

# Java資料庫程式設計 與應用(JDBC)

# 目錄 了解Java資料庫程式設計(JDBC)

- Module 1. JDBC API 簡介
- Module 2. JDBC Driver 介紹
- Module 3. Driver Manager 介紹
- Module 4. JDBC Driver 向 Driver Manager 註冊
- Module 5. 設定連接資料庫的 URL 字串
- Module 6. 建立與關閉資料庫連線

# 目錄 學會如何在Java中存取資料庫

- Module 7. 產生靜態 SQL 指令- Statement介面
- Module 8. ResultSet 介面
- Module 9. 產生動態 SQL 指令- PreparedStatement介面
- Module 10. 動態 SQL 指令的運用
- Module 11. 呼叫資料庫的預存程序-- CallableStatement 介面
- Module 12. Blob & Clob介紹

# 目錄 JDBC的進階應用

- Module 13. 圖形資料處理
- Module 14. MetaData
- Module 15. ResultSetMetaData
- Module 16. 錯誤處理
- Module 17. 批次更新(batch Updates)
- Module 18. 交易(Transaction)
- 補充: Data Access Object (DAO) 設計模式
- 補充: 連線池簡介 (Connection Pool)

# 本課程需要技術

- Java 8 以上版本
- 熟悉關聯式資料庫的 SQL 語法

# 確認資料庫可接受連線

- 1. SQL Server 2019 設定管理員(紅色工具箱) > SQL Server 網路組態 > MSSQLServer 的通訊協定 > 共用記憶體(啟用) > TCP/IP(啟用)



TCP/IP 右鍵內容 > 標籤選 IP ，拉到最下 > IPALL TCPIP 通訊埠改 1433

- 2. SSMS > 資料庫上層 > 右鍵 > 屬性 > 安全性 > 伺服器驗證請選擇 > SQL Server 及 Windows 驗證模式



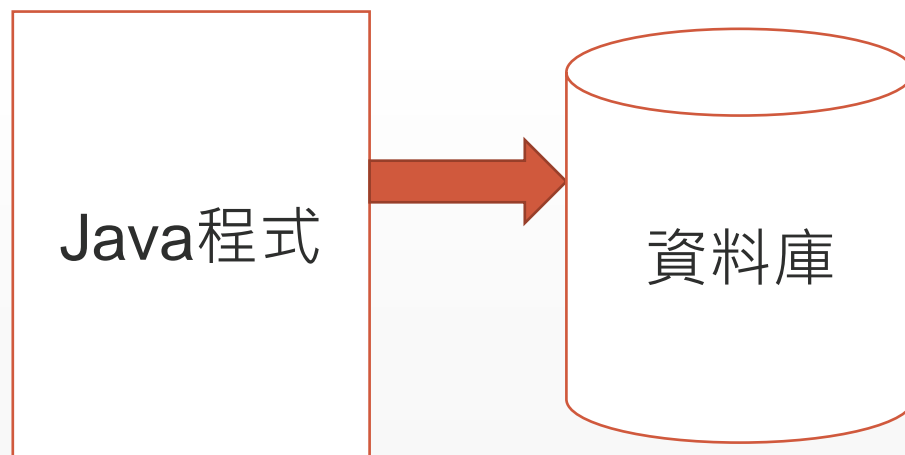
註: 若沒有紅色工具箱，則用 Windows 的電腦管理。

# 確認資料庫可接受連線

- 3. 控制台 > 系統及安全性 > Windows Defender 防火牆 > 左方進階設定 > 輸入規則 > 新增規則 > 通訊埠 > 下一步 > TCP, 特定欄位輸入 1433 > 允許連線 > 網域、私人、共用全部打勾 > 下一步 > 名稱輸入 SQL Server > 完成。
- 4. SQL Server 設定管理工具(紅色工具箱) > SQL Server 服務 > SQL Server 右鍵 > 重新啟動
- 5. 若以上 4 個步驟做完還是有問題，請重新開機。

# 學習目的

- 本課程學習如何透過 Java 程式控制資料庫，並執行 SQL 指令，達成我們要將用戶、訂單等資料，透過 Java 程式儲存到資料庫中，又或是將資料庫中的資料讀取出來，讓資料可以被利用。





# Module 1

## JDBC API簡介

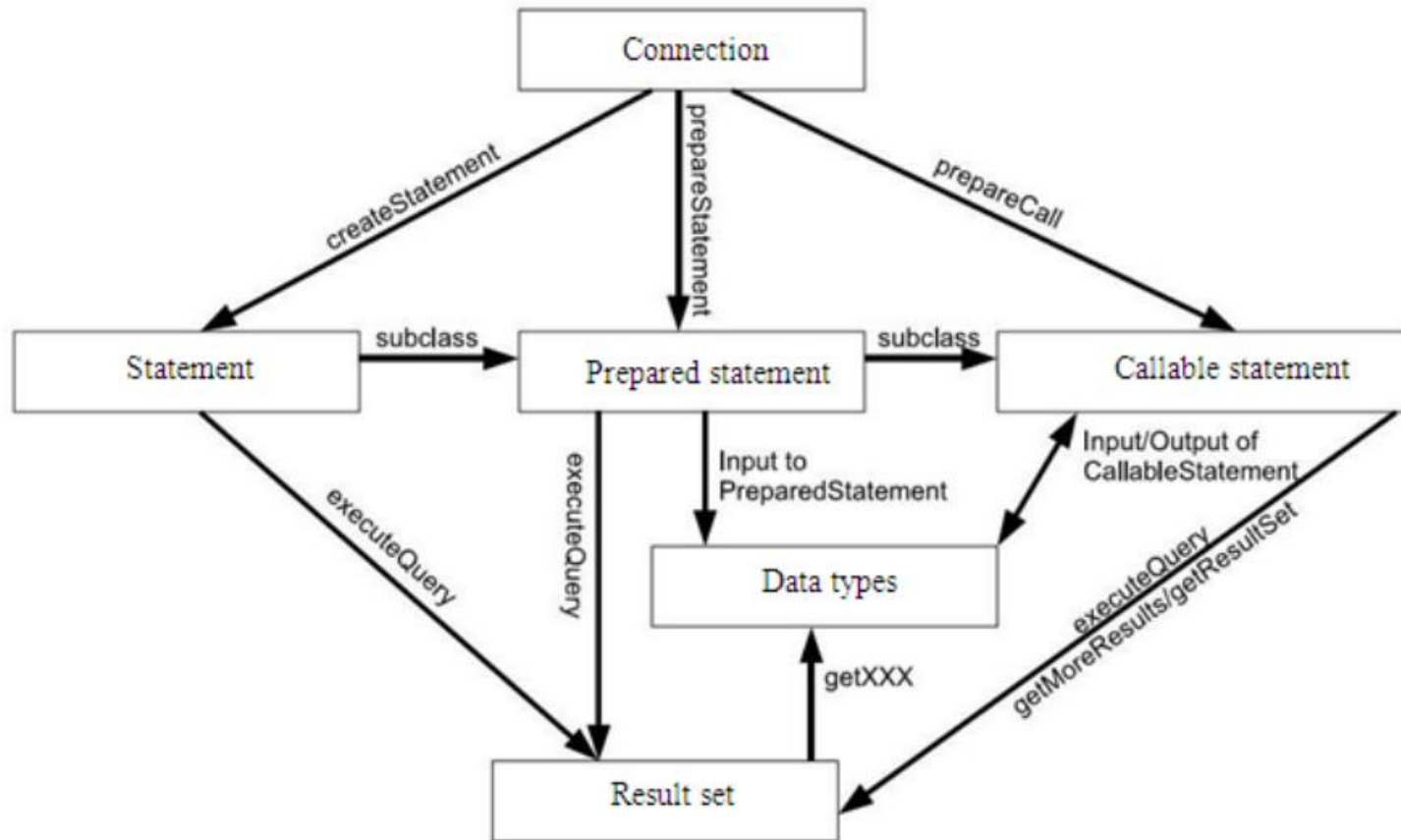
# Module 1 : 什麼是API

- API 是 **A**pplication **P**rogramming **I**nterface 的縮寫  
中文為: 應用程式介面
- API 是指軟體內提本身是抽象的，它只定義一個介面
- API 的特性：我們只需要知道其方法名稱，不需要知道底下的程式如何實作，呼叫其方法就可以應用。非常方便。

