





Bộ môn Công nghệ Phần mềm  
Viện CNTT & TT  
Trường Đại học Bách Khoa Hà Nội

## LẬP TRÌNH WEB HƯỚNG JAVA

### Bài 15: Cơ bản về JDBC

Giảng viên: ThS. Trịnh Tuấn Đạt  
Bộ môn CNPM  
Email: [trinhthuandat.bk@gmail.com](mailto:trinhthuandat.bk@gmail.com) / [dattt@soict.hut.edu.vn](mailto:dattt@soict.hut.edu.vn)



## Nội dung


- 1. JDBC là gì?
- 2. Các bước sử dụng API của JDBC
- 3. DataSource & kỹ thuật Connection Pooling
- 4. Transaction
- 5. Prepared & Callable Statements

DatTT-DSE-SOICT-HUST 2



## 1. JDBC là gì?


DatTT-DSE-SOICT-HUST 3



## JDBC là gì?

- Là các API Java chuẩn tắc để truy cập CSDL quan hệ
  - Ứng dụng không cần quan tâm tới chi tiết cụ thể của CSDL
- Nằm trong Java SE (J2SE)
  - Java SE 6 có phiên bản JDBC 4


DatTT-DSE-SOICT-HUST 4



## JDBC API

- Định nghĩa một tập các Java Interfaces, được cài đặt bởi các vendor khác nhau, thành các JDBC Drivers
  - Các ứng dụng sử dụng tập các giao diện này để thực hiện các thao tác với CSDL → Tính portability
- Phần lớn API của JDBC nằm trong gói java.sql
  - DriverManager, Connection, ResultSet, DatabaseMetaData, ResultSetMetaData, PreparedStatement, CallableStatement và Types
- Một số chức năng nâng cao khác nằm trong gói javax.sql package
  - DataSource

DatTT-DSE-SOICT-HUST 5



## JDBC Driver

- Là cài đặt cụ thể của các JDBC interfaces
  - Tất cả các database server đều có JDBC driver(s) tương ứng
- Có thể xem danh sách các drivers đã có trên
  - <http://industry.java.sun.com/products/jdbc/drivers>

DatTT-DSE-SOICT-HUST 6

## Database URL

- Được sử dụng để tạo một kết nối tới database
  - Có thể chứa server, port, protocol, etc.
- Cú pháp:
  - `jdbc:subprotocol_name:driver_dependant_databasename`
- Ví dụ:
  - Oracle thin driver
    - `jdbc:oracle:thin:@machinename:1521:dbname`
  - Derby
    - `jdbc:derby://localhost:1527/sample`
  - Pointbase
    - `jdbc:pointbase:server://localhost/sample`

DatTT-DSE-SOICT-HUST

7

## 2. Các bước sử dụng JDBC API

DatTT-DSE-SOICT-HUST

8

## Các bước sử dụng JDBC

- B1. Load JDBC driver cho từng loại CSDL
- B2. Lấy đối tượng Connection
- B3. Lấy đối tượng Statement
- B4. Thực hiện câu truy vấn, câu lệnh update
- B5. Đọc kết quả trả về
- B6. Đọc các Meta-data (tùy chọn)
- B7. Đóng đối tượng Statement và đối tượng Connection

DatTT-DSE-SOICT-HUST

9

### B1. Load JDBC driver cho từng loại CSDL

- Để load về driver cho CSDL và đăng ký nó với DriverManager, cần load class tương ứng

```

■ Class.forName(<database-driver>)
try {
    // This loads an instance of the Pointbase DB Driver.
    // The driver has to be in the classpath.
    Class.forName("org.apache.derby.jdbc.ClientDriver");
} catch (ClassNotFoundException cnfe) {
    System.out.println(" " + cnfe);
}

```

DatTT-DSE-SOICT-HUST

10

## B2. Lấy ra đối tượng Connection

- Lớp `DriverManager` chịu trách nhiệm tạo kết nối tới CSDL
  - Sử dụng `DataSource` là cách hay dùng hơn khi muốn lấy ra một đối tượng connection (trình bày ở phần sau)
- Ví dụ tạo kết nối tới CSDL như sau:

```

try {
    Connection connection = DriverManager.
        getConnection("jdbc:derby://localhost:1527/sample",
            "app", "app");
} catch (SQLException sqle) {
    System.out.println(" " + sqle);
}

