« LINQ 快速上手 »

多奇數位創意有限公司

技術總監 黃保翕 (Will 保哥)

部落格：http://blog.miniasp.com/

Introducing LINQ

LINQ 簡介

LINQ is one of Microsoft's most exciting, powerful new development technologies.

from MSDN (2009)

什麼是 LINQ

- 全名：Language INtegrated Query
- LINQ 是 C# 3.0 與 VB 9 的語言特性
  - 從 Visual Studio 2008 開始支援
  - 可將強大的查詢功能擴充至 C# 和 Visual Basic 語言中
- LINQ 推出標準且容易學習的資料查詢與更新模式
  - 這項技術可擴充為支援幾乎所有類型的資料存放區
- LINQ 提供者組件 (.NET Framework 3.5+)
  - 預設可搭配以下資料來源進行資料查詢：.NET Framework 集合物件、SQL Server 資料庫、ADO.NET 資料集 和 XML 文件 等等。

LINQ to Objects

- LINQ to Object 可以針對所有實作 `IEnumerable` 或 `IEnumerable<T>`泛型介面的集合物件進行查詢或更新。
  - Array, ArrayList, IList, …
  - ILists<T>, Collection<T>, IQueryable<T>, …

- 範例

```
int[] source = new int[] { 0, -5, 12, -54, 5, -67, 3, 6 };

// Query from source to results.

foreach (int item in results)
{
    Console.WriteLine(item);
}
```

範例：對資料進行篩選（C# 2.0）

```
int[] source = new int[] { 0, -5, 12, -54, 5, -67, 3, 6 };

List<int> results = new List<int>();
foreach (int integer in source)
{
    if (integer > 0)
    {
        results.Add(integer);
    }
}

Comparison<int> comparison = delegate(int a, int b)
                                     {
                                         return b - a;
                                     };
results.Sort(comparison);

foreach (int item in results)
{
    Console.WriteLine(item);
}
```

範例：對資料進行篩選（C# 3.0）

```
int[] source = new int[] { 0, -5, 12, -54, 5, -67, 3, 6 };

var results = from integer in source
              where integer > 0
              orderby integer descending
              select integer;

foreach (int item in results)
{
    Console.WriteLine(item);
}
```

LINQ to SQL

- LINQ to SQL 可以針對 SQL Server 資料庫中的資料進行查詢或更新。

- 範例
  - 以北風資料庫為例
    http://go.microsoft.com/fwlink/?linkid=30196

範例：對資料進行篩選（C# 2.0）

```
Dictionary<string, decimal> results = new Dictionary<string, decimal>();

using (SqlConnection connection = new SqlConnection(
    @"Data Source=localhost;Initial Catalog=Northwind;Integrated Security=True"))
using (SqlCommand command = new SqlCommand(
    @"SELECT Products.ProductName, Products.UnitPrice FROM
    (
        Products
        LEFT JOIN Categories
        ON Products.CategoryID = Categories.CategoryID
    )
    WHERE Categories.CategoryName = @CategoryName", connection))
{
    command.Parameters.AddWithValue("@CategoryName", "Beverages");
    connection.Open();
    using (SqlDataReader reader = command.ExecuteReader())
    {
        while (reader.Read())
        {
            string productName = (string)reader["ProductName"];
            decimal unitPrice = (decimal)reader["UnitPrice"];
            results.Add(productName, unitPrice);
        }
    }
}

foreach (KeyValuePair<string, decimal> item in results)
{
    Console.WriteLine("{0}: {1}", item.Key, item.Value.ToString(CultureInfo.InvariantCulture));
}
```

範例：對資料進行篩選（C# 3.0）

```
using (NorthwindDataContext database = new NorthwindDataContext())
{
    var results = from product in database.Products
                  where product.Category.CategoryName == "Beverages"
                  select new
                            {
                                product.ProductName,
                                product.UnitPrice
                            };
    foreach (var item in results)
    {
        Console.WriteLine("{0}: {1}", item.ProductName, item.UnitPrice.ToString());
    }
}
```

LINQ to XML

- LINQ to XML 可以針對記憶體中的 XML 資料進行查詢或更新。

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
  <channel>
    <title>Dixin's Blog</title>
    <link>http://weblogs.asp.net/dixin/default.aspx</link>
    <!-- ... -->
    <item>
      <title>Title</title>
      <pubDate>Thu, 26 Nov 2009 05:01:00 GMT</pubDate>
      <!-- ... -->
      <category>C#</category>
      <category>.NET</category>
      <category>LINQ</category>
    </item>
    <item />
    <item />
    <item />
    <!-- ... -->
  </channel>
</rss>
```

範例：對資料進行篩選（C# 3.0）

