


Blog

< Previous

 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

September 11  
Delphi 2010 JSON程式設計

從Delphi 2009開始Delphi便開始支援JSON，DataSnap 2009開始使用JSON做為開發分散式應用系統的基礎技術，開始逐漸捨棄使用COM/DCOM/COM+。由於JSON具備跨平台，跨程式語言和跨工具的特性，因此DataSnap在結合JSON和dbExpress之後也逐漸具備了跨平台的基礎。但是Delphi 2009對於JSON的支援只限於使用在DataSnap之中，因此如果開發人員需要進行通用的JSON程式設計，Delphi 2009在這方面仍然顯得不足。

到了Delphi 2010情形開始改變，因為VCL框架開始內建支援JSON的類別，因此Delphi的開發人員就可以利用這些和JSON相關的類別來進行JSON程式設計的工作。本章的重點就是討論VCL框架對於JSON的支援，本章將討論如何使用VCL框架中和JSON相關的類別來進行JSON開發的工作。不過在討論這些類別之前，我們需要先簡單的介紹什麼是JSON。

X-1 JSON是什麼？

簡單的說，JSON是一種資料封裝格式以及資料交換格式，JSON是JavaScript Object Notation的縮寫，它是一種輕量級的資料交換格式，易於一般人閱讀和編寫，同時也易於機器解析和生成。

JSON是基於JavaScript（Standard ECMA-262 3rd Edition - December 1999）的一個子集，JSON採用完全獨立于語言的文字格式，但是也使用了類似於C語言家族的習慣（包括C, C++, C#, Java, JavaScript, Perl, Python等），這些特性使JSON成為理想的資料交換語言。

JSON是由下面的兩個基本架構組成的：

- “名稱/值”對的集合（A collection of name/value pairs）。在不同的程式語言中，這個架構經常以物件（object），紀錄（record），結構（struct），字典（dictionary），雜湊表（hash table），有鍵列表（keyed list），或者關聯陣列（associative array）來實作。
- 值的有序列表（An ordered list of values）。在大部分的程式語言中，這個架構經常使用陣列（array）來實作。

OK，看了上述的定義之後，讀者可能還是覺得模模糊糊，也許讓我們使用實際的範例來說明可以讓讀者更容易瞭解。

假設現在我們有一個DataSnap伺服器，它使用JSON和用戶端進行資料交換的工作，那麼現在用戶端向這個DataSnap伺服器查詢筆者所撰寫的書籍，例如本書『實戰Delphi 2010』，那麼如何把這個資訊以JSON的格式傳遞給用戶端呢？

首先讓我們觀察下圖，在JSON中物件的代表方式是以“{”開始，以“}”結束，另外在前面我們也說明JSON是以『名稱/值』的格式來代表資料，因此從下圖中我們可以看到名稱和值之間是以“:”符號分隔，其中『名稱』是字串格式，而值則是以數值格式來代表：

