Dynamic trong C# 4 - Phần 1

Từ khóa dynamic và Dynamic Language Runtime (DLR) là những tính năng mới trong C# 4 và Microsoft .NET Framework 4. Bài viết này cung cấp những cái nhìn khái quát về tính năng dynamic trong C# 4.

**Dynamic là gì?**

C# và Java thường được xem là ngôn ngữ lập trình tĩnh, trong khi Python, Ruby và JavaScript là những ví dụ của ngôn ngữ động.  
Thông thường, ngôn ngữ động không thực hiện kiểm tra kiểu tại thời điểm biên dịch chương trình và chỉ nhận dạng kiểu đối tượng tại thời điểm chạy chương trình. Điều này giúp cho việc viết code đơn giản và nhanh hơn, nhưng lúc đó bạn không nhận được lỗi biên dịch và chỉ đến khi thực hiện kiểm thử rất cẩn thận mới đảm bảo được chương trình bạn được viết chính xác.  
Trong C# 4, từ khóa dynamic đã được bổ sung để cải thiện khả năng tương thích với các ngôn ngữ và nền tảng (framework) động..  
Dưới đây là ví dụ về việc sử dụng từ khóa dynamic:

|  |  |  |
| --- | --- | --- |
| 01 | static void ChangeDynamicDataType() | |
| 02 | { |

|  |  |
| --- | --- |
| 03 | dynamic t = "Hello!"; |
| 04 | Console.WriteLine("t is of type: {0}", t.GetType()); | |

|  |  |
| --- | --- |
| 05 |  |
| 06 | t = false; | |

|  |  |  |
| --- | --- | --- |
| 07 | Console.WriteLine("t is of type: {0}", t.GetType()); | |
| 08 |  |

|  |  |
| --- | --- |
| 09 | t = new System.Collections.Generic.List<int>(); |
| 10 | Console.WriteLine("t is of type: {0}", t.GetType()); | |

|  |  |
| --- | --- |
| 11 | } |
| 12 | </int> | |

Output:

t is of type: System.String

t is of type: System.Boolean

t is of type: System.Collections.Generic.List`1[System.Int32]

Như bạn thấy, nó có thể gắn với các đối tượng có kiểu dữ liệu khác nhau vào biến được khai báo dynamic. Đoạn code được biên dịch bình thường và được định danh tại thời điểm chạy (runtime). Tuy nhiên, đoạn code sau sẽ được biên dịch bình thường, nhưng sẽ xuất hiện ngoại lệ (exception) tại thời điểm chạy:

|  |  |  |
| --- | --- | --- |
| 1 | dynamic d = "Hello!"; | |
| 2 |  |

|  |  |  |
| --- | --- | --- |
| 3 | //Dòng sau sẽ xuất hiện ngoại lệ thời điểm chạy. | |
| 4 | d++; |

Output:

Microsoft.CSharp.RuntimeBinder.RuntimeBinderException was unhandled

Message=Operator '++' cannot be applied to operand of type 'string'

Source=Anonymously Hosted DynamicMethods Assembly

Bởi vì trình biên dịch không thể biết kiểu của đối tượng tại thời điểm chạy và do đó không thể nói với bạn rằng thao tác '++' không được hổ trợ trong trường hợp này.

**Sử dụng từ khóa dynamic, object hay var?**

Vậy sự khác nhau giữa dynamic, object và var như thế nào và khi nào bạn nên dùng chúng?  
Từ khóa object mô tả cho kiểu System.Object, là lớp cha trên cùng trong phân cấp lớp C#, và có thể diễn tả cho bất cứ lớp nào khác. Nó được sử dụng khi không thể định danh được kiểu đối tượng tại thời điểm biên dịch và bạn cần sử dụng lệnh chuyển đổi sang một kiểu định nghĩa trước nếu muốn sử dụng các phương thức hoặc thuộc tính của kiểu đó.