```

DatTT-DSE-SOICT-HUST

11

## DriverManager & Connection

- `java.sql.DriverManager`
  - `getConnection(String url, String user, String password)` throws `SQLException`
- `java.sql.Connection`
  - `Statement createStatement()` throws `SQLException`
  - `void close()` throws `SQLException`
  - `void setAutoCommit(boolean b)` throws `SQLException`
  - `void commit()` throws `SQLException`
  - `void rollback()` throws `SQLException`

DatTT-DSE-SOICT-HUST

12

### B3. Lấy ra đối tượng Statement

- Tạo một đối tượng **Statement** từ đối tượng **Connection**
  - `java.sql.Statement`
    - `ResultSet executeQuery(string sql)`
    - `int executeUpdate(String sql)`
  - Ví dụ:
    - `Statement statement = connection.createStatement();`
- Cùng đối tượng **Statement** có thể được dùng cho nhiều queries không liên quan tới nhau

DatTT-DSE-SOICT-HUST

13

### B4. Thực thi các câu truy vấn/các lệnh

- Từ đối tượng **Statement**, 2 lệnh được sử dụng nhiều nhất là
  - (a) **QUERY (SELECT)**
    - `ResultSet rs = statement.executeQuery("select * from customer_tbl");`
  - (b) **ACTION COMMAND (UPDATE/DELETE)**
    - `int iReturnValue = statement.executeUpdate("update manufacture_tbl set name = 'IBM' where mfr_num = 19985678");`

DatTT-DSE-SOICT-HUST

14

### B5. Xử lý kết quả nhận về

- Duyệt trên **ResultSet** để xử lý thông tin
  - `java.sql.ResultSet`
    - `boolean next()`
    - `xxx getXxx(int columnNumber)`
    - `xxx getXxx(String columnName)`
    - `void close()`
- Đầu tiên, con trỏ lập nằm ở trước hàng đầu tiên
  - LTV cần gọi phương thức `next()` để chuyển con trỏ đến hàng đầu tiên

DatTT-DSE-SOICT-HUST

15

### B5. Xử lý kết quả nhận về (2)

- Khi đã có **ResultSet**, LTV dễ dàng xử lý dữ liệu
    - Lưu ý: Chỉ số của **ResultSet** bắt đầu từ 1
- ```

while (rs.next()){
    // Wrong this will generate an error
    String value0 = rs.getString(0);

    // Correct!
    String value1 = rs.getString(1);
    int value2 = rs.getInt(2);
    int value3 = rs.getInt("ADDR_LN1");
}

```

DatTT-DSE-SOICT-HUST

16

### B5. Xử lý kết quả nhận về (3)

- Muốn lấy dữ liệu từ **ResultSet**, sử dụng phương thức `getXXX()` cho phù hợp
  - `getString()`
  - `getInt()`
  - `getDouble()`
  - `getObject()`
- Mỗi kiểu dữ liệu trong `java.sql.Types`, đều có phương thức `getXXX` tương ứng

DatTT-DSE-SOICT-HUST

17

### B6. Đọc metadata của ResultSet và metadata của CSDL (tùy chọn)

- Khi đã có đối tượng **ResultSet** hoặc **Connection**, LTV có thể lấy về metadata của CSDL hoặc của câu truy vấn
- → Đem lại thông tin hữu ích về dữ liệu lấy về, hoặc về CSDL đang sử dụng
  - `ResultSetMetaData rsMeta = rs.getMetaData();`
  - `DatabaseMetaData dbmetadata = connection.getMetaData();`
    - Có khoảng 150 phương thức trong lớp `DatabaseMetaData`

