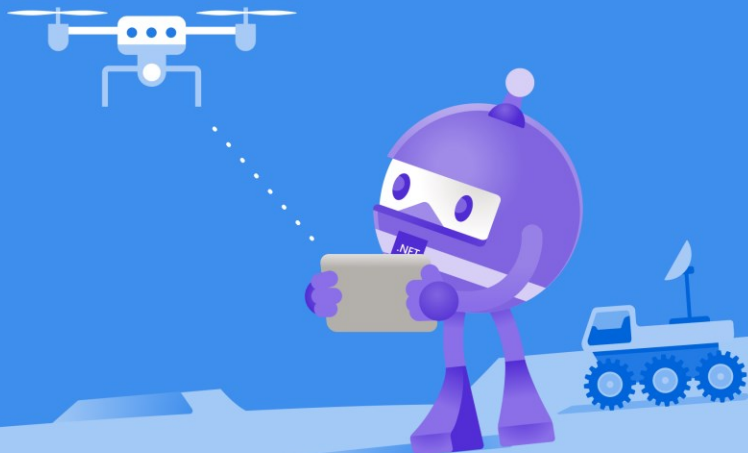


.NET Conf

探索 .NET 新世界



C# 9.0

C# 的過去、現在和未來

Ouch Liu (劉耀群)

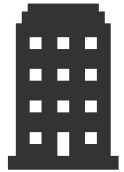


關於我



Ouch Liu (劉耀群)

略懂設計的軟體工程師 / 略懂程式的設計師



軒昂有限公司

Principal Solution Architect



2011~2016 MVP / CSM / CSPO / CSD / PSM

曾任 TechDays Taiwan 講師 / 微軟客座講師



ouch1978.github.io



[ouch1978](#)



大綱







- C# 的過去
- C# 的現在 – C# 9.0
- C# 的未來
- Q & A
- 參考資料

C# 的過去




C# 1.0 ~ C# 8.0



C# 的過去 – 1.0 ~ 6.0

| 1.0  | 2.0  | 3.0  | 4.0  | 5.0  | 6.0  |
|--|--|--|--|---|---|
| 2002 | 2005 | 2008 | 2010 | 2012 | 2014 |
| <ul style="list-style-type: none"> Classes Structs Interfaces Events Properties Delegates Expressions Statements Attributes Literals | <ul style="list-style-type: none"> Generics Partial types Anonymous methods Iterators Nullable types Getter/setter separate accessibility Method group conversions Co- and Contravariance for delegates and interfaces Static classes Delegate inference | <ul style="list-style-type: none"> Implicitly typed local variables Object and collection initializers Auto-Implemented properties Anonymous types Extension methods Query expressions Lambda expression Expression trees Partial methods | <ul style="list-style-type: none"> Dynamic binding Named and optional arguments Generic co- and contravariance Embedded interop types | <ul style="list-style-type: none"> Asynchronous methods Caller info attributes | <ul style="list-style-type: none"> Import of static type Exception filters Await in catch/finally blocks Auto property initializers Default values for getter-only properties Expression-bodied members Null propagator String interpolation nameof operator Dictionary initializer |

C# 的過去 – 7.0 ~ 8.0

| 7.0  15.0 | 7.1  15.3 | 7.2  15.5 | 7.3  15.7 | 8.0  16.3 |
|---|--|---|---|--|
| Mar 2017 | Aug 2017 | Dec 2017 | May 2018 | Sep 2019 |
| <ul style="list-style-type: none"> ▪ Out variables ▪ Pattern matching ▪ Tuples ▪ Deconstruction ▪ Discards ▪ Local functions ▪ Binary literals ▪ Digit separators ▪ Ref returns and locals ▪ Generalized async return types ▪ More expression-bodied members ▪ Throw expressions | <ul style="list-style-type: none"> ▪ Async main ▪ Default expressions ▪ Reference assemblies ▪ Inferred tuple element names ▪ Pattern-matching with generics | <ul style="list-style-type: none"> ▪ ref readonly ▪ BlitTable ▪ StrongNameProvider ▪ Interior Pointer/Span/ref Struct ▪ Non-Trailing Named Arguments ▪ private protected ▪ Conditional ref Operator | <ul style="list-style-type: none"> ▪ System.Enum, System.Delegate and unmanaged constraints ▪ Ref local re-assignment ▪ Stackalloc initializers ▪ Indexing movable fixed buffers ▪ Custom fixed statement ▪ Improved overload candidates ▪ Expression variables in initializers and queries ▪ Tuple comparison ▪ Attributes on backing fields | <ul style="list-style-type: none"> ▪ Nullable reference types ▪ Default interface members ▪ Recursive patterns ▪ Async streams ▪ Enhanced using ▪ Ranges and indexes ▪ Null-coalescing assignment ▪ Static local functions ▪ Unmanaged generic structs ▪ ReadOnly members ▪ Stackalloc in nested contexts ▪ Alternative interpolated verbatim strings ▪ Obsolete on property accessors |