```
XDocument xml = XDocument.Load(@"http://weblogs.asp.net/dixin/rss.aspx");
var results = from item in xml.Root.Element("channel").Elements("item")
              where item.Elements("category").Any(category => category.Value == "C#")
              orderby DateTime.Parse(item.Element("pubDate").Value)
              select item.Element("title").Value;
foreach (var item in results)
{
    Console.WriteLine(item);
}
```

LINQ to Wikipedia

- LINQ to Wikipedia 是一個自訂的 LINQ 查詢提供者，透過 Mediawiki API 進行實作，提供一個 LINQ 查詢介面，可以針對 Wikipedia 網站的內容進行查詢。

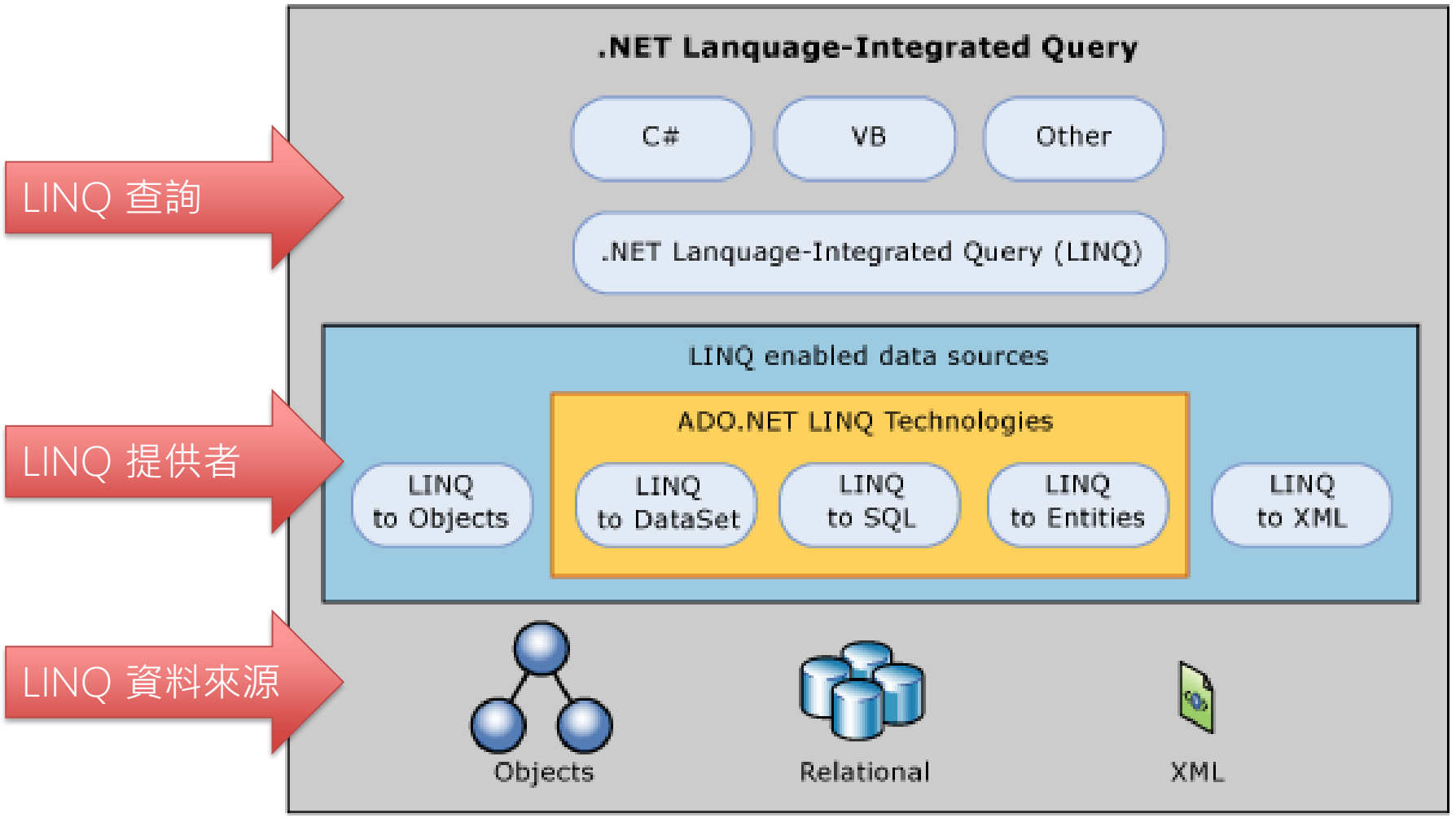
```
WikipediaContext datacontext = new WikipediaContext();

