Build School

Build School 課程 – C# 基礎課程 03

Bill Chung
Build School 講師
V2022.3 台北冬季班

Build School

遞增與遞減運算子

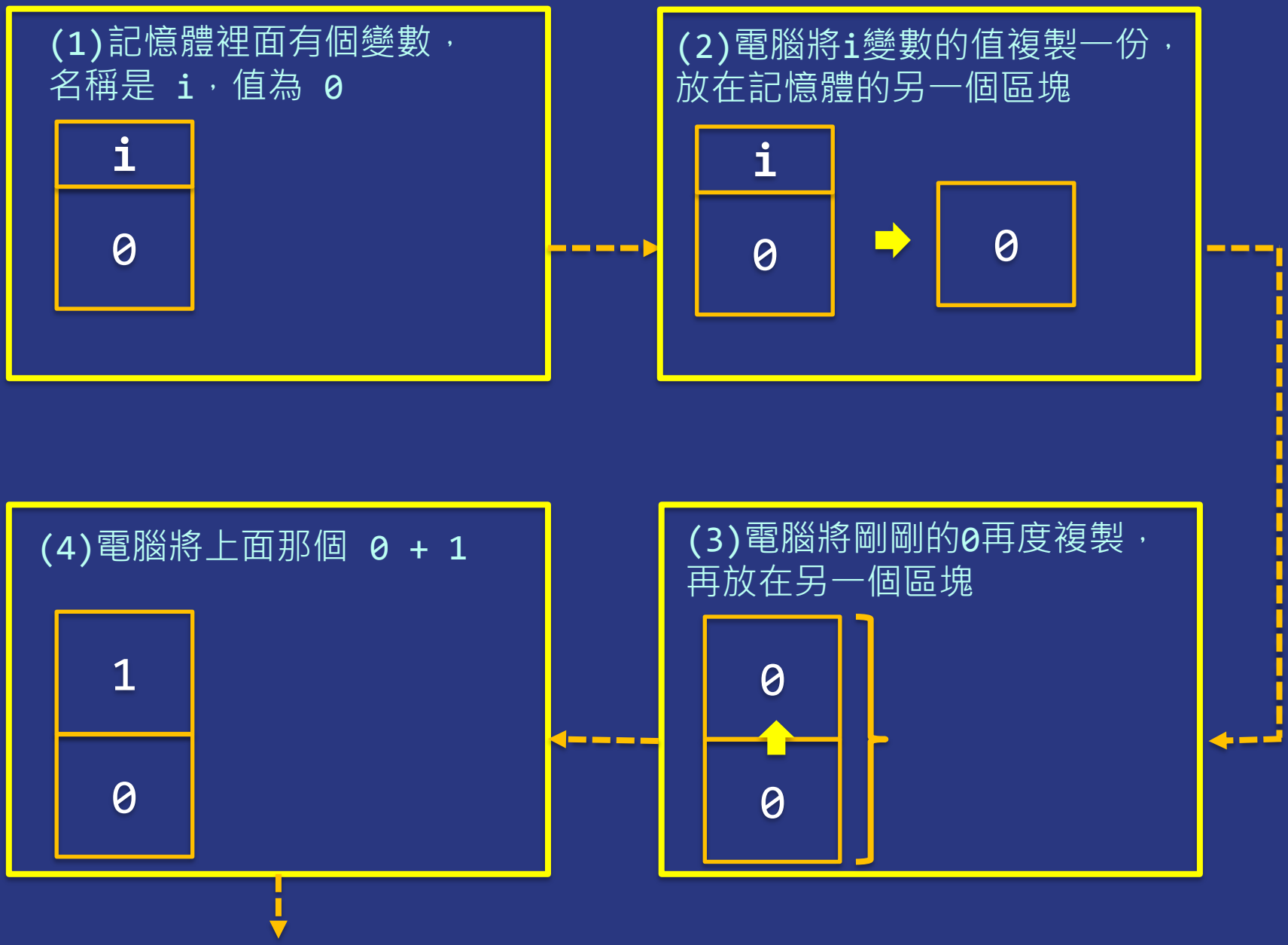
理解以下的流程

- `++i`
- `i++`
- `--i`
- `i--`
- 它們都是遞增（`++`）或遞減（`--`），但究竟有何不同。

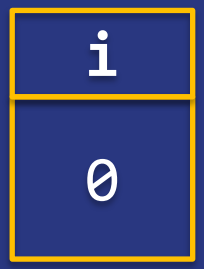
先記住以下關於 i++ 的程式碼

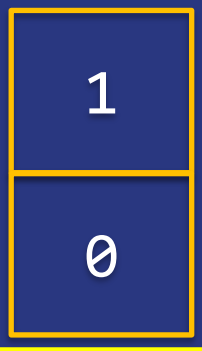
```
static void Main(string[] args)
{
    int i = 0;
    i = i++;
    Console.WriteLine(i);
    Console.ReadLine();
}
```

DoubleSamples\DoubleSample001

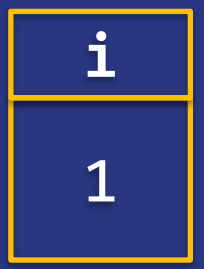

(1)記憶體裡面有個變數，
名稱是 i，值為 0
