и им подобных.

[к оглавлению](#)

## Как вызвать хранимую процедуру?

**Хранимые процедуры** – это именованный набор операторов SQL хранящийся на сервере. Такую процедуру можно вызвать из Java-класса с помощью вызова методов объекта реализующего интерфейс `java.sql.Statement` .

Выбор объекта зависит от характеристик хранимой процедуры:

- без параметров → `Statement`
- с входными параметрами → `PreparedStatement`
- с входными и выходными параметрами → `CallableStatement`

Если неизвестно, как была определена хранимая процедура, для получения информации о хранимой процедуре (например, имен и типов параметров) можно использовать методы `java.sql.DatabaseMetaData` позволяющие получить информацию о структуре источника данных.

Пример вызова хранимой процедуры с входными и выходными параметрами:

```
public void runStoredProcedure(final Connection connection) throws Exception {
    // описываем хранимую процедуру
    String procedure = "{ call procedureExample(?, ?, ?) }";

    // подготавливаем запрос
    CallableStatement cs = connection.prepareCall(procedure);

    // устанавливаем входные параметры
    cs.setString(1, "abcd");
    cs.setBoolean(2, true);
    cs.setInt(3, 10);
}
```

```
// описываем выходные параметры
cs.registerOutParameter(1, java.sql.Types.VARCHAR);
cs.registerOutParameter(2, java.sql.Types.INTEGER);

// запускаем выполнение хранимой процедуры
cs.execute();

// получаем результаты
String parameter1 = cs.getString(1);
int parameter2 = cs.getInt(2);

// заканчиваем работу с запросом
cs.close();
}
```

[к оглавлению](#)

## Как закрыть соединение с базой данных?

Соединение с базой данной закрывается вызовом метода `close()` у соответствующего объекта `java.sql.Connection` или посредством использования механизма `try-with-resources` при создании такого объекта, появившегося в Java 7.

**NB!** Предварительно необходимо закрыть все запросы созданные этим соединением.

[к оглавлению](#)

## Источники

- [Википедия - JDBC](#)
- [IBM developerWorks®](#)
- [Документация к пакету java.sql](#)
- [Википедия - Уровень изолированности транзакции](#)

[Вопросы для собеседования](#)