**Transaction**[#](http://ws93006ed:8080/JSPWiki/Wiki.jsp?page=CM_Freshman_Develop_norm#section-CM_Freshman_Develop_norm-Transaction)

**Transaction使用說明**[#](http://ws93006ed:8080/JSPWiki/Wiki.jsp?page=CM_Freshman_Develop_norm" \l "section-CM_Freshman_Develop_norm-Transaction_E4_BD_BF_E7_94_A8_E8_AA_AA_E6_98_8E)

* **Transaction.begin() 、 Transaction.commit() 及 Transaction.rollback()**
  + 用 Transaction.begin() 來宣告交易起始，並用 Transaction.commit() 來結束交易，有任何錯誤則以 Transaction.rollback() 來將所有變的資料更回覆為變更前的狀態。
  + 在 Transaction.begin() 至 Transaction.commit() 之間，若透過 Transaction.getDataSet() 取得 DataSet 會操作在同一個 JDBC connection。
    - 反之在 Transaction.begin() 至 Transaction.commit() 之外，即使以 Transaction.getDataSet() 取得 DataSet ，所取得的 DataSet 也不會操作在同一個 JDBC connection 上。
    - 所以簡單的說，Transaction.begin() 至 Transaction.commit() 之間的所有對 DB 的操作，視為一個交易的【作業單元】。
  + 查詢或其他相關判斷若在非必要時，需在控制 Transaction 前進行，以避免 Dead lock 或 Long Transaction 的情況發生。

* **Transaction 區間宣告範例**

import com.cathay.util.Transaction;

import com.igsapp.db.DataSet;

public class TransactionDemo {

public void doUpdate(String arg1, String arg2) throws ModuleException {

Transaction.begin();

try{

DataSet ds = Transaction.getDataSet();

ds.clear();

ds.setField("COL1", arg1);

ds.setField("COL2", arg2);

DBUtil.executeUpdate(ds, "SQL\_KEY\_UPDATE\_1");

DBUtil.executeUpdate(ds, "SQL\_KEY\_UPDATE\_2");

Transaction.commit();

}catch(Exception e){

Transaction.rollback();

throws e;

}

}

}

**Transaction 與 DataSet 關係**[#](http://ws93006ed:8080/JSPWiki/Wiki.jsp?page=CM_Freshman_Develop_norm" \l "section-CM_Freshman_Develop_norm-Transaction_E8_88_87DataSet_E9_97_9C_E4_BF_82)

使用 DataSet 跨子系統存取 DB 時，會牽扯到 DB 使用者權限問題。目前 DB2 上的設定為每個子系統會建立各自的 DB access 的使用者帳號，預設僅能存取該子系統所屬的 schema 下的 table，包含增刪查改。若需要從子系統A 對子系統B 的 table 進行增刪查改時，便要看是以何種使用者的身分去對子系統B 的 table 進行存取，以及該 table是否有開放權限給子系統A 的使用者使用。以下為模擬狀況:

* **模擬狀況說明**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DataSource連線** | **DS對應使用者帳號** | **欲存取的子系統TABLE** | **結果** | **說明** |
| DS\_RC | dbrc | DBRC.DTRCG001 | OK | 由於屬於同一個子系統下，所以使用者DBRC，擁有對該表格的增刪查改權限 |
| DS\_RC | dbrc | DBRJ.DTRJB002 | ? | 由於 DTRJB002 屬於 RJ 子系統，因此要看DTRJB002是否有開放權限給DBRC進行存取。 |

*註:關於開放 table 權限給其他子系統使用的說明，請參考*[*此文*](http://ws93006ed:8080/JSPWiki/Wiki.jsp?page=PG_EditTablePriviledge)

從模擬狀況可以看出 DataSource 的連線即決定以何種 DB 使用者帳號去存取 table，而 DataSource 的連線又與 DataSet 的取得方式及是否使用 Transaction 有關係。歸納出來，主要分為以下兩種狀況：