C# 的現在

C# 9.0



在我們開始之前

先別急，有些事我得先說清楚...



C# 語言版本與目標架構之間的關係

| 目標架構 | 目標架構版本 | C# 語言版本預設值 |
|----------------|-----------|------------|
| .Net Framework | 所有版本 | 7.3 |
| .Net Standard | 1.0 ~ 2.0 | 7.3 |
| | 2.1 | 8.0 |
| .Net Core | 2.x | 7.3 |
| | 3.x | 8.0 |
| .NET | 5 | 9.0 |

手動將專案預設的 C# 語言版本修改成更新的版本可能會導致編譯時期與執行時期產生難以辨識的錯誤。

C# 9.0 使用注意事項

- 內建於 Visual Studio 2019 16.8 版
- 只支援 .Net 5 以上專案

手動將專案預設的 C# 語言版本修改成更新的版本可能會導致編譯時期與執行時期產生難以辨識的錯誤。

C# 9.0 新功能列表

- Records and with expressions
- Init-only setters
- Top-level statements
- Pattern matching enhancements
- Native sized integers
- Function pointers
- Suppress emitting localsinit flag
- Target-typed new expressions
- Static anonymous functions
- Target-typed conditional expressions
- Covariant return types
- Lambda discard parameters
- Attributes on local functions
- Module initializers
- Extension GetEnumerator
- Partial methods with returned values

手動將專案預設的 C# 語言版本修改成更新的版本可能會導致編譯時期與執行時期產生難以辨識的錯誤。

C# 9.0 新功能列表 – Cont.

- Records and with expressions
- Init-only setters
- Top-level statements
- Pattern matching enhancements
- Native sized integers
- Function pointers*
- Suppress emitting localsinit flag
- Target-typed new expressions
- Static anonymous functions
- Target-typed conditional expressions
- Covariant return types
- Lambda discard parameters
- Attributes on local functions
- Module initializers
- Extension GetEnumerator
- Partial methods with returned values

手動修改專案預設的 C# 語言版本後可以使用的功能 (*代表只有特定平台能使用，例如 UWP)

說好的能在舊框架下使用呢?

- 在專案檔中 (.csproj) 加入：

<LangVersion>9.0</LangVersion>

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <LangVersion>9.0</LangVersion>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>
</Project>
```

手動修改專案預設的 C# 語言版本有好有壞，修改前請詳閱風險說明書。

Init-only setters

- 讓我們可以建立出只有在建構時期能被初始化或賦值的成員。

```
interface Icar
{
    string Brand { get; init; }
    int Year { get; init; }
    string Color { get; set; }
}

public class Alphard : ICar
{
    public string Brand { get; init; }
    public int Year { get; init; }
    public string Color { get; set; }
}

Alphard alphard = new Alphard { Brand = "Toyota" , TypeName = "Alphard" , Color = "Red" ,
    Year = 2020 };
alphard.Year = 2021; //錯誤
alphard.Color = "White";
```

Records

- Records 讓我們可以輕鬆的在 .NET 中建立不可變(唯讀)的參考型別。

```
public record Member
{
    public string LastName { get; }
    public string FirstName { get; }

    public Member( string first , string last ) => ( FirstName, LastName ) = ( first,
last );
}

public record VipMember : Member
{
    public int VipLevel { get; }
    public VipMember( string first , string last , int vipLevel )
        : base( first , last ) => VipLevel = vipLevel;
}
```


With expressions

- 可以透過 with 運算式來建立 Records 的副本，並且修改其內容。

