Java資料庫程式設計與應用

張春生

目錄 了解Java資料庫程式設計

目錄 學會如何在Java中存取資料庫

目錄 JDBC的進階應用

前導知識點

本課程需要技術

- Java SE 8 以上版本
- SQL 無論是MySQL或是MS SQL 等關聯式資料庫

學習目的

- 本課程目的是為了學習如何讓程式控制資料庫來執行資料庫的語句，來達成我們要將用戶、訂單等需要被儲存的資料，透過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。

JDBC API簡介

先了解什麼是API

- API:是Application Programming Interface的縮寫
  應用程式介面

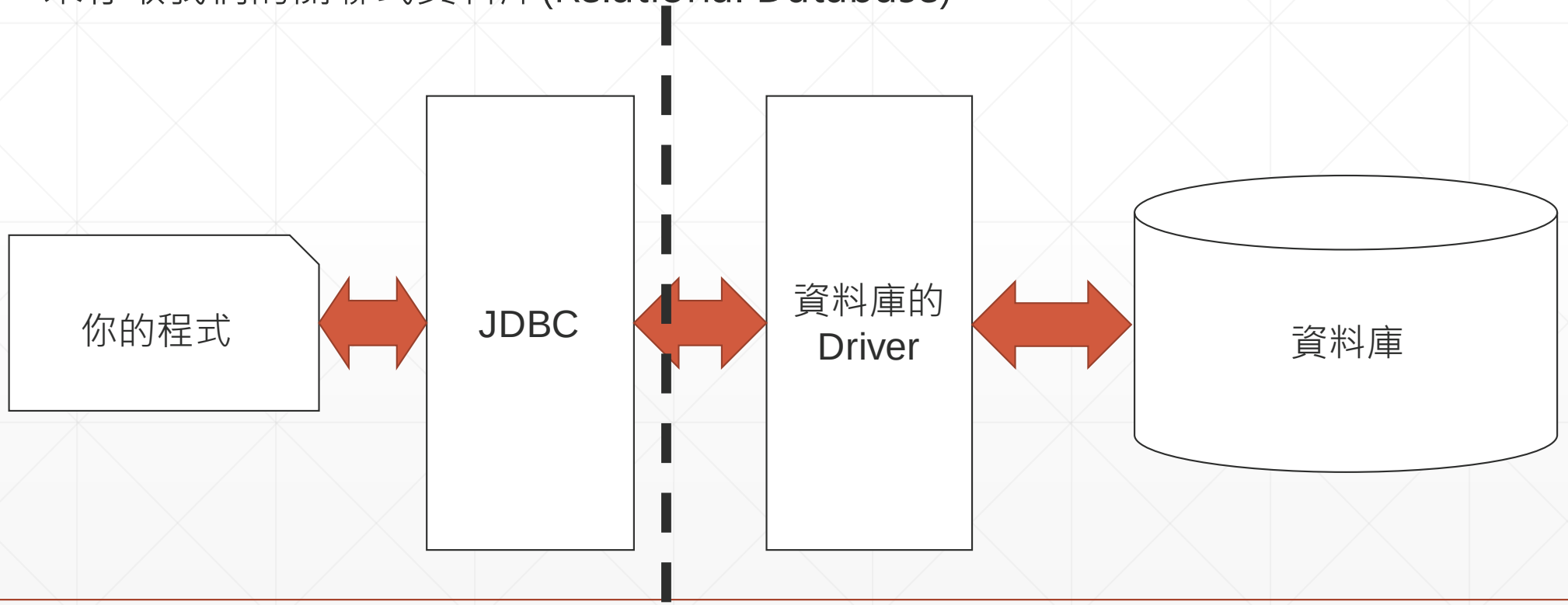
- 所謂的介面就是像USB接頭一樣的觀念
- 如：滑鼠、鍵盤.....

JDBC API

- 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
- 這個API已經包含在Java SE版本中，而我們所要做的事情，是取得、安裝對應的JDBC Driver

JDBC到底做什麼

- JDBC API他的作用是拿來連接Java與Database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)。



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)

JDBC Driver

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設計提供，我們不需要關心它是如何實作資料庫與我們的程式是如何進行溝通的，但我們必須要知道如何去使用我們的JDBC來設計資料庫相關的程式。

JDBC Driver的取得

- JDBC API彷彿就是一個插座一樣，一個插座在後方需要有一些提供電源的"東西"存在，這個東西在這裡就是JDBC Driver，我們可以從資料庫的廠商或相關的網站取得。
- 例如MS SQL：https://docs.microsoft.com/zh-tw/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-ver15
- 或是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

JDBC的部署

- 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。

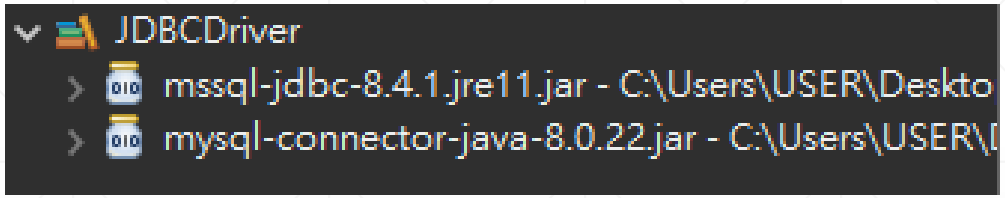
載入完成後會在專案中出現如圖所示的Library

JDBC的部署

- 有時，我們在一個專案中會用到不同的Driver或是套件，為了讓集中管理，我們會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，我們可以將這個程式庫直接部署到專案中，可以節省大量的時間。
- 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