i
0
(2)電腦將i變數的值複製一份，
放在記憶體的另一個區塊
i
0
0
(3)電腦將剛剛的0再度複製，
再放在另一個區塊
0
0
(4)電腦將上面那個 0 + 1
1
0

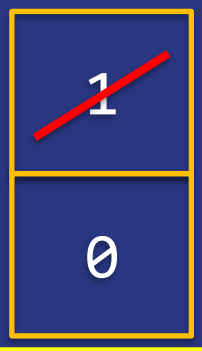
(5)此時記憶體是這樣的



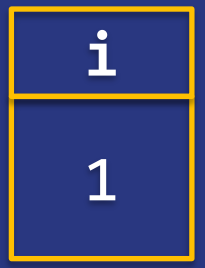


(6)電腦將上方的1複製給i變數
並且將上方的1移除

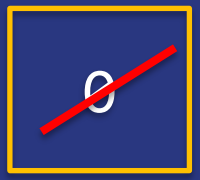




(7)此時記憶體是這樣，完成了
指派運算子右方的運算式



(8)電腦將剩下的那個0複製給
i變數，並將那個0移除

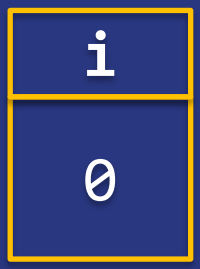


先記住以下關於 ++i 的程式碼

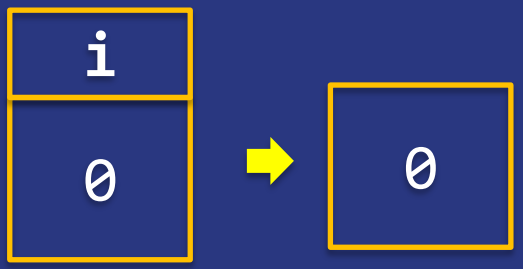
```
static void Main(string[] args)
{
    int i = 0;
    i = ++i;
    Console.WriteLine(i);
    Console.ReadLine();
}
```