|  |  |  |
| --- | --- | --- |
| 1 | static void UseObjectVarible() | |
| 2 | { |

|  |  |
| --- | --- |
| 3 | // Giả sử có một lớp tên Person. |
| 4 | object o = new Person() { FirstName = "Thanh", LastName = "Ho" }; | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | // Phải đổi kiểu đối tượng thành Person | |

|  |  |
| --- | --- |
| 7 | // để có thể sử dụng các thuộc tính của Person. |
| 8 | Console.WriteLine("Person's first name is {0}", ((Person)o).FirstName); | |

|  |  |
| --- | --- |
| 9 | } |

Từ khóa var được đưa ra từ C# 3.0, được sử dụng đại diện các kiểu đã được định nghĩa sẳn hoặc kiểu nặc danh. Một biến được khai báo var sẽ được định danh kiểu tại thời điểm biên dịch và không thể thay đổi kiểu lúc runtime. Nếu trình biên dịch không thể xác định được kiểu, nó sẽ tạo ra lỗi biên dịch:

|  |  |
| --- | --- |
| 1 | var varExample = 10; |
| 2 | Console.WriteLine(varExample.GetType()); // In ra System.Int32 | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | //Biến varExample là kiểu System.Int32: | |

|  |  |  |
| --- | --- | --- |
| 5 | varExample = varExample + 10; | |
| 6 |  |

|  |  |  |
| --- | --- | --- |
| 7 | //Dòng này sẽ bị lỗi khi biên dịch và chỉ có thể gán giá trị số nguyên cho varExample: | |
| 8 | varExample = "test"; |

Từ khóa dynamic, được giới thiệu trong C# 4, tạo ra các kịch bản khác dựa trên từ khóa object truyền thống nhưng đơn giản hơn trong viết và maintain code. Thực tế, kiểu dynamic sử dụng kiểu System.Object nhưng không như object, nó không đòi hỏi việc đổi kiểu lúc biên dịch, mà chỉ định danh thời điểm chạy:

|  |  |  |
| --- | --- | --- |
| 1 | static void UseDynamicVarible() | |
| 2 | { |

|  |  |
| --- | --- |
| 3 | // Giả sử có một lớp tên Person. |
| 4 | dynamic p = new Person() { FirstName = "Thanh", LastName = "Ho" }; | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | //Có thể sử dụng các thuộc tính của Person mà không cần đổi kiểu. | |

|  |  |  |
| --- | --- | --- |
| 7 | Console.WriteLine("Person's first name is {0}", p.FirstName); | |
| 8 | } |

Qua phần này, bạn có thể hiểu và sử dụng được từ khóa dynamic, một trong những tính năng mới của C# 4, Dynamic Language Runtime (DLR) sẽ được trình bày trong phần tiếp theo.

Dynamic trong C# 4 - Phần 2

Đây là phần tiếp theo của các bài viết về từ khóa dynamic và Dynamic Language Runtime (DLR), đây là tính năng mới trong C# 4 và .NET Framework 4. Trong [phần 1](http://thanhhh.blogspot.com/2011/03/dynamic-trong-c-4-phan-1.html) đã giới thiệu về từ khóa dynamic, nội dung của bài này sẽ mô tả về DLR và hai trong số các trường hợp sử dụng thông thường của nó.

**Dynamic Language Runtime**

Thuật ngữ "dynamic" trong C# thường nhắc đến một trong hai khái niệm: từ khóa dynamic trong C# 4 hoặc là Dynamic Language Runtime (DLR).  
  
DLR phục vụ hai mục đích chính. Một là cho phép liên kết giữa ngôn ngữ động và .NET Framework serves two main goals. Và hai là mang khả năng động vào C# và Visual Basic.  
  
DLR đã được tạo ra dựa trên các bài học trong khi xây dựng IronPython (ironpython.net), là ngôn ngữ động đầu tiên được cài đặt trong .NET Framework. Trong khi làm việc với IronPython, nhóm đã tìm ra phương pháp sử dụng lại cài đặt của mình cho nhiều ngôn ngữ khác, vì vậy họ đã tạo ra một nền tảng cơ bản dùng chung cho các ngôn ngữ động trong .NET.  
  
Sau đó DLR đã được giới thiệu trong .NET Framework 4 để hỗ trợ các tính năng động trong C# và Visual Basic. Nếu bạn chỉ cần từ khóa dynamic trong C# 4, bạn chỉ cần sử .NET Framework và trong hầu hết trường hợp nó sẽ xử lý tất cả các tương tác với DLR trong chính bản thân mình. Nhưng nếu bạn muốn cài đặt hoặc mang một ngôn ngữ động mới vào .NET, bạn cần trợ giúp từ các lớp trợ giúp bên ngoài trong dự án nguồn mở (dlr.codeplex.com) có nhiều tính năng và dịch cho việc cài đặt ngôn ngữ.

