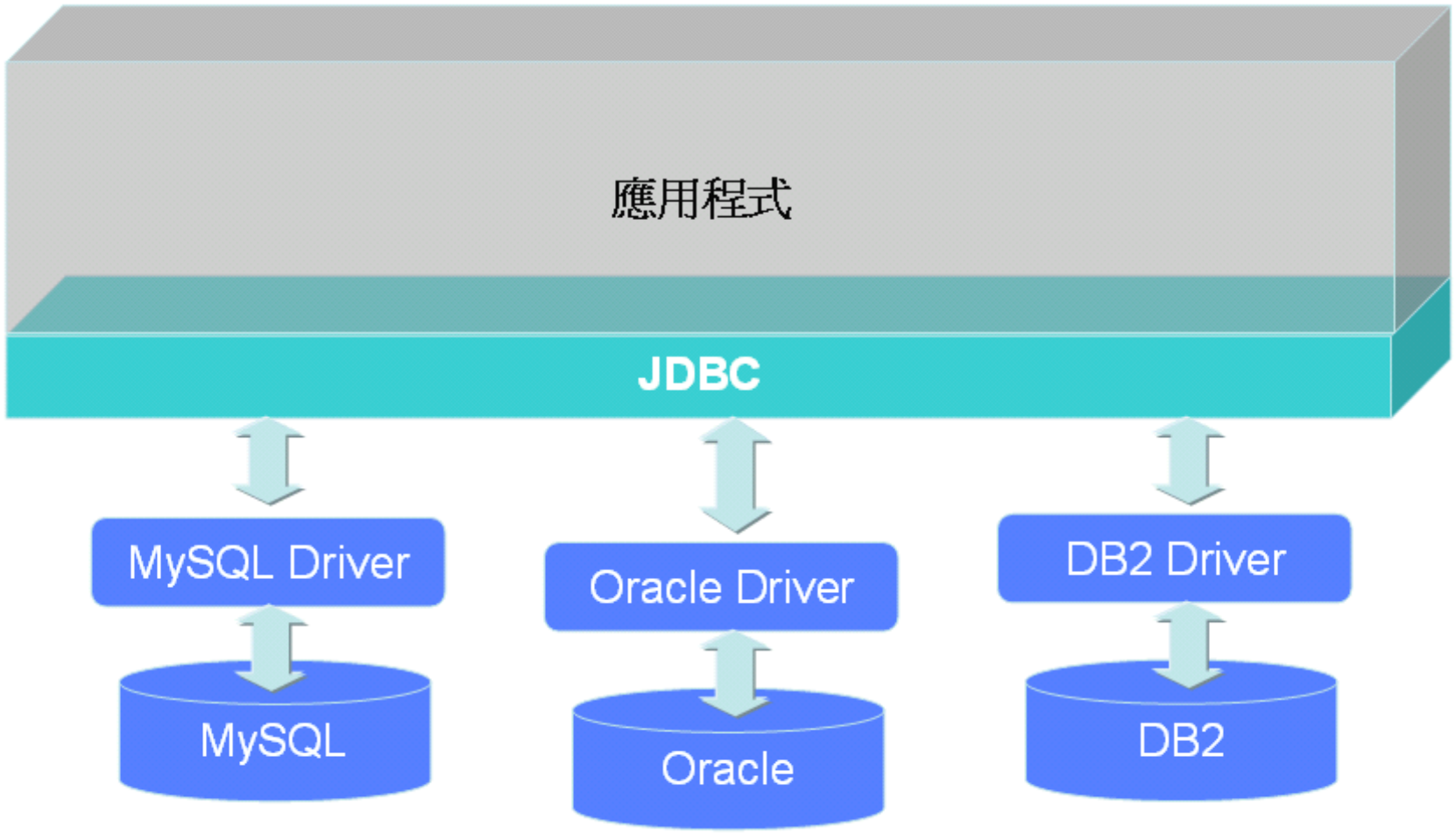
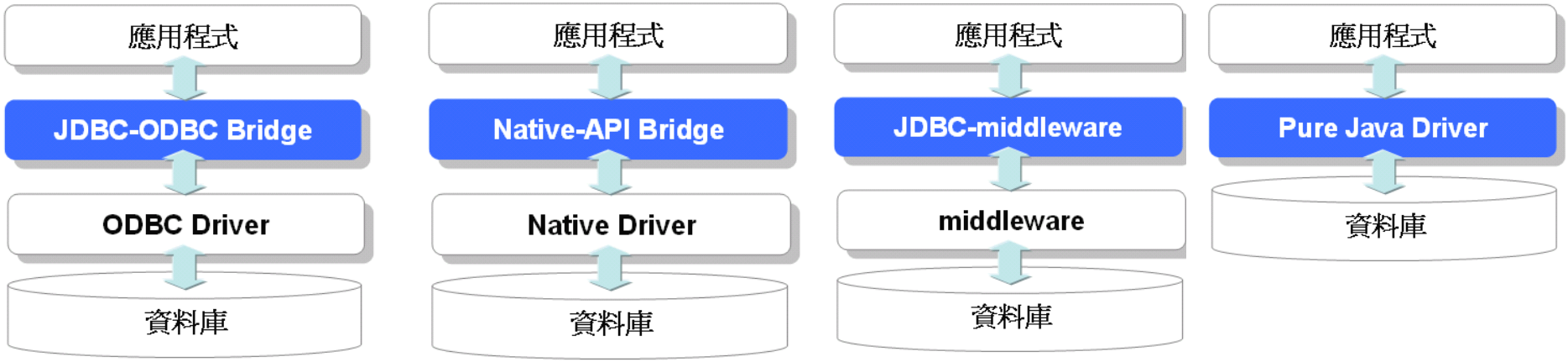
Java 課程