# Module 1 : JDBC API 與 java.sql 套件

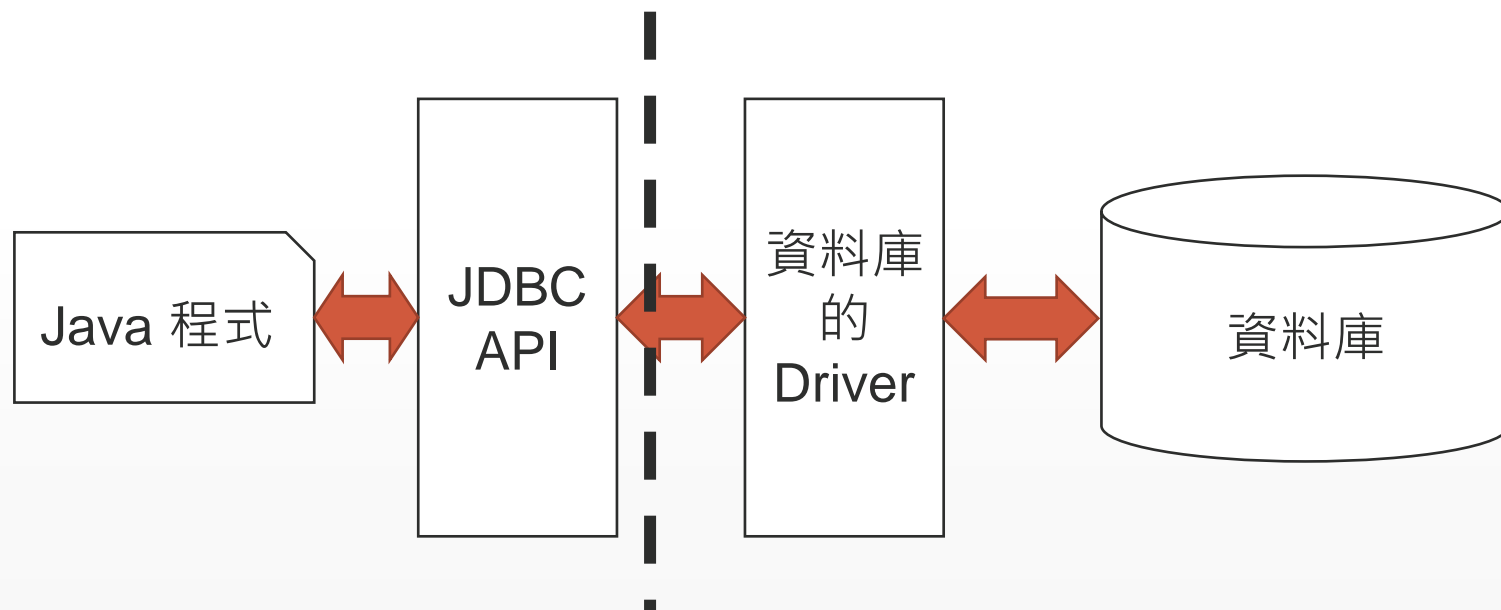
- Java 裡面提供了一個 API，以提供我們對資料庫進行存取
- 這個 API 名為 JDBC API
- JDBC 全名為 (Java Database Connectivity)
- 這個 API 已經包含在官方 Java 的 java.sql 和 javax.sql 套件內
- 只要取得、安裝對應的 JDBC Driver，就可以利用這些 API 內的方法操作資料庫內的資料。

# Module 1 : JDBC API 物件圖示(java.sql 套件)



# Module 1 : JDBC到底做什麼

- JDBC API 他的作用是拿來連接 Java 與資料庫之間，讓我們可以使用 Java 的程式，來存取我們的關聯式資料庫 (Relational Database)。



# Module 1 : 寫JDBC程式的步驟

1. 引入/導入 JDBC 的 Driver
2. 註冊 JDBC Driver
3. 建立起一個連線
4. 執行查詢或其他 SQL 指令
5. 從結果集(ResultSet)中得到資料
6. 清理環境(關閉物件)

# Module 2

## JDBC Driver

# Module 2 : JDBC Driver

- 所謂的 JDBC Driver 是指
  - 資料庫的廠商，依據 JDBC API 所提供的介面，對這個介面進行了實作後打包而成的檔案，這個檔案就稱為 JDBC Driver (驅動程式)。
- 這個檔案主要是由資料庫的廠商所提供，可以透過其官方文件說明或 Java 內的 `sql package` 文件查詢如何使用。



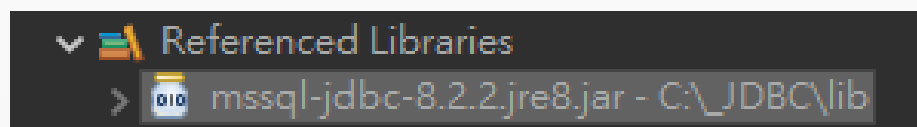
## Module 2 : JDBC Driver的取得

- MS SQL : <https://docs.microsoft.com/zh-tw/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-ver15>  
或是 Google 搜尋 sql server driver ，找到微軟的官網底下的網站。
- MySQL : <https://dev.mysql.com/downloads/connector/j/>
- 如果是其他品牌的資料庫，通常也可以在那些廠商的網站中找到他們的JDBC Driver

# Module 2 : JDBC的部署(1)

- JDBC 的部屬主要有兩種方式，我們可以依目的分為
  - 為單一專案部署、建立 library 提供給多個專案使用。
- 為單一專案部署方式 - 在專案上右鍵->Build Path->Add External Archives...  
選擇要載入的JDBC Driver(為.jar檔) 就可以順利載入。

載入完成後會在專案中出現如圖所示的Library

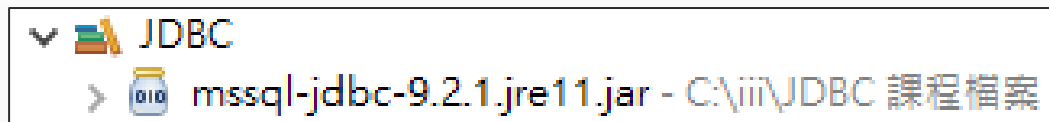


# Module 2 : JDBC的部署(2)

- 第二種方式
- 有時在一個專案中會用到不同的Driver或是套件，為了集中管理，將這些檔案先建立一個 Library (程式庫)，以後需要使用同一組套件來進行開發時，開發人員可以將這個程式庫直接部署到專案中，節省大量的時間。
- 部署方式-
  1. 建立 Library (程式庫)
  2. 放入 JDBC Driver (jar包)
  3. 在專案中部署

# JDBC的部署-Library的建立(2)

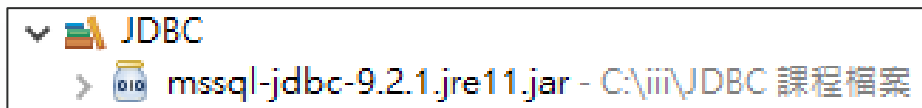
1. 選擇 Window->Preferences
2. 在 Preferences畫面下找到 Java->Build Path->User Libraries
3. 選擇 new 對 Library進行命名
4. 選擇 Add External JARs...
5. 選擇要使用的套件後 Apply and Close



建立完成後會在框中產生如圖所示之結果就表示成功建立

# JDBC的部署-部署到專案(2)

1. 在專案中右鍵->Build Path->Add Libraries...
2. 選擇User Library後下一步
3. 勾選剛剛設置完成的程式庫
4. Finish 部署完成