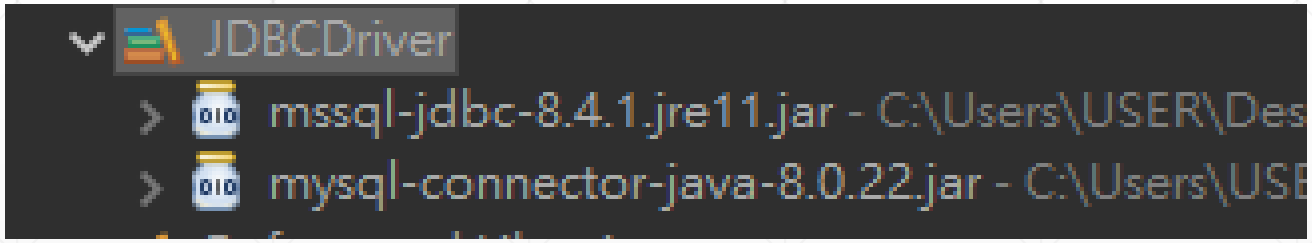
1. 選擇Window->Preferences
2. 在Preferences畫面下找到Java->Build Path->User Libraries
3. 選擇new 對Library進行命名
4. 選擇Add External JARs...
5. 選擇要使用的套件後 Apply and Close