第14章：JDBC入門

簡介JDBC


應用程式
JDBC
MySQL Driver
Oracle Driver
DB2 Driver
MySQL
Oracle
DB2

- JDBC資料庫驅動程式依實作方式可以分為四個類型
  - Type 1：JDBC-ODBC Bridge
  - Type 2：Native-API Bridge
  - Type 3：JDBC-middleware
  - Type 4：Pure Java Driver



連接資料庫

- 載入JDBC驅動程式

```
try {
    Class.forName("com.mysql.jdbc.Driver");
}
catch(ClassNotFoundException e) {
    System.out.println("找不到驅動程式類別");
}
```

連接資料庫

- 提供JDBC URL
  - 協定:子協定:資料來源識別

```
jdbc:mysql://主機名稱:連接埠/資料庫名稱?參數=值&參數=值

  jdbc:mysql://localhost:3306/demo?user=root&password=123

  jdbc:mysql://localhost:3306/demo?user=root&password=123&
  useUnicode=true&characterEncoding=Big5
```

取得Connection

```
try {
    String url =  "jdbc:mysql://localhost:3306/demo?" +
                  "user=root&password=123";
    Connection conn = DriverManager.getConnection(url);
    ....
}
catch(SQLException e) {
    ....
}
```

```
String url = "jdbc:mysql://localhost:3306/demo";
String user = "root";
String password = "123";
Connection conn = DriverManager.getConnection(url,
                                              user, password);
```

簡單的Connection工具類別

- 取得Connection的方式，依所使用的環境及程式需求而有所不同
- 設計一個DBSource介面

```
package onlyfun.caterpillar;
import java.sql.Connection;
import java.sql.SQLException;
public interface DBSource {
    public Connection getConnection() throws SQLException;
    public void closeConnection(Connection conn) throws SQLException;
}
```

```
public class SimpleDBSource implements DBSource {
    …

    public SimpleDBSource(String configFile) throws IOException,
                                                    ClassNotFoundException {
        props = new Properties();
        props.load(new FileInputStream(configFile));

        url = props.getProperty("onlyfun.caterpillar.url");
        user = props.getProperty("onlyfun.caterpillar.user");
        passwd = props.getProperty("onlyfun.caterpillar.password");

        Class.forName(
                    props.getProperty("onlyfun.caterpillar.driver"));
    }
    public Connection getConnection() throws SQLException {
        return DriverManager.getConnection(url, user, passwd);
    }
    public void closeConnection(Connection conn) throws SQLException {
        conn.close();
    }
}
```

```
onlyfun.caterpillar.driver=com.mysql.jdbc.Driver
onlyfun.caterpillar.url=jdbc:mysql://localhost:3306/demo
onlyfun.caterpillar.user=root
onlyfun.caterpillar.password=123456
```

```
DBSource dbsource = new SimpleDBSource();
Connection conn = dbsource.getConnection();

if(!conn.isClosed()) {
    System.out.println("資料庫連接已開啟…");
}

dbsource.closeConnection(conn);

if(conn.isClosed()) {
    System.out.println("資料庫連接已關閉…");
}
```