var results = from wikipedia in datacontext.OpenSearch
              where wikipedia.Keyword == "LINQ"
              select wikipedia;
foreach (var item in results)
{
    Console.WriteLine("{0} - {1}", item.Text, item.Url);
}
```

LINQ 的重要特性

- 獨立於資料來源
  - 使用統一的查詢語法(LINQ)查詢資料
- 強型別 & 編譯時期檢查
  - T-SQL 是個語法，以字串表示，無法進行編譯
  - LINQ 是個語法，以 C# 表示 (強型別)，可編譯
- 延遲執行 / 延遲載入
  - 建立完查詢時，並沒有真正執行查詢命令
  - 透過 ToList() 或透過 foreach 才會真的開始執行
- 比「查詢」多更多！
  - 可結合函式語言特性、平行處理、reactive programming
  - 不僅僅改變了處裡資料的方法，更改變了思考問題的方式

LINQ 架構


LINQ 查詢
LINQ 提供者
LINQ 資料來源
.NET Lanquage-Integrated Query
C#
VB
Other
.NET Lanquage-Integrated Query (LINQ)
LINQ enabled data sources
ADO.NET LINQ Technologies
LINQ to Objects
LINQ to DataSet
LINQ to SQL
LINQ to Entities
LINQ to XML
Objects
Relational
XML

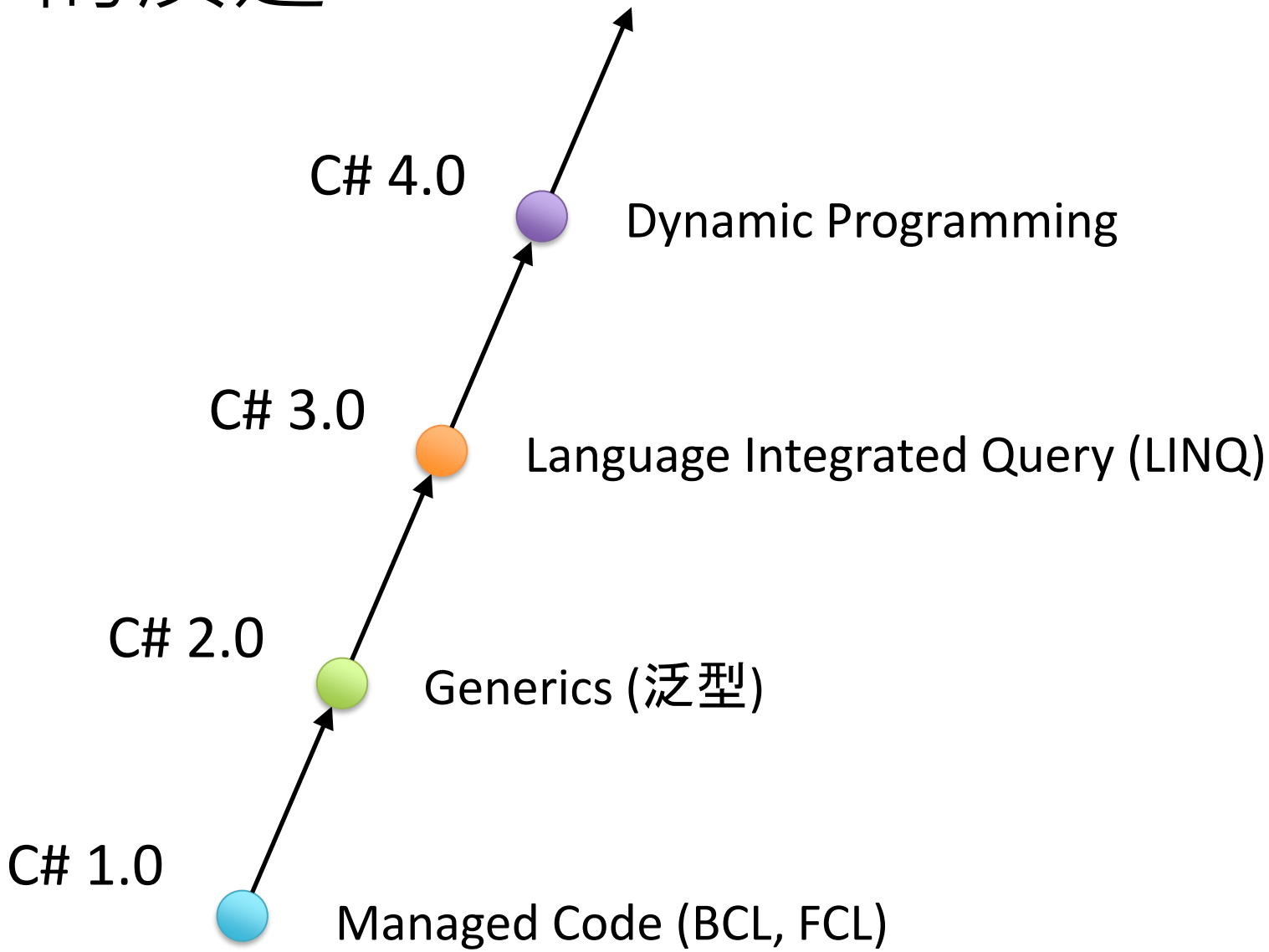
.NET 3.5 FCL 內建的 LINQ 提供者

- LINQ to Objects
