12

MIDP 資料庫程式設計

沒經過處理的資料當然叫做垃圾，但是也不見得經過處理的資料就是資訊。

- 前言
- 紀錄管理系統概觀
- 紀錄倉儲的使用
- 資料儲存
- 監視紀錄倉儲的變化
- 走訪紀錄倉儲
- 多緒執行的相關議題
- 實際範例：通訊錄
- 總結

前　言

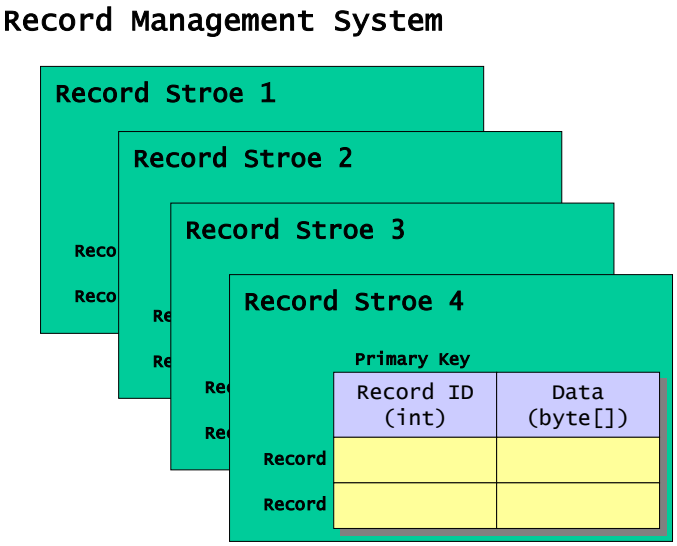
不管是開發什麼類型的程式，資料庫永遠是一個最重要的議題。雖然在 Java 2 Standard/Enterprise Edition 之中已經有 JDBC 這個技術，可是由於 JDBC 是針對桌上平台或企業用戶所設計，並不適合一般記憶體寸土寸金的行動通訊裝置。

因此在 MIDP 之中，提供了所謂的紀錄管理系統（Record Management System, RMS)，這個紀錄管理系統就是一個小型簡單的資料庫管理系統（Database Management System)。我們除了能用紀錄管理系統儲存資料之外，也可以將任何物件的狀態紀錄起來，儲存到紀錄管理系統之中以達成永續儲存（Persistent Storage）的功能。

本章將介紹紀錄管理系統的使用方式，並在最後以一個通訊錄程式做結尾。

紀錄管理系統概觀

前面我們曾提到，紀錄管理系統就是一個小型的資料庫管理。在一般的資料庫管理系統裡頭，存放著許多的表格（Table)，而在紀錄管理系統中，和表格扮演央同角色的物件就叫做紀錄倉儲（Record Store)。一般的表格之中存放著許多筆記錄（Record)，在紀錄倉儲之中的每一筆資料我們也以紀錄稱呼他們。這些名稱之間的關係可以用下圖表示：



所有和紀錄管理系統相關的類別都放置於 javax..microedition.rms 這個套件之中。javax..microedition.rms 套件之中共包含:

1. 兩個類別：RecordStore、RecordEnumeration。
2. 三個介面：RecordComparator、RecordFilter、RecordListener。
3. 五種例外：InvalidRecordException、RecordStoreException、RecordStroeFullException、RecordStroeNotFoundException、RecordStoreNotOpenException。

紀錄倉儲的使用

紀錄倉儲的角色如同一般關聯式資料庫之中所謂的表格一樣。在每一個 MIDlet Suite 之中，每一個資料倉儲都有它獨一無二的名字，大小不能超過 32 個 Unicode 字元，且大小寫有差別。因此同一

個 MIDlet Suite 之中的 MIDlet 都可以共享這些紀錄倉儲，而不同 MIDlet Suite 之間無法共享紀錄倉儲。

要使用紀錄倉儲之前，要先開啟它，我們使用的 RecordStore 類別的 openRecordStore()函式：

```
RecordStore.openRecordStore(紀錄倉儲的名稱,true) ;
```

或

```
RecordStore.openRecordStore(紀錄倉儲的名稱,false) ;
```

第二個參數如果傳入的是 true，代表如果我們所要開啟的資料倉儲不存在，請系統幫我們建立一個新的資料倉儲；反之，如果傳入的是 false，代表如果我們所要開啟的資料倉儲不存在，就丟出 RecordStoreNotFoundException 例外。

一般來說，為了方便起見，我們會撰寫工具函式如下：

如果不管如何您都要開啟紀錄倉儲，函式如下：

```
public RecordStore openRSAnyway(String rsname)
{
    RecordStore rs = null ;
    //名稱大於 32 個字元就不接受
    if(rsname.length() > 32)
        return null ;

    try
    {
        rs = RecordStore.openRecordStore(rsname,true) ;
        return rs ;
    }catch(Exception e)
    {
        return null ;
    }
}
```

如果不管如何您只要開啟已存在的紀錄倉儲，函式如下：

```
public RecordStore openRSExisted(String rsname)
{
    RecordStore rs = null ;
    //名稱大於 32 個字元就不接受
    if(rsname.length() > 32)
        return null ;

    try
    {
        rs = RecordStore.openRecordStore(rsname,false) ;
        return rs ;
    }catch(Exception e)
    {
        return null ;
    }

}
```

開啟紀錄倉儲之後，我們可以利用一些 RecordStore 類別中的函式來取得它的相關資訊，工具函式有下列幾項：

getLastModified() → 取得上次修改時間。

getName() → 取得紀錄倉儲的名稱。

getNextRecordID() → 下一筆記錄的識別碼。

getNumRecords() → 取得紀錄倉儲之中紀錄比數。

getSize() → 取得目前紀錄倉儲所佔據的記憶空間。

getSizeAvailable() → 看看目前還剩下多少記憶空間可以寫入。

getVersion() → 取得紀錄倉儲的版本號碼（每次修改紀錄倉儲的資料，版本號碼就會更新）。

RecordStore 類別中提供了一個 listRecordStores()函式，可以讓我們取得目前 MIDlet 所存在的 MIDlet Suite 之中所有紀錄倉儲的名稱。

