Trong C#, base là một từ khóa đặt biệt được dùng trong các lớp dẫn xuất (lớp con) để tương tác với lớp cơ sơ (lớp cha) của nó

1. bắt buộc khi lớp cha có constructor có tham số, để chọn constructor nào của lớp cha sẽ được gọi hoặc truyền dữ liệu lên lớp cha.

2. Dùng base. khi muốn truy cập thuộc tính của lớp cha.

virtual: cho phép ghi đề nếu muốn, lớp cha có cung cấp sẵn logic mặc định, ko bắt buộc override

abstract: là khai báo trừu tượng, không có nội dung, bắt buộc lớp kế thừa phải override phương thức đó, chỉ dùng được trong lớp abstract.

Parallel.For is indeed built on top of the Task Parallel Library (TPL). So you're using Task indirectly - but in a very optimized and high-level way.

Task uses the .Net ThreadPool unless you explicitly configure it not to.

Luồng nền (Background Thread) là gì?

Luồng nền là một luồng (thread) chạy trong ứng dụng của bạn nhưng nó không giữa cho ứng dụng hoạt động. Nếu tất cả các luồng chính ( foreground thread ) của ứng dụng kết thúc, thì ứng dụng sẽ dừng hoàn toàn, dù luồng nền có đang chạy.

C# .Net Knowledge 3

Entity Framework là một Object-Relational Mapper (ORM) của .NET do Microsoft phát triển. Nó giúp bạn làm việc với cơ sở dữ liệu bằng sử dụng các đối trượng trong code thay vì phải viết trực tiếp câu lệnh SQL.

Tạo một class student (ID, Name)

sau đó dùng LINQ: var students = dbContext.Students.Where(s => s.Name == "John").ToList();

Object-Relation Mapper (ORM): là một công cụ trung gian giữa mô hình hướng đối tượng (OOP) trong lập trình và một hình quan hệ (Relation) của cơ sở dữ liệu=.

Ánh xạ (mapping) giữa class trong code và table trong cơ sở dữ liệu.

Tự động chuyển đổi truy vấn LINQ sang SQL phù hợp để tương tác với database.

Theo dõi thay đổi của object và cập nhật dữ liệu tự động khi gọi SaveChanges()

Data Annotation: là cách gắn attribute (thuộc tính) trực tiếp lên class và property trong C# để mô tả thông tin Entity Framework.

public class Student

{

[Key] // khoa chinh

public int Id {get; set;}

[Required] // khong duoc null

[MaxLength(100)] // Toi da 100 ky tu

public string Name {get; set;}

[Column("BirthDate")] // Dat ten cot la "BirthDate"

public DateTime DateOfBirth {get; set;}

}

Fluent API: là cách cấu hình mô hình bằng code trong OnModelCreating() của DbContext, sử dụng các phương thức gọi nối tiếp (fluent = trôi chảy)

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Student>(entity =>

{

entity.HasKey(s => s.Id); // Khóa chính

entity.Property(s => s.Name)

.IsRequired() // Không null

.HasMaxLength(100); // Giới hạn 100 ký tự

entity.Property(s => s.DateOfBirth)

.HasColumnName("BirthDate"); // Đặt tên cột

});

}

ASP.NET MVC: là một framework của Microsoft dùng để xây dựng ứng dụng web theo mô hình MVC (Model-View-Controller)

Chạy trên nền tảng .Net Framework(Window only), đã bị thay thế bởi ASP.NET Core kể từ 2016

ASP.NET Core: là một framework hiện đại để phát triền ứng dụng web, API, Realtime app. Đa nền tảng, họp chung Web API, MVC, Razor Pages thành một nền tảng duy nhất. Nhanh hơn, nhẹ hơn.

Middleware: là các thành phần trong pipeline xử lý request/reponse của ASP.NET Core.

Đặc điểm: Request đi qua từng middleware theo thứ tự bạn đăng ký.