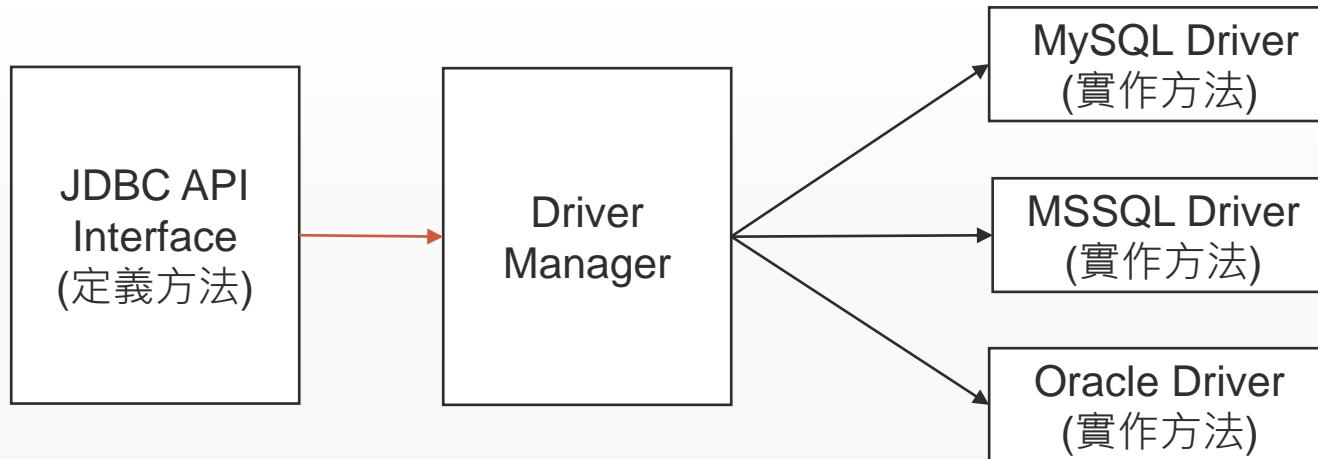
部署完成後會出現與當初命名一致的程式庫  
裡面的檔案與程式庫中的檔案應一致

# Module 3

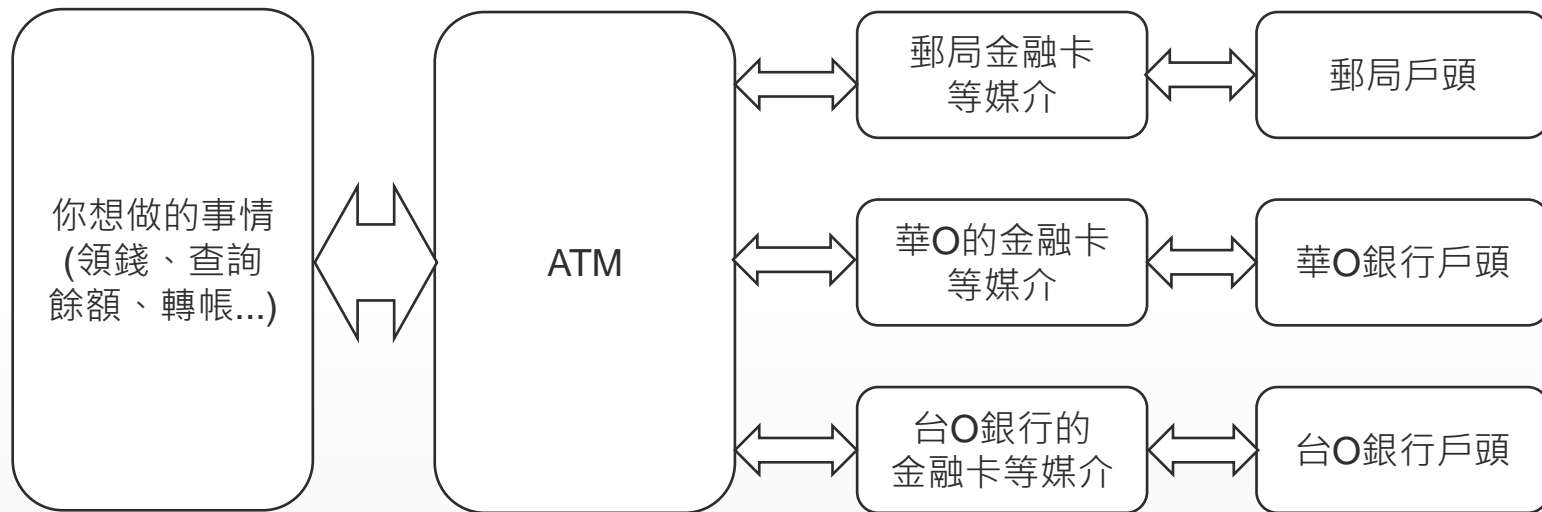
## Driver Manager

# Module 3 : Driver Manager

- Driver Manager 負責協助確認連線是否支援該註冊的驅動程式 Driver，並透過使用者的帳號密碼取得資料庫連線。
- 一個Driver Manager可以讓多個Driver 給註冊。



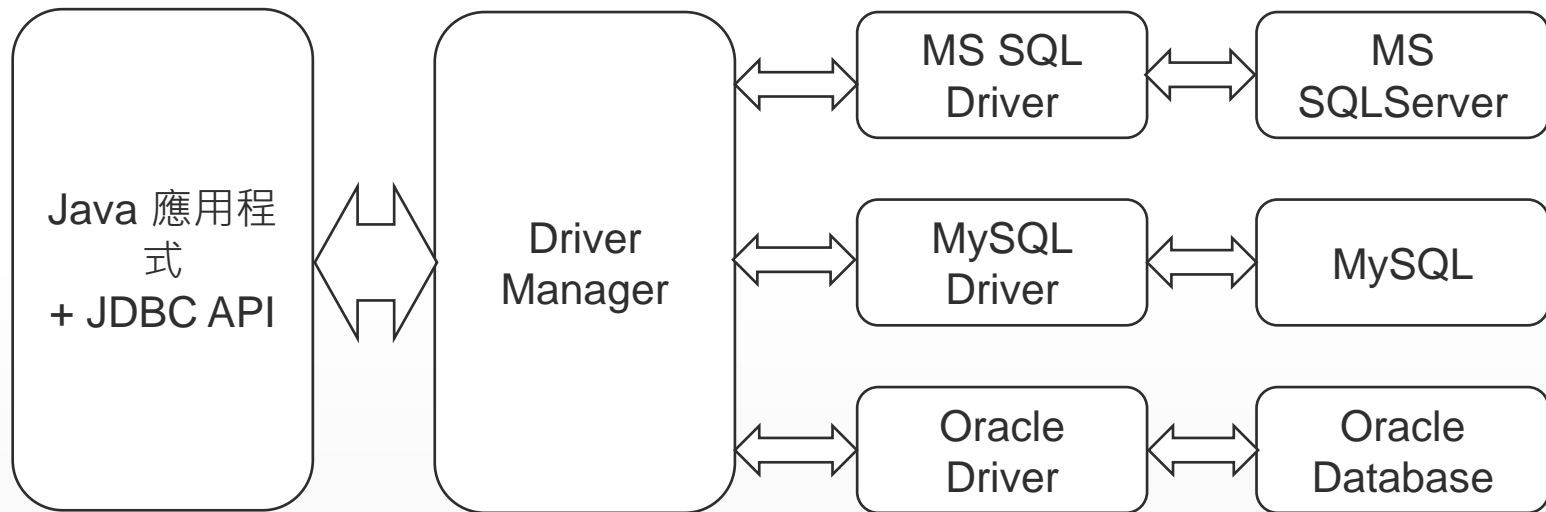
# Module 3 : Driver Manager的工作





# Module 3 :

## Driver Manager的工作



# Module 3 :

## Driver Manager 常用的方法

1. `getConnection(String)` 方法: 負責取得資料庫連線物件，傳入參數資料為資料庫連結的 URL (已包含使用者帳號及密碼)
2. `getConnection(String, String, String)` 方法: 負責取得資料庫連線物件，參數分別為資料庫連結的 URL、使用者帳號、使用者密碼。

# 第一隻Java資料庫程式

# Module 4

## 註冊Driver

# 寫JDBC程式的步驟

1. 引入/導入 JDBC 的 Driver
2. 註冊 JDBC Driver
3. 建立起一個連線
4. 執行查詢或其他SQL指令
5. 從"結果集"中得到資料
6. 清理環境(關閉物件)

# Module 4

## 註冊 DriverManager

- 要使 Java 程式與資料庫互動，必須要透過廠商所建立的套件，也就是Driver - 驅動程式。
- 須將這個套件進行註冊才能在程式中使用。
- Driver 的註冊：透過 DriverManager 進行註冊，
- 註冊之後 DriverManager 就可以管理 JDBC Driver 。

# Module 4

## JDBC Driver 註冊的三種方式

1. 透過 DriverManager 註冊驅動程式:

```
DriverManager.registerDriver(new xxxDriver());
```

2. 透過系統的屬性設定註冊驅動程式:

```
System.setProperty("jdbc.drivers", "驅動程式名稱");
```

3. 透過 Class.forName 的方式載入驅動程式(建議使用)

```
Class.forName("驅動程式名稱");
```

# Module 4

## DriverManager.registerDriver(舊版寫法)

- 程式片段：`DriverManager.registerDriver(new com.microsoft.sqlserver.jdbc.SQLServerDriver());`
- 這個方法會因為 Driver 中也對其進行實體化，所以會建立 2 個驅動程式物件在程式中，( DriverManager 和 SQLServerDriver 物件 )，導致效能降低，所以這種方式已逐漸被淘汰。
- 這方法可能會在較舊的專案中看到
- 雖本課程不使用此方式，但還是要看過。