  - 提供 .NET 物件 在 LINQ 查詢相關的類別庫實作
- LINQ to SQL
  - 提供 SQL Server 在 LINQ 查詢相關的類別庫實作
- LINQ to XML
  - 提供 XML 物件 在 LINQ 查詢相關的類別庫實作
- LINQ to DataSets
  - 提供 DataSet 物件 在 LINQ 查詢相關的類別庫實作
- LINQ to Entities
  - 提供 關聯式資料庫 在 LINQ 查詢相關的類別庫實作

支援 LINQ 的程式語言

- 官方支援
  - C# 3.0+
  - VB 9.0+
- 語言特性
  - 查詢語法 (from … in … where … select …)
  - 匿名型別 與 隱含類型區域變數 (var)
  - 擴充方法
  - Lambda 表達式

C# 的演進



PDC2008: The Future of C#

C# 2.0

- C# 2.0
  - 部分類別 (Partial class)
  - 泛型 (Generics) → List<T>
  - 靜態類別 (Static classes) → static
  - 產生器功能 (Generator functionality) → yield
    - IEnumerable<T>
  - 匿名委派 (Anonymous delegates) → delegate
  - 委派的協變與逆變 (Delegate covariance and contravariance)
  - 屬性 (Property) 的 get / set 存取子可不同存取性
    (The accessibility of property accessors can be set independently)
  - 允許空值的實值型別 (Nullable types) → int?