Mỗi middleware có thể: Xử lý request (xác thực), Gọi middleware tiếp teo (next()) Hoặc chặn requets tại chỗ.

app.Use(async (context, next ) =>

{

Console.WriteLine("Middleware 1: Before");

await next(); // gọi middleware tiếp theo

Console.WriteLine("Middleware 1: After");

}

app.UseMiddleware<YourCustomMiddleware>();

app.UseRouting();

app.UseEndpoints(...);

Action Filter ( hoặc Filter): là cơ chế trong ASP.NET Core MVC để thêm logic trước hoặc sau khi action/controller chạy. Dùng để xác thực, logging, handle lỗi, chuyển hướng.

DI( Dependency Injection ): là cơ chế inject các dependency (phụ thuộc) như service, logger, repository vào controller hoặc class khác thay vì khởi tạo thủ công. (Dễ test, Tách biệt logic, dễ thay đổi implementation.

builder (WebApplicationBuilder) chính là nơi bạn đăng ký và cấu hình các dịch vụ vào Dependency Injection container (DI container)

var builder = WebApplication.CreateBuilder(args);

AddSingleTon<T>: Toàn bộ vòng đời ứng dụng, tạo một instance duy nhất.

AddScoped<T>: Mỗi request mới, Mỗi request tạo một instance.

AddTransient<T>: Mỗi lần gọi, Tạo mới mỗi lần được yêu cầu. (cannot reuse)

Mô hình Thác nước (Waterfall Model): là một trong những mô hình SDLC(Software Development Life Cycle) ( Làm đúng ngay từ đầu )

1. Thu thập và phân tích yêu cầu (Requirements Gathering & Analysis), Thu thập toàn bộ yêu cầu từ khách hàng và ghi lại cho tiết trong tài liệu SRS( Software Requirement Specification)

2. Thiết kế( Design ): Dựa trên tài liệu y/c, thiết kế hệ thống, thiết kế csdl, thiết kế giao diện.

3. Triển khai/ lập trình ( implementation/Coding): viết mã nguồi dựa trên thiết kế.

4. Kiểm thử (Testing): tiến hành kiểm tra để tìm lỗi, đảm bảo đúng yêu cầu.

5. Triển Khai ( Deployment ): Phần mềm được cài đặt và triển khai trên môi trường thực tế, để người dùng sử dụng.

6. Bảo trì (Maintenace): Sau khi triển khai, phần mềm tiếp tục được hỗ trợ, sửa lỗi pháp sinh và cải tiến nhỏ.

Mô hình Agile (Agile Model): mềm dẻo, linh hoạt, Thích ứng với thay đổi và mang lại giá trị nhanh chóng thông qua sự công tác.

1. Lập kế hoạch Sprint (Sprint Planning): Đội ngũ và khách hàng thống nhát các tính năng sẽ phát triển trong sprint này.

2. Thiết kế & phát triển (Design & Development): Lập trình viên và designer làm việc để hiện thực hóa các tính năng đã chọn.

3. Kiểm thử (Testing): Các tính năng được phát triển trong sprint được kiểm thử ngay lập tức.

4. Xem xét Sprint( Sprint Review ): Đội ngũ trình bày sản phẩm hoạt dộng được cho khách hàng và thu thập phản hồi.

5. Hồi cứu Sprint (Sprint Retrospective): Đội ngũ tự đánh giá quy trình làm việc để cải tiến cho sprint tiếp theo.

Sự khác biệt giữa class và struct là gì?

Class: có thể kế thừa, kiểu dữ liệu là reference type, các thành viên là private mặc định, phù hợp với những object lớn, phức tạp, có thể dùng waste collector cho quản lý

Struct: không kế thừa, Value Type, thành viên là public, phù hợp với những model cô lập, không dùng garbage collector.

Sự khác biệt giữa Interface và abstract class là gì?