簡單的連接池（Connection pool）

- 資料庫連接的取得是一個耗費時間與資源的動作
  - 建立Socket connection
  - 交換資料（使用者密碼驗證、相關參數）
  - 資料庫初始會話（Session）
  - 日誌（Logging）
  - 分配行程（Process）
  - …

簡單的連接池（Connection pool）

```
public synchronized Connection getConnection()
                                                    throws SQLException {
    if(connections.size() == 0) {
        return DriverManager.getConnection(url, user, passwd);
    }
    else {
        int lastIndex = connections.size() - 1;
        return connections.remove(lastIndex);
    }
}

public synchronized void closeConnection(Connection conn)
                                                      throws SQLException {
    if(connections.size() == max) {
        conn.close();
    }
    else {
        connections.add(conn);
    }
}
```

簡單的連接池（Connection pool）

```
DBSource dbsource = new BasicDBSource("jdbc2.properties");
Connection conn1 = dbsource.getConnection();
dbsource.closeConnection(conn1);
Connection conn2 = dbsource.getConnection();
System.out.println(conn1 == conn2);
```

```
onlyfun.caterpillar.driver=com.mysql.jdbc.Driver
onlyfun.caterpillar.url=jdbc:mysql://localhost:3306/demo
onlyfun.caterpillar.user=root
onlyfun.caterpillar.password=123456
onlyfun.caterpillar.poolmax=10
```

簡單的連接池（Connection pool）

- 初始的Connection數量
- Connection最大idle的數量
- 如果超過多久時間，要回收多少數量的Connection
- Proxool
  - http://proxool.sourceforge.net/index.html
- Apache Jakarta的Common DBCP
  - http://jakarta.apache.org/commons/dbcp

Statement、ResultSet

- 要執行SQL的話，必須取得java.sql.Statement物件，它是Java當中一個SQL敘述的具體代表物件

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

- 插入一筆資料，可以如下使用Statement的executeUpdate()方法

```
stmt.executeUpdate("INSERT INTO t_message VALUES(1, 'justin', " +
        "'justin@mail.com', 'mesage...')");
```

Statement、ResultSet

- executeUpdate()會傳回int結果，表示資料變動的筆數
- executeQuery()方法則是用於SELECT等查詢資料庫的SQL
- executeQuery()會傳回java.sql.ResultSet物件，代表查詢的結果
- 可以使用ResultSet的next()來移動至下一筆資料，它會傳回 true 或 false表示是否有下一筆資料
- 使用getXXX()來取得資料

Statement、ResultSet

- 指定欄位名稱來取得資料

```
ResultSet result =
                stmt.executeQuery("SELECT * FROM t_message");
while(result.next()) {
    System.out.print(result.getInt("id") + "\t");
    System.out.print(result.getString("name") + "\t");
    System.out.print(result.getString("email") + "\t");
    System.out.print(result.getString("msg") + "\t");
}
```

Statement、ResultSet

- 使用查詢結果的欄位順序來顯示結果

```
ResultSet result =
                stmt.executeQuery("SELECT * FROM t_message");
while(result.next()) {
    System.out.print(result.getInt(1) + "\t");
    System.out.print(result.getString(2) + "\t");
    System.out.print(result.getString(3) + "\t");
    System.out.print(result.getString(4) + "\t");
}
```

