

C# 7.0

new features

Bill Chung

關於我

- Bill Chung
- 海角點部落
- 專長：說故事

大綱

- 新增的數字表示法
- out 引數的新用法
- ValueTuple
- Deconstruct
- ref local and return
- ValueTask
- pattern matching
- local functions
- expression-bodied
- throw expression

新增的數字表示法

```
//二進位數字表示法
int i = 0b0010;
Console.WriteLine(i);

// 使用底線讓數字更清晰
int j = 100_000;
Console.WriteLine(j);

int k = 1_00;
Console.WriteLine(k);

int l = 0b0010_0000;
Console.WriteLine(l);

double d = 3.141_592_653;
Console.WriteLine(d);
```

out 引數的新用法

out 只有一點小改變

- 可以直接在呼叫時宣告 out 引數

```
string s="123";



// 以前你要這樣寫
int i;
int.TryParse(s,out i);
Console.WriteLine(i);
// 現在你可以直接這樣寫
int.TryParse(s, out int j);
Console.WriteLine(j);

// ref 不能比照 out 使用
//Test(ref int y = 10);
```


ValueTuple

ValueTuple

- ValueTuple 以泛型結構形式定義
- 可以大幅簡化過去使用 Tuple<> 的麻煩
- 以 nuget 套件的形式加入參考

	System.ValueTuple 依 Microsoft, 176K 項下載 Provides the System.ValueTuple structs, which implement the underlying types for C# 7 tuples.	v4.3.0
	ValueTupleBridge 依 Nobuyuki Iwanaga, 20 項下載 Backporting of System.ValueTuple for .NET 3.5.	v0.1.0

描述

Provides the System.ValueTuple structs, which implement the underlying types for C# 7 tuples.

Commonly Used Types:

- System.ValueTuple
- System.ValueTuple<T1>
- System.ValueTuple<T1, T2>
- System.ValueTuple<T1, T2, T3>
- System.ValueTuple<T1, T2, T3, T4>
- System.ValueTuple<T1, T2, T3, T4, T5>
- System.ValueTuple<T1, T2, T3, T4, T5, T6>
- System.ValueTuple<T1, T2, T3, T4, T5, T6, T7>
- System.ValueTuple<T1, T2, T3, T4, T5, T6, T7, TRest>

When using NuGet 3.x this package requires at least

那一年，我們使用的 Tuple

```
static void Main(string[] args)
{
    var data = GetSomthing();
    Console.WriteLine($" {data.Item1} : {data.Item2}");
    Console.ReadLine();
}

private static Tuple<int, string> GetSomthing()
{
    int i = 100;
    string s = "ABC";
    return Tuple.Create(i, s);
}
```

ValueTuple 的各種宣告方式

```
ValueTuple<int, string> x1 = ValueTuple.Create<int, string>(8, "ABC");  
Console.WriteLine($"(1) {x1.Item1} : {x1.Item2}");  
  
var x2 = (8, "ABC");  
Console.WriteLine($"(2) {x2.Item1} : {x2.Item2}");  
  
var x3 = (length: 8, letters: "ABC");  
Console.WriteLine($"(3) {x3.length} : {x3.letters}");  
  
(int length, string letters) x4 = (8, "ABC");  
Console.WriteLine($"(4) {x4.length} : {x4.letters}");  
  
(int length, string letters) x5 = (first: 8, second: "ABC");  
Console.WriteLine($"(5) {x5.length} : {x5.letters}");
```

ValueTuple 在方法回傳值的應用

```
static void Main(string[] args)
{
    var range = GetRange();
    Console.WriteLine($"{range.min} -- {range.max}");

    (int lower, int upper) limit = GetRange();
    Console.WriteLine($"{limit.lower} -- {limit.upper}");

    Console.ReadLine();
}

private static (int min, int max) GetRange()
{
    int min = 0;
    int max = 100;
    return (min, max);
}
```