**1. 以 DBModule 的「getDataSet()」取得獨立連線DS**

DataSet ds = getDataSet();    //直接呼叫父類別(DBModule) 取得DataSet

* **說明**

若模組繼承 DBModule，再透過 DBModule 的 getDataSet() 取得的獨立的 DataSet 進行連線，此時無論是否有用 Transaction 宣告交易起始，只要跨子系統存取，便會改用該子系統對應的 DataSource 進行連線。

* **優點**

只要跨子系統存取，便會改用該子系統對應的DataSource進行連線，因此存取上較無權限問題。

* **缺點**

a. 造成交易的不一致，由於即使有用 Transaction 宣告交易起始，仍會以該子系統對應的 DataSource 進行連線。因此，若交易中途發生錯誤，便有可能發生 rollback 不完全的狀況。甚至是正常執行完成，進行 commit 時，仍會因跨子系統，而只能 commit 部分資料的狀況。以下圖為例，若程式執行到 RJ 時發生錯誤需要 rollback ，但因目前的 DataSource 為 DS\_RJ，因此只能對 RJ 的資料進行 rollback ， RC 的資料便無法 rollback 。同理，若程式執行至 RZ 時完成，但由於目前的 DataSource 為 DS\_RZ，因此便無法正常對 RJ 及 RC 的資料進行 commit。

b. 因為採用的是獨立連線，所以若每支模組都是用這種方式，會占用 DB 過多連線數目。

c. 造成 Dead Lock (死結)。由於交易的不一致，若在同一個交易中，同時有兩條連線，先後對同一個 table 存取，便會造成死結。舉例來說，若有一主程式，裡面宣告交易起始，先後呼叫兩個模組進行資料存取，而兩模組取得的 DataSet 不同：

//\*\*模組A\*\*//  
DataSet dsA = Transaction.getDataSet();      
  
//\*\*模組B\*\*//  
DataSet dsB = getDataSet();  
  
  
//主程式  
Transaction.begin();    //交易開始  
  
//1. 呼叫模組A 對 DBRC.DTRCG001 進行Update  
moduleA.updateG001();  
  
//2. 呼叫模組B 對 DBRC.DTRCG001 進行讀取  
// 死結發生! 因為模組A 是透過 Transaction 取得 DataSet ，  
// 此時模組B 採獨立連線的 DataSet 來進行讀取，但因為交易還未  
// commit ，所以 DTRCG001 仍然被 Lock 住，於是模組B 等不  
// 到模組A 釋放 DTRCG001 ，造成死結。  
moduleB.queryG001();  
  
  
Transaction.commit();    //交易結束

* **示意圖**

|  |
| --- |
| http://ws93006ed:8080/JSPWiki/attach/CM_Freshman_Develop_norm/Transaction_2010-07-26_230000.jpg |

**2. 透過 Transaction 取得「Transaction.getDataSet()」共用連線DS**

DataSet ds = Transaction.getDataSet();    // 透過 Tranaction 物件取得DataSet

* **說明**

透過 Tranaction 物件取得的共用 DataSet 進行連線，若有用 Tranaction 宣告交易起始，當跨子系統存取時，仍然會使用原子系統的 DataSource 進行。

* **優點**

交易上具有一致性，能確保 commit 、 rollback 能正常運作。且無 Dead Lock (死結) 及連線數過多的情況。

* **缺點**

跨子系統時，存取上會有權限問題。

* **示意圖**

|  |
| --- |
| http://ws93006ed:8080/JSPWiki/attach/CM_Freshman_Develop_norm/Transaction_2010-07-26_230001.jpg |

**結論**

為了確保主程式呼叫各模組進行交易時功能能夠正常，因此在模組取得DataSet時，

若無特別需要，請一律使用 Transaction.getDataSet(); 的方式!

