Blog

Entries

Summary

Listed by:

Date

October 2009

September 2009

August 2009

July 2009

June 2009

May 2009

April 2009

March 2009

February 2009

January 2009

December 2008

November 2008

October 2008

September 2008

August 2008

July 2008

June 2008

May 2008

April 2008

< Previous

October 23

JSON 程式設計 – DataSnap的回叫機制

DataSnap 2010加入了回叫機制，當伺服端方法在執行的過程中可以回叫用戶端提供的方法以通知用戶端有關伺服端方法執行的狀態。

DataSnap的回叫機制非常適合使用在需要較長時間執行的伺服端方法，例如如果伺服端方法需要執行長時間的查詢時就很適合使用，或是當程式啟動或是執行時需要進行許多查詢的工作，那麼也都可以使用回叫機制。

要使用DataSnap 2010的回叫機制非常的簡單，開發人員只需要實作一個從TDBXCallback繼承下來的實體類別，並且在呼叫伺服端方法時把此實體類別的樣例當做參數傳遞給伺服端方法即可。

由於使用DataSnap 2010的回叫機制並不困難，因此讓我們使用一個範例來說明讀者就可以很快的瞭解。

在下列的範例中本文將使用Delphi 2010中新的IOUtils程式單元中的類別進行檔案搜尋和計數的工作，由於這將花上一些時間，因此我們正好使用它來展示使用同步和回叫機制的差異。

範例**DataSnap**伺服器

首先讓我們建立一個DataSnap伺服端，下面是這個伺服器輸出的伺服端方法，請注意的是，TServerMethods1輸出了兩個方法GetServerDirectoryInfo和GetServerDirectoryInfoAsync。這兩個方法都執行相同的工作，它們使用TDirectory類別搜尋和計數特定伺服端目錄下的檔案總數，它們的差異是GetServerDirectoryInfo使用同步的方式執行，因此當用戶端呼叫它時，用戶端會同時暫停反應直到GetServerDirectoryInfo執行完畢。

而GetServerDirectoryInfoAsync則是使用回叫機制的方式執行，因此當用戶端呼叫它之後，GetServerDirectoryInfoAsync在執行的過程中則可以藉由用戶端傳遞來的參數**ACallback: TDBXCallback**，來回叫回用戶端，告訴用戶端執行的狀態，用戶端因此也根據目前伺服端執行的情形來更新用戶端的資訊。

```
{$METHODINFO ON}

TServerMethods1 = class(TPersistent)

private

{ Private declarations }

        fTotalFiles : Integer;

        FResult : TJSONArray;
```

March 2008
February 2008
January 2008
December 2007
November 2007
October 2007
September 2007
August 2007
July 2007
June 2007
May 2007
April 2007
March 2007
February 2007
January 2007
December 2006
November 2006
October 2006
September 2006
August 2006
July 2006
June 2006
May 2006
April 2006
March 2006
February 2006
January 2006

```
        FCallback: TDBXCallback;
        procedure ProcessPath(const sPath : string);
        procedure ProcessPathAsync(const sPath : string);
        procedure ShowMessage(sMessage : string);
        procedure ProcessThisDirectory(const sPath : string);
public
{ Public declarations }
        function EchoString(Value: string): string;
        function GetServerDirectoryInfo(const sPath : string) : TJSONArray;
        function GetServerDirectoryInfoAsync(ACallback: TDBXCallback; const sPath : string): TJSONArray;
end;
{$METHODINFO OFF}
```

GetServerDirectoryInfoAsync是如何回叫回用戶端呢?其實非常的簡單，因為用戶端在呼叫它時已經把用戶端的回叫方法當成參數傳遞過來了，因此GetServerDirectoryInfoAsync只需要藉由這個參數即可回叫回用戶端。

因此我們可以從下面18行的程式碼看到，伺服端直接使用這個回叫參數呼叫用戶端，並且建立一個TJSONString型態的物件做為參數，在這個TJSONString物件中告訴了用戶端目前伺服端正在處理那一個目錄。

```
001 function TServerMethods1.GetServerDirectoryInfoAsync(ACallback: TDBXCallback;
002 const sPath: string): TJSONArray;
003 begin
004     FCallback := ACallBack;
005     FTotalFiles := 0;
006     FResult := TJSONArray.Create;
007     ProcessPathAsync(sPath);
008     FResult.AddElement(TJSONString.Create('總檔案數' + ' : ' + IntToStr(FTotalFiles)));
009     Result := FResult;
```

December 2005
November 2005
October 2005
September 2005
August 2005
July 2005

```
010 end;
011
012 procedure TServerMethods1.ProcessPathAsync(const sPath: string);
013 var
014     rootDirectories : TStringDynArray;
015     i: Integer;
016 begin
017     ProcessThisDirectory(sPath);
018     FCallback.Execute(TJSONString.Create('處理目錄' + sPath + '中...'));
019     rootDirectories := TDirectory.GetDirectories(sPath);
020     for i := 0 to Length(rootDirectories) - 1 do
021     ProcessPathAsync(rootDirectories[i]);
022 end;
```