DatTT-DSE-SOICT-HUST

18

## Ví dụ về ResultSetMetaData

```
ResultSetMetaData meta = rs.getMetaData();
//Return the column count
int iColumnCount = meta.getColumnCount();

for (int i = 1 ; i <= iColumnCount ; i++){
    System.out.println("Column Name: " + meta.getColumnName(i));
    System.out.println("Column Type" + meta.getColumnType(i));
    System.out.println("Display Size: " + meta.getColumnDisplaySize(i) );
    System.out.println("Precision: " + meta.getPrecision(i));
    System.out.println("Scale: " + meta.getScale(i) );
}
```

DatTT-DSE-SOICT-HUST

19

## 3. DataSource & kỹ thuật Connection Pooling

DatTT-DSE-SOICT-HUST

20

## 3. DataSource & kỹ thuật Connection Pooling

- 3.1. Giao diện DataSource và đối tượng DataSource
- 3.2. Các thuộc tính của đối tượng DataSource
- 3.3. Đăng ký JNDI của đối tượng DataSource
- 3.4. Connection Pooling
- 3.5. Lấy về đối tượng DataSource

DatTT-DSE-SOICT-HUST

21

### 3.1. Giao diện javax.sql.DataSource và đối tượng DataSource

- Từng nhà cung cấp Driver sẽ thực thi cài đặt interface
- Đối tượng DataSource dùng để tạo các kết nối CSDL (**database connections**)

DatTT-DSE-SOICT-HUST

22

### Giao diện javax.sql.DataSource và đối tượng DataSource

- Có 3 kiểu cài đặt interface
  - **Basic implementation**: cung cấp đối tượng Connection chuẩn tắc
  - **Connection pooling implementation**: cung cấp đối tượng Connection tự động nằm trong **connection pooling**
  - **Distributed transaction implementation**: cung cấp đối tượng Connection có thể dùng được cho các giao dịch (transactions) phân tán, hầu hết nằm trong connection pooling

DatTT-DSE-SOICT-HUST

23

### 3.2. Các thuộc tính của đối tượng DataSource

- Một đối tượng DataSource có các thuộc tính có thể sửa đổi khi cần thiết-được định nghĩa trong file cấu hình của container
  - Địa chỉ của database server
  - Tên của database
  - Network protocol được sử dụng để giao tiếp với server
- Lợi ích: vì thay đổi được thuộc tính của DataSource, các đoạn code truy cập tới DataSource đó không cần thay đổi
- Trong **Sun Java System Application Server** (và **GlassFish V2**), một data source được gọi là một **JDBC resource**

DatTT-DSE-SOICT-HUST

24

### Các thuộc tính của một DataSource được định nghĩa ở đâu?

- Trong file cấu hình của container
- Trong **Sun Java System App Server**:
  - <J2EE\_HOME>/domains/domain1/config/domain.xml
- Trong **Tomcat**:
  - <TOMCAT\_HOME>/conf/server.xml

DatTT-DSE-SOICT-HUST

25

### DataSource (JDBC Resource) trong file domain.xml của Sun Java System App Server

```
<resources>
  <jdbc-resource enabled="true" jndi-name="jdbc/BookDB"
    object-type="user" pool-name="PointBasePool"/>
  <jdbc-connection-pool connection-validation-method="auto-
    commit" datasource-classname="com.pointbase.xa.xaDataSource"
    fail-all-connections="false" idle-timeout-in-seconds="300"
    is-connection-validation-required="false" is-isolation-
    level-guaranteed="true" max-pool-size="32" max-wait-time-in-
    millis="60000" name="PointBasePool" pool-resize-quantity="2"
    res-type="javax.sql.XADataSource" steady-pool-size="8">
    <property name="DatabaseName"
      value="jdbc:pointbase:server://localhost:9092/sun-appserv-
      samples"/>
    <property name="Password" value="pbPublic"/>
    <property name="User" value="pbPublic"/>
  </jdbc-connection-pool>
</resources>
```

DatTT-DSE-SOICT-HUST

26

### Định nghĩa DataSource trong Sun Java System App Server



DatTT-DSE-SOICT-HUST

27

### Định nghĩa DataSource trong Sun Java System App Server (2)



DatTT-DSE-SOICT-HUST

28

### 3.3. Đăng ký JNDI của đối tượng DataSource

- Một driver được truy cập từ một đối tượng DataSource sẽ không đăng ký nó với DriverManager
- Thực tế, một đối tượng DataSource được đăng ký với **JNDI naming service** nhờ container và trả về cho client qua thao tác tra cứu (lookup operation)
- Với kiểu **basic implementation**, kết nối lấy được từ đối tượng DataSource là giống với kết nối lấy được từ DriverManager