Deconstruct

Deconstruct 的用途

- 讓型別具有使用指派運算子，也就是 `=`，把特定內容的值指派給 `ValueTuple` 型別的變數。

建立 Deconstruct 執行個體方法

```
class Program
{
    static void Main(string[] args)
    {
        MyRectangle rect = new MyRectangle() { Width = 10, Height = 30 };
        (int x, int y) = rect;
        Console.WriteLine($"{x} -- {y}");
        Console.ReadLine();
    }
}

public class MyRectangle
{
    public int Width { get; set; }
    public int Height { get; set; }

    public void Deconstruct(out int width, out int height)
    {
        width = this.Width;
        height = this.Height;
    }
}
```


Deconstruct 也可以用擴充方法形式

```
public class MyRectangle
{
    public int Width { get; set; }
    public int Height { get; set; }
}

public static class MyExtension
{
    public static void Deconstruct
        (this MyRectangle rect, out int width, out int height)
    {
        width = rect.Width;
        height = rect.Height;
    }
}
```

Deconstruct 一個有趣的小地方

```
MyRectangle rect = new MyRectangle() { Width = 5, Height = 60 };  
(int x, int y) = rect;  
Console.WriteLine($"{x} -- {y}");  
  
Console.ReadLine();
```



```
(new MyRectangle() { Width = 5, Height = 60 })  
    .Deconstruct(out int x, out int y);  
Console.WriteLine($"{x} -- {y}");  
Console.ReadLine();
```

ref local and return

ref 的新變革

- 在區域變數內直接取得變數指標
- 在方法回傳值直接回傳變數指標

以前要操作變數指標是一件挺麻煩的事

```
static void Main(string[] args)
{
    int number = 100;
    unsafe
    {
        int* p = &number;
        Console.WriteLine(*p);
        *p = 999;
        Console.WriteLine(number);
    }
    Console.ReadLine();
}
```

現在你只要這麼做就行了

```
static void Main(string[] args)
{
    int number = 100;
    ref int p = ref number;
    Console.WriteLine(p);
    p = 999;
    Console.WriteLine(p);
    Console.ReadLine();
}
```

陣列處理 – 以前

```
static void Main(string[] args)
{
    string[] data = new string[] { "鼠", "牛", "虎", "兔" };
    int index = GetTigerIndex(data);
    data[index] = "老虎";
    Display(data);
    Console.ReadLine();
}

private static int GetTigerIndex(string[] data)
{
    return Array.IndexOf(data, "虎");
}
```

