PART 4

資料存取篇

Q31 如何利用離線資料存取 (Disconnected Data Access) 提昇資料庫存取效率？

適用範圍： V ASP.NET 1.0 V ASP.NET 1.1 V ASP.NET 2.0 V ASP.NET 3.5

問題

我的網站是用 ASP.NET 撰寫的, 用 ADO.NET 連接資料庫, 最近網站的資料存取速度似乎有點變慢, 我知道 ADO.NET 有離線資料存取的能力, 但我要怎麼利用它來加快一些常用資料的存取速度？

問題說明

離線資料存取模型一直是 ADO.NET 所主打的特色之一, 早期 ADO.NET 剛推出時, DataSet 可紅的不得了, 有不少的應用都會用 DataSet 來實作中間層的資料存取工作, 以減輕資料庫伺服器的負擔, 而前端就利用 DataSet 來查詢資料, 不用再與資料庫伺服器連接存取, 速度會快的多。

DataSet 本身的特性就是一個儲存在記憶體中的資料庫 (In-Memory Database) , 它的組成就像是一堆 Collection 相互管理, 並且提供物件存取的屬性和函數 (例如 Select(), Compute() 等) 供應用程式介面取用, 在某些程度下的運算真的很方便, 以及在資料的管理上, DataSet 確實有相當程度的便利性。

不過 DataSet 的缺點也是因爲它的特性所致, 在使用上也會有一些缺點, 尤其是在 Web 應用程式上使用時 (可參考「Q32：要用 DataSet, DataTable 好, 還是用 DataReader 好」問答說明) , 所以需要適當的規劃與運用, 才可以發揮出 DataSet 離線資料存取的能力, 以下爲筆者的 DataSet 運用經驗與心法, 供讀者參考。

首先, DataSet 應該是存放應用程式層次的共用資料, 例如組態設定, 或者是一些需要在應用程式中共用的資訊 (例如上線人數) , 這些資訊可以存放在 Application 變數中, 再搭配 SqlCacheDependency 來偵測資料表有沒有改變, 若有再重新填充更新資料, 例如程式 1 所示。

程式1：DataSet 快取線上人員清單範例

```
<%@ Application Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
// 負責快取工作的 SqlCacheDependency
private SqlCacheDependency sqlcache =
  new SqlCacheDependency("myDB", "UserOnline");

void Application_Start(object sender, EventArgs e)
{
  // 載入起始資料。
  SqlConnection conn = new SqlConnection(
    ConfigurationManager.ConnectionStrings[
      "myConnectionString"].ConnectionString);
  SqlCommand cmd = new SqlCommand("SELECT * FROM UserOnline", conn);
  SqlDataAdapter adapter = new SqlDataAdapter(cmd);
  DataSet ds = new DataSet();

  adapter.Fill(ds, "UserOnline");
  Application["UserOnline"] = ds;
  adapter = null;

  cmd.Dispose();
  conn.Dispose();
}
```

```
void Application_End(object sender, EventArgs e)
{
}

void Application_Error(object sender, EventArgs e)
{
}

void Application_BeginRequest(object sender, EventArgs e)
{
  // 加入上線使用者
  // 註：在此只供測試，讀者可將此程式移到使用者登入的程式碼中
  SqlConnection conn = new SqlConnection(
    ConfigurationManager.ConnectionStrings[
      "myConnectionString"].ConnectionString);
  SqlCommand cmdAddUser = new SqlCommand(
    "INSERT INTO UserOnline VALUES (NEWID(), GETDATE(), '"
    + Request.UserHostAddress + "')");

  cmdAddUser.Connection = conn;
  conn.Open();
  cmdAddUser.ExecuteNonQuery();
  conn.Close();
  cmdAddUser.Dispose();

  if (this.sqlcache.HasChanged) // 若發現資料有變動時，重新填入資料
  {
    SqlCommand cmd = new SqlCommand("SELECT * FROM UserOnline",
      conn);
    SqlDataAdapter adapter = new SqlDataAdapter(cmd);
    DataSet ds = new DataSet();
```

```
    adapter.Fill(ds, "UserOnline");
    Application["UserOnline"] = ds;
    adapter = null;

    cmd.Dispose();
  }

  conn.Dispose();
}

void Session_Start(object sender, EventArgs e)
{
}

void Session_End(object sender, EventArgs e)
{
}

</script>
```

小常識 SqlCacheDependency

SqlCacheDependency 是一個定時偵測資料表有沒有異動的工具類別, 開發人員可以設定輪詢的時間 (例如每分鐘偵測一次) , SqlCacheDependency 的內部程式會依輪詢的時間, 向資料庫發出查詢指令, 以偵測資料表是否有異動過, 若有的話, SqlCacheDependency 的 HasChanged 屬性就會設爲 true, 表示資料表已有異動, 此時應用程式即可更新快取資料或是做其他必要的工作。

接下頁

若要啓用 SqlCacheDependency, 必須要在 Web.config 中設定啓用並且設定輪詢時間：

```
<configuration>
  <connectionStrings>
    <add name="Pubs" connectionString=
       "Data Source=(local); Initial Catalog=pubs; Integrated
Security=true;"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
  <system.web>
    <caching>
      <sqlCacheDependency enabled="true" pollTime="60000" >
        <databases>
          <add name="pubs"
            connectionStringName="pubs"
            pollTime="9000000"
            />
        </databases>
      </sqlCacheDependency>
    </caching>
  </system.web>
</configuration>
```

第二, DataSet 不應放置大量資料, 就算是共用資料亦然, 如果一定要放, 則放置的時間要愈短愈好, 因爲 DataSet 佔用記憶體的量會視 DataSet 的大小而有所不同, 如果只是一些小型資料, 那麼也許還可以接受, 若放的是檔案或圖片這類的大資料, 可能沒多久伺服器的記憶體就被吃光光了。

以一台 1GB RAM 的伺服器爲例, 不考量運算成本, 且在伺服器上有安裝 SQL Server 的話, 應用程式就應該只使用記憶體的 35%, 最高不要超過 40%, 因爲作業系統和 SQL Server 也需要記憶體的, 尤其是 SQL Server 這種記憶體愈多速度愈快的軟體, 作業系統和 SQL Server 的比例大約 20%和 40%, 因此應用程式不應吃太多記憶體爲宜, 若是共用型資料就還可接受, 若是 Session 這種只有單一使用者的資料, 就更應該控制使用量, 否則記憶體空間愈小, 會嚴重影響到系統效能的。

如果 DataSet 佔用太多記憶體的話, 在工作管理員中可以看到 w3wp.exe (IIS 6.0 Web Worker Process) 佔用的記憶體會暴增, 要回收記憶體除了釋放 DataSet 外, 只有回收行程重新啓動一途了 (但這會讓所有使用者的資料消失) , 所以要小心使用 DataSet。

第三, 如果 DataSet 控制的夠小, 可以將它放到 ViewState 中, 以釋放記憶體空間, 不過如果網路頻寬太小的話, 可能就不太適合, 這個要視網路的運作狀況來決定。

第四, 對於多層次 (n-tier) 型的應用程式, 如果要將 DataSet 放在中間層以隔離資料庫的話, 也應只放較沒有風險的資料, 因爲 DataSet 不像 DBMS, 一旦當機或停電, 則放在 DataSet 中的資料就會全部不見, 若放的是交易這種重要資料, 則也會跟著不見, 所以重要的資料不要放在中間層的 DataSet 。另外, 由於 DataSet 沒有交易與並行最佳化能力, 因此儘可能只讀取, 而不要寫入, 否則可能也發生並行讀取的問題。

解決方案

要視存放的資料來決定, 必須要事先規劃, 以及分類資料, 才能決定哪些可以放在 DataSet 中, 並且要在 Global.asax 中實作 SqlCacheDependency 的服務, 讓 DataSet 可以在資料庫被更新的時候立刻得到新的資料, 如此可以減少資料庫的存取負擔。

但也要注意, 若伺服器記憶體不足或預期會有記憶體不足的風險時, 則應減少 DataSet 的資料量 (或增加伺服器記憶體) , 以維持應用程式的穩定。

相關問題

- Q32：要用 DataSet/DataTable 好, 還是用 DataReader 好？

Q32 要用 DataSet/DataTable 好, 還是用 DataReader 好？

適用範圍：V ASP.NET 1.0 V ASP.NET 1.1 V ASP.NET 2.0 V ASP.NET 3.5

問題

ADO.NET 有提供 DataSet、DataTable 與 DataReader 三個資料存取物件, 每一個都有它的作用, 而且也可以順利的讀取資料。我的問題是, 在 ASP.NET 這樣的環境, 用哪種物件會比較好？DataTable 可以存在 Session 中重覆使用, 而 DataReader 又可以很快的存取資料, 那麼在一個大量使用者的 Web 應用程式中, 用哪種方式可以得到最好的效率？

問題說明

DataSet 和 DataTable 是一種離線型的資料物件, 會持續存在記憶體中, 而且 DataSet 可以包含多個 DataTable, 並可以建立關聯 (DataRelation) , 因爲它是一個離線的快取物件, 所以不會有連線佔用的問題, 而且也不需要一直連接到資料庫去存取, 只要留下一份在記憶體中就可以了。DataSet/DataTable 的組合其實可以做很多的應用, 例如 Master/Detail 的表單, 或者是在資料庫之間做一個中繼層等等, 甚至於可以利用 DataTable 來協助應用程式的集合資料處理等等, 可說是功能強大又簡單好用的物件。

雖然 DataSet 和 DataTable 很方便, 但在方便的背後, 它卻有個很大的缺點－佔用資源。尤其是像在 Web 應用程式這種多人環境, 若沒有適當的使用, 會讓 Web Server 的資源消耗的很快 (主要是記憶體的耗量) 。有一些開發人員習慣在 Session 中放 DataTable, 其實這是一個不好的習慣, 因爲只要人數愈來愈多, Server 的記憶體就會愈來愈少 (因爲都被佔用了) , 到後來, 整個應用程式的速度就會變慢, I/O 的量會變的很重 (虛擬記憶體的存取量變大) , 到最後就是變得很慢很慢, 甚至當機。

例如一個 DataTable 佔用 15K 記憶體來算, 可能看起來不多, 100 人可能也只有 1.5MB 左右, 10, 000 人則是 150MB, 在現今動輒 GB 級的記憶體量來看, 也許還算是少的, 然而 Server 的記憶體不是只有 Web 應用程式在用而已, 包括作業系統與相關服務 (防毒軟體等) , 甚至是 SQL Server 資料庫在同一台的時候, 記憶體的消耗可能就會很重, 通常伺服器的正常工作的記憶體耗用量最好是在 50-70%之間, 保留 20-30%可供作業系統調度用, 如果高於 70%的話, 就有可能會讓硬碟的 I/O 增加, 速度就會變慢。

就算把 DataSet 放到 Application 變數中, 除了記憶體的消耗外, 也可能會有多人存取的並行作業問題, 因此, 謹慎的考量 DataSet 和 DataTable 的使用是很重要的。

若想要快速型的資料存取, 那大概除了 DataReader 別無選擇, DataReader 是一種順向型 (Forward-Only) 的資料存取器, 它會在執行時產生資料庫中的列游標 (Server-side Cursor) , 然後透過這個游標來讀取資料列。由於是順向指標, 所以速度會很快, 但只能夠單向的讀取, 無法回溯或隨意的移動游標, 這點和 ADO 時代的 Recordset 就有些不同。

除了只支援順向讀取的功能外, 它還有一個比較明顯的缺點－佔用連線, DataReader 一經開啓, 連線就會被佔用, 若想要做互動型的資料處理 (寫-->讀-->寫-->…) 的話, 可能就沒有辦法, 需要開啓不同的連線才行 (可參考「執行 ExecuteNonQuery()時都會顯示連線被佔用的訊息」問答) 。

在一個大量使用者的 Web 應用程式環境中, 不論是 CPU/RAM 或者是網路頻寬都是要考量的重點, 例如以一個 POS 系統來說, 最重要的就是即時存取的速度, 像是在結帳或是刷條碼的時候, 系統的回應速度會變的很重要, 這些都需要納入考量, 像是在用戶端快取一份清單, 或者是以 XML 加上 Ajax 通訊方式, 這時在 Server 上面就不一定要暫存資料, 而是以快速的 DataReader 來產生清單的 XML。

但在像是較大量的資料庫存取的環境中, 快取資料集 (DataSet) 可以減輕資料庫的負擔, 也可以幫助資料庫分散流量, 只有寫入和少量存取需要連接到資料庫, 並且可以實作以資料集爲主的複寫機制, 將資料寫入不同的資料庫, 或者是不同的資料來源等等。

因此 DataSet/DataTable 和 DataReader, 只有適不適用的問題, 沒有用哪種較好的問題, 將正確的物件放在正確的位置上, 不論哪一種都是好的, 反之則都是壞的。

解決方案

要視應用程式的設計以及實際預估 (或執行時) 的資料負載量, 來決定要使用哪一種資料存取物件, 可多方參考一些相關文章或者是他人的設計 (Best Practices), 來輔助在選擇物件上的考量。

功能比較	DataTable/DataSet	DataReader
由資料庫讀取資料的速度	較慢 (由 DataReader 寫入 DataTable 物件) 。	較快。
巡覽資料	方便, 透過 DataTable.Rows 即可來回的巡覽。	不方便, 只能順向讀取。
記憶體耗用	大, 因爲資料都放在記憶體中。	小, 只儲存資料游標。
修改資料	可直接編輯資料, 再經由 DataAdapter.Update()寫入資料庫。	無法編輯資料, 若要修改則需另外產生 SQL 指令, 並由 SqlCommand.ExecuteNonQuery() 執行。
適用情況	● 需實作 Master/Detail 功能時。 ● 需減輕資料庫負擔時。 ● 需實作快取功能者。 ● 小量資料存取時。 ● 需經常的來回瀏覽資料時。	● 大量資料讀取時。 ● 伺服器效能受限, 需要節省資源耗用時。 ● 不需來回瀏覽資料時。 ● 需要高效率資料讀取時。

考生停看聽

本問題所討論的內容, 可以用來準備 Exam 70-547: PRO: Designing and Developing Web Applications by using Microsoft .NET Framework

- 設計與開發元件 (Components)
  - 開發元件的資料存取與資料處理功能。
    - 分析元件的資料處理需求。
- 設計與開發應用程式機能 (Application Framework)
  - 根據應用程式的設計邏輯, 決定適合的實作方法。
    - 選擇合適的資料儲存機制。

相關問題

- Q31：如何利用離線資料存取 (Disconnected Data Access) 提昇資料庫存取效率？
- Q33：執行 ExecuteNonQuery()時都會顯示連線被佔用的訊息？

Q33 執行 ExecuteReader()時都會顯示連線被佔用的訊息？

適用範圍：V ASP.NET 1.0 V ASP.NET 1.1 V ASP.NET 2.0 V ASP.NET 3.5

問題

我在程式中撰寫了利用 ADO.NET 存取 SQL Server 資料庫的程式碼, 因爲某些需求, 我需要在資料庫更新後讀取資料, 然後再做下一步的更新, 例如像這樣子的程式碼：

```
SqlConnection conn = new SqlConnection("…");
SqlCommand cmd = new SqlCommand("…", conn);
SqlCommand cmd2 = new SqlCommand("…", conn);

conn.Open();

dr = cmd.ExecuteReader();
…
cmd2.ExecuteNonQuery();
```

但是每次我執行時, 都會出現這個例外：

```
已經開啓一個與這個 Command 相關的 DataReader, 必須先將它關閉。
```

請問我要如何才能夠實作出我想要的那種功能？

問題說明

DataReader 是一種會佔用連線的資料存取物件, 它依賴伺服器端 (資料庫) 的 Server-side Cursor (伺服端游標) 來識別資料列的位置, 並且以順向讀取的方式向資料庫要求資料, 在 DataReader 開啓的期間, 連線會被鎖定, 只允許一個 DataReader 存在, 而其他的物件將會無法使用, 因爲像 ExecuteScalar()和 ExecuteNonQuery() 的內部都會使用到 DataReader。

在這種情況下, 需要另外開啓連線, 才可以被其他資料存取物件使用, 例如這樣的方法：

```
string strConnectionString =
  "initial catalog=Northwind; Integrated Security=SSPI";

// 讀取資料的連線
SqlConnection conn1 = new SqlConnection(strConnectionString);
SqlCommand cmd1 = new SqlCommand("…", conn1); // 讀取資料的指令

// 寫入資料的連線
SqlConnection conn2 = new SqlConnection(strConnectionString);
SqlCommand cmd2 = new SqlCommand("…", conn2); // 寫入資料的指令

conn1.Open();
conn2.Open();

SqlDataReader dr =
  cmd1.ExecuteReader(CommandBehavior.CloseConnection);

while (dr.Read())
{
```

```
  string sql = null;
  … // 建立寫入的 SQL 指令。
  cmd2.CommandText = sql;
  cmd2.ExecuteNonQuery();
}
conn2.Close();
dr.Close();
```

以上的程式碼適用於各類型的資料庫, 而且這也是處理這類型問題的普遍做法, 但若使用的是 SQL Server 2005 或 SQL Server Express 資料庫時, 可以利用一個更簡單的作法：MARS (Multiple Active Result Set) , 它可以允許在同一個連線中, 開啓多個 DataReader, 也可以在 DataReader 開啓的時候, 執行 SQL 指令, 方法很簡單, 只要在連線字串中加上這一個陳述：

```
string strConnectionString = "initial catalog=Northwind; Integrated
Security=SSPI; MultipleActiveResultSets=true";
SqlConnection conn = new SqlConnection(strConnectionString); // 連線
SqlCommand cmd1 = new SqlCommand("…", conn); // 讀取資料的指令
SqlCommand cmd2 = new SqlCommand("…", conn); // 寫入資料的指令

conn.Open();

SqlDataReader dr =
  cmd1.ExecuteReader(CommandBehavior.CloseConnection);

while (dr.Read())
{
```

```
    string sql = null;
    … // 建立寫入的 SQL 指令
    cmd2.CommandText = sql;
    cmd2.ExecuteNonQuery();
}

dr.Close();
```

就可以在只使用單一連線的情況下, 開啓 DataReader 與執行 SQL 指令。

不過 MARS 必須要小心使用, 尤其是在多人環境下的並行存取, 很容易出現交易鎖定 (locking) 或是並行處理問題, 此時可能要以像是快照式隔離 (Snapshot Isolation) 的隔離層級 (Isolation Level) 來做處理, 避免發生交易與並行鎖定的問題。

解決方案

若是 SQL Server 2005 資料庫, 請在連線字串中加上 MultipleActive ResultSets=true, 以啓用 MARS 功能, 若不是, 則只能利用兩個以上的連線來處理個別的資料存取工作。

小常識　隔離層級 (Isolation Level)

隔離層級是指資料庫在執行交易時, 對於異動的資料列的鎖定機制與版本控制機制, 這會在多人同時存取的並行處理 (Concurrency Processing) 時, 對於資料列存取採取一些行動, 而不同的行動可能會造成一些不同的影響, 像是 Dirty Read；幽靈資料 (Phantom) 或是不可重覆讀取 (Unrepeatable Read) 的狀況, 而這些狀況也會影響到後續的資料存取, 在 SQL Server 與 ADO.NET 中, 預設的隔離層級是 Read Commited, 而最穩固的方法是 Serializable 與 Snapshot (SQL Server 2005 新增) 的隔離層級。

目前在 SQL Server 中支援的隔離層級有：

- Read Uncommitted：允許其他交易在目前交易未完成前讀取資料, 但會有 Dirty Read (骯髒讀取, 即每個使用者讀到的數值可能會不相同) 的問題。

接下頁

- Read Committed：這是 SQL Server 的預設值, 不允許其他交易在目前交易未完成前讀取資料, 但會出現 Unrepeatable Read (不可重覆讀取, 即讀出的資料無法再由資料庫讀取) 或 Phantom (幽靈資料, 即原先讀取到的資料在資料庫中不存在) 的異常狀況。
- Repeatable Read：不允許其他使用者在交易未完成前讀取資料, 亦不允許其他交易在目前交易完成前修改資料, 但因爲 Repeatable Read 使用共用鎖定 (Shared Lock) , 若其他交易使用的查詢符合目前交易的限制, 則仍可以新增資料列, 導致可能的 Phantom Read (幽靈資料讀取) 問題。
- Serializable：不允許其他查詢讀取目前交易未認可的資料, 其他交易不允許修改目前交易已讀取的資料, 亦不可以插入新資料列到目前交易已讀取索引所在的資料表。Serializable 會使用獨佔鎖定 (Exclusive Lock) , 因此不會有異常狀況發生, 但會讓鎖定時間拉長, 並且可能造成死結 (Deadlock) 或飢餓 (Starvation) 現象。
- Snapshot：在交易啓始時, 將交易影響範圍的資料列和表格在記憶體中產生一份快照集, 交易依照這個快照集來修改資料, 交易完成時以比對資料列版本的方式寫回資料表, 此種隔離層級除了快照寫入時期外, 不會鎖定資料表和資料列。而因爲目前交易的資料是來自快照集, 所以也不會有資料表獨佔鎖定問題, 其他交易也不會發生 Phantom 和 Unrepeatable Read 現象, 其他使用者亦不會發生 Dirty Read 現象。

若想知道更多並行處理和隔離層級的相關知識, 可查閱資料庫系統的書籍。

Q34 如何在 DataList 中取得 ItemTemplate 中的控制項？

適用範圍： V ASP.NET 1.0 V ASP.NET 1.1 V ASP.NET 2.0 V ASP.NET 3.5

問題

我使用 ASP.NET 2.0 開發網站, 透過 DataList 來實作公布欄的功能, 我在 ItemTemplate 中加入一些我想要顯示的資料, 並且用資料繫結的方式呈現資料, 但現在有個問題, 因爲我的 ItemTemplate 中有放一個按鈕, 這個按鈕會視會員的等級來決定要不要禁用 (等級不夠的就要禁用) , 那我要如何做這樣的功能？

