# Pattern và Pattern matching (so khớp mẫu) trong C#



Pattern matching (so khớp mẫu) là một tính năng quen thuộc trong các ngôn ngữ lập trình hàm như F#. Tuy nhiên, trong các ngôn ngữ lập trình hướng đối tượng như C#, pattern matching không phải là một tính năng phổ biến.

Pattern matching bắt đầu xuất hiện trong C# 7 nhưng có nhiều giới hạn. C# 8 tiếp tục mở rộng khả năng của pattern matching. Tất cả những cải tiến trên kéo C# về hướng lập trình hàm và giúp ngôn ngữ ngày càng đa dạng, phong phú nhưng lại ngắn gọn súc tích hơn.

**NỘI DUNG CỦA BÀI** [ Ẩn ]



## Khái niệm pattern và pattern matching trong C#

Pattern matching, tạm dịch là so khớp mẫu, là một khái niệm khá trừu tượng và hơi khó hiểu. Thay vì giải thích dài dòng, chúng ta hãy cùng xem một ví dụ đơn giản.

```
switch (car.Color) {
    case Color.Red:
        Console.WriteLine("Color is red!");
        break;
    case Color.Blue:
        Console.WriteLine("Color is blue!");
        break;
    default:
        Console.WriteLine("Color is not red or blue!");
        break;
}
```

Đây là một cấu trúc switch-case quen thuộc mà bạn làm quen ngay từ những bài học C# đầu tiên. Và đây cũng là một ví dụ về pattern matching trong C#!

Mỗi hằng của một case (như Color.Red, Color.Blue) chính là một **pattern**. Nếu một biến có giá trị trùng với hằng của một case, chúng ta gọi trường hợp đó là một **match**. Khi có một match, chương trình sẽ thực hiện một/một số lệnh nào đó.

Hiểu theo cách đơn giản nhất,

- Pattern là một đặc điểm nào đó của dữ liệu. Pattern có thể là một giá trị cụ thể cố định (hằng), cũng có thể là kiểu của dữ liệu, có thể là một bộ phận của dữ liệu, v.v.. Nhìn chung, tất cả nhưng gì của dữ liệu giúp chúng ta phân biệt được nó với dữ liệu khác đều có thể sử dụng làm pattern.
- Pattern matching là quá trình kiểm tra xem dữ liệu có những đặc điểm chúng ta mong muốn hay không. Nếu có – sẽ thực hiện công việc gì đó.

Với đặc điểm trên, pattern và pattern matching thường được thực thi trong các cấu trúc điều khiển rẽ nhánh.

Cấu trúc switch-case cổ điển chính là cấu trúc vận dụng pattern matching, và loại pattern trong cấu trúc này là **constant pattern** (giá trị cố định cụ thể của dữ liệu). Đây là trường hợp đơn giản nhất của pattern matching trong C#.

Với ý nghĩa trên, cấu trúc if-else cũng có thể xem là cấu trúc vận dụng pattern matching.

Các phiên bản C# về sau đưa vào nhiều loại pattern mới. Những thay đổi này giúp viết mã C# ngắn gọn súc tích và hiệu quả hơn:

- C# 7 đưa vào từ khóa `is`, `when` và cho phép sử dụng từ khóa `var` trong biểu thức case.
- C# 8 bổ sung thêm các loại pattern mới (positional, property, tuple) và switch expression.

> Pattern matching cũng là một khái niệm trong lĩnh vực học máy (machine learning). Bạn cũng gặp khái niệm này khi học về biểu thức chính quy (regular expression). Trong bài viết này chúng ta đề cập tới pattern matching với vai trò một tính năng của ngôn ngữ lập trình.

## Từ khóa `is`, so khớp kiểu

Khả năng so khớp kiểu xuất hiện trong C# 7. So khớp kiểu cho phép bạn sử dụng kiểu của object làm pattern.

Bạn có thể thực hiện so khớp kiểu trong cấu trúc if-else với từ khóa is, hoặc sử dụng kiểu làm pattern trực tiếp trong cấu trúc switch-case.

Hãy cùng xem một vài ví dụ nhỏ. Giả sử bạn định nghĩa 3 class mô tả các hình hình học như sau:

```
1.  class Rectangle {
2.      public int Width { get; set; }
3.      public int Height { get; set; }
4.  }
5.
6.  class Cirle {
7.      public int Radius { get; set; }
8.  }
9.
```

```
class Square {
    public int Length { get; set; }
}
```

Giờ bạn cần viết một phương thức tính diện tích của một hình bất kỳ: `double Area(object shape) { ... }`

Ở đây phát sinh mấy vấn đề.

