



## Blog

[Entries](#) [Summary](#)

Listed by:

[Date](#)[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](#)

September 16

## JSON程式設計

上篇文章簡單的介紹了如何使用DataSnap開發基於JSON分散式應用系統，Delphi 藉由原本的Midas/DataSnap和dbExpress的元件提供這個新的分散式計算能力，但使用這些元件都是資料的方式來存取遠端服務，然而我們也可以使用前面介紹的JSON類別再結合DataSnap來實作出存取遠端數值物件(Value Object)或是所謂的DTO(Data Transfer Object)的應用，讓用戶端可以存取遠端的物件。

在下面的範例中我們將展示如何開發使用VO/DTO的DataSnap應用程式伺服器，並且在用戶端擷取其中封裝的資料和物件。這個範例將開發一個可以讓用戶端查詢資料和物件的DataSnap應用程式伺服器，當用戶端查詢物件時，我們將使用TJSONObject封裝物件並且傳遞到用戶端。

## X-3-1 開發數值物件伺服器

首先建立一個VCL Form應用程式專案，在它的主表單中加入如下的元件：

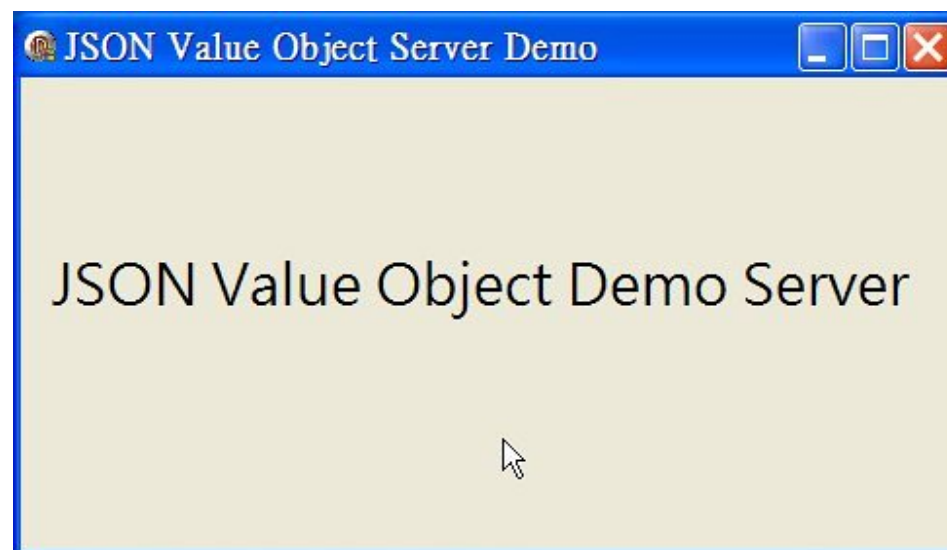


圖7 VO DataSnap應用程式伺服器

接著在專案中建立一個新的程式單元，於其中實作一個簡單的TEmployee類別如下：

```
unit uEmployee;

interface

uses SysUtils, classes;

type
```

March 2008	TEmployee = class
February 2008	private
January 2008	{ Private declarations }
December 2007	FName : string;
November 2007	FEmail : string;
October 2007	FPhone : string;
September 2007	public
August 2007	{ Public declarations }
July 2007	constructor Create(sName : string = ""; sEmail : string = ""; sPhone : string = "");
June 2007	
May 2007	property Name : string read FName write FName;
April 2007	property EMail : string read FEmail write FEmail;
March 2007	property Phone : string read FPhone write FPhone;
February 2007	end;
January 2007	
December 2006	implementation
November 2006	{ TEmployee }
October 2006	constructor TEmployee.Create(sName, sEmail: string; sPhone: string);
September 2006	begin
August 2006	FName := sName;
July 2006	FEmail := sEmail;
June 2006	FPhone := sPhone;
May 2006	end;
April 2006	
March 2006	end.
February 2006	再於專案中建立一個新的程式單元稱為uEmployeeValueObject，其中定義TEmployeeVO類別如下，請注意TEmployeeVO類別使用了{\$MethodInfo ON}和{\$MethodInfo Off}編譯器指令要求編譯器把TEmployeeVO類別的Reflection資訊編譯到執行檔中。
	{ \$MethodInfo ON }
	TEmployeeVO = class(TComponent)
	private
	{ Private declarations }
	FEmployees : TObjectList<TEmployee>;
	function CreateEmployeeJSONObject(employee : TEmployee) : string;
	function CreateEmployeeJSONArray : string;
	function CreateEmployeeJSONObjectJ(employee : TEmployee) : TJSONObject;
	public
	{ Public declarations }
	destructor Destroy; override;
	function GetEmployeeJ(const sName : string) : TJSONObject;
	function GetAllEmployeesJ : TJJSONArray;