  - null 聯合運算子 (Null-coalescing operator) → ??

C# 3.0

- C# 3.0
  - LINQ 查詢運算式 (language-integrated query)
  - 物件和集合初始設定式 (Object and Collection initializers)
  - 匿名型別 (Anonymous types)
  - 隱含類型區域變數 (Local variable type inference)
  - Lambda 運算式 (Lambda expressions)
  - 運算式樹狀架構 (Expression trees)
  - 自動實作的屬性 (Automatic properties)
  - 擴充方法 (Extension methods)
  - 部分類別和方法 (Partial class and methods)

C# 4.0

- C# 4.0
  - 動態型別 (Dynamic member lookup)
  - 泛型中的共變數和反變數
    (Covariant and contravariant generic type parameters)
  - 選擇性參數與具名參數
    (Optional parameters and named arguments)
  - 使用 COM Interop 物件時不用再加上 ref 關鍵字
    (Optional ref keyword when using COM)
  - 在 COM Interop 程式設計中使用索引的屬性
    (Indexed properties)

泛型與 LINQ 命名空間

- System.Collections.Generic
  - 包含會定義泛型集合的介面和類別，可讓使用者建立強型別集合，提供比起非泛型強型別集合更佳的型別安全和效能。
- System.Linq
  - 提供能夠支援查詢使用 LINQ 的類別和介面。