當紀錄倉儲使用完畢之後，我們必須使用的 RecordStore 類別的 closeRecordStore()函式來關閉紀錄倉儲，如果不關閉它，會浪費系統資源。其實 closeRecordStore()並非關閉資料庫本身，而只是告知 Java Application Manager，目前使用該紀錄倉儲的 thread 已經不使用該紀錄倉儲了。

如果我們已經不需要該紀錄倉儲，我們必須使用的 RecordStore 類別的 deleteRecordStore()函式，並傳入紀錄倉儲名稱以刪除該紀錄倉儲。我們會寫一個方便的函式幫我們處理細節：

```
public boolean deleteRS(String rsname)
{
    if(rsname.length() > 32)
        return false ;
    try
    {
        RecordStore.deleteRecordStore(rsname) ;
    }catch(Exception e)
    {
        return false ;
    }
    return true ;
}
```

根據上述說明，我們撰寫範例程式如下：

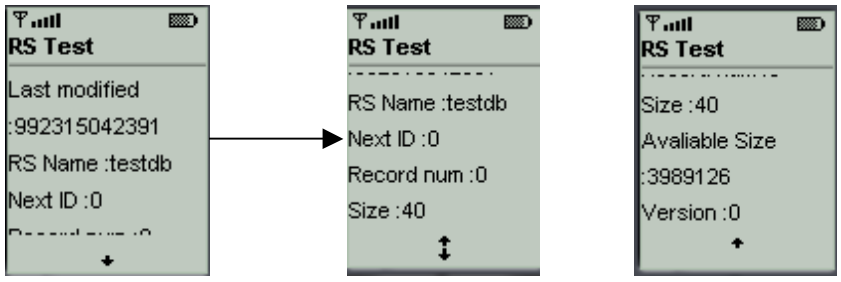
RecordStoreTest.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.rms.* ;
public class RecordStoreTest extends MIDlet implements
ItemStateListener
{
   private Display display;
   public RecordStoreTest()
   {
      display = Display.getDisplay(this);
   }
   public void startApp()
   {
     String dbname = "testdb" ;
     Form f = new Form("RS Test") ;
     RecordStore rs = openRSAnyway(dbname) ;
     if( rs == null )
     {
        //開啓失敗停止 MIDlet
        f.append("Table open fail") ;
     }else
     {
        try
        {
             f.append("Last modified :" +
rs.getLastModified()) ;
             f.append("\nRS Name :" + rs.getName() ) ;
             f.append("\nNext ID :" + rs.getNextRecordID()) ;
             f.append("\nRecord num :" + rs.getNumRecords()) ;
             f.append("\nSize :" + rs.getSize()) ;
             f.append("\nAvaliable Size :" +
rs.getSizeAvailable()) ;
             f.append("\nVersion :" + rs.getVersion()) ;
             rs.closeRecordStore() ;
             deleteRS(dbname) ;
        }catch(Exception e)
        {
        }
     }
     display.setCurrent(f) ;
```

```
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean unconditional)
    {
    }

    public void itemStateChanged(Item item)
    {

    }

    public RecordStore openRSAnyway(String rsname)
    {
      RecordStore rs = null ;
      //名稱大於 32 個字元就不接受
      if(rsname.length() > 32)
         return null ;

      try
      {
         rs = RecordStore.openRecordStore(rsname,true) ;
         return rs ;
      }catch(Exception e)
      {
         return null ;
      }

    }
    public boolean deleteRS(String rsname)
    {
      if(rsname.length() > 32)
         return false ;
      try
      {
         RecordStore.deleteRecordStore(rsname) ;
      }catch(Exception e)
      {
         return false ;
      }
      return true ;
     }
}
```

【執行結果】

初始畫面　　捲動　　捲動



資料儲存

開啟資料庫之後，接下來我們要學習如何將資料存入資料庫之中。RecordStore 類別之中提供了五個用來存取紀錄倉儲的函式，他們分別是:

1. int addRecord(byte[] data, int offset, int numBytes) → 將 bytes 陣列存放到資料倉儲之中，並傳回其 Record ID。請注意，一旦資料寫入資料倉儲之後，其 Record ID 就不會再改變，不管您刪除了某一筆記錄或插入了某一筆記錄，都不會改變，Record ID 在紀錄倉儲所扮演的角色類似一般關聯式資料表格中的 Primary Key。
2. int getRecord(int recordid, byte[] data, int offset) → 取出特定 Record ID 的那筆資料，彈性大。

3. byte[]getRecord(int recordid) ➞ 取出特定 Record ID 的那筆資料，彈性沒有 2 的大。
4. void setRecord(int recordid, byte[] newdata, int offset, int num Bytes) ➞ 設定特定 Record ID 的那筆資料，彈性大
5. void deleteRecord(int recordid) ➞ 刪除特定 Record ID 的那筆資料。

由這些函式我們可以看出，紀錄倉儲只提供寫入 byte 陣列的服務，因此當我們寫入一般非 byte 的資料型態時就比較麻煩，我們必須撰寫一些工具程式做額外處理：

把整數化成 byte 陣列存入

```
public int writeInt2RS(RecordStore rs,int data)
{
    byte []tmp = new byte[4] ;
    tmp[0] = (byte)(0xff&( data >> 24)) ;
    tmp[1] = (byte)(0xff&( data >> 16)) ;
    tmp[2] = (byte)(0xff&( data >>  8)) ;
    tmp[3] = (byte)(0xff&( data >>  0)) ;
    try
    {
        return rs.addRecord(tmp,0,tmp.length) ;
    }catch(Exception e)
    {
    }
    return -1 ;
}
```

將 byte 陣列取出之後化成整數