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

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

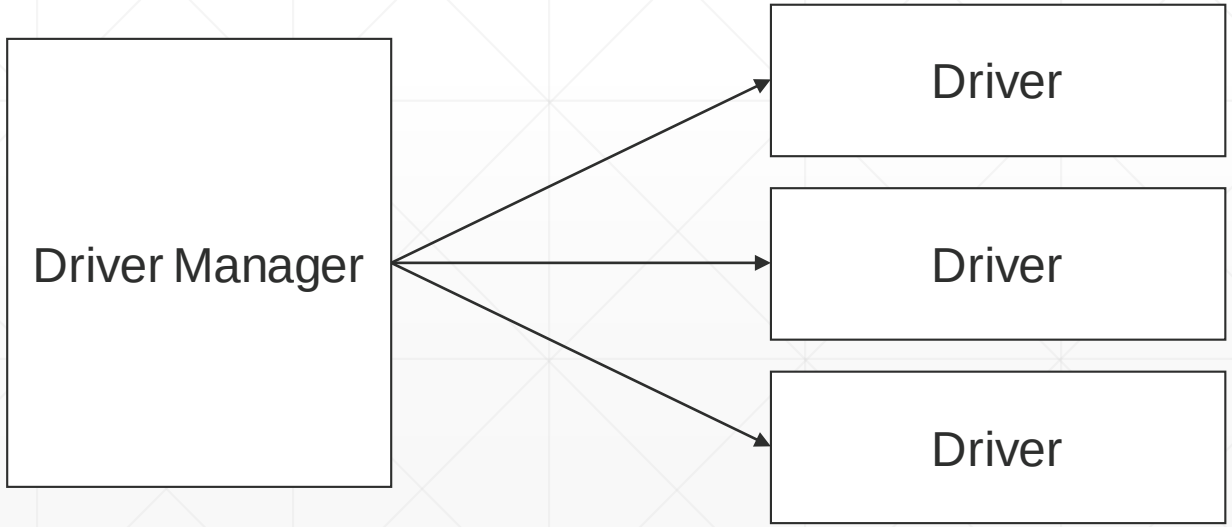


部署完成後會出現與當初命名一致的程式庫
裡面的檔案與程式庫中的檔案應一致

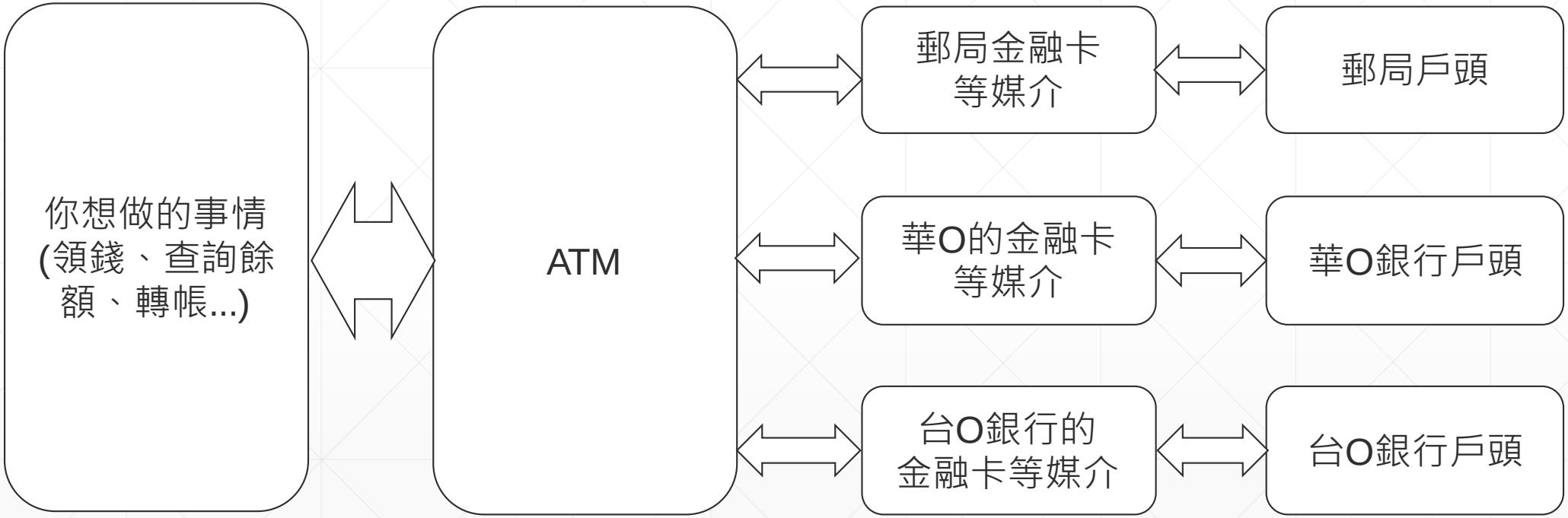
Driver Manager

Driver Manager

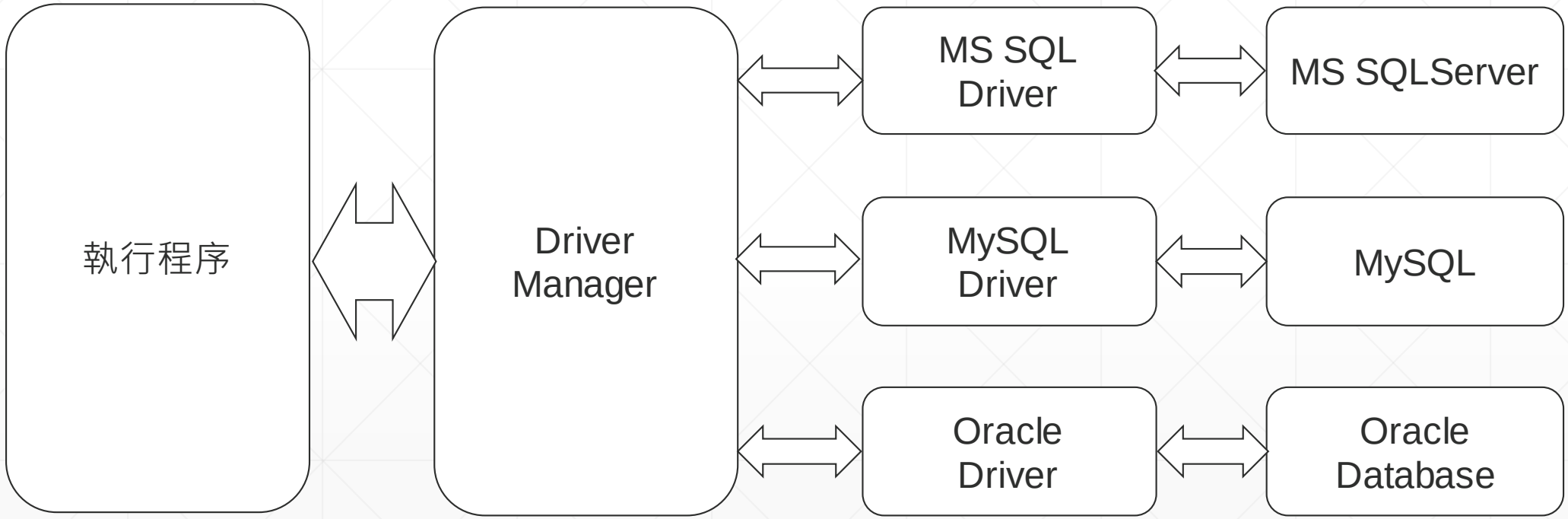
- Driver Manager的工作是負責處理Java應用程式與JDBC Driver之間的連接。
- 我們可以向程式(Driver Manager)註冊我們的Driver ，
- 一個Driver Manager可以讓多個Driver 給註冊。



Driver Manager的工作



Driver Manager的工作



第一隻Java資料庫程式

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)

為什麼要註冊

- 我們知道，要讓我們程式使用資料庫，必須要透過廠商所建立的套件，也就是Driver - 驅動程式。
- 但我們不是將驅動程式的套件放置在我們的專案中，就表示我們已經可以使用這個套件了，我們必須將這個套件進行註冊才能在程式中使用。
- Driver的註冊是使用我們的DriverManager進行註冊，註冊之後，DriverManager就可以對我們的JDBC Driver進行管理控制。

三種註冊、載入的方式

1. Class.forName方式
   - 傳遞Driver的名稱字串，使ClassLoader尋找驅動程式。
2. System.setProperty方式
   - 提供、設置jdbc.driver名稱的參數 - 驅動程式名稱
3. DriverManager.registerDriver方式
   - 建立驅動程式物件，直接註冊給DriverManager

Class.forName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、載入你的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，這個例外是需確認例外。
  - 程式片段：

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

System.setProperty方式

- 這種設置方式是將字串儲存到" jdbc.drivers"為鍵的鍵值儲存對中的值。
- 在DriverManager準備進行連接資料庫時，會去找尋鍵為jdbc.drivers的值，將其註冊到驅動管理列表中(DriverManager內建的一個List，用來儲存驅動程式)。
- 然而內部也是使用Class.forName方法來進行註冊。
- 優點：可以一次註冊多個驅動，中間用冒號進行分隔。
- 缺點：註冊時比較複雜。

System.setProperty程式片段

- System.setProperty("jdbc.drivers","com.microsoft.sqlserver.jdbc.SQLServerDriver:com.mysql.cj.jdbc.Driver");
- 這個範例示範註冊了2個JDBC Driver。

DriverManager.registerDriver

- 這方法可能會在較為老舊的專案中看到，因現在的JDBCDriver會進行自動註冊，所以不需要在程式中直接註冊，而是改為在驅動程式中或是在連接時自動載入註冊。
- 程式片段：DriverManager.registerDriver(new com.microsoft.sqlserver.jdbc.SQLServerDriver());
- 由於這個方法，會因為Driver中也對其進行實例化，故建立了2個驅動程式物件在程式中，導致效能降低，所以這種方式已逐漸被淘汰。