- System.Linq.Expressions
  - 包含類別、介面和列舉，可使用運算式樹狀結構格式將這些語言層級程式碼運算式表示為物件。

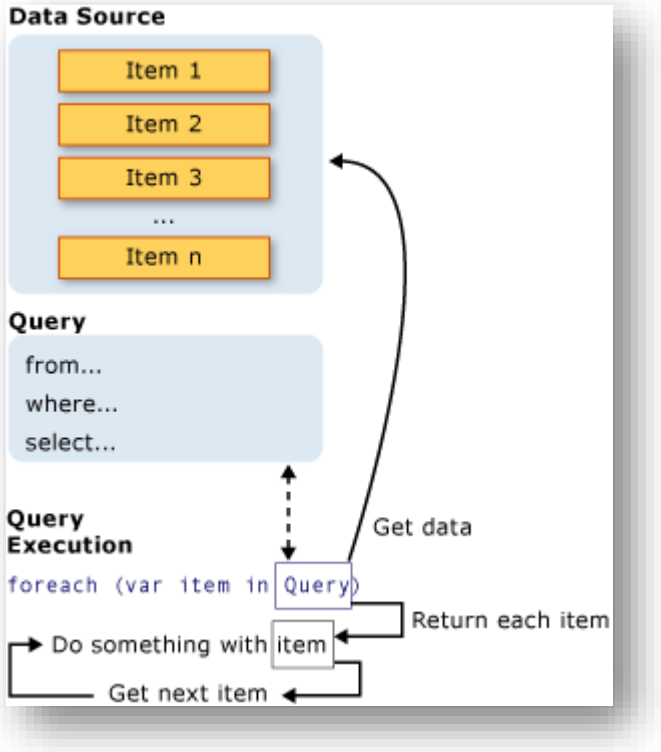
※ LINQ 全名是 Language-Integrated Query

Getting Started with LINQ in C#

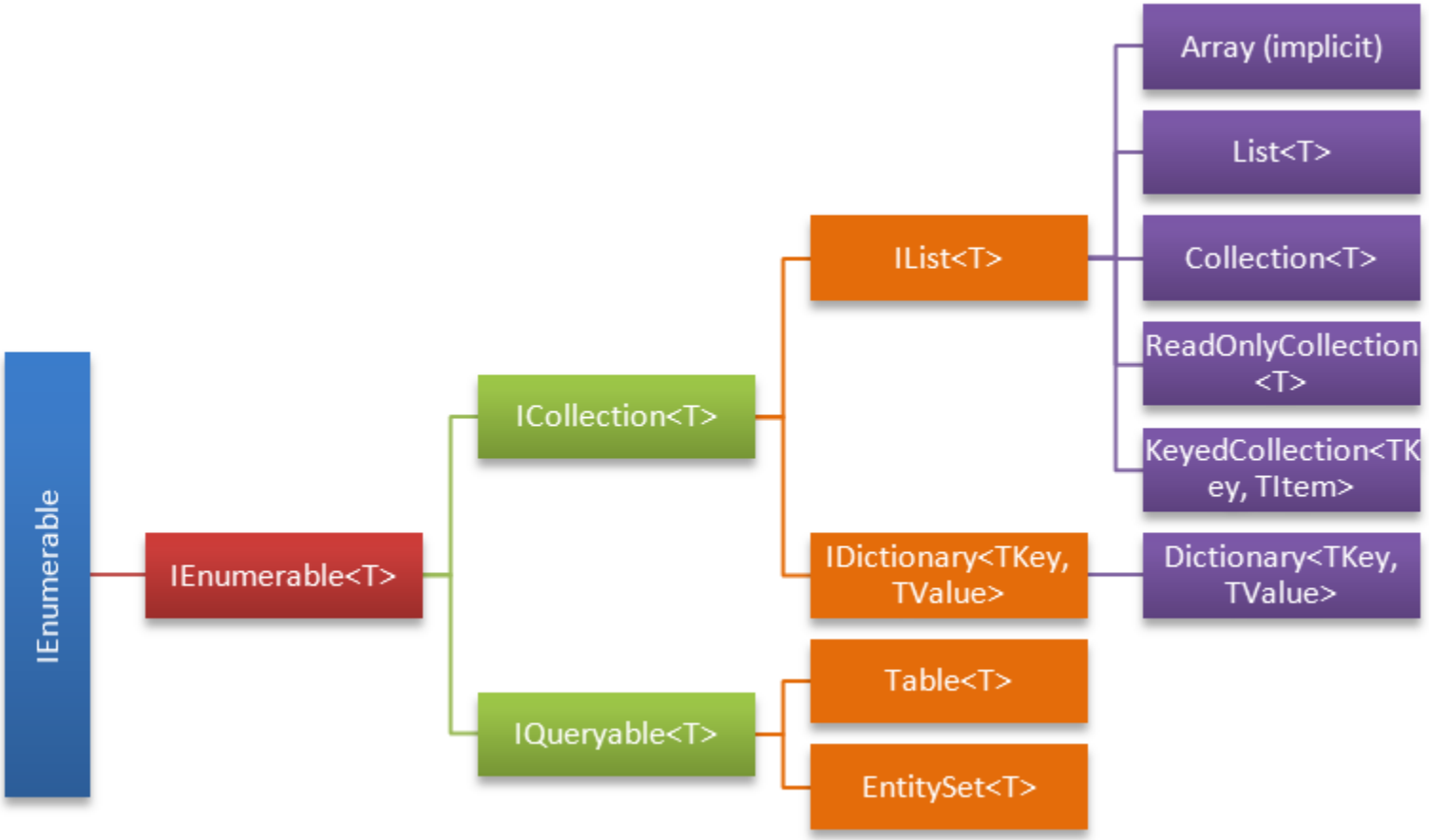
開始使用 LINQ 查詢

關於 LINQ 查詢

- 所有的 LINQ 查詢 皆包含以下三步驟
  - 取得資料來源
  - 建立查詢物件
    (LINQ 查詢運算式)
  - 執行查詢



1. 取得資料來源


IEnumerable
IEnumerable<T>
ICollection<T>
IQueryable<T>
IList<T>
IDictionary<TKey, TValue>
Array (implicit)
List<T>
Collection<T>
ReadOnlyCollection <T>
KeyedCollection<TKey, TItem>
Dictionary<TKey, TValue>
Table<T>
EntitySet<T>

2. 建立查詢（LINQ 查詢運算式）

- Enumerable 方法
  - Restriction: Where, OfType
  - Projection: Select, SelectMany
  - Ordering: OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse
  - Join: Join, GroupJoin
  - Grouping: GroupBy
  - Set: Zip, Distinct, Union, Intersect, Except