**Gọi động các phương thức và truyền các tham số**

Đây không phải là sự kỳ vọng rằng các bạn nên sử dụng động bất kỳ khi nào có thể thay vì các khai báo kiểu tĩnh. Việc kiểm tra lúc biên dịch là một công cụ mạnh mẽ và nhiều lợi ích giúp bạn làm việc tốt hơn. Và hơn nữa, các đối tượng động trong C# không hỗ trợ IntelliSense, mà có thể làm năng suất công việc của bản giảm xuống.  
  
Sau đây là một ví dụ tuyệt vời cho việc sử dụng DLR, đó là việc thay thế các API reflection để triệu gọi đến các phương thức có truyền tham số.  
  
1. Bạn hãy sử dụng Visual Studio, tạo một ứng dụng C# Console Application có tên LateBindingWithDynamic.   
2. Tiếp theo, bạn hãy thêm một Class Library project trong solution hiện tại với tên MathLibrary.  
3. Đặt lại tên Class1.cs của MathLibrary thành SimpleMath.cs và cài đặt lớp như sau:

|  |  |
| --- | --- |
| 1 | public class SimpleMath { |
| 2 | public int Add(int x, int y) | |

|  |  |  |
| --- | --- | --- |
| 3 | { return x + y; } | |
| 4 | } |

4. Tiến hành build MathLibrary và copy MathLibrary.dll vào thư mục bin/Debug của LateBindingWithDynamic (Bạn có thể dùng tính năng ShowAllFiles trong VS và kéo thả MathLibrary.dll từ thư mục bin/Debug của MathLibrary vào thư mục bin/Debug của LateBindingWithDynamic).  
5. Triệu gọi phương thức Add của thư viện MathLibrary trong LateBindingWithDynamic: Có hai phương pháp triệu gọi ở đây là sử dụng các API reflection và sử dụng từ khóa dynamic.  
a/ Phương thức sau sẽ gọi phương thức Add sử dụng các triệu gọi API reflection:

|  |  |  |
| --- | --- | --- |
| 01 | private static void AddWithReflection() | |
| 02 | { |

|  |  |  |
| --- | --- | --- |
| 03 | Assembly asm = Assembly.Load("MathLibrary"); | |
| 04 |  |

|  |  |
| --- | --- |
| 05 | // Get metadata for the SimpleMath type. |
| 06 | Type math = asm.GetType("MathLibrary.SimpleMath"); | |

|  |  |
| --- | --- |
| 07 |  |
| 08 | // Create a SimpleMath on the fly. | |

|  |  |  |
| --- | --- | --- |
| 09 | object obj = Activator.CreateInstance(math); | |
| 10 |  |

|  |  |
| --- | --- |
| 11 | // Get info for Add. |
| 12 | MethodInfo mi = math.GetMethod("Add"); | |

|  |  |
| --- | --- |
| 13 |  |
| 14 | // Invoke method (with parameters). | |

|  |  |
| --- | --- |
| 15 | object[] args = { 10, 70 }; |
| 16 | Console.WriteLine("Result is: {0}", mi.Invoke(obj, args)); | |

|  |  |
| --- | --- |
| 17 |  |
| 18 | } | |

b/ Trường hợp sử dụng từ khóa dynamic:

|  |  |  |
| --- | --- | --- |
| 01 | private static void AddWithDynamic() | |
| 02 | { |

|  |  |  |
| --- | --- | --- |
| 03 | Assembly asm = Assembly.Load("MathLibrary"); | |
| 04 |  |

|  |  |
| --- | --- |
| 05 | // Get metadata for the SimpleMath type. |
| 06 | Type math = asm.GetType("MathLibrary.SimpleMath"); | |

|  |  |
| --- | --- |
| 07 |  |
| 08 | // Create a SimpleMath on the fly. | |

|  |  |
| --- | --- |
| 09 | dynamic obj = Activator.CreateInstance(math); |
| 10 | Console.WriteLine("Result is: {0}", obj.Add(10, 70)); | |

|  |  |
| --- | --- |
| 11 | } |

Trong hai trường hợp trên, bạn sẽ thấy được lợi điểm của việc sử dụng DLR, với việc sử dụng DLR giúp cho việc gọi phương thức và truyền các tham số trong đơn giản và dể hiểu hơn so với cách sử dụng API Reflection.