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。
- 這種載入方式無須顯式的去載入、註冊我們的JDBC Driver，可以直接的使用我們的驅動。
- 機制：
  - 在現在的jar包中，META-INF/services路徑下會有一個java.sql.Driver的檔案，它可以透過這個檔案去找到要來進行載入的Driver，故在這之後我們無需要顯式去進行載入、註冊的行為。

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)

JDBC的URL介紹

- JDBC的URL，是將連線時所需要的資訊寫成一個字串，又名連線字串。
- 所使用的JDBC Driver不同 ，所使用的連線字串就會有所不同。
- 連線字串的格式與範例：
  jdbc:sqlserver://localhost:1433;databaseName=jdbcDemoServer
  協定:子協定://資料來源
- 在JDBC中我們所使用的協定就是jdbc
- 而子協定通常是資料庫管理系統的名稱
  如:sqlserver,mysql,oracle……
- 而資料來源則是寫上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。

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=jdbcDemoServer
- 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:用來連線服務的號碼，類似於電腦的插孔一般，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。

URL範例

- MS SQL:
- jdbc:sqlserver://localhost:1433;databaseName=jdbcDemoServer
- MySQL:
- jdbc:mysql://localhost:3306/jdbcDemoServer
- 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/jdbcDemoServer?useSSL=false&serverTimezone=UTC&characterEncoding=UTF-8

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)

建立連線

- 在註冊完JDBCDriver後，我們就可以來去找我們的DriverManager來取得我們的連線物件了。
- 在取得連線物件時有幾個需要注意的重點，也是我們必須要的素材：

1. 連線字串URL
2. 資料庫的使用者名稱(user name)
3. 資料庫的使用者密碼(password)

取得連線getConnection

- 我們準備好剛剛所要求的3個元素之後，我們按照URL、帳號、密碼的順序將這三個字串提供給DriverManager的getConnection方法中
- 範例：
  - DriverManager.getConnection(“jdbc:sqlserver://localhost:1433;databaseName=jdbcDemoServer”,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。

資源釋放

- 對於資料庫相關的程式、我們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。
- 我們可以對Connection物件使用當中的close()方法，來將連線的資源進行歸還。
- 一般來說、為了確保我們的連線能正確關閉，我們會將close方法寫在finally區。
- 由於我們並不能確認每一次連線都絕對會成功，所以我們可以透過確認Connection物件是否為null以及使用isClosed方法來檢查是否有連線物件，以及連線物件是否存在。
- 如此一來，我們可以保證連線物件的存在以及是否被關閉。

程式片段

```
try {
    conn=DriverManager.getConnection("jdbc:sqlserver://localhost;database=JDBCDemo"
            ,"sa","P@ssw0rd");
    System.out.println("連線成功");
} catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}finally {

    try {
        if(conn!=null) {
            if(!conn.isClosed()) conn.close();
            System.out.println("連線已關閉");
        }
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

自動釋放資源

- 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代碼的長度。
- 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。
- 如此一來，我們便不需要再寫finally區塊，就能在執行完成或是發生例外時執行資源的釋放。

程式片段

- try(
- Connection conn=DriverManager.*getConnection("jdbc:sqlserver://localhost;database=JDBC Demo","sa","P@ssw0rd")*
-){
- System.***out.println("連線成功");***
- } catch (SQLException e) {
- e.printStackTrace();
- System.***out.println("連線失敗");***
- }

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)

Statement介面是什麼

- Statement介面是對資料庫進行訪問，使用的是靜態的SQL語句。
- 我們可以將SQL語法交給Statement物件，這個物件可以訪問資料庫後取得回傳的結果。
- 這邊所講的靜態SQL語句是指在傳遞後不可改變，只能決定執行或取消。
- 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。

使用Statement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

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getStatement方法取得
- 程式片段：
  - Statement stmt=conn.createStatement();
- 產生這個物件後根據要做的事情選擇對應的方法
  - executeQuery
  - executeUpdate
  - execute

executeQuery

- executeQuery這個方法只能使用在查詢時，不能更動資料表的內容時使用。
- 這個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

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");

execute

- 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boolean，如果有ResultSet回傳true否則回傳false。
- 程式片段：
  - boolean b=stmt.execute(sql);//執行sql語句
  - int row=stmt.getUpdateCount();//取得更新筆數
  - ResultSet rs=stmt.getResultSet();//取得結果集

ResultSet

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。

什麼是ResultSet

- 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

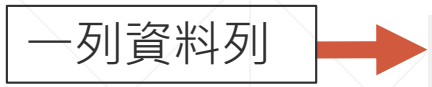
結果集示意圖

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()

cursor的控制

- 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



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

結果集示意圖

next()	下一筆
previous()	前一筆
first()	第一筆
last()	最後一筆
beforeFirst()	到第一筆之前
afterLast()	最後一筆之後

一般來說，資料庫在預設狀態下僅能向後讀取
除非在取得Statement物件先告知資料庫要使用
什麼樣子的讀取方式。

欄位讀取

- 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方法來只到第一筆資料。
- 當ResultSet用cursor選擇到資料後，要使用get+資料型別來取得資料。
- 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)
- 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()

程式片段

```
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));

}
```

產生動態SQL指令-PreparedStatement

什麼是PreparedStatement

- PreparedStatement繼承了Statement的介面，他除了可以處理靜態的SQL語句之外，我們還可以用來處理"動態"的SQL語句。
- 此物件與Statement一樣 ，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，我們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。

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特殊符號(?)，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。

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?，另一種是傳入值。
- 而後面的傳入值要傳入相對應的物件。
- 程式片段：
- stmt.setString(1, "1");

PreparedStatement的執行SQL指令

- 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

(1 個資料列受到影響)

動態SQL指令的運用

查詢SQL指令範例