January 2006

December 2005

November 2005

October 2005

September 2005

August 2005

July 2005

```
procedure AddEmployee(const sName : string; const sEMail : string; const sPhone : string);
end;
{$MethodInfo Off}
```

TEmployeeVO宣告了三個方法說明如下：

方法名稱	說明	備註
AddEmployee	讓用戶端呼叫增加員工資訊	
GetEmployeeJ	讓用戶端以員工姓名查詢員工物件	請注意GetEmployeeJ回傳TJSONObject型態的結果值，這代表範例應用程式將把員工物件封裝在TJSONObject物件之中並且回傳給用戶端
GetAllEmployeesJ	讓用戶端查詢所有的員工資訊	查詢後端所有員工資訊，請注意GetAllEmployeesJ回傳型態為TJSONArray的結果值，這代表範例應用程式將把所有查詢結果封裝在TJSONArray物件中回傳

下面小節將簡單的說明這些方法的實作。

### AddEmployee方法的實作

AddEmployee方法非常的簡單，它從用戶端接受三個字串型態的參數，並且根據它們來建立TEmployee物件，最後再把建立的TEmployee物件加入在宣告為TObjectList<TEmployee> 泛型型態的變數FEmployees之中。

```
procedure TEmployeeVO.AddEmployee(const sName, sEMail, sPhone: string);
var
  employee : TEmployee;
begin
  if (FEmployees = nil) then
    FEmployees := TObjectList<TEmployee>.create(true);
  employee := TEmployee.Create(sName, sEmail, sPhone);
  FEmployees.Add(employee);
end;
```

### GetEmployeeJ方法的實作

GetEmployeeJ方法就非常的有趣了，它展示了如何於DataSnap應用程式伺服器中使用TJSONValue的相關類別，GetEmployeeJ回傳型態為TJSONObject的物件回用戶端。

GetEmployeeJ根據用戶端傳遞來查詢的員工姓名，在Fem ployees中一一的搜尋具有相同名稱的TEmployee物件，找到之涕在013行呼叫CreateEmployeeJSONObjectJ來建立回傳的TJSONObject物件。

```
001 function TEmployeeVO.GetEmployeeJ(const sName: string): TJSONObject;  
002 var  
003     ie : TList<uEmployee.TEmployee>.TEnumerator;  
004     employee : TEmployee;  
005 begin  
006     Result := nil;  
007     ie := FEmployees.GetEnumerator;  
008     while (ie.MoveNext) do  
009     begin  
010         employee := ie.Current;  
011         if (employee.Name = sName) then  
012         begin  
013             Result := CreateEmployeeJSONObjectJ(employee);  
014             break;  
015         end;  
016     end;  
017 end;
```

CreateEmployeeJSONObjectJ方法根據找到的TEmployee物件來建立TJSONObject物件，它呼叫我們前面學習過的AddPair方法，把每一個TEmployee物件的特性名稱和特性值做為一個JSON的中『名稱/值』配對加入到TJSONObject物件。

```
function TEmployeeVO.CreateEmployeeJSONObjectJ(employee: TEmployee): TJSONObject;  
var  
    aJO : TJSONObject;  
begin  
    aJO := TJSONObject.Create;  
    aJO.AddPair(TJSONString.Create('姓名'), TJSONString.Create(employee.Name));  
    aJO.AddPair(TJSONString.Create('Email'), TJSONString.Create(employee.Email));  
    aJO.AddPair(TJSONString.Create('電話'), TJSONString.Create(employee.Phone));  
    Result := aJO;  
end;
```