Interface và abstract class đều dùng để định nghĩa khuôn mẫu cho các lớp con, nhưng khác nhau ở mục đích và cách sử dụng. Interface chỉ chứa khai báo phương thức (không có logic), cho phép đa kế thừa và dùng khi các lớp không liên quan nhưng cần tuân thủ cùng một hợp đồng. Abstract class có thể chứa cả khai báo và cài đặt phương thức, dùng khi các lớp có quan hệ kế thừa, cần chia sẻ logic chung và chỉ hỗ trợ kế thừa đơn. Interface linh hoạt hơn trong việc thiết kế hệ thống mở rộng, còn abstract class phù hợp khi có nhiều điểm chung giữa các lớp.

Interface cho phép đa kế thừa: class Human : IWalk, ITalk {}

T

C# Knowledge 1

Hàm vô danh là hàm không có tên và được sử dụng tại chỗ

List<int> evenNumbers = numbers.FindAll(delegate(int num) // Ham vo danh

{

return n%==0;

}

Ham lambda sử dụng toán tử suy ra (=>)

+ Phía bên trái của => là các tham số đầu vào (giống như tham số của một phương thức)

+ Phía bên phải của => là thân của biểu thức (khối mã sẽ được thực thi)

Biểu thức lambda là cách hiện đại và được khuyển nghị để viết hàm vô danh.

LINQ (Language Integrated Query) là một tính năng mạnh mẽ trong C# dùng để truy vấn dữ liệu giống cú pháp của SQL. (ascending tang dan) (descending giam dan)

var largeNumbers = from num in numbers

where num > 10

orderby num ascending

select num;

Các cấp độ truy cập trong C# (public, private, protected, internal)

Public: Thành viên hoặc kiểu có thể truy cập từ bất cứ đâu trong cùng một assembly(project) hoặc từ assembly khác.

Private: Thành viên hoặc kiểu chỉ có thể được truy cập bởi các thành viên khác bên trong định nghĩa của chính nó. Không thể truy cập từ bên ngoài lớp đó, ngay cả từ các lớp con.

Protected: Thành viên chỉ có thể được truy cập bởi các thành viên bên trong lớp khai báo nó hoặc bởi các thành viên trong các lớp kế thừa từ lớp đó. Không thể truy cập từ bên ngoài hệ thông phân cấp kế thừa.

Internal: Thành viên hoặc kiểu có thể được truy cập bởi bất kỳ mã nào trong cùng một assembly mà nó được khai báo. Không thể truy cập từ các assembly khác,

Từ hạn chế nhất đế không hạn chế nhất là private -> protected -> internal -> public

Dành cho lớp chỉ có public và internal, còn dành cho thành viên của lớp thì có đầy đủ

4 hướng đối tượng:

Tính đóng gói, tính kế thừa, tính đa hình, tính trừu tượng

Tính đóng gói (Encapsulation): Là việc gói gọn dữ liệu và hành vi(phương thức) liên quan vào một đơn vị duy nhất, và đồng thời ẩn đi các chi tiết triển khai bên trong.

+ Đặc điểm: Bảo vệ dữ liệu, ngăn truy cập, giảm sự phụ thuộc, dùng private cho dữ liệu, và Properties với get/set để kiểm soát truy cập.

Tính kế thừa (Inheritance): Cho phép một lớp mới(lớp con) nhận tất cả các thuộc tính và phương thức từ một lớp hiện có (lớp cha). Lớp con có thể tái sử dụng, mở rộng hoặc đổi hành vi đã kế thừa.

+ Đặc điểm: Tái sử dụng mã, Quan hệ "is-a": Biểu thị mối quan hệ phân cấp (Dog is a Animal), Thêm tính năng mới hoặc ghi đè cái đã có. Dùng dấu : và từ khóa base, virtual, override.

Tính đa hình (Polymorphism): Khả năng các đối tượng khác nhau (thuộc các lớp liên quan) có thể phản ứng khác nhau với cùng một thông điệp/ lời gọi phương thức ( nhiều hình thức )

+ Đặc điểm: Linh hoạt mã, Cho phép viết code tổng quan hơn, xử lý nhiều loại đối tượng khác nhau qua cùng một giao diện, thêm lớp mới không cần sửa code cũ, Dùng ghi đè phương thức( virtual/ override), abstract class, Interface và nạp chồng phương thức.