```
Member member = new Member( "Ouch" , "Liu" );

Member clone = member with { }; //建立一模一樣的

//透過 with 運算式複製一個，並且修改 FirstName 的值。
Member anotherMember = clone with { FirstName = "Nicolas" };
```

- 彩蛋：Positional Records

```
//Positional Records
public record Product( string Name , string Brand , int Price );

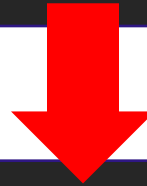
public Product BuildProduct( string name , string brand , int price )
{
    //Construction by position
    return new( name , brand , price );
}
```



Top level statements

```
using System;

namespace CSharpNine
{
    class Program
    {
        static void Main( string[] args )
        {
            Console.WriteLine( "C# 9.0 rocks!!!" );
        }
    }
}
```



```
System.Console.WriteLine( "C# 9.0 rocks!!!" );
```

Pattern matching enhancements

- 讓我們可以透過 is 、 and 、 or 、 not 、 when 、 > 、 < 、 = 和括弧來拼湊出更有彈性的模式比對邏輯。

```
private static bool IsNightTime( DateTime time ) =>
    time.Hour switch
    {
        > 6 and < 23 => true,
        _ => false
    };
```

```
public int CalculateFare( object vehicle ) =>
    vehicle switch
    {
        Taxi { IsNightTime: true } or Uber { IsNightTime: true } => 70 + 20,
        Uber or Taxi => 70,
        Bus b when( b.Segments > 1 ) => 15 * b.Segments,
        Mrt { IsTransfer: true } => 20 - 8,
        Mrt => 20,

        { } => throw new ArgumentException( message: "Not a known vehicle type" , paramName: nameof(
vehicle ) ),
        null => throw new ArgumentNullException( nameof( vehicle ) )
    };
```

Native sized integers

- 以前，有一個判斷是否為 64 位元 Runtime 的方法：

```
var framework = ( IntPtr.Size == 8 ) ? "x64" : "x86";
```

- C# 9.0 借用了以前在 mono 等平台就有的概念，以便於 Interop 和其它低階功能的互動，創造了 nint 和 nuint。

```
nint myNint = 27; int myInt = 27; nuint myNuint = 27; long myLong = 10; dynamic myDynamic = 1;
const nint myConst = int.MaxValue; //myConst=2147483647

//const nint myConst2 = nint.MaxValue; //會錯，nint.MaxValue 不是常數

bool isMyNintAndMyIntEquals = nint.Equals( myNint , myInt ); //False
bool isMyNintAndCastedIntEquals = nint.Equals( myNint , ( nint ) myInt ); //True

Type typeOfNintAddInt = ( myNint + myInt ).GetType(); //System.IntPtr
Type typeOfNintAddLong = ( myNint + myLong ).GetType(); // System.Int64

//Type typeOfNintAddNuint = ( myNint + myNuint ).GetType(); //會錯，nint 和 nuint 不能相加

Type typeOfNintAddDynamic = ( myNint + myDynamic + myNuint ).GetType(); //RuntimeBinderException: 'Op
erator '+' cannot be applied to operands of type 'System.IntPtr' and 'System.UIntPtr'
```

Static anonymous functions

- 基於 lambda 對效能的耗損相對的大(而且容易被忽視)，所以在 C# 9.0 中導入了可以實作靜態的匿名函式的功能 (PHP也有)。

```
readonly string _message = "{0} 好帥!"; const string ConstMessage = "{0} 超級帥!!!";

public void PraiseSomeone( Func<string , string> func )
{
    var names = new List<string> { "Ouch" , "Nicolas" , "Rex" };
    foreach( var name in names )
        Console.WriteLine( func( name ) );
}

public void Praise()
{
    // _message 變數會不小心的被保存，或是佔用額外的記憶體
    PraiseSomeone( name => string.Format( _message , name ) );
}

public void PraiseWithStatic()
{
    // ConstMessage 常數並不會被捕捉
    PraiseSomeone( static name => string.Format( ConstMessage , name ) );
}
```

Target-typed new expressions

- 從 C# 3.0 開始，編譯器可以透過 var 來推論等號左邊的型別，而這個新功能則讓編譯器可以推論等號右邊的型別。