問題說明

DataList 是個很簡單的資料控制項, 在某些實作案例中, 它還比 GridView 或 DataGrid 更好用, 除了分頁沒辦法做到, 要人工來做以外, 和 GridView 或 DataGrid 有著幾乎相同的操作能力, 而且它完全是樣板化的控制項, 可以由開發人員自己訂定顯示的畫面, 以下程式就是簡單的 Data List 範例。

```
<asp:DataList ID="myDataList" DataSourceID="myDS" Width="100%"
  runat="server">
  <ItemTemplate>
    <asp:Label ID="labelCaption" runat="server"
      Text='<%# Eval("CompanyName") %>' /><br /><br />
    <asp:LinkButton ID="cmdViewOrders" runat="server"
      Text="瀏覽訂單..." />
  </ItemTemplate>
</asp:DataList>
```

DataList 也提供了在執行時期, 依照特定條件來修改樣板的能力, 比較多的應用是依照資料來決定某些控制項的狀態或是數值內容， 此時就可以用 ItemDataBound 事件來實作, 這個事件會在每個資料列完成資料繫結時引發, 而開發人員可以透過處理這個事件, 來對樣板內容的控制項做修改。

首先, 要利用 FindControl() 來取得控制項的物件, 然後才能夠控制及修改它的屬性, 接著就是取得資料, 以及檢查資料是否符合條件 (必要時還要再執行查詢), 然後再修改控制項的資訊 (屬性) 即可, 如程式 1 的實作。

程式1：DataList 的 ItemDataBound 事件常式範例實作

```
protected void myDataList_ItemDataBound(object sender,
  DataListItemEventArgs e)
{
  if (e.Item.ItemType == ListItemType.Item ||
      e.Item.ItemType == ListItemType.AlternatingItem)
  {
    // 取得繫結在這一列的 DataRow
    DataRow row = ((DataRowView)e.Item.DataItem).Row;
    // 取得要修改屬性的連結控制項。
    LinkButton cmdViewOrders =
      e.Item.FindControl("cmdViewOrders") as LinkButton;
    // 設定查詢參數
    if (this.myOrderDS.SelectParameters["customerID"] == null)
      this.myOrderDS.SelectParameters.Add(
        "customerID", row["CustomerID"].ToString());
    else
      this.myOrderDS.SelectParameters["customerID"].DefaultValue=
        row["CustomerID"].ToString();
    // 執行查詢。
    IDataReader reader = this.myOrderDS.Select(
      new DataSourceSelectArguments()) as IDataReader;

    reader.Read();
```

```
    // 檢查資料值, 若為 0 則將連結關閉
    if (0 == Convert.ToInt32(reader.GetValue(0)))
      cmdViewOrders.Enabled = false;
    else
      cmdViewOrders.Enabled = true;

    reader.Close();
  }
}
```

另一個常用的事件則是 ItemCreated 事件, 這個事件和 ItemDataBound 很類似, 都可以取出繫結在資料列中的資料, 最主要的差別, 則在於 ItemCreatd 事件無法改變控制項的屬性, 但可以改變資料 (例如 LinkButton.Text 屬性) ; ItemDataBound 則是控制項屬性和資料都可以改變, 如程式 2 的實作。

程式 2：DataList 的 ItemCreated 事件範例實作

```
protected void myDataList_ItemCreated(object sender,
  DataListItemEventArgs e)
{
  if (e.Item.ItemType == ListItemType.Item ||
      e.Item.ItemType == ListItemType.AlternatingItem)
  {
    DataRow row = ((DataRowView)e.Item.DataItem).Row;
    LinkButton cmdViewOrders = e.Item.FindControl("cmdViewOrders")
      as LinkButton;

    if (this.myOrderDS.SelectParameters["customerID"] == null)
      this.myOrderDS.SelectParameters.Add(
        "customerID", row["CustomerID"].ToString());
    else
```

```
        this.myOrderDS.SelectParameters["customerID"].DefaultValue =
          row["CustomerID"].ToString();

      IDataReader reader = this.myOrderDS.Select(
        new DataSourceSelectArguments()) as IDataReader;

      reader.Read();

      cmdViewOrders.Text = "訂單數 : " +
        Convert.ToInt32(reader.GetValue(0)).ToString();

      reader.Close();
    }
  }
```

讀者可以把 ItemDataBound 的指令做註解, 然後執行範例程式, 接著把 ItemDataBound 的註解取消, 再執行一次, 就可以比較出兩者之間的差異了。

解決方案

可在 ItemDataBound 事件中加入處理常式, 然後根據繫結的資料來決定按鈕的禁用即可, 但不可放在 ItemCreated 事件, 因爲會無法設定控制項的屬性。

考生停看聽

本問題所討論的內容, 可用來準備 Exam 70-528: TS: Microsoft .NET Framework 2.0 Web Client Development 的下列主題。

- 使用 ADO.NET 將資料整合到 Web 應用程式中。
  - 實作 Data-Bound (資料繫結) 控制項
    - 使用表格化資料來源控制項回傳表格資料。
    - 使用簡單資料繫結控制項來顯示資料。
    - 使用複合資料繫結控制項來顯示資料。

相關問題

- Q35：如何在 GridView 中動態加入控制項？

Q35 如何在 GridView 中動態加入控制項？

適用範圍：VASP.NET 2.0 VASP.NET 3.5

問題

我使用 ASP.NET 2.0 開發網站, 並且使用 GridView 來實作會員瀏覽的網頁, 但是我想要在會員有訂單時可以有一個介面來瀏覽訂單資訊, 因此想在GridView 中動態加入按鈕來實作, 請問我要如何做呢？

問題說明

GridView 的能力可以說是所有 ASP.NET 2.0 內建的資料繫結控制項中最強的, 它除了支援完整的樣板化能力外, 還具備排序和分頁的能力, 而且分頁也支援樣板化 (請參閱「Q45：如何像是 Google 那樣分頁顯示查詢結果？」問答) , 因此可以說是功能最強的資料控制項了。

本問題有兩種解法, 一種是直接在樣板中套用按鈕, 然後再控制它的 Visible 屬性來決定是否要顯示, 這是比較簡單的作法；另一種則是在決定需要按鈕時, 再動態加入控制項。在大多數的情況, 其實只要用前面那一種就可以了, 除非有特殊的情形, 才會需要用動態的方式加入控制項。

在GridView 中若要對列做資料存取以及控制項操作, 必須要在RowDataBound 事件中處理, 這點和 DataList 不太一樣, 但其實也就只有這點不同, 在 RowDataBound 的事件處理常式中, 仍然可以對樣板控制項做處理, 以及進一步的運算等工作。

筆者在「Q24：如何動態產生控制項」問答中說明了如何動態生成控制項以及如何處理控制項的事件, GridView 中的動態控制項生成大致也差不多, 但有一個地方不同, 就是按鈕觸發的事件, 都要在 GridView 的 RowCommand 事件來處理, 所以在動態生成控制項時, 務必要設定 CommandName 屬性, 以及需要的參數到 CommandArgument 屬性中, ID 可以不必設。

還有一點要注意的, 動態生成控制項必須要在每次頁面載入時都要產生, 所以 GridView 的資料必須持續的做 DataBind(), 才能夠保持控制項的狀態以及事件的呼叫程序。

有了這些知識, 就可以解這個問題了。

解決方案

範例的網頁如程式 1 所示。

程式1：範例網頁

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>在 GridView 中動態生成控制項</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    <asp:GridView ID="CustomerDataGrid" AutoGenerateColumns="false"
      DataSourceID="myDS" runat="server" CellPadding="4"
      ForeColor="#333333" OnRowCommand="CustomerDataGrid_RowCommand"
      OnRowDataBound="CustomerDataGrid_RowDataBound">
      <FooterStyle BackColor="#990000" Font-Bold="True"
        ForeColor="White" />
      <RowStyle BackColor="#FFFBD6" ForeColor="#333333" />
      <SelectedRowStyle BackColor="#FFCC66" Font-Bold="True"
        ForeColor="Navy" />
      <PagerStyle BackColor="#FFCC66" ForeColor="#333333"
        HorizontalAlign="Center" />
      <HeaderStyle BackColor="#990000" Font-Bold="True"
        ForeColor="White" />
```

```
      <AlternatingRowStyle BackColor="White" />
      <Columns>
        <asp:BoundField DataField="CustomerID" />
        <asp:BoundField DataField="CompanyName" />
      </Columns>
    </asp:GridView>
    <asp:SqlDataSource ID="myDS" DataSourceMode="DataSet"
      ConnectionString=
        "<%$ ConnectionStrings: northwindConnectionString %>"
      runat="server" SelectCommand="SELECT * FROM Customers">
    </asp:SqlDataSource>
    <asp:SqlDataSource ID="myOrderDS" DataSourceMode="DataReader"
      ConnectionString=
      "<%$ ConnectionStrings: northwindConnectionString %>"
     runat="server" SelectCommand="SELECT ISNULL(COUNT(OrderID), 0) FROM
Orders WHERE CustomerID = @customerID">
    </asp:SqlDataSource>
  </div>
  </form>
</body>
</html>
```

首先, 先將正常的資料繫結實作好, 並且在 Page_Load 事件中加入資料主動繫結的指令, 確保每次頁面載入時都會繫結資料。

```
protected void Page_Load(object sender, EventArgs e)
{
  this.CustomerDataGrid.DataBind();
}
```

接著, 實作 RowDataBound 事件常式, 將要加入的動態生成控制項指令加進來, 並設定 CommandName 與 CommandArgument 等必要參數, 如程式 2。

程式 2：RowDataBound 事件常式實作

```
protected void CustomerDataGrid_RowDataBound(object sender,
  GridViewRowEventArgs e)
{
  if (e.Row.RowType == DataControlRowType.DataRow)
  {
    DataRow row = ((DataRowView)e.Row.DataItem).Row;
    // 查詢資料。
    if (this.myOrderDS.SelectParameters["customerID"] == null)
      this.myOrderDS.SelectParameters.Add(
        "customerID", row["CustomerID"].ToString());
    else
      this.myOrderDS.SelectParameters["customerID"].DefaultValue =
        row["CustomerID"].ToString();

    IDataReader reader = this.myOrderDS.Select(
      new DataSourceSelectArguments()) as IDataReader;

    reader.Read();

    if (0 < Convert.ToInt32(reader.GetValue(0)))
    {
      Label labelCaption = new Label();  // 生成 Label 控制項
      labelCaption.Text = e.Row.Cells[1].Text;
      e.Row.Cells[1].Controls.Add(labelCaption); // 加到指定的 Cell 中
```

```
      // 生成 LinkButton 控制項
      LinkButton cmdViewOrders = new LinkButton();
      cmdViewOrders.Text = "ViewOrders"; // 設定文字
      cmdViewOrders.CommandName = "ViewOrders"; // 設定 CommandName
      // 設定參數
      cmdViewOrders.CommandArgument = row["CustomerID"].ToString();
      // 加到指定的 Cell 中
      e.Row.Cells[1].Controls.Add(cmdViewOrders);
    }

    reader.Close();
  }
}
```

接著, 實作 RowCommand 事件常式, 以接收來自 LinkButton 的事件, 如程式 3 。

程式 3：RowCommand 事件常式實作

```
protected void CustomerDataGrid_RowCommand(object sender,
  GridViewCommandEventArgs e)
{
  if (e.CommandName == "ViewOrders") // 檢查指令名稱。
  {
    Response.Write(e.CommandArgument.ToString());
  }
}
```

如此, 具動態生成控制項能力的 GridView 就完成了。

考生停看聽

本問題所討論的內容, 可用來準備 Exam 70-528: TS: Microsoft .NET Framework 2.0 Web Client Development 的下列主題。

- 使用 ADO.NET 將資料整合到 Web 應用程式中。
  - 實作 Data-Bound (資料繫結) 控制項
    - 使用表格化資料來源控制項回傳表格資料。
    - 使用簡單資料繫結控制項來顯示資料。
    - 使用複合資料繫結控制項來顯示資料。

相關問題

- Q34：如何在 DataList 中取得 ItemTemplate 中的控制項？

Q36 如何在 GridView 沒資料時顯示提示訊息？

適用範圍：VASP.NET 2.0 VASP.NET 3.5

問題

我使用 ASP.NET 2.0 開發網站, 透過 GridView 來實作會員瀏覽的網頁, 並且有撰寫搜尋功能, 但是有個問題, 就是當搜尋沒有找到資料時, GridView 就會消失, 我要如何讓 GridView 提示找不到資料的訊息呢？

問題說明

這個問題的詢問率也不低, 因爲在 Visual Studio .NET 2003 時代的 DataGrid, 就算沒有資料, DataGrid 也會顯示標題而不會消失, 但 GridView 在沒有資料時, 連標題都不會顯示, 這樣的改變其實不太方便, 因爲會讓人認爲系統沒有處理搜尋的指令, 或是操作有錯誤。

不過也不是沒有辦法讓 GridView 在沒有資料時顯示, GridView 提供了 EmptyDataText 以及 EmptyDataTemplate 兩個方法來設定無資料時的訊息, 若只要顯示文字訊息, 則可以利用 EmptyDataText, 而若要顯示圖文並茂的訊息, 就要利用 EmptyDataTemplate, 筆者建議使用 EmptyDataTemplate, 除了可以顯示訊息以外, 還可以額外加入一些協助使用者的資訊, 方便使用者進行下一步動作。

不過美中不足的是, 在無資料的情況下, 無法將標題顯示出來。

解決方案

設定 GridView 的 EmptyDataText 或 EmptyDataTemplate 來顯示無任何資料時的訊息。

考生停看聽

本問題所討論的內容, 可用來準備 Exam 70-528: TS: Microsoft .NET Framework 2.0 Web Client Development 的下列主題。

- 使用 ADO.NET 將資料整合到 Web 應用程式中。
  - 實作 Data-Bound (資料繫結) 控制項
    - 使用表格化資料來源控制項回傳表格資料。
    - 使用簡單資料繫結控制項來顯示資料。
    - 使用複合資料繫結控制項來顯示資料。

Q37 如何讓下拉式方塊的內容可以依據另一個下拉式方塊的選取結果而變化？

適用範圍：V ASP.NET 1.0 V ASP.NET 1.1 V ASP.NET 2.0 V SQL Server 2005

問題

我想要實作一個輸入地址的工具, 並且想要利用二個下拉式方塊來提供使用者輸入縣市與鄉鎮的資料, 但我要如何在使用者選取縣市後才列出對應的鄉鎮資料？

問題說明

在很多資料編輯工具中 (尤其是像客戶或廠商這類的輸入表單) , 都可以看到地址的輸入欄位, 如果是對資料輸入與資料正確性較重視的公司, 會要求使用下拉式的方塊 (DropDownList) 來輸入縣市與鄉鎮, 以核對郵遞區號是否正確 (或自動對應郵遞區號) , 而通常都是由使用者先選擇縣市後才會將鄉鎮的資料填入, 因此會需要兩個以上的下拉式方塊做連動的處理, 具有這種能力的下拉式方塊稱爲 Cascading DropDownList (連動型下拉式方塊) 。

在 ASP 的時代, 若要實作連動型的下拉式方塊, 則必須要使用 JavaScript 先將資料輸出後, 再用 JavaScript 來操作下拉式方塊並填入資料。在 ASP.NET 1.x 與 2.0 時, 連動型下拉式方塊則是要經過一次 PostBack, 就可以產生連動的效果, 但會讓畫面重刷新一次, 在 Web 2.0 的規格下, 這似乎也不是最好的做法, 因此 ASP.NET AJAX Control Toolkit 中, 另外提供了一個 CascadingDropDown 控制項, 讓 ASP.NET 2.0 的開發人員得以應用它輕鬆的發展連動型下拉式方塊。

以往使用 ASP.NET 做伺服端處理的連動型下拉式方塊, 其實也是很簡單的事, 只要將第一個 DropDownList 設定 AutoPostBack="true", 處理伺服端的 SelectedIndexChanged 事件, 在事件常式中填入第二個 DropDownList 的資料即可。

```
<!-- in ASP.NET HTML code -->
<asp:DropDownList ID="cboCity" runat="server" AutoPostBack="true"
  OnSelectedIndexChanged="cboCity_SelectedIndexChanged"
  DataTextField="City" DataValueField="City" />
<asp:DropDownList ID="cboDistrict" runat="server"
  DataTextField="District" DataValueField="PostalCode" />

// in Code-behind CS code:
private void cboCity_SelectedIndexChanged(object sender, EventArgs e)
{
  // 取得第一個 DropDownList 的選取值。
  string city = this.cboCity.SelectedValue;
  // 搜尋資料庫, 取回第二個 DropDownList 的資料。
  DataTable districtTable = PostalCodeProvider.GetByCity(city);
  // 將資料填入第二個 DropDownList
  this.cboDistrict.DataSource = districtTable;
  this.cboDistrict.DataBind();
}
```

不過ASP.NET AJAX Control Toolkit所提供的CascadingDropDown控制項, 可以預防伺服端處理連動型下拉式方塊所造成的閃動問題, 因此本問題以CascadingDropDown控制項爲主要說明方向。

解決方案

首先, 準備好地址資料庫 (或資料表) , 如果沒有, 可以到郵局的網站 (http://www.post.gov.tw) 中, 下載 3+2 郵遞區號程式, 或者是 XML 資料檔, 本範例以 XML 資料檔來實作, 但是最好是能將這些資料放入資料庫, 速度會比 XML 資料檔快很多。

接著, 實作一個提供資料給 CascadingDropDown 控制項的 Web Service, 這個 Web Service 可以是新的也可以是現有的, 但是一定要在 Web Service 上宣告 System.Web.Script.Service.ScriptService, 否則 AJAX 會傳回「方法錯誤 500」的訊息。

Web Service 的方法必須要和 CascadingDropDown 控制項預設呼叫的方法引數要相同, 否則也會傳回「方法錯誤 500」的訊息。CascadingDropDown 控制項所要求的方法引數爲：

```
public CascadingDropDownNameValue[] method(
  string knownCategoryValues, string category)
```

其中：

引數	說明
knowCategoryValues	前一個 CascadingDropDownList 的選取值。
category	呼叫這個 WebMethod 的 CascadingDropDownList 的 Category 值。

除了方法名稱 method 可以改變外, 其他的回傳值及參數個數都不能改變, 否則 CascadingDropDown 控制項會抓不到方法而傳回錯誤。

同時爲了要防止資料溢位 (Overflow) 錯誤, 必須要擴充 Web.config 中, ASP.NET AJAX 的 jsonSerialization 的 maxJsonLength 屬性, 以確保大量資料的穩定性, 預設是 500, 筆者實作這個功能時, 將這個大小擴增到 50000, 當然讀者也可以設的更高一些。

接著, 就可以來撰寫處理資料的 Web Service 的程式碼了, 由 zip32_9605.xml (郵局網站下載的 XML 郵遞區號資料) 讀出資料, 它的結構是這樣的：

```
<NewDataSet>
  <zip32>
    <zipcode>10058</zipcode>
    <city>台北市</city>
    <area>中正區</area>
    <road>八德路 1 段</road>
    <scoop>全</scoop>
  </zip32>
  ....
</NewDataSet>
```

結構很簡單, 但由於重覆資料較多 (尤其是縣市和鄉鎮市區) , 因此需要在讀取資料時過濾掉多餘的資料, 也就是類似於 SELECT DISTINCT 的作法, 取出縣市的程式碼如程式 1 所示。

程式 1：取得縣市資料的 GetCityInfoList()方法

```
[WebMethod]
public CascadingDropDownNameValue[] GetCityInfoList(
  string knownCategoryValues, string category)
{
  XmlDocument doc = new XmlDocument();
  doc.Load(Server.MapPath("zip32_9605.xml")); // 讀取 XML

  Dictionary<string, CascadingDropDownNameValue> valueList =
    new Dictionary<string, CascadingDropDownNameValue>();
  XmlNodeList nodes = doc.SelectNodes("//NewDataSet/zip32/city");

  foreach (XmlNode node in nodes)
  { // 檢查重覆
    if (!valueList.ContainsKey(node.InnerText.Trim()))
```

```
    {
      CascadingDropDownNameValue v =
        new CascadingDropDownNameValue();
      v.name = node.InnerText.Trim();
      v.value = node.InnerText.Trim();

      valueList.Add(node.InnerText.Trim(), v);
    }
  }

  doc = null;
  nodes = null;

  // 產生 CascadingDropDownNameValue, 供 CascadingDropDown 控制項使用
  CascadingDropDownNameValue[] data =
    new CascadingDropDownNameValue[valueList.Count];
  valueList.Values.CopyTo(data, 0);
  valueList = null;
  return data;
}
```

若是要負責接取前一個 DropDownList 傳回值的方法, 則要多一個解析 knownCategoryValues 的步驟, 以取出必要的條件資料, 其實解析也不難, 只是要多呼叫一個CascadingDropDown.ParseKnownCategoryValuesString()方法, 這個方法會傳回StringDictionary類別(在System.Collection.Specialized命名空間中) , 並且裝載了 knownCategoryValues 內含的資料, 格式爲[category]:[data], 所以只要使用 StringDictionary[category]就可以取得前一個(或前幾個) DropDownList 的資料了。其程式碼如程式 2 所示, 程式 3 則展示了檢查多個 DropDownList資料的方法。

程式 2：負責取出鄉鎮市區的 GetDistrictList()方法

```
[WebMethod]
public CascadingDropDownNameValue[] GetDistrictList(
  string knownCategoryValues, string category)
{
  // 解析 KnownCategoryValues 資料
  StringDictionary sd =
    CascadingDropDown.ParseKnownCategoryValuesString(
      knownCategoryValues);

  if (!sd.ContainsKey("City")) // 檢查有沒有存在縣市的資料
    return null;