DoubleSamples\DoubleSample002

(1)記憶體裡面有個變數，
名稱是 i，值為 0

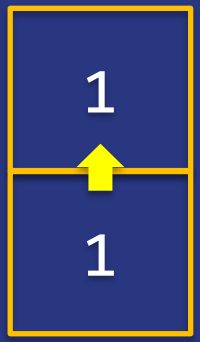


(2)電腦將i變數的值複製一份，
放在記憶體的另一個區塊

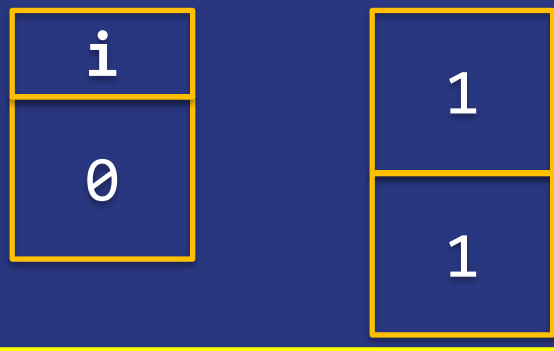


(3)電腦將剛剛的0+1

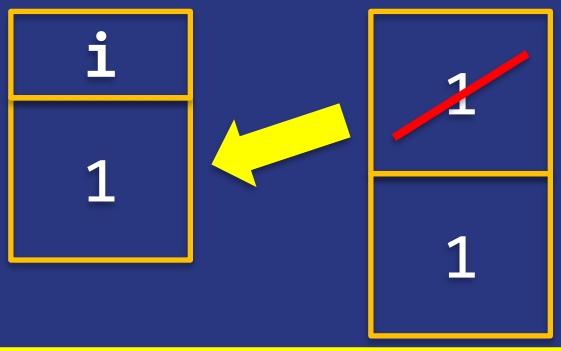
(4)電腦將剛剛的1再度複製，
再放在另一個區塊



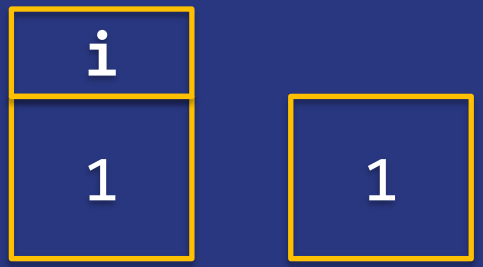
(5)此時記憶體是這樣的



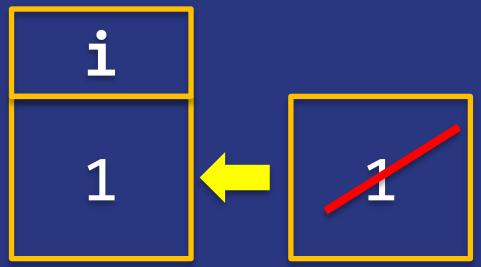
(6)電腦將上方的1複製給i變數
並且將上方的1移除



(7)此時記憶體是這樣，完成了
指派運算子右方的運算式



(8)電腦將剩下的那個1複製給
i變數，並將那個1移除



討論

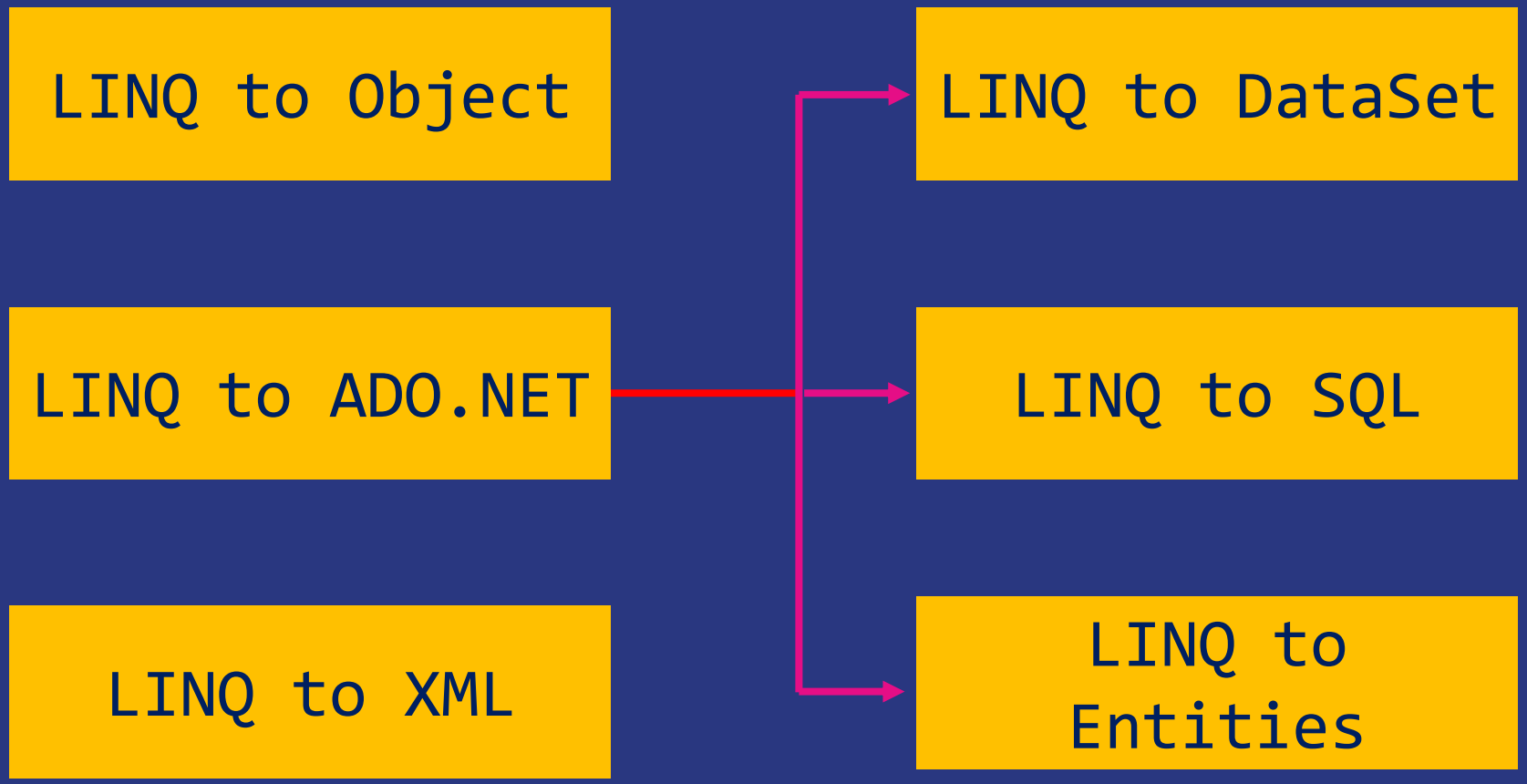
- 經由上述的分析，你們是否可以順利地解釋 i=i++ 和 i=++i 的不同？

linq

MSDN 文件庫對 linq 的介紹

- 「查詢」(Query) 是一種從資料來源擷取資料的運算式， 而且使用專用的查詢語言來表示。
- 一段時間之後，針對不同類型的資料來源已開發出不同的語言，例如 SQL 是用於關聯式資料庫，而 XQuery 則是 XML。 因此，開發人員必須針對他們所需支援的資料來源類型或資料格式學習新的查詢語言。

- LINQ 提供一致的模型來使用各種資料來源和格式的資料，從而簡化此情況。 在 LINQ 查詢中，您所處理的一定是物件。 不論您要查詢及轉換的資料是存在 XML 文件、SQL 資料庫、ADO.NET 資料集，還是 .NET 集合，以及其他任何有可用 LINQ 提供者 (Provider) 的格式中，都是使用相同的基本程式碼撰寫模式。



Enumerable Class

- https://docs.microsoft.com/zh-tw/dotnet/api/system.linq.enumerable?redirectedfrom=MSDN&view=netframework-4.7.2

- Linq to object 的實作在這個類別中
- 這些方法都是【擴充方法】
- 【擴充方法】也是一種語法糖，它使得靜態方法可以像執行個體方法一般的使用。(只有像而已)