```
Member member = new Member( "Ouch" , "Liu" ); //C# 3.0 之前，我們得這樣寫
```

```
var member2 = new Member( "Niqolas" , "Liu" ); //C# 3.0 之後，我們可以這樣寫
```

```
Member member3 = new( "Rex" , "Liu" ); //C# 9.0 之後，我們可以這樣寫
```

```
//C# 9.0之後，還可以這樣寫
```

```
List<Member> members = new()  
{  
    new( "Ouch" , "Liu" ),  
    new( "Niqolas" , "Liu" ),  
    new( "Rex" , "Liu" )  
};
```

```
//當然，這樣寫也是可以的
```

```
MemoryStream reader = new();  
XmlReader.Create( reader , new() { IgnoreWhitespace = true } );
```

Target-typed conditional expressions

- C# 9.0 也帶來了更多針對型別的判斷式，例如針對型別的三元運算。

```
IVehicle vehicle = new Taxi();  
var myVehicle = vehicle ?? new Uber(); //C# 9.0 OK  
  
bool isReal;  
var anotherVehicle = isReal ? new Uber() : null; //C# 9.0 OK  
  
int? x = new Random().NextDouble() > 0.5 ? 1 : null;  
IEnumerable<int> _ = x is null ? new List<int>() { 0 , 1 } : new int[]  
{ 2 , 3 };  
  
//當使用 var 時，冒號兩邊的變數必需是相同的型別  
var another = x is null ? ( int? ) null : 3;
```

Covariant return types

- 在之前，當我們覆寫某個方法 (method) 的時候，是無法改變它的回傳型別的，C# 9.0 打破了這個限制。

```
public class CovariantParentClass
{
    public virtual Taxi BuildTaxi()
    {
        return new Taxi();
    }
}

public class CovariantChildClass : CovariantParentClass
{
    //C# 9.0 之前只能以 Taxi 型別回傳，現在可以用 Uber 回傳了
    public override Uber BuildTaxi()
    {
        return new Uber();
    }
}
```


Lambda discard parameters

- 從 C# 7.0 開始，可以透過 `_` 來保留刻意未使用的預留位置參數。現在，`_` 也可以被運用在 Lambda 語法中。

```
//兩個參數都不使用
```

```
Func<int , int , int> myNumber = ( _ , _ ) => 27;
```

```
//只使用第一和第三個參數
```

```
Func<int , int , int , int , int> firstPlusThird = ( x , _ , y , _ ) => x + y;
```

```
//C# 8.0 之前得要這樣寫
```

```
btnMyButton.Click += ( s , e ) => { MessageBox.Show( "MyButton Clicked" ); };
```

```
//C# 9.0 之後可以這樣寫
```

```
btnMyButton.Click += ( _ , _ ) => { MessageBox.Show( "MyButton Clicked" ); };
```

```
//為了向下相容，如果陳述式中只包含一個_的話，它就會被當成參數名稱
```

```
Func<int , int> doubleNumber = ( _ ) => _ * 2;
```

Attributes on local functions

- 這個功能讓我們可以在本地的函式上使用屬性 (Attribute) 。

```
class Program
{
    static void Main( string[] args )
    {
        static void PrintText( string text )
        { System.Console.WriteLine( text ); }
#if DEBUG
PrintText( "C# 8.0 rocks!!!" );
#endif
    }
}
```



```
[System.Diagnostics.Conditional( "DEBUG" )]
static void PrintText( string text )
{ System.Console.WriteLine( text ); }

PrintText( "C# 9.0 rocks!!!" );
```

Module initializers

- 我們可以透過 `ModuleInitializerAttribute` 來幫助我們在程式起
始時執行一次我們指定的方法。

```
using System.Runtime.CompilerServices;
```

```
class Program
{
    public static string Name;

    static void Main( string[] args )
    { System.Console.WriteLine( $"{Name} rocks!!!" ); }

    [ModuleInitializer]
    public static async void Init()
    { Name = await GetName( "Ouch" ); }

    public static Task<string> GetName( string name ) => Task.FromResult( name );
}
```

Extension GetEnumerator

- 讓我們從得要實作執行個體的 GetEnumerator 方法，變成可以透過擴充方法來實作。