同步用戶端

範例的同步用戶端非常的簡單，它只是直接呼叫伺服端的GetServerDirectoryInfo方法。

```
procedure TForm3.btnGetServerInfoClick(Sender: TObject);
var
   aServer : TServerMethods1Client;
   ja : TJSONArray;
   jv : TJSONValue;
   I: Integer;
begin
   lStart := GetTickCount;
   aServer := TServerMethods1Client.Create(Self.SQLConnection1.DBXConnection);
```

```
    try
      ja := aServer.GetServerDirectoryInfo(Edit1.Text);
      lEnd := GetTickCount;
      for I := 0 to ja.Size - 1 do
      begin
        jv := ja.Get(I);
        lbResult.Items.Add(jv.ToString);
      end;
    finally
      aServer.Free;
      ShowRunTime(lEnd - lStart);
    end;
end;
```

在用戶端呼叫GetServerDirectoryInfo的過程中，用戶端暫停反應，使用者也無從瞭解伺服端執行的狀態。

回叫用戶端

再看看回叫用戶端，這個用戶端的關鍵從下面的012行開始，012行建立了TDSCallbackWithMethod物件，並且建立一個匿名方法做為用戶端的回叫方法傳遞給伺服端。從013行開始的匿名方法在被用戶端回叫的時候首先在013行把伺服端傳遞來的參數型態轉換為TJSONString的型態，接著更新用戶端的UI以通知使用者伺服端目前正在處理那一個目錄。最後在023行用戶端回叫方法如果執行成功就需要回傳TJSonTrue物件，如果失敗的話就需要回傳TJSonFalse物件。

```
001 procedure TForm3.btnGetServerInfoAsyncClick(Sender: TObject);
002 var
003     aServer : TServerMethods1Client;
004     LCallback : TDSCallbackWithMethod;
005     ja : TJSONArray;
006     jv : TJSONValue;
```

```
007     I: Integer;
008 begin
009     lStart := GetTickCount;
010     aServer := TServerMethods1Client.Create(Self.SQLConnection1.DBXConnection);
011     try
012         LCallback := TDSCallbackWithMethod.Create(
013             function(const Args: TJSONValue): TJSONValue
014             var
015                 asyncResult: TJSONString;
016                 I: Integer;
017                 LMessage: string;
018             begin
019                 asyncResult := TJSONString(Args);
020                 lbAsync.Items.Add(asyncResult.ToString);
021                 lbAsync.Update;
022                 Application.ProcessMessages;
023                 Result := TJSonTrue.Create;
024              end
025 );
026         ja := aServer.GetServerDirectoryInfoAsync(LCallback, Edit1.Text);
027
028
029         lEnd := GetTickCount;
```

```
030         for I := 0 to ja.Size - 1 do
031         begin
032             jv := ja.Get(I);
033             lbResult.Items.Add(jv.ToString);
034         end;
035     finally
036         aServer.Free;
037         ShowRunTime(lEnd - lStart);
038     end;
039 end;
```

那麼什麼是TDSCallbackWithMethod類別呢? 這要從TDBXCallback抽象類別開始談起。

TDBXCallback抽象類別

在討論TDSCallbackMethod之前我們必須先說明TDBXCallback抽象類別，因為TDSCallbackMethod是從TDBXCallback繼承下來的實體類別。事實上TDBXCallback類別即是使用DataSnap 2010回叫機制的關鍵，要使用回叫機制，開發人員必須實作一個從TDBXCallback繼承下來的實體類別，並且傳遞此實體類別的樣例給伺服端方法做為參數，如此一來伺服端方法就可以藉由這個樣例參數呼叫回用戶端，以通知用戶端伺服端方法執行的狀態。

TDBXCallback的虛擬方法Execute是衍生類別需要複載實作的，Execute接受一個型態為TJSONValue的參數並且回傳一個型態為TJSONValue的結果值，伺服端方法在回叫用戶端的方法時，可以把需要傳遞給用戶端的數值或是物件轉換為TJSONValue型態並且當成Execute方法的參數傳遞回用戶端，而用戶端的方法在被回叫執行完畢之後，也可以把執行結果轉換為TJSONValue型態並且回傳給伺服端。下面即是TDBXCallback抽象類別的宣告:

```
001 TDBXCallback = class abstract
002 public
003 function Execute(const Arg: TJSONValue): TJSONValue; virtual; abstract;
004 protected
005 procedure SetConnectionHandler(const ConnectionHandler: TObject); virtual;
006 procedure SetOrdinal(const Ordinal: Integer); virtual;
```

```
007 public

008 property ConnectionHandler: TObject write SetConnectionHandler;

009 property Ordinal: Integer write SetOrdinal;

010 end;
```