## **GetAllEmployeesJ**方法的實作

GetAllEmployeesJ方法會把所有存在泛型型態變數FEmployees之中的TEmployee物件封裝在TJSONArray物件中回傳給用戶端。在013行仍然是呼叫CreateEmployeeJSONObjectJ為每一個CreateEmployeeJSONObjectJ建立一個TJSONObject物件，並且加入到回傳的TJSONArray物件中。

```
001 function TEmployeeVO.GetAllEmployeesJ: TJSONArray;  
002 var  
003     ie : TList<uEmployee.TEmployee>.TEnumerator;  
004     employee : TEmployee;  
005     jo : TJSONObject;  
006     ja : TJSONArray;  
007 begin  
008     ie := FEmployees.GetEnumerator;
```

```
009   ja := TJSONArray.Create;  
010   while (ie.MoveNext) do  
011   begin  
012     employee := ie.Current;  
013     jo := CreateEmployeeJSONObject(employee);  
014     ja.AddElement(jo);  
015   end;  
016   Result := ja;  
017 end;
```

最後不要忘記在主表單中註冊TEmployeeVO類別，如此一來用戶端才能看見並且呼叫TEmployeeVO輸出的方法：

```
procedure TForm10.FormCreate(Sender: TObject);  
begin  
  if DSServer1.Started then  
    DSServer1.Stop;  
  RegisterServers;  
  DSServer1.Start;  
end;
```

```
procedure TForm10.FormDestroy(Sender: TObject);  
begin  
  DSServer1.Stop;  
end;
```

```
procedure TForm10.RegisterServers;  
begin  
  uEmployeeValueObject.RegisterServerClasses(Self, DSServer1);  
end;
```

現在編譯並且執行DataSnap應用程式伺服器，並且準備開發用戶端。

### 開發範例用戶端

其實這個範例的重點就在於用戶端如何呼叫遠端支援TJSONValue相關類別的方法，這是因為在DataSnap 2009中無法支援TJSONValue相關類別，到了DataSnap 2010才支援。但是在筆者撰寫本章時Delphi 2010的Beta版仍然無法自動產生代表遠端類別的用戶端Proxy類別，因此想要呼叫前面實作的GetEmployeeJ和GetAllEmployeesJ方法，我們必須瞭解如何修改由Delphi產生的DataSnap用戶端類別。筆者認為當Delphi 2010正式版釋出時，CodeGear應該會把這個問題修正，如果您發現您使用的Delphi已經能夠產生正確的用戶端Proxy類別，那麼就不需要如下所敘述的修改了。

首先建立一個VCL Form應用程式，放入TSQLConnection元件，設定Driver為DataSnap再把connected設定為True(請確定DataSnap應用程式伺服器已經在執行)，接著用點選滑鼠右鍵，選擇Generate DataSnap client classes選項，儲存產生的程式單元為uServerProxy.pas，然後搜尋GetEmployeeJ方法，把012行修改為如下：

(註，筆者已經在Delphi 2010正式版中試過現在沒有問題了，而筆者之所以沒把上述段落內容刪除是因為這段內容可以讓讀者更瞭解修改下面程式碼的意義，即使讀者使用Delphi 2010正式版而不需要再修改下面的程式碼，也可以把uServerProxy.pas開啟，再看看由Delphi 2010產生的程式碼代表的意義)

```

001 function TEmployeeVOClient.GetEmployeeJ(sName: string): TJSONObject;
002 begin
003     if FGetEmployeeJCommand = nil then
004     begin
005         FGetEmployeeJCommand := FDBXConnection.CreateCommand;
006         FGetEmployeeJCommand.CommandType := TDBXCommandTypes.DSServerMethod;
007         FGetEmployeeJCommand.Text := 'TEmployeeVO.GetEmployeeJ';
008         FGetEmployeeJCommand.Prepare;
009     end;
010     FGetEmployeeJCommand.Parameters[0].Value.SetWideString(sName);
011     FGetEmployeeJCommand.ExecuteUpdate;
012     Result := FGetEmployeeJCommand.Parameters[1].Value.GetJSONValue as TJSONObject;
013 end;