Statement、ResultSet

- Statement的execute()可以用來執行SQL，並可以測試所執行的SQL是執行查詢或是更新
- 傳回 true的話表示SQL執行將傳回ResultSet表示查詢結果，此時可以使用getResultSet()取得ResultSet物件
- 如果execute()傳回false，表示SQL執行會傳回更新筆數或沒有結果，此時可以使用getUpdateCount()取得更新筆數
- 如果事先無法得知是進行查詢或是更新，就可以使用execute()

Statement、ResultSet

```
finally {
    if(stmt != null) {
        try {
            stmt.close();
        }
        catch(SQLException e) {
            e.printStackTrace();
        }
    }
    if(conn != null) {
        try {
            dbsource.closeConnection(conn);
        }
        catch(SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Statement、ResultSet

- Connection物件預設為「自動認可」（auto commit）
- getAutoCommit()可以測試是否設定為自動認可
- 無論是否有無執行commit()方法，只要SQL沒有錯，在關閉Statement或Connection前，都會執行認可動作

PreparedStatement

- preparedStatement()方法建立好一個預先編譯（precompile）的SQL語句
- 當中參數會變動的部份，先指定"?"這個佔位字元

```
PreparedStatement stmt = conn.prepareStatement(
                    "INSERT INTO t_message VALUES(?, ?, ?, ?)");
```

PreparedStatement

- 需要真正指定參數執行時，再使用相對應的setInt()、setString()等方法，指定"?"處真正應該有的參數

```
stmt.setInt(1, 2);
stmt.setString(2, "momor");
stmt.setString(3, "momor@mail.com");
stmt.setString(4, "message2...");
```

LOB讀寫

- BLOB全名Binary Large Object，用於儲存大量的二進位資料
- CLOB全名Character Large Object，用於儲存大量的文字資料
- 在JDBC中也提供了java.sql.Blob與java.sql.Clob兩個類別分別代表BLOB與CLOB資料

LOB讀寫

- 取得一個檔案，並將之存入資料庫中

```
File file = new File("./logo_phpbb.jpg");
int length = (int) file.length();
InputStream fin = new FileInputStream(file);
 // 填入資料庫
PreparedStatement pstmt = conn.prepareStatement(
                          "INSERT INTO files
VALUES(?, ?, ?)");
pstmt.setInt(1, 1);
pstmt.setString(2, "filename");
pstmt.setBinaryStream (3, fin, length);
pstmt.executeUpdate();
pstmt.clearParameters();
pstmt.close();
fin.close();
```

LOB讀寫

- 從資料庫中取得BLOB或CLOB資料

```
Blob blob = result.getBlob(2);  // 取得BLOB
Clob clob = result.getClob(2)  // 取得CLOB
```

交易（Transaction）

- 可以操作Connection的setAutoCommit()方法，給它false引數
- 在下達一連串的SQL語句後，自行呼叫Connection的commit()來送出變更

交易（Transaction）

```
try {
    …
    conn.setAutoCommit(false); // 設定auto commit為false

    stmt = conn.createStatement();
    stmt.execute("....."); // SQL
    stmt.execute("....");
    stmt.execute("....");

    conn.commit(); // 正確無誤，確定送出
}
catch(SQLException e) { // 喔喔！在commit()前發生錯誤
    try {
        conn.rollback(); // 撤消操作
    } catch (SQLException e1) {
        e1.printStackTrace();
    }
    e.printStackTrace();
}
```

交易（Transaction）

- 設定儲存點（save point）

```
conn.setAutoCommit(false);
Statement stmt = conn.createStatement();
stmt.executeUpdate("....");
stmt.executeUpdate("....");
Savepoint savepoint = conn.setSavepoint(); // 設
定save point
stmt.executeUpdate("....");
// 如果因故rollback
conn.rollback(savepoint);
. . .
conn.commit();
// 記得釋放save point
stmt.releaseSavepoint(savepoint);
```

批次處理

- 使用addBatch()方法將要執行的SQL敘述加入，然後執行 executeBatch()

```
conn.setAutoCommit(false);
Statement stmt = conn.createStatement();
stmt.addBatch("..."); // SQL
stmt.addBatch("...");
stmt.addBatch("...");
...
stmt.executeBatch();
conn.commit();
```

批次處理

- 使用PreparedStatement可以進行批次處理

