

[淺談如何使用 Delphi 2009 的泛型容器類別\(續\)](#)

使用 TDictionary<T,T> 容器類別

Delphi 2009 中的容器類別 TDictionary<T,T>對於許多日常開發的工作中非常的有用，例如我們經常需要在應用程式中使用一個鍵值來搜尋相關的資料，在這種應用中 TDictionary<T,T>容器類別便非常適合。

TDictionary<T,T>容器類別的第一個<T>即代表鍵值，第二個<T>代表這個鍵值相關的數值，由於 TDictionary<T,T>是泛型容器類別，所以它的鍵值和數值都可以是任何的型態。現在就讓我們看看如何使用 TDictionary<T,T>，我們仍然繼續使用前面討論的範例做為說明。

假設現在我們希望統計一下 glProduct 中 Delphi 和 BCB 的產品個數是多少，那麼我們可以使用 TDictionary<T,T>來儲存<產品名稱, 個數>這樣的資訊，因此我們需要建立一個 TDictionary<string, Integer>的物件來使用。

TDictionary<T,T>提供了許多不同的原型的建構函式，其中最簡單的可建構函式擁有如下的原型宣告：

```
constructor Create(ACapacity: Integer = 0); overload;
```

這個建構函式接受一個內定大小的參數 ACapacity，這個參數是指

TDictionary<T,T>物件在建立之後先在其中預先建立多少個元素。為什麼需要這樣?這是因為我們在建立了 TDictionary<T,T>物件之後一定會在其中加入元素(不

然爲什麼需要建立 TDictionary<T,T>物件呢?)，由於 TDictionary<T,T>比較複雜，因此如果我們預先在其中先建立一些元素空間以便稍後使用，這樣的執行效率會比較好，否則稍後當我們在 TDictionary<T,T>物件中加入元素時 TDictionary<T,T>物件就需要動態增加它的元素空間大小，這樣會影響執行效率。

下面的程式碼就是實作統計 glProduct 中 Delphi 和 BCB 的產品個數是多少的程式碼，首先我們先建立 TDictionary<string, Integer>物件 aDic 並且預先在其中建立可儲存 5 個元素大小的空間，接著我們藉由 TEnumerator 物件一一從 glProduct 中最出產品名稱，然後呼叫 TDictionary<T,T>的 ContainsKey 方法來查詢這個產品名稱鍵值是否已經存在 aDic 之中，如果產品名稱鍵值已經存在，那麼就呼叫 TDictionary<T,T>的 AddOrSetValue 方法增加這個產品名稱的個數，否則就在 aDic 中加入這個<產品名稱, 1>這對鍵值和數值，最後呼叫 DisplayProductCount 來顯示統計的數值。

```
procedure TForm17.btn 統計產品總數 Click(Sender: TObject);
```

```
var
```

```
  aDic: TDictionary<string, Integer>;
```

```
  aEnum : TEnumerator<TProduct>;
```

```
begin
```

```
  aDic := TDictionary<string,Integer>.Create(5);
```

```
  try
```

```
    aEnum := glProduct.GetEnumerator;
```

```

while (aEnum.MoveNext) do
begin
    if (aDic.ContainsKey(aEnum.Current.GetCategory)) then
        aDic.AddOrSetValue(aEnum.Current.GetCategory,
aDic.Items[aEnum.Current.GetCategory] + 1)
    else
        aDic.Add(aEnum.Current.GetCategory, 1);
end;

DisplayProductCount(aDic);

finally
    aDic.Free;

end;

end;

```

DisplayProductCount 接受型態為 TDictionary<string, Integer>的參數，然後也是藉由 TDictionary<T, T>的 Enumerator 來一一取出其中的元素來處理，不過 TDictionary<T, T>提供了三種 Enumerator，分別是 TPairEnumerator，TKeyEnumerator 和 TValueEnumerator。從這三個 Enumerator 的名稱我們便可以知道，TPairEnumerator 可以取出<T,T>這對<鍵值, 數值>的元素，而 TKeyEnumerator 則可取出<鍵值>元素，最後的 TValueEnumerator 則是取出<數值>元素。

