

## Tiburon 遊記 3 動手建立一個 DataSnap JSON 伺服器吧

也許讓我們先動手用 Tiburon 實作一個 DataSnap JSON 分散式架構再搭配前面說明的觀念的話，各位將會更加瞭解 Tiburon 把這些強大的功能做得多麼的方便。

DataSnap 新的 JSON 分散式架構可以有許多不同的型態，更可以結合資料庫和 Web 應用程式，不過在一開始讓我們先學習如何建立最簡單的 JSON 分散式架構，下面是我們即將實作的 JSON 分散式架構的簡單說明：

1. 建立一個分散式 JSON 伺服器，在 JSON 伺服器中提供伺服端的服務方法讓用戶端可以呼叫使用
2. 建立一個 DataSnap 用戶端應用程式，藉由 TCP + JSON 架構呼叫分散式 JSON 伺服器提供的服務

在步驟 1 中我們將看到 Tiburon 新的 Reflect 功能如何把伺服端的服務方法輸出，而步驟 2 我們則可以看到 Tiburon 的 DataSnap 元件如何在原有的元件中加入可以處理 JSON 分散式架構的功能。

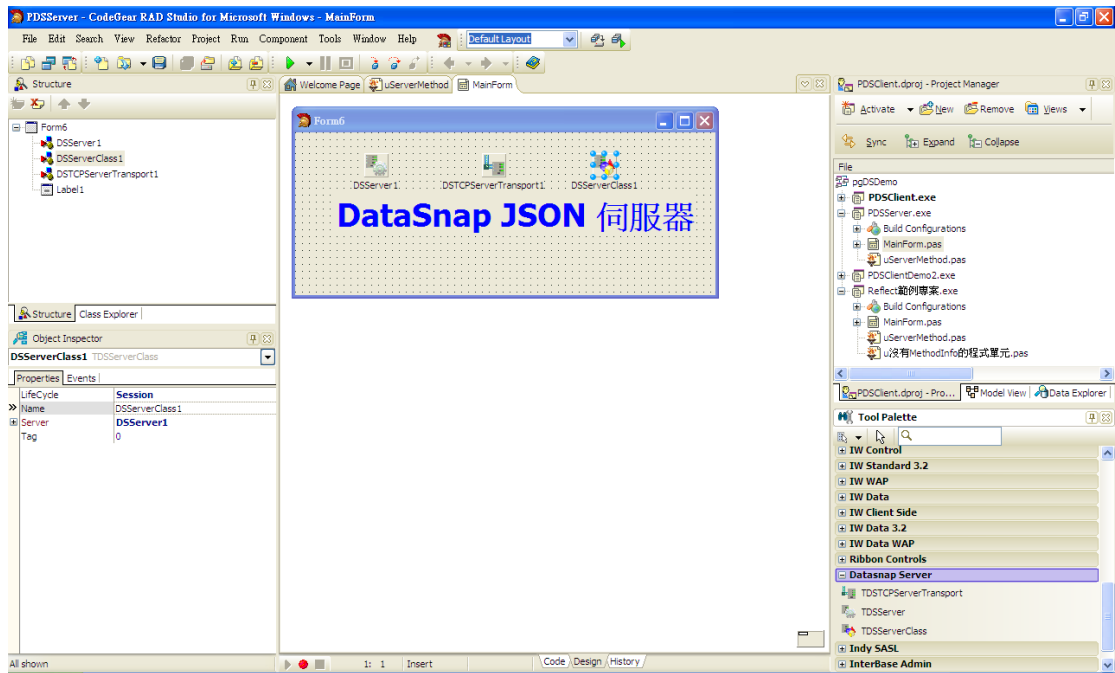
### 步驟 1 – 建立分散式 JSON 伺服器

---

要建立分散式 JSON 伺服器非常的簡單，它的步驟如下：

1. 建立一個 VCL Form 的應用程式
2. 在主表單中加入 TDSServer 元件，TDSServer 元件負責 JSON 伺服器和用戶端的生命週期管理，並且提供了許多管理的機制。
3. 在主表單中加入 TDSTCPServerTransport 元件，TDSTCPServerTransport 元件使用 Indy 做為底層 TCP 通訊元件，TDSTCPServerTransport 元件內定上使用 211 通信埠做為和用戶端連結的通信埠，它並且可以提供通訊池的功能以便有效率的使用伺服器的資源
4. 在主表單中加入 TDSServerClass 元件，TDSServerClass 元件可以把伺服器中的類別輸出給用戶端呼叫，或是把資料模組，遠端資料模組中的方法輸出給用戶端呼叫。TDSServerClass 元件藉由 Tiburon 新強化的 RTTI 和 Reflect 程式單元中的功能以達成這種能力。對於需要輸出服務給用戶端的類別，資料模組或是遠端資料模組，必須使用新的編譯器指令 `{ $MethodInfo ON }` 和 `{ $MethodInfo OFF }` 包圍類別宣告。