Linq 的敘述的種類

- Query Expression
  - 類似 SQL 查詢的寫法
- Method Expression
  - 以 C# Method 的形式呈現

Query Expression 入門

```
from <某個東西>
in  <來源 or 另一個 query expression >
<
  <where expression> or
  <group expression> or
  <join expression>
>
<select expression>
```

LAB

基本的 Query Expression

新增一個方案及專案

- 方案名稱：LinqSamples
- 專案名稱：LinqSample001
- 範本：Console Application

加入 MyData Class，並建立其內容

```
class MyData
 {
     public string Name
     { get; set; }

     public int Age
     { get; set; }
 }
```

在 Program Calss 加入產生 List<MyData> 的方法 並且在 Main Method 中呼叫它以產生一個集合

```
static void Main(string[] args)
{
    List<MyData> list = CreateList();
}
static List<MyData> CreateList()
{
    return new List<MyData>()
    {
        new MyData { Name = "Bill", Age = 47 },
        new MyData { Name = "John", Age = 37 },
        new MyData { Name = "Tom", Age = 48 },
        new MyData { Name = "David", Age = 36 },
        new MyData { Name = "Bill", Age = 35 },
    };
 }
```

在 Main Method 加入 Query Expression 和 顯示結果的程式碼

```
static void Main(string[] args)
{
    List<MyData> list = CreateList();

    //Query Expression
    IEnumerable<MyData> people =
        from data in list
        where data.Name == "Bill"
        select data;

    //顯示結果
    foreach (MyData person in people)
    {
        Console.WriteLine($"{person.Name} 是 {person.Age} 歲");
    }
    Console.ReadLine();
}
```

執行

討論

- 請和你的夥伴們討論 Query Expression 的運作

var 宣告

- 強型別
- 隱含型別 (所以這是個語法糖)
- 或稱右(後)決議型別
- 只能做為宣告區域變數使用

討論

- 請開啟 VarSample 範例，和你的同學討論裡面的程式碼。
- 務必了解 var 怎麼用

LAB

將剛剛 Query Expression 轉成 Method Expression 的寫法

在 LinqSamples 加入新專案

- 方案名稱：LinqSamples
- 專案名稱：LinqSample002
- 範本：Console Application

加入 MyData Class，並建立其內容

```
class MyData
 {
     public string Name
     { get; set; }

     public int Age
     { get; set; }
 }
```

在 Program Calss 加入產生 List<MyData> 的方法並且在 Main Method 中呼叫它以產生一個集合

```
static void Main(string[] args)
{
    List<MyData> list = CreateList();
}
static List<MyData> CreateList()
{
    return new List<MyData>()
    {
        new MyData { Name = "Bill", Age = 47 },
        new MyData { Name = "John", Age = 37 },
        new MyData { Name = "Tom", Age = 48 },
        new MyData { Name = "David", Age = 36 },
        new MyData { Name = "Bill", Age = 35 },
    };
 }
```

在 Main Method 加入 Method Expression 和 顯示結果的程式碼

```
static void Main(string[] args)
{
    var list = CreateList();

    // Method Expression
    var people = list.Where((x) => x.Name == "Bill");

    //顯示結果
    foreach (MyData person in people)
    {
        Console.WriteLine($"{person.Name} 是 {person.Age} 歲");
    }

    Console.ReadLine();
}
```

執行

Select 哪裡去了？

- 在剛剛這麼簡單的查詢下，Method Expression 的 Select 是可以省略的。若要包含 Select 語法則需寫成以下型式

```
var people = list.Where((x) => x.Name == "Bill").Select((x) => x);
```

找到第一個符合條件的資料

- `First`
  - 找到符合條件的第一個，沒找到就會發生例外
- `FirstOrDefault`
  - 找到符合條件的第一個，沒找到就回傳預設值

預設值的概念

- 對於 byte, short, int, long, float, double, decimal 這些數值型的資料，預設值是 0。
- 對於 bool 預設值是 false。
- 對於 string, object 或其他參考型別的資料，預設值是 null。
- null 是一種特殊的資料，代表甚麼都沒有。