```

這是因為遠端**GetEmployeeJ**方法回傳**TJSONObject**型態的物件，因此我們需要把**012**行呼叫遠端方法執行的結果轉變型態為**TJSONObject**型態。同樣的，我們也需要修改**GetAllEmployeesJ**方法，轉變型態為回傳**TJSONArray**物件，如下所示：

```

function TEmployeeVOClient.GetAllEmployeesJ: TJSONArray;
begin
    if FGetAllEmployeesJCommand = nil then
    begin
        FGetAllEmployeesJCommand := FDBXConnection.CreateCommand;
        FGetAllEmployeesJCommand.CommandType := TDBXCommandTypes.DSServerMethod;
        FGetAllEmployeesJCommand.Text := 'TEmployeeVO.GetAllEmployeesJ';
        FGetAllEmployeesJCommand.Prepare;
    end;
    FGetAllEmployeesJCommand.ExecuteUpdate;
    Result := FGetAllEmployeesJCommand.Parameters[0].Value.GetJSONValue as TJSONArray;
end;

```

瞭解了如何以及為什麼需要修改**DataSnap**用戶端類別之後，我們就可以開始實作呼叫遠端方法的程式碼了。

首先下面的程式碼藉由自動產生的**TEmployeeVOClient**類別呼叫**AddEmployee**方法，把用戶端輸入的員工資料加入在遠端的應用程式伺服器中：

```

procedure TForm11.btnAddEmployeeClick(Sender: TObject);
var
    evo : TEmployeeVOClient;
begin
    evo := TEmployeeVOClient.Create(Self.SQLConnection1.DBXConnection);
    try
        evo.AddEmployee(edtAddName.Text, edtAddEMail.Text, edtAddPhone.Text);
        edtAddName.Text := '';
        edtAddEMail.Text := '';
        edtAddPhone.Text := '';
    finally
        evo.Free;
    end;
end;

```

```
end;
end;
```

下圖是執行範例用戶端應用程式並且點選『增加員工』按鈕以執行上述程式碼的畫面：



圖8 用戶端應用程式呼叫遠端AddEmployee方式加入員工資訊

接著是藉由TEmployeeVOClient呼叫GetEmployeeJ查詢員工資料的實作程式碼：

```
procedure TForm11.btnQueryEmployeeClick(Sender: TObject);
var
  evo : TEmployeeVOClient;
  jo : TJSONObject;
  employee : TEmployee;
begin
  evo := TEmployeeVOClient.Create(Self.SQLConnection1.DBXConnection);
  try
    jo := evo.GetEmployeeJ(edtQname.Text);
    employee := TEmployee.Create(jo.Get(0).JsonValue.ToString, jo.Get(1).JsonValue.ToString, jo.Get(2).JsonValue.ToString);
    Self.edtQEMail.Text := employee.EMail;
    Self.edtQPhone.Text := employee.Phone;
  finally
    FreeAndNil(employee);
    FreeAndNil(jo);
    evo.Free;
  end;
end;
```

在上面的程式碼中呼叫GetEmployeeJ取得代表員工的TJSONObject物件，再根據TJSONObject物件中的資訊於用戶端建立TEmployee物件，再顯



示員工資訊於表單中，最後不要忘記釋放TEmployee物件，TJSONObject物件和TEmployeeVOClient物件。  
下圖是先增加李維這筆員工資料之後，再輸入李維來查詢的畫面：