- Thứ nhất, các class trên hoàn toàn không có liên hệ gì.
- Để tính được diện tích, bạn cần xác định đó là hình gì.

Theo các kỹ thuật thông thường đã biết, bạn cần thử cast object shape về các kiểu đã biết. Nếu cast thành công, shape chính là object của class đó:

```
var rectangle = shape as Rectangle;
if (rectangle != null) return rectangle.Width * rectangle.Height;
```

Khi sử dụng kỹ thuật này, bạn sẽ viết phương thức Area như sau:

```
private static double Area(object shape) {
    var rectangle = shape as Rectangle;
    if (rectangle != null) return rectangle.Width * rectangle.Height;

    var circle = shape as Cirle;
    if (circle != null) return Math.PI * circle.Radius * circle.Radius;

    return double.NaN;
}
```

Type matching của C# 7 cho phép bạn viết lại phương thức Area theo cách sau:

```
private static double Area(object shape) {
    if (shape is Rectangle r) return r.Width * r.Height;
    if (shape is Cirle c) return Math.PI * c.Radius * c.Radius;
    if (shape is Square s) return s.Length * s.Length;
    return double.NaN;
}
```

Như vậy cặp lệnh cast kiểu và kiểm tra null tương đương với một lệnh is duy nhất

```
var rectangle = shape as Rectangle;
if (rectangle != null) ...

tương đương với

if (shape is Rectangle rectangle) ...
```

C# cho phép sử dụng type pattern với cấu trúc switch-case như sau:

```
private static double Area(object shape) {
    switch (shape) {
        case Rectangle r: return r.Width * r.Height;
        case Cirle c: return Math.PI * c.Radius * c.Radius;
        case Square s: return s.Length * s.Length;
        default: return double.NaN;
    }
}
```

Trong cấu trúc switch sử dụng type pattern matching đôi khi bạn muốn kiểm tra thêm các điều kiện bổ sung. Lấy ví dụ, bạn có thể muốn tính diện tích nếu kích thước các hình nằm trong khoảng từ 10 đến 100.

Type pattern cho phép sử dụng từ khóa when để đặt thêm các điều kiện bổ sung. Hãy cùng xem ví dụ sau:

```
1.  private static double AreaSwitchWhen(object shape) {
2.      switch (shape) {
3.          case Rectangle r: return r.Width * r.Height;
4.          case Cirle c when c.Radius > 10 && c.Radius < 100: return Math.PI * c.Radius * c
5.          case Square s when s.Length > 10 && s.Length < 100: return s.Length * s.Length;
6.          default: return double.NaN;
7.      }
8.  }
```

Từ khóa when cho phép bạn viết thêm các kiều kiện bổ sung khi so khớp kiểu. Giờ đây bạn chỉ thực hiện tính diện tích hình tròn nếu đường kính hình tròn nằm trong khoảng (10, 100). Tương tự như vậy khi tính diện tích hình vuông.

## Từ khóa var, so khớp biến

Giờ hãy xem một ví dụ khác:

```
1.  private static string GreetingSwitch(string name) {
2.      switch (name) {
3.          case var n when n.ToLower().Contains("putin"): return "Privet Vova!";
4.          case string n when n.ToLower().Contains("trump"): return "Hello, Mr. president";
5.          case var n when n.Trim() == "": return "Sorry, who are you?";
6.          default: return $"Hi, {name}";
7.      }
8.  }
```

Ví dụ này có điểm khác biệt: cụm `var n when` và `string n when` trong các case. Hai cụm này có ý nghĩa: (1) hãy lấy giá trị của name (biến kiểm tra) và gán vào n cho tôi, (2) kiểm tra điều kiện đi sau when.

Ở đây bạn không còn sử dụng type pattern nữa, và nó cũng không phải là constant pattern truyền thống. Ở đây bạn đang sử dụng var pattern.

Trong var pattern, giá trị của biến kiểm tra được truyền vào cho từng case để thực hiện các biến đổi và kiểm tra riêng rẽ.

Trong ví dụ này, hai case đầu tiên chúng ta chuyển sang chữ thường và kiểm tra xem có chứa cụm “putin” hoặc “trump” không. Ở case thứ 3 chúng ta xóa bỏ các ký tự trống rồi so với xâu rỗng.