TDSCallbackMethod實體類別

瞭解了TDBXCallback扮演的角色之後解釋TDSCallbackMethod實體類別就簡單了，由於此範例應用程式要使用非同步呼叫機制，因此宣告TDSCallbackMethod從TDBXCallback繼承下來並且實作虛擬方法Execute。在TDSCallbackMethod的Execute方法中它只是簡單的呼叫在建構函式中儲存下來的用戶端方法的方法指標。

```
unit AsyncUtils;

interface

uses

Classes,

DbxDatasnap,

DBXJson;

type

TDSCallbackMethod = reference to function(const Args: TJSONValue): TJSONValue;

TDSCallbackWithMethod = class(TDBXCallback)

private

    FCallbackMethod: TDSCallbackMethod;

public

    constructor Create(ACallbackMethod: TDSCallbackMethod);

    function Execute(const Args: TJSONValue): TJSONValue; override;

end;

implementation

constructor TDSCallbackWithMethod.Create(ACallbackMethod: TDSCallbackMethod);
```

```
begin
    FCallbackMethod := ACallbackMethod;
end;
function TDSCallbackWithMethod.Execute(const Args: TJSONValue): TJSONValue;
var
aString: string;
begin
    Assert(Assigned(FCallbackMethod));
    Result := FCallbackMethod(Args);
end;
end.
```

下面的圖形分別是DataSnap伺服端執行的畫面以及同步用戶端，回叫用戶端的執行畫面，從圖2同步用戶端畫面可以看到它雖然比回叫用戶端執行的快(在筆者的機器中執行了7.078秒)，但是在同步用戶端呼叫DataSnap伺服器時它的整個UI是暫停的，因此它無法在表單下方的ListBox中顯示任何伺服端執行的資訊，使用者必須等待它完全執行完畢才能取回用戶端應用程式的控制權。

