

.NET delegate 解釋.

Web Clip

以下請用台灣繁體中文回答，請說明.net 的delegate是什麼



.NET 的 delegate 是一種類型，它可以將方法當作參數傳遞，也可以將其視為回傳值。Delegate 可以用來建立事件處理方法，也可以實現委派模式。Delegate 類型的變數可以持有一個或多個方法的參考，並且可以在需要時呼叫這些方法。透過 delegate，可以實現將方法當作參數進行傳遞的功能，使程式碼更具彈性和可讀性。

它與event的關係是什麼



在 .NET 中，delegate 和 event 經常搭配使用，可以用來實現事件驅動的程式設計模式。Event 是一種 delegate 的實例，它表示當特定事件發生時要執行的方法清單。事件的訂閱者可以向 event 註冊自己的方法，當事件發生時，這些方法就會被執行。Delegate 則可以被用來定義 event 的型別，以及用來訂閱和取消訂閱事件的方法。因此，delegate 和 event 是密切相關的，它們一起提供了一個方便的方式來處理和觸發事件。

請示範如何宣告一個class Member ,包含Email property,並且在這屬性值被異動時，觸發DataChanged event



以下是一個範例程式碼，宣告一個名為 Member 的 class，該

class 包含 Email property，並且當 Email 屬性的值變更時會觸發 DataChanged event：

csharp

```
public class Member
{
    private string email;

    public event EventHandler DataChanged;

    public string Email
    {
        get
        {
            return email;
        }
        set
        {
            if (email != value)
            {
                email = value;
                OnDataChanged();
            }
        }
    }

    protected virtual void OnDataChanged()
    {
        DataChanged?.Invoke(this, EventArgs.Empty);
    }
}
```

在這個 class 中，我們宣告了一個 private field `email`，以及一個 public 屬性 `Email`，透過這個屬性來存取 `email` 值。在 `Email` 屬性的 setter 中，我們加入了條件判斷，確保只有在 `email` 值有變更時才會觸發 `DataChanged event`。在 `OnDataChanged` 方法中，我們使用 null 條件運算子 `?.` 來確認 `DataChanged event` 是否為

null，並且使用 `Invoke` 方法來觸發該事件。當事件被觸發時，我們將 `this` 和一個空的 `EventArgs` 物件傳遞給事件的註冊方法。

請示範如何呼叫它並訂閱此事件



以下是一個範例程式碼，示範如何呼叫 `Member` class 並訂閱其 `DataChanged` 事件：

```
csharp

// 建立 Member 物件
Member member = new Member();

// 訂閱 DataChanged 事件
member.DataChanged += Member_DataChanged;

// 變更 Email 屬性的值
member.Email = "example@example.com";

// 定義 DataChanged 事件的處理方法
private void Member_DataChanged(object sender, EventArgs e)
{
    Console.WriteLine("Member data has changed.");
}
```

在這個範例中，我們先建立一個 `Member` 物件，然後訂閱其 `DataChanged` 事件，使用 `+=` 運算子將事件處理方法 `Member_DataChanged` 加入事件訂閱清單中。接著，我們變更 `Email` 屬性的值，觸發 `DataChanged` 事件。當事件被觸發時，註冊的事件處理方法 `Member_DataChanged` 就會被執行，輸出 "Member data has changed." 的訊息。

請說明自訂 delegate 與 Action, Func 的區別



自訂的 delegate、Action 和 Func 都是用來定義方法的型別，這些型別可以作為參數或回傳值來傳遞方法。它們的區別如下：

1. 自訂 delegate：自訂 delegate 是一個使用 delegate 關鍵字所宣告的型別，它可以定義任意的方法簽章，並且可以被用來建立委派實例。使用自訂 delegate 可以增加程式碼的可讀性和重複使用性。
2. Action：Action 是 .NET Framework 中已經預先定義好的一個 delegate 型別，它可以接受最多 16 個參數且沒有回傳值。可以把 Action 當成是一個沒有回傳值的委派型別，可用於定義事件處理方法、委派、回呼方法等。
3. Func：Func 也是 .NET Framework 中已經預先定義好的一個 delegate 型別，不過與 Action 不同的是，Func 可以接受最多 16 個參數且必須有一個回傳值。在使用 Func 時，最後一個泛型型別參數表示回傳值的型別，前面的泛型型別參數表示各個參數的型別。可以把 Func 當成是一個有回傳值的委派型別，可用於定義回呼方法、委派等。