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

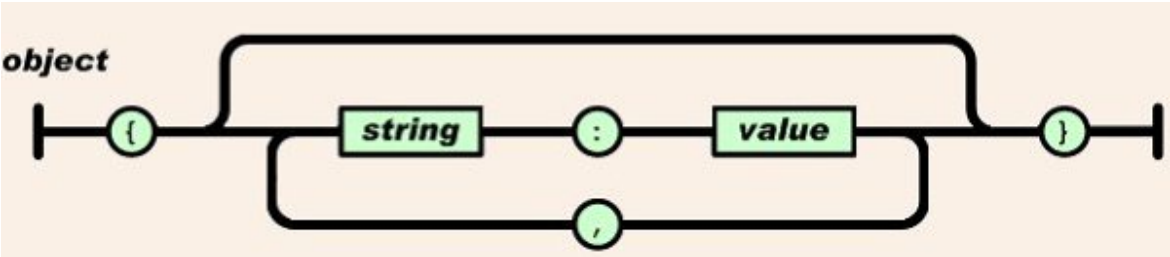


圖1 JSON封裝物件的格式

在JSON規範中上圖的字串規範如下圖所示：

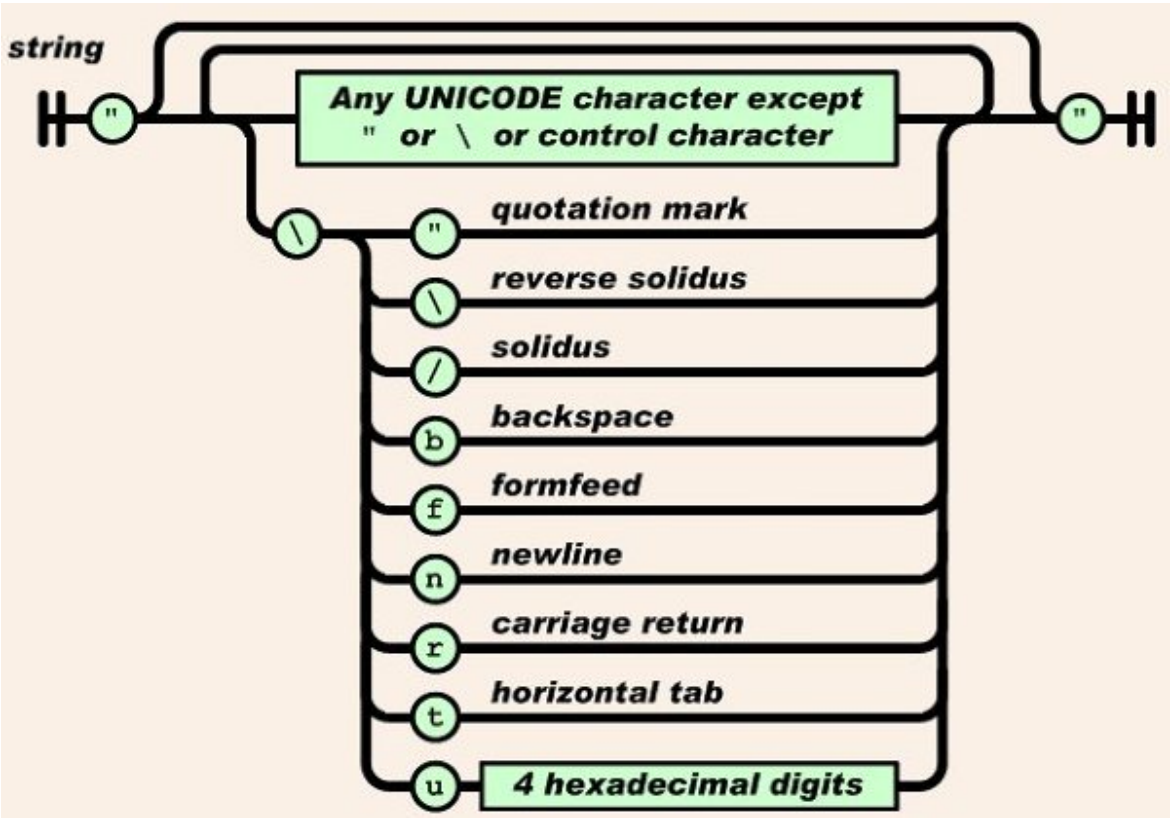


圖2 JSON封裝字串的格式

從上圖中我們可以知道在JSON中字串是以"開始，以"結束，可包含Unicode的字元組。而圖1中的數值格式則如下圖所示：

January 2006  
December 2005  
November 2005  
October 2005  
September 2005  
August 2005  
July 2005