```
PreparedStatement pstmt = conn.prepareStatement(
“SELECT  [Customerid],[CusName],[CusAddress] ,[CusTel]"
        + “FROM   [JDBCDemo].[dbo].[sampletable]"
          + "where Customerid=?");
pstmt.setInt(1, Integer.valueOf(Customerid));
ResultSet rs = pstmt.executeQuery();
```

新增Insert的程式片段

```
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO [dbo].[sampletable]"
 +"([CusName],[CusAddress],[CusTel]) VALUES (?,?,?)");
pstmt.setString(1, name);
pstmt.setString(2, address);
pstmt.setString(3, tel);
pstmt.executeUpdate();
System.out.println("新增成功");
pstmt.close();
```

更新 Update

```
PreparedStatement pstmt = conn.prepareStatement("UPDATE [dbo].[sampletable]"
                + "   SET [CusName] = ?,[CusAddress] =?,[CusTel] =?"
                + " WHERE [Customerid]=?");
pstmt.setString(2, address);
pstmt.setString(1, name);
pstmt.setString(3, tel);
pstmt.setInt(4, id);
pstmt.executeUpdate();
System.out.println("更新成功");
pstmt.close();
```

删除Delete

```
PreparedStatement pstmt = conn.prepareStatement("DELETE FROM [dbo].[sampletable]"
            + "     WHERE [Customerid]=? and [CusName]=?");
pstmt.setInt(1, id);
pstmt.setString(2, name);
pstmt.executeUpdate();
System.out.println("删除成功");
pstmt.close();
```

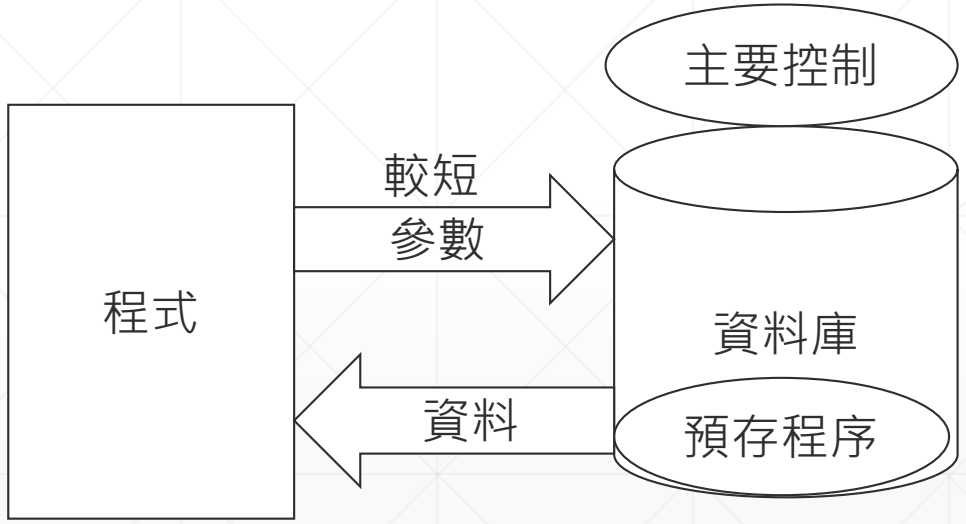
呼叫資料庫的預存程序 -- CallableStatement

預存程序

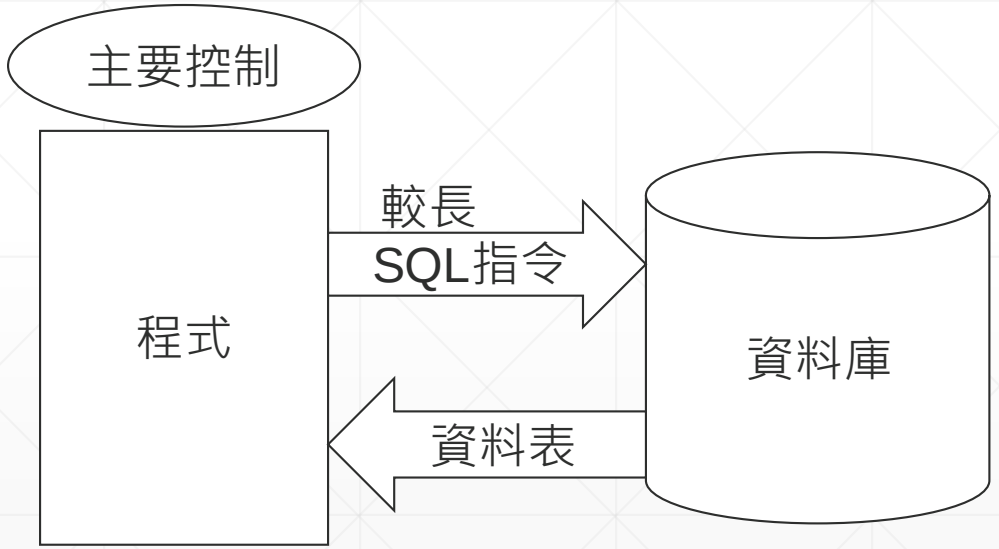
- 預存程序是我們將寫好的商業邏輯的部分，以事先編譯的方式儲存在資料庫中，我們可以藉由呼叫該預存程序並提供參數給該程序，以讓預存程序完成我們的商業邏輯。
- 優點：
  - 預存程序在編輯時就能進行除錯。
  - 預存程序因需要傳遞的物件縮小，且已在資料庫進行編譯，效率大幅上升。
- 缺點：
  - 預存程序是建立在資料庫上，所以要切換到其他系統時，會因為使用的語法有所不同，而需要重寫原有的預存程序。
  - 預存程序的效能調整、編寫預存程序會被受限於資料庫的系統。

CallableStatement與PreparedStatement比較

CallableStatement



PreparedStatement



執行步驟

- 前提：資料庫需要有預存程序

1. 從連線物件取得CallableStatement
2. 取得連線時要傳入呼叫預存程序的指令。
3. 設置、註冊參數與資料型別
4. 執行
5. 取出回傳值
6. 釋放資源

示範用預存程序