```
public int readInt4RS(RecordStore rs,int recordid)
{
    byte []tmp = new byte[4] ;
    try
    {
        tmp = rs.getRecord(recordid) ;
    }catch(Exception e)
    {
    }
    int result = tmp[0] ;
    result = (result << 8) + tmp[1] ;
    result = (result << 8) + tmp[2] ;
    result = (result << 8) + tmp[3] ;
    return result ;
}
```

由於 Java 中的字元型態為 16 bits，所以上述程式經過修改之後就可以用來處理文字的儲存與取出。範例程式如下：

CharRecordStoreTest.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.rms.* ;
public class CharRecordStoreTest extends MIDlet
implements ItemStateListener
{
   private Display display;
   public CharRecordStoreTest ()
   {
      display = Display.getDisplay(this);
   }
   public void startApp()
   {
    String dbname = "testdb" ;
    Form f = new Form("RS Test") ;
    RecordStore rs = openRSAnyway(dbname) ;
```

```
if( rs == null )
{
     //開啓失敗停止 MIDlet
     f.append("Table open fail") ;
}else
{
     try
     {
          int firstid = writeChar2RS(rs,'測') ;
          int secondid = writeChar2RS(rs,'試') ;
          f.append(""+readChar4RS(rs,secondid)) ;
          f.append(""+readChar4RS(rs,firstid)) ;
          rs.closeRecordStore() ;
          deleteRS(dbname) ;
     }catch(Exception e)
     {
     }
}

display.setCurrent(f) ;
}
public void pauseApp()
{
}
public void destroyApp(boolean unconditional)
{
}

public void itemStateChanged(Item item)
{

}
public int writeChar2RS(RecordStore rs,char data)
{
    byte []tmp = new byte[2] ;
    tmp[0] = (byte)(0xff&( data >>  8)) ;
    tmp[1] = (byte)(0xff&( data >>  0)) ;
try
{
     return rs.addRecord(tmp,0,tmp.length) ;
}catch(Exception e)
```

```
 {
 }
 return -1 ;
}
public char readChar4RS(RecordStore rs,int recordid)
{
   byte []tmp = new byte[2] ;
   try
   {
      tmp = rs.getRecord(recordid) ;
   }catch(Exception e)
   {
   }
   char result = (char)tmp[0] ;
   result = (char)((result << 8) + (char)tmp[1]) ;
   return result ;
}
public RecordStore openRSAnyway(String rsname)
{
   RecordStore rs = null ;
   //名稱大於 32 個字元就不接受
   if(rsname.length() > 32)
      return null ;

   try
   {
      rs = RecordStore.openRecordStore(rsname,true) ;
      return rs ;
   }catch(Exception e)
   {
      return null ;
   }
}
public boolean deleteRS(String rsname)
{
   if(rsname.length() > 32)
      return false ;
   try
   {
      RecordStore.deleteRecordStore(rsname) ;
   }catch(Exception e)
```

```
      {
         return false ;
      }
      return true ;
   }
}
```

【執行結果】

先取出 Record ID 爲 secondid 的字元
再取出 Record ID 爲 firstid 的字元

上述範例都是講述如何儲存基本型別，您一定會問：看起來紀錄倉儲只有一個型態為 byte 陣列的欄位供我們儲存資料，如果我們需要多欄位的時候該怎麼辦呢？底下我們提供您一個解決方案，可以讓您將整個物件序列化成 byte 陣列存入紀錄倉儲，也可以從資料倉儲之中讀入一個 byte 陣列，然後將其回復成原本物件內部的狀態。

在此我們要借助四個類別的協助，他們分別是：

ByteArrayOutputStream、**ByteArrayInputStream**、**DataOutputStream**、**DataInputStream**。

假設我們我要設計一個通訊錄程式，裡頭需要一個資料結構叫做 FriendData，其結構如下：

```
class FriendData
{
    String name ;
    String tel ;
    boolean sex ;
    int age ;
}
```

為了要將此類別內的狀態轉換成 byte 陣列，或是將 byte 陣列的資料回復成此類的狀態，我們撰寫 decode()函式與 encode 函式，其內容如下：

```
class FriendData
{
    String name ;
    String tel ;
    boolean sex ;
    int age ;
    public FriendData()
    {
         name = "NO NAME" ;
         name = "NO TEL" ;
         sex = false ;
         age = 0 ;
    }
    public byte[] encode()
    {
         byte[] result = null ;
         try
         {
              ByteArrayOutputStream bos =
new ByteArrayOutputStream() ;
              DataOutputStream dos =
new DataOutputStream (bos) ;
```

```
            dos.writeUTF(name) ;
            dos.writeUTF(tel) ;
            dos.writeBoolean(sex) ;
            dos.writeInt(age) ;
            result = bos.toByteArray() ;
            dos.close() ;
            bos.close() ;

        }catch(Exception e)
        {
        }
        return result ;
    }
    public void decode(byte[] data)
    {
        try
        {
            ByteArrayInputStream bis =
new ByteArrayInputStream(data) ;
            DataInputStream dis =
new DataInputStream (bis) ;
            name = dis.readUTF() ;
            tel = dis.readUTF() ;
            sex = dis.readBoolean() ;
            age = dis.readInt() ;
            dis.close() ;
            bis.close() ;

        }catch(Exception e)
        {
        }
    }
}
```