**Khai báo phương thức và thuộc tính động lúc thực thi (Dynamic Method Bags)**

Một ví dụ khác mà của dynamic có thể làm được là tạo ra nơi chứa các phương thức động (method bag), tức là các đối tượng có thể thêm và xoá các thuộc tính và phương thức lúc thực thi, và sau đó triệu gọi các phương thức này thông qua triệu gọi động (dynamic dispatch) trong C#.  
  
Trong .NET Framework 4 có một namespace mới: System.Dynamic. Namespace này thực ra là một phần của DLR và được định nghĩa trong gói (assembly) System.Core. Các lớp System.Dynamic.ExpandoObject và System.Expando.DynamicObject cùng với từ khóa dynamic có thể giúp bạn tạo ra các cấu trúc (structure) động và hệ thống thứ bậc (hierarchic) động theo một cách đơn giản và dễ hiểu.  
Sau đây là ví dụ diễn tả cách bạn có thể thêm một thuộc tính và phương thức bằng cách sử dụng lớp ExpandoObject như ví dụ dưới đây:

|  |  |
| --- | --- |
| 1 | dynamic expando = new ExpandoObject(); |
| 2 | expando.SampleProperty = "Đây là thuộc tính được thêm lúc thực thi"; | |

|  |  |
| --- | --- |
| 3 | expando.SampleMethod = (Action)( |
| 4 | () => Console.WriteLine(expando.SampleProperty)); | |

|  |  |
| --- | --- |
| 5 | expando.SampleMethod(); |

Dưới đây là một ví dụ về việc đọc dữ liệu từ XML. Trong ví dụ này đã sử dụng class ExpandoObject để lưu trữ dữ liệu nhận được từ XML, bạn có thể truy cập dữ liệu này thông qua các thuộc tính tự động được thêm vào lúc đọc dữ liệu và thực hiện thêm phương thức GetFullName để mô tả khả năng đưa thêm phương thức trong lúc chạy chương trình.  
Dữ liệu trong XML có cấu trúc như sau:

|  |  |
| --- | --- |
| 01 | <persons> |
| 02 | <person> | |

|  |  |  |
| --- | --- | --- |
| 03 | <firstname> Thanh </firstname> | |
| 04 | <lastname> Ho Hai </lastname> |

|  |  |  |
| --- | --- | --- |
| 05 | </person> | |
| 06 | <person> |

|  |  |  |
| --- | --- | --- |
| 07 | <firstname> Phuong Linh </firstname> | |
| 08 | <lastname> Ho Hoang </lastname> |

|  |  |  |
| --- | --- | --- |
| 09 | </person> | |
| 10 | </persons> |

Và dưới đây là toàn bộ đoạn code :

|  |  |  |
| --- | --- | --- |
| 01 | public static IList<dynamic> GetExpandoFromXml(String file) | |
| 02 | { |

|  |  |  |
| --- | --- | --- |
| 03 | var persons = new List<dynamic>(); | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | var doc = XDocument.Load(file); | |
| 06 | var nodes = |

|  |  |  |
| --- | --- | --- |
| 07 | from node in doc.Root.Descendants("Person") | |
| 08 | select node; |

|  |  |
| --- | --- |
| 09 |  |
| 10 | foreach (var n in nodes) { | |

|  |  |  |
| --- | --- | --- |
| 11 | dynamic person = new ExpandoObject(); | |
| 12 |  |

|  |  |  |
| --- | --- | --- |
| 13 | //Đọc các giá trị tương ứng với FirstName và LastName của mỗi Person | |
| 14 | foreach (var child in n.Descendants()) { |

|  |  |
| --- | --- |
| 15 | // Bởi vì ExpandoObject được cài đặt interface IDictionary<string, object> |
| 16 | // và xem như mỗi phần tử của Dictionary gồm có thuộc tính và giá trị |

|  |  |  |
| --- | --- | --- |
| 17 | var p = person as IDictionary<string, object>; | |
| 18 |  |

|  |  |  |
| --- | --- | --- |
| 19 | //person được thêm 2 thuộc tính là FirstName và LastName | |
| 20 | p[child.Name.ToString()] = child.Value.Trim(); |