```
CREATE PROCEDURE dbo.selectId
        @Id int ,
        @Name nvarchar(50) out,
        @Address nvarchar(100) out
AS
        SELECT @Name=CusName,@Address=CusAddress from
dbo.sampletable where Customerid=@Id
GO
```

預存程序如何呼叫

- 我們在使用JDBC呼叫MS SQL的預存程序時使用的的字串是：
- {call 預存程序名稱(參數,參數...)}
- 在這邊使用的參數使用？來進行動態的方式來做為參數的輸入輸出。

預存程序的輸入方式與
PreparedStatement相同
使用set的方法來做為設置。

預存程序的輸出
使用regesterOutParameter方法來作為輸出的設置，一樣需要按照參數的順序設置。
待資料庫將預存程序將結果傳回時，會去比對是否符合該資料型別。
資料型別可以使用java.sql的Types中來去選擇。

注意事項

- 因為呼叫預存程序是呼叫儲存在SQL中的程式，所以呼叫的格式會因為不同的關聯式資料庫系統不同而有所不同。
- 舉例：
- MS SQL：{call 預存程序名稱(參數,參數...)}
- MySQL：call 預存程序名稱(參數,參數...)
- 執行完預存程序後，可以使用get資料型別的方法加上回傳值的名稱、或是索引值來取得回傳的資料。

程式片段

```
CallableStatement cstmt=conn.prepareCall("{call selectId(?,?,?)}");


cstmt.setInt(1,1);
cstmt.registerOutParameter(2, Types.NVARCHAR);
cstmt.registerOutParameter(3, Types.NVARCHAR);
cstmt.execute();
System.out.println("NAME="+cstmt.getString(2)+",Address="+cstmt.getString(3));
cstmt.close();
```

Blob & Clob介紹

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。

對應資料庫的格式

- 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，但這些遠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使用了varbinary(max)、image來進行儲存BLOB
- 使用text、varchar(max)來儲存CLOB

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。

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

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。

圖形資料處理

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)

將圖形檔案寫入資料庫程式片段

```
FileInputStream fis=new FileInputStream("fileFolder/a01.jpg");

PreparedStatement pstmt=conn.prepareStatement(“INSERT INTO
[dbo].[MyFile_Table] ([FileName],[File])  VALUES (?,?)");

pstmt.setString(1,"a01");

pstmt.setBinaryStream(2, fis);

pstmt.execute();
```

圖形資料處理 - 讀取輸出

- 讀取輸出時由於是從資料庫->程式中，所以得到的是inputStream的物件。

1. 執行查詢指令、取得結果集
2. 取得結果集，從結果集得到串流(InputStream)
3. 將串流輸入的資料輸出成檔案
4. 取得下一個結果集，重複輸出直到結束。
5. 釋放資源

將圖形檔案寫出成檔案程式片段

```
PreparedStatement pstmt2=conn.prepareStatement("SELECT [File]“
                        + "  FROM [JDBCDemo].[dbo].[MyFile_Table]");
ResultSet rs=pstmt2.executeQuery();
while(rs.next()) {
InputStream is=rs.getBinaryStream(1);
 byte[] b=is.readAllBytes();
FileOutputStream fout=new FileOutputStream("OutFolder/a02.jpg");
fout.write(b);fout.flush();fout.close();
}
```

使用BLOB介面寫出成檔案程式片段

```
PreparedStatement pstmt2=conn.prepareStatement("SELECT [FileName],[File]"
+ "  FROM [JDBCDemo].[dbo].[MyFile_Table]");
ResultSet rs=pstmt2.executeQuery();
while(rs.next()) {
String s=rs.getString(1);
Blob fiBlob=rs.getBlob(2);
byte[] b2=fiBlob.getBytes(1,(int) fiBlob.length());
FileOutputStream fout2=new FileOutputStream("OutFolder/a03.jpg");
fout2.write(b2);fout2.flush();fout2.close();
}
```

MetaData

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。
- 就像是這個投影片本身的內容就是資料，
- 而右方這些就是他的MetaData。
- 而JDBC的MetaData 主要是希望能透過
- Java程式來取得資料庫的系統資訊。
- 而JDBC提供下面兩種MetaData。
- DatabaseMetaData (資料庫)
- ResultSetMetaData (資料表、結果集)

DatabaseMetaData

下面是常用的方法以及得到的資料，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

程式片段示範

- DatabaseMetaData dbmd=conn.getMetaData();
- StringBuilder sb=new StringBuilder();
- sb.append("資料庫名稱："+dbmd.getDatabaseProductName());
- sb.append("\n資料庫版本："+dbmd.getDatabaseProductVersion());
- sb.append("\n驅動程式名稱"+dbmd.getDriverName());
- sb.append("\n驅動版本："+dbmd.getDriverVersion());
- sb.append("\nDBMS的URL："+dbmd.getURL());
- sb.append("\n使用者名稱："+dbmd.getUserName());
- System.out.println(sb.toString());

ResultSetMetaData

ResultSetMetaData

- ResultSet 結果集中，不僅僅只有我們查詢的資料而已，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
getColumnLable(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)

程式片段