此類別的使用範例如下：

DataRecordStoreTest.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

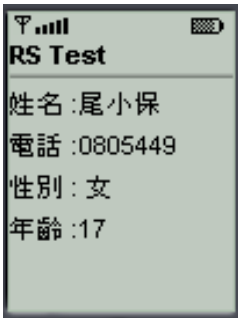
```
import java.io.*;
import javax.microedition.rms.* ;
public class DataRecordStoreTest extends MIDlet implements
ItemStateListener
{
   private Display display;
   public DataRecordStoreTest()
   {
      display = Display.getDisplay(this);
   }
   public void startApp()
   {
    String dbname = "testdb" ;
    Form f = new Form("RS Test") ;
    RecordStore rs = openRSAnyway(dbname) ;
    if( rs == null )
    {
        //開啓失敗停止 MIDlet
        f.append("Table open fail") ;
    }else
    {
        try
        {
            FriendData fd = new FriendData() ;
            fd.name = "尾小保" ;
            fd.tel = "0805449" ;
            fd.sex = false ;
            fd.age = 17 ;
            byte tmp[] = fd.encode() ;
            int id = rs.addRecord(tmp,0,tmp.length) ;
            FriendData fd2 = new FriendData() ;
            fd2.decode(rs.getRecord(id)) ;
            f.append("姓名 :"+ fd2.name) ;
            f.append("\n電話 :"+ fd2.tel) ;
            if(fd2.sex)
            {
                f.append("\n性別 : 男") ;
            }else
            {
                f.append("\n性別 : 女") ;
            }
```

```
            f.append("\n年齡 :"+ fd2.age) ;
            rs.closeRecordStore() ;
            deleteRS(dbname) ;
        }catch(Exception e)
        {
        }
 }

 display.setCurrent(f) ;
}
public void pauseApp()
{
}
public void destroyApp(boolean unconditional)
{
}

public void itemStateChanged(Item item)
{

}

public RecordStore openRSAnyway(String rsname)
{
   RecordStore rs = null ;
   //名稱大於 32 個字元就不接受
   if(rsname.length() > 32)
      return null ;

   try
   {
      rs = RecordStore.openRecordStore(rsname,true) ;
      return rs ;
   }catch(Exception e)
   {
      return null ;
   }
}
public boolean deleteRS(String rsname)
{
   if(rsname.length() > 32)
```

```
            return false ;
        try
        {
            RecordStore.deleteRecordStore(rsname) ;
        }catch(Exception e)
        {
            return false ;
        }
        return true ;
    }

}
```

【執行結果】

從紀錄倉儲取出資料之後顯示在螢幕上



除了新增紀錄之外，我們隨時都可以利用 deleteRecord()函式來刪除特定 Record ID 的資料，或者使用 setRecord()函式來重新設定特定 Record ID 的資料。

在次提醒您，每一筆記錄的 Record ID 在資料寫入紀錄倉儲的時候就已經確定，之後不會因為資料的增加或刪除而變動。因為我們使用 deleteRecord()或 setRecord()函式時都需要 Record ID 作為參數，所以除了在使用前確定該 Record ID 所存放的就是您要的

資料之外，也請同時確認該 Record ID 的確存在於紀錄倉儲之中，否則在執行時期會引發 InvalidRecordIDException 例外。

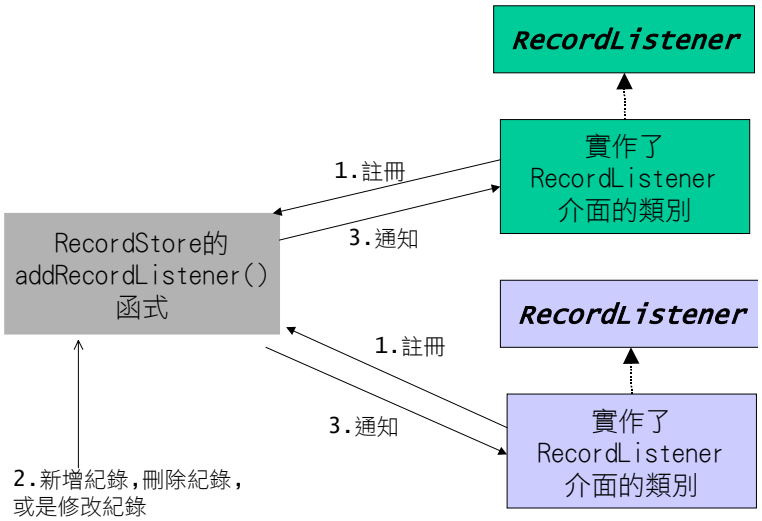
監視紀錄倉儲的變化

紀錄管理系統提供了一個監視機制，讓我們的程式可以隨時得知資料庫目前的情況，比方說是否有新的一筆紀錄加入紀錄倉儲中、紀錄被刪除、或者紀錄被修改。這個監視機制是使用 RecordListener 介面來達成。

注 意

RecordListener 支援 multicast，也就是同時間可以有許多人可以註冊監聽紀錄倉儲的狀況。

整體運作方式如下圖所示：



使用範例如下：

RLTest.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
import javax.microedition.rms.* ;
public class RLTest extends MIDlet implements RecordListener
{
   private Display display;
   public RLTest()
   {
      display = Display.getDisplay(this);
   }
   public void startApp()
   {
    String dbname = "testdb" ;
    Form f = new Form("RS Test") ;
    RecordStore rs = openRSAnyway(dbname) ;
    rs.addRecordListener(this) ;
    if( rs == null )
    {
        //開啓失敗停止 MIDlet
        f.append("Table open fail") ;
    }else
    {
        try
        {
            byte []data = new byte[2] ;
            data[0] = 15 ;
            data[1] = 16 ;
            int id = rs.addRecord(data,0,data.length) ;
            data[0] = 25 ;
            data[1] = 26 ;
            rs.setRecord(id,data,0,data.length) ;
            rs.deleteRecord(id) ;
            rs.addRecord(data,0,data.length) ;
            rs.closeRecordStore() ;
            deleteRS(dbname) ;
```

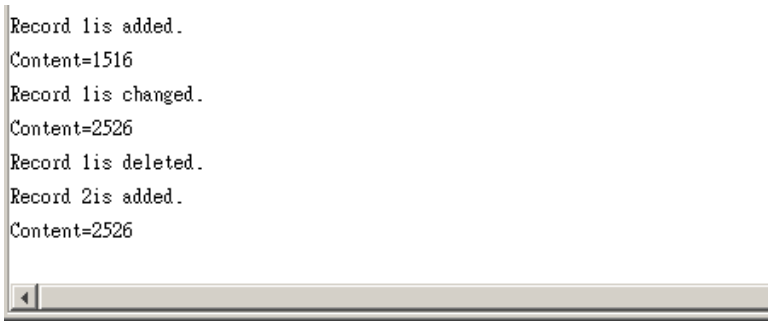
```
        }catch(Exception e)
        {
        }
    }