  - Aggregation: Aggregate, Count, LongCount, Sum, Min, Max, Average
  - Partitioning: Take, TakeWhile, Skip, SkipWhile
  - Cancatening: Concat
  - Conversion:ToArray, ToList, ToDictionary, ToLookup, Cast
  - Equality: SequenceEqual
  - Elements: First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault, ElementAt, ElementAtOrDefault, DefaultIfEmpty
  - Generation: Range, Repeat, Empty
  - Qualifiers: Any, All, Contains

3. 執行查詢

- IEnumerable and IEnumerable<T>
- 免除所有基礎的原理，用 `foreach` 就對了！
- 執行 foreach 時同時逐步取得集合物件的內容
  (並非一次將資料抓完，而是抓一筆算一筆)
- 延遲載入 v.s. 預先載入

LINQ 的兩種寫法

- **編程式** (Imperative) ➔ λ (**讀音**: lambda)

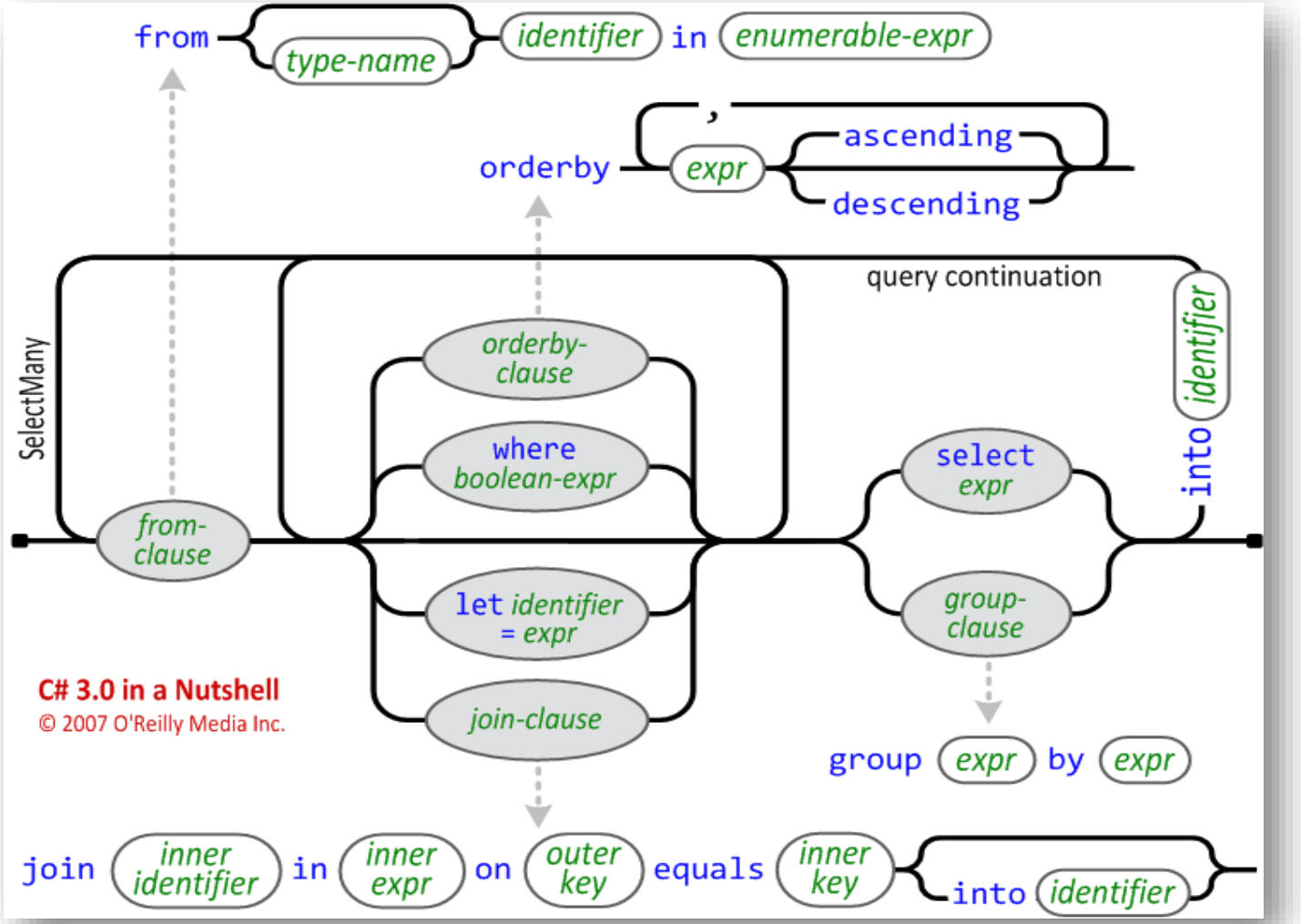
```
IEnumerable<int> results = source
    .Where(integer => integer > 0)
    .OrderByDescending(integer => integer);
```

※ 透過**擴充方法**對**資料來源**進行操作。

- **宣告式** (Declarative) ➔ LINQ (**讀音**: Link)