Khi dùng từ khóa var như trên, giá trị của biến kiểm tra name sẽ được truyền vào biến n của case. Từ đây bạn có thể thực hiện bất kỳ thao tác biến đổi và kiểm tra nào với n, cũng chính là với name.

Nếu không muốn dùng var, bạn có thể chỉ định trực tiếp kiểu dữ liệu của biến kiểm tra trong mỗi case. Như trong case thứ hai, do biến name có kiểu string, biến n trong case có thể được chỉ định kiểu trực tiếp là string (vì n chính là name).

Nếu trong cấu trúc switch thông thường trước đây bạn không thể thực hiện những biến đổi riêng rẽ và kiểm tra với từng case như vậy.

So khớp kiểu và so khớp biến giúp cấu trúc switch-case trở nên rất mạnh mẽ. Tuy nhiên nó vẫn còn cách xa với khả năng của các ngôn ngữ lập trình hàm.

C# 8 tiếp tục đưa vào một biểu thức mới giúp C# tiến gần hơn nữa với lập trình hàm: switch expression.

## Switch expression

Ở các phần trên chúng ta đều nói về cấu trúc điều khiển switch-case. Các cấu trúc điều khiển trong C# đều là các lệnh (statement).

Ngôn ngữ C# phân biệt **lệnh** (statement) với **biểu thức** (expression). Lệnh không trả về kết quả. Biểu thức trả về kết quả.

Các cấu trúc if-else, switch-case đều là các lệnh. Chúng không trả về kết quả. Tuy nhiên biểu thức điều kiện `a ? b : c` lại là một biểu thức vì nó trả lại kết quả (giá trị b hoặc c) tùy thuộc vào điều kiện a.

C# 8 đưa vào một loại biểu thức mới: biểu thức switch. Đây là cấu trúc rẽ nhiều nhánh (như switch) nhưng trả lại kết quả.

Hãy cùng xem một số ví dụ về switch expression.

```
1.  private static string Greeting(string name) {
2.      var greeting = name switch
3.      {
4.          "Putin" => "Privet Vova!",
5.          "Elizabeth" => "Your Majesty!",
6.          "Trump" => "Hello, Mr. president!",
7.          _ => $"Hi, {name}!"
8.      };
9.      return greeting;
10. }
```

Trong ví dụ trên bạn đã sử dụng một biểu thức switch để ánh xạ đầu vào (name) thành một biến đầu ra (greeting). Tùy vào giá trị đầu vào, giá trị đầu ra sẽ khác nhau.

Bạn có thể thấy trong biểu thức switch không có các nhánh "case". Thay vào đó là các cặp `pattern => giá trị`. Mỗi cặp này được gọi là một **arm**. Mỗi arm là một trường hợp đặc biệt trong quá trình ánh xạ.

Trong ví dụ nhỏ trên chúng ta đã sử dụng constant pattern quen thuộc.

Riêng arm cuối cùng, `_ => $"Hi, {name}!"`, được gọi là discard pattern. Nó hoạt động giống như default case trong lệnh switch thông thường.

Hãy xem một ví dụ khác:

```
1.  private static double Area(object shape) {
2.      var area = shape switch
3.      {
4.          Rectangle r => r.Width * r.Height,
5.          Cirle c => Math.PI * c.Radius * c.Radius,
6.          Square s => s.Length * s.Length,
```

```
            _ => double.NaN
        };
        return area;
    }
```

Ở đây chúng ta gặp lại type pattern. Ví dụ trên minh họa cách tính diện tích một object mà chúng ta không xác định được từ trước. Thay vì truyền một object của một hình cụ thể, chúng ta truyền object thuộc kiểu chung nhất, kiểu `object`.

Cấu trúc switch expression sẽ so khớp với từng kiểu đã biết để tính diện tích. Nếu object thuộc về một kiểu khác biệt với 3 loại hình chúng ta đã biết thì sẽ trả về giá trị không xác định `double.NaN`.

> Trong switch expression bạn có thể sử dụng tất cả các loại pattern đã biết trong C#.