    display.setCurrent(f) ;
   }
   public void pauseApp()
   {
   }
   public void destroyApp(boolean unconditional)
   {
   }

   public void recordAdded(RecordStore rs,int recordid)
   {
    try
    {
        byte []tmp = rs.getRecord(recordid) ;
        System.out.println("Record " + recordid + "is added.") ;
        System.out.println("Content=" + tmp[0] + tmp[1]) ;
    }catch(Exception e)
    {
        System.out.println(e.getMessage()) ;
    }
   }
   public void recordChanged(RecordStore rs,int recordid)
   {
    try
    {
        byte []tmp = rs.getRecord(recordid) ;
        System.out.println("Record " + recordid + "is
changed.") ;
        System.out.println("Content=" + tmp[0] + tmp[1]) ;
    }catch(Exception e)
    {
        System.out.println(e.getMessage()) ;
    }
   }
   public void recordDeleted(RecordStore rs,int recordid)
   {
```

```
    try
    {
         //請勿在此使用 byte []tmp = rs.getRecord(recordid) ;
         //因爲此時該筆記錄已被刪除
         System.out.println("Record " + recordid + "is
deleted.") ;

    }catch(Exception e)
    {
         System.out.println(e.getMessage()) ;
    }
   }

   public RecordStore openRSAnyway(String rsname)
   {
     RecordStore rs = null ;
     //名稱大於 32 個字元就不接受
     if(rsname.length() > 32)
        return null ;

     try
     {
        rs = RecordStore.openRecordStore(rsname,true) ;
        return rs ;
     }catch(Exception e)
     {
        return null ;
     }

   }
   public boolean deleteRS(String rsname)
   {
     if(rsname.length() > 32)
        return false ;
     try
     {
        RecordStore.deleteRecordStore(rsname) ;
     }catch(Exception e)
     {
        return false ;
     }
```

```
      return true ;
     }
}
```

【執行結果】

命令列輸出結果



注 意

這些回呼函式都是在紀錄倉儲完成動作之後才被呼叫，而非之前，因此在本範例之中唯一需要留意的地方只有在 recordDeleted()函式。我們在 recordAdded()或 recordChanged()裡頭仍然可以利用 getRecord()取出該筆記錄，但是無法在 recordDeleted()函式之中這樣做。因為我們的函式是在記錄被刪除之後才被呼叫，因此當時該筆紀錄早已不復在，所以請勿在 recordDeleted()函式裡頭使用 getRecord()取得剛被刪除的資料，這將引發執行時期例外。

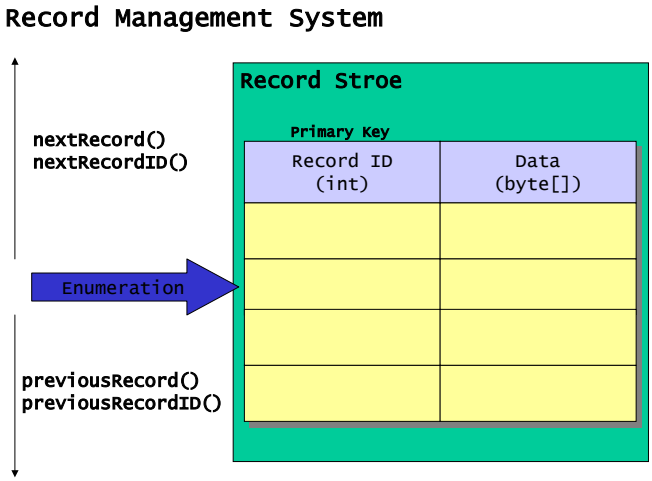
走訪紀錄倉儲

到目前為止，您一定發現了一個紀錄倉儲的重大缺點，就是不管新增、修改、或刪除紀錄，都需要 Record ID 的輔助，因此我們常常需要額外的變數幫我們紀錄 Record ID，因為沒有 Record ID，我們根本無法存取紀錄倉儲之中的紀錄。那麼，有沒有辦法可以不靠 Record ID 來存取資料庫呢？在您腦中閃過的想法大概如下：