DatTT-DSE-SOICT-HUST

29

### 3.3. Đăng ký JNDI của đối tượng DataSource (JDBC Resource)

- Định danh JNDI của một JDBC resource có dạng **java:comp/env/jdbc**
  - Ví dụ: JNDI name cho BookDB database **java:comp/env/jdbc/BookDB**
- Vì tất cả các định danh JNDI của các resource đều có dạng java:comp/env, nên khi cần chỉ ra định danh JNDI của một JDBC resource, chỉ cần nhập vào **jdbc/name**.
  - Ví dụ: **jdbc/BookDB**

DatTT-DSE-SOICT-HUST

30

### 3.4. Tại sao cần Connection Pooling?

- Kết nối CSDL là tài nguyên có chi phí cao, và bị hạn chế
  - Sử dụng connection pooling, chỉ cần 1 số nhỏ các connections có thể được dùng chung cho 1 số lớn clients
- Tạo và hủy các kết nối CSDL là thao tác có chi phí cao
  - Sử dụng connection pooling, có sẵn một tập các kết nối sẵn dùng, hạn chế tạo mới/hủy các connection

DatTT-DSE-SOICT-HUST

31

### 3.5. Lấy và sử dụng đối tượng DataSource

- Ứng dụng thực hiện thao tác tra cứu JNDI (**JNDI lookup**) để lấy về đối tượng DataSource
- Đối tượng DataSource sau đó dùng để lấy ra đối tượng Connection
- Trong file web.xml của ứng dụng, cần chỉ ra thông tin về **external resource**, & về đối tượng DataSource
- Trong **Sun Java System App server**, cần ánh xạ giữa **external resource** với **JNDI name**
  - Tăng tính flexibility

DatTT-DSE-SOICT-HUST

32

### Ví dụ: Lấy đối tượng DataSource qua JNDI

- BookDBAO.java trong ứng dụng bookstore1 ([http://java.sun.com/j2ee/tutorial/1\\_3-fcs/doc/Servlets3.html](http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets3.html))

```
public class BookDBAO {
    private ArrayList books;
    Connection con;
    private boolean conFree = true;

    public BookDBAO() throws Exception {
        try {
            Context initCtx = new InitialContext();
            Context envCtx = (Context) initCtx.lookup("java:comp/env");
            DataSource ds = (DataSource) envCtx.lookup("jdbc/BookDB");
            con = ds.getConnection();
        } catch (Exception ex) {
            throw new Exception("Couldn't open connection to database: " +
                ex.getMessage());
        }
    }
}
```

DatTT-DSE-SOICT-HUST

33

### Thông tin về JNDI Resource trong file web.xml của ứng dụng bookstore1

```
<resource-ref>
  <res-ref-name>jdbc/BookDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

DatTT-DSE-SOICT-HUST

34

### Mapping giữa JNDI và Resource trong file sun-web.xml

```
<sun-web-app>
  <context-root>/bookstore1</context-root>
  <resource-ref>
    <res-ref-name>jdbc/BookDB</res-ref-name>
    <jndi-name>jdbc/BookDB</jndi-name>
  </resource-ref>
</sun-web-app>
```

DatTT-DSE-SOICT-HUST

35

## 4. Transaction

DatTT-DSE-SOICT-HUST

36

## Transaction

- Việc commit từng câu lệnh một ngay khi được yêu cầu tiêu tốn nhiều thời gian
- Khi thiết lập AutoCommit là false, LTV có thể cập nhật CSDL nhiều lần, sau đó commit toàn bộ trong một transaction
- Ngoài ra, nếu các lệnh phụ thuộc lẫn nhau, toàn bộ transaction có thể được **rolled back**

DatTT-DSE-SOICT-HUST

37

## Các phương thức trong JDBC Transaction

- setAutoCommit()**
  - Nếu thiết lập là true, tất cả các lệnh thực thi (executed statement) sẽ được commit ngay lập tức
- commit()**
  - Chỉ hợp lệ khi đã thiết lập setAutoCommit(false)
  - Commit tất cả các thao tác được thực hiện, từ lúc mở một Connection hoặc từ lời gọi commit() hoặc rollback() gần nhất
- rollback()**
  - Chỉ hợp lệ khi đã thiết lập setAutoCommit(false)
  - Hủy tất cả các thao tác vừa thực hiện

DatTT-DSE-SOICT-HUST

38

## Ví dụ về Transactions