由於 TPairEnumerator，TKeyEnumerator 和 TValueEnumerator 都是宣告在 TDictionary<T, T>之中的內嵌類別，因此在宣告這三個 Enumerator 類別的變數時

需要在之前加上 `TDictionary<T, T>`。例如下面的程式碼中 `aEnum` 是宣告為 `TPairEnumerator`，但是在這之前我們需要加入 `TDictionary<T,T>`的宣告並且正確的帶入資料型態，因此 `aEnum` 的正確型態宣告是 `TDictionary<string, Integer>.TPairEnumerator`。

`TPairEnumerator` 的使用方法和 `TEnumerator` 是一樣的，但是 `TPairEnumerator` 的 `Current` 特性值的型態是：

```
TPair<TKey,TValue>
```

而 `TPair` 只是鍵值和數值成對的記錄型態而已

```
TPair<TKey,TValue> = record
```

```
  Key: TKey;
```

```
  Value: TValue;
```

```
end;
```

`TPair` 的目的是方便讓開發人員一次可以藉由 `Current` 取出對應的鍵值和數值。

```
procedure TForm17.DisplayProductCount(aDic: TDictionary<string, Integer>);
```

```
var
```

```
  aEnum : TDictionary<string, Integer>.TPairEnumerator;
```

```
begin
```

```
  Self.lb 產品資料.Items.Clear;
```

```
  aEnum := aDic.GetEnumerator;
```

```
  while (aEnum.MoveNext) do
```

```
begin
```

```
Self.lb 產品資料.Items.Add(aEnum.Current.Key + ':' +
```

```
IntToStr(aEnum.Current.Value));
```

```
end;
```

```
end;
```



TCollectionNotifyEvent<T>

Delphi 2009 的泛型容器類別也提供了通知事件，允許開發人員連結此事件以便撰寫程式碼來處理元素在容器類別中異動的情形。在 `Generics.Collections` 程式單元中定義了如下的通知事件：

```
TCollectionNotification = (cnAdded, cnRemoved, cnExtracted);
```

```
TCollectionNotifyEvent<T> = procedure(Sender: TObject; const Item: T;
```

```
Action: TCollectionNotification) of object;
```

開發人員可以藉由實作型態為 `TCollectionNotifyEvent<T>` 的函式，再連結到容器類別物件即可在元素被加入，移除或是取出容器類別被容器類別物件呼叫知會。

例如我們可以使用下面的程式碼連結 `glProduct` 容器類別物件：

```
procedure TForm17.btn 連結通知事件 Click(Sender: TObject);
```

```
begin
```

```
glProduct.OnNotify := ProductNotifier;
```

```
end;
```

而 `ProductNotifier` 就是實作 `TCollectionNotifyEvent<T>` 原型的方法，在這裡

`ProductNotifier` 會在容器類別物件中的元素異動時顯示簡單的訊息：

```
procedure TForm17.ProductNotifier(Sender: TObject; const Item: TProduct;
```

```
Action: TCollectionNotification);
```

```
begin
```

```
case Action of
```

```
cnAdded:
```

```
Self.lb 通知事件.Items.Add('加入 : ' + Item.GetName);
```

```
cnRemoved:
```

```
Self.lb 通知事件.Items.Add('移除 : ' + Item.GetName);
```

```
cnExtracted:
```

```
Self.lb 通知事件.Items.Add('取出 : ' + Item.GetName);
```

end;

end;



深入討論匿名方法

前面討論了匿名方法，匿名方法除了可以和泛型一起使用之外，也可以使用在非泛型的應用程式之中。由於匿名方法是在程式碼中定義函式本身，因此匿名方法也需要一個定義的範圍和呼叫模式，否則匿名方法本身要儲存在那裡呢?因此現在讓我們稍微深入討論一下匿名方法的實作機制，讓各位對於匿名方法有更基礎的瞭解。

Delphi 2009 是使用介面的方式來實作匿名方法，這樣做有許多的好處，主要的原因是使用介面可以進行型態檢查以及以及在 Win32 下缺少資源回收機制的環境中可以比較方便進行資源管理，此外 Delphi(Object Pascal)又是強型程式語言，因此在編譯時期提供較為嚴格的檢查也和 Delphi 程式語言相當的契合。