```
for(int i = 1 ; i < rs.getNextRecordID() ; i++)
{
        byte []data = rs.getRecord(i) ;
        ... ...
}
```

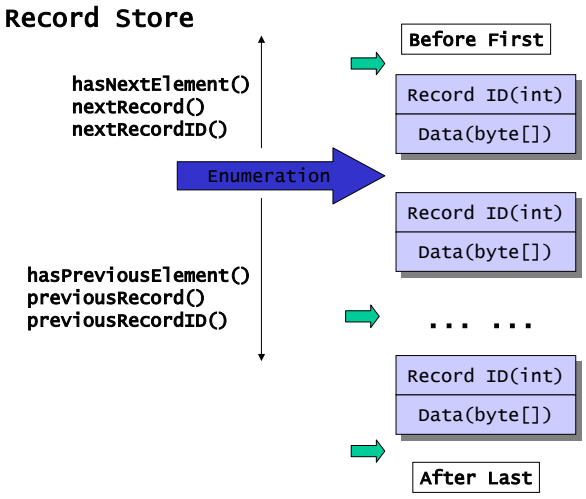
問題是，這種方法有潛在的問題，我們知道 Record ID 是在紀錄一寫進紀錄倉儲之後就固定的，雖然使用 getNextRecordID()可以保證存取到所有可能出現的 Record ID，但是卻無法保證所有的 Record ID 都存在於資料倉儲之中（可能被 deleteRecord()函式所刪除），因此取得的 byte 陣列會是 null。所以上述方法並不安全。因此接下來將講述紀錄管理系統所提供的另外一種安全的存取方法 - RecordEnumeration 介面。

RecordEnumeration 就像是一個指向紀錄倉儲的指位器，我們可以使用如下圖般的抽象概念：



也就是說，一旦我們取得 RecordEnumeration，就可以在紀錄倉儲之中來回移動，並取得儲存於紀錄倉儲中的資料。

但是，實際上的 RecordEnumeration 的設計概念應該如下圖所示：



由上圖我們可以看出，RecordEnumeration 的設計和標準 Java Collection Framework 裡頭的 Collection 介面幾乎大同小異。重要的特點就在於 RecordEnumeration 並非指向紀錄本身，而是指向紀錄與紀錄之間。

當我們建立了 RecordEnumeration 之後，首先他會指向 Before First 和第一筆記錄之間，然後我們可以利用 hasNextElement()看看之後是否還有資料，當 RecordEnumeration 移動到 After Last 與最後一筆資料中間時，hasNextElement()就回傳回 false。hasPreviousElement()函式的用法和 hasNextElement()相反，如果您需要從最後一筆資料往前移動，那麼 hasPreviousElement()會在資料移動到 Before First 和第一筆記錄之間時傳回 false。

注　意

呼叫 hasNextElement()、hasPreviousElement()、nextRecordID()、previousRecordID()並不會改變 RecordEnumeration 的位置，這些函式只是試探性質，取得相關資訊的功能而已。真正能夠改變 RecordEnumeration 位置的函式為 nextRecord()和 previousRecord()。nextRecord()會傳回下一筆資料的 byte 陣列，並將 RecordEnumeration 下一個紀錄與紀錄之間的空格，反之，previousRecord()會傳回上一筆資料的 byte 陣列，並將 RecordEnumeration 上一個紀錄與紀錄之間的空格。

任何時候，我們都可以呼叫 reset()函式將 RecordEnumeration 重置到 Before First 與第一筆記錄之間。也可以使用 numRecords()

函式取得紀錄倉儲之中的紀錄數目。當您使用完之後，我們建議您使用 destory()函式以釋放 RecordEnumeration 因為走訪紀錄倉儲所使用的系統資源。如果您的紀錄倉儲常常增加、修改、刪除資料，那麼最好在走訪前使用 rebuild()函式重建 RecordEnumeration。

RecordEnumeration 的用法如以下範例：

```
public void travelRS(RecordStore rs)
{
    try
    {
        RecordEnumeration re =
        rs.enumerateRecords(null,null,false) ;
        System.out.println("There are " +
re.numRecords() + " in RecordStore") ;
        for(int i = 0 ; i < re.numRecords() ; i++)
        {

            int id = re.nextRecordId() ;
            byte tmp[] = rs.getRecord(id) ;
            System.out.println(tmp[0] + " " + tmp[1]) ;
        }
    }
    catch(Exception e)
    {
    }
}
```

或者

```
public void travelRS(RecordStore rs)
{
    try
    {
        RecordEnumeration re =
rs.enumerateRecords(null,null,false) ;
        System.out.println("There are " + re.numRecords() +
```

```
" in RecordStore") ;
        while(re.hasNextElement())
        {
             byte tmp[] = re.nextRecord() ;
             System.out.println(tmp[0] + " " + tmp[1]) ;
        }
    }
    catch(Exception e)
    {
    }
}
```

完整的使用範例如下：

RETest.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
import javax.microedition.rms.* ;
public class RETest extends MIDlet
{
   private Display display;
   public RETest()
   {
      display = Display.getDisplay(this);
   }
   public void startApp()
   {
    String dbname = "testdb" ;
    Form f = new Form("RE Test") ;
    RecordStore rs = openRSAnyway(dbname) ;
    if( rs == null )
    {
         //開啓失敗停止 MIDlet
         f.append("Table open fail") ;
    }else
    {
         try
         {
```

```
            byte []data = new byte[2] ;
            data[0] = 15 ;
            data[1] = 16 ;
            rs.addRecord(data,0,data.length) ;
            data[0] = 25 ;
            data[1] = 26 ;
            rs.addRecord(data,0,data.length) ;
            data[0] = 35 ;
            data[1] = 36 ;
            rs.addRecord(data,0,data.length) ;
            travelRS(rs) ;
            rs.closeRecordStore() ;
            deleteRS(dbname) ;
        }catch(Exception e)
        {
        }
    }

    display.setCurrent(f) ;
   }
   public void pauseApp()
   {
   }
   public void destroyApp(boolean unconditional)
   {
   }

   public void travelRS(RecordStore rs)
   {
     try
     {
        RecordEnumeration re =
rs.enumerateRecords(null,null,false) ;
        System.out.println("There are " + re.numRecords() +
" in RecordStore") ;
        while(re.hasNextElement())
        {
            byte tmp[] = re.nextRecord() ;
            System.out.println(tmp[0] + " " + tmp[1]) ;
        }
     }
```

```
      catch(Exception e)
      {
      }
   }
   public RecordStore openRSAnyway(String rsname)
   {
    RecordStore rs = null ;
    //名稱大於 32 個字元就不接受
    if(rsname.length() > 32)
         return null ;

    try
    {
         rs = RecordStore.openRecordStore(rsname,true) ;
         return rs ;
    }catch(Exception e)
    {
         return null ;
    }

   }
   public boolean deleteRS(String rsname)
   {
    if(rsname.length() > 32)
         return false ;
    try
    {
         RecordStore.deleteRecordStore(rsname) ;
    }catch(Exception e)
    {
         return false ;
    }
    return true ;
   }

}
```