# Module 4

## Class Loader 方式

- `Class.forName` -
  - `forName` 方法是在 `Class` 類別下的一個靜態方法，這個方法會去尋找、並載入 JDBC Driver 到程式當中。
  - 這個方法會透過類別載入器找到類別，如果找不到這個類別，則會產生一個 `ClassNotFoundException` 的例外。
  - 範例：

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
```

# Module 4

## 新式載入方式，自動載入

- 這種載入方式是在 Java SE 1.6 版本之後誕生的。
- 直接使用 Java 原生物件 `java.sql.DriverManger` 就會自動註冊，方便之後準備連線。

程式如下：

```
DriverManager.getConnection(url, user, password);
```

# Module 5

## 連接資料庫的URL

# 寫JDBC程式的步驟

1. 引入/導入 JDBC 的 Driver
2. 註冊 JDBC Driver
3. 建立起一個連線(事前準備部分)
4. 執行查詢或其他SQL指令
5. 從結果集中得到資料
6. 清理環境(關閉物件)

# Module 5 : JDBC的URL介紹

- JDBC的URL 定義了連接資料庫時相關的協定(protocol)、子協定(subprotocol)、資料來源(subname)。
- 連線字串的格式與範例：  
`jdbc:sqlserver://localhost:1433;databaseName=JDBCDemoDB`

語法:

協定:子協定://資料來源

- 在JDBC中使用的協定就是 jdbc
- 子協定通常是資料庫管理系統的名稱  
如 : sqlserver,mysql,oracle.....
- 資料來源包含資料庫的位址以及連接所使用的埠號等資料庫所需要使用的資訊。

# Module 5 :

## URL細節

- 在URL中，會因每個資料庫的不同，連線字串的寫法會有所差異
- 協定:子協定://主機位置:埠號[ MSSQL:(;),MySQL:(/),Oracle:(:)]資料庫名稱  
`jdbc:sqlserver://localhost:1433;databaseName= JDBCdemoDB`
- 主機位置：寫上主機的 IP 位置或是一個註冊的名稱，localhost 表示連接於本機。
- 埠號(port)：用來連線服務的號碼，用來連接各種服務，著名的服務的埠號基本上都介於0-1023，而1024後的號碼則未被限制，上限為65535。
- 基本上每個資料庫都有慣用的埠號 如:MS SQL=>1433,MySQL=>3306等
- 而後方是資料庫名稱，其設置方式與資料庫品牌會有所不同，下面一頁投影片則作為舉例，詳細部分則需要查詢相關的資料庫連線URL。

# Module 5 :

## URL範例

- MS SQL:  
jdbc:sqlserver://localhost:1433;databaseName=JDBCDemoDB
- MySQL:  
jdbc:mysql://localhost:3306/JDBCDemoDB
- 在MySQL較新的版本中需要指定是否為SSL連線及時區與文字編碼方式，多個參數要設定時中間以&分隔。
- jdbc:mysql://localhost:3306/  
JDBCDemoDB?useSSL=false&serverTimezone=UTC&characterE  
ncoding=UTF-8

# Module 6

## 建立與關閉資料庫連線



# 寫JDBC程式的步驟

1. 引入/導入 JDBC 的 Driver
2. 註冊 JDBC Driver
3. 建立起一個連線
4. 執行查詢或其他SQL指令
5. 從結果集中得到資料
6. 清理環境(關閉物件)

# Module 6

## 建立連線

- 在註冊完 JDBC Driver 後，就可以透過 DriverManager 取得連線物件。
- 在取得連線物件時須要以下資訊：
  1. 連線字串URL
  2. 資料庫的使用者名稱(username)
  3. 資料庫的使用者密碼(password)

# Module 6

## 取得連線getConnection

- 準備好剛剛所要求的 3 個資訊之後，請按照 URL、帳號、密碼的順序將這三個字串提供給 DriverManager 的 getConnection 方法中
- 架構：
  - `DriverManager.getConnection("jdbc:sqlserver://localhost:1433; databaseName=JDBCDemoDB";帳號;密碼);`
- 這個方法會回傳一個連線物件(Connection)，這個連線物件就是幫助我們傳遞資料庫所使用的 SQL 語句給資料庫進行執行的一個類別，後續要對資料庫做新增、查詢、刪除、修改等都會使用到這個物件。

# Module 6

## 資源釋放

- 對於資料庫相關的程式、最好是使用完時就立即將連線關閉，因為資料庫的連線是相當的貴重(比較占記憶體)。
- 使用 `Connection` 物件中的 `close()` 方法，將連線的資源歸還(釋放記憶體)。
- 由於不能確認每一次連線都會成功，所以可以先使用 `Connection` 物件中的 `isClosed()` 方法，檢查是否有該連線物件。
- `isClosed()` 方法在有連線物件的情況下，會回傳 **false**。
- 如此一來，就可以確定連線物件的存在以及是否可以被關閉。

# Module 6

## 取得連線 getConnection 程式片段

```
try {  
    Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");  
    String urlStr = "jdbc:sqlserver://localhost:1433;databaseName=JDBCdemoDB;user=sa;password=passw0rd!";  
  
    Connection conn = DriverManager.getConnection(urlStr);  
  
    boolean status = !conn.isClosed();  
  
    System.out.println("Connection Open status:" + status);  
  
    conn.close();  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

# Module 6

## 自動釋放資源: try-with-resource

- 在 Java SE 7 開始，Java 提供了 try-with-resource 語法，這種語法可以代替我們執行 close() 方法來進行釋放資源，可以使資源得到更好的利用，同時大大縮短了程式碼的長度。
- 使用 try-with-resource 語法時需要注意，寫在 try() 中需要繼承 AutoCloseable 的介面才能進行所謂的自動關閉，且這個變數只能在 try 區塊所使用。
- Connection 物件與 Statement 物件都有實作 AutoCloseable 所以可以使用 Try-with-resource 方式 進行資源控制
- 可見 Java11 的 Connection 官方文件，如下方網址。
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.sql/java/sql/Connection.html>

# Module 6 :自動釋放資源: try-with-resource 程式片段

```
try (Connection conn = DriverManager.getConnection(urlStr)) {  
    System.out.println("連線成功");  
} catch (SQLException e) {  
    e.printStackTrace();  
    System.out.println("連線失敗");  
}
```

註：urlStr 為連線字串

# Module7

## 產生靜態SQL指令- Statement



# 寫JDBC程式的步驟

1. 引入/導入 JDBC 的 Driver
2. 註冊 JDBC Driver
3. 建立起一個連線
4. 執行查詢或其他SQL指令
5. 從結果集中得到資料
6. 清理環境(關閉物件)

# Module 7

## Statement 介面簡介

- 將 SQL 敘述交給 Statement 物件，這個物件可以訪問資料庫後取得回傳的結果。
- 在傳遞前雖然可以用串接字串的方式進行傳遞參數，但會有風險(SQL -Injection)。
- SQL -Injection 為基本的資料庫攻擊，稍後在 PreparedStatement 章節說明其原理，以及防範方法。

# 使用 Statement 物件的步驟

1. 由連線物件產生 Statement 物件
2. 編寫要使用的 SQL 指令字串
3. 提供給 Statement 的 executeQuery 方法，同時取得結果集 ResultSet
4. 取出結果集資料
5. 釋放資源

# Module 7

## 連線物件產生Statement物件

- Statement物件是從連線物件(Connection)中取得的
- 這個物件需要透過 Connection 的 createStatement 方法取得
- 範例如下：  

```
Statement stmt=conn.createStatement();
```
- 選擇對應的方法
  - executeQuery : 查詢使用，回傳資料 (ResultSet)
  - executeUpdate : 會修改到資料時使用，回傳修改筆數(int)
  - execute : 只回傳布林值(boolean)，若是查詢語法(Select)則回傳 true