|  |  |  |
| --- | --- | --- |
| 21 | } | |
| 22 |  |

|  |  |  |
| --- | --- | --- |
| 23 | //Cài đặt phương thức lấy đầy đủ họ tên cho person | |
| 24 | person.GetFullName = (Func<string>)(() => |

|  |  |
| --- | --- |
| 25 | { |
| 26 | return string.Format("{0}, {1}", person.FirstName, person.LastName); | |

|  |  |  |
| --- | --- | --- |
| 27 | }); | |
| 28 |  |

|  |  |  |
| --- | --- | --- |
| 29 | persons.Add(person); | |
| 30 | } |

|  |  |
| --- | --- |
| 31 |  |
| 32 | return persons; | |

|  |  |
| --- | --- |
| 33 | } |

Và bạn có thể truy cập dữ liệu đọc được ở trên bằng các thuộc tính và phương thức đã được thêm vào cho mỗi đối tượng ExpandoObject lúc thực thi như sau:

|  |  |  |
| --- | --- | --- |
| 1 | IList<dynamic> persons = GetExpandoFromXml("Persons.xml"); | |
| 2 | foreach (var p in persons) |

|  |  |
| --- | --- |
| 3 | { |
| 4 | Console.WriteLine(p.GetFullName()); | |

|  |  |
| --- | --- |
| 5 | } |

Trên đây là hai trong số những kịch bản mà bạn có thể sử dụng DLR trong C# 4, trong những bài tiếp theo, tôi sẽ mô tả về các tính năng cao cấp khác của DLR.

Dynamic trong C# 4 - Phần 3

Trong phần này tôi sẽ tiếp tục nói đến việc tận dụng khả năng của DLR trong việc mở rộng khả năng của một lớp sẳn có.

**Mở rộng lớp đang tồn tại bằng DynamicObject**

Với việc sử dụng lớp DynamicObject, bạn có thể cung cấp cú pháp tốt hơn cho mã nguồn của bạn hoặc mở rộng khả năng của các thư viện đang tồn tại. Bạn có thể tận dụng các phương thức sẳn có của một class cùng với khả năng thêm các thuộc tính và phương thức một cách tự động tại thời điểm runtime mà không phải viết toàn bộ code này một cách rõ ràng.   
  
Dưới đây là một ví dụ tận dụng khả năng của DynamicObject kết hợp với ADO.NET trong việc lấy dữ liệu bằng câu lệnh SELECT và lưu từng cột giá trị nhận được của mỗi dòng dữ liệu trong các đối tượng IDataRecord.

Để chạy được ví dụ dưới đây, bạn cần sử dụng AdventureWorks2008 Database sample có thể download từ địa chỉ:[Microsoft SQL Server Product Samples: Database](http://msftdbprodsamples.codeplex.com/" \t "_blank).   
Thông thường, nếu bạn dùng ADO.NET để truy vấn và hiển thị dữ liệu như ví dụ sau đây:

|  |  |  |
| --- | --- | --- |
| 01 | private static void ReadOrderData(string connectionString) | |
| 02 | { |

|  |  |  |
| --- | --- | --- |
| 03 | string queryString = "SELECT [SalesOrderID], [CustomerID] FROM [Sales].[SalesOrderHeader];"; | |
| 04 | using (SqlConnection connection = new SqlConnection(connectionString)) |

|  |  |
| --- | --- |
| 05 | { |
| 06 | SqlCommand command = new SqlCommand(queryString, connection); | |

|  |  |
| --- | --- |
| 07 | connection.Open(); |
| 08 | SqlDataReader reader = command.ExecuteReader(); | |

|  |  |  |
| --- | --- | --- |
| 09 | try | |
| 10 | { |

|  |  |  |
| --- | --- | --- |
| 11 | while (reader.Read()) | |
| 12 | { |

|  |  |  |
| --- | --- | --- |
| 13 | Console.WriteLine(String.Format("{0}, {1}", | |
| 14 | reader[0], reader[1])); |

|  |  |  |
| --- | --- | --- |
| 15 | } | |
| 16 | } |

|  |  |  |
| --- | --- | --- |
| 17 | finally | |
| 18 | { |

|  |  |  |
| --- | --- | --- |
| 19 | // Always call Close when done reading. | |
| 20 | reader.Close(); |