讓我們看一個範例也許會讓讀者更為瞭解匿名方法的實作方式。假設我們有下面

的程式碼:

```
type
```

```
TFuncOfInt = reference to function(x: Integer): Integer;
```

```
function MakeAdder(addendum: Integer): TFuncOfInt;
```

```
begin
```

```
Result := function(x: Integer)
```

```
begin
```

```
Result := addendum + x;
```

```
end;
```

```
end;
```

```
procedure Use;
```

```
var
```

```
f: TFuncOfInt;
```

```
begin
```

```
f := MakeAdder(20);
```

```
Writeln(f(22)); // prints '42'
```

```
end;
```

```
begin
```

```
Use;
```

```
end.
```

在 `MakeAdder` 中定義了一個匿名方法，而且又把這個匿名方法當成 `MakeAdder` 的回傳參數，而這個匿名方法的原型則是由 `TFuncOfInt` 定義的。OK，那麼 Delphi 2009 是如何實作上面使用匿名方法的程式碼呢？首先 Delphi 2009 的編譯器會把 `TFuncOfInt` 重新定義為如介面，並且在其中定義一個 `Invoke` 方法，類似如下：

```
type
```

```
TFuncOfInt = interface
```

```
function Invoke(x: Integer): Integer;
```

```
end;
```

接著在 `MakeAdder` 中實際定義了匿名方法，因此在 `MakeAdder` 方法中必須定義此匿名方法的程式區塊，因為匿名方法可以擁有它自己的參數，變數，堆疊資源等。因此下面是 `MakeAdder` 可能實作的程式碼：

```
function MakeAdder(addendum: Integer): TFuncOfInt;
```

```
type
```

```
IClosure1 = interface
```

```
function Invoke(x: Integer): Integer;
```

```
end;
```

```
TMakeAdderFrame = class(TInterfacedObject, IClosure1)
```

```
addendum: Integer;
```

```
function IClosure1.Invoke = Closure1;
```

```
function Closure1(x: Integer): Integer;
```

```
end;
```

```
function TMakeAdderFrame.Closure1(x: Integer);
```

```
begin
```

```
Result := Self.addendum + x;
```

```
end;
```

首先 `MakeAdder` 會宣告一個 `IClosure` 介面，其中定義 `Invoke` 方法，接著 `MakeAdder` 會宣告一個內嵌類別 `TMakeAdderFrame`，這個類別將實作 `Closure` 介面並且把匿名方法拉出去成為內嵌類別的方法，接著把匿名方法的父方法的參數宣告為物件變數，如此一來就可以解決巢狀參數/變數的問題。

最後在 `MakeAdder` 方法的實作程式碼中，就建立內嵌類別物件，最後 `MakeAdder` 方法要回傳匿名方法時，這裡是稍微複雜的地方。

```
var
```

```
frameInstance: TMakeAdderFrame;
```

```
begin
```

```
frameInstance := TMakeAdderFrame.Create;
```

```
frameInstance.addendum := addendum;
```

```
Result := reinterpret_cast<TFuncOfInt>(interface_cast<IClosure1>(frameInstance));
```

```
end;
```

首先 `MakeAdder` 必須把 `TMakeAdderFrame` 物件轉變型態為 `IClosure` 介面，這可以使用 `interface_cast` 來調整 `vTable` 的內容指標，接著使用 `reinterpret_cast` 強迫轉變型態為 `TFuncOfInt`，也就是 `MakeAdder` 回傳的型態，如此一來就大功告成了。

```
procedure Use;
```

```
var
```

```
f: TFuncOfInt;
```

```
begin
```

```
f := MakeAdder(20);  
  
Writeln(f.Invoke(22));  
  
end;  
  
begin  
  
Use;  
  
end.
```

OK，我這篇淺談如何使用 Delphi 2009 泛型容器類別的文章也就到此結束了，希望對於想使用 Delphi 2009 泛型實體類別的讀者有一些基本的幫助。

Have Fun!