```
var results = from integer in source
              where integer > 0
              orderby integer descending
              select integer;
```

LINQ 查詢語法一覽



http://www.albahari.com/nutshell/linqsyntax.aspx

Practices

練習各式 LINQ 查詢

基本資料過濾

- 取得資料來源

```
int[] source = new int[] { 0, -5, 12, -54, 5, -67, 3, 6 };
string[] names = { "Tom", "Dick", "Harry", "Mary", "Jay" };
```

- 建立查詢
  - 請在 LINQPad 中撰寫程式碼 (可使用 LINQ 或 λ 語法)
    - 篩選出 source 陣列中小於 0 的數字
    - 篩選出 names 陣列中字串長度大於 3 個字元的字串

```
var data = …;
```

- 執行查詢

```
data.Dump();
```

流體風格(Fluent Syntax)

- 取得資料來源

```
string[] names = { "Tom", "Dick", "Harry", "Mary", "Jay" };
```

- 建立查詢

```
IEnumerable<string> query = names
    .Where   (n => n.Contains ("a"))
    .OrderBy (n => n.Length)
    .Select  (n => n.ToUpper());
```

- 執行查詢

```
query.Dump();
```

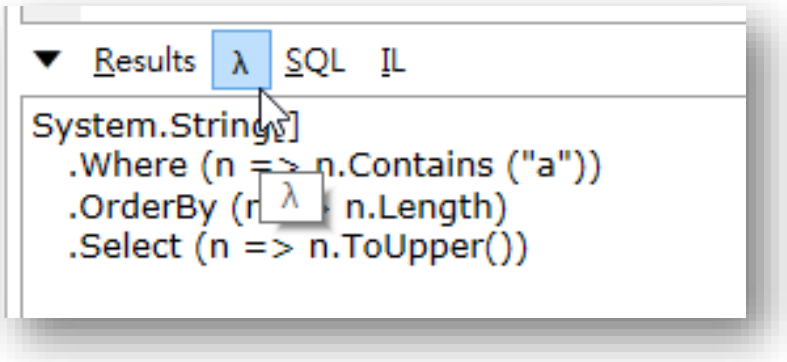
透過 LINQPad 翻譯 λ 程式寫法

- **來源資料 (IEnumerable<T>)**

```
var names = new[] { "Tom", "Dick", "Harry", "Mary",
"Jay" };
```

- **來源資料 (IQueryable<T>)**

```
var names = new[] { "Tom", "Dick", "Harry", "Mary",
"Jay" }.AsQueryable();
```



混搭 LINQ 與 λ 語法

- `(from n in names where n.Contains ("a") select n).Count()`
- `(from n in names orderby n select n).First()`
- `names.Where (n => n.Contains ("a")).Count()`
- `names.OrderBy (n => n).First()`

延遲載入 v.s. 預先載入

- 標示**粗體**的都是「**預先載入**」的方法，加上 **[ ]** 的都是最佳化過的方法
  - Restriction: Where, OfType
  - Projection: Select, SelectMany
  - Ordering: OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse
  - Join: Join, GroupJoin
  - Grouping: GroupBy
  - Set: Zip, **Distinct**, Union, Intersect, Except
  - Aggregation: **Aggregate**, **[Count]**, **LongCount**, **Sum**, **Min**, **Max**, **Average**
  - Partitioning: Take, TakeWhile, Skip, SkipWhile
  - Cancatening: Concat
  - Conversion: **ToArray**, **ToList**, **ToDictionary**, **ToLookup**, [Cast]
  - Equality: **SequenceEqual**
  - Elements: **[First]**, **[FirstOrDefault]**, **[Last]**, **[LastOrDefault]**, **[Single]**, **[SingleOrDefault]**, **[ElementAt]**, **[ElementAtOrDefault]**, **DefaultIfEmpty**
  - Generation: Range, Repeat, **Empty**
  - Qualifiers: **Any**, **All**, **[Contains]**

延遲載入範例

- 建立清單
  - `var numbers = new List<int>();`
  - `numbers.Add (1);`
- 建立查詢
  - `IEnumerable<int> query = numbers.Select (n => n * 10);`
- 增加清單項目
  - `numbers.Add (2);`
- 執行查詢
  - `query.Dump();`

※ 請回答 query 輸出的結果為何？

預先載入範例

- 建立清單
  - `var numbers = new List<int>() { 1, 2 };`
- 建立查詢並且預先載入資料
  - `List<int> timesTen = numbers.Select (n => n * 10).ToList();`
- 清空清單項目
  - `numbers.Clear();`
- 執行查詢
  - `timesTen.Dump();`

※ 請回答 timesTen 輸出的結果為何？

閉包範例

```
int[] numbers = { 1, 2 };

int factor = 10;
IEnumerable<int> query = numbers.Select (n => n * factor);

factor = 20;

query.Dump ();
```

※ 請回答 query 輸出的結果為何？

子查詢範例

```
names.Where(n => n.Length == names.OrderBy (n2 => n2.Length)
     .Select (n2 => n2.Length).First())
     .Dump();

var query =
     (
     from   n in names
     where  n.Length == (from n2 in names orderby n2.Length select n2.Length).First()
     select n
     ).Dump();
```

※ 少用子查詢，因為效能不好！

相關連結

- LINQ via C#
- LINQ (Language-Integrated Query) [中文]
- LINQ 查詢運算式 (C# 程式設計手冊)

聯絡資訊

- The Will Will Web
  記載著 Will 在網路世界的學習心得與技術分享
  - http://blog.miniasp.com/
- Will 保哥的技術交流中心 (臉書粉絲專頁)
  - http://www.facebook.com/will.fans
- Will 保哥的推特
  - https://twitter.com/Will_Huang