總結來說，自訂 delegate 的彈性最大，可以定義任意的方法簽章；Action 可以用於定義沒有回傳值的方法；Func 可以用於定義有回傳值的方法。

請修改剛才的 Member class, 將事件改用 Action



以下是一個修改後的 Member class，將事件改用 Action：

```
csharp
```

```
class Member
{
    private string email;

    public Action DataChanged { get; set; }

    public string Email
    {
        get { return email; }
        set
        {
            email = value;
            DataChanged?.Invoke();
        }
    }
}
```

在這個修改後的 Member class 中，我們將 DataChanged 事件改為使用 Action 委派型別。在 Email 屬性被設定值的 setter 方法中，我們使用了 null 條件運算子 `?.`，當 DataChanged 不為 null 時才會觸發 DataChanged 事件，這也可以防止因為沒有訂閱事件而造成的例外狀況。

請修改程式碼，仍然保留 OnDataChanged method; 此外，事件的委派改成 `Action<Member, string newValue>`



以下是一個修改後的 Member class，保留 OnDataChanged 方法，並將事件的委派改為 `Action<Member, string>`：

```
csharp
```

```
class Member
{
    private string email;

    public event Action<Member, string> DataChanged;

    public string Email
    {
        get { return email; }
        set
        {
            email = value;
            OnDataChanged(email);
        }
    }

    protected void OnDataChanged(string newValue)
    {
        DataChanged?.Invoke(this, newValue);
    }
}
```

在這個修改後的 Member class 中，我們將 DataChanged 事件的委派改為 Action<Member, string>，它可以接受兩個參數，第一個是 Member 型別，代表事件發生的物件，第二個是 string 型別，代表 Email 屬性設定的新值。在 Email 屬性的 setter 方法中，我們不再直接呼叫事件，而是呼叫 OnDataChanged 方法，這個方法負責觸發 DataChanged 事件。在 OnDataChanged 方法中，我們先檢查 DataChanged 是否為 null，如果不為 null 就呼叫 DataChanged 事件，並且傳遞 this 和 newValue 這兩個參數。這樣可以保證當 DataChanged 事件有多個訂閱者時，每個訂閱者都會接收到正確的參數。