# Module 7

## executeQuery

- `executeQuery()` 這個方法只能使用在查詢時，不能在更動資料時使用。
- 本方法是用來執行查詢的SQL語句，它會回傳一個結果集(`ResultSet`)，這個結果集裡面包含了查詢的結果、欄位等表格資訊。
- 程式片段：
  - `String sql="select * from demotable";`
  - `ResultSet rs = stmt.executeQuery(sql);`

	id	name	address	tel
1	1007	Kitty	USA	0012345678
2	1008	Bob	Taiwan	0987654321

結果集示意圖

# Module 7

## executeUpdate

- `executeUpdate` 這個方法是用在刪除、新增、修改或 DDL (建立刪除表格一類)，會對資料表產生變動的 SQL 語句進行執行。
- 這個方法會回傳一個整數值 `int`，這個數值代表已成功變更、新增、刪除的筆數。
- 如果是 DDL，成功回傳 0, 失敗回傳 -1
- 程式片段：
  - `int row = stmt.executeUpdate(sql);`
  - `System.out.println("成功變更"+row+"筆資料");`

# Module 7

## execute() 方法

- 可以執行前面介紹的 executeQuery 與 executeUpdate 能執行的SQL語句。
- 回傳一個布林值，如果執行的是查詢語句(Select)會回傳 true ，否則回傳 false 。
- 範例：
  - `boolean b=stmt.execute(sql);`

# Module8

# ResultSet



# Module8

## ResultSet 使用步驟

1. 從 Statement 取得查詢過後的結果集 ResultSet
2. 控制游標、選擇要讀取的項目。
3. 讀取欄位資料。
4. 釋放資源。

# Module8

## ResultSet 簡介

- ResultSet 是使用 `executeQuery` 過後，Statement 會回傳一個結果集，這個結果集中包含著查詢的結果。
- 這是一個已經打包好的物件，我們要透過他的一些方法來取出結果集中每一個欄位的資料。
- 取出資料的時後要注意資料指標(cursor)的位置

	id	name	address	tel
1	1007	Kitty	USA	0012345678
2	1008	Bob	Taiwan	0987654321

結果集示意圖

# Module8

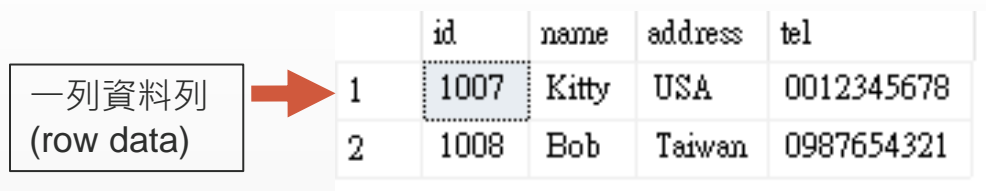
## ResultSet 的產生

- ResultSet 是由Statement 執行查詢語句後產生的物件，這個物件裝有查詢的結果。
- 產生的方式：
  - 由 Statement 的executeQuery方法執行查詢語句
    - `ResultSet rs = stmt.executeQuery(sql);`
  - 由 Statement 的 execute 方法執行查詢語句
    - `stmt.execute(sql); ResultSet rs = stmt.getResultSet();`

# Module8

## ResultSet 的 Cursor(指標) 控制

- 在 ResultSet 中要取得想要的資料，要先控制 cursor 的位置。
- cursor 所控制的是選擇第N筆資料列
- 透過 next() 方法取的資料列指標，判斷是否有資料
- 第一個 next() 是第一筆資料



	id	name	address	tel
1	1007	Kitty	USA	0012345678
2	1008	Bob	Taiwan	0987654321

結果集示意圖

# Module8

## ResultSet 一筆資料的欄位值讀取

- 一開始 cursor 的位置是在**第一筆資料之前**，所以 cursor 要先執行一次 next 方法來到第一筆資料。
- 當 ResultSet 用 cursor 選擇到資料後，要使用 get+ 資料型別來取得資料。  
E.g. getString(), getTime()
- 在 get 方法中可以傳入兩種參數
  - 欄位名稱
  - 欄位索引(**索引值從1開始，根據順序**)
- 常用的get方法除基本資料型別外(不包含字元)，還有getString(), getDate(), getTime(), getTimestamp(), getBinaryStream()...等方法
- 其中有兩種方法可以取得所有資料型別的資料：
  - getString(), getObject()

# Module8

## ResultSet 程式範例

```
ResultSet rs = stmt.executeQuery(sql); //執行sql語句
while(rs.next()) {
    System.out.println(rs.getString(1)+rs.getString(2)+rs.getString(3)+rs.getString(4));
}
```

# Module8

## 實作練習與作業

- 請在上課的範例 Demo3CreateStatement 中加個幾方法：
  1. 方法 updateDataHW() : 請把 1001 這筆資料的價錢改為15，成功要印出 Update Product Success。
  - 2.方法 insertDataHW() :請薪資一筆商品資料名為 Mac mini，其他欄位自訂，成功要印出 Insert Product Success。
  3. 方法 deleteOneDataHW() : 請刪除名稱為 mask 的這筆資料，成功要印出 Delete Product Success。

# Module 9

## 產生動態SQL指令- PreparedStatement



# Module 9

## PreparedStatement 介紹

- PreparedStatement 繼承了 Statement 的介面，他除了可以處理靜態的SQL語句之外，還可以用來處理動態的SQL語句。
- PreparedStatement 可以透過 Connection 物件得到，在利用該物件取得 PreparedStatement 時，需要先輸入 SQL 語句，這樣就會回傳一個儲存好預先編譯的 PreparedStatement 物件，此時就可以利用這個物件來做到所謂的動態處理 SQL 指令。
- 預先編譯的好處就是 - > 資料庫對於需要重複使用的 SQL 敘述先進行編譯及快取，這樣可以避免資料庫重複編譯同一句 SQL 指令，如此可以提升效能。因為只換參數而已。
- 也可以避免 SQL Injection 的攻擊。

# Module 9

## 動態SQL指令介紹

- PreparedStatement 在建立時需要傳入一個動態的 SQL 指令字串。
- 動態的SQL指令字串，是在需要填入的位置中設置一個問號(?)，這種方式傳入的參數會是一個字串，除了可以動態的傳入SQL指令，還可以將程式碼重複利用，不需要再建立 PreparedStatement 物件。
- 動態SQL指令字串示範：

`select * from sampletable where Customerid=?`

- ?的部分需要用 set 方法來設置，傳入過後會將傳入的字串調整成合適的模樣。

# Module 9

## set方法設置變數

- ? 的部分需要用 set 方法來設置，set方法有：  
setBoolean()、setShort()、setInt()、setLong()、setDate()、  
setTime()、setTimestamp()、setBinaryStream()、setAsciiStream()、  
setObject()、setBlob()、setClob()、setString()等。
- 在set方法中需要傳入 2 種參數，一種是第幾個(用?表示)，另一種是傳入的值。
- 而後面的傳入值要傳入相對應的物件。
- 範例：

`stmt.setString(1, "早安");` // 表示第 1 個問號參數帶 "早安" 進去

# Module 9

## PreparedStatement 執行 SQL

- `executeQuery()`  
回傳 `ResultSet` 物件，這個方法是用來進行資料庫的查詢，會回傳一個結果集的物件。
- `executeUpdate()`  
回傳一個整數 `int`，這個方法是用來執行資料庫的更新，並回傳變更的資料列數目。

ResultSet 物件



	id	name	address	tel
1	1007	Kitty	USA	0012345678
2	1008	Bob	Taiwan	0987654321

`executeUpdate` 的回傳值(int)



(1 個資料列受到影響)

# Module 10

## 動態SQL指令的運用

# Module 10

## 動態新增 SQL 語法(Insert)

```
String sqlStr = "Insert Into Profiles(name, address, phone)  
Values(?,?,?)";
```

```
PreparedStatement preState = conn.prepareStatement(sqlStr);
```

```
preState.setString(1, "katy");
```

```
preState.setString(2, "Hawaii");
```

```
preState.setString(3, "006-126-654");
```

```
preState.executeUpdate();
```

```
preState.close();
```

# Module 10

## 動態修改 SQL 語法(update)

```
String sqlStr = "Update Profiles Set phone=? Where name=? and  
address=?";
```

```
PreparedStatement preState = conn.prepareStatement(sqlStr);
```

```
preState.setString(1, "001-123-456");
```

```
preState.setString(2, "katy");
```

```
preState.setString(3, "hawaii");
```

```
int count = preState.executeUpdate();
```

```
preState.close();
```

```
System.out.println("Update data count:" + count);
```