```
ResultSet rs = pstmt.executeQuery();
ResultSetMetaData rsmd = rs.getMetaData();
System.out.println("getColumnCount:" + rsmd.getColumnCount());
for (int i = 1; i <= rsmd.getColumnCount(); i++) {
        System.out.println("getColumnName:" + rsmd.getColumnName(i));
        System.out.println("getColumnLabel:" + rsmd.getColumnLabel(i));
        System.out.println("getColumnType:" + rsmd.getColumnTypeName(i));
        System.out.println("getColumnDisplaySize:" + rsmd.getColumnDisplaySize(i));
        System.out.println("isNullable:" + rsmd.isNullable(i));
}
```

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。

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。

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。

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。
- 我們可以使用這個方法，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。

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。

異常處理

異常處理

- 當我們在傳遞錯誤格式的資料的時候，資料庫會回傳錯誤的訊息，這個例外我們應當要進行處理，不進行處理就代表著這個程式的結束。
- 異常處理要做的事情：
  - 回饋給使用者錯誤的輸入情形。
  - 防止異常造成程式的崩潰。
  - 了解程式發生的錯誤。
- 主要要處理的例外是SQLException。

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，我們得到這些代碼擊訊息之後就能夠有效地排除錯誤。

錯誤的處理方式

- catch (SQLException e) {
- this.closeConn();
- System.out.println("SQL查詢失敗");
- System.out.println("Message:"+e.getMessage());
- System.out.println("ErrorCode:"+e.getErrorCode());
- e.printStackTrace();
- }

批次更新
(batch Updates)

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。
- 批次處理的目的是：
- 可以把不需操作者干預的作業處理轉移到電腦資源不太繁忙的時段(如深夜)。
- 避免計算資源閒置，使昂貴的資源保持一定的使用率，以減低平均開銷。
- 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、不需要人力進行輸入的作業，這一類型的工作就很適合進行所謂的批次處理。

addBatch()

- 我們可以使用Statement或是PrepareStatement這兩個介面的物件中這個方法，來將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。
- PrepareStatement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。

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，會將放置進List物件中的SQL指令清空。

程式片段