```
Connection connection = null;
try {
    connection =
        DriverManager.getConnection("jdbc:oracle:thin:@machinename:1521:db
name","username","password");
    connection.setAutoCommit(false);

    PreparedStatement updateQty =
        connection.prepareStatement("UPDATE STORE_SALES SET QTY = ?
WHERE ITEM_CODE = ?");
```

DatTT-DSE-SOICT-HUST

39

## Ví dụ về Transactions (2)

```
int [][] arrValueToUpdate =
{ {123, 500} ,
  {124, 250},
  {125, 10},
  {126, 350} };

int iRecordsUpdate = 0;
for ( int items=0 ; items < arrValueToUpdate.length ; items++) {
    int itemCode = arrValueToUpdate[items][0];
    int qty = arrValueToUpdate[items][1];
```

DatTT-DSE-SOICT-HUST

40

## Ví dụ về Transactions (3)

```
updateQty.setInt(1,qty);
updateQty.setInt(2,itemCode);
iRecordsUpdate += updateQty.executeUpdate();
}
connection.commit();
System.out.println(iRecordsUpdate +
    " record(s) have been updated");
} catch(SQLException sqle) {
    System.out.println(sqle);
```

DatTT-DSE-SOICT-HUST

41

## Ví dụ về Transactions (4)

```
try {
    connection.rollback();
} catch(SQLException sqleRollback) {
    System.out.println(sqleRollback);
}
} finally {
    try {
        connection.close();
    }
    catch(SQLException sqleClose) {
        System.out.println(sqleClose);
    }
}
```

DatTT-DSE-SOICT-HUST

42

## 5. Prepared & Callable Statements

DatTT-DSE-SOICT-HUST

43

### Định nghĩa?

- PreparedStatement
  - Câu lệnh SQL được gửi đến CSDL, được biên dịch hoặc được chuẩn bị trước
- CallableStatement
  - Thực thi các **SQL Stored Procedures**

DatTT-DSE-SOICT-HUST

44

### PreparedStatement

- Đôi khi, nhiều câu lệnh có cấu trúc tương tự nhau, chỉ có giá trị là thay đổi
- PreparedStatement có thể được sử dụng để soạn trước câu lệnh có cấu trúc cần thiết
- PreparedStatement có thể nhận các tham số VÀO, hoạt động tương tự các đối số cho một phương thức
- PreparedStatement deal with data conversions that can be error prone in straight ahead, built on the fly SQL
  - handling quotes and dates in a manner transparent to the developer

DatTT-DSE-SOICT-HUST

45

### Các bước làm việc với PreparedStatement

- B1: Tạo DB connection như bình thường
- B2: Tạo đối tượng prepared statement từ connection

```
PreparedStatement updateSales = con.prepareStatement("UPDATE
OFFER_TBL SET QUANTITY = ? WHERE ORDER_NUM = ? ");
// "?" are referred to as Parameter Markers
// Parameter Markers are referred to by number,
// starting from 1, in left to right order.
// PreparedStatement's setXXX() methods are used to set
// the IN parameters, which remain set until changed.
```

DatTT-DSE-SOICT-HUST

46

### Các bước làm việc với PreparedStatement (2)

- B3. Truyền vào các đối số theo đúng vị trí
 

```
updateSales.setInt(1, 75);
updateSales.setInt(2, 10398001);
```
- B4. Thực thi prepared statement
 

```
int iUpdatedRecords =
    updateSales.executeUpdate();
```

DatTT-DSE-SOICT-HUST

47

### Các bước làm việc với PreparedStatement (3)

- Nếu thuộc tính AutoCommit là true, khi thực thi câu lệnh, thay đổi sẽ được commit. Từ đó về sau, có thể sử dụng lại đối tượng PreparedStatement này.