圖9 輸入員工姓名查詢員工資料

點選上圖中的『查詢單一員工』按鈕之後我們的確可以取得遠端員工物件的資訊，如下圖所示：

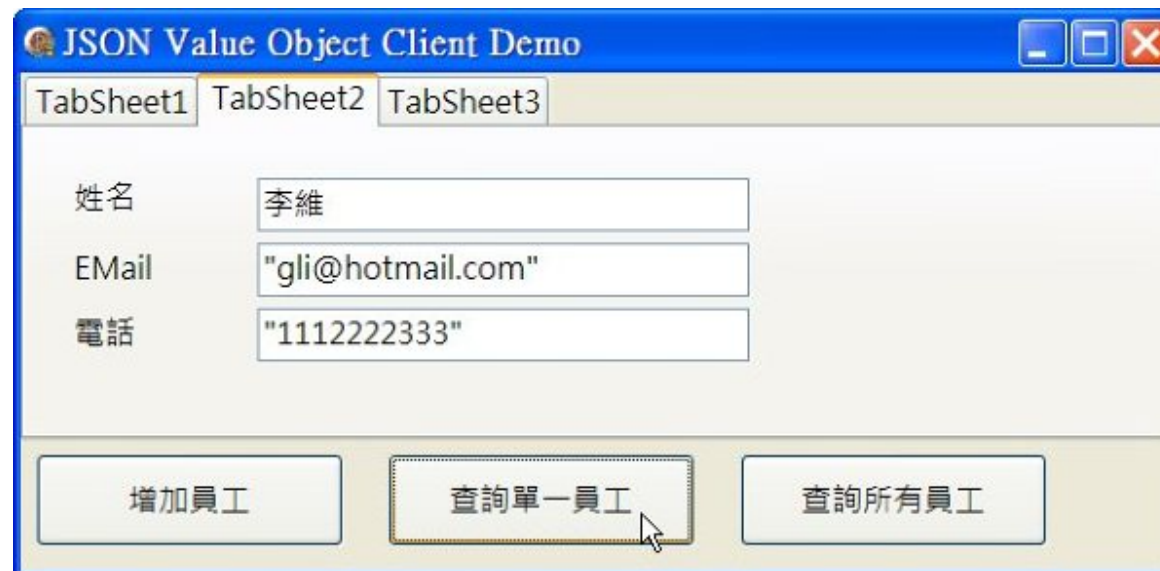


圖10 查詢的結果畫面

但是為什麼上圖中的員工資訊，例如員工的Email有引號包圍呢？這當然是因為這是JSON封裝字串的規範，而在上面的程式碼中當我們建立TEmployee物件時是直接呼叫ToString方法，如果我們不希望TEmployee物件的特性值有引號包圍，那麼我們可以修改程式碼如下，改呼叫Value方法：



```
employee := TEmployee.Create(jo.Get(0).JsonValue.Value, jo.Get(1).JsonValue.Value, jo.Get(2).JsonValue.Value);
```

那麼就會有如下正確的結果：



圖11 查詢的結果畫面

最後讓我們看看如何查詢所有的員工資料。下面的程式碼藉由TEmployeeVOClient呼叫GetAllEmployeesJ取得TJSONArray物件，然後進入for迴圈把其中的每一個元素取出再轉變型態為TJSONObject物件，最後再根據TJSONObject物件一一的建立用戶端的員工物件。

```
procedure TForm11.btnQueryAllEmployeesClick(Sender: TObject);
var
  evo : TEmployeeVOClient;
  jo : TJSONObject;
  ja : TJSONArray;
  employee : TEmployee;
  iIndex: Integer;
begin
  evo := TEmployeeVOClient.Create(Self.SQLConnection1.DBXConnection);
  try
    ja := evo.GetAllEmployeesJ;
    for iIndex := 0 to ja.Size - 1 do
      begin
        jo := ja.Get(iIndex) as TJSONObject;
        employee := TEmployee.Create(jo.Get(0).JsonValue.ToString, jo.Get(1).JsonValue.ToString, jo.Get(2).JsonValue.ToString);
        Memo1.Lines.Add(employee.Name);
        FreeAndNil(employee);
      end;
    finally
      FreeAndNil(ja);
    end;
```

```
FreeAndNil(employee);  
evo.Free;  
end;  
end;
```