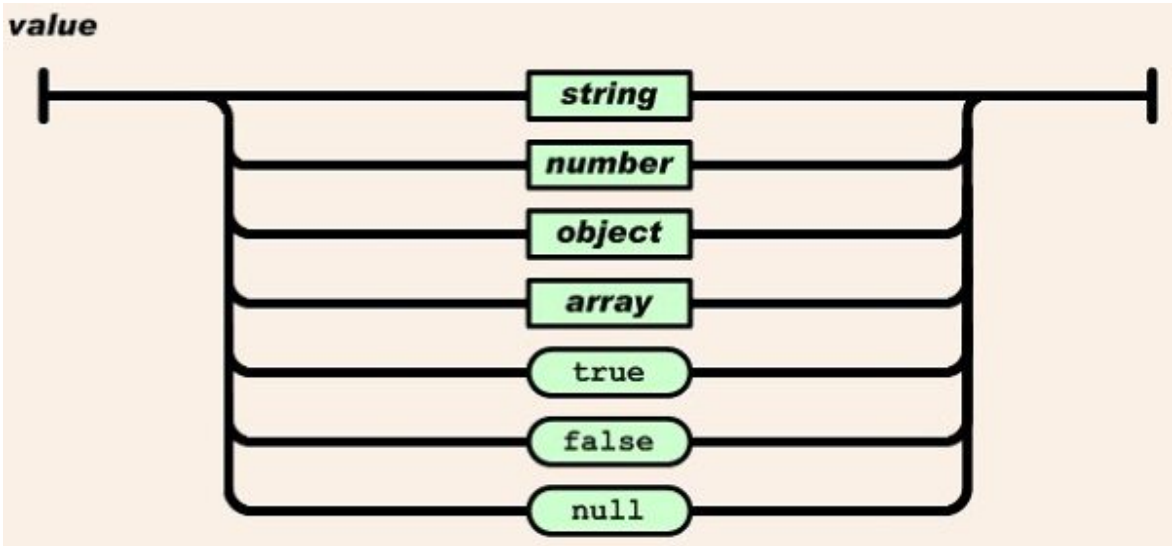


圖3 JSON封裝數值的格式

從圖3中可知，JSON的數值可以是字串，數字，物件，陣列或是true/false/null值。由於JSON的數值可以是物件或是陣列，而物件又可以包含字串:數值配對，因此JSON規範可以封裝任何複雜的資料。

有了上述對於JSON基本的瞭解之後，我們就可以使用如下的格式從DataSnap伺服器傳遞資料到用戶端：

```
{“書名”:“實戰Delphi 2010”}
```

從這個格式中我們可以知道這是用JSON中封裝物件的規則，把『字串:數值』配對包含在{和}符號之中。

如果我們希望也傳遞作者的資訊，那麼可以使用逗號分離每一個『字串:數值』配對，如下所示，讀者也可以回頭再參考圖1就可以發現這是圖1的規則。

```
{“書名”:“實戰Delphi 2010”,“作者”:“李維”}
```

如果現在再把書籍出版年份也傳遞，那麼就可以使用下面的格式：

```
{"書名":"實戰Delphi 2010","作者":"李維","出版年份":2010}
```

請注意的是，出版年份的數值是2010這個數字，因為如圖3所示數值可以是數字(number)，而在JSON中數字的定義如下：

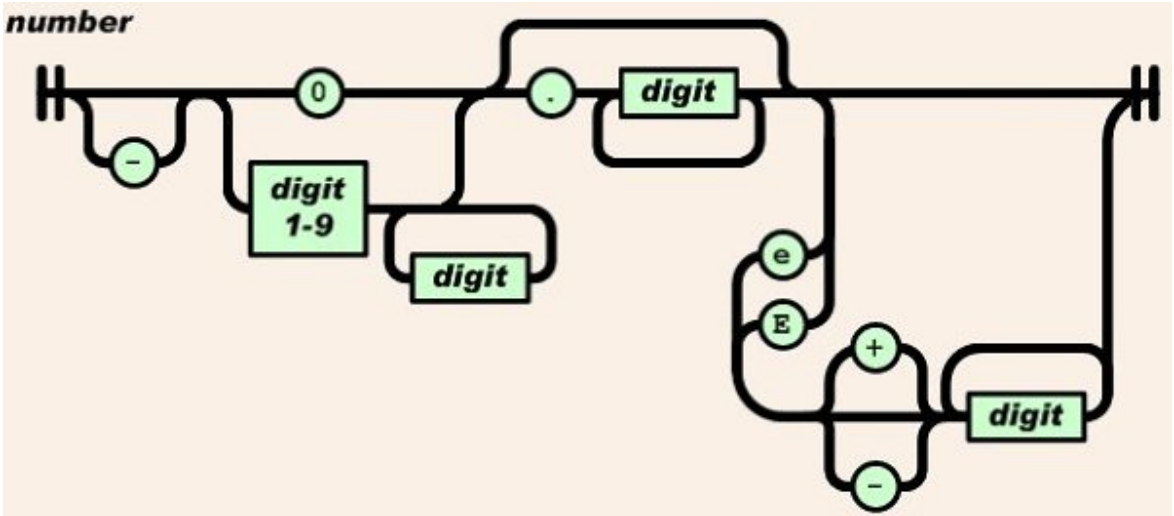


圖4 JSON封裝數字的格式

從圖4我們可以知道，在JSON中數字可以是整數或是浮點數。

```
{"書名":"實戰Delphi 2010","作者":"李維","出版年份":2010}
```