  XmlDocument doc = new XmlDocument();
  doc.Load(Server.MapPath("zip32_9605.xml"));

  Dictionary<string, CascadingDropDownNameValue> valueList =
    new Dictionary<string, CascadingDropDownNameValue>();
  XmlNodeList nodes = doc.SelectNodes("//NewDataSet/zip32");

  foreach (XmlNode node in nodes)
  { // 檢查是否符合由 City 的下拉式方塊所選的縣市
    if (node.SelectSingleNode("city").InnerText == sd["City"])
    {
      if (!valueList.ContainsKey(
        node.SelectSingleNode("area").InnerText.Trim()))
      {
        CascadingDropDownNameValue v =
          new CascadingDropDownNameValue();
        v.name = node.SelectSingleNode("area").InnerText.Trim();
        v.value = node.SelectSingleNode("area").InnerText.Trim();
```

```
        valueList.Add(
          node.SelectSingleNode("area").InnerText.Trim(), v);
      }
    }
  }

  doc = null;
  nodes = null;

  CascadingDropDownNameValue[] data =
    new CascadingDropDownNameValue[valueList.Count];
  valueList.Values.CopyTo(data, 0);
  valueList = null;
  return data;
}
```

程式 3：負責取回路街資料的 GetRoadStreetList() 方法

```
[WebMethod]
public CascadingDropDownNameValue[] GetRoadStreetList(
  string knownCategoryValues, string category)
{
  StringDictionary sd =
    CascadingDropDown.ParseKnownCategoryValuesString(
      knownCategoryValues);

  if (!sd.ContainsKey("District"))
    return null;

  XmlDocument doc = new XmlDocument();
  doc.Load(Server.MapPath("zip32_9605.xml"));
```

```
  Dictionary<string, CascadingDropDownNameValue>valueList =
    new Dictionary<string, CascadingDropDownNameValue>();
  XmlNodeList nodes = doc.SelectNodes("//NewDataSet/zip32");

  foreach (XmlNode node in nodes)
  { // 多檢查一道鄉鎮市區的資料。
    if (node.SelectSingleNode("city").InnerText == sd["City"] &&
        node.SelectSingleNode("area").InnerText == sd["District"])
    {
      if (!valueList.ContainsKey(
        node.SelectSingleNode("road").InnerText.Trim()))
      {
        CascadingDropDownNameValue v =
          new CascadingDropDownNameValue();
        v.name = node.SelectSingleNode("road").InnerText.Trim();
        v.value = node.SelectSingleNode("road").InnerText.Trim();

        valueList.Add(
          node.SelectSingleNode("road").InnerText.Trim(), v);
      }
    }
  }

  doc = null;
  nodes = null;

  CascadingDropDownNameValue[] data =
    new CascadingDropDownNameValue[valueList.Count];
  valueList.Values.CopyTo(data, 0);
  valueList = null;
  return data;
}
```

最後再實作出取回 Postal Code 的程式, Web Service 的部份就完成了, 如程式 4 所示。

程式 4：取回郵遞區號的 GetPostalCode() 方法

```
[WebMethod]
public string GetPostalCode(string City, string District,
  string Road)
{
  XmlDocument doc = new XmlDocument();
  doc.Load(Server.MapPath("zip32_9605.xml"));
  string postalCode = null;

  XmlNodeList nodes = doc.SelectNodes("//NewDataSet/zip32");

  foreach (XmlNode node in nodes)
  {
    if (node.SelectSingleNode("city").InnerText.Trim() ==
          City.Trim() &&
        node.SelectSingleNode("area").InnerText.Trim() ==
          District.Trim() &&
        node.SelectSingleNode("road").InnerText.Trim() == Road.Trim())
    {
      postalCode += (postalCode == null)
          ? node.SelectSingleNode("zipcode").InnerText + "-" +
            node.SelectSingleNode("scoop").InnerText.Trim()
          : ", " + node.SelectSingleNode("zipcode").InnerText + "-" +
            node.SelectSingleNode("scoop").InnerText.Trim();
    }
  }
```

```
  doc = null;
  nodes = null;

  return postalCode;
}
```

Web Service 完成後, 就可以實作畫面的部份了, 由於使用 CascadingDropDown 控制項, 所以伺服端的程式很少, 而 HTML 的部份也需要做一些設定, 如程式 5 所示。

程式 5：郵遞區號查詢程式的 ASP.NET 頁面 HTML 碼

```
<form id="form1" runat="server">
<ajaxToolkit:ToolkitScriptManager runat="server"
  ID="toolkitScriptManager" EnablePartialRendering="true" />
<div>
  縣市：
  <asp:DropDownList ID="cboCity" DataTextField="City"
    DataValueField="City" runat="server" />
  , 鄉鎮市區：<asp:DropDownList ID="cboDistrict" runat="server" />
  , 路街：<asp:DropDownList ID="cboRoadStreet" AutoPostBack="True"
    runat="server"
    OnSelectedIndexChanged="cboRoadStreet_SelectedIndexChanged" />
  <ajaxToolkit:CascadingDropDown ID="cddCity" runat="server"
    TargetControlID="cboCity" PromptText="請選擇縣市" PromptValue=""
    ServicePath="CascadingDropDownService.asmx"
    ServiceMethod="GetCityInfoList" Category="City" />
  <ajaxToolkit:CascadingDropDown ID="cddDistrict" runat="server"
    TargetControlID="cboDistrict" ParentControlID="cboCity"
    Category="District" PromptText="請選擇鄉鎮市區" PromptValue=""
    ServicePath="CascadingDropDownService.asmx"
    ServiceMethod="GetDistrictList" />
```

```
  <ajaxToolkit:CascadingDropDown ID="cddRoadStreet" runat="server"
    PromptText="請選擇路街" PromptValue=""
    TargetControlID="cboRoadStreet"
    ParentControlID="cboDistrict" Category="RoadStreet"
    ServicePath="CascadingDropDownService.asmx"
    ServiceMethod="GetRoadStreetList" />
  <asp:UpdatePanel ID="upShow" runat="server">
     <ContentTemplate>
       <asp:Label ID="labelPostalCode" runat="server">
        [請選擇縣市, 鄉鎮及路街]</asp:Label>
     </ContentTemplate>
     <Triggers>
       <asp:AsyncPostBackTrigger ControlID="cboRoadStreet" />
     </Triggers>
  </asp:UpdatePanel>
</form>
```

其中CascadingDropDown有幾個屬性需要介紹：

TargetControlID	要套用的控制項 ID 。
ParentControlID	父控制項的 ID，表示由哪個控制項連動，若無則不必設 。
Category	資料的名稱, 會套用在存取 Web Service 時的 Known Category Values 的 key 中。
PromptText與PromptValue	表示下拉式方塊第一個項目的文字和值 。
ServicePath	CascadingDropDown 要存取的 Web Service 的名稱。如果設爲 null 則表示使用 Page 提供的方法。
ServiceMethod	CascadingDropDown 要存取的 Web Service 中, 用來填入資料的 Web 方法, 此方法必須回傳 CascadingDropDownNameValue[]陣列。

最後, 由最後一個 DropDownList 控制項觸發 PostBack, 取出郵遞區號的資料, 並且在網頁上布置了 UpdatePanel 控制項, 可防止 PostBack 的閃動, 這也是這個範例唯一會寫在頁面中的程式碼, 如程式 6。

程式6：取回郵遞區號的程式

```
protected void cboRoadStreet_SelectedIndexChanged(object sender,
  EventArgs e)
{
  CascadingDropDownService service = new CascadingDropDownService();
  this.labelPostalCode.Text = service.GetPostalCode(
    this.cboCity.SelectedValue,
   this.cboDistrict.SelectedValue,
   this.cboRoadStreet.SelectedValue);
  service = null;
}
```

相關問題

有關組態設定問題, 可參考：

- Q12：如何在將 ASP.NET AJAX 整合到現有的 ASP.NET 2.0 應用程式中？

Q38 使用 SqlDataSource 時如何取得資料筆數、新增資料的自動編號欄位值、以及根據另一個 SqlDataSource 篩選資料？

適用範圍：V ASP.NET 2.0　V ASP.NET 3.5

問題

1. 我用 SqlDataSource 連結資料庫, 並且寫入表單的資料, 在我的資料表欄位中的編號是自動產生, 那我要如何在資料寫入後, 取得新的編號？
2. 我用網頁寫了一個資料瀏覽的程式, 可以順利的列出資料, 但我想要先取得資料的筆數, 請問我要如何做到？
3. 我在網頁中放置了兩個 SqlDataSource, 一個用在 GridView, 另一個是用在 GridView 中的 ListBox, 但是每次取出來的 ListBox 資料都是全部的, 我只想顯示出使用者所選擇的, 我要如何做？

問題說明

SqlDataSource 是 ASP.NET 的新元件, 它主要是負責與資料庫的連接功能, 和.NET 1.x 時代的 SqlDataAdapter 有異曲同工之妙, 只不過 SqlDataSource 免除掉了開發人員還要設定資料來源, 以及下 DataBind()這道指令的麻煩, 例如以往用 SqlDataAdapter 時, 要繫結 DropDownList 控制項的資料時, 都要這樣下：

```
SqlDataAdapter adapter = new SqlDataAdapter(cmd);
DataTable table = new DataTable();
adapter.Fill(table);

this.DropDownList1.DataSource = table;
this.DropDownList1.DataBind();
```

但到了 SqlDataSource, 則只要這樣的一段即可：

```
<asp:DropDownList DataSourceID="myDS" runat="server"
  DataTextField="Caption" DataValueField="ID" />
```

可以說是省下不少的程式設計工作。

SqlDataSource 除了可以直接在網頁上設定以外, 也支援由程式來產生, 不過設定的工作就要由程式碼來做, 就算利用程式碼實作, 感覺上也不會太過複雜。也就是因爲簡單, 所以很多人反而忽略了細部的功能, 而不知道要怎麼活用它。

其實, SqlDataSource 只是骨子裡包裝了 ADO.NET 連線與指令功能的元件而已, 只要對 ADO.NET 夠熟, 就能輕易的駕馭它, 並且會爲它提供的一些額外功能而驚豔, 例如它可以支援在搜尋時做好排序, 以及支援 DataSet 與 DataReader 兩種方法, 可以讓開發人員自由選擇要如何應用, 它也適用於一般型式的 SQL 指令, 以及預存程序等等, 它也能由開發人員設定參數類型及值等, 亦提供事件給開發人員做處理。

但是天下可沒有白吃的午餐, 若對 ADO.NET 或 SQL 不熟, SqlDataSource 沒有設定好或不懂如何設定的話, 再簡單的工具也無法發揮作用。另外, SqlDataSource 也有不適用的地方, 例如想對 ADO.NET 物件有更多控制以及客制化時, 用 SqlDataSource 反而會造成不便。

解決方案

1. 若想要在資料庫插入資料後, 取得自動編號的值, 則需要在插入資料之後, 使用 SELECT @@IDENTITY, 或是 SELECT SCOPE_IDENTITY FROM sourcetable 的 SQL 指令, 取得由資料庫產生的自動編號值。

   設定 SqlDataSource 時, 要在 InsertParameters 這一區, 多設定一個回傳編號的參數值, 然後在 InsertCommand 原有的插入資料的指令後面, 多加一個類似 SELECT @id=@@IDENTITY 的指令。

然後處理 SqlDataSource.Inserted 方法, 就能夠由事件參數 e 中取得回傳編號的參數值了。

程式碼 1：利用 SqlDataSource 取得最後插入資料的識別碼值

```
// ASP.NET 頁面中的 SqlDataSource 設定
<asp:SqlDataSource ID="myDS" runat="server"
  ConnectionString="initial catalog=TestDB; integrated security=SSPI"
  SelectCommand="SELECT * FROM Employees"
  InsertCommand="INSERT INTO Employees (Name) VALUES (@name); SELECT
@id = @@IDENTITY"
  OnInserted="myDS_Inserted">
  <InsertParameters>
    <asp:ControlParameter ControlID="T_EmployeeName" Name="name"
      Type="string" Direction="Input" Size="50" />
    <asp:Parameter Name="id" Type="Int32" Direction="Output"
      Size="4" />
  </InsertParameters>
</asp:SqlDataSource>

// 程式碼中, 處理 SqlDataSource.Inserted 事件
protected void myDS_Inserted(object sender,
  SqlDataSourceStatusEventArgs e)
{
  this.labelNewEmpID.Text =
    e.Command.Parameters["@id"].Value.ToString();
}
```

2. 若想要經由 SqlDataSource 取得資料筆數時, 可利用 SqlDataSource.Select() 先取回查詢結果, 然後計算結果的資料列數, 在大多數的情況下, 可以先設定 SqlDataSource 的 DataSourceMode 爲 DataSet, 然後就可直接呼叫 SqlDataSource.Select(new DataSourceSelectArguments())取得IEnumerable 物件, 在取用之前要記得轉型, 而不同的 DataSourceMode 傳回的物件是不同的：

DataSourceMode 參數	傳回型別
SqlDataSourceMode.DataSet	DataView
SqlDataSourceMode.DataReader	IDataReader

程式碼 2：使用 SqlDataSource 取得結果集的筆數

```
// ASP.NET 頁面中的 SqlDataSource 宣告。
<asp:SqlDataSource ID="myDS" runat="server"
  ConnectionString="initial catalog=TestDB; integrated security=SSPI"
  SelectCommand="SELECT * FROM Employees" DataSourceMode="DataSet" >
</asp:SqlDataSource>
// 在程式碼呼叫 SqlDataSource.Select()
protected void Page_Load(object sender, EventArgs e)
{
  DataView view = this.myDS.Select(new DataSourceSelectArguments())
    as DataView;
  this.labelRowCount.Text = view.Count.ToString();
}
```

3. SqlDataSource 有一個特性, 就是一次只能做一種資料查詢, 就算有多個控制項, 也只能給一種資料, 因此若想要給不同的控制項不同的資料, 則必須要建立相同個數的 SqlDataSource 物件, 或者利用動態產生的方式來查詢 (如下列程式)。不過這是否會引發過多的查詢 (例如GridView 中若有 50 個 ListBox, 那豈不是要做 50 個查詢？) , 是需要商榷及評估的。

若是需要做到這個功能, 筆者提供另一個思考方向：將 ListBox 的資料放到 DataTable 中, 然後在 GridView 的 RowDataBound 事件常式中將資料加入 ListBox 即可, 這樣就可以省下多餘的查詢。

程式 3-1：ASP.NET 網頁的 GridView 與 SqlDataSource 設定

```
<asp:GridView ID="OrderGrid" runat="server" CellPadding="4"
  AutoGenerateColumns="false" DataSourceID="myDS"
  ForeColor="#333333" GridLines="None"
  OnRowDataBound="OrderGrid_RowDataBound">
  <FooterStyle BackColor="#1C5E55" Font-Bold="True"
    ForeColor="White" />
  <RowStyle BackColor="#E3EAEB" />
  <PagerStyle BackColor="#666666" ForeColor="White"
    HorizontalAlign="Center" />
  <SelectedRowStyle BackColor="#C5BBAF" Font-Bold="True"
    ForeColor="#333333" />
  <HeaderStyle BackColor="#1C5E55" Font-Bold="True"
    ForeColor="White" />
  <EditRowStyle BackColor="#7C6F57" />
  <AlternatingRowStyle BackColor="White" />
  <Columns>
    <asp:BoundField DataField="CustomerID" />
    <asp:BoundField DataField="ContactName" />
    <asp:TemplateField>
      <ItemTemplate>
        <asp:ListBox ID="listBox" runat="server" />
      </ItemTemplate>
    </asp:TemplateField>
  </Columns>
```

```
</asp:GridView>
<asp:SqlDataSource ID="myDS" runat="server" ConnectionString=
"Data Source=localhost;Initial Catalog=Northwind;Integrated Security=True"
  ProviderName="System.Data.SqlClient"
  SelectCommand="SELECT * FROM [Customers]" />
```

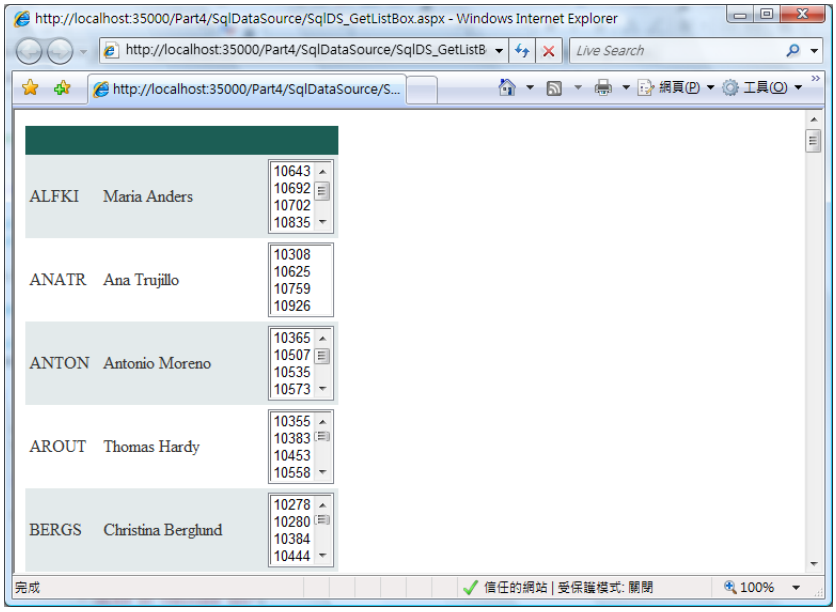
程式碼 3-2：GridView 的 RowDataBound 事件常式

```
protected void OrderGrid_RowDataBound(object sender,
  GridViewRowEventArgs e)
{
  if (e.Row.RowType == DataControlRowType.DataRow)
  {
    string customerID = e.Row.Cells[0].Text;
    ListBox listBox = e.Row.FindControl("listBox") as ListBox;

    SqlDataSource ds = new SqlDataSource();
    ds.ConnectionString = this.myDS.ConnectionString;
    ds.SelectCommand =
      "SELECT OrderID FROM Orders WHERE CustomerID = @customerID";
    ds.SelectParameters.Add(
      new Parameter("customerID", TypeCode.String, customerID));
    ds.DataSourceMode = SqlDataSourceMode.DataSet;
    DataView view = ds.Select(new DataSourceSelectArguments())
      as DataView;