Tính trừu tượng (Abstraction): Là quá trình đơn giản hóa những gì phức tạp bằng cách hiển thị những thông tin cốt lõi, cần thiết và ẩn đi các chi tiết triển khai không quan trọng.

Tập trung vào "cái gì" một đối tượng làm, thay vì "làm như thế nào"

+ Đặc điểm: Giảm độ phức tạp, chỉ cần biết cách dùng ko cần biết bên trong, tăng tính dễ hiểu, bảo trì, Dùng Abstract Classes và Interfaces là chính, kết hợp với các access modifies ( private, protected)

Virtual: dùng để đánh dấu một thành viên (phương thức, thuộc tính, sự kiện,..) trong lớp cha cho phép các lớp con có thể ghi đè ( override ) lại hành vi của thành viên đó.

Đặc điểm: Nền tảng của tính đa hình, cho phép đối tượng thực tế quyết định hành vi. Nếu lớp con không ghi đè, nó sẽ dùng hành vi mặc định của lớp cha.

Sealed: dùng để ngăn chặn việc kế thừa hoặc ngăn chặn việc ghi đè thêm.

Đặc điểm: Khi áp dụng một phương thức đã được ghi đè (override): các lớp con xa hơn sẽ không thể ghi đè lại phương thức đó nữa, đảm bảo tính toàn vẹn của thiết kế, ngăn chặn sự thay đổi không mong muốn trong hệ thống phân cấp lớp.

C# Knowledge

Sự khác biệt giữa System.Boolean và bool ?

Dùng bool ( tên thân mật, đơn giản ngắn gọn) 99 % dùng này

Dùng System.Boolean ( tên đầy đủ, kiểm tra kiểu dữ liệu )

Sự khác biệt giữa dynamic và var ?

Var: trình biên dịch sẽ đoán kiểu dữ liệu khi khai báo, var test = "string" (kiểu string)

Dynamic: trình biên dịch sẽ không đoán, nhưng khi chạy sẽ báo lỗi nếu ko đúng kiểu dynamic test = "string" - chạy test = 10 lỗi. nhưng code ko hiện lỗi còn var thì ngược lại. Dùng khi cần tương tác API dạng data json, khi dữ liệu nhận là động hoặc với các ngôn ngữ dynamic như python hoặc Ruby.

Compile-time error: lỗi biên dịch ( khi code, dịch sang mã máy)

Run-time error: khi chạy ( lỗi ngoại lệ)

Reference Type: Kiểu tham chiếu

(Class, Interface, Array, Delegate) Class, string, object, List, IEnumerable, interface, int[], string[], Action, Func, custom delegate, Queue, Dic.

Value Type: Kiểu dữ liệu

(Numberic, Logic, struct, Enum) int, double, byte, bool, char, Datatime, Guid, custom struct, enum.

Sự khác biệt giữa ToString và $'' ?

Dấu $ sẽ ngắn gọn hơn và nếu kết hợp với {} mình có thể bỏ nhiều giá trị vào 1 chuỗi.

Sự khác biệt giữa string và StringBuilder ?

string: bất biến, không thể thay thế được, khi nối chuỗi, thay thế ký tự, cắt chuỗi con thuộc chất .Net sẽ tạo ra một đối tượng string Mới trong bộ nhớ lưu nội dung đã thay đổi.

StringBuilder: cấp phát một vùng nhớ đệm (buffer) và cho phép bạn thay đổi nội dung của chuỗi ngay tại chỗ( hiệu xuất tốt hơn, tiết kiệm bộ nhớ hơn)

int (kiểu giá trị không thể null - Non-Nullable Value Type) Runtime error khi giá trị null

int? (kiểu giá trị có thể null - Nullable Value Type)

Rỗng(Empty) có nghĩa là có một đối tượng nhưng không có giá trị bên trong (nồi cơm không có cơm)

Null: có nghĩa nó không trỏ vào đâu ko đối tượng ( không có cả nồi cơm, không tồn tại)