

[Design Pattern] – Dependency Injection trong ASP.NET Core

Dependency Injection là một kỹ thuật vô cùng thông dụng để *nói lỏng* các objects và class phụ thuộc vào chúng. Nghe khó hiểu quá phải ko các bạn?

Bài viết này sẽ mô tả kỹ thuật này, cho bạn một cái nhìn tổng quan (hy vọng là khách quan) về DI

- 1. Vấn đề
- 2. Lợi ích
- 3. Back to code
 - 3.1. Interface
 - 3.2. Implementation
- 4. Register
 - 4.1. Lifetime
 - 4.2. Constructor Injection
 - 4.3. Action Injection
 - 4.4. Service trong Service

1. Vấn đề

- 1 | Hãy tưởng tượng bạn muốn uống coca, nhà ko còn chai coca nào
- 2 |
- 3 | Bạn phải ra Circle K, đi lòng vòng trong cửa hàng để kiểm 1 chai coca, trả tiền, đi về

- Bạn có thể quên ko trả tiền

- Bạn có thể ko tìm thấy Circle K nào gần cả
- Bạn có thể ko tìm thấy chai coca nào trong circle k cả
- Bạn có thể lấy nhầm 1 chai xá xì chương dương thay vì coca

Với Dependency Injection, mọi việc sẽ khác đi

- 1 | Bạn đưa tiền cho 1 thằng nhóc có nhiệm vụ chuyên đi mua nước ngọt cho khu xóm
- 2 |
- 3 | 5 phút sau, nó xuất hiện với chai coca của bạn

Một người khác trong cùng khu xóm cũng có nhu cần gần giống bạn, nhưng họ muốn uống pepsi

Thay vì cũng phải đi ra cửa hàng và tự mua pepsi và gặp các vấn đề như bạn gặp, họ cũng gọi thằng nhóc đó lại, và 5 phút sau, chai pepsi ướp lạnh ở trong tay họ.

Thằng nhóc đó chính là 1 *Service*, và bạn phụ thuộc vô thằng nhóc đó để có nước ngọt uống

2. Lợi ích

Quay trở lại với ví dụ trên, lợi ích của bạn khi dùng *Service* mua nước ngọt của thằng nhóc là gì?

- Bạn ko cần quan tâm thằng nhóc đó nó làm gì để có nước ngọt cho bạn
- Bạn có thể yêu cầu nhiều loại nước ngọt khác nhau mà ko cần biết hình dạng hay chỗ bán

3. Back to code

Quay trở lại với code, bạn sẽ implement DI như thế nào?

3.1. Interface

```
1 public interface IDrinkBuyer
2 {
3     void BuyDrink(string name);
4 }
```

Khi thằng nhóc mua nước ngọt ở bên trên đã già, nó sẽ muốn truyền lại nhiệm vụ mua nước ngọt cho 1 thằng nhóc khác, và cứ thế.

Mọi thằng nhóc mua nước ngọt trong xóm đều sẽ có 1 phương thức cơ bản là `BuyDrink` chấp nhận 1 param là tên của loại nước cần mua

Bạn đóng vai trò là `WebApplication`, sẽ gọi phương thức này để có nước ngọt uống

3.2. Implementation

Người quản lý của mấy thằng nhóc mua nước ngọt này sẽ phân nhiệm vụ cho từng thằng nhóc 1

- Nhóc A mua ở Circle K
- Nhóc B mua ở Vinmart

Và họ sẽ implement như sau

```
1 public class CircleKDrinkBuyer : IDrinkBuyer
2 {
3     public void BuyDrink(string name)
4     {
5         // Go to circle k
6
7         // Find the -name- drink
8
9         // Buy it and deliver
10    }
11 }
12
13 public class VinmartDrinkBuyer : IDrinkBuyer
14 {
15     public void BuyDrink(string name)
16     {
17         // Go to Vinmart
18
19         // Find the -name- drink
20
21         // Buy it and deliver
22    }
23 }
```