LAB

First 與 FirstOrDefault

在 LinqSamples 加入新專案

- 方案名稱：LinqSamples
- 專案名稱：LinqSample003
- 範本：Console Application

加入 MyData Class，並建立其內容

```
class MyData
 {
     public string Name
     { get; set; }

     public int Age
     { get; set; }
 }
```

在 Program Calss 加入產生 List<MyData> 的方法並且在 Main Method 中呼叫它以產生一個集合

```
static void Main(string[] args)
{
    var list = CreateList();
}
static List<MyData> CreateList()
{
    return new List<MyData>()
    {
        new MyData { Name = "Bill", Age = 47 },
        new MyData { Name = "John", Age = 37 },
        new MyData { Name = "Tom", Age = 48 },
        new MyData { Name = "David", Age = 36 },
        new MyData { Name = "Bill", Age = 35 },
    };
 }
```

在 Main Method 加入 程式碼

```
static void Main(string[] args)
{
    // 開始用 var 了
    var list = CreateList();
    // 這裡的 person1 是單個物件, 也就是 MyData person1
    var person1 = list.FirstOrDefault((x) => x.Age < 37);
    Console.WriteLine($"小於 37 歲的人第一個被找到的是 : {person1.Name}");

    // 因為找不到, 就會跳出例外
    var person2 = list.First((x) => x.Age < 30);
    Console.WriteLine($"小於 30 歲的人第一個被找到的是 : {person2.Name}");

    Console.ReadLine();
}
```

執行

找到最後一個符合條件的資料

- `Last`
  - 找到符合條件最後一個，沒找到就會發生例外
- `LastOrDefault`
  - 找到符合條件的最後一個，沒找到就回傳預設值

LAB

Last 與 LastOrDefault

在 LinqSamples 加入新專案

- 方案名稱：LinqSamples
- 專案名稱：LinqSample004
- 範本：Console Application

加入 MyData Class，並建立其內容

```
class MyData
 {
     public string Name
     { get; set; }

     public int Age
     { get; set; }
 }
```

在 Program Calss 加入產生 List<MyData> 的方法並且在 Main Method 中呼叫它以產生一個集合

```
static void Main(string[] args)
{
    var list = CreateList();
}
static List<MyData> CreateList()
{
    return new List<MyData>()
    {
        new MyData { Name = "Bill", Age = 47 },
        new MyData { Name = "John", Age = 37 },
        new MyData { Name = "Tom", Age = 48 },
        new MyData { Name = "David", Age = 36 },
        new MyData { Name = "Bill", Age = 35 },
    };
 }
```

在 Main Method 加入 程式碼

```
static void Main(string[] args)
{
    var list = CreateList();
    // 這裡的 person1 是單個物件, 也就是 MyData person1
    var person1 = list.LastOrDefault((x) => x.Age > 35);
    Console.WriteLine($"大於 35 歲的人最後一個被找到的是 : {person1.Name}");

    // 因為找不到, 就會跳出例外
    var person2 = list.Last((x) => x.Age > 50);
    Console.WriteLine($"大於 50 歲的人最後一個被找到的是 : {person2.Name}");

    Console.ReadLine();
}
```

執行

找到符合條件而且是唯一一個的資料

- Single
  - 找到符合條件而且是唯一一個，沒找到就會發生例外
  - 如果符合條件的結果有兩個(含)以上，發生例外
- SingleOrDefault
  - 找到符合條件而且是唯一一個，沒找到就回傳預設值
  - 如果符合條件的結果有兩個(含)以上，發生例外