# Module 10

## 動態刪除 Delete

```
String sqlStr = "Delete From Profiles Where id=?";  
PreparedStatement preState =  
conn.prepareStatement(sqlStr);  
preState.setInt(1, 1);  
preState.execute();  
preState.close();  
System.out.println("Delete Finish");
```



# Module 10

## SQL Injection 說明與防範

- SQL Injection Attack 又稱為 SQL 隱碼攻擊，攻擊者使用資料庫查詢語法入侵該網站的資料庫，通常是透過正常查詢指令夾雜著惡意命令，攻擊者就可以對資料庫的資料進行更動。
- 假設今日發生在登入頁面，使用同手法，在帳號處出入 `OR 1=1 --`，讓前面單引號封閉對應的單引號，註解掉後面的判斷式，攻擊者便可跨越驗證，直接登入進去。
- PreparedStatement 可以避免攻擊者拼湊惡意的指令，因為?是由程式內部給予。
- 上述只是其中一種 SQL Injection 攻擊，其他詳細請見下方。
- 詳細參考: <https://ithelp.ithome.com.tw/articles/10189201>

# Module 11

呼叫資料庫的預存程序

-- CallableStatement

# Module 11

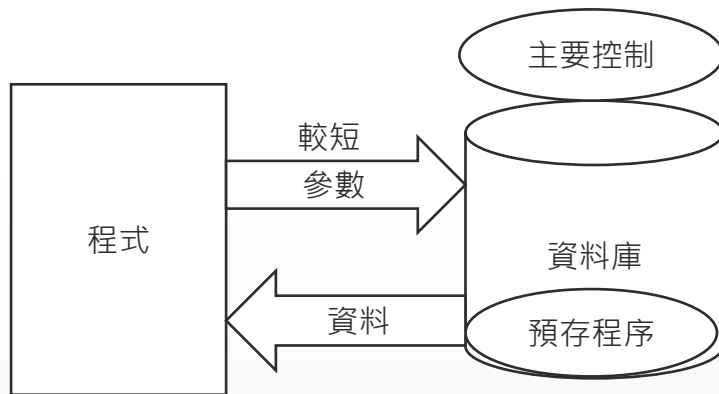
## CallableStatement 預存程序簡介

- 預存程序是將寫好的商業邏輯的部分，以事先編譯的方式儲存在資料庫中，之後可以藉由呼叫該預存程序並提供參數給該程序，以讓預存程序完成特定的商業邏輯。
- 優點：
  - 預存程序在編輯時就能進行除錯。
  - 預存程序因需要傳遞的物件縮小，且已在資料庫進行編譯，效率大幅上升。
- 缺點：
  - 預存程序是建立在資料庫上，所以要切換到其他系統時，會因為使用的語法有所不同，而需要**重寫**原有的預存程序。
  - 預存程序的效能調整、編寫預存程序會被受限於資料庫的系統。

# Module 11

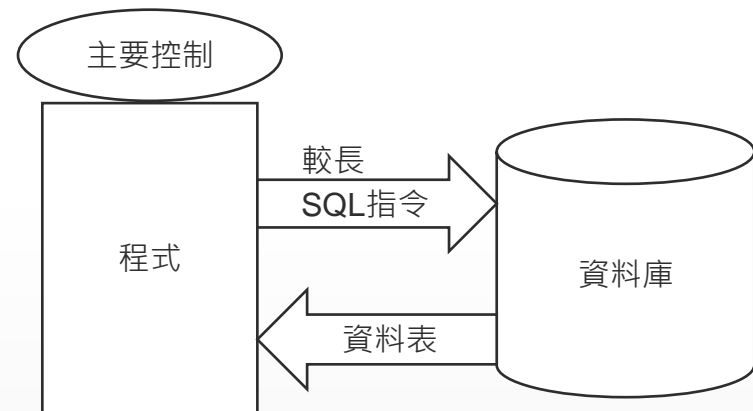
## CallableStatement 與 PreparedStatement 比較

### CallableStatement



以前主流

### PreparedStatement



目前主流

# Module 11

## 預存程序: 常用的方法

- 定義在 java.sql 套件內，常用方法:
- prepareCall()方法: 負責取得指定的預存程序，並回傳一個 CallableStatement
- registerOutParameter(int, SQL Type)方法: 用來註冊輸出參數的資料型別，傳入參數分別為參數位置、參數的 SQL 型別(java.sql.Types.xxx)
- executeQuery() 方法 : 執行預存程序為 SQL 語法查詢的 CallableStatement 敘述。
- execute() 方法 : 負責執行所有 SQL 語法的 CallableStatement 敘述。

# Module 11

## 示範新增一筆預存程序

```
Create Procedure productProc(@proId int, @proName varchar(40) OUT)
AS
BEGIN
    SELECT @proName = productname From product Where productid =
    @proId
END
GO
```

註: 如果只有一行SQL(如上) , Begin 跟 End 可以省略

# Module 11

## 預存程序 IN 和 OUT 參數

```
CallableStatement callState = conn.prepareCall("{call productProc(?,?)}");  
callState.setInt(1, 1002); //設定輸入參數(IN)  
callState.registerOutParameter(2, java.sql.Types.VARCHAR); // 輸出參數  
(OUT)  
callState.execute();  
String name = callState.getString(2);  
System.out.println("name:" + name);  
callState.close();
```

Module 12

Blob & Clob介紹



# Module 12

## BLOB與CLOB

- BLOB與CLOB的主要目的是為了將檔案寫進資料庫中。
- BLOB：二進位大型物件binary large object的縮寫，用來儲存圖片、影像、音樂等檔案類型的物件。
- CLOB：字元大型物件Character Large Object的縮寫，用來儲存大篇幅的文章、文件或是很長的文字，如留言板、專欄文章。

# Module 12

## 對應資料庫的格式(型別)



- BLOB 與 CLOB 在各版本的資料庫中有許多用不同的資料型別來儲存這兩種類型。
- 而儲存文字、小型的檔案則原本資料庫就又提供所謂的 char、varchar 或是 binary、varbinary 這一類型的資料庫格式，但這些原先都是設計給少量的、一次讀取完畢的資料。
- 故產生了 BLOB、CLOB 的格式，而這些常使用串流的方式(stream)來傳送，因為串流並非一次讀取完畢，這種施行的方式更適合傳遞大型檔案以降低網路負載。
- 而 MS SQL 使用 varbinary(max)、image 來進行儲存 BLOB
- 使用 text、varchar(max)來儲存 CLOB
- 新版(2017以後) MS SQL 請避免使用 text, ntext, image 型別(詳情請見下一頁)

# Module 12

## 對應資料庫的格式(型別)

### ntext、text 和 image (Transact-SQL)

2017/07/22 ·   

適用範圍：  SQL Server (所有支援的版本)  Azure SQL Database

用來儲存非 Unicode 字元和 Unicode 字元及二進位資料的固定和可變長度資料類型。Unicode 資料使用 UNICODE UCS-2 字元集。

**重要！** SQL Server 的未來版本將會移除 **ntext**、**text** 及 **image** 資料類型。請避免在新的開發工作中使用這些資料類型，並規劃修改目前在使用這些資料類型的應用程式。請改用 **nvarchar(max)**、**varchar(max)**和 **varbinary(max)**。

來源: <https://docs.microsoft.com/zh-tw/sql/t-sql/data-types/ntext-text-and-image-transact-sql?view=sql-server-ver15>  
或是在 Google 搜尋 MSSQL Image

# Module 12

## 如何寫進資料庫

- 雖然JDBC 2.0開始支援Blob、Clob介面來處理SQL的BLOB、CLOB，然而並不建議使用此種資料型態來進行寫入，因為此兩種資料型態並非是最有效率的方式。
- 雖然PreparedStatement中有setBlob()及setClob()方法，然而寫入、讀取資料建議使用資料流(Stream)的方式來進行。
- 使用資料流來設置：
- PreparedStatement中有setBinaryStream()方法，這個方法可以用來處理BLOB。
- PreparedStatement中有setAsciiStream()方法與setCharacterStream()，這些方法可以用來處理CLOB。
- 這幾個方法都可以提供給它們資料流InputStream或Reader(setCharacterStream())，來傳送資料進入資料庫。

# Module 12

## 讀取資料庫的BLOB、CLOB

- ResultSet 提供幾個方法來讀取資料庫中的大型物件(LOB)：

方法名稱	得到物件
getBinaryStream(int 欄位索引/String 欄位名稱)	InputStream
getAsciiStream(int 欄位索引/String 欄位名稱)	InputStream
getCharacterStream(int 欄位索引/String 欄位名稱)	Reader
getBlob(int 欄位索引/String 欄位名稱)	Blob