5. 連結 TDSTCPServerTransport 元件到 TDSServer 元件，此時主表單應該看起來如下：



現在在這個範例 VCL Form 的應用程式專案中加入一個新的程式單元，並且撰寫如下的程式碼：

```
unit uServerMethod;  
  
interface  
  
uses Sysutils, classes;  
  
type  
  
{ $MethodInfo ON }  
  TServerMethodClass = class(TPersistent)  
  public  
    function Hello(const Name: string): string;  
  end;  
{ $MethodInfo OFF }  
  
implementation  
  
{ TServerMethodClass }
```

```
function TServerMethodClass.Hello(const Name: string): string;
begin
    Result := 'Hello ' + Name;
end;

end.
```

各位可以看到上面宣告了一個 `TServerMethodClass`，它必須從 `TPersistent` 類別繼承下來，由於它需要函式 `Hello` 到用戶端，因此在 `TServerMethodClass` 類別之前和之後使用新的編譯器指令 `{$MethodInfo ON}` 和 `{$MethodInfo OFF}` 包圍，如此一來 `Tiburon` 的編譯器在編譯這個類別時便會產生額外的 `RTTI` 資訊。

6. 在 `TDSServerClass` 元件的 `OnGetClass` 事件處理函式中撰寫如下的程式碼:

```
procedure TForm6.DSServerClass1GetClass (DSServerClass: TDSServerClass;
    var PersistentClass: TPersistentClass);
begin
    PersistentClass := TServerMethodClass;
end;
```

`TDSServerClass` 元件的 `OnGetClass` 事件是用戶端向 `DataSnap JSON` 伺服器查詢可呼叫的服務類別時被呼叫，由於在這個範例中我們是要輸出 `TServerMethodClass` 類別之中的 `Hello` 函式，因此在 `OnGetClass` 事件處理函式中把參數 `PersistentClass` 設定為 `TServerMethodClass` 類別。

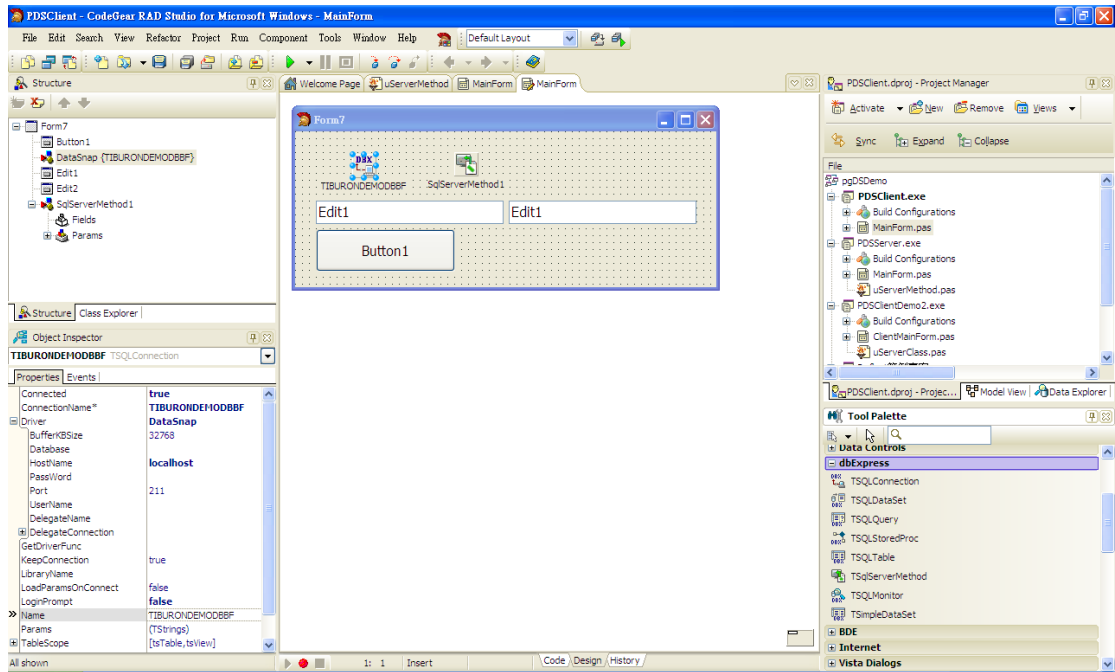
現在這個分散式 `JSON` 伺服器已經完成，編譯它並且執行它，現在我們就擁有了一個不再需要使用 `COM/COM+` 的分散式伺服器了，它簡單，方便又快速，接下來我們就可以建立用戶端應用程式來呼叫它提供的服務方法了。