```
PreparedStatement stmt = conn.prepareStatement(
    "INSERT INTO t_message VALUES(?, ?, ?, ?)");
Message[] messages = ...;

for(int i = 0; i < messages.length; i++) {
     stmt.setInt(1, messages[i].getID());
     stmt.setString(2, messages[i].getName());
     stmt.setString(3, messages[i].getEmail());
     stmt.setString(4, messages[i].getMsg());
     stmt.addBatch();
}

stmt.executeBatch();
```

ResultSet游標控制

- 可以在建立Statement物件時指定resultSetType
  - ResultSet.TYPE_FORWARD_ONLY
  - ResultSet.TYPE_SCROLL_INSENSITIVE
  - ResultSet.TYPE_SCROLL_SENSITIVE
- 預設是第一個，也就是只能使用next()來逐筆取得資料
- 指定第二個或第三個時，則可以使用ResultSet的afterLast()、previous()、absolute()、relative()等方法

ResultSet游標控制

- 還必須指定resultSetConcurrency
  - ResultSet.CONCUR_READ_ONLY
  - ResultSet.CONCUR_UPDATABLE
- createStatement()不給定參數時，預設是TYPE_FORWARD_ONLY、CONCUR_READ_ONLY

ResultSet游標控制

```
dbsource = new SimpleDBSource();
conn = dbsource.getConnection();

stmt = conn.createStatement(
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);

ResultSet result = stmt.executeQuery(
                        "SELECT * FROM t_message");
result.afterLast();

while(result.previous()) {
    System.out.print(result.getInt("id") + "\t");
    System.out.print(result.getString("name") + "\t");
    System.out.print(result.getString("email") + "\t");
    System.out.println(result.getString("msg"));
}
```

ResultSet新增、更新、刪除資料

- 建立Statement時必須在createStatement()上指定TYPE_SCROLL_SENSITIVE（或TYPE_SCROLL_INSENSITIVE，如果不想取得更新後的資料的話）與CONCUR_UPDATABLE

```
Statement stmt = conn.createStatement(
                                ResultSet.TYPE_SCROLL_SENSITIVE,
                                 ResultSet.CONCUR_UPDATABLE);
```

ResultSet新增、更新、刪除資料

- 針對查詢到的資料進行更新的動作

```
ResultSet result = stmt.executeQuery(
            "SELECT * FROM t_message WHERE name='justin'");
result.last();
result.updateString("name", "caterpillar");
result.updateString("email", "caterpillar@mail.com");
result.updateRow();
```

ResultSet新增、更新、刪除資料

- 如果想要新增資料

```
ResultSet result = stmt.executeQuery(
            "SELECT * FROM t_message WHERE
name='caterpillar'");
result.moveToInsertRow();
result.updateInt("id", 4);
result.updateString("name", "jazz");
result.updateString("email", "jazz@mail.com");
result.updateString("msg", "message4...");
result.insertRow();
```

ResultSet新增、更新、刪除資料

- 要刪除查詢到的某筆資料

```
ResultSet result = stmt.executeQuery(
            "SELECT * FROM t_message WHERE
name='caterpillar'");
result.last();
result.deleteRow();
```

ResultSetMetaData

- Meta Data即「資料的資料」（Data about data）
- ResultSet用來表示查詢到的資料，而ResultSet資料的資料，即描述所查詢到的資料背後的資料描述，即用來表示表格名稱、欄位名稱、欄位型態
- 可以透過ResultSetMetaData來取得

ResultSetMetaData

```
dbsource = new SimpleDBSource();
conn =  dbsource.getConnection();

stmt = conn.createStatement();
ResultSet result = stmt.executeQuery(
                        "SELECT * FROM t_message");
ResultSetMetaData metadata =
                        result.getMetaData();

for(int i = 1; i <= metadata.getColumnCount(); i++) {
    System.out.print(
            metadata.getTableName(i) + ".");
    System.out.print(
            metadata.getColumnName(i) + "\t|\t");
    System.out.println(
            metadata.getColumnTypeName(i));
}
```