請列出教學delegate的大綱



以下是一個教學 `delegate` 的大綱：

1. Delegate 是什麼？

委派是一種類似指標的型別，可以指向一個方法。

委派可以用來實現事件、回呼函式等功能。

2. 宣告委派

委派可以使用 C# 的 `delegate` 關鍵字來宣告。

委派宣告時需要指定方法的簽名。

委派可以宣告在類別內或者外部。

3. 委派的使用

委派可以用來呼叫方法。

委派可以被當成一個參數傳遞到其他方法中。

委派可以用來實現事件。

委派可以用來實現回呼函式。

4. 委派的種類

無傳回值委派

委派的種類

委派指向的方法沒有傳回值。

可以使用 **Action** 委派型別來宣告。

有傳回值委派

委派指向的方法有傳回值。

可以使用 **Func** 委派型別來宣告。

5. 委派的多重呼叫

可以使用 **+**、**-** 操作子將多個委派合併成一個委派。

多個委派會依照它們加入的順序被依序呼叫。

可以使用 **GetInvocationList** 方法取得委派的方法列表。

6. 委派與多執行緒

多個執行緒可以同時呼叫委派。

需要注意多執行緒的同步問題，例如使用鎖定等機制。

7. 委派的優點與缺點

委派的優點與缺點

委派可以簡化程式碼，增加可讀性。

委派可以提供彈性的設計，例如實現事件、回呼函式等功能。

委派的呼叫效能較差，需要消耗額外的記憶體與計算資源。

請詳細說明「使用 `GetInvocationList` 方法取得委派的方法列表」，並提供範例程式



`GetInvocationList` 方法是用來取得委派所包含的所有方法的列表。當一個委派包含多個方法時，可以使用 `GetInvocationList` 方法來取得這些方法的列表。這些方法會按照它們加入委派的順序排序。這個方法返回的是一個 `Delegate[]` 類型的陣列，可以逐一讀取每個 `Delegate` 物件，並取得它所包含的方法資訊。

以下是一個使用 `GetInvocationList` 方法的範例程式：

```
csharp

using System;

public class Program
{
    public static void Main()
    {
        Action<int> delegatel = PrintNumber;
        delegatel += DoubleNumber;
```

```
Console.WriteLine("Delegate invocation list:");
    Delegate[] invocationList = delegatel.GetInvocationList();
    foreach (var d in invocationList)
    {
        Console.WriteLine($"Method name: {d.Method.Name}");
    }

    delegatel(5);
}

public static void PrintNumber(int number)
{
    Console.WriteLine($"Number is: {number}");
}

public static void DoubleNumber(int number)
{
    Console.WriteLine($"Double number is: {number * 2}");
}
}
```

這個程式宣告了一個 `Action<int>` 委派，並把 `PrintNumber` 和 `DoubleNumber` 兩個方法加入委派的呼叫列表中。接著使用 `GetInvocationList` 方法來取得委派的方法列表，並逐一顯示每個方法的名稱。最後呼叫委派，會依照加入委派的順序呼叫兩個方法。這個程式的輸出結果如下：

yaml

```
Delegate invocation list:
Method name: PrintNumber
Method name: DoubleNumber
Number is: 5
Double number is: 10
```

可以看到 `GetInvocationList` 方法返回了委派所包含的兩個方法，並且這兩個方法都被成功呼叫。

請說明delegate在LINQ裡的應用



在 LINQ 中，Delegate 常用來定義條件和轉換函數，以在資料集合中執行查詢操作。

例如，在 LINQ 中使用 Where 方法可以根據指定的條件篩選資料集合中的元素，這個條件可以使用 Func<T, bool> 委派來定義。這個委派表示一個接受一個 T 型別的參數並返回一個 bool 值的方法，可以根據這個委派來篩選資料集合。

以下是一個使用 Where 方法的範例程式：

```
csharp

using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };

        // 使用 Where 方法和 Func<int, bool> 委派來篩選奇數
        var oddNumbers = numbers.Where(x => x % 2 != 0);

        Console.WriteLine("Odd numbers:");
        foreach (var number in oddNumbers)
        {
            Console.WriteLine(number);
        }
    }
}
```

這個程式宣告了一個整數型別的集合，並使用 `Where` 方法和 `Func<int, bool>` 委派來篩選奇數。在 `Where` 方法中傳入了一個 `lambda` 運算式，這個運算式可以被轉換為一個 `Func<int, bool>` 委派。這個委派表示一個接受一個整數參數並返回一個布林值的方法，用來判斷一個整數是否為奇數。接著遍歷篩選後的集合，並將所有奇數顯示出來。這個程式的輸出結果如下：

```
yaml
```

```
Odd numbers:
```

```
1
```

```
3
```

```
5
```

可以看到只有奇數被篩選出來了。這個範例中使用了 `Func<int, bool>` 委派來定義篩選條件，但是在 LINQ 中還有很多其他的委派類型，例如 `Func<TSource, TKey>` 委派用來定義轉換函數、`Action<T>` 委派用來定義操作等。這些委派類型可以根據需要進行使用。