因此如果我們再傳遞書籍價格，那麼可以使用如下的格式：

```
{"書名":"實戰Delphi 2010","作者":"李維","出版年份":2010,"價格":45.95}
```

最後如果我們再加入書籍已出版否資訊，那麼可以使用如下的格式：

```
{"書名":"實戰Delphi 2010","作者":"李維","出版年份":2010,"價格":45.95,"已出版否":false}
```

從上面的範例中我們可以看到JSON規範如何封裝各種不同型態的資料。那麼如果我們需要一次傳遞多筆資料到用戶端的話，又要如何封裝呢？這可以使用JSON中陣列的規範。

圖5是JSON陣列的封裝規則，陣列以“[”開始，以“]”結束，而陣列中的每一個元素都是數值，每一個元素使用逗號分隔。回頭參考圖3 JSON封裝數值的格式，由於數值可以是物件，因此我們如果需要一次傳遞多筆資料到用戶端，那麼就可以使用JSON的陣列來封裝。

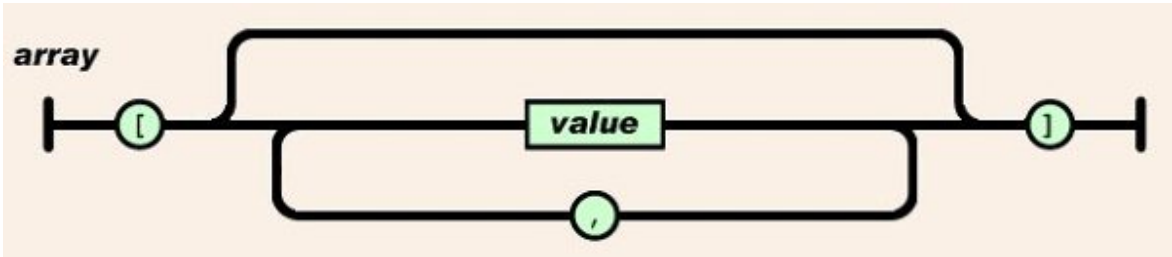


圖5 JSON封裝陣列的格式

例如下面就是使用JSON的陣列封裝性三個代表書籍的物件，陣列中的每一個元素數值是物件，每一個元素數值使用逗號分隔：

```
[{"書名": "實戰Delphi 2010"}, {"書名": "Inside VCL"}, {"書名": "Borland傳奇"}]
```

如何？一旦瞭解了JSON封裝資料的規則之後讀者是不是覺得使用JSON非常的簡潔呢？正由於JSON在封裝資料的規則比較簡單，因此在解析JSON資料時速度也比較快，JSON的簡潔規則讓JSON擁有較少的資料流量和較快的處理速度，因此讓JSON在Web應用方面佔有優勢，也愈來愈受歡迎，這也是DataSnap現在選擇使用JSON做為基礎技術的原因。

在離開本小節之前，讓我們比較一下使用XML和JSON在封裝和交換資料方面的差異。下面是使用XML封裝一個員工的資訊：

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<user>
```

```
<name>李大明</name>
```

```
<password>123456</password>
```

```
<department>R&D</department>
```

```
<gender>男</gender>
```

```
<age>26</age>
```

```
</user>
```

如果我們使用JSON來封裝的話，那麼就如下所示：

```
{
```

```
"name": "李大明",
```

```
"password": "123456",
```

```
"department": "R&D",
```

```
"gender": "男",  
  
"age": "26"  
  