下圖是執行此查詢的結果，我們可以看到用戶端的確可以查詢到遠端的所有員工的資訊。

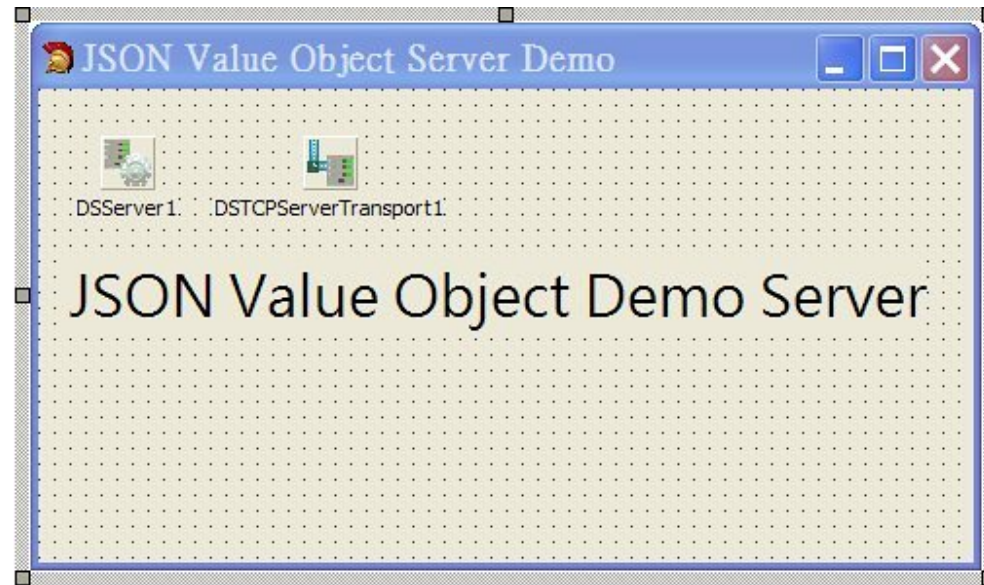


圖12 所有員工資料查詢的結果畫面

當然，我們一樣可以修改上面的程式碼如下：

```
employee := TEmployee.Create(jo.Get(0).JsonValue.Value, jo.Get(1).JsonValue. Value, jo.Get(2).JsonValue. Value);
```

那麼我們會看到如下的結果：



圖13 所有員工資料查詢的結果畫面

本章敘述了Delphi VCL框架中支援JSON開發的相關類別，讀者從本章中可以瞭解這些不但是依照JSON規範設計的，而且非常的直覺，好用，只要我們對於JSON規範有基本的掌握就可以順利的使用它們。

此外，新版的DataSnap不但使用JSON做為封裝和傳遞資料的基礎，現在也加入支援使用TJSONValue和衍生類別做為遠端方法的參數和回傳值，如此一來可以讓開發人員撰寫客製化的JSON應用，例如使用VO/DTO設計樣例來傳遞資料和物件，使用這樣的技術，開發人員也可以使用.NET，Java或是任何支援JSON的程式語言來開發用戶端應用程式了。

練完2篇文章的基本功之後，在下篇文章開發，我們將介紹如何使用Delphi 2010新的DataSnap Server精靈來大幅簡化開發的工作，並且會介紹DataSnap強大的Marshaling和unMarshaling的功能，Marshaling和unMarshaling將可以讓我們使用JSON傳遞物件變得非常簡單，使用Marshaling和unMarshaling也可以讓上面傳遞TEmployee物件的程式碼大幅減少。我們下次再見，Have FUN!

11:26 AM | [Blog it](#) | [Delphi 2010](#)程式設計

## Comments (1)

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](#)

xyzhq wrote:

请问大师在大陆出版此书吗？都有哪些内容？



6 hours ago

## Trackbacks

The trackback URL for this entry is:

<http://gordonliwee.spaces.live.com/blog/cns!CCE1F10BD8108687!3788.trak>

Weblogs that reference this entry

- None