要注意得到物件的參考型別，進行轉換、編碼時才不容易出錯。

# Module 13

## 圖形資料處理

# Module 13

## 圖形資料處理 - 寫入

- 由於前面的介紹主要儲存的對象以數值資料為主，接下來要講解如何處理圖形檔案。
  1. 開啟圖片的檔案串流(Stream)
  2. 使用 `setBinaryStream()`，將開啟的圖片串流物件交給 `Statement` 物件。
  3. 執行(`execute`的方法)

# Module 13

## 將圖形檔案寫入資料庫程式範例

```
File myFile = new File("c:/temp/my_image.jpg");  
FileInputStream fis1 = new FileInputStream(myFile);  
String sqlStr = "Insert Into Gallery(imageName, imageFile)  
Values(?,?)";  
  
PreparedStatement preState = conn.prepareStatement(sqlStr);  
preState.setString(1, "SomeName");  
preState.setBinaryStream(2, fis1);  
preState.execute();  
preState.close();
```



# Module 13

## 使用BLOB介面寫出成檔案程式片段

```
String sqlStr = "Select * From Gallery Where galleryId = ?";  
PreparedStatement preState = conn.prepareStatement(sqlStr);  
preState.setInt(1, 1);  
ResultSet rs = preState.executeQuery();  
rs.next();  
Blob blob = rs.getBlob(3); // 3 是欄位順序  
System.out.println("blob.length(): " + blob.length());  
BufferedOutputStream bos1 = new BufferedOutputStream(new  
FileOutputStream("c:/temp/myImage.jpg"));  
bos1.write(blob.getBytes(1, (int)blob.length()));  
bos1.flush(); bos1.close(); rs.close(); preState.close();
```

**getBytes(開始位置,要取得的資料長度)** ·  
開始位置從1開始

# Module 14

## MetaData

# Module 14

## MetaData

- MetaData，中文稱為元資料、中介資料等，這個資料是用來描述資料用的資料。
- JDBC的MetaData 可以取得資料庫的系統資訊、資料表資訊。
- JDBC提供下面兩種MetaData。
- DatabaseMetaData (資料庫)
- ResultSetMetaData (資料表、結果集)

# Module 14

## DatabaseMetaData

下方為常用的方法以及回傳的資訊，回傳結果皆為 **String** 形式。  
這個介面的物件主要是用來了解資料庫的各種資料(版本、驅動程式、登入的角色等)。

getDatabaseProductName()	資料庫名稱
getDatabaseProductVersion()	資料庫版本
getDriverName()	驅動程式名稱
getDriverVersion()	驅動版本
getURL()	DBMS的URL
getUserName()	使用者名稱

**Module 15**

**ResultSetMetaData**

# Module 15

## ResultSetMetaData

- ResultSet 結果集中，不僅僅只有查詢的資料而已，當中還包含了許多查詢結果的一些狀態的資訊，我們可以使用ResultSet 中的 getMetaData來取得ResultSetMetaData物件。

getColumnCount()	欄位總數-int
getColumnName(int 欄位索引)	欄位名稱-String
getColumnLabel(int 欄位索引)	欄位標題名稱-String
getColumnTypeName(int 欄位索引)	欄位資料型別-String
getColumnDisplaySize(int 欄位索引)	欄位有效長度-int
isNullable(int 欄位索引)	欄位是否允許空值-int(0不允許1允許2不確定)

# Module 15

## getColumnCount

- `getColumnCount`方法會回傳我們在進行的查詢下的結果集中所具有的欄位數量。
- 請注意，是查詢結果(`ResultSet`)的欄位數量，並非是表格的欄位數量。
- 功能：只要知道有多少表格欄位，便可使用迴圈的方式處理我們的結果集，以便整理，讓程式更加簡潔。

# Module 15

## getColumnName/getColumnLable

- getColumnName/getColumnLable，這兩個方法需要提供給他一個欄位索引位置，這兩個方法皆會回傳欄位的名稱。
- getColumnName 這個方法會因為使用的資料庫不同，回傳的欄位名稱而不同，如在 MySQL 時，此方法會回傳欄位的名稱，MS SQL會回傳別名。
- getColumnLable 這個方法則會回傳欄位的別名，若無別名則是回傳欄位名稱。
- 這個方法與上一個方法聯合使用，就可以做出使用迴圈取得該欄位名稱的動作。



# Module 15

## getColumnTypeName

- `getColumnTypeName` 提供該方法一個欄位的索引位置，這個方法可以取得該欄位所使用的 SQL 資料型別。
- 這個方法可以讓使用者知道該欄位是使用什麼資料型別。
- 回傳的資料型別會是 **SQL的資料型別**，不是JAVA的資料型別。

# Module 15

## getColumnDisplaySize

- `getColumnDisplaySize`這個方法需要提供一個欄位的索引位置，這個方法將會回傳該欄位可以儲存的最大尺寸。
- 舉例：如果使用這個方法去察看一個資料型別為`varchar(50)`的欄位，這個方法會將欄位最大值50回傳至程序中。
- 可以使用這個方法來確認一個資料表的最大欄位尺寸，用來告知使用者可以輸入多少文字，或是在進入資料庫之前先行限制長度，以避免無用的傳遞資訊。

# Module 15

## isNullable

- isNullable，需要提供一個欄位的索引位置，則該方法將會回傳一個整數值，該整數代表該資料表的欄位是否允許空值。
- 傳回值：
- 0 代表不允許NULL
- 1 代表允許NULL
- 2 代表無法判斷的情形
- 這個方法可以應用在事先取得資料表的是否允許空值，取得後用來判斷使用者輸入是否正確，避免將錯誤格式的資料提供給資料庫在進行錯誤處理。

# Module 16

## 異常處理

# Module 16

## 異常處理

- 當有關資料庫的錯誤發生時，資料庫會回傳錯誤的訊息，這訊息是 `SQLException` 拋出來的。
- JDBC 主要要處理的例外是 `SQLException`。
- 異常處理主要功能：
  - 防止當異常發生時，讓使用者看到程式 shutdown 的畫面。
  - 預期發生錯誤時，`catch` 起來處理錯誤(例如: 跳回主頁面)。
  - 了解程式發生的錯誤。

# Module 16

## SQLException

- SQLException 是我們在使用JDBC時常常見到的一種例外(Exception)。
- SQLException 這個類別可以提供給我們一些有用的資訊，我們可以用下面的方法將這個例外取出。
- String getMessage()
- 這個方法可以取得錯誤、例外訊息，這個訊息由於是從資料庫方提供、所以每一家的資料庫廠商提供的訊息都不盡相同。
- int getErrorCode()
- 這個方法會回傳一個整數值，這個整數是資料庫提供的錯誤代碼，我們得到這些代碼查詢訊息之後，就能夠有效地排除錯誤。

Module 17

批次更新

(batch Updates)

# Module 17

## 批次處理

- 批次處理是指在電腦上不需操作者干預而執行一系列的程式的作業方式。
- 通常會超過 1 個 SQL 敘述才會稱為批次處理
- 現實中利用批次處理的工作，例如：銀行利息處理，特定時間間隔的統計報表生成等。
- 這一類型的工作都是要處理大量的資料，同時這些工作都是依些重複性高，而非互動性的應用程式



# Module 17

## addBatch()

- 使用 Statement 或是 PreparedStatement 這兩個介面的物件中的 addBatch() 方法，將 SQL 語法添加進批次處理中。
- 這個方法會將 SQL 指令儲存進一個隱含的 List。
- 由於在批次處理中，並沒有使用者需要接收回饋，所以 addBatch()方法僅能使用 INSERT、UPDATE、DELETE指令。
- Statement 使用 addBatch() 方法時，需要在 addBatch() 方法中直接添加 sql 指令。
- PreparedStatement 使用 addBatch()方法時，可以使用 setXXX 的方式先行設置，再使用 addBatch()方法進行添加。

# Module 17

## executeBatch()

`executeBatch()` 這個方法會將新增進入批次的資料進行執行，這個方法將會回傳一個整數陣列，這個陣列代表著每一筆的SQL語法進行新增、修改所更動的筆數，故有多少筆SQL指令，就會有多少個整數產生。