}
```

我們可以看到JSON使用了較XML少的資料量來封裝相同的資訊，此JSON在資料交換方面擁有比較多的優勢。

讀者可以參考[www.json.org](http://www.json.org)以瞭解更多有關JSON的資訊。

在對於JSON有了基本的瞭解之後，更重要的是我們需要知道如何在Delphi中進行JSON的程式設計。

## X-2 VCL框架中支援JSON的類別

Delphi在2009開始使用JSON做為DataSnap 2009封裝和傳遞資料的格式，但是Delphi 2009對於JSON的支援和DataSnap的功能撰寫的非常緊密，開發人員除了能夠在DataSnap 2009中使用JSON之外，並不容易使用Delphi 2009來開發其他應用型態的JSON應用程式。到了Delphi 2010這個現象獲得了大幅的改善，Delphi 2010提供了許多通用的JSON相關類別，開發人員可以使用這些通用的JSON相關類別開發任何型態的JSON應用程式而不只限於DataSnap應用程式。

Delphi 2010在新的DBXJSON.pas程式單元中提供了這些JSON相關的類別，如果讀者仔細觀察前面JSON規則圖的話，就可以發覺如果我們需要使用類別來實作支援JSON的規範，那麼我們至少需要下面的三個類別：

類別	說明
JSONValue	用來代表和實作圖3的實體和關係
JSONPair	用來代表JSON規範中的『名稱/值』配對實體和關係
JSONObject	用來代表和實作圖1的實體和關係
JSONString	用來代表和實作圖2的實體和關係
JSONArray	用來代表和實作圖5的實體和關係
JSONNumber	用來代表和實作圖4的實體和關係

當然在上述的基礎類別中還存在繼承的關係，例如由於JSON規範中數值可以代表字串，物件，陣列等實體，因此上面的JSONObject，JSONString等類別可以從JSONValue類別繼承下來。

Delphi 2010中支援JSON等相關類別就是使用這個觀念實作出來的，因此只要讀者瞭解了前面解釋JSON規範的觀念，就可以非常直覺的瞭解這些實作類別。在下面的表格中整理了這些類別，並且提供了簡單的敘述：

類別	說明
TJSONAncestor	抽象類別，是所有JSON相關類別的根類別



- TJSONPair 實作JSON規範中『名稱/值』配對的類別
- TJSONValue 實作JSON規範中數值的類別，TJSONValue是TJSONAncestor的繼承類別
- TJSONTrue 實作JSON規範中代表True的類別，從TJSONValue繼承下來
- TJSONString 實作JSON規範中代表字串的類別，從TJSONValue繼承下來
- TJSONNumber 實作JSON規範中代表數值的類別，它從TJSONString繼承下來，因為JSON使用文字格式傳遞封裝和傳遞資料，因此所有數值都將轉換為文字字串型態傳遞，到達目的地之後再反轉回數值