Ngoài ra, C# 8 đưa thêm vào một số pattern mới để sử dụng cùng với switch expression:

- Positional pattern
- Property pattern
- Tuple pattern

## Property pattern

Hãy cùng xem một ví dụ:

```
private static string Position(Point point) {
    return point switch
    {
        { X: 0, Y: 0 } => "At the origin",
        { X: _, Y: 0 } => "On the X axis",
        { X: 0, Y: _ } => "On the Y axis",
        { X: var x, Y: var y } => $"({x} {y})",
        _ => "Somewhere"
    };
}
```

Trong đó class Point được định nghĩa như sau:

```
class Point {
    public double X { get; set; }
    public double Y { get; set; }
}
```

Trong ví dụ này, phương thức Position nhận một object kiểu Point. Tùy thuộc vào giá trị của tọa độ X và Y sẽ trả lại những thông báo khác nhau. Nếu X = 0, Y = 0 thì báo "nằm ở gốc tọa độ"; Nếu Y = 0 thì báo "nằm trên trục X; Nếu X = 0 thì báo "nằm trên trục Y; Trong những trường hợp còn lại thì in ra tọa độ ở dạng (X, Y).

Đây là một ví dụ về cách sử dụng property pattern trong C# 8.

Trong property pattern, đặc điểm nhận dạng của mỗi pattern chính là danh sách giá trị của các public property của object.

Trong ví dụ trên, mỗi object Point có hai publich property X và Y. Các tổ hợp khác nhau của X và Y có thể dùng để phân biệt các object khác nhau của Point. Ví dụ, X = 0 và Y = 0; X = 0 và Y bất kỳ; X bất kỳ và Y = 0; X bất kỳ và Y bất kỳ.

Để biểu diễn property pattern, bạn sử dụng cặp dấu {}. Trong cặp dấu này chứa các tổ hợp tên của property và giá trị của nó phân tách nhau bởi dấu hai chấm.

Giá trị của property có thể là hằng số hoặc biến số.

- Trường hợp là hằng số, bạn đặt thẳng hằng số sau dấu hai chấm, ví dụ `X:0`, `Y:0`.
- Nếu giá trị là biến số (để về sau sử dụng trong biểu thức), bạn đặt tên biến cùng từ khóa var, ví dụ `X: var x`, `Y: var y`.
- Nếu không quan tâm đến giá trị, bạn dùng ký tự discard `_`. Giá trị discard có nghĩa là bạn không quan tâm giá trị đó bằng bao nhiêu, và bạn cũng không có ý định sử dụng nó.

## Tuple pattern

Tuple pattern là loại pattern dựa trên sử dụng một kiểu dữ liệu đặc biệt của C#: tuple.

Tuple là kiểu dữ liệu kết hợp nhiều dữ liệu theo thứ tự. Ví dụ, (string, string, int) là một tuple với 3 giá trị theo thứ tự lần lượt là string, string và int.

Các giá trị thành viên của tuple có thể được đặt tên. Ví dụ, (string fname, string lname, int age). Tuple này và (string, string, int) là tương đương nhau.

Nếu không đặt tên, các thành viên của tuple sẽ được tự động đặt tên là Item1, Item2, v.v..

Hãy cùng xem ví dụ sau:

```
1.  private static string Position(int x, int y) {
2.      return (x, y) switch
3.      {
4.          (0, 0) => "At the origin",
5.          (_, 0) => "On the X axis",
6.          (0, _) => "On the Y axis",
7.          (var a, var b) => $"({a} {b})"
8.      };
9.  }
```

Đây là cách dùng switch expression để viết ra vị trí của một điểm dựa trên tọa độ. Thay vì sử dụng lớp Point như trước, giờ chúng ta sử dụng tuple (int, int) để mô tả tọa độ.

(x, y) là một biến tuple có kiểu (int, int) và là biến đầu vào cho switch expression.

Trong switch expression, mỗi tổ hợp giá trị của tuple trở thành đặc điểm nhận dạng của nó.

Ví dụ tuple (bool, bool) có thể tạo ra các tổ hợp giá trị (true, true), (true, false), (false, true), (false, false) phân biệt nhau. Tuple (int, int) có thể tạo ra vô số tổ hợp giá trị phân biệt nhau.

Tương tự như đối với property pattern, mỗi giá trị trong tuple có thể là hằng số, là ký tự discard _, hoặc là biến. Chúng ta đã thấy cách sử dụng cả ba loại giá trị này trong tuple ở ví dụ trên.

## Kết luận

Trong bài học này chúng ta đã xem xét chi tiết vấn đề sử dụng pattern và pattern matching trong cấu trúc điều khiển switch-case và biểu thức switch của C#.

Pattern matching giúp viết code ngắn gọn, súc tích và dễ đọc hơn rất nhiều.

Các phiên bản sau này của C# hỗ trợ pattern ngày càng tốt hơn trong xu hướng chuyển dịch lại gần lập trình hàm.