陣列處理 – ref return

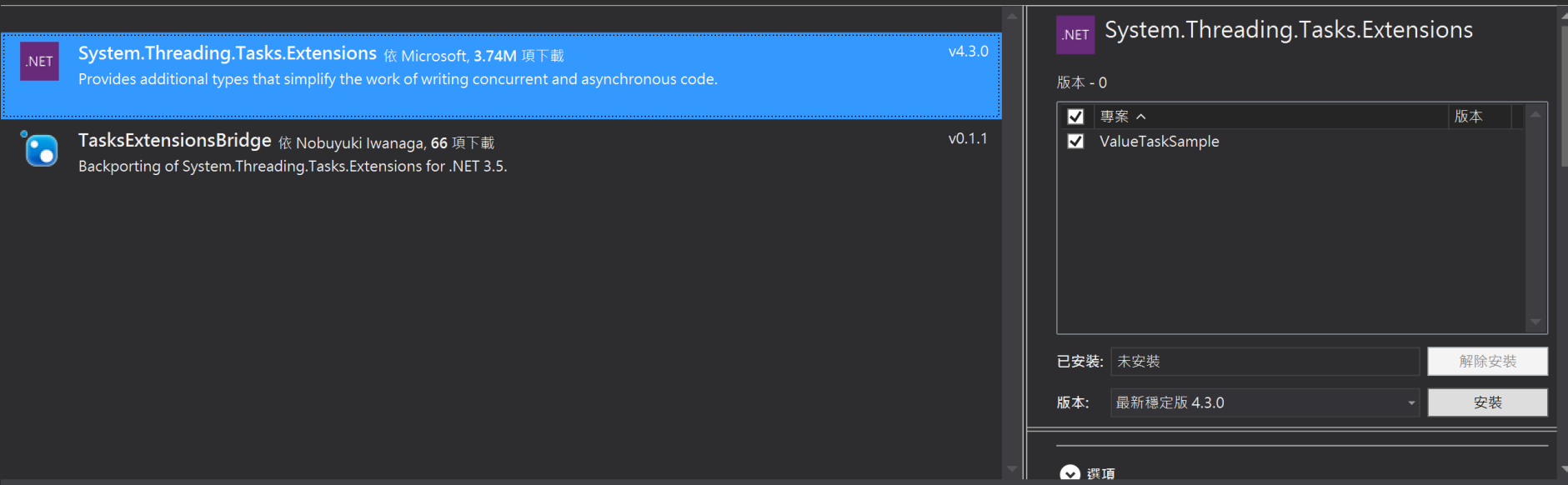
```
static void Main(string[] args)
{
    string[] data = new string[] { "鼠", "牛", "虎", "兔" };
    ref string s = ref GetTiger(data);
    s = "大老虎";
    Display(data);
    Console.ReadLine();
}

private static ref string GetTiger(string[] data)
{
    return ref data[Array.IndexOf(data, "虎")];
}
```


ValueTask

ValueTask

- ValueTask 以泛型結構形式定義
- 以 nuget 套件的形式加入參考



```
async private static void Begin()
{
    int x = await Execute();
    Console.WriteLine(x);
}
async private static ValueTask<int> Execute()
{
    await Task.Delay(5000);
    return 300;
}
```

長出好多結構

pattern matching

pattern matching 重點

- 加強 `is` 運算子
- 加強 `switch` 敘述

以前你得要這樣寫

```
static void Main(string[] args)
{
    int i = 10;
    Execute(i);
    Console.ReadLine();
}

private static void Execute(object value)
{
    if (value is int)
    { Console.WriteLine((int)value); }
    else
    { Console.WriteLine("not int"); }
}
```



現在你可以這麼寫

```
static void Main(string[] args)
{
    int i = 10;
    Execute(i);
    Console.ReadLine();
}

private static void Execute(object value)
{
    if (value is int x) ←
    { Console.WriteLine(x); }
    else
    { Console.WriteLine("not int"); }
}
```


在 switch 上的運用

```
private static void InitialShape(IShape shape, double x, double y)
{
    switch (shape )
    {
        case MyRectangle s:
            s.Height = x;
            s.Width = y;
            break;
        case MyCircle s:
            s.Radius = x;
            break;
        case null:
            break;
    }
}
```

在 switch 上加入 when

```
private static void Execute(object value)
{
    switch (value)
    {
        case int x when x > 0 && x < 100:
            Console.WriteLine($"x 是小整數 : {x}");
            break;
        case int x when x > 99 && x < 1000:
            Console.WriteLine($"x 是大整數 : {x}");
            break;
        case int x:
            Console.WriteLine($"x 超出範圍");
            break;
        case string x:
            Console.WriteLine($"x 是字串 : {x}");
            break;
        default:
            Console.WriteLine("不在 case 內");
            break;
    }
}
```

local functions

local functions

- 改善只會被某個特定 `method` 呼叫的 `method` 到處躲來躲去的問題。
- `async / await` 也能用。

沒有 local function 以前

```
static void Main(string[] args)
{
    List<string> list1 = new List<string>() { "1", "2", "3", "4", "5" };
    Display(list1);
    List<string> list2 = new List<string>() { "A", "B", "C", "D", "E" };
    Display(list2);
    Console.ReadLine();
}

private static void Display(List<string> list)
{
    foreach (var item in list)
    {
        Console.WriteLine(list);
    }
}
```