## 步驟 2 – 建立分散式用戶端

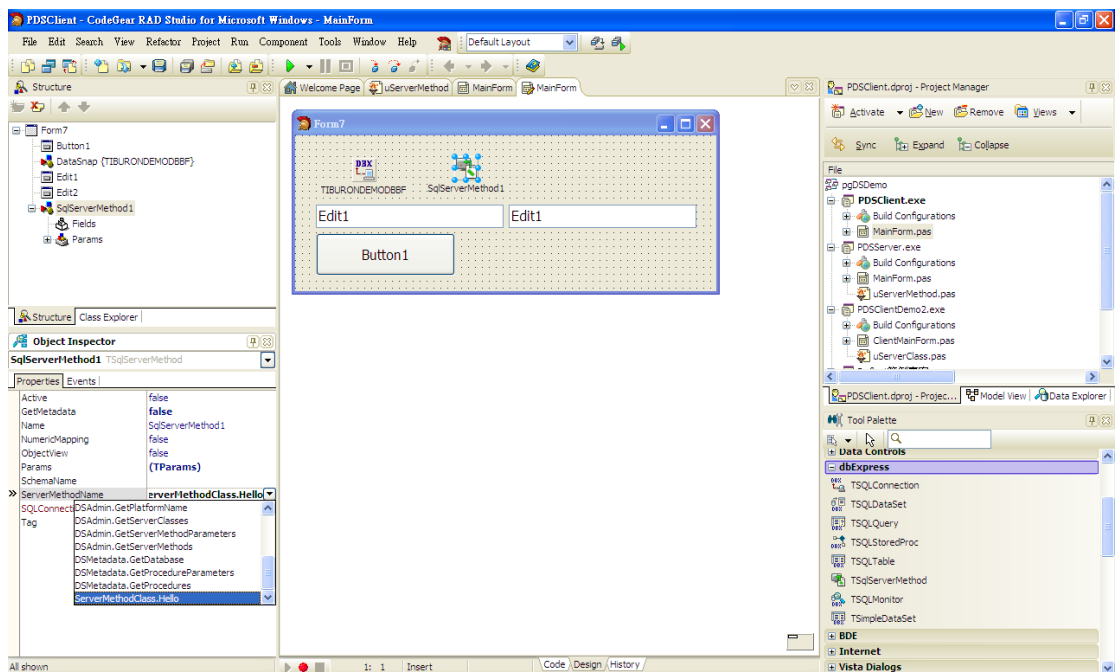
---

要建立分散式用戶端也非常的簡單，它的步驟如下:

1. 建立一個 `VCL Form` 的應用程式
2. 在主表單中加入 `TSQLConnection` 元件，在它的 `Driver` 特性中選擇使用 `DataSnap`，並且在 `HostName` 中輸入分散式 `JSON` 伺服器的名稱，由於我們是在相同的機器中執行，因此可以輸入 `localhost`。另外注意它的 `Port` 特性值也是內定為 `211`，各位可以參考下面的圖形:



3. 在主表單中加入 TSqlServerMethod 元件，連結它到步驟 1 的 TSQLConnection 元件，此時下拉它的 ServerMethodName 特性便可以看到許多伺服器輸出的方法，其中有許多方法是和管理有關的，我們以後有機會再說明，在這些輸出的方法中我們可以找到分散式 JSON 伺服器輸出的 TServerMethodClass.Hello，如下圖所示，請選擇 TServerMethodClass.Hello。