- TJSONObject 實作JSON規範中代表物件的類別，從TJSONValue繼承下來
- TJSONNull 實作JSON規範中代表Null的類別，從TJSONValue繼承下來
- TJSONFalse 實作JSON規範中代表False的類別，從TJSONValue繼承下來

下圖是這些類別之間的繼承和關連關係圖，請讀者對照下圖和以前圖1到圖5的JSON規範，就可以瞭解這些類別的實作是完全從JSON規範中設計出來的，簡單又符合直覺：

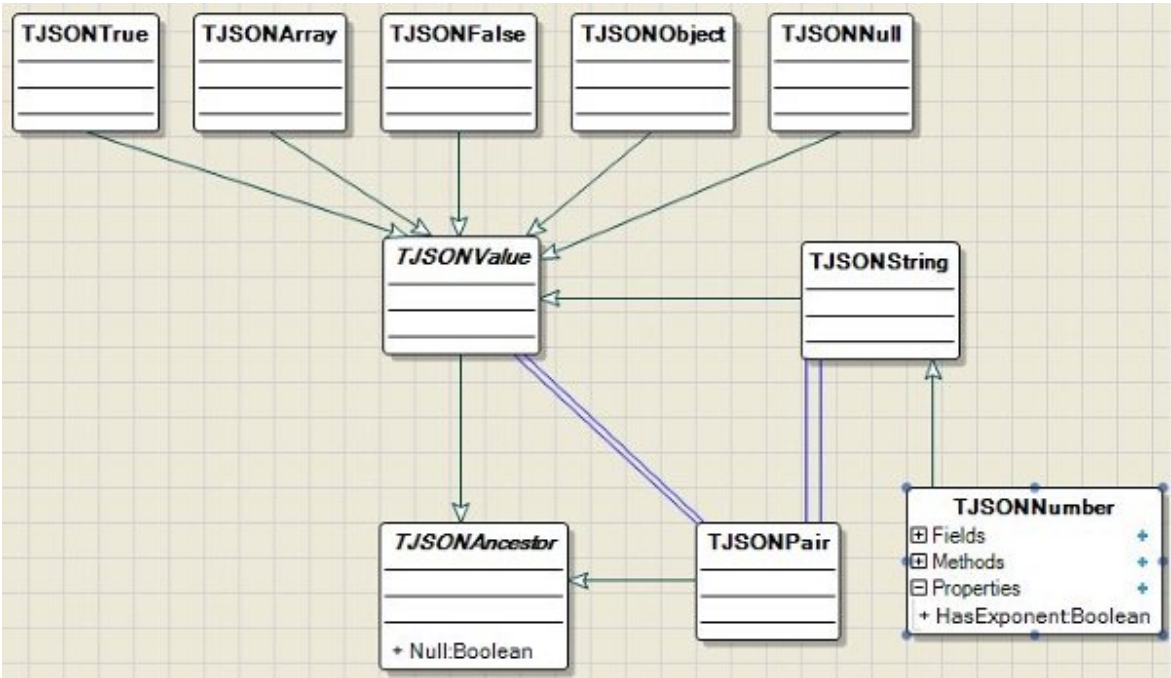


圖6 Delphi中支援JSON的類別架構

接下來讓我們簡單的說明這些類別提供的服務，如此一來讀者就可以瞭解如何在程式碼中使用它們。

## X-2-1 TJSONAncestor類別

TJSONAncestor類別是所有JSON相關類別的根類別，它主要定義了三個虛擬方法讓它的衍生類別來複載實作，下面的表格敘述了這些虛擬方法：

虛擬方法	說明
<code>function ToString: UnicodeString; virtual;</code>	ToString虛擬方法會把JSON的內容轉換為文字字串形式回傳，TJSONAncestor的衍生類別需要實作這個虛擬方法，例如TJSONObject類別會實作這個虛擬方法並且把物件內容轉換為字串的形態回傳
<code>function EstimatedByteSize: Integer; virtual; abstract;</code>	EstimatedByteSize虛擬方法回傳JSON物件預估的位元組大小，TJSONAncestor的衍生類別需要實作這個虛擬方法，每一個衍生類別都會回傳它本身需要佔據位元組的大小。EstimatedByteSize虛擬方法的目的是在封裝和傳遞JSON物件時程式碼能夠配置足夠的記憶體大小之用。
<code>function ToBytes(const Data: TBytes; const Offset: Integer): Integer; virtual; abstract;</code>	ToBytes虛擬方法能夠把JSON物件以位元組的形式回傳，TJSONAncestor的衍生類別需要實作這個虛擬方法，每一個衍生類別都會在這個複載的虛擬方法中把自己轉換為位元組形態。

TJSONAncestor本身是一個抽象類別，因此開發人員並不應該直接在程式碼中使用它，而是應該使用下面介紹的衍生類別。

## X-2-2 TJSONValue類別

TJSONValue類別本身也是一個抽象類別，它是直接從TJSONAncestor繼承下來的，TJSONValue類別只是做為一個Placeholder，它主要的目的是代表圖3中JSON規範value的概念，因此它的定義非常簡單，如下所示：