執行完畢後，會將放進List物件中的SQL指令清空。

**Module 18**

**交易 (Transaction)**

# Module 18

## 處理交易的目的和方法

- 在資訊系統中，交易是指由一系列資料庫操作組成的一個不可拆分的邏輯過程，並具有 **ACID** 的特性，若此交易過程中發生任何錯誤皆會導致交易失敗，此時資料庫必須恢復至交易前的狀態。

# Module 18

## 交易三模式

一般在資料庫中，交易可以分成3種模式來進行處理。

- 自動認可交易

預設的交易機制，執行一次SQL指令就變更一次資料庫中的資料。

- 外顯交易

明確的宣告交易的開始、提交、回滾。

- 隱含交易

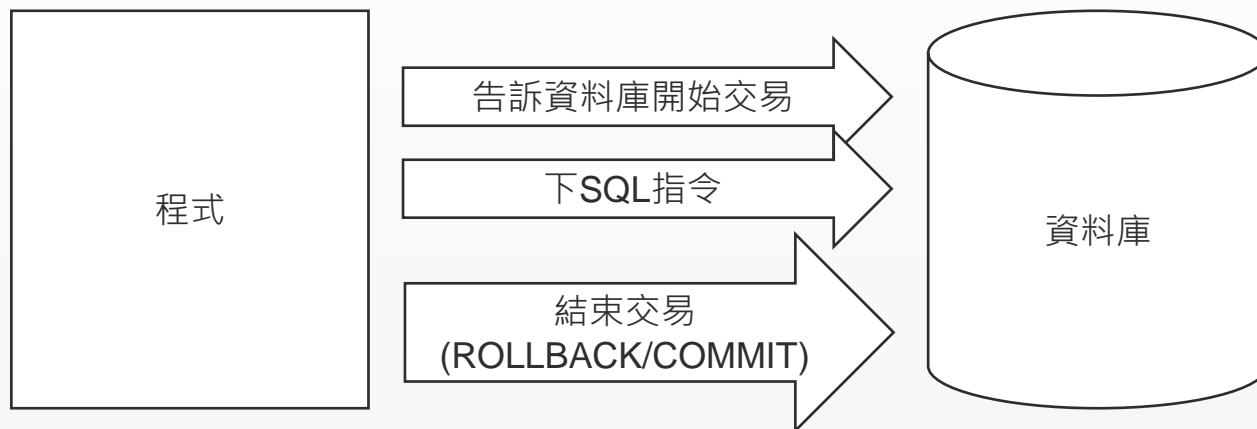
不明確宣告交易的開始，但必須執行提交來完成資料庫中資料的變更或是回滾不變更。



# Module 18

## 外顯交易

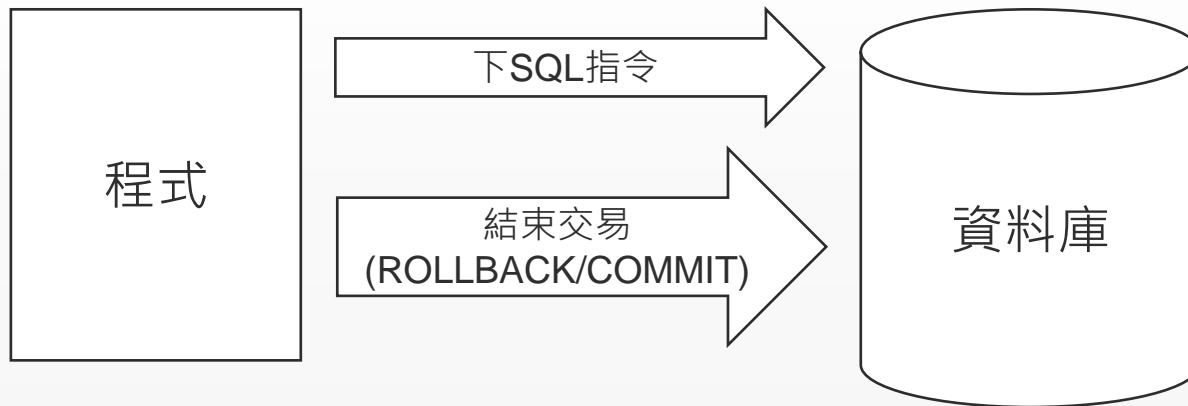
- 這個交易模式需要明確的宣告交易的開始，如MS SQL的BEGIN TRANSACTION 用來表示一筆交易的起始，而在結束時以 COMMIT 或 ROLLBACK 陳述式來明確結束一筆交易的完成或是對於交易失敗進行回復。
- 在這種交易模式下，需要所有的SQL指令成功時才會被寫進資料庫，否則ROLLBACK後將會全部復原。



# Module 18

## 隱含交易

- 透過設定將自動認可交易模式關閉後，所有交易將變為隱含交易，在這種交易模式下並不需要明示交易的啟動句，直接就是一筆交易的開始，然而還是需要COMMIT 或 ROLLBACK 陳述式來明確結束一筆交易的完成。
- 在上一筆交易完成後，會隱含的自動開啟一筆新的交易。





# Module 18

## 使用JDBC來完成交易

- 由於在 JDBC 的預設中，所採用的交易模式為自動認可交易 `auto-commit` 的模式，所以如果我們想要將一連串的 SQL 指令視為一筆交易的話，必須先將這個模式關閉，也就是切換成所謂的隱含交易模式。
- 交易模式的變更、處理是透過連線物件 `Connection` 來進行設置，我們可以使用
- `Connection` 物件下的 `setAutoCommit` 方法將其設置為 `false` 來啟動隱含交易模式，並使用 `commit()` 來確認交易。
- 如果發生異常，我們可以執行 `rollback()` 方法來將交易回滾，復原成原本狀態。

補充 1

Data Access Object  
(DAO)設計模式  
與 Java Bean

# 補充1

## Java Bean

- 在介紹 DAO 以前，先介紹 JavaBean
- JavaBean 為容器中的元件(Java 類別)，也稱為 Value Object，主要用來儲存系統中主要物件的屬性值，Java Bean 本身為標準的 Java 類別並具備一下特徵：
  1. 可以有多個建構子，但是一定要有不帶參數的建構子
  2. 不可有 public 修飾字的屬性變數(instance variable)。
  3. 提供 public 的 getter 跟 setter，讓其他物件存取。

# 補充1

## Data Access Object (DAO) 設計模式

- 將商業邏輯與資料庫操作邏輯分開編寫，將資料庫存取的程式封裝於 DAO 物件中，集中管理。
- 有效降低程式之間的耦合度，達到業務邏輯改變，只需改變業務邏輯的程式，不須改變資料庫存取的程式。
- 維護上更有彈性，如下圖，各司其職。



# 補充1

## 實作 DAO 設計模式步驟

- 將商業邏輯與資料庫操作分開
- 將資料庫操作的程式編寫在有命名DAO的類別
- 原本類別只剩商業邏輯。

# 補充 2 連線池

(Connection Pool)

## 補充2

# 連線池(Connection Pool)簡介

- 資料庫每次建立連線都會花費許多資源，尤其在網頁應用程式中，常常要透過資料庫讀或寫資料，像是登入、轉頁後拿資料、搜尋資料，若每次請求資訊都要重新建立連線，很耗費資源與時間。這時就必須使用連線池。
- 連線池會預先建立一些連線數量，儲存在連線池中，在程式需要連線的時候，`getConnection()` 其實是到池裡面去拿，不需要再花時間跟資料庫重新建立連線。



## 補充2

# 常用的連線池

- C3p0 連線池：老牌，不錯
- Tomcat 連線池：也不錯用
- HikariCP 光：目前火紅，Spring Boot 預設連線池  
<https://github.com/brettwooldridge/HikariCP>



## 補充2

# 連線池與DataSource

- 連線池通常設定在 DataSource 字樣的物件。
- 可透過該 DataSource 拿到連線。

```
HikariConfig config = new HikariConfig();

config.setJdbcUrl("jdbc:sqlserver://localhost:1433;databaseName=JDBCDemoDB");
config.setUsername("sa");
config.setPassword("passw0rd!");
config.setDriverClassName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

HikariDataSource ds = new HikariDataSource(config);
return ds;
```

## 補充2

# 連線池與DataSource

- 連線池通常設定在 DataSource 字樣的物件。
- 可透過該 DataSource 拿到連線。

```
Connection conn = HikariUtil.hiDataSoruce().getConnection;
```

就可以拿到該連線池的連線囉

**Thanks**