    listBox.DataSource = ds;
    listBox.DataTextField = "OrderID";
    listBox.DataValueField = "OrderID";
    listBox.DataBind();
  }
}
```

這個程式的執行結果如下圖：



小常識　與 SqlDataSource 地位相當的 ObjectDataSource 控制項

SqlDataSource 雖然給了開發人員簡化資料存取的功能，但它只能存取資料庫，若資料來源不是資料庫 (例如 Web Service) 時，就要改用 ObjectDataSource 控制項。ObjectDataSource 讓開發人員得以透過中間物件 (例如商業邏輯層) 所開放出來的方法，來執行資料的處理工作，操作的方法和 SqlDataSource 類似，但有幾點不同：

- ObjectDataSource 的資料來源是物件，所以必須要給定 Select/Insert/Update/Delete 所對應的方法，分別由 SelectMethod/InsertMethod/UpdateMethod 與 DeleteMethod 所管理。
- 作爲 ObjectDataSource 的來源物件，必須要是無狀態 (State-less) 的物件，若資料處理方法不是以 static 開放的話，就必須要有一個不帶參數的預設建構式，讓 ObjectDataSource 得以自動建立物件的實體。所有的物件產生與釋放都由 ObjectDataSource 來處理。
- 透過實作 SelectCountMethod 的方法，才能讓 ObjectDataSource 支援分頁能力。

因此若想要以 ObjectDataSource 連接元件的話，必須要先打量看看元件有沒有足夠的支援。

相關問題

- Q46：我使用 DataAdapter 做資料修改，但下了 Update()指令後，爲何沒有更新？

Q39 對 Access 資料庫新增／修改或刪除資料時發生「運作必須使用更新查詢」的錯誤？

適用範圍：V ASP.NET 1.0 V ASP.NET 1.1 V ASP.NET 2.0 V ASP.NET 3.5

問題

我用 Access 資料庫設計一個留言板, 程式和所使用的 SQL 指令都沒有問題, 但是在執行新增時都會出現「運作必須使用更新查詢」的錯誤訊息, 請問要如何解決這個錯誤呢？

問題說明

Microsoft Access 資料庫是程式設計初學者經常會使用到的資料庫類型, 其使用者介面以及簡單的操作環境深受初學者的愛用, 然而就是因爲初學者, 對於一些系統的設定或運作不夠了解, 使得在不知道資料庫本身限制的情況下, 在設計時到處碰壁, 或是遇到不知所從的現象。

Microsoft Access 資料庫採用 Jet Database Engine 做爲資料庫引擎 (Database Engine) , 這是個適用於檔案型資料庫 (File-Based Database) 的小型引擎, 舉凡 Excel, Access, Dbase, FoxPro 等等資料庫, 都能夠運用 Jet Database Engine 存取, 在小型資料庫上的應用相當廣泛, 不過因爲檔案型資料庫的體質限制, 所以只能用在應用的範圍不大 (少量使用者、少量資料或是不過於講求效能等) 的資料管理上。

「運作必須使用更新查詢」就是最典型的初學者易犯錯誤, 這個錯誤會發生在資料庫無法寫入時, 資料庫引擎在接收到寫入之 SQL 指令時, 會解譯並執行資料庫動作, 但若資料庫引擎發現了資料來源無法寫入時, 便會丟出一個 0x8004005 訊息, 其英文訊息是「Operation must use an updatable query」, 意爲「動作必須要使用一個『可更新』的查詢」。而不是官方翻譯的「運作必須使用更新查詢」(這可要好好訓一下負責翻譯的人了 ☺) 。

Access 資料庫本身是一種檔案型資料庫, 其基本體質仍然會受到檔案本身限制的影響, 例如唯讀 (Read Only) 屬性；以及資料庫引擎本身的限制, 例如 Access 資料庫可以允許 250 個使用者同時 SELECT, 但卻只能允許一個使用者執行寫入工作, 這是因爲 Jet 資料庫引擎會在使用者發出寫入指令時, 將資料表鎖定起來 (以資料庫並行行爲來說, 稱爲悲觀鎖定策略) , 此時 Jet 資料庫引擎會將其他寫入呼叫都封鎖起來, 被封鎖的使用者就會看到這個錯誤訊息。

其實這個錯誤訊息還不夠明白, 因爲不是只有唯讀的狀況會造成無法寫入, 任何會造成更新失敗的指令都會出現這種訊息, 例如想要寫入資料到計算欄位 (Calculated field) 或是想寫入目前被鎖定的資料時, 都會產生這個錯誤。

有些初學者爲了想要馬上看到資料輸入的結果, 都會不自覺的把 Access 開著, 當 Access 打開資料庫時, 會將資料庫鎖定住, 造成指令執行時出現這個錯誤。

還有一些狀況就是使用權限 (Permission) 問題, 例如 ASP.NET 的執行帳戶 (ASPNET 或 Network Service) 不具有存取 Access 資料庫的權限時, 資料庫引擎會收到 Access Denied (拒絕存取) 錯誤, 然後產生這個錯誤訊息給使用者。

解決方案

如果要排除這類的問題, 就需要將無法寫入查詢的因素排除掉, 通常最可能造成的因素, 大多數是權限問題, 例如檔案被設定爲唯讀、或是可存取的權限不夠、檔案被鎖定或是目的資料表被鎖定等等, 只要能夠排除無法寫入的問題, 這個訊息就不會出現了。

小常識　小型應用程式的資料庫選擇

Access 資料庫是一個很好用的小型資料庫, 對於小型應用是非常適合, 不過它其實並不適合使用於網站型資料庫, 或是有多人存取的解決方案, 原因是 Access 本身的限制。對於網站資料庫, 可以利用像 SQL Server Express, MySQL 或是其他以主從式設計的資料庫管理系統, 這樣在可擴充性或是延展性都會有比較大的彈性。

不過如果像是只有單一用戶端的話, 使用 Access 資料庫對於部署與管理上會比較方便 (例如 Microsoft DHCP Server 用的資料庫就是 Access 資料庫 ☺) 。

Q40 我的網站需要登入才能使用，但使用者卻說帳號遭到冒用？

適用範圍：VASP.NET 1.0 VASP.NET 1.1 VASP.NET 2.0 VASP.NET 3.5

問題

我用 ASP.NET 開發了一套公司的內部管理系統, 目前使用狀況都正常, 只是有時候會有使用者反應說他的帳戶被冒用了, 有些登入記錄不是由他所操作的, 請問我要如何避免這種冒用的情況發生？

問題說明

登入帳戶是系統安全的關卡之一, 雖然使用者帳戶不是最後一道防線, 但是帳戶被冒用或竊取而產生的資訊安全風險與破壞, 並不好調查, 就算做好了稽核, 也只能看出帳戶的登入與登出時間及 IP 位置, 若要進一步的調查出是誰執行的, 可能也要調閱其他的資料, 才可進一步的查出來。

要保護帳戶, 除了平常將帳戶的密碼原則設定較高安全度的限制外, 就是要避免因社交工程而造成的帳戶外洩風險。在程式中可以實作的, 就只有消極的檢查, 稽核以及保存記錄了。

不過還是可以多做一些防護措施, 例如重覆的使用者登入, 除非是由使用者自己下的, 否則都應被視爲有帳戶外洩而造成的。此時, 系統就應該要能夠拒絕這種重覆登入的要求, 並且將當時的系統資訊 (IP, 登入時間與次數等) 記錄下來, 日後可供追查原因時參考之用。

要實作出這樣能力的系統, 其實並不難, 只要在使用者帳戶資料表中加入幾個欄位：

欄位	說明
LastLoginDateTime	最後一次的登入日期。
SessionToken	登入時產生的 SessionToken。
LoginFromIP	登入時電腦的 IP 位址。
LastLogoutDateTime	最後一次登出日期。

然後運用下列的規則去比對：

- 當 LastLogoutDateTime 大於 LastLoginDateTime 時, 代表使用者當未登入, 應允許其登入, 並記錄登入日期、SessionToken 及 IP 位址。
- 當 LastLogoutDateTime 小於 LastLoginDateTime 時, 再比對 SessionToken, 若 SessionToken 不相同, 則應視爲使用者已經登入, 此時就要拒絕同一個使用者帳戶的登入。
- 當使用者登出時, 應記錄 LastLogoutDateTime, 若是 Session Timeout 時, 也要更新這個值, 以確保下次使用者仍可登入。
- 系統管理員可以手動更新 LastLogoutDateTime 的值, 以確保因爲操作不當而造成無法 Login 的問題可以解決。

如果需要記錄登入活動的話, 還可以加上登入稽核的資料表, 並且設定下列欄位：

欄位	說明
Serial	流水號（使用識別欄位即可）。
LoginName	登入名稱。
LoginDateTime	登入日期（使用預設值 GETDATE()即可）。
Result	登入成功與否（1 = 成功, 0 = 失敗）。

當然, 筆者的作法只是參考用的作法, 如果讀者有更好的方式, 也可以依照讀者自己的方式來實作, 畢竟各公司的作法不同, 也最好是不要太類似, 以免安全機制被有心人士識破, 那就不好了 ☺。

解決方案

在資料庫中加入適當的登入記錄與登入稽核記錄的表格, 並且在程式中實作檢查登入記錄的程式碼, 若登入失敗時應記入稽核記錄中, 程式 1 是依照上述規則所簡單實作的範例程式碼。

程式 1. 預防重覆登入行爲的程式碼 (僅表示筆者的意思, 請在實作時詳加測試)

```
public bool Login(string SessionToken, string IP, string UserName,
  string Password)
{
  // 驗證使用者。
  string userID = AuthenticationService.Login(UserName, Password);

  if (userID == null)
    return false;

  // 檢查是否已登入過。
  if (!AuthenticationService.CheckDuplicateLogin(SessionToken))
  {
    AuthenticationService.LogInfo(SessionToken, IP); // 記錄登入資訊
    ... // 其他登入程序。
    return true;
  }
  else
    return false; // 發現重覆登入, 拒絕登入。
}

public void Logout(string SessionToken)
{
```

```
  AuthenticationService.Logout(SessionToken); // 記錄登出資訊
  ... // 其他必要的登出程序。
}

// AuthenticationService 中的 CheckDuplicateLogin 範例。
public bool CheckDuplicateLogin (string SessionToken)
{
  DateTime lastLoginDateTime, lastLogoutDateTime;
  string sessionID = null;
  ... // 填入時間資訊及 SessionID 資訊

  // 檢查目前是否在登入時間內
  if (lastLoginDateTime >= lastLogoutDateTime)
  {
    if (sessionID != SessionToken) // 檢查 SessionID 是否相同
      return true; // 不同，表示有別的使用者試圖用同一帳戶登入
    else
      return false;
  }
  else
    return false;
}
```

Q41 如何獲取資料庫資訊與資料表結構？

適用範圍：VASP.NET 1.0 VASP.NET 1.1 VASP.NET 2.0 VASP.NET 3.5 VSQL Server 2005

問題

我想要製作一個在 Web 上監控資料庫狀態的工具程式, 使用的資料庫是 SQL Server 2005, 請問我有哪些方法, 可以獲取資料庫的相關資訊, 包括資料表、檢視表、預存程序、使用者函數等等？

問題說明

大多數的資料庫管理系統 (DBMS) 都會以資料庫來儲存系統的一些資訊, 然後利用一些方式開放出來, 例如檢視表或是系統預存程序等等, 這些方法稱爲系統目錄 (System Catalog) , 專門開放給開發人員、系統管理人員或是第三方的軟體廠商來設計一些工具程式或軟體, 在大部份的情況下, 是不需要存取系統目錄的。

不過有些特殊需求例外：

- 透過 Web 管理資料庫。
- 透過 Web 列舉, 修改與更新資料庫物件。
- 透過 Web 收集效能資料。
- 監控資料庫活動。
- 列舉與處理多餘的資料表 (這在暫存資料表過多時會很好用) 。

SQL Server 2005 的系統資料表以及工具程序相當完整, 除了基本的系統資訊外, 像是管理資料庫活動所需要用到的動態管理檢視表 (Dynamic Management View; DMV) 以及輔助功能的查詢介面, 都可以讓開發人員可以很輕易的獲取資料庫的資訊。

例如, 若想要獲得目前在伺服器中的資料庫, 可以下這個指令 (這個指令在 SQL Server 2000 亦可適用) ：

```
EXEC sp_databases
```

這個指令可以查詢到所有在資料庫伺服器中的資料庫列表, 包含資料庫名稱以及資料庫目前的大小, 例如：

	DATABASE_NAME	DATABASE_SIZE	REMARKS
1	C1	9195392	NULL
2	L1	3236096	NULL
3	L2	4724288	NULL
4	master	5376	NULL
5	model	3264	NULL
6	msdb	8192	NULL
7	Northwind	4288	NULL
8	P1	653376	NULL
9	tempdb	8704	NULL
10	TestDB	4096	NULL
11	W1	183296	NULL

若想要知道更多的資料庫選項設定, 則可以透過 sys.databases 檢視表來做：

```
SELECT * FROM sys.databases
```

在得到資料庫的資訊後, 想當然也會想知道資料庫的一些狀態 (例如檔案群組的大小, 資料庫的物件等) , 像是若想要得到資料庫的大小, 可以利用這樣的查詢：

```
-- size 以 8KB 爲單位, 故乘以 8 才會是正確的大小。
SELECT SUM(Size) * 8 FROM sys.database_files
```

資料庫的物件可以分爲很多種, 資料表、檢視表、預存程序、預設值、觸發程序與條件限制等都是, 在資料庫的系統目錄中, 都是以個別的識別方法來儲存, 在每個資料庫中, 都會有一個 sysobjects 系統資料表, 用來儲存物件。各種物件的識別代字如表所示。

在 SQL Server 2005 中, 由 sys.objects 檢視表取代 sysobjects 資料表, 但 sysobjects 資料表仍然可以查詢, 但爲了未來版本可能會移除 sysobjects 不提供查詢的可能性, 最好是改由 sys.objects 來查詢。

資料庫物件的識別代字

代碼	名稱	代碼	名稱	代碼	名稱
AF	彙總函數 (CLR)	R	規則 (舊式、獨立式)	FS	組件 (CLR) 純量函數
C	CHECK 條件約束	RF	複寫篩選程序	FT	組件 (CLR) 資料表值函數
D	DEFAULT (條件約束或獨立式)	S	系統基底資料表	TF	SQL 資料表值函數
F	FOREIGN KEY 條件約束	SN	同義字	U	資料表 (使用者自訂)
PK	PRIMARY KEY 條件約束	SQ	服務佇列	UQ	UNIQUE 條件約束
P	SQL 預存程序	TA	組件 (CLR) DML 觸發程序	V	檢視表
PC	組件 (CLR) 預存程序	TR	SQL DML 觸發程序	X	擴充預存程序
FN	SQL 純量函數	IF	SQL 嵌入資料表值函數	IT	內部資料表

若想要查詢資料庫中有哪些物件, 可以利用這些指令：

```
-- 查詢由使用者建立的資料表
SELECT * FROM sys.objects WHERE type = 'U'
-- 查詢有哪些預存程序
SELECT * FROM sys.objects WHERE type = 'P'
-- 查詢有哪些 DML 觸發程序
SELECT * FROM sys.objects WHERE type = 'TR'
-- 查詢有哪些鍵值, 包含 PK 和 FK
SELECT * FROM sys.objects WHERE type IN ('F', 'PK')
```

當列出了資料表或檢視表後, 開發人員一定也會希望取得資料表或檢視表的欄位列表, 在 SQL Server 2005 中有提供 syscolumns 與 sys.columns 檢視表來給開發人員查詢使用, 並且可查詢到欄位所用的型別以及大小, 可說是相當豐富的資訊。

例如, SQL Server 將資料庫物件 (如資料表) 的資訊儲存在 sysobjects 資料表, 並且用 sys.objects 檢視表顯露出來；將欄位的資訊儲存在 syscolumns 資料表, 並且 sys.columns 檢視表顯露出來；而系統內建的資料型態儲存在 systypes 資料表, 並且用 sys.types 檢視表顯露出來。

在 sys.objects 檢視表中, 每一個物件都有一個 object_id, 而 sys.columns 檢視表中, 每一個欄位都可以對應到一個 object_id, 所以如果要查詢一個資料表中有哪些欄位, 可以利用下列指令：

```
SELECT
  o.name AS TableName, c.name AS ColumnName, c.max_length AS Size
FROM sys.columns c
  INNER JOIN sys.objects o ON c.object_id = o.object_id
WHERE o.type = 'U'
```

若想要進一步查出欄位使用的資料型別, 則可透過 sys.columns 中的 user_type_id, 它可和 sys.types 中的 user_type_id 對應, 所以若想查詢資料庫中使用者資料表的名稱, 以及其所屬欄位名稱和資料型別與大小, 就可以用這個指令：

```
SELECT
  o.name AS TableName, c.name AS ColumnName,
  t.name AS DataType, c.max_length AS Size
FROM sys.columns c
  INNER JOIN sys.objects o ON c.object_id = o.object_id
  INNER JOIN sys.types t ON c.user_type_id = t.user_type_id
WHERE o.type = 'U'
```

另外若想要取得檢視表、預存程序、觸發程序、使用者函數等的定義, 可以透過 sys.sql_modules 檢視表來取得。

```
SELECT * FROM sys.sql_modules
```

解決方案

利用 SQL Server 2005 內建的許多系統目錄的檢視表, 來查詢想要查詢的內容, 可多參考 SQL Server 2005 線上書籍來取得這些資料。

Q42 我要如何讀取 XML 資料並存入資料庫？

適用範圍：V ASP.NET 1.0 V ASP.NET 1.1 V ASP.NET 2.0 V ASP.NET 3.5

問題

我的應用程式會在每天半夜時由合作廠商那裡接收銷售的資料, 格式是 XML 檔, 我想要把這個 XML 檔和我的資料庫做整合, 請問我要如何做？

問題說明

Microsoft .NET Framework 對 XML 資料的支援相當的充份, 除了提供了 System.Xml 命名空間的類別使用外, 也在 DataSet 中加入了和 XML 資料進行交換的功能, 透過 DataSet.ReadXml()及 DataSet.WriteXml()來輸入與輸出 XML 資料流, 可簡化部份轉換工作。

或者, 自己操作 XML 的 DOM (Document Object Model) , 這也是筆者自己的習慣作法, 因爲自己操作的話, 就可以避免因爲黑箱作業而造成的問題, 並且也具有最大的自訂化能力, 但缺點就是要寫較多一點的程式碼。

DOM 是一種將 XML 資料視爲物件的一種存取方法, 就像一般 HTML 控制一樣, HTML 也是一種 DOM 介面, 開發人員需要在 DOM 中巡覽以及存取各層次的資料, 並且處理來自 DOM 的事件, 而 DOM 是由 XML Parser (剖析器) 所建立, 實際的資料讀寫由 XML Parser 來負責, 開發人員只需要遵循 DOM 的方法來存取各個部份的資料即可。

例如下列的程式碼就是使用 XML DOM 方式存取 XML 資料的簡單範例：

```
using System.Xml;
...

public void ConvertToDB()
{
```

```
  XmlDocument doc = new XmlDocument();
  doc.load("http://acme.com/xmlbase/supplier.xml");

  foreach (XmlNode node in doc.DocumentElement.ChildNodes)
  {
    db.WriteData(node.InnerText, data);
  }

  doc = null;
}
```

如果 XML 的資料愈複雜, 可能就會需要做一些搜尋或列舉的功能, 或者是要依需要新增或刪除節點等。

例如現有個 Stock 資料表, 其結構爲：

欄位名稱	大小	資料型別
ProductID	16	uniqueidentifier
Barcode	13	varchar(13)
Qty	4	int

這個資料表是儲存產品的庫存資訊, 而每日都會由合作廠商傳回目前的庫存, 其格式是：

```
<StockList>
<Stock ProductID="66B0A09B-9C11-42e1-BBD0-655A6B7AA182"
  Barcode="558666981258" Qty="13" />
<Stock ProductID="66B0A09B-9C11-42e1-BBD0-655A6B7AA182"
  Barcode="5583264158974" Qty="9" />
```

```
<Stock ProductID="66B0A09B-9C11-42e1-BBD0-655A6B7AA182"
  Barcode="5583369458742" Qty="1" />
<Stock ProductID="66B0A09B-9C11-42e1-BBD0-655A6B7AA182"
  Barcode="5584698451147" Qty="25" />
<Stock ProductID="66B0A09B-9C11-42e1-BBD0-655A6B7AA182"
  Barcode="5588632184169" Qty="4" />
...
</StockList>
```

在處理完庫存資料後, 還要再產生一份相同格式的 XML 資料檔, 以回報目前的庫存資料, 這二個工作都可以利用 XmlDocument 搭配資料存取功能來做到, 下列程式即爲將 Stock 資料表中的資料輸出爲 XML 的程式碼：

```
using System.Xml;
using System.Data;
...

public string ConvertStockListToXML(string ShopID)
{
  XmlDocument doc = new XmlDocument();
  doc.loadXML("<StockList></StockList>");

    DataTable table = db.GetStockList(ShopID);

    foreach (DataRow row in table.Rows)
    {
      // 建立節點, 以及屬性
      XmlNode node = doc.CreateNode(XmlNodeType.Element, "Stock",
        null);
      XmlNode attrProdID = doc.CreateNode(XmlNodeType. Attribute,
        "ProductID", null);
```

```
        XmlNode attrBarcode = doc.CreateNode(XmlNodeType.Attribute,
          "Barcode", null);
        XmlNode attrQty = doc.CreateNode(XmlNodeType. Attribute,
          "Qty", null);
        // 設定屬性值
        attrProdID.Value = row["ProductID"].ToString();
        attrBarcode.Value = row["Barcode"].ToString();
        attrQty.Value = row["Qty"].ToString();
        // 將屬性加入節點
        node.Attributes.SetNamedItem(attrProdID);
        node.Attributes.SetNamedItem(attrBarcode);
        node.Attributes.SetNamedItem(attrBarcode);
        // 將節點加入 XML 資料結構中
        doc.DocumentElement.AppendChild(node);
      }

      string xml = doc.InnerXml;
  doc = null;
  return xml;
}
```

下列的程式碼爲讀取 XML 資料到資料庫中的程式碼：

```
using System.Xml;
using System.Data;
...

public void ConvertXMLToStock(string ShopID, string DataXML)
{
```

```
  XmlDocument doc = new XmlDocument();
  doc.loadXML(DataXML);
  // 搜尋 XML 資料, 列舉搜尋的結果。
  XmlNodeList nodes =
    doc.DocumentElement.SelectNodes("//StockList/Stock");

  foreach (XmlNode node in nodes)
  { // 更新資料庫
    db.UpdateStock(
      node.Attributes.GetNamedItem("ProductID").Value,
      node.Attributes.GetNamedItem("Barcode").Value,
      Convert.ToInt32(node.Attributes.GetNamedItem("Qty").Value));
  }

  doc = null;
}
```

因此, 若想要靈活的操作 XML 資料的話, 就應該要對 XML DOM 熟悉, 並且要試著利用它來建構與讀寫 XML 的資料, 如此才會增加對 XML DOM 的應用經驗, 以備未來在複雜或需要的環境中活用 XML 資料, 解決企業的問題。

解決方案

在每日定時接取來自合作廠商的程式中, 利用 XML DOM 來讀取 XML 的資料, 並且配合資料庫程式來寫入資料庫即可。

相關問題

- Q32：要用 DataSet/DataTable 好, 還是用 DataReader 好？
- Q33：執行 ExecuteNonQuery()時都會顯示連線被佔用的訊息？
- Q38：如何活用 SqlDataSource 來提供網頁所需的資料？

Q43 如何動態產生 XML 資料以提供像是 RSS 訂閱的服務？

適用範圍： ☑ASP.NET 1.0 ☑ASP.NET 1.1 ☑ASP.NET 2.0 ☑ASP.NET 3.5

問題

我設計了一個公布欄系統, 我想要提供 RSS (Really Simple Syndication) 的訂閱功能, 不過我應該要如何產生 XML 資料？

問題說明

在「我要如何讀取 XML 資料並存入資料庫」問答中, 筆者使用了 XML DOM 模式來讀取與寫入 XML 格式的資料, 以這個問題來說, 使用 XML DOM 的方法, 確實是可以很簡單的解決這個問題, 不過筆者要介紹另一種方法, 在 System.Xml 命名空間中, 除了 XML DOM 外, 還有一個讀寫的管道, 由 XmlReader 與 XmlWriter 組成, 它可以允許開發人員配合 XML Schema 做資料格式的驗證, 並且用很直覺的方式來產生 XML 的資料。

要使用 XmlReader 還是 XML DOM 來讀寫資料, 由開發人員個人習慣來決定, 因爲筆者使用的是 XML DOM, 但讀者可能已經習慣用 XmlReader 來做, 因此只要能夠順利的讀寫 XML, 這兩個方法都可以做到, 讀者只要選用自己喜歡的方法即可。

RSS 是 XML 上的資料應用服務之一, 和 Atom 以及其他的相關服務, 都是利用 syndication feed (所以會有 RSS feed 這個名詞, 筆者稱它叫 RSS 種子 ☺) 的方式, 將資料以 Push 的方法散布到各個有訂閱 syndication feed 的用戶端, 這類型的技術被稱爲 Podcasting, 也就是不論用戶端是用何種設備 (可能是手機, i-Pod 或 MP3 Player 等) , 都可以經由 syndication feed 來取得最新的資訊與情報。Podcasting 技術經常應用在行銷與廣告上, 例如定時更新的新品情報, 或者即時的促銷活動。在某些方面, 它甚至可以取代電子報, 做爲主要的廣告工具之一。

因此, 若想要在自己的網站上實作 Podcasting 能力, 以網路瀏覽器來說, RSS 已經具備了充份的功能, 只要依照 RSS 的結構來實作 XML 訊息, 並且開放出來, 就代表網站已經具備了 Podcasting 的能力了。

RSS 的規格可以在 RSS Advisory Board 的網站上找到：