```
TJSONValue = class abstract(TJSONAncestor)

end;
```

稍後介紹的許多類別都是從TJSONValue繼承下來的，幾乎和圖3顯示的規範架構一樣。

## X-2-3 TJSONPair類別

TJSONPair類別是實作JSON規範中『名稱/值』配對的類別，它從TJSONAncestor直接繼承下來，由於TJSONPair是實作『名稱/值』配對，因此它提供了兩個最重要的特性，JsonString和JsonValue：

```
property JsonString: TJSONString read GetJsonString write SetJsonString;

property JsonValue: TJSONValue read GetJsonValue write SetJsonValue;
```

JsonString是代表『名稱/值』配對中名稱的特性，而JsonValue則代表其中值的特性。請注意的是JsonValue的型態是TJSONValue，而根據前面表格所敘述，TJSONValue可以代表任何從TJSONValue繼承下來的類別，因此JsonValue可以代表TJSONString，TJSONNumber或是TJSONObject等。



那麼如何建立TJSONPair呢？TJSONPair定義了四個複載的建構函式，其中最重要的三個如下所示：

```
constructor Create(const Str: TJSONString; const Value: TJSONValue); overload;  
  
constructor Create(const Str: UnicodeString; const Value: TJSONValue); overload;  
  
constructor Create(const Str: UnicodeString; const Value: UnicodeString); overload;
```

上面第一個建構函式可以藉由TJSONString和TJSONValue物件來建立TJSONPair物件，第二個建構函式可以使用Unicode字串和TJSONValue物件來建立TJSONPair物件，至於第3個建構函式則是使用兩個Unicode字串來建立TJSONPair物件。

例如，如果我們要建立如下的JSON『名稱/值』配對：

“書名”：“實戰Delphi 2010”

那麼我們可以使用如下的程式碼：

```
var  
  
    jp : TJSONPair;  
  
begin  
  
    jp := TJSONPair.Create(TJSONString.Create('書名'), TJSONString.Create('實戰Delphi 2010'));  
  
    try  
  
        Memo1.Lines.Add(jp.ToString);  
  
    finally  
  
        jp.Free;  
  
    end;
```

上面的程式碼使用了第一個建構函式建立TJSONPair物件，TJSONPair類別的ToString方法可以把JSON物件之中的內容以文字字串形式回傳，最後釋放TJSONPair物件時，TJSONPair物件會自動釋放傳入建構函式之中的兩個TJSONString物件。

當然您也可以使用第二個建構函式來建立TJSONPair物件，如下所示：

```
jp := TJSONPair.Create('書名', TJSONString.Create('實戰Delphi 2010'));
```

或是使用第3個建構函式，因為在這個範例中『名稱/值』配對中的名稱和值都是字串：

```
jp := TJSONPair.Create('書名', '實戰Delphi 2010');
```

當然，如果您需要建立如下的JSON『名稱/值』配對：

```
"價格":45.95
```

由於值是數字，因此我們只能使用第一個或是第二個建構函式來建立：

```
jp := TJSONPair.Create(TJSONString.Create('價格'), TJSONNumber.Create(45.95));
```

或是：

```
jp := TJSONPair.Create('價格', TJSONNumber.Create(45.95));
```

一旦建立了TJSONPair物件，我們也可以藉由存取它的JsonString和JsonValue特性來存取『名稱/值』配對中的名稱或是值了，例如：

```
Memo1.Lines.Add('JSONString : ' + jp.JsonString.ToString);
```

```
Memo1.Lines.Add('JSONValue : ' + jp.JsonValue.ToString);
```

## X-2-4 TJSONString類別

TJSONString是從TJSONValue類別繼承下來，它是使用來代表Unicode字串的資料，它可以是『名稱/值』配對中的名稱或是值或是兩者。

TJSONString最重要的方法應該是它的建構函式了，它接受一個Unicode字串做為參數：

```
constructor Create(const Value: UnicodeString); overload;
```

在Unicode字串使用建立TJSONString物件之後，如果開發人員需要加入額外的字元，那麼可以呼叫AddChar虛擬程序，AddChar會在原本的Unicode字串之後加入參數Ch的字元內容。

```
procedure AddChar(const Ch: WideChar); virtual;
```

如果開發人員需要TJSONString物件中字串的內容值，可以呼叫ToString：

```
function ToString: UnicodeString; override;
```

如果開發人員只是需要TJSONString物件中字串的內容值，而不需要額外的開始”符號以及結束的”符號，那麼可以呼叫TJSONString的Value函式

```
function Value: UnicodeString; override;
```

例如下面的程式碼：

```
var
```

```
js : TJSONString;
```

```
begin

  js := TJSONString.Create('實戰Delphi 2010');

  try

    Memo1.Lines.Add('TJSONString.ToString : '+ js.ToString);