LAB

Single 與 SingleOrDefault

在 LinqSamples 加入新專案

- 方案名稱：LinqSamples
- 專案名稱：LinqSample005
- 範本：Console Application

加入 MyData Class，並建立其內容

```
class MyData
 {
     public string Name
     { get; set; }

     public int Age
     { get; set; }
 }
```

在 Program Calss 加入產生 List<MyData> 的方法並且在 Main Method 中呼叫它以產生一個集合

```
static void Main(string[] args)
{
    var list = CreateList();
}
static List<MyData> CreateList()
{
    return new List<MyData>()
    {
        new MyData { Name = "Bill", Age = 47 },
        new MyData { Name = "John", Age = 37 },
        new MyData { Name = "Tom", Age = 48 },
        new MyData { Name = "David", Age = 36 },
        new MyData { Name = "Bill", Age = 35 },
    };
 }
```

在 Main Method 加入 程式碼

```
static void Main(string[] args)
{
    var list = CreateList();
    // 這裡的 person1 是單個物件, 也就是 MyData person1
    var person1 = list.SingleOrDefault((x) => x.Name  == "Tom");
    Console.WriteLine($"找到唯一的 : {person1.Name} - {person1.Age}");

    // 因為找不到唯一 (裡面有兩個 Bill)  就會跳出例外
    var person2 = list.Single((x) =>x.Name == "Bill");
    Console.WriteLine($"找到唯一的 : {person2.Name} - {person2.Age}");

    Console.ReadLine();
}
```

註：SingleOrDefault 若是遇到有兩個符合條件也會跳出例外

執行

討論

- 討論 First、Last 和 Single 的差異
- 討論 xxxOrDefault 的作用

正確使用 xxxOrDefault

- 不論是 FirstOrDefault、LastOrDefault 或任何及其他 xxxOrDefault 都有可能會回傳 null。
- 該怎麼處理這樣的情形？

LAB

正確處理預設值

在 LinqSamples 加入新專案

- 方案名稱：LinqSamples
- 專案名稱：LinqSample006
- 範本：Console Application

加入 MyData Class，並建立其內容

```
class MyData
 {
     public string Name
     { get; set; }

     public int Age
     { get; set; }
 }
```

在 Program Calss 加入產生 List<MyData> 的方法並且在 Main Method 中呼叫它以產生一個集合

```
static void Main(string[] args)
{
    var list = CreateList();
}
static List<MyData> CreateList()
{
    return new List<MyData>()
    {
        new MyData { Name = "Bill", Age = 47 },
        new MyData { Name = "John", Age = 37 },
        new MyData { Name = "Tom", Age = 48 },
        new MyData { Name = "David", Age = 36 },
        new MyData { Name = "Bill", Age = 35 },
    };
 }
```

在 Main Method 加入 程式碼
一定找不到李小龍

```
static void Main(string[] args)
{
    var list = CreateList();
    var person = list.FirstOrDefault((x) => x.Name == "李小龍");
    // 判斷回傳結果是否為 null
    if (person == null )
    {
        //如果是 null 則另行處理
        Console.WriteLine("查無此人");
    }
    else
    {
        Console.WriteLine($"找到 : {person.Name} - {person.Age}");
    }

    Console.ReadLine();
}
```

執行

如果有可能出現 null
一定要處理

取出某個位置的資料

- `ElementAt`
  - 找到特定位置的資料，沒找到就會發生例外
- `ElementAtOrDefault`
  - 找到特定位置的資料，沒找到就回傳預設值

在 LinqSamples 加入新專案

- 方案名稱：LinqSamples
- 專案名稱：LinqSample007
- 範本：Console Application

加入 MyData Class，並建立其內容

```
class MyData
 {
     public string Name
     { get; set; }

     public int Age
     { get; set; }
 }
```