```
PreparedStatement pstmtBatch=conn.prepareStatement(
"INSERT INTO [dbo].[sampletable]" + "([CusName],[CusAddress],[CusTel]) VALUES (?,?,?)");
for (CustomerBean customerBean : Customers) {
pstmtBatch.setString(1, customerBean.getCusName());
pstmtBatch.setString(2, customerBean.getCusAddress());
pstmtBatch.setString(3, customerBean.getCusTel());
pstmtBatch.addBatch();
}
pstmtBatch.executeBatch();
pstmtBatch.close();
```

交易
(Transaction)

何謂交易

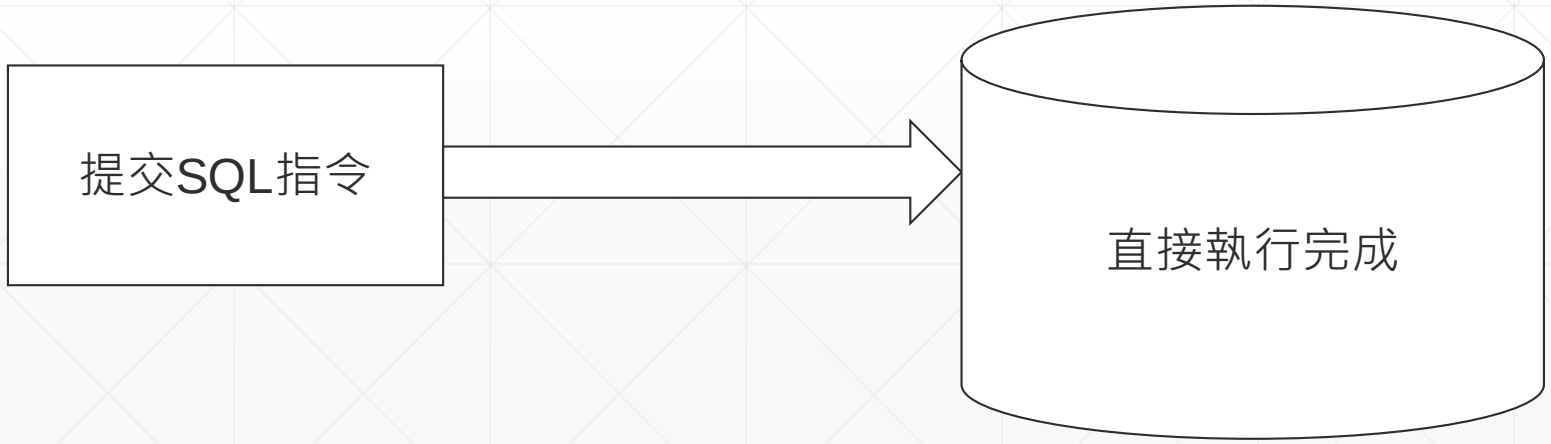
- 所謂的交易是指一連串不可分開執行的資料庫指令。
- 例如：銀行的轉帳，需要將錢從A帳戶移出，然後把移出的錢存入B的帳戶中。
- 在這個行為中，A、B的帳戶皆需要對狀態進行修改，如果再指修改的一方的過程中發生事故、問題時，如果沒有透過一些方法處理則有可能造成帳目錯誤等嚴重的問題，所以我們需要對交易進行控制、管理。

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。

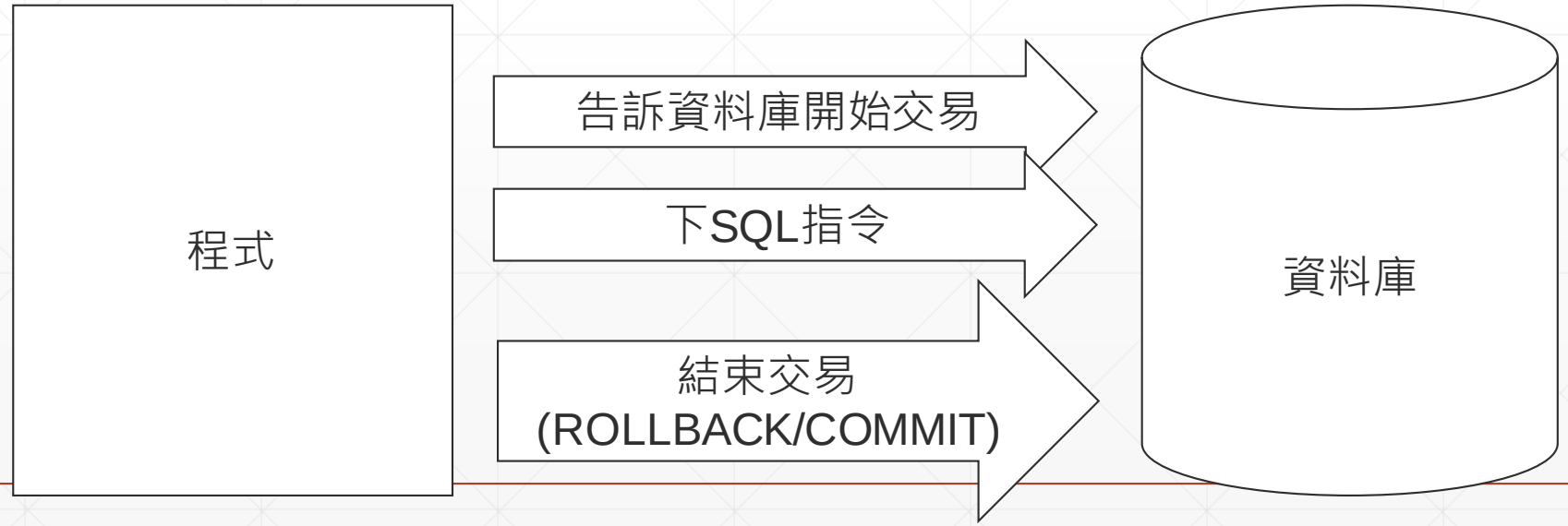
自動認可交易

- 此種模式是在JDBC中被默認的一種交易模式，在這種模式下SQL的單一語句都會被是唯一筆交易，當交易成功時就會自動確認(Commit)，失敗則Rollback復原回原本的資料。
- 這種模式適合處理僅需要單句SQL指令就能完整敘述的交易，如更新個人資料等。



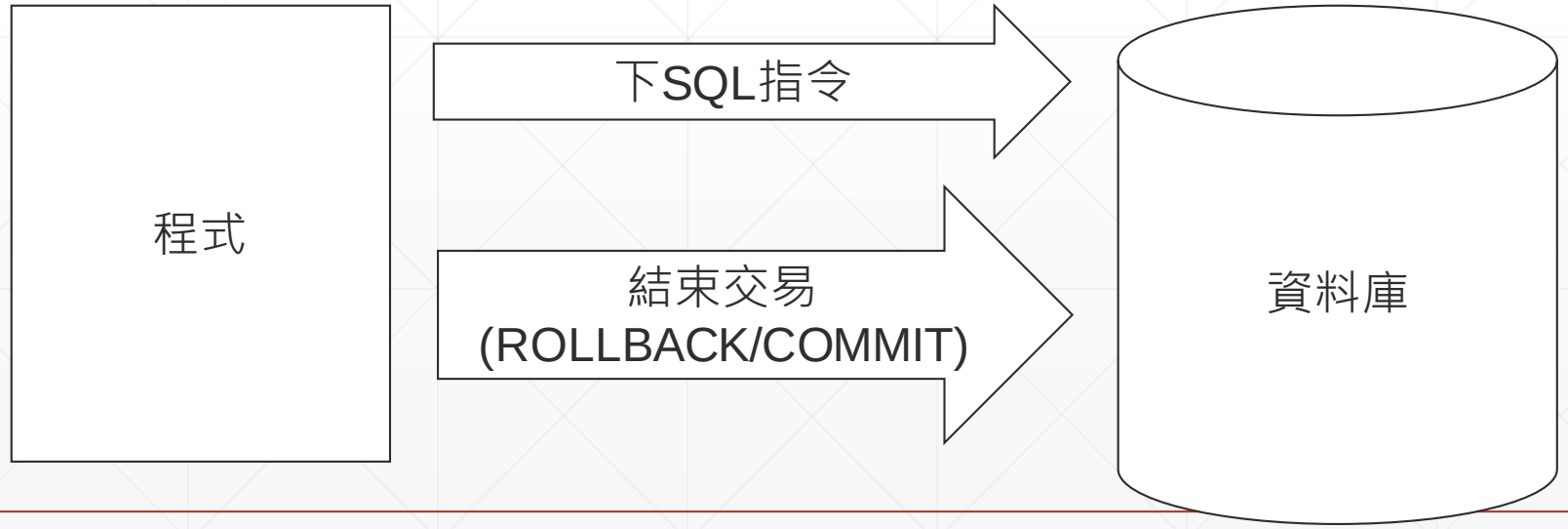
外顯交易

- 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。



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。



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。
- 完成後使用setAutoCommit方法，將AutoCommit設置為true，將自動認可交易重新開啟。

程式片段

```
conn.setAutoCommit(false);//關閉自動交易

PreparedStatement pstmt = conn.prepareStatement("UPDATE [dbo].[sampletable]"
+ "   SET [CusTel] =?" + " WHERE [Customerid]=?");

pstmt.setString(1,telId1);  pstmt.setInt(2, id2);//更新第一筆

pstmt.execute();

pstmt.setString(1,telId2);  pstmt.setInt(2, id1);//更新第二筆

pstmt.execute();

conn.commit(); //提交

conn.setAutoCommit(true);//重新開啟自動交易
```