【執行結果】

命令列輸出結果

```
There are 3 in RecordStore
35 36
25 26
15 16
```

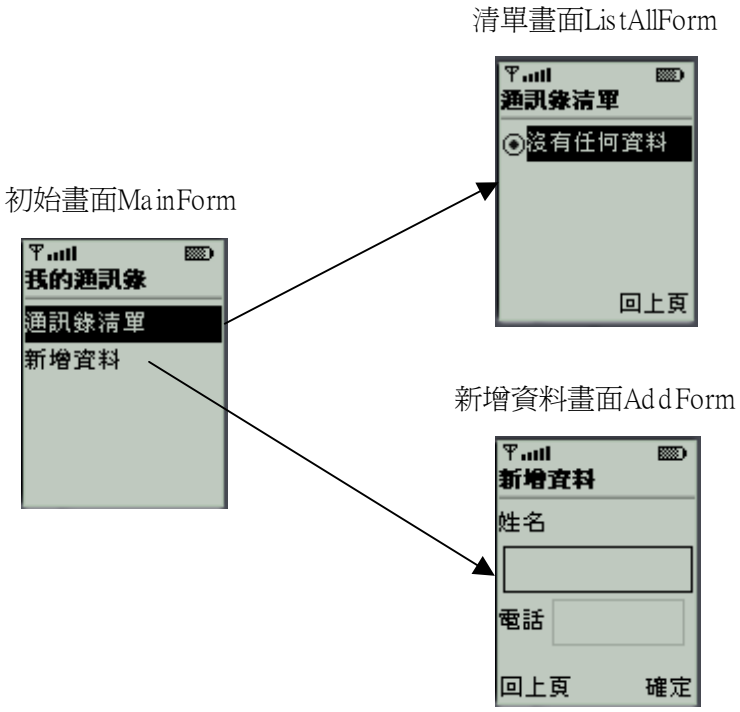
多緒執行的相關議題

同一個紀錄倉儲允許同時間有個 Thread 去存取它，而 RecordStore 所提供的所有資料庫存取功能都是不可分割的動作，意思就是說，即使同時間有多個 Thread 使用紀錄倉儲的功能，每個功能都會在完成之後再執行下一個功能，不會在中途被打斷，如此一來可以保護資料的完整性。

但是，如果您必須同時間好幾個 Thread 一同存取好幾個紀錄倉儲，比方說您設計了一個類似關聯式資料庫的系統，修改了紀錄倉儲 A 的資料時必須連帶修改紀錄倉儲 B 之中的資料。這時紀錄管理系統並無法幫您處理，您必須自己手動利用 Java 的同步機制來處理跨越不同紀錄倉儲所產生的資料一致性問題。

實際範例：通訊錄

最後，我們將以一個通訊錄程式範例做個總結，次範例將會結合之前所有討論到的所有紀錄管理相關功能。



完整的程式碼如下：

MyAddressBook.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
import javax.microedition.rms.* ;
public class MyAddressBook extends MIDlet implements
CommandListener
```

```
{
   private Display display;
   String dbname = "MyAddressBook" ;
   String current = "" ;

   Form addForm = new Form("新增資料") ;
   TextField tf1 = new TextField("姓名","",20,TextField.ANY);
   TextField tf2 = new TextField("電話
","",10,TextField.NUMERIC);

   public MyAddressBook()
   {
      display = Display.getDisplay(this);
   }
   public void startApp()
   {
       Command add = new Command("確定",Command.SCREEN,1) ;
       Command back = new Command("回上頁",Command.SCREEN,2) ;
       addForm.append(tf1) ;
       addForm.append(tf2) ;
       addForm.addCommand(add) ;
       addForm.addCommand(back) ;
       addForm.setCommandListener(this) ;
    MainForm() ;

   }
   public void MainForm()
   {
       List l = new List("我的通訊錄",Choice.IMPLICIT) ;
       l.append("通訊錄清單",null) ;
       l.append("新增資料",null) ;
       l.setCommandListener(this) ;
       current = "MainForm" ;
       display.setCurrent(l) ;
   }
   public void ListAllForm()
   {
       List l = new List("通訊錄清單",Choice.EXCLUSIVE) ;
       Command back = new Command("回上頁",Command.SCREEN,2) ;
       l.addCommand(back) ;
       l.setCommandListener(this) ;
       RecordStore rs = openRSAnyway(dbname) ;
```

```
        if( rs == null )
        {
               //開啓失敗停止 MIDlet
              notifyDestroyed();
        }else
        {
            try
            {
                RecordEnumeration re
= rs.enumerateRecords(null,null,false) ;
                FriendData data = new FriendData() ;
                if(re.numRecords() == 0)
                {
                    l.append("沒有任何資料",null) ;
                    current = "ListAllForm" ;
                    display.setCurrent(l) ;
                    return ;
                }

                while(re.hasNextElement())
                {
                    byte tmp[] = re.nextRecord() ;
                    data.decode(tmp) ;
                    l.append(data.name,null) ;
                }
                rs.closeRecordStore() ;
            }catch(Exception e)
            {
            }
        }
        Command info = new Command("詳細",Command.SCREEN,1) ;
        l.addCommand(info) ;
        current = "ListAllForm" ;
        display.setCurrent(l) ;
    }