**J2EE Transaction 的 ACID 原則**[#](http://ws93006ed:8080/JSPWiki/Wiki.jsp?page=CM_Freshman_Develop_norm" \l "section-CM_Freshman_Develop_norm-J2EETransaction_E7_9A_84ACID_E5_8E_9F_E5_89_87)

* **交易是什麼**
  + 交易通常指的是雙方之間的某種交換，如付錢在 7-11 買東西，你在付錢後，在沒找錢之前是不會離開的。在這個交換的過程中，參與的雙方都需確保最後的結果是可以滿足的。店員在你未付錢之前不會讓你走出商店，你在店員未找錢之前也不會離開。
  + 對應用程式而言，交易通常是指一組工作，或說是『作業單元』(unit-of-work)。在這組工作中，每個動作彼此相關，而且必須一起完成。為確保應用程式交易的可靠性，基本而言，交易的動作必需遵循四項原則。
* **ACID 原則**
  + atomic(不可分割性)
    - 交易必需完整地執行完畢，否則就不執行。簡單的說，一個『作業單元』內的動作必需全部正確的執行，有任何的錯誤，就放棄整個『作業單元』，並復原之前變更過的資料。全部成功，或全部失敗，沒有第三種狀況
  + consistent(一致性)
    - 一致性指的是完整基本資料儲存，以確保『作業單元』和企業邏輯不會不一致。此交易特性和實際的商業邏輯有一定的相關，通常實務的商業邏輯會決定什麼樣的資料是完整的，我們也經常使用資料庫的 schema 來約制被儲存下來的資料是合乎限制的(如 primary key、foreign key .....)。以實際的例子來說明，要保書主檔資料中的要保人，一定會參考到客戶資料檔中的某個客戶資料，否則就違反了一致性的原則。
  + isolated(絕緣性)
    - 在進行交易的過程中，應避免其它程式或交易干擾。也就是說在目前進行中的交易完成之前，其它的程式或交易，應該避免存取目前交易正在使用中的資料。
  + durable(恆久性)
    - 恆久性指的是，交易過程式所做的任何改變，必須在交易結束前儲存至某種儲存實體中(如資料庫或是檔案)。如此才可以確保當將來系統出現問題時，這些改變的結果不會流失。
* **參考書籍**
  + Enterprise JavaBeans 技術，第三版

|  |
| --- |
| http://ws93006ed:8080/JSPWiki/attach/CM_Freshman_Develop_norm/Transaction_2010-07-26_230002.gif |

**two phase commit 簡易說明**[#](http://ws93006ed:8080/JSPWiki/Wiki.jsp?page=CM_Freshman_Develop_norm" \l "section-CM_Freshman_Develop_norm-TwoPhaseCommit_E7_B0_A1_E6_98_93_E8_AA_AA_E6_98_8E)

Transaction.setXAMode();  
Transaction.begin();

* **說明**
  + 透過 Tranaction 物件設定 two phase commit 機制，需在 Tranaction 宣告交易起始前呼叫所提供 static method「setXAMode」，以達到針對不同資料庫可以同時進行 commit 或 rollback。
  + 當啟動此機制時，則會使用 IBM DB2 所提供 Driver「com.ibm.db2.jcc.DB2XADataSource」進行連線（預設連線使用的 Driver 為com.ibm.db2.jcc.DB2ConnectionPoolDataSource），此時 Application Server 會透過自身的 Transaction Coordinator 與 DataBase 的 Transaction Manager 溝通，確認進行交易是否可正常處理後，再同時發出 commit 或 rollback。
  + 目前公司所使用的 Application Server 為 WebSphere、DataBase 為 DB2。
  + 此機制需看伺服器及資料庫是否有提供支援，並非所有皆有支援。

* **優點**
  + 針對多個資料庫在交易上可保有一致性，能確保 commit 、 rollback 能同時運作。

* **缺點**
  + 較單一資料庫存取（single phase commit）的處理效能較差。
  + 若所使用 SQL 內有 join 的動作，當資料庫不為同一個時，則會無法操作而產生錯誤。

* **示意圖**

|  |
| --- |
| http://ws93006ed:8080/JSPWiki/attach/CM_Freshman_Develop_norm/twoPhaseCommit_01.png |

|  |
| --- |
| http://ws93006ed:8080/JSPWiki/attach/CM_Freshman_Develop_norm/twoPhaseCommit_02.png |

|  |
| --- |
| http://ws93006ed:8080/JSPWiki/attach/CM_Freshman_Develop_norm/twoPhaseCommit_03.png |