```
public static class CharExtension
{
    public static IEnumerator<char> GetEnumerator( this char end )
    {
        int count = ( int ) end - ( int ) 'A';
        return Enumerable.Range( 0 , count ).Select( i => Convert.ToChar( 'A' + i ) ).GetEnumerator();
    }
}
```

```
foreach( char c in 'G' )
{
    Console.WriteLine( c );
}
```

Partial methods with returned values

- 以前的 Partial Methods 回傳型別只能是 void、不能包含 out 參數，也不能使用任何存取子。 C# 9.0 把這些限制都移除了。

```
partial class MyService
{
    partial void MyFirstFunc(); //可以，沒存取子的話不用有實作方法
    //private partial void MySecondFunc(); //會錯，有存取子就要有實作方法
    private partial object MyThirdFunc(); //可以
    private partial void MyFourthFunc( out int result ); //可以
}

partial class MyService
{
    private partial object MyThirdFunc() { return new { }; } //可以
    private partial void MyFourthFunc( out int result )
    { result = 0; }
}
```

C# 的未來

C# 10.0



C# 10.0

- Pattern match Span<char> on a constant string
 - Proposed
 - Prototype: Completed
 - Implementation: In Progress
 - Specification: Not Started
- 現在只有一個新功能提案，歡迎大家一起參與許願/設計的行列

Q & A

我問 & 你答



請說出一個 C# 8.0 的功能

第一題



請說出一個 C# 9.0 的新功能

第二題



C# 10.0

目前有多少新功能提案?

第三題



參考資料

意猶未盡嗎? 更多原汁原味的資訊在這裡...



參考資料

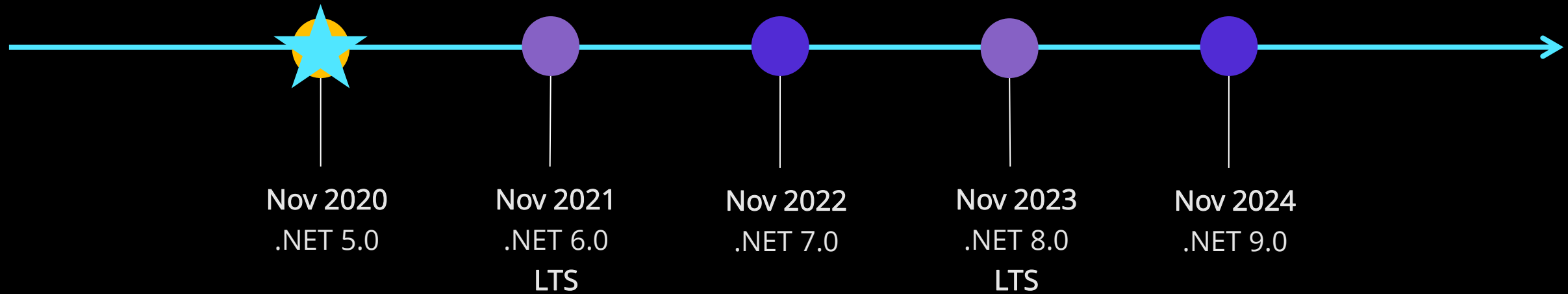
- Deep Dive Into C# 9
 - <https://www.c-sharpcorner.com/article/deev-dive-into-c-sharp-9/>
- Enabling and using C# 9 features on older and “unsupported” runtimes
 - <https://medium.com/@SergioPedri/enabling-and-using-c-9-features-on-older-and-unsupported-runtimes-ce384d8debb>
- .Net 海角點部落 - C# 9.0 搶先看
 - <https://dotblogs.com.tw/billchung/Series?qq=C%23%209.0%20%E6%90%B6%E5%85%88%E7%9C%8B>

Sample Code

<https://github.com/Ouch1978/CSharp9Demo>

(尚未重構，請小心/耐心使用)

.NET Schedule



- .NET 5.0 released today!
- Major releases every year in November
- LTS for even numbered releases
- Predictable schedule, minor releases as needed

感謝您的參與

- 請使用 #dotNETConf 在 Twitter 上發問



.NET Conf
2020

特別感謝

91APP
Technical Network



KKKTIX



HackMD



STUDY4
為 學 習 而 生

以及各位參與活動的你們