```
http://www.rssboard.org/rss-specification
```

在開始動手之前, 先來了解一下 XmlWriter, 它是一個基於資料流之上所建立的寫入程式, 會將指令所產生的資料寫入資料流中, 資料流可以是檔案、網路或是記憶體的資料流, 只要是可寫入的資料流都可以。在 XmlWriter 中, 有部份的指令是成對的, 例如寫入文件資料的WriteStartDocument()和WriteEndDocument() 方法是成對的, 或是寫入元素的WriteStartElement()和WriteEndElement()方法等, 這些方法的使用方式都很直覺, 有開就有關, 下了一個 Start 指令, 就一定要下一個 End 指令將它結束, 而且 Start 和 End 指令必須在同一個階層, 不可交錯使用, 否則 XmlWriter 會產生錯誤。

例如, 如果要產生下列結構的 XML 資料：

```
<rss version="2.0">
  <channel>
    <title>Northwind</title>
    <url></url>
    <description>Northwind</description>
  </channel>
</rss>
```

可以用下列的程式碼來達成：

```
MemoryStream ms = new MemoryStream();
XmlTextWriter xtw = new XmlTextWriter();
// 起始 XML 文件
xtw.WriteStartDocument();
// <rss>
xtw.WriteStateElement("rss");
// <rss version="2.0">
xtw.WriteAttributeString("version", "2.0");
// <channel>
xtw.WriteStateElement("channel");
// <title>Northwind</title>
xtw.WriteElementString("title", "Northwind");
// <url></url>
xtw.WriteElementString("url", string.Empty);
// <description>Northwind</description>
xtw.WriteElementString("description", "Northwind");
// </channel>
xtw.WriteEndElement();
// </rss>
xtw.WriteEndElement();
// 結束文件
xtw.WriteEndDocument();

xtw.Flush();
```

對 XmlWriter 比較了解之後, 就可以來實作 RSS 的 XML 產生程式了。

解決方案

筆者以北風範例資料庫爲資料來源, 輸出產品的資料, 包含品名, 單位數量以及單價資訊到 RSS 結構中, 這可以應用在新產品的通知上, 例如透過 RSS 來發布新產品的資訊, 並且提供特惠方案的通知, 以吸引消費者購買。

首先, 先要建構出 RSS 的基本元素, 即：

```
<rss version="2.0">
  <channel>
    <title>Northwind</title>
    <url></url>
    <description>Northwind</description>
  </channel>
</rss>
```

然後再將各個項目的元素填入 RSS 結構中, 如：

```
<rss version="2.0">
  <channel>
    <title>Northwind</title>
    <url></url>
    <description>Northwind</description>
    <item>
      <title></title>
      <link></link>
      <description></description>
    </item>
    ...
  </channel>
</rss>
```

所以, XmlWriter 的階層寫法就需要注意, 在填入子項資料時, 要用資料庫的資料來填, 完整的程式碼如下列程式所示。

程式 1. RSS 的產生程式

```
MemoryStream ms = new MemoryStream();
XmlTextWriter xtw = new XmlTextWriter(ms,
  System.Text.Encoding.Default);
SqlConnection conn = new SqlConnection(
  "initial catalog=Northwind; user id=xx; password=xx");
SqlDataReader reader = null;
SqlCommand cmd = new SqlCommand("SELECT * FROM Products", conn);

xtw.WriteStartDocument();

conn.Open();

reader = cmd.ExecuteReader(CommandBehavior.CloseConnection);
// 產生 RSS 宣告
xtw.WriteStartElement("rss");
xtw.WriteAttributeString("version", "2.0");
xtw.WriteStartElement("channel");

// 產生 RSS 檔頭
xtw.WriteElementString("title",
  "Northwind Product Notification Feed");
xtw.WriteElementString("url",
  "http://www.northwind.com/notification/productinfo.xml");
xtw.WriteElementString("description",
  "Northwind Product Notification Feed");
```

```
while (reader.Read())
{
  // 產生 RSS 的項目, 填入資料庫中的資料。
  xtw.WriteStartElement("item");

  xtw.WriteElementString("title", reader.GetValue(
    reader.GetOrdinal("ProductName")).ToString());
  xtw.WriteElementString("link", string.Empty);
  xtw.WriteStartElement("description");
  xtw.WriteString("Qty Per Unit: " +
    reader.GetValue(reader.GetOrdinal("QuantityPerUnit")).ToString() +
    ", Unit Price: " +
    reader.GetValue(reader.GetOrdinal("UnitPrice")).ToString());
  xtw.WriteEndElement();

  xtw.WriteEndElement();
}

xtw.WriteEndElement();
xtw.WriteEndElement();
reader.Close();
xtw.WriteEndDocument();
// 將緩衝區的資料寫入資料流中
xtw.Flush();
ms.Flush();
ms.Position = 0;
// 將 RSS 資料讀出
StreamReader sr = new StreamReader(ms);
string xmldata = sr.ReadToEnd();
```

```
sr.Close();
xtw.Close();
ms.Dispose();
// 輸出 RSS 資料
Response.ContentType = "text/xml";
Response.Write(xmldata);
Response.End();
```

在測試的時候, 可用支援 RSS 的 Reader 或是 IE7, 如果是符合 RSS 規格的話, 應會顯示出訂閱 RSS 的畫面：

如果格式不正確, 大小寫錯誤或是資料內容有問題時, 就會顯示不支援摘要內容的訊息：

考生停看聽

本問題所討論的內容, 可用來準備 Exam 70-528: TS: Microsoft .NET Framework 2.0 Web Client Development 的下列主題。

- 使用 XmlReader 與 XmlWriter 來讀寫 XML 資料。
  - 使用 XmlWriter 來寫入 XML 資料。

Q44 如何利用 XSLT 來轉換 XML 資料格式或套用 XML 到 HTML 中？

適用範圍： V ASP.NET 1.0 V ASP.NET 1.1 V ASP.NET 2.0 V ASP.NET 3.5

問題

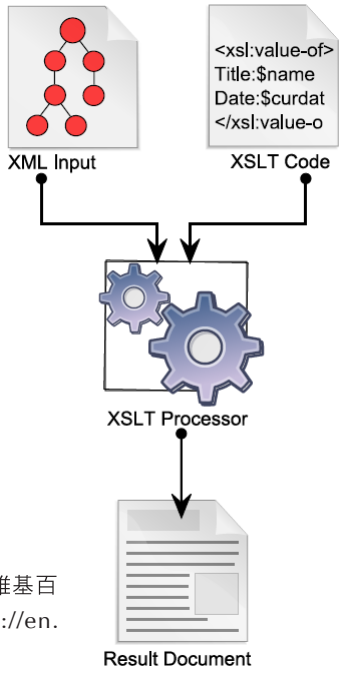
公司日前和合作廠商簽約, 議定未來兩邊的資訊系統要共享銷售資料, 但兩邊系統的資料結構並不相同, 雖然初步討論後, 決定要用 XML 來做交換的格式, 但是要如何把他們公司的 XML 格式轉換成我們的 XML 格式？

問題說明

在「Q43：如何動態產生 XML 資料」問答中, 筆者使用 XmlWriter 來產生資料, 不過一定有一些讀者認爲太麻煩, 但又對 XML DOM 不熟, 那有沒有其他的方法可以直接做 XML 和資料的轉換？特別是在兩個 XML 格式資料之間的轉換, 如果要用讀／寫方式來轉換 XML 格式的話, 可能要寫的程式會很多 (讀一支, 寫一支) , 而且可能要轉換的 XML 資料有很多種, 那豈不是每種都要寫一次, 20 種就要寫 20 次, 100 種就要寫 100 次...可能開發人員要寫到昏倒了。

所以, 筆者在這個問題中, 要介紹另一種的 XML 資料交換方法, 只要透過簡單的宣告設定, 以及少量的程式碼, 就能輕鬆的在不同的 XML 資料格式間做轉換, 而開發人員就只要針對不同的 XML 格式撰寫轉換用的樣板就可以了。

若對 XML 熟悉的讀者應該知道, 以前 XML 就只單純是資料時, 要在網頁上使用的話是很困難的, 而且還要花時間與心力去寫 XML DOM 的程式, 若 XML 資料種類多的話, 要寫的程式可能會超乎開發人員的想像。因此 W3C 特別制定了一種可以讓 XML 套用樣式能力的規格, 稱爲 XSL (XML Style-sheet) , XSL 可以藉由格式化 XML 資料的能力, 讓它能夠只利用瀏覽 XML 的方式, 呈現出較具親和力的外貌。



XSLT 處理示意圖 (取自維基百科 XSLT 說明網頁 http://en.wikipedia.org/wiki/XSLT)

但 XSL 不是只能轉換使用者介面的 HTML 而已, 它還可以和 XML 格式間做互轉, 這種能力就是 Transformations, 所以 XSL 也被稱爲 XSL/T, 表示 XSL 具有轉換格式的能力。

例如, 如果想要轉換下列的 XML 資料檔：

```
<?xml version="1.0"?>
<ProductList>
  <Product ID="1" Caption="Intel QX4850 CPU" Qty="25" />
  <Product ID="2" Caption="Intel QX4860 CPU" Qty="12" />
  <Product ID="3" Caption="Intel Pentium D" Qty="8" />
  <Product ID="4" Caption="Intel 6600 CPU " Qty="11" />
  <Product ID="5" Caption="Intel Xeon 5330" Qty="5" />
</ProductList>
```

到這樣的格式：

```
<?xml version="1.0"?>
<ProductList>
  <Product ID="1" Qty="25"> Intel QX4850 CPU</Product>
  <Product ID="2" Qty="12"> Intel QX4860 CPU</Product>
  <Product ID="3" Qty="8"> Intel Pentium D</Product>
  <Product ID="4" Qty="11"> Intel 6600 CPU</Product>
  <Product ID="5" Qty="5"> Intel Xeon 5330</Product>
</ProductList>
```

則可以利用下列的 XSLT 指令碼來實作：

```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="yes" encoding="utf-8" version="1.0"/>

<xsl:template match="/">
<ProductList>
<xsl:for-each select="//ProductList/Product">
  <Product>
    <xsl:attribute name="ID">
      <xsl:value-of select="@ID" />
    </xsl:attribute>
    <xsl:attribute name="Qty">
      <xsl:value-of select="@Qty" />
    </xsl:attribute>
```

```
    <xsl:value-of select="@Caption" />
  </Product>
</xsl:for-each>
</ProductList>
</xsl:template>

</xsl:stylesheet>
```

有了 XSLT 檔案後, 在伺服器端即可撰寫簡單的轉換程式, 將來源的 XML 和轉換的 XSLT 傳給 XslTransform 類別, 再呼叫 Transform() 方法, 即可產生轉換後的資料檔：

```
using System.IO;
using System.Xml;
using System.Xml.Xsl;

MemoryStream ms = new MemoryStream();
XmlDocument doc = new XmlDocument();
XmlDocument docXslt = new XmlDocument();
XslTransform xslt = new XslTransform();

doc.LoadXml(myXML); // 載入來源 XML。
docXslt.LoadXml(myXSLT); //載入要處理轉換的 XSLT。
xslt.Load(docXslt); // 將 XSLT 載入到 XslTransform 類別物件中。
// 執行 XML->XSLT->XML 轉換, 並將結果寫入到 MemoryStream。
xslt.Transform(doc, null, ms);

ms.Flush();
ms.Position = 0;
```

```
StreamReader reader = new StreamReader(ms);
string xml = reader.ReadToEnd(); // 讀出轉換後的結果。
ms.Close();

Response.ContentType = "text/xml";
Response.Write(xml); // 將轉換後的結果輸出。
Response.End();
```

XSLT 的指令相當的多, 除了一定要的宣告以外, 還要配合來源檔的格式來設定 XPath 和要產生的目標格式的結構, 如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 宣告 XSLT 指令版本及所使用的命名空間 -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- 宣告輸出類型以及編碼等 -->
  <xsl:output method="xml" indent="yes" encoding="utf-8"
    version="1.0"/>
  <!-- 設定 template 對應的來源資料 ("/"表示根節點) -->
  <xsl:template match="/">
    <ProductList>
      <Items>
        <!-- 設定瀏覽用 XPath 指定路徑的節點 -->
        <xsl:for-each select="ProductList/Items/Item[@AddTime != '']">
          <!-- 設定排序 -->
          <xsl:sort select="@AddTime" order="ascending"
            data-type="text" />
          <Item>
            <!-- 輸出屬性 -->
            <xsl:attribute name="ProductID">
              <!-- 輸出由 XPath 指定路徑的來源資料的值 -->
```

```
              <xsl:value-of select="@ProductID"/>
              </xsl:attribute>
                <xsl:attribute name="Barcode">
              <xsl:value-of select="@Barcode"/>
              </xsl:attribute>
                <xsl:attribute name="AddTime">
              <xsl:value-of select="@AddTime"/>
              </xsl:attribute>
                <xsl:attribute name="TypeNum">
              <xsl:value-of select="@TypeNum"/>
              </xsl:attribute>
                <xsl:attribute name="Description">
              <xsl:value-of select="@Description"/>
              </xsl:attribute>
              ... <!-- 由於太多, 其他雷同的程式就不列舉了  -->
          </Item>
        </xsl:for-each>
      </Items>
    </ProductList>
  </xsl:template>

</xsl:stylesheet>
```

由於 XSLT 的指令與使用方法, 多到可以撰寫成一本書了, 因此筆者無法做更詳細的介紹, 若對 XSLT 有興趣的讀者, 可到書局找尋與 XML 有關的專書, 裡面通常都會介紹到 XSLT。

解決方案

可依據對方公司系統產生的 XML 格式, 撰寫 XSLT 轉換指令檔, 然後經由 System.Xml.Xsl 命名空間中的 XslTransform 執行轉換即可。

Q45 如何像是 Google 那樣分頁顯示查詢結果？

適用範圍： V ASP.NET 2.0 V ASP.NET 3.5

問題

我使用 ASP.NET 2.0 開發網站, 並且使用 GridView 來做資料瀏覽的網頁, 但是 GridView 預設的分頁功能太不方便了, 我想要做一個自己的分頁方式, 就像 Google 那樣的, 那我要如何做？

問題說明

分頁功能 (Paging) 是在實作資料處理的用戶端程式時的基本能力, 在大量資料的情況下, 分頁功能可以減少傳送到用戶端的資料量, 使用者也不必直接面對大量的資料, 而且可以做到像瀏覽進度, 或是頁數提示這樣的功能, 在增進使用者導覽性 (User Navigation) 上, 有正面的幫助。

不過相信不管是用過.NET 1.x 的 DataGrid 還是.NET 2.0 的 GridView, 都對預設的分頁導覽器 (Pager) 有著同樣的觀感：太單調了, 只有前後移動和數字頁碼的方式可以選擇, 如果使用者想要做到更多, 例如下拉式選單的分頁瀏覽法, 或者要提供像是下載資料 (例如 GridView 中的資料另存成 CSV 檔案) 等特異功能的話, 依預設的作法是做不到的, 但 GridView 保留了由開發人員自訂分頁導覽列的能力, 即分頁導覽列樣板 (Page Template) 。

```
<asp:GridView ID="myGridView" runat="server">
  <PagerTemplate>
    <!-- 放置控制分頁的控制項 -->
  </PagerTemplate>
</asp:GridView>
```

分頁導覽列樣版中控制項的處理方法, 和處理一般的<TemplateField>是一樣的, 只有幾點不同：

- 在 RowDataBound 填充資料時, 分頁導覽列的 RowType 是 Pager。
- 如果在分頁導覽列樣板中由按鈕控制項(Button, LinkButton 或ImageButton)引發分頁事件, 則要自行處理 PageIndexChanging 事件。
- 分頁導覽列樣板中的控制項不可以使用資料繫結語法。
- 如果要在分頁導覽列樣板中的非按鈕型控制項 (例如 DropDownList) 引發事件, 則控制項事件必須要各自實作相對應的處理常式, 而不是由 RowCommand 事件常式處理。

由於分頁導覽列樣板的功能與<TemplateField>相近, 因此也是相當的強大, 只要能夠善用它, 各式各樣的分頁導覽方法都可以實作出來。

解決方案

首先, 先將 GridView 中的分頁導覽列樣板設計出來, 筆者以一個較簡單, 具有下拉式方塊, 以及上下頁導覽的功能來說明如何實作自訂的分頁導覽列, 頁面使用的 HTML 指令碼如程式 1 所示。

程式 1. 具分頁能力的 GridView 的 HTML 指令碼 (重點在 PagerTemplate)

```
<asp:GridView ID="GridView1" runat="server" AllowPaging="True"
  CellPadding="4" ForeColor="#333333"
  onpageindexchanging="GridView1_PageIndexChanging"
  onrowcommand="GridView1_RowCommand"
  onrowdatabound="GridView1_RowDataBound">
  <PagerSettings Position="TopAndBottom" />
  <PagerStyle BackColor="White" ForeColor="Black"
    HorizontalAlign="Center" />
  <PagerTemplate>
```

```
  <asp:LinkButton ID="cmdToPreviousPage"
    CommandName="GoToPreviousPage"
    runat="server" Text="<< 上一頁" />    
    資料筆數：<asp:Label ID="labelRowCount" runat="server" />
      
    目前頁數：<asp:Label ID="labelPageCount" runat="server"
    Font-Bold="true"/>    
    前往第<asp:DropDownList AutoPostBack="true"
      OnSelectedIndexChanged="PagerDropDown_SelectedIndexChanged"
      ID="cboPageList" runat="server" />頁    
  <asp:LinkButton ID="cmdToNextPage" CommandName="GoToNextPage"
    runat="server" Text="下一頁 >>" />
  </PagerTemplate>
</asp:GridView>
```

這個導覽列的樣子是：

資料筆數：77 目前頁數：0 前往第 0 頁 下一頁 >>

SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	Uni
1	1	10 boxes x 20 bags	18.0000	39	0
1	1	24 - 12 oz bottles	19.0000	17	40
1	2	12 - 550 ml bottles	10.0000	13	70
2	2	48 - 6 oz jars	22.0000	53	0
2	2	36 boxes	21.3500	0	0
3	2	12 - 8 oz jars	25.0000	120	0
3	7	12 - 1 lb pkgs.	30.0000	15	0
3	2	12 - 12 oz jars	40.0000	6	0
4	6	18 - 500 g pkgs.	97.0000	29	0
4	8	12 - 200 ml jars	31.0000	31	0

資料筆數：77 目前頁數：0 前往第 0 頁 下一頁 >>

點選上一頁或下一頁時, 會自動跳至上一頁或下一頁, 使用下拉式清單可以快速的翻頁, 當然讀者也可以將手動輸入的文字方塊加入, 但是要額外實作頁碼範圍檢查的機制, 以避免使用者輸入了超出範圍的頁碼, 造成錯誤發生, GridView 做分頁處理的程式碼如程式 2。

程式 2：做分頁處理的 GridView 相關事件程式

```
protected void GridView1_RowDataBound(object sender,
  GridViewRowEventArgs e)
{
  // 檢查是否在分頁導覽列。
  if (e.Row.RowType == DataControlRowType.Pager)
  {
    // 取得在分頁導覽列中的控制項。
    Label labelPageCount =
      e.Row.FindControl("labelPageCount") as Label;
    Label labelRowCount =
      e.Row.FindControl("labelRowCount") as Label;
    DropDownList cboPageList =
      e.Row.FindControl("cboPageList") as DropDownList;
    LinkButton cmdGoToPreviousPage =
      e.Row.FindControl("cmdToPreviousPage") as LinkButton;
    LinkButton cmdGoToNextPage =
      e.Row.FindControl("cmdToNextPage") as LinkButton;
    // 將分頁數加入下拉式方塊。
    for (int i = 0; i < (sender as GridView).PageCount; i++)
      cboPageList.Items.Add(
        new ListItem(i.ToString(), i.ToString()));
    // 將下拉式方塊的頁數設爲目前的頁數。
    cboPageList.SelectedValue =
      (sender as GridView).PageIndex.ToString();
    labelPageCount.Text =
      (sender as GridView).PageIndex.ToString();
    labelRowCount.Text = (((sender as GridView).DataSource) as
      DataTable).Rows.Count.ToString("###, ###, ##0");
```

```
    // 控制上一頁與下一頁分頁的按鈕狀態。
    if ((sender as GridView).PageIndex == 0)
      cmdGoToPreviousPage.Visible = false;
    if ((sender as GridView).PageIndex ==
      (sender as GridView).PageCount - 1)
      cmdGoToNextPage.Visible = false;
  }
}

protected void GridView1_PageIndexChanging(object sender,
  GridViewPageEventArgs e)
{
  // 在分頁發生時, 重設目前的分頁數 (PageIndex) 。
  this.GridView1.DataSource = this.LoadData();
  this.GridView1.PageIndex = e.NewPageIndex;
  this.GridView1.DataBind();
}

protected void GridView1_RowCommand(object sender,
  GridViewCommandEventArgs e)
{
  (sender as GridView).DataSource = this.LoadData();
  // 處理上一頁與下一頁的按鈕行為。
  switch (e.CommandName)
  {
    case "GoToPreviousPage":
      (sender as GridView).PageIndex -= 1;
      break;
```

```
    case "GoToNextPage":
      (sender as GridView).PageIndex += 1;
      break;

    default:
      break;
  }

  (sender as GridView).DataBind();
}
```