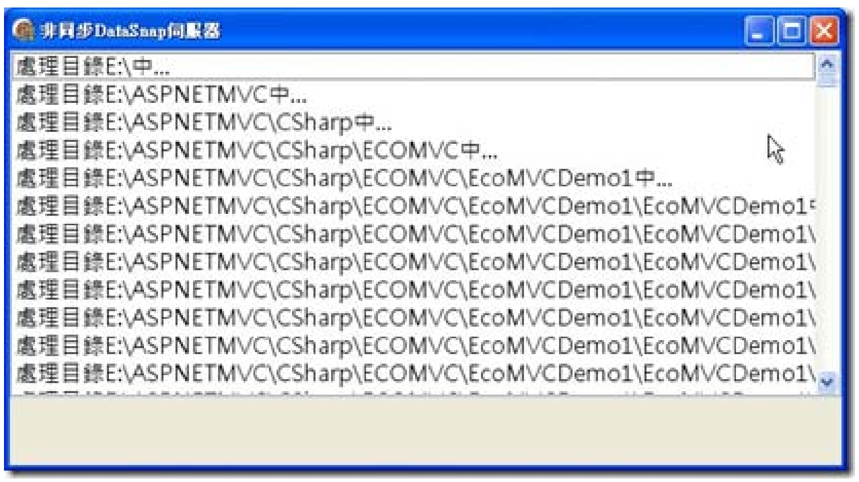

圖1 DataSnaps伺服器的執行畫面

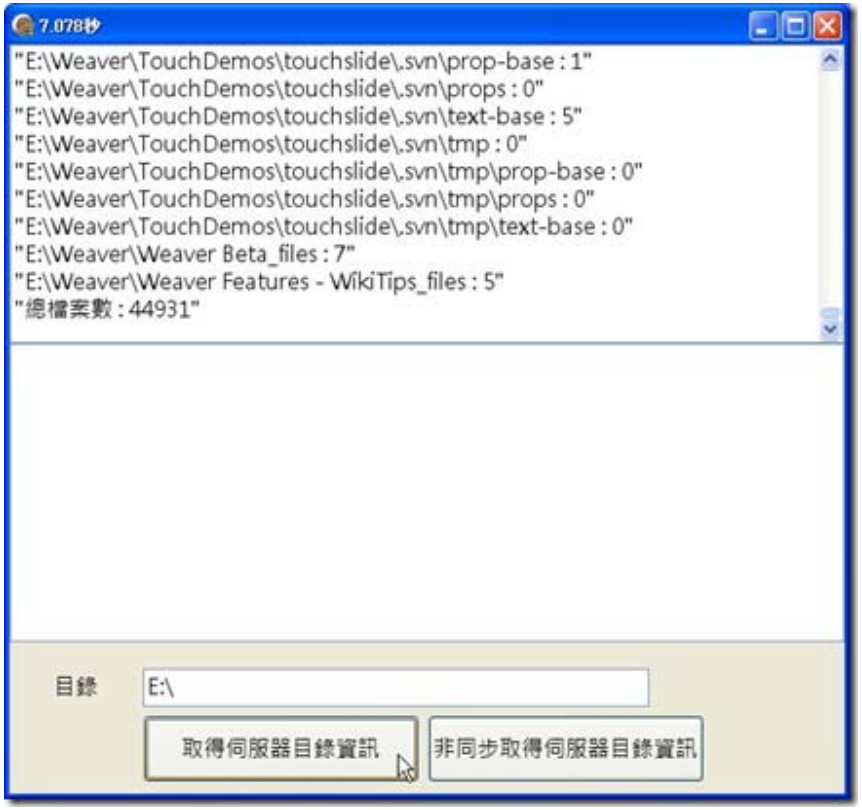

圖2 同步客戶端的執行畫面

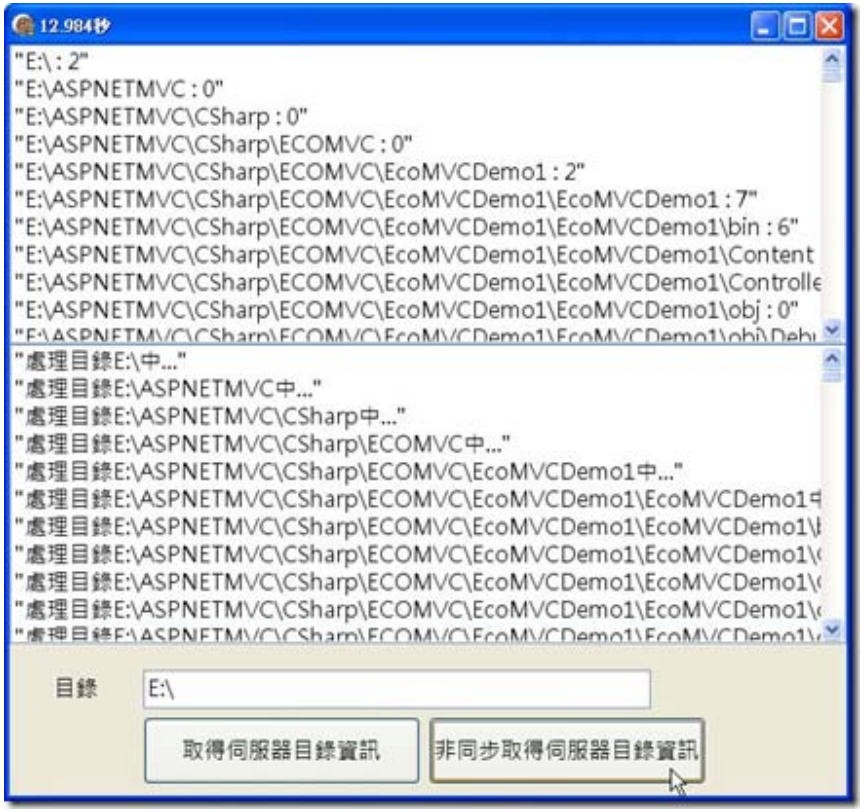

圖3 回客戶端的執行畫面

相反的畫面3則是回叫用戶端的執行結果，我們看到它是比同步用戶端慢(在筆者的機器中執行了12.984秒)，但是在整個執行過程中使用者仍然可以控制用戶端應用程式，而且回叫用戶端能夠不停的在表單下方的ListBox中顯示目前伺服端正在處理的目錄資訊。因此就使用者經驗來說，回叫用戶端是比同步用戶端好多了。

也許可以更好

DataSnap 2010的回叫機制雖然使用上非常的簡單，但在許多的應用中卻仍然可能不便，例如現在的機制需要我們在把用戶端回叫方法傳遞給伺服端方法當成參數，但這實在有些囉嗦，因為如果用戶端有許多的回叫方法，那麼在每次呼叫伺服端方法時都需要傳遞一次。因此如果DataSnap 2010能夠提供一個全域的回叫方法註冊機制，讓用戶端只需要為每一個回叫方法註冊一次即可，而無需每次呼叫即傳遞一次，那麼在使用上將更為簡化。不過DataSnap 2010的確是一個大幅進步的版本，相信未來Delphi/BCB團隊會繼續的增強DataSnap的功能。

Have Fun!

4:52 PM | Blog it

Comments

To add a comment, sign in with your Windows Live ID (if you use Hotmail, Messenger, or Xbox LIVE, you have a Windows Live ID).

Sign in

Don't have a Windows Live ID? Sign up

Trackbacks

The trackback URL for this entry is:
http://gordonliwei.spaces.live.com/blog/cns!CCE1F10BD8108687!4041.trak
Weblogs that reference this entry

- None