使用 local function

```
static void Main(string[] args)
{
    List<string> list1 = new List<string>() { "1", "2", "3", "4", "5" };
    Display(list1);
    List<string> list2 = new List<string>() { "A", "B", "C", "D", "E" };
    Display(list2);
    Console.ReadLine();

    void Display(List<string> list)
    {
        foreach (var item in list)
        {
            Console.WriteLine(item);
        }
    }
}
```

還有一個超神奇的用法

```
static void Main(string[] args)
{
    List<string> list1 = new List<string>() { "1", "2", "3", "4", "5" };
    var list = list1;
    Display();
    List<string> list2 = new List<string>() { "A", "B", "C", "D", "E" };
    list = list2;
    Display();
    Console.ReadLine();

    void Display()
    {
        foreach (var item in list)
        {
            Console.WriteLine(item);
        }
    }
}
```

更狂的 expression-bodied members

expression-bodied members

- 在 C# 6.0 開始出現此形式的寫法。本來只用在 method 和 read only property。
- C# 7.0 將這個形式擴張到
 - constructor
 - finalizer
 - getter and setter on property
 - indexer

```
public class MyCircle
{
    private double _radius;
    private string _name;
    public double Radius
    {
        get => _radius;
        set => this._radius = value;
    }

    public string Name
    {
        get => _name;
        set => this._name = value ?? "就是這個圓";
    }

    public MyCircle() => _radius = 2;

    public double GetArea() =>
        (_radius > 0) ? Math.PI * (Math.Pow(_radius, 2)) : 0;
}
```

throw expression

throw

- 以前 `throw` 是個單純的敘述(statement) 所以沒有辦法直接放在運算式中使用
- 在 C# 7.0 中新增了 `throw` 運算式(expression) 解決了這個問題。

```
public class MyCircle
{
    private double _radius;
    private string _name;
    public double Radius
    {
        get => _radius;
        set => _radius = value > 0 ? value : throw new ArgumentException();
    }

    public string Name
    {
        get => _name;
        set => this._name = value ?? throw new ArgumentException();
    }
}
```

Blog 是記錄知識的最佳平台



<https://dotblogs.com.tw>

史上最強的注音輸入法



<http://www.iq-t.com/PRODUCTS/going11.asp>



千呼萬喚

MAC 平台上，最聰明的注音輸入法
現在立即前往購買！



SKILLTREE



完全支援

請放心在 Windows 8 上使用 新自然輸入法 10 系統升級免煩惱，平板模式也沒問題！

OzCode

Your Road to Magical Debugging



```
Id: 1 Name: "Tom Lee" "India Express"
float CalculateCost( Customer customer , string restaurant)
{
    float courseCost = GetCourseCost(restaurant);
    bool shouldTip = waiter.IsNice && courseCost > COSTLY_MEAL;
```

Exceptions Trail:



ReservationException > HotelException > IndexOutOfRangeException

Exception:

System.IndexOutOfRangeException

Message:

"Invalid customer index"

[Go to where exception was thrown](#) [Go to where exception was handled](#)

<http://www.oz-code.com/>

```
foreach (var hay in haystack)
```

haystack

[793]

ItemType

Oz.Samples+Stuff

needle

needle

學員可使用 Yammer 取得優惠價

謝謝各位

<http://skilltree.my>

-
- 本投影片所包含的商標與文字皆屬原作者所有，僅供教學之用。
 - 本投影片的內容包括標誌、設計、文字、圖像、影片、聲音...等著作財產權均屬電魔小鋪有限公司所有，受到中華民國著作權法及國際著作權法律的保障。對本投影內容進行任何形式的引用、轉載、重製前，請務必取得電魔小鋪有限公司的"書面授權"，否則請勿使用，以免侵權。