最後, 再處理下拉式清單的選擇程式 (SelectedIndexChanged 事件) 即可, 如程式 3。

程式3：處理下拉式清單選取分頁瀏覽的程式

```
protected void PagerDropDown_SelectedIndexChanged(object sender,
  EventArgs e)
{
  DropDownList dropdownList = sender as DropDownList;

  this.GridView1.DataSource = this.LoadData();
  this.GridView1.PageIndex =
    Convert.ToInt32(dropdownList.SelectedValue);
  this.GridView1.DataBind();
}
```

小常識 爲何不直接指定控制項？

由前面的程式 2 和程式 3 來看，讀者可能會認爲，直接指定控制項不是比較快嗎？爲何要用轉型的方式來取得 GridView？

這個問題問的好，筆者平時在撰寫程式時，有很大的機會會將程式碼搬來搬去，和其他頁面的控制項結合，此時若是直接指定控制項，等於搬一次就要改一次，用轉型的方法可以省下這個工夫，無形中可以省下不少的時間。

考生停看聽

本問題所討論的內容，可用來準備 Exam 70-528: TS: Microsoft .NET Framework 2.0 Web Client Development 的下列主題。

- 使用 ADO.NET 將資料整合到 Web 應用程式中。
  - 實作 Data-Bound (資料繫結) 控制項
    - 使用表格化資料來源控制項回傳表格資料。
    - 使用簡單資料繫結控制項來顯示資料。
    - 使用複合資料繫結控制項來顯示資料。

相關問題

- Q48：分頁要如何處理才會是速度最快的？

Q46 我使用 DataAdapter 做資料修改, 但下了 Update()指令後, 爲何沒有更新？

適用範圍：V ASP.NET 1.0 V ASP.NET 1.1 V ASP.NET 2.0 V ASP.NET 3.5

問題

我使用 ASP.NET 設計 Web 應用程式, 在程式碼中使用 SqlDataAdpater 來連線資料庫, 並且讀寫資料, 在讀取的時候沒問題, 但在寫入的時候, 資料庫卻沒有寫入資料, 請問原因爲何？

問題說明

SqlDataAdapter 是一個資料存取的管道之一, 它可以自動管理資料存取的連線, 並且可以用 SqlDataAdapter.Fill()一次把資料填入到 DataSet 或 DataTable 中, 然後在填入資料後自動將連線關閉, 讓資料庫連線得以釋出給其他的使用者使用。它也支援資料庫的更新, 透過 SqlDataAdapter.Update()指令來寫入資料。

SqlDataAdapter.Update()會偵測 SqlDataAdapter 的三個指令是否有設定, 若有, 才會透過指令來修改資料, 否則它不會做任何動作, 這三個指令是：

指令	說明
InsertCommand	插入資料的 SQL 指令。
UpdateCommand	修改資料的 SQL 指令。
DeleteCommand	刪除資料的 SQL 指令。

這三個指令在預設的時候是不會設定的, 需要由開發人員來設定, 例如可以用這樣的程式碼來實作：

```
SqlDataAdapter adapter = new SqlDataAdapter(
  "SELECT * FROM Customers", conn);
SqlCommandBuilder cb = new SqlCommandBuilder(adapter);
```

```
// 設定更新指令。
adapter.InsertCommand = cb.GetInsertCommand();
adapter.UpdateCommand = cb.GetUpdateCommand();
adapter.DeleteCommand = cb.GetDeleteCommand();

// 更新資料。
adapter.Update(dataTable);
```

或者, 可以自己設定指令文字與參數的方式來處理。

```
SqlDataAdapter adapter = new SqlDataAdapter(
  "SELECT CustomerID, CustomerName FROM Customers", conn);

// 設定更新指令。
adapter.InsertCommand = new SqlCommand(
  @"INSERT INTO Customers (CustomerID, CustomerName) VALUES (@customerID,
@customerName)", conn);
adapter.InsertCommand.Parameters.Add("@customerID",
  SqlDbType.Uniqueidentifier);
adapter.InsertCommand.Parameters.Add("@customerName",
  SqlDbType.Nvarchar);r.niqueidentifier);ameters.Add("ustomerName)"

// 更新資料。
adapter.Update(dataTable);
```

小常識 **除非必要, 否則應避免使用指令自動產生器, 例如 SqlCommandBuilder 或 OleDbCommandBuilder 這些物件。**

SqlCommandBuilder 這類的 SQL 指令自動產生器可以自動的產生必要的 INSERT/UPDATE 及 DELETE 指令, 雖然可以很快的做到想要做的工作, 但是似乎很少人去看它產生的是怎麼樣的 SQL 指令, 筆者輸出一個用 SqlCommandBuilder 產生的 UPDATE 指令：

```
UPDATE [Customers] SET [CustomerID] = @p1, [ShopID] = @p2, [Barcode]
= @p3, [ChtName] = @p4, [EngName] = @p5, [SIDKey] = @p6, [Birthday]
= @p7, [Address] = @p8, [District] = @p9, [City] = @p10, [Country] =
@p11, [PostalCode] = @p12, [Phone] = @p13, [BusinessPhone] = @p14,
[Cellphone] = @p15, [Gender] = @p16, [MarryState] = @p17, [BonusPoint]
= @p18, [Email] = @p19, [Upper] = @p20, [Lower] = @p21, [Waist] =
@p22, [Career] = @p23, [IsAcceptDM] = @p24, [Comments] = @p25,
[CreateDate] = @p26 WHERE (([CustomerID] = @p27) AND ([ShopID] =
@p28) AND ([Barcode] = @p29) AND ((@p30 = 1 AND [ChtName] IS NULL) OR
([ChtName] = @p31)) AND ((@p32 = 1 AND [EngName] IS NULL) OR ([EngName]
= @p33)) AND ((@p34 = 1 AND [SIDKey] IS NULL) OR ([SIDKey] = @p35))
AND ([Birthday] = @p36) AND ([Address] = @p37) AND ([District] = @p38)
AND ([City] = @p39) AND ([Country] = @p40) AND ([PostalCode] = @p41)
AND ((@p42 = 1 AND [Phone] IS NULL) OR ([Phone] = @p43)) AND ((@p44
= 1 AND [BusinessPhone] IS NULL) OR ([BusinessPhone] = @p45)) AND
([Cellphone] = @p46) AND ([Gender] = @p47) AND ([MarryState] = @p48)
AND ([BonusPoint] = @p49) AND ((@p50 = 1 AND [Email] IS NULL) OR
([Email] = @p51)) AND ((@p52 = 1 AND [Upper] IS NULL) OR ([Upper] =
@p53)) AND ((@p54 = 1 AND [Lower] IS NULL) OR ([Lower] = @p55)) AND
((@p56 = 1 AND [Waist] IS NULL) OR ([Waist] = @p57)) AND ((@p58 = 1
AND [Career] IS NULL) OR ([Career] = @p59)) AND ([IsAcceptDM] = @p60)
AND ((@p61 = 1 AND [Comments] IS NULL) OR ([Comments] = @p62)) AND
([CreateDate] = @p63))
```

一個資料列的 Update, 卻要判斷如此多的條件才能做, 若是在一個資料存取繁忙的系統上, 這種資料存取方法可能會拖慢效能, 因此, 除非欄位眞的多到寫 SQL 太麻煩, 才考慮使用 Command Builder, 而且要有效能被拖慢的心理準備。

那爲什麼會發生 Update 沒有寫入資料庫的狀況呢？

除了設定指令以外, DataRow 本身的狀態也是影響是否執行 Update 指令的因素之一。在 DataRow 的成員中, 有一個 RowState 屬性, 它是用來決定目前資料列的更新狀態, 當使用者在編輯修改 DataTable 中的 DataRow 時, 會觸動這個屬性按照使用者編輯的結果而改變。DataRow.RowState 的可用屬性值如表所示。

屬性值	對應的指令	說明
Added	InsertCommand	此資料列爲新增的資料列。
Unchanged	無	此資料列未做任何變更。
Modified	UpdateCommand	此資料列已修改過。
Deleted	DeleteCommand	此資料列已刪除。
Detached	無	在 DataRow 未加入 DataTable 時, 或是在 DataRow 被刪除, 而且變更已認可後。

因此, 若是 Update()指令未被觸動時, 就表示 DataRow 的 RowState 可能是在 Unchanged, 或者是 Detached, 大多數的案例以 Unchanged 爲多, 有可能是因爲在 Update()之前, 呼叫了 DataTable.AcceptChanges()或是 DataTable.RejectChanges() 將資料列的 RowState 改變回 Unchanged, 讓 Update()無法得知要修改的資料列有哪些。例如下列的程式碼：

```
SqlDataAdapter adapter = new SqlDataAdapter(
  "SELECT CustomerID, CustomerName FROM Customers",
  "initial catalog=TestDB; integrated security=SSPI");

DataTable table = new DataTable();

adapter.Fill(table);
```

```
// 資料列被新增, 新增列的 RowState 爲 Added。
table.Rows.Add(new object[] { Guid.NewGuid(), this.T_Name.Text });
// 呼叫了 AcceptChanges(), 所有資料列的 RowState 會變爲 Unchanged
table.AcceptChanges();

adapter.InsertCommand = new SqlCommand(
  "INSERT INTO Customers (CustomerID, CustomerName) VALUES" +
  " (@customerID, @customerName)", adapter.SelectCommand.Connection);
adapter.InsertCommand.Parameters.Add("@customerID",
  SqlDbType.UniqueIdentifier, 16, "CustomerID");
adapter.InsertCommand.Parameters.Add("@customerName",
  SqlDbType.NVarChar, 50, "CustomerName");

// 資料更新會失效。
adapter.Update(table);
```

解決方案

爲了確保修改過的 DataRow 可以寫入資料庫中, 程式碼中要做到兩件事：

- 設定好 DataAdapter 的 InsertCommand, UpdateCommand 及 DeleteCommand。
- 在下 DataAdapter.Update()前, 不要呼叫 DataTable.AcceptChanges()或 DataTable.RejectChanges(), 以確保 DataRow 的 RowState 是修改的狀態。

小常識 使用 SqlDataAdapter.UpdateBatchSize 做批次的更新

SqlDataAdapter 在大量進行更新時, 可以透過設定 UpdateBatchSize 來加大一次傳送的批次指令數量, 例如設定爲 10, 代表可一次傳送 10 個陳述式, 設爲 50 則是一次傳送 50 個陳述式, 若設爲 1 則是停用批次更新。若批次太大可能會降低資料庫的效能, 因此想要利用批次更新加快速度, 最好是先測試資料庫的負載狀況, 再決定要設定的值爲何。

考生停看聽

本問題所討論的內容, 可用來準備 Exam 70-528: TS: Microsoft .NET Framework 2.0 Web Client Development 的下列主題。

- 使用 ADO.NET 將資料整合到 Web 應用程式中。
  - 在離線環境中新增、刪除與編輯資料。
    - 使用 Command Builder 自動產生 DataAdapter 的指令。
    - 程式化產生 DataAdapter 的指令。
    - 使用 DataAdapter 更新資料庫。
    - 使用 DataAdapter 執行批次作業。

Q47 如何將圖檔儲存在資料庫中？

適用範圍：VASP.NET 1.0 VASP.NET 1.1 VASP.NET 2.0 VASP.NET 3.5

問題

公司要求要開發一套知識管理系統 (KMS) , 而且在設計上需要允許使用者上傳檔案, 我們公司使用的資料庫是 SQL Server 2005, 那我要如何撰寫儲存與讀取檔案的程式碼, 開放給使用者下載？

問題說明

檔案上傳與管理在多數的 Web 應用程式中是很常見的, 例如像是文件檔案上傳、圖片上傳 (相簿) 、檔案分享等等, 檔案上傳在 ASP.NET 可以說是很簡單的事, 只要利用 FileUpload (ASP.NET 2.0) 控制項就可以了。

不過有時問題往往不會這麼簡單：檔案傳到伺服器後, 要如何管理呢？檔案是否是機密？檔案要儲存在哪裡？檔案是否要壓縮？檔案的類型是什麼, 檔案的存取權限爲何？…這麼多的問題, 才是檔案上傳眞正要碰觸到的, 和這些問題比起來, 檔案上傳的方法其實眞的只是小 Case…。

在較重視知識管理與工作流程 (Workflow) 簽核作業的公司中, 對於檔案管理是相當要求的, 而且可能有許多檔案是涉及公司機密的, 這類型檔案的管理就很重要, 也必須要考量到較多的安全機制, 這時資料庫的地位就變得重要多了。資料庫可以集中控管, 並可避免具有權限的網管人員私下經由網路分享路徑來存取機密檔案, 資料庫的管理和控管都交由 DBA (資料庫管理員) 來負責, 管理上就會單純不少, 不過, 可累了 DBA 了。

用資料庫還是用檔案系統來儲存檔案, 要看公司的政策, 以及資料庫伺服器是否能夠承受以及快速的存取檔案, 原因是檔案在讀寫進資料庫時, I/O 的量會變的很大, 如果伺服器的 I/O 速度不夠快, 系統的速度就會變慢, 這個差異在大型檔案的讀寫會特別明顯, 但若用檔案系統來儲存, 又會有安全性以及 Web 應用程式的存取識別問題, 因此要用哪一種方式來儲存, 需要好好的考量與設計。

若決定要存在資料庫時, 請注意幾個事項：

- 儲存檔案的欄位不可設定索引, 因爲會讓索引大小暴增。
- 使用 varbinary(max) (SQL Server 2005) 或是 image (SQL Server 2000 以前版本) 來存放, 最大可以存放 2GB 的檔案。
- 伺服器的 I/O 速度要夠快, 不然就是存入的檔案不要超過 10MB 以上 (若伺服器 I/O 夠快則可不必考慮) , 若是 TIFF 或是 BMP 影像檔, 建議先壓縮後再存。
- 將檔案的名稱, 以及檔案的類型存起來, 會在讀出的時候用到。

SQL Server 2005 的 varbinary(max)是用來取代原本的 image (BLOB) 資料型別, 以放置大型二進位資料的資料型別, 雖然資料型別有異動, 但讀寫方法沒有太大的變化, 仍然是利用 SqlCommand 來讀寫, 在設計 SQL 指令的時候, 應採用參數化查詢的方法來處理, 而不是用字串連結的方法做, 畢竟檔案的二進位資料流可不像一般的資料一樣。

首先, 先將上傳的檔案取出位元組陣列的資料, 然後將資料指定給 SqlCommand 的參數, 長度必須要設定爲檔案的大小 (亦即位元組陣列的大小) , 然後執行指令, 將 SQL 送入資料庫即可 (如程式 1) 。讀取的話則要注意, 在讀取大型二進位資料時, 最好是能夠單純化, 也就是只要讀取檔案內容就好了 (如程式 2) 。

程式碼1：寫入檔案到資料庫的範例程式

```
public void AddFile(string FileName, string FileType, byte[] data)
{
  SqlConnection conn = new SqlConnection("...");
  SqlCommand cmd = new SqlCommand(
"INSERT INTO Files (FileName, FileType, FileData) VALUES (@fname, @ftype,
@fd)",
    conn);
```

```
  cmd.Parameters.Add("@fname", SqlDbType.NVarChar, 50);
  cmd.Parameters.Add("@ftype", SqlDbType.VarChar, 10);
  // 設定大小
  cmd.Parameters.Add("@fd", SqlDbType.VarBinary, data.Length);
  cmd.Parameters["@fname"].Value = FileName;
  cmd.Parameters["@ftype"].Value = FileType;
  cmd.Parameters["@fd"].Value = data; // 設定檔案資料。

  conn.Open();
  cmd.ExecuteNonQuery();
  conn.Close();

  cmd.Dispose();
  conn.Dispose();
}
```

程式碼2：由資料庫讀取檔案的範例程式

```
public byte[] GetFile(string FileID, out string FileName)
{
  byte[] data = null;
  SqlConnection conn = new SqlConnection(
    "initial catalog=TestDB; integrated security=SSPI");
  SqlCommand cmd = new SqlCommand(
    "SELECT FileName FROM Files WHERE FileID = @fileID; " +
    "SELECT FileData FROM Files WHERE FileID = @fileID", conn);

  cmd.Parameters.Add("@fileID", SqlDbType.UniqueIdentifier, 16);
  cmd.Parameters["@fileID"].Value = new Guid(FileID);

  conn.Open();
```

```
  SqlDataReader reader = cmd.ExecuteReader(
    CommandBehavior.SequentialAccess |
    CommandBehavior.CloseConnection);

  FileName = string.Empty;

  while (reader.Read())
    FileName = reader.GetValue(0).ToString();

  reader.NextResult();
  reader.Read();
  // 建立檔案存放區。
  data = new byte[reader.GetBytes(0, 0, null, 0, Int32.MaxValue)];
  // 讀取檔案資料。
  reader.GetBytes(0, 0, data, 0, data.Length);
  reader.Close();

  reader = null;
  cmd.Dispose();
  conn.Dispose();

  return data;
}
```

解決方案

利用 SQL Server 2005 資料庫的 varbinary(max)型別, 在資料庫中開一個儲存檔案的資料表, 然後撰寫讀取與寫入檔案資料的程式碼 (如程式 1 及程式 2) 即可。

相關問題

- Q25：如何動態產生圖片?

Q48 分頁要如何處理速度才會快？

適用範圍：V ASP.NET 1.0 V ASP.NET 1.1 V ASP.NET 2.0 V ASP.NET 3.5

問題

我用 ASP.NET 開發系統, 在瀏覽資料的時候也是用 GridView 並加上分頁來做, 但是因爲資料量很大 (約 100 萬筆) , 現在在做分頁的時候都很慢, 請問我要如何改善？

問題說明

在大型的應用程式中, 資料庫的筆數通常都會很大, 尤其像是交易頻繫的 TPS (Transaction Processing System) , 或是規模較大的 POS 系統, 以及製造業的 ERP 系統等, 資料量都會在幾百萬筆甚至幾千萬或幾億都有, 在這種大型資料庫中如果要實作分頁瀏覽功能, 就不能用太馬虎的方法做了。

一般在資料量少的時候, 會因應 ASP.NET 平台上的需要, 先撈取全部的資料後再做分頁, 也就是這樣的程式碼：

```
SqlConnection conn = new SqlConnection(
  "initial catalog=db; user id=xx; password=xx");
SqlDataAdapter adapter = new SqlDataAdapter(
  "SELECT * FROM Products", conn);
DataTable table = new DataTable();

adapter.Fill(table); // 一次撈取全部的資料
adapter.Dispose();
conn.Dispose();

this.GridView1.DataSource = table;
this.GridView1.PageIndex = 5;
this.GridView1.DataBind();
```

這樣的作法會有一個大問題, 若資料量是 1, 000, 000 筆的話, 如果爲了一個 50 筆的分頁去撈 20, 000 倍的資料回來, 對應用程式和資料庫的效能其實都是很傷的, 假設取 10, 000 筆資料要 0.5 秒, 那麼 1, 000, 000 筆資料就要 50 秒才會回傳, 等於前端要出現分頁結果得要一分鐘左右 (加上前端處理的時間) , 撇開時間不算, 若 1 筆資料是 0.5KB, 1, 000, 000 筆就要 500MB 的資料量, 而且在 Web 環境又是多人使用, 只要有幾個人去查詢的話, 網路可能就會爆掉了。

所以在大型資料庫或具有大量資料列的應用程式中, 都要利用分頁化查詢 (Paged Query) 來實作分頁機制, 如此可以減輕許多在資料庫 I/O, 網路頻寬以及前端處理上的負載。

所謂的分頁化查詢, 意指在資料庫查詢時, 就做好分頁的處理, 讓查詢可以只傳回分頁所需要的資料, 要做到這樣程度, 就需要在每個查詢設計上要抓住查詢的重點, 亦即利用條件式, 以及 TOP 等限制資料回傳數目的指令, 做資料提取的工作。

例如, 若想要查詢交易記錄中的 21-40 筆, 可以這樣下 SQL 指令：

```
SELECT TOP 20 TxDate, TotalAmount FROM
(SELECT TOP 40 TxDate, TotalAmount FROM SaleTxRecords
  ORDER BY TxDate DESC) q
ORDER BY TxDate
```

這種查詢法, 筆者稱它爲「夾擠查詢法」, 它是先將前 40 筆輸出後, 再反排序取回前 20 筆, 如此就可以取到 21-40 筆的資料, 這個方法適用於大多數的 DBMS, 只要支援 TOP 以及 ORDER BY 的資料庫都可以使用這種技巧。

在 SQL Server 2005 中, 新增了一個 RANK()函數, 它可以依順序產生序列編號, 因此可以用來做過濾：

```
SELECT * FROM (
SELECT RANK() OVER (ORDER BY TxDate DESC)
  AS RankNum, TxDate, TotalAmount FROM SaleTxRecords ) q
WHERE RankNum BETWEEN 21 AND 40
```

另一個要考量的是查詢最佳化 (Query Optimization) , 查詢最好掃瞄的資料列愈少愈好, 而且能夠將條件落在索引 (index) 上, 如此可以讓分頁查詢的速度更快, 也就是要避免表格掃瞄 (Table Scan) , 表格掃瞄是 SQL Server 的執行過程之一, 通常表格掃瞄會產生較高的 I/O, 而且要花的時間也會比較長, 若是索引掃瞄 (Index Scan) 的話, 花費的時間就會比較短一些, 當然也要視資料庫當時的負載如何, 就一般情況而言, Index Scan 會比 Table Scan 要快。當然查詢最佳化不是只有這樣而已, 讀者可以找尋資料庫的專書來學習。