    public void AddForm()
    {
        current = "AddForm" ;
        display.setCurrent(addForm) ;
    }
```

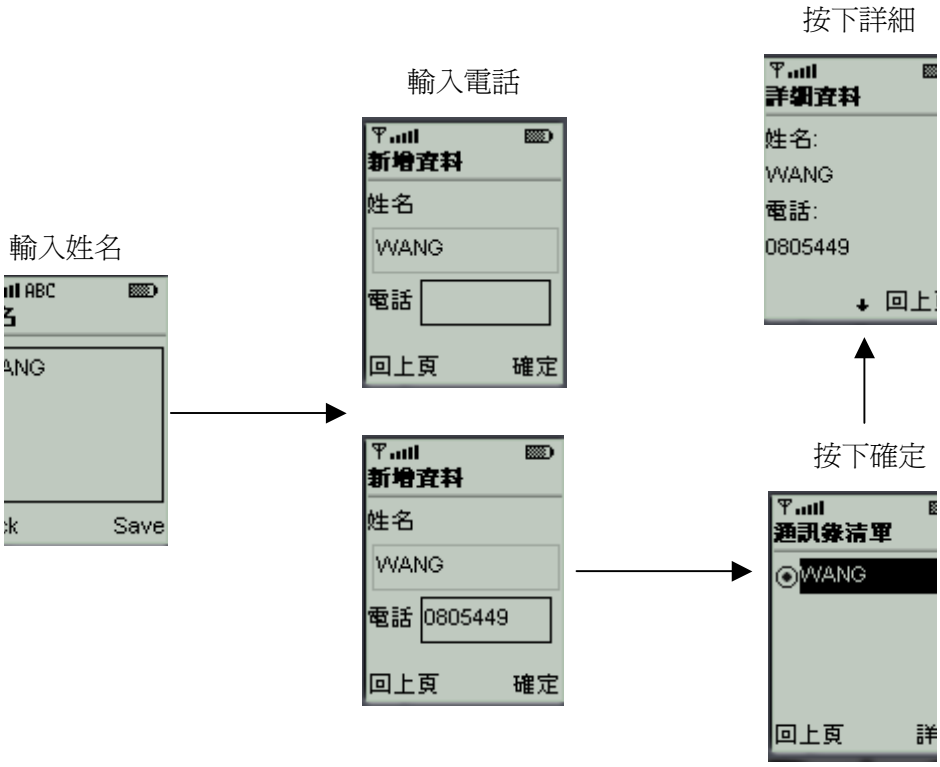
```
public void commandAction(Command c,Displayable s)
{
 if(c == List.SELECT_COMMAND && current.equals("MainForm"))

 {
     List tmp = (List) s ;
     switch(tmp.getSelectedIndex())
     {
          case 0 :
               ListAllForm() ;
               break ;
          case 1 :
               AddForm() ;
               break ;
     }
 }
 if(c.getLabel().equals("回上頁"))
 {
     MainForm() ;

 }
 if(c.getLabel().equals("確定"))
 {
     addData() ;
     ListAllForm() ;
 }
 if(c.getLabel().equals("詳細"))
 {
     List tmp = (List) s ;
     searchData(tmp.getString(tmp.getSelectedIndex())) ;
 }

}
public void searchData(String name)
{
 Form f = new Form("詳細資料") ;
 Command back = new Command("回上頁",Command.SCREEN,1) ;
 f.addCommand(back) ;
 f.setCommandListener(this) ;
 RecordStore rs = openRSAnyway(dbname) ;
 if( rs == null )
 {
```

```
        //開啓失敗停止 MIDlet
             notifyDestroyed();
    }else
    {
        try
        {
            RecordEnumeration re
= rs.enumerateRecords(null,null,false) ;
            FriendData data = new FriendData() ;
            while(re.hasNextElement())
            {
                byte tmp[] = re.nextRecord() ;
                data.decode(tmp) ;
                if(data.name.equals(name))
                {
                    f.append("姓名: \n") ;
                    f.append(data.name+"\n") ;
                    f.append("電話: \n") ;
                    f.append(data.tel+"\n") ;
                    display.setCurrent(f) ;
                    return ;
                }
            }
            rs.closeRecordStore() ;
        }catch(Exception e)
        {
        }
    }
   }
   public void addData()
   {
    FriendData data = new FriendData() ;
    data.name = tf1.getString() ;
    data.tel = tf2.getString() ;
    byte []tmp = data.encode() ;
    RecordStore rs = openRSAnyway(dbname) ;
    if( rs == null )
    {
        //開啓失敗停止 MIDlet
             notifyDestroyed();
    }else
    {
```

```
        try
        {
             rs.addRecord(tmp,0,tmp.length) ;
             rs.closeRecordStore() ;
        }catch(Exception e)
        {
        }
    }
   }

   public void pauseApp()
   {
   }
   public void destroyApp(boolean unconditional)
   {
   }

   public RecordStore openRSAnyway(String rsname)
   {
    RecordStore rs = null ;
    //名稱大於 32 個字元就不接受
    if(rsname.length() > 32)
         return null ;

    try
    {
         rs = RecordStore.openRecordStore(rsname,true) ;
         return rs ;
    }catch(Exception e)
    {
         return null ;
    }

   }
}

class FriendData
{
    String name ;
    String tel ;
    public FriendData()
    {
```

```
        name = "NO NAME" ;
        name = "NO TEL" ;
    }
    public byte[] encode()
    {
        byte[] result = null ;
        try
        {
            ByteArrayOutputStream bos = new
ByteArrayOutputStream() ;
            DataOutputStream dos = new DataOutputStream (bos) ;
            dos.writeUTF(name) ;
            dos.writeUTF(tel) ;
            result = bos.toByteArray() ;
            dos.close() ;
            bos.close() ;

        }catch(Exception e)
        {
        }
        return result ;
    }
    public void decode(byte[] data)
    {
        try
        {
            ByteArrayInputStream bis = new
ByteArrayInputStream(data) ;
            DataInputStream dis = new DataInputStream (bis) ;
            name = dis.readUTF() ;
            tel = dis.readUTF() ;
            dis.close() ;
            bis.close() ;

        }catch(Exception e)
        {
        }
    }
}
```

【執行結果】



本範例並不完整，比方說少了刪除紀錄或清空整個紀錄倉儲的功能。這兩個小功能就當作是給大家的練習題吧!

總　結

在本章中，我們為大家說明了 MIDP 所提供的紀錄管理系統之相關用法，也在最後提供了一個實際使用紀錄管理系統的程式範例。

其實紀錄管理系統之中還提供了其他小功能，不過因為不常使用，所以本章中只稍微提及，卻並沒有深入探討，有些甚至連提都沒提到。如果本章所討論的功能仍然不敷您的使用，請參閱 MIDP API 參考手冊尋找更多資料。

memo