    Memo1.Lines.Add('TJSONString.Value : '+ js.Value);

  finally

    js.Free;

  end;
```

下面是呼叫ToString和Value的差異：

TJSONString.ToString : "實戰Delphi 2010"

TJSONString.Value : 實戰Delphi 2010

## X-2-5 TJSONObject類別

TJSONObject類別從TJSONValue直接繼承下來，它代表JSON規範中封裝物件的類別。要建立TJSONObject物件非常簡單，只需要呼叫它的建構函式即可：

```
constructor Create;
```

那麼如果我們需要建立如下的JSON物件內容，那麼應該如何做呢？

{“書名”：“實戰Delphi 2010”}

請注意上面的結構，其實是在JSON物件之中包含一個TJSONPair物件，因此我們只需要執行下面的步驟即可：

- 建立TJSONObject物件
- 建立TJSONPair物件
- 把TJSONPair物件加入到TJSONObject物件

要把TJSONPair物件或是JSON『名稱/值』配對加入到TJSONObject物件中，我們可以使用TJSONObject類別中下面兩個複載的AddPair方法：

```
procedure AddPair(const Pair: TJSONPair); overload; virtual;
```

```
procedure AddPair(const Str: TJSONString; const Val: TJSONValue); overload; virtual;
```

因此如果我們需要建立如下內容的TJSONObject物件：

```
{"書名":"實戰Delphi 2010","出版年份":2010}
```

那麼可以使用如下的程式碼：

```
var

  jo : TJSONObject;

  jp : TJSONPair;

begin

  jo := TJSONObject.Create;

  try

    jp := TJSONPair.Create('書名', '實戰Delphi 2010');

    jo.AddPair(jp);

    jo.AddPair(TJSONString.Create('出版年份'), TJSONNumber.Create(2010));

    Memo1.Lines.Add(jo.ToString);

  finally

    jo.Free;

  end;
```

同樣的TJSONObject類別的ToString方法可以把其中的內容以文字字串型態回傳：

```
function ToString: UnicodeString; override;
```

## X-2-5 TJSONNumber類別

TJSONNumber類別是從TJSONString類別繼承下來的，這是因為在JSON規範中整數和浮點數都必須使用字串形式來代表，由於TJSONString已經提供了使用字串形式來代表JSON內容的功能，因此TJSONNumber只需要從它繼承下來並且把整數和浮點數轉換為文字字串型態即可。

在前面我們看過多次建立TJSONNumber物件的範例了，它的建構函式接受一個double值的參數：

```
constructor Create(const Value: Double); overload;
```

一旦建立了TJSONNumber物件，呼叫它的ToString方法即可取得其內容。

在離開本小節之前再讓我們看看三個剩下簡單的類別，它們是TJSONTrue，TJSONFalse和TJSONNull。這三個類別分別實作圖3中的true，false和null三個JSON數值。例如我們如果需要下面的JSON內容：

```
{"書名":"實戰Delphi 2010","出版否":false, "撰寫中":true, "出版商":null}
```

那麼我們使用下面的程式碼即可：

```
var

  jo : TJSONObject;

  jp: TJSONPair;

begin

  jo := TJSONObject.Create;

  try

    jp := TJSONPair.Create('書名', '實戰Delphi 2010');

    jo.AddPair(jp);

    jp := TJSONPair.Create('書名', TJSONFalse.Create);

    jo.AddPair(jp);

    jp := TJSONPair.Create('撰寫中', TJSONTrue.Create);

    jo.AddPair(jp);

    jp := TJSONPair.Create('出版商', TJSONNull.Create);

    jo.AddPair(jp);

    Memo1.Lines.Add(jo.ToString);

  finally

    jo.Free;

  end;
```

在讀者瞭解了如何使用這些JSON相關的類別之後，讓我們看看如何使用它們在分散式應用系統之中。

4:25 PM | [Blog it](#)

### Comments (2)

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](#)



[維 李](#) wrote:

>大哥，前五張圖片掛點了耶

謝了，剛才應該修好了。

9 hours ago



[c37](#) wrote:

大哥，前五張圖片掛點了耶

3 days ago

### Trackbacks

The trackback URL for this entry is:  
<http://gordonliwei.spaces.live.com/blog/cns!CCE1F10BD8108687!3766.trak>  
Weblogs that reference this entry

- None