|  |  |  |
| --- | --- | --- |
| 21 | } | |
| 22 | } |

|  |  |
| --- | --- |
| 23 | } |

Trong ví dụ này, bạn thấy rằng để truy cập vào từng cột giá trị trong từng dòng đọc được, ta sẽ phải dùng reader[0] (hoặc reader["SaleOrderID"]. Với việc mở rộng IDataRecord bằng DynamicObject, bạn có thể sử dụng reader.SaleOrderID, trong ví dụ dưới đây định nghĩa một lớp có tên DataRecordDynamic:

|  |  |  |
| --- | --- | --- |
| 01 | public class DataRecordDynamic : DynamicObject | |
| 02 | { |

|  |  |  |
| --- | --- | --- |
| 03 | private IDataRecord m\_dataRecord; | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | public DataRecordDynamic(IDataRecord dataRecord) | |
| 06 | { |

|  |  |  |
| --- | --- | --- |
| 07 | m\_dataRecord = dataRecord; | |
| 08 | } |

|  |  |
| --- | --- |
| 09 |  |
| 10 | public override bool TryGetMember( | |

|  |  |  |
| --- | --- | --- |
| 11 | GetMemberBinder binder, out object result) | |
| 12 | { |

|  |  |  |
| --- | --- | --- |
| 13 | result = m\_dataRecord[binder.Name]; | |
| 14 | return (result != null); |

|  |  |  |
| --- | --- | --- |
| 15 | } | |
| 16 | } |

Lớp này đã thực hiện định nghĩa lại phương thức xác định thao tác Get của một thuộc tính động, nó thực hiện xác định giá trị bằng cách sử dụng tên của thuộc tính xem như tên cột và lấy dữ liệu cột tương ứng với tên cột đó trong row dữ liệu nhận được (= Item[string]).   
Và dưới đây là ví dụ ban đầu đã được viết lại sử dụng lớp DataRecordDynamic, bạn sẽ thấy rằng bây giờ sẽ dùng row.OrderID và row.CustomerID.

|  |  |  |
| --- | --- | --- |
| 01 | private static IList<dynamic> ExecuteOrderDataReader(string connectionString) | |
| 02 | { |

|  |  |  |
| --- | --- | --- |
| 03 | var orderDataRows = new List<dynamic>(); | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | string queryString = "SELECT [SalesOrderID], [CustomerID] FROM [Sales].[SalesOrderHeader];"; | |
| 06 | using (SqlConnection connection = new SqlConnection(connectionString)) |

|  |  |
| --- | --- |
| 07 | { |
| 08 | SqlCommand command = new SqlCommand(queryString, connection); | |

|  |  |
| --- | --- |
| 09 | connection.Open(); |
| 10 | SqlDataReader reader = command.ExecuteReader(); | |

|  |  |  |
| --- | --- | --- |
| 11 | try | |
| 12 | { |

|  |  |  |
| --- | --- | --- |
| 13 | foreach (IDataRecord record in reader) | |
| 14 | { |

|  |  |  |
| --- | --- | --- |
| 15 | orderDataRows.Add(new DataRecordDynamic(record)); | |
| 16 | } |

|  |  |
| --- | --- |
| 17 |  |
| 18 | return orderDataRows; | |

|  |  |
| --- | --- |
| 19 | } |
| 20 | finally | |

|  |  |
| --- | --- |
| 21 | { |
| 22 | // Always call Close when done reading. | |

|  |  |  |
| --- | --- | --- |
| 23 | reader.Close(); | |
| 24 | } |

|  |  |  |
| --- | --- | --- |
| 25 | } | |
| 26 | } |

|  |  |
| --- | --- |
| 27 |  |
| 28 | private static void ReadOrderData2(string connectionString) | |

|  |  |
| --- | --- |
| 29 | { |
| 30 | var orderDataRows = ExecuteOrderDataReader(connectionString); | |

|  |  |
| --- | --- |
| 31 |  |
| 32 | foreach (var row in orderDataRows) | |

|  |  |
| --- | --- |
| 33 | { |
| 34 | Console.WriteLine(String.Format("{0}, {1}", row.SalesOrderID, row.CustomerID)); | |

|  |  |  |
| --- | --- | --- |
| 35 | } | |
| 36 | } |

Qua ví dụ trên, bạn cũng có thể thấy được sự khác nhau giữa ExpandoObject và DynamicObject, đó là DynamicObject còn cho phép bạn tận dụng được các phương thức, thao tác và thuộc tính sẳn có của một class.