**TÀI LIỆU KĨ THUẬT**

**BÁO CÁO TÌM HIỂU VỀ NHỮNG ĐIỀU MỚI TRONG C#12 VÀ .NET8**

**Phần I: C sharp 12( C#12)**

1. Những tính năng mới
   1. Hàm tạo chính ( Primary Constructors)

Với phiên bản này, chúng ta có thể thiết kế các hàm tạo chính trong bất kì class( lớp) và struct( cấu trúc) nào, nó cũng không bị giới hạn bởi các dạng record( bản ghi). Các tham số của hàm tạo chính và giá trị có thể dùng trong toàn bộ phạm vi của lớp. Và để sử dụng thì chúng ta phải sử dụng cú pháp this().

Khi thêm hàm tạo chính vào một lớp sẽ ngăn việc tạo ra một hàm không tham số ngầm định khi biên dịch chương trình. Trong một cấu trúc, hàm này sẽ tạo ra tất cả các trường, bao gồm cả tham số của hàm tạo chính thành 0-bit.

Trình biên dịch sẽ tạo ra các thuộc tính công khai( public) cho các tham số của hàm tạo chính bên trong các loại bản ghi( record type), các lớp bản ghi( record class) hoặc các cấu trúc của bản ghi(record struct). Các lớp và cấu trúc không bản ghi có thể không luôn luôn muốn có các hành động, phản ứng đối với các tham số của cấu trúc chính.

Ví dụ:

public class Widget(string name, int width, int height, int depth) : NamedItem(name)

{

public Widget() : this("N/A", 1,1,1) {}

public int WidthInCM => width;

public int HeightInCM => height;

public int DepthInCM => depth;

public int Volume => width \* height \* depth;

}

Bất kỳ hàm tạo nào được viết rõ ràng đều phải sử dụng cú pháp khởi tạo this() để gọi hàm tạo chính. Điều đó đảm bảo rằng các tham số của hàm tạo chính chắc chắn được gán bởi tất cả các hàm tạo.

* 1. Biểu thức tập hợp ( Collection expressions)

Biểu thức tập hợp là tính năng đã được giới thiệu ở phiên bản C#11, là một cú pháp ngắn gọn mới để tạo danh sách giá trị. Toán tử trải rộng “..” dùng thay thế các đối số bằng các phần tử từ chính tập hợp đó.

Ví dụ:

int[] row0 = [1, 2, 3];

int[] row1 = [4, 5, 6];

int[] row2 = [7, 8, 9];

int[] single = [.. row0, .. row1, .. row2];

foreach (var element in single)

{

Console.Write($"{element}, ");

}

// output:

// 1, 2, 3, 4, 5, 6, 7, 8, 9,

* 1. Tham số chỉ đọc( ref readonly)
     1. C# đã thêm tham số “in” như một cách để chuyển các tài liệu thành chỉ đọc, nó cho phép cả biến( variables) và giá trị( values) có thể được dùng mà không cần bất kì đối số nào. Việc bổ sung tham số “ref readonly” cho phép rõ rang việc API có thể dùng tham số “ref” hoặc “in”.
     2. Các API đã được tạo trước khi “in” được giới thiệu có thể sử dụng “ref” mặc dù đối số không được sửa đổi. Những API này có thể được cập nhật với “ref readonly”. Tuy nhiên đó không phải điểm đột phá , cũng giống như việc tham số “ref” thay đổi thành “in”.
     3. Các API nhận tham số “in” , tuy nhiên về mặt yêu cầu một biến một cách logic, một biểu thức giá trị có thể không hoạt động. Ví dụ: [System.ReadOnlySpan<T>.ReadOnlySpan<T>(T)](https://learn.microsoft.com/en-us/dotnet/api/system.readonlyspan-1.-ctor#system-readonlyspan-1-ctor(-0@)).
     4. Các API dùng “ref” bởi chúng yêu cầu một biến nhưng không được thay đổi nó
  2. Tham số mặc định lambda

Bây giờ ta có thể xác định giá trị mặc định cho tham số trên biểu thức lambda. Cú pháp và quy tắc giống như việc thêm các giá trị mặc định cho các đối số vào bất kỳ phương thức hoặc hàm cục bộ nào.

* 1. Tính năng “Alias any type”

Cho phép sử dụng "using alias" để tạo ra một tên gọi dễ hiểu (semantic alias) cho bất kỳ loại dữ liệu nào, không chỉ là các loại có tên. Điều này giúp làm cho mã nguồn trở nên dễ hiểu hơn và linh hoạt hơn.

* 1. Mảng nội tuyến ( Inline arrays)

Được sử dụng để cải thiện hiệu suất trong ứng dụng mình. Mảng nội tuyến cho phép người phát triển tạo một mảng có kích thước cố định trong kiểu “struct”. Một cấu trúc nhúng có thể cung cấp đặc trưng hiệu suất tương tự như bộ đệm có kích thước cố địng không an toàn.

* 1. Thuộc tính thử nghiệm( Experimental attribute)

Các kiểu, phương thức, hoặc các tập hợp có thể được đánh dấu bằng [System.Diagnostics.CodeAnalysis.ExperimentalAttribute](https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.codeanalysis.experimentalattribute) để biểu thị là một tính năng thử nghiệm. Trình biên dịch sẽ đưa ra một cảnh báo nếu ta truy cập vào một phương thức hoặc một kiểu, loại đã được chú thích với “ExperimentalAttribute”. Tất cả các kiểu bao gồm một tập hợp được đánh dấu với thuộc tính “Experimental” đều mang tính thử nghiệm.

* 1. Thành phần triển khai bộ chặn( Interceptors)

Một bộ chặn là một phương thức có thể khai báo thay thế lệnh gọi thành một phương thức có thể tự chặn lệnh gọi đến chính nó tại thời điểm biên dịch chương trình. Sự thay thế này xảy ra bằng cách yêu cầu thiết bị chặn khai báo vị trí nguồn của lệnh gọi mà nó chặn. Bộ chặn cung cấp một cơ sở hạn chế để thay đổi ngữ nghĩa của mã hiện có bằng cách thêm mã mới vào trình biên dịch. Có thể sử dụng bộ chặn như một phần của trình tạo nguồn để sửa đổi, thay vì thêm mã vào trình biên dịch nguồn hiện có. Trình tạo nguồn thay thế các lệnh gọi đến một phương thức có thể chặn bằng một lệnh gọi đến phương thức chặn.

Lưu ý: đây là tính năng có sẵn trong chế độ xem trước với C#12, và tính năng này có thể bị loại bỏ trong các bản phát hành sau này.

1. Thay đổi mang tính đột phá

Cho phép các đối số “ref” có thể chuyển thành “in” khi “LangVersion” thay đổi lên phiên bản 12 hoặc muộn hơn.

**Phần II: dotNET 8( .NET8)**

1. .NET Aspire
2. ASP.NET Core
3. Thư viện Core .NET
4. Tuần tự hóa( Serialization)
5. Hỗ trợ các dạng bổ sung( Built-in support for additional types)
6. Tạo mã nguồn( Source generator)
7. Chuỗi tạo mã nguồn( Chain source generator)
8. Phân cấp giao diện( Interface hierarchies)