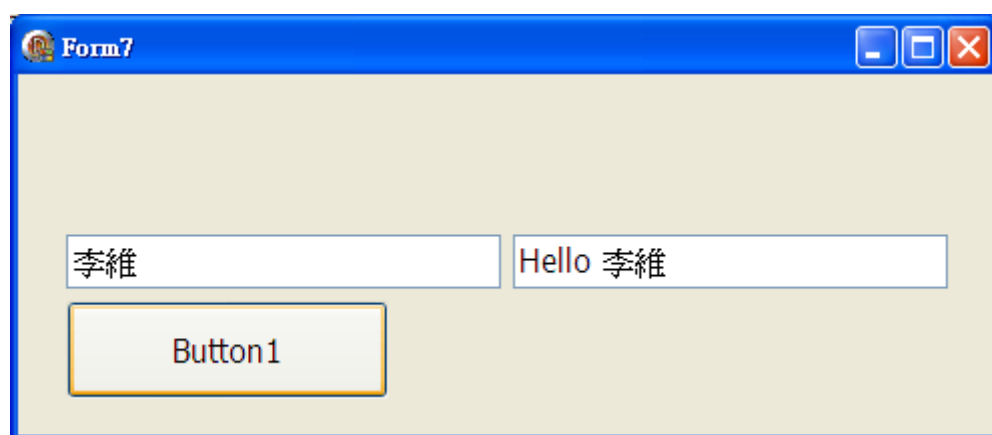
TSqlServerMethod 元件可以讓開發人員在用戶端呼叫遠端分散式 JSON 伺服器輸出的服務方法。

4. 最後在主表單的 **Button** 控制項的 **OnClick** 事件處理函式中撰寫如下的程式碼:

```
01 procedure TForm7.Button1Click(Sender: TObject);
02 begin
03   Self.SqlServerMethod1.ParamByName('Name').Value := Edit1.Text;
04   Self.SqlServerMethod1.ExecuteMethod;
05   Edit2.Text :=
Self.SqlServerMethod1.ParamByName('ReturnParameter').AsString;
06 end;
```

請注意在 03 行中我們設定 **TSqlServerMethod** 元件的 **Param** 特性中'Name'這個參數的數值，但是'Name'這個參數是從那裡來的？請回頭看看前面 **TServerMethodClass.Hello** 函式的宣告原型，它接受一個參數，它的名稱就是'Name'，因此 **TSqlServerMethod** 元件可以分析遠端 JSON 伺服器輸出的方法，每一個參數的名稱就成為 **TSqlServerMethod** 元件的 **Param** 特性中的名稱，而當 **TSqlServerMethod** 元件呼叫 **ExecuteMethod** 方法真正的執行完畢遠端 JSON 伺服器輸出的方法後，如果遠端 JSON 伺服器輸出的方法會回傳函式值，接著開發人員可以藉由 'ReturnParameter'這個名稱的參數來取得執行的結果，如同上面 05 行所示。

現在我們執行用戶端，點選按鈕就可以看到遠端 JSON 伺服器輸出的方法果然正確回傳了執行結果。



## 使用簡單的 Delphi Reflect 功能

---

Tiburon 強化了 RTTI 並且提供了簡易的 **Reflect** 程式單元可以讓 JSON 伺服器輸出服務到用戶端，我們也可以利用這個功能來驗證一下。

首先我建立一個新的 VCL 應用程式，在專案中建立兩個額外的類別，一個類別使用新的編譯器指令 `{MethodInfo ON}` 和 `{MethodInfo OFF}` 包圍：

```
unit uServerMethod;  
  
interface  
  
uses Sysutils, classes;  
  
type  
  
{MethodInfo ON}  
  TServerMethodClass = class(TPersistent)  
  public  
    function SayHello(const Name: string): string;  
    function GetReflectMessage : String;  
    procedure ReflectMe;  
  private  
    FData : String;  
  end;  
{MethodInfo OFF}
```

另外一個則否：

```
unit u沒有MethodInfo的程式單元;  
  
interface  
  
uses Sysutils, classes;  
  
type  
  
  TServerMethodWithoutMethodInfoClass = class(TPersistent)  
  public  
    function SayHello(const Name: string): string;  
    function GetReflectMessage : String;  
    procedure ReflectMe;  
  private  
    FData : String;  
  end;
```