在 Program Calss 加入產生 List<MyData> 的方法並且在 Main Method 中呼叫它以產生一個集合

```
static void Main(string[] args)
{
    var list = CreateList();
}
static List<MyData> CreateList()
{
    return new List<MyData>()
    {
        new MyData { Name = "Bill", Age = 47 },
        new MyData { Name = "John", Age = 37 },
        new MyData { Name = "Tom", Age = 48 },
        new MyData { Name = "David", Age = 36 },
        new MyData { Name = "Bill", Age = 35 },
    };
 }
```

在 Main Method 加入 程式碼

```
static void Main(string[] args)
{
    int index = 1;
    var list = CreateList();
    // 這裡的 person 是單個物件, 也就是 MyData person
    var person = list.ElementAtOrDefault(index);
    if (person == null)
    {
        Console.WriteLine("查無此人");
    }
    else
    {
        Console.WriteLine($"找到索引為 : {index} 的人是 {person.Name}
                                                    - {person.Age}");
    }
    Console.ReadLine();
}
```

執行

判斷是否有符合條件的資料

- Any
  - 判斷是否有一個或以上數量的資料符合條件
- All
  - 判斷是否所有的資料都符合條件

LAB

Any

在 LinqSamples 加入新專案

- 方案名稱：LinqSamples
- 專案名稱：LinqSample008
- 範本：Console Application

加入 MyData Class，並建立其內容

```
class MyData
 {
     public string Name
     { get; set; }

     public int Age
     { get; set; }
 }
```

在 Program Calss 加入產生 List<MyData> 的方法並且在 Main Method 中呼叫它以產生一個集合

```
static void Main(string[] args)
{
    var list = CreateList();
}
static List<MyData> CreateList()
{
    return new List<MyData>()
    {
        new MyData { Name = "Bill", Age = 47 },
        new MyData { Name = "John", Age = 37 },
        new MyData { Name = "Tom", Age = 48 },
        new MyData { Name = "David", Age = 36 },
        new MyData { Name = "Bill", Age = 35 },
    };
 }
```

在 Main Method 加入 程式碼

```
static void Main(string[] args)
{

    var list = CreateList();
    string name = "David";
    bool result = list.Any((x) => x.Name == name);
    if (result)
    {
        Console.WriteLine($"找到了 : {name}");
    }
    else
    {
        Console.WriteLine($"找不到 : {name}");
    }
    Console.ReadLine();
}
```

執行

LAB

All

在 LinqSamples 加入新專案

- 方案名稱：LinqSamples
- 專案名稱：LinqSample009
- 範本：Console Application

加入 MyData Class，並建立其內容

```
class MyData
 {
     public string Name
     { get; set; }

     public int Age
     { get; set; }
 }
```

在 Program Calss 加入產生 List<MyData> 的方法並且在 Main Method 中呼叫它以產生一個集合

```
static void Main(string[] args)
{
    var list = CreateList();
}
static List<MyData> CreateList()
{
    return new List<MyData>()
    {
        new MyData { Name = "Bill", Age = 47 },
        new MyData { Name = "John", Age = 37 },
        new MyData { Name = "Tom", Age = 48 },
        new MyData { Name = "David", Age = 36 },
        new MyData { Name = "Bill", Age = 35 },
    };
 }
```

在 Main Method 加入 程式碼

```
static void Main(string[] args)
{
    var list = CreateList();
    string name = "Bill";
    bool isAllBill = list.All((x) => x.Name == name);
    if (isAllBill)
    {
        Console.WriteLine($"全都是 {name}");
    }
    else
    {
        Console.WriteLine($"有些人不叫 {name}");
    }
    bool isAllOverForty = list.All((x) => x.Age >= 40);
    if (isAllOverForty)
    {
        Console.WriteLine("大家都超過 40 歲");
    }
    else
    {
        Console.WriteLine("有人不到 40 歲");
    }
    Console.ReadLine();
}
```

執行

在這裏你將學到 ....

Learn How to Learn

- **學新東西、新技術的能力**
- **尋找解答的能力**
- **隨時吸取新知識的能力**

跟著你一輩子的能力 ...

Build School