```
updateSales.setInt(1, 150);
updateSales.setInt(2, 10398002);
```

DatTT-DSE-SOICT-HUST

48



## PreparedStatement

- Nếu đối tượng prepared statement có kiểu là câu lệnh select, sau khi thực thi và lấy về kết quả, tiến hành duyệt trên ResultSet như phần trước

```
PreparedStatement itemsSold =
    con.prepareStatement("select o.order_num,
        o.customer_num, c.name, o.quantity from order_tbl
        o, customer_tbl c where o.customer_num =
        c.customer_num and o.customer_num = ?;");
itemsSold.setInt(1, 10398001);
ResultSet rsItemsSold = itemsSold.executeQuery();
while (rsItemsSold.next()){
    System.out.println( rsItemsSold.getString("NAME")
        + " sold " + rsItemsSold.getString("QUANTITY") + "
        unit(s)");
}
```

DatTT-DSE-SOICT-HUST

49

## CallableStatement

- Là giao diện được sử dụng để thực thi các **SQL stored procedures**
- Một **stored procedure** là một nhóm các câu lệnh SQL thực hiện một công việc nào đó
- Stored procedures** được sử dụng để đóng gói một tập các thao tác hoặc truy vấn trên một database server.

DatTT-DSE-SOICT-HUST

50

## CallableStatement (2)

- Một đối tượng CallableStatement chứa lời gọi đến một stored procedure; nó không chứa chính stored procedure này.
- Ví dụ: gọi đến stored procedure SHOW\_SUPPLIERS sử dụng connection con và trả về ResultSet

```
CallableStatement cs = con.prepareCall("call
    SHOW_SUPPLIERS");
ResultSet rs = cs.executeQuery();
```

DatTT-DSE-SOICT-HUST

51

## Ví dụ CallableStatement

- Ví dụ sử dụng các tham số IN, OUT và INOUT

```
// set int IN parameter
cstmt.setInt( 1, 333 );
// register int OUT parameter
cstmt.registerOutParameter( 2, Types.INTEGER );
// set int INOUT parameter
cstmt.setInt( 3, 666 );
// register int INOUT parameter
cstmt.registerOutParameter( 3, Types.INTEGER );
//You then execute the statement with no return value
cstmt.execute(); // could use executeUpdate()
// get int OUT and INOUT
int iOUT = cstmt.getInt( 2 );
int iINOUT = cstmt.getInt( 3 );
```

DatTT-DSE-SOICT-HUST

52

## Ví dụ Stored Procedure

```
FUNCTION event_list (appl_id_in  VARCHAR2,
                    dow_in       VARCHAR2,
                    event_type_in VARCHAR2 OUT,
                    status_in     VARCHAR2 INOUT)
RETURN ref_cur;
```

DatTT-DSE-SOICT-HUST

53

## Ví dụ với Oracle DB

```
try {
    Connection connection = DriverManager.getConnection("");
    CallableStatement queryreport = connection.prepareCall("{
        ? = call SRO21208_PKG.QUEUE_REPORT ( ? , ? , ? , ? , ?
        , ? ) }");

    queryreport.registerOutParameter(1, OracleTypes.CURSOR);
    queryreport.setInt(2, 10);
    queryreport.setString(3, "000004357");
    queryreport.setString(4, "01/07/2003");
    queryreport.setString(5, "N");
    queryreport.setString(6, "N");
    queryreport.setString(7, "N");
    queryreport.setInt(8, 2);
}
```

DatTT-DSE-SOICT-HUST

54



## Ví dụ với Oracle DB

```
queryreport.execute();
ResultSet resultset = (ResultSet)queryreport.getObject(1);

while (resultset.next())
{
    System.out.println(" " + resultset.getString(1) + " "
        + resultset.getString(2));
}

catch( SQLException sqle){
    System.out.println(" " + sqle);
}
```