各位可以看到 **Tiburon** 支援了 **Unicode** 之後連程式單元的名稱都可以使用中文命名。

接著我在主表單中加入參考 **ObjAuto** 程式單元，並且使用下面的程式碼就可以取得類別方法的資訊：

```
procedure TForm7.btn 有 MethodInfo 的按鈕 Click(Sender: TObject);
var
  aClassObj : TServerMethodClass;
  mdArray : TMethodInfoArray;
  iCount: Integer;
begin
  aClassObj := TServerMethodClass.Create;
  try
    mdArray := ObjAuto.GetMethods(aClassObj.ClassType);
    for iCount := 0 to Length(mdArray) - 1 do
    begin
      ListBox1.Items.Add(mdArray[iCount].Name);
    end;
  finally
    aClassObj.Free;
  end;
end;

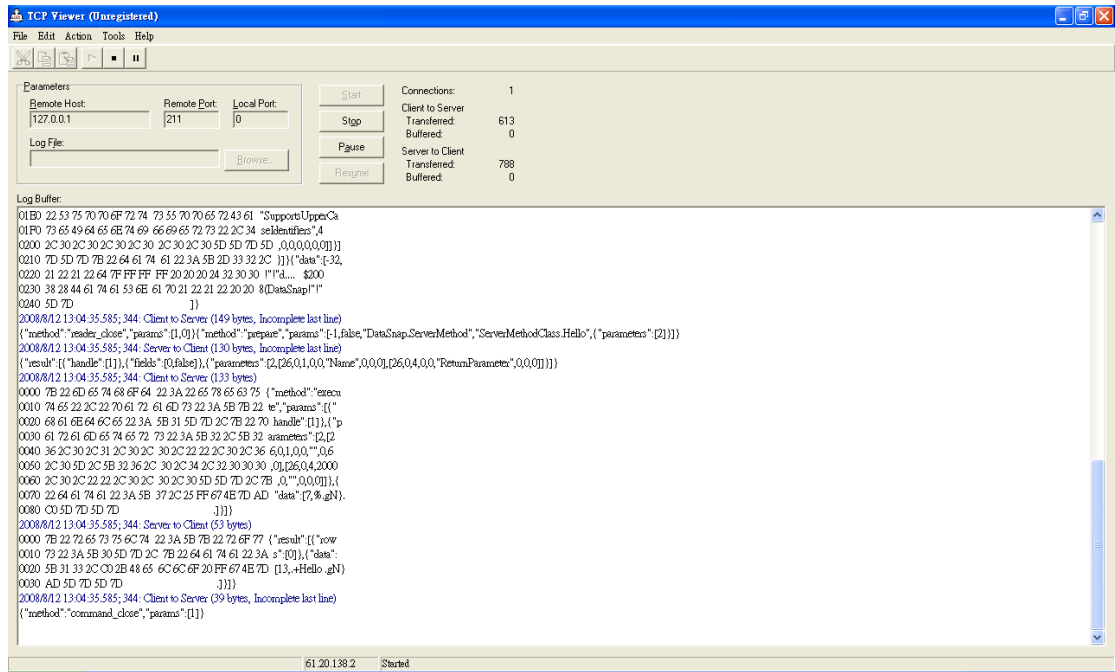
procedure TForm7.Button2Click(Sender: TObject);
var
  aClassObj : TServerMethodWithoutMethodInfoClass;
  mdArray : TMethodInfoArray;
  iCount: Integer;
begin
  aClassObj := TServerMethodWithoutMethodInfoClass.Create;
  try
    mdArray := ObjAuto.GetMethods(aClassObj.ClassType);
    for iCount := 0 to Length(mdArray) - 1 do
    begin
      ListBox1.Items.Add(mdArray[iCount].Name);
    end;
  end;
```

```
finally
    aClassObj.Free;
end;
end;
```

`ObjAuto.GetMethods` 可以取得類別方法的資訊，如果類別沒有使用新的編譯器指令 `{MethodInfo ON}` 和 `{MethodInfo OFF}` 包圍，那麼就無法產生足夠的 RTTI 資訊，因此用戶端將無法存取(看到)類別中的方法。現在執行這個範例 VCL 應用程式，我們可以看到左邊的 `ListBox` 可以顯示 `TServerMethodClass` 類別中的方法，而右邊的 `ListBox` 則無法顯示 `TServerMethodWithoutMethodInfoClass` 類別中的方法。



下圖是我使用 `TCP Viewer` 觀查前面分散式 `JSON` 伺服器 and 分散式用戶端之間訊息的結果，我們可以看到它們之間的确是使用 `JSON` 格式在傳遞訊息。



Tiburion 的分散式 JSON 架構除了可以輸出伺服器的服務之外，也可以結合資料庫實現真正的 Thin-Client，離線資料架構型態的應用系統，這些都基於 Delphi 開發人員原本就熟悉的 DataSnap 架構，我很快會寫一小篇文章展示它是多麼的容易而且又是使用我們熟悉的 DataSnap 元件。

Have Fun!