解決方案

將需要分頁的查詢, 以分頁化查詢的方式改寫, 並且要注意查詢執行時的效能, 盡可能的把查詢量降低, 並善用索引來降低對資料表的 I/O, 如此可以讓速度加快。

Q49 我要如何輸出 Excel 檔案？

適用範圍： V ASP.NET 2.0 V ASP.NET 3.5

問題

我用 ASP.NET 開發系統, 使用 GridView 來製作資料瀏覽器, 但使用者要求資料可以下載成 Excel 檔案, 請問我有什麼辦法, 可以在 ASP.NET 中輸出 Excel 的試算表檔案？

問題說明

可輸出 Excel 檔案的問題, 在論壇上幾乎可以算是 FAQ 的常客, 這也反映出了有很大的需求, 畢竟 Excel 有太多使用者, 他們都已經習慣而且仰賴 Excel 做一些資料分析的工作, 若要叫這些使用者自己去抓 GridView 中的資料再貼到 Excel 的話, 可能會被使用者追著打吧 ☺。

在 Office 2000 之前, Excel 的試算表檔案格式是二進位的, 也就是若要讀寫 Excel 檔案, 只能透過 Excel 本身的物件模型和 OLE DB Provider for Jet 兩個管道來做, 開發人員基本上是無法直接去存取和修改的 (除非知道 Excel 二進位檔案格式) , 不過自 Office 2000 開始, Excel 檔案格式已經可以由 HTML 來取代 (Excel 會辨識 HTML 和 Excel 二進位檔格式) , 到了 Office XP 以後的版本, 更能夠透過 Spreadsheet ML (試算表標記語言) , 以產生 XML 文件的方法, 讓 Excel 可以存取以 Spreadsheet ML 所產生的 XML 資料, 到了 Office 2003, Spreadsheet ML 更名爲 Excel XML, 在 2007 則更進一步納入 Office Open XML 規格中, 至此, Excel 和 XML 的資料交換已經完全成形。

然而是否要使用到 Excel XML, 見人見智, 畢竟 Excel XML 和 Word XML 以及其他的 Office Open XML 的複雜程度都差不多, 而且有很多的 namespace 以及宣告要學習 (經過一個版本就會至少有一個以上的 namespace) , 倘若沒這麼多時間, 其實還是可以直接用 HTML, 輸出表格資料的方式, 來讓 Excel 辨識出資料, 並產生試算表的介面。

若想要輸出 HTML 表格讓 Excel 可以開啓時, 可利用下列的程式碼：

```
// 建立 StringWriter, 用來裝載輸出的 HTML 碼。
StringWriter sw = new StringWriter();
// 使用 StringWriter 建立 HTML TextWriter, 產生 HTML 碼。
HtmlTextWriter hw = new HtmlTextWriter(sw);
// 設定檔案內容爲 Excel 檔案。
Response.ContentType = "application/vnd.ms-excel";
// 設定內文的編碼。
Response.ContentEncoding = System.Text.Encoding.Default;
// 設定檔案下載的屬性。
Response.AddHeader("Content-Disposition",
  string.Format("attachment; filename=MyExcel.xls"));
// 產生 GridView
this.GridView1.DataSource = this.LoadData();
this.GridView1.DataBind();
// 以程式自動對 GridView 進行繪製,
// 此程式會讓 GridView 輸出 HTML 碼到 HTML TextWriter 中。
this.GridView1.RenderControl(hw);
// 將 HTML 碼輸出到前端。
Response.Write(sw.ToString());
Response.End();
```

另外, 在 ASP.NET 2.0 中, 在預設的情況下, ASP.NET 執行引擎會檢查來自用戶端的事件參數內容, 以防止有可能以 HTTP POST 爲主的攻擊手法, 不過這會讓在程式使用的 GridView.RenderControl()發生錯誤狀況：「型別'GridView'的控制項'GridView1' 必須置於有 runat=server 的表單標記之中。」, 因此需要把檢查事件參數的功能關閉, 才能讓 GridView.RenderControl()正常運作。

爲了防止 GridView.RenderControl()檢查事件資訊, 要先在 Page 的宣告中加入EnableEventValidation="false",再於程式碼檔案中加入覆寫VerifyRenderingInServerForm 方法的程式, 將預設的行爲取消掉。

```
<!-- 在 ASPX 檔案中 -->
<%@ Page Language="C#" AutoEventWireup="true"
  EnableEventValidation="false" CodeFile="DownloadExcel.aspx.cs"
  Inherits="Part4_DownloadExcel" %>

// 在程式碼檔案中
public override void VerifyRenderingInServerForm(Control control)
{
  // 取消預設的驗證行爲。
  // base.VerifyRenderingInServerForm(control);
}
```

如此即可將 GridView 的資料輸出到 Excel 檔案中, 唯一的缺點就是每次在 Excel 開啓檔案時都會提示「檔案格式不是預設的格式, 是否要開啓」的訊息。如果使用者不在意的話, 這樣做其實就夠了。

不過輸出 Excel 資料檔不是只有這個方法, 另外還可以透過存取 Excel 物件模型的方式產生 Excel 檔案, 這個方法可以確實的產生原生的 Excel 二進位資料檔, 不過它卻有一個重大的缺點, 每次在建立 Excel 的物件模型時, 都會產生一個 Excel 的執行個體在伺服器上, 在多人的環境下, 如果有 100 個使用者產生了 Excel 執行個體, 那麼可能記憶體被耗用的狀況會非常嚴重 (Excel 一個執行個體可能要 500KB～數 MB) , 而且在多人環境的 ASP.NET 應用程式中, 可能會發生像資源無法釋放, 以及鎖定之類無法事先預期的情況發生, 因此應不要在 Web 應用程式中去存取 Excel 物件模型 (不僅是 Excel, 其他的 Office 應用程式的物件模型都一樣) , 以避免資源佔用以及可能發生的未預期錯誤狀況, 所以筆者就不展示由 Excel 物件模型來產生 Excel 檔案的方法。

另一個方式就是利用 OLE DB Provider for Jet 來實作, 它可以在沒有 Excel 檔案的情況下, 產生 Excel 試算表, 每一個試算表都是一個表格, 可以透過 INSERT 或 UPDATE 的方式來輸入或修改資料；使用 CREATE TABLE 來建立新的試算表。

不過它仍然有缺點, 就是檔案管理以及資源獨佔性的問題。試想, 如果每個使用者都用 OLE DB Provider for Jet 來存取相同檔名的檔案, 那一定會有獨佔性的問題, 而若要產生不同名稱的檔案, 時間一長, 又會有檔案過多的管理問題, 因此在多人的環境下, 最好也是不要使用 OLE DB Provider for Jet 來讀寫 Excel 檔案資料。

```
using System.Data;
using System.Data.OleDb;
...
// 設定連線, 並指向 Excel 檔案的格式。
OleDbConnection conn = new OleDbConnection(
"Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\MyExcel.xls; Extended
Properties='Excel 8.0;HDR=YES'");
// 設定查詢指令
OleDbCommand cmd = new OleDbCommand("SELECT * FROM [Sheet1$]", conn);
OleDbDataAdapter adapter = new OleDbDataAdapter(cmd);
DataTable table = new DataTable();
// 填充 DataTable
adapter.Fill(table);

adapter.Dispose();

// 處理由 Excel 的 Sheet1 試算表取出的資料。
```

若想要知道更多的 OLE DB Provider for Jet 存取 Excel 檔案的方法, 可以查詢微軟知識庫中的 Q316934 號文章, 有針對 Excel 可用的 SQL 範圍以及限制的說明。或者也可以參考 Microsoft Press 所出的「.NET Development for Office」一書。

解決方案

將 GridView 以輸出 HTML 的方法, 來轉換成 Excel 的檔案, 在 Office XP 以後的版本可獲得較充份的支援, 若是 Office 2000 前的版本, 則可以用 OLE DB Provider for Jet 的方法來產生 Excel 檔案, 但要注意檔案管理的問題, 最好不要用到 Excel 物件模型, 否則要自行處理記憶體資源以及釋放的問題, 容易造成系統不穩定。

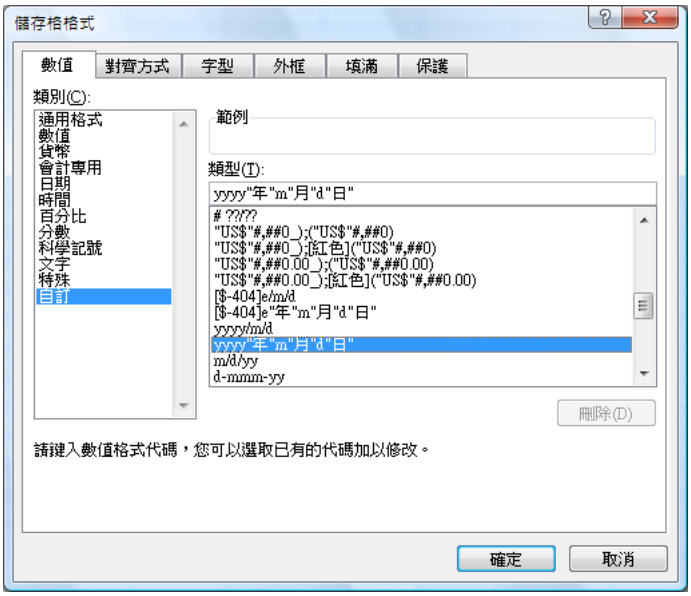
小常識　如何控制 GridView 中的資料格式？

如果使用的是 GridView 匯出 Excel 資料的話, 可以善加利用 Excel 的數值格式功能來格式化字串, 只要於 RowDataBound 事件常式, 在輸出的<td>中套用 mso-number-format 的樣式值即可, 例如可以用下列的程式碼來輸出文字字串：

```
private void GridView1_RowDataBound(object sender,
  GridViewRowEventArgs e)
{
  if (e.Row.RowType == DataControlRowType.DataRow)
  {
   e.Row.Cells[0].Attributes.Add("style", "mso-number-format: \\@");
  }
}
```

接下頁

若宣告時碰到特殊字元時, 要用 "\" 來做區分處理 (C# 才有的狀況, VB 可只用 \@ 來做) , 在 Excel 的儲存格格式對話方塊中的「自訂」一區所列出的格式化碼：



都可以適用在此, 也可以自己撰寫, 但是要注意是否能成功, 否則如果沒有解析成功時, 一律都會被當成通用格式處理, 建議讀者可以先在 Excel 中測試格式化字串, 如果可以使用時, 再將它套用到 mso-number-format 中。

Q50 如何在網頁中顯示子母 (Master-detail) 型資料？

適用範圍： ✓ASP.NET 2.0 ✓ASP.NET 3.5

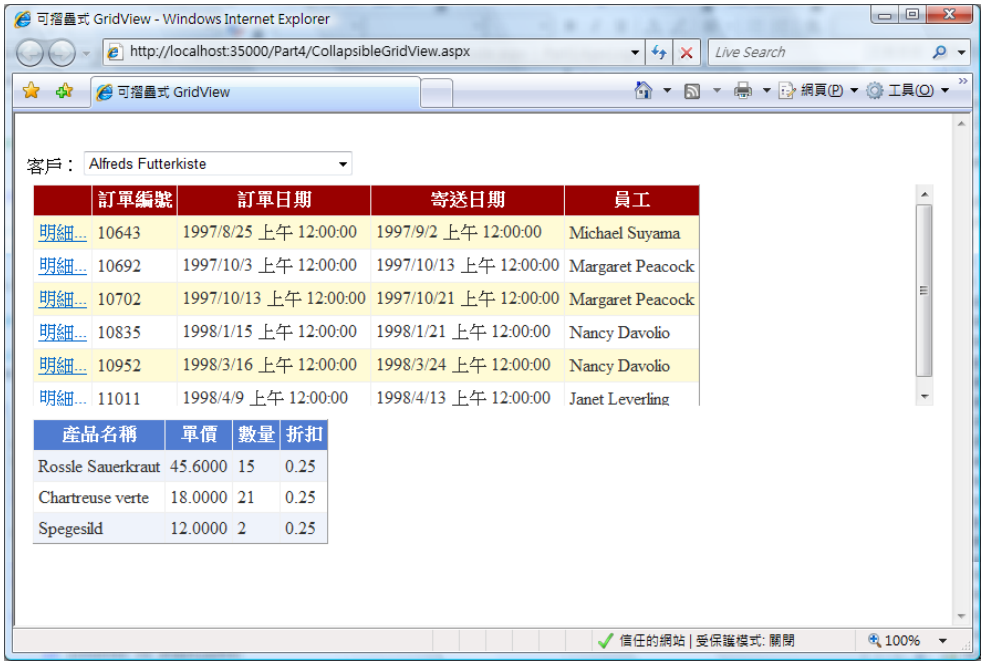
問題

我使用 ASP.NET 2.0 開發網站, 並且使用 GridView 來實作會員瀏覽的網頁, 但是我想要在會員有訂單時可以有一個介面來瀏覽訂單資訊, 而且就直接在同一個網頁中顯示, 不另外打開視窗或網頁, 請問我要如何做呢？

問題說明

誠如筆者在 Q35 中所提, GridView 的能力是超乎許多開發人員或網頁設計人員的想像, 只要能夠善用 HTML、Scripting (JavaScript) 以及 ASP.NET 的能力, 其實可以變化出的花樣, 也許之前想都沒想過。

在多數 Master-Detail 的應用程式中, 往往都是用不同的視窗來顯示主檔以及附檔的資料, 例如像這樣的模式：



不過這樣子的感覺總是有點怪怪的, 因爲每次點了 Master (主檔) 的明細時, 它就會跳到最上方, 如果是用在大量資料的話, 這似乎就不是那麼好, 雖可能有方法改善 (這不是本篇文章的重點) , 但是如果將明細直接顯示到 Master 記錄, 像這樣：

是不是就比較清楚了呢？

如果想要做到這樣的能力, 必須要滿足三件事情：

- 取得要求明細的主檔列的列位置 (Row-position) 資料。
- 新增新的資料列, 並將明細資料插入新的資料列中。
- 將明細資料列插入要求明細的主檔列下方。

其中以第一件事情最難做到, 筆者待會再提, 先提比較容易的第二件與第三件事。

解決方案

GridView 本身有一個在技術文件上鮮少看到的功能, 就是它可以由開發人員自行在 GridView 中自行插入新的列, 這個列物件是 GridViewRow, 讀者可以把它當成一般的 TableRow 來看, 但它多了幾個性質, 從它的建構函式就可以得知：

```
public GridViewRow (
  // 列索引
  int rowIndex,
  // 資料來源索引
  int dataItemIndex,
  // 列型態 (DataRow, Pager, Footer, Separator, EmptyDataRow
  // 或 Header 等)
  DataControlRowType rowType,
  // 列狀態 (Normal, Selected, Insert, Alternate, Edit 等)
  DataControlRowState rowState
)
```

而若想要插入 GridViewRow 到 GridView 中的話, 必須要在 GridView.RowCreated 事件常式中, 並且依照順序來插入, 否則新插入的 GridViewRow 會走位, 而順序是 RowIndex+2, 若是 RowIndex, 則新的資料列會出現在標題列上方：

而若是 RowIndex+1, 則新的資料列會出現在點選的那一列的上方：

除了要新增 GridViewRow 到 GridView 外, 還要將顯示明細資料的 GridView 插入新的列中, 前面有說到, GridViewRow 可視爲是一個 TableRow, 而一個新的 TableRow 是沒有儲存格 (TableCell) 的, 所以要將新的 TableCell 插入到 GridViewRow 中, 程式很簡單, 只要呼叫 GridViewRow.Cells.Add() 然後給予 TableCell 物件即可, 例如：

```
GridViewRow r = new GridViewRow(0, -1, DataControlRowType.DataRow,
  DataControlRowState.Normal);
// 插入新的 TableCell
r.Cells.Add(new TableCell());
```

然後, 將明細的 GridView 放到儲存格中, 在這裡有兩種方法, 一種是使用 User Control 來包裝, 然後用下列的程式來插入：

```
r.Cells[1].Controls.Add(Page.LoadControl("myControl.ascx");
```

但筆者在這裡要使用的是直接將在網頁中的明細 GridView 轉換成 HTML, 然後插入儲存格中, 其作業方式與將 GridView 資料下載成 Excel 檔案做法相同, 請參考 Q49：

```
// 建立 HTML 輸出工具
StringWriter sw = new StringWriter();
HtmlTextWriter hw = new HtmlTextWriter(sw);
// 設定 SqlDataSource 的參數值 (data 爲裝載參數的變數)
this.dsDetail.SelectParameters["orderID"].DefaultValue = data;
// 繫結資料
this.DetailView.DataSource =
  (this.dsDetail.Select(new DataSourceSelectArguments())
    as DataView);
this.DetailView.DataBind();
```

```
// 產生 HTML
this.DetailView.RenderControl(hw);
// 將 HTML 插入儲存格中
r.Cells[0].Text = sw.ToString();
```

準備好 GridViewRow 後, 就可以把它插入到 GridView 中, 如下列程式碼：

```
// r 是主檔 GridView 中的 GridViewRow
MasterView.Controls[0].Controls.AddAt(r.RowIndex + 1, row);
```

接著, 將前述部份的程式碼放到 GridView.RowCreated 事件常式中, 處理新增列的部份就完成了。不過, GridView.RowCreated 事件可用的 DataControlRowType 有很多種, 必須要選用一種可以涵蓋到所有 DataRow 的 DataControlRowType, 這樣才能夠對所有的資料列做處理, 這個 DataControlRowType 也必須要同時支援有無分頁的瀏覽行爲, 綜上觀點, 筆者認爲 Footer 最適合, 它是在所有 DataRow 完畢後才處理, 並且可同時支援有無分頁的狀況。

另外, 在分頁瀏覽的狀況下, 還必須考慮是否有雙向 Pager (分頁瀏覽器) 的問題, GridView 的 PageSettings.Position 屬性支援了單向與雙向的 Pager (上下都有), 此時, 列的計算就必須要再加 1, 否則會出現前述 RowIndex + 1 的問題, 也就是說, 如果開發人員設定使用雙向 Pager 的話, RowIndex 就要加 3。

至此, 新增與插入明細 GridView 的程式即宣告完成。

```
protected void MasterView_RowCreated(object sender,
  GridViewRowEventArgs e)
{
  // 只有在 DataControlRowType 爲 Footer 時才啓動。
  if (e.Row.RowType == DataControlRowType.Footer)
     this.CreateCollapseRow();
  }
```

```
private void CreateCollapseRow()
{
foreach (GridViewRow row in MasterView.Rows)
{
  //只針對被選取的主檔列做處理
  //this.fireCommandRowIndex 的功用稍後會說明
  if(row.RowIndex == this.fireCommandRowIndex)
   {
    // 產生新的 GridViewRow
    GridViewRow r = new GridViewRow(row.RowIndex, -1,
    DataControlRowType.DataRow, DataControlRowState.Normal);
    StringWriter sw = new StringWriter();
    HtmlTextWriter hw = new HtmlTextWriter(sw);

    // 插入儲存格
    r.Cells.Add(new TableCell());
    r.Cells.Add(new TableCell());
   // 設定儲存格跨越其他儲存格, 這是爲了要與主檔列對齊。
    r.Cells[1].ColumnSpan = MasterView.Columns.Count - 1;
   // 設定明細資料查詢參數。
    this.dsDetail.SelectParameters["orderID"].DefaultValue =
      row.Cells[1].Text;
   // 繫結與產生明細資料的 HTML。
    this.DetailView.Visible = true;
    this.DetailView.DataSource =
      (this.dsDetail.Select(new DataSourceSelectArguments())
        as DataView);
    this.DetailView.DataBind();
    this.DetailView.RenderControl(hw);
    this.DetailView.Visible = false;
```

```
        // 將 HTML 插入儲存格中。
        r.Cells[1].Text = sw.ToString();
        sw.Close();
        // 將 GridViewRow 插入 GridView 中
        // 若 GridView 允許分頁時, 需檢查 Pager 是否爲雙向,
        // 若是則+3, 否則爲+2
        if (MasterView.AllowPaging &&
          MasterView.PagerSettings.Position ==
          PagerPosition.TopAndBottom)
          MasterView.Controls[0].Controls.AddAt(row.RowIndex + 3, r);
        else
          MasterView.Controls[0].Controls.AddAt(row.RowIndex + 2, r);
      }
    }
  }
```

接下來就要處理比較棘手的問題, 就是捕捉來自於使用者選取的主檔列位置, 爲什麼棘手呢？因爲一般若要捕捉使用者選取的資料列, 都會使用 GridView.SelectedIndexChanging 或GridView.SelectedIndexChanged 事件常式, 但這兩種事件執行的順序卻比 GridView.RowCreated 要晚, 就算用 GridView.RowCommand 事件常式, 也會有同樣的問題, 因此需要有一個方法, 能夠在GridView.RowCreated 事件常式執行前, 取得選取的列索引。

在沒有快速可用的方法之下, 土法煉鋼就是唯一的選擇了, 尤其是在現有控制項無法支援的情況, 不土法煉鋼是做不出想要的效果的, 而這個土法, 就是 Request.Form。在每次 ASP.NET 控制項執行時, 都會以 Request.Form 將必要的資料回傳, 其中會包含引發 Post-Back 的控制項 ID, 只要在 ID 中動手腳, 就可以得到想要的資料了。

爲了要更方便的比對指令以及列位置, 筆者使用了 Dictionary 泛型集合物件 (Generic Collection) 來儲存控制項 ID 以及所對應的列位置, 並且將它放在 GridView.RowDataBound 事件常式中, 讓它來重設命令的控制項 ID：

```
protected void MasterView_RowDataBound(object sender,
  GridViewRowEventArgs e)
{
  if (e.Row.RowType == DataControlRowType.DataRow)
  {
    LinkButton cmdDetails = e.Row.FindControl("cmdDetails")
      as LinkButton;
    // 建立指令與列位置對應表。
    Dictionary<string, int> commandIndex =
      ViewState["CommandIndex"] as Dictionary<string, int>;
    // 將指令插入對應表中。
    if (!commandIndex.ContainsKey(cmdDetails.UniqueID))
      commandIndex.Add(cmdDetails.UniqueID, e.Row.RowIndex);
    // 儲存對應表。
    ViewState["CommandIndex"] = commandIndex;
  }
}
```

接著, 在 Page_Load 事件常式中, 將 Request.Form 中的資料和指令對應表比對, 如果出現相同的指令, 則輸出列位置給 GridView.RowCreated 事件常式, 筆者使用一個全域變數 fireCommandRowIndex 來記錄列索引, 以供 RowCreated 事件常式取用, 其程式如下：

```
private int fireCommandRowIndex = -1; // 儲存列索引變數。

protected void Page_Load(object sender, EventArgs e)
{
```

```
  // 第一次載入，新增指令列位置索引資料結構。
  if (!Page.IsPostBack)
    ViewState.Add("CommandIndex", new Dictionary<string, int>());
  // 取出指令列位置索引表。
  Dictionary<string, int> commandIndex =
  ViewState["CommandIndex"] as Dictionary<string, int>;
  // 比對 Request.Form 中的指令是否出現在指令列位置索引表。
  for (int i = 0; i < Request.Form.Count; i++)
  {
    if (commandIndex.ContainsKey(Request.Form[i]))
    {
      // 找到資料，將對應的列位置設定給 fireCommandRowIndex,
      // 並中止迴圈。
      fireCommandRowIndex = commandIndex[Request.Form[i]];
      break;
    }
  }
}
```

至此, 此功能即大功告成, 接著只要修訂一些細部的地方即可, 例如查詢不同客戶時, GridView.SelectedIndex 要設定爲-1。

本程式最終的完整程式碼如程式 1 與程式 2 所示。

程式1：本程式的 HTML 碼

```
<%@ Page Language="C#" EnableEventValidation="false"
  AutoEventWireup="true" CodeFile="CollapseGridView2.aspx.cs"
  Inherits="Part4_CollapseGridView2" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>可摺疊式 GridView</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    <asp:ScriptManager ID="ScriptManager" runat="server"
      EnablePartialRendering="true"></asp:ScriptManager>
    <br />
    <asp:UpdatePanel ID="updatepanelGridViews" runat="server">
      <ContentTemplate>
        客戶：
        <asp:DropDownList ID="cboCustomerList"
          DataSourceID="dsCustomers" runat="server"
          AutoPostBack="True" DataTextField="CompanyName"
          DataValueField="CustomerID">
        </asp:DropDownList>
        <table border="0" cellpadding="6" cellspacing="0"
          style="width: 100%">
          <tr>
            <td>
              <asp:GridView DataSourceID="dsMaster" ID="MasterView"
                runat="server" CellPadding="4"
                AutoGenerateColumns="false" ForeColor="#333333"
                OnRowDataBound="MasterView_RowDataBound"
                OnRowCreated="MasterView_RowCreated"
                OnRowCommand="MasterView_RowCommand">
                <FooterStyle BackColor="#990000" Font-Bold="True"
                  ForeColor="White" />
```

```
            <RowStyle BackColor="#FFFBD6" ForeColor="#333333" />
            <PagerStyle BackColor="#FFCC66" ForeColor="#333333"
              HorizontalAlign="Center" />
            <SelectedRowStyle BackColor="#FFCC66"
              Font-Bold="True" ForeColor="Navy" />
            <HeaderStyle BackColor="#990000" Font-Bold="True"
              ForeColor="White" />
            <AlternatingRowStyle BackColor="White" />
            <Columns>
              <asp:TemplateField>
                <ItemTemplate>
                  <asp:LinkButton ID="cmdDetails" Text="明細..."
                    runat="server"
                    CommandName="LookupDetail" />
                </ItemTemplate>
              </asp:TemplateField>
              <asp:BoundField DataField="OrderID"
                HeaderText="訂單編號" />
              <asp:BoundField DataField="OrderDate"
                HeaderText="訂單日期" />
              <asp:BoundField DataField="ShippedDate"
                HeaderText="寄送日期" />
              <asp:BoundField DataField="EmployeeName"
                HeaderText="員工" />
            </Columns>
            <PagerSettings Position="TopAndBottom" />
          </asp:GridView>
        </td>
      </tr>
      <tr>
        <td>
```

```
            <asp:GridView ID="DetailView" runat="server"
              Visible="false" CellPadding="4"
              AutoGenerateColumns="false" ForeColor="#333333">
              <FooterStyle BackColor="#507CD1" Font-Bold="True"
                ForeColor="White" />
              <RowStyle BackColor="#EFF3FB" />
              <PagerStyle BackColor="#2461BF" ForeColor="White"
                HorizontalAlign="Center" />
              <SelectedRowStyle BackColor="#D1DDF1"
                Font-Bold="True" ForeColor="#333333" />
              <HeaderStyle BackColor="#507CD1" Font-Bold="True"
                ForeColor="White" />
              <EditRowStyle BackColor="#2461BF" />
              <AlternatingRowStyle BackColor="White" />
              <Columns>
                <asp:BoundField DataField="OrderID"
                  Visible="False" />
                <asp:BoundField DataField="ProductName"
                  HeaderText="產品名稱" />
                <asp:BoundField DataField="UnitPrice"
                  HeaderText="單價" />
                <asp:BoundField DataField="Quantity"
                  HeaderText="數量" />
                <asp:BoundField DataField="Discount"
                  HeaderText="折扣" />
              </Columns>
            </asp:GridView>
          </td>
        </tr>
      </table>
    </ContentTemplate>
```

```
      <Triggers>
        <asp:AsyncPostBackTrigger ControlID="cboCustomerList" />
        <asp:AsyncPostBackTrigger ControlID="MasterView" />
      </Triggers>
    </asp:UpdatePanel>

    <asp:SqlDataSource ID="dsCustomers" runat="server"
      ConnectionString=
        "<%$ ConnectionStrings:northwindConnectionString %>"
      SelectCommand="SELECT CustomerID, CompanyName FROM Customers"
      SelectCommandType="Text">
    </asp:SqlDataSource>
    <asp:SqlDataSource ID="dsMaster" runat="server"
      ConnectionString=
        "<%$ ConnectionStrings:northwindConnectionString %>"
     SelectCommand="SELECT e.FirstName + ' ' + e.LastName AS EmployeeName,
OrderID, OrderDate, ShippedDate FROM Orders o INNER JOIN Employees e ON
o.EmployeeID = e.EmployeeID WHERE o.CustomerID = @customerID"
SelectCommandType="text">
      <SelectParameters>
        <asp:ControlParameter ControlID="cboCustomerList"
          Name="customerID" />
      </SelectParameters>
    </asp:SqlDataSource>
    <asp:SqlDataSource ID="dsDetail" runat="server"
      ConnectionString=
        "<%$ ConnectionStrings:northwindConnectionString %>"
     SelectCommand="SELECT p.ProductName, p.UnitPrice, od.Quantity, od.
Discount FROM [Order Details] od INNER JOIN Products p ON od.ProductID =
p.ProductID WHERE od.OrderID = @orderID" SelectCommandType="text">
```

```
      <SelectParameters>
        <asp:Parameter Name="orderID" />
      </SelectParameters>
    </asp:SqlDataSource>
  </div>
  </form>
</body>
</html>
```

程式2：完整程式碼實作 (大部份程式在前面皆已說明)

```
using System;
using System.Data;
using System.Configuration;
using System.Collections.Generic;
using System.Collections;
using System.IO;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class Part4_CollapseGridView2 : System.Web.UI.Page
{
  private int fireCommandRowIndex = -1;

  protected void Page_Load(object sender, EventArgs e)
  {
```

```
  if (!Page.IsPostBack)
    ViewState.Add("CommandIndex", new Dictionary<string, int>());

  Dictionary<string, int> commandIndex =
    ViewState["CommandIndex"] as Dictionary<string, int>;

  for (int i = 0; i < Request.Form.Count; i++)
  {
    if (commandIndex.ContainsKey(Request.Form[i]))
    {
      fireCommandRowIndex = commandIndex[Request.Form[i]];
      break;
    }
  }
}

public override void VerifyRenderingInServerForm(Control control)
{
  // base.VerifyRenderingInServerForm(control);
}

protected void MasterView_RowDataBound(object sender,
  GridViewRowEventArgs e)
{
  if (e.Row.RowType == DataControlRowType.DataRow)
  {
    LinkButton cmdDetails = e.Row.FindControl("cmdDetails")
      as LinkButton;
    Dictionary<string, int> commandIndex =
    ViewState["CommandIndex"] as Dictionary<string, int>;
```

```
    if (!commandIndex.ContainsKey(cmdDetails.UniqueID))
      commandIndex.Add(cmdDetails.UniqueID, e.Row.RowIndex);

    ViewState["CommandIndex"] = commandIndex;
  }
}

protected void MasterView_RowCreated(object sender,
  GridViewRowEventArgs e)
{
  System.Diagnostics.Debug.Print("RowCreated event");
  if (e.Row.RowType == DataControlRowType.Footer)
    this.CreateCollapseRow();
}

private void CreateCollapseRow()
{
  foreach (GridViewRow row in MasterView.Rows)
  {
    if (row.RowIndex == this.fireCommandRowIndex)
    {
      MasterView.SelectedIndex = this.fireCommandRowIndex;

      GridViewRow r = new GridViewRow(row.RowIndex, -1,
        DataControlRowType.DataRow, DataControlRowState.Normal);
      StringWriter sw = new StringWriter();
      HtmlTextWriter hw = new HtmlTextWriter(sw);

      r.Cells.Add(new TableCell());
      r.Cells.Add(new TableCell());
```

```
        r.Cells[1].ColumnSpan = MasterView.Columns.Count - 1;

        this.dsDetail.SelectParameters["orderID"].DefaultValue =
          row.Cells[1].Text;

        // 在輸出 HTML 前, 必須先設爲 true, 否則會無法輸出 HTML
        this.DetailView.Visible = true;
        this.DetailView.DataSource =
          (this.dsDetail.Select(new DataSourceSelectArguments())
            as DataView);
        this.DetailView.DataBind();
        this.DetailView.RenderControl(hw);
        this.DetailView.Visible = false;

        r.Cells[1].Text = sw.ToString();
        sw.Close();

        if (MasterView.AllowPaging &&
          MasterView.PagerSettings.Position ==
            PagerPosition.TopAndBottom)
          MasterView.Controls[0].Controls.AddAt(row.RowIndex + 3, r);
        else
          MasterView.Controls[0].Controls.AddAt(row.RowIndex + 2, r);
      }
    }
}
```

```
  protected void MasterView_RowCommand(object sender,
    GridViewCommandEventArgs e)
  {
    this.MasterView.DataBind();
  }
}
```

延伸閱讀

有關於 Collapsible-GridView (本文所說明的 GridView 類型), 在台灣微軟網站上有一篇文章「ASP.NET 2.0 GridView 範例集」, 由黃忠成老師執筆, 文中也有提到這種 GridView 的設計方法, 只不過他用的是以元件角度來設計的, 採用許多 ViewState 狀態管理的程式, 有興趣的讀者可到台灣微軟 MSDN 官方網站搜尋。

考生停看聽

本問題所討論的內容, 可用來準備 Exam 70-528: TS: Microsoft .NET Framework 2.0 Web Client Development 的下列主題。

- 使用 ADO.NET 將資料整合到 Web 應用程式中。
  - 實作 Data-Bound (資料繫結) 控制項
    - 使用表格化資料來源控制項回傳表格資料。
    - 使用簡單資料繫結控制項來顯示資料。
    - 使用複合資料繫結控制項來顯示資料。

Q51 如何在資料庫新增具有自動編號欄位的資料後，得到系統給它的編號？

適用範圍：VASP.NET 1.0 VASP.NET 1.1 VASP.NET 2.0 VASP.NET 3.5

問題

我在 SQL Server 中設計了一個銷售記錄的資料表, 使用自動編號當做識別碼, 目前可以正常運作, 但我想要在主檔記錄新增後, 抓取系統自動給予的識別碼做後續的處理, 請問我要怎麼做？

問題說明

這種應用在企業應用程式中很常見, 像是流水單號或是依特定格式的編號 (例如 A02930-2007-11-29-009) , 開發人員最可能使用的就是識別碼欄位 (Identity Column) , 用識別欄位的最大好處就是由資料庫來產生編號, 開發人員可以不必自己管理編號。

不過方便的背後, 總會隱藏一些不好處理的地方, 像是如果要馬上知道新的識別碼的話, 如果不是由開發人員自行管理, 就是要由資料庫來抓, 但是若是在多人使用的系統下, 要如何知道哪一筆是最新的？如果抓不到或是抓錯的話, 系統的處理就會發生失誤 (例如甲單的流水號被乙單抓到, 讓處理的程式誤以爲抓到的流水號是乙單, 因而發生錯誤的現象) 。

SQL Server 中有三個可以取出最新的識別值的函數, 分別爲全域變數 @@IDENTITY, 函數 IDENT_CURRENT 以及函數 SCOPE_IDENTITY, 這三個函數的使用會根據影響程度以及範圍的不同, 所回傳的最新識別值也會有所不同, 其差別如表。

函數	使用方法	受影響類型	適用範圍
@@IDENTITY	SELECT @@ IDENTITY	同一工作階段，但不同範圍的新增指令，巢狀指令或是資料庫複製 (Replication) 等。	要取得工作階段層次的最新識別碼值，適用於單一工作階段環境，但要注意此值會受到同一工作階段中不同的新增指令影響。
IDENT_ CURRENT	SELECT IDENT_ CURRENT("table")	不同工作階段，且不同範圍的新增指令。	要取得全域層次的最新識別碼值，適用於多工作階段環境。
SCOPE_ IDENTITY	SELECT SCOPE_ IDENTITY()	同一工作階段，且同一範圍的新增指令。	要取得工作階段層次的最新識別碼值，適用於單一工作階段環境。

- 範圍是指在同一個工作階段 (每一個連線都算是一個工作階段) 中的指令群組 (如預存程序、函數、觸發程序或是以 BEGIN...END 所包裝的指令集) , 有可能同一個範圍中執行了多個新增的陳述式 (例如在同一個預存程序中, 插入許多的訊息資料, 例如 Email 記錄或是使用者活動狀態等) , 雖然大多數情況下都只有一個。
- 在較繁忙的環境中, 使用 IDENT_CURRENT 可以取得不限工作階段的最新識別值, 而 SCORP_IDENTITY 可以將範圍鎖定在同一範圍 (不受其他範圍影響) 下的最新識別碼值, 常用於具有巢狀指令的工作上。

不管是用哪一種變數, 都要在執行新增的指令跑過後, 立即下取出變數的指令, 如此才可以確保取出的識別碼值是最新 (最後新增的那一個) 的值。

```
SqlConnection conn = new SqlConnection("...");
SqlCommand cmd = new SqlCommand(
"INSERT INTO myTable (Key, Value) VALUES (@key, @Value); SELECT @@IDENTITY",
conn);

... // 套用 @key 和@Value 的參數值。

conn.Open();
SqlDataReader reader = cmd.ExecuteReader(
  CommandBehavior.CloseConnection);

reader.Read(); // 執行 INSERT
reader.NextResult(); // 執行下一個指令

int keyID = Convert.ToInt32(reader.GetValue(reader.GetOrdinal(0)));
reader.Close();

SqlConnection conn = new SqlConnection("...");
SqlCommand cmd = new SqlCommand(
"INSERT INTO myTable (Key, Value) VALUES (@key, @Value)", conn);

... // 套用 @key 和@Value 的參數值。

conn.Open();
cmd.ExecuteNonQuery();
cmd.Parameters.Clear();

cmd.CommandText = "SELECT @@IDENTITY";
int keyID = Convert.ToInt32(cmd.ExecuteScalar());

conn.Close();
```

解決方案

若是 SQL Server, 可以在執行完新增的指令後, 使用 SQL Server 的 @@IDENTITY、SCOPE_IDENTITY 或是 IDENT_CURRENT 函數來取得最新的識別碼值, 若是其他的資料庫, 則要參考資料庫的文件, 大多數的資料庫對於識別碼的處理都有一套方法。

小常識 用識別碼好, 還是自己管理比較好？

識別碼管理 (Identity Management) 有時候也是門學問, 而且往往是在便利性和嚴謹性之間拔河, 又想要便利, 而且要要求嚴謹的話, 大多數是無法兩全的, 如果硬要兩全的話, 可能會付出很大的代價。再者, 有些大型的應用系統 (ERP/SCM 這類型的) 對於流水號的管理 (Flow-Number Management) 是較集中, 或者是更加嚴謹的, 例如每個流水號代表每張單據, 而且不能夠跳號 (稽核需求) , 這樣的系統若要用識別碼式的流水號管理, 可能很容易發生問題, 因此就比較傾向於自己實作管理的機能, 如此的話彈性就會很大, 而且還可以自己控制流水號產生的過程, 不過缺點就是要自己寫比較多的程式碼, 但兩相權衡之下, 自己管理流水號反而會比較好, 這也是筆者自己的作法。

但不是說識別碼不好, 資料庫管理的識別碼適合用在較不受限的系統中, 如果受限的話, 硬要用識別碼有時反而會造成反效果。

Q52 如何以自訂編號格式處理編號遞增？

適用範圍：VASP.NET 2.0 VASP.NET 3.5

問題

我用 ASP.NET 發展了一套 Web 上的訂單管理系統, 用來接受訂單, 原本我使用的是自動編號, 但老闆有意見, 要我改成像 [日期]-[店號]-[流水號] 這樣的格式, 而且流水號固定要用七碼, 但這樣要做遞增時變成無法沿用 SQL Server 本身的自動編號, 請問我要如何做？

問題說明

自訂編號格式 (Custom Identifier Format) 在許多的商用應用程式中, 可是相當重要的需求, 尤其是在導入 ISO 規範的情況下, 編號格式更形重要了, 像是收據編號、出貨編號或是表單編號等各式各樣的編號, 充斥在系統的各個角落中。通常編號的格式, 都是由業管的人員來決定的, 或者是要由部門的主管經由討論來決定, 決定後就不可再更改 (若要更改可能要花更多的工來轉換格式) 。

另外, 編號的流水號要放多少也是要衡量的, 如果是資料量大的應用系統 (如 POS 或訂單管理系統) , 可考慮使用較長 (較多位數) 的流水編號來實作, 反之, 可用較短的編號來實作。另一種可行的方法, 則是靠檢視表或預存程序來動態產生編號格式, 如此雖然可以將格式化的工作降低, 但是卻有著查詢時都要解譯編號的問題, 而且若是高度複雜的編號, 解譯的時間就有可能拉長, 再加上 T-SQL 本身對字串的拆解不比 VB 或 C#來的強, 故這個方法可能不太適用, 但仍然要看系統的規劃而定。

要實作一個自訂格式編號的功能, 首先要先決定格式, 之後就要實作一個可建構且可解譯編號的程式碼, 並且要實作遞增的邏輯, 如此才能在資料輸入時能夠順利的建構出順序的編號出來。

自訂編號的功能可能也會隨著不同的系統, 而有著不同的需求, 例如編號的順序性, 以及是否允許跳號或保留編號等功能, 製造業 ERP 系統就有這種類似的多變需求, 此時就要想辦法去修正以及預留編號的空間, 防止像跳號或空號等異常狀況 (若能容許則可免) 。

筆者以一個簡單的自訂格式編號, 來說明要如何建構自訂格式編號的處理程式。

解決方案

首先, 決定好編號格式, 筆者以[日期][客戶碼][流水號] (流水號爲七碼) 爲格式, 來示範要如何實作編碼與解譯的程式, 以及要如何取得最後的編號。

接著, 撰寫處理的程式碼, 自訂編號的解析工作主要使用 System.String.SubString() 來切割字串, 並且用 System.String.Format() 來格式化字串, 以建立出指定的格式, 配合 SQL 指令來產生新的編號字串。

```
protected void cmdCreate_Click(object sender, EventArgs e)
{
  SqlConnection conn = new SqlConnection(
    "initial catalog=TestDB; user id=xx; password=xx");
  SqlCommand cmd = new SqlCommand(
    "SELECT ISNULL(MAX(OrderNumber), 0) FROM OrderList", conn);
  int num = 0;

  conn.Open();
  // 取回最後的自訂編號, 如果沒有資料,
  // 則傳回 0 (由 ISNULL(MAX(OrderNumber), 0) 決定)
  // 如果有最後的自訂編號, 則要解析出流水號,
  // 並轉換成數值型別, 以利遞增。
  if (cmd.ExecuteScalar().ToString() != "0")
    num = Convert.ToInt32(
```

```
        cmd.ExecuteScalar().ToString().Substring(11));
    // 遞增編號。
    num++;

    conn.Close();
    // 產生新的自訂編號。
    string number = DateTime.Now.ToString("yyyyMMdd") +
    this.cboCustomerList.SelectedValue + num.ToString("0000000");

    cmd.CommandText = "INSERT INTO OrderList VALUES (NEWID(), '" +
      number + "', " + this.T_TotalAmount.Text + ")";
    conn.Open();
    cmd.ExecuteNonQuery();
    conn.Close();
    cmd.Dispose();
    conn.Dispose();

    this.GridView1.DataSource = this.LoadData();
    this.GridView1.DataBind();
}
```

當然, 實際實作的編號管理可能不會這麼簡單 (當然也有可能會就這麼簡單 ☺), 所以要視需求以及編號的格式來做變動。

小常識　自訂編號的管理

在筆者開發自訂格式編號的經驗中, 以日期加上櫃號加上流水號的案例最多, 當然也有一些特別的編號格式, 不過這還不是最重要的, 重要的是編號的管理 (多人環境下更需要管理), 像是不允許空號或跳號的要求, 會額外掛上一個表格來管理, 以避免跳號的問題。管理編號不像是產生編號那樣簡單, 有時候企業的需求會比較高一些, 要如何能夠實作出企業需要的自訂編號管理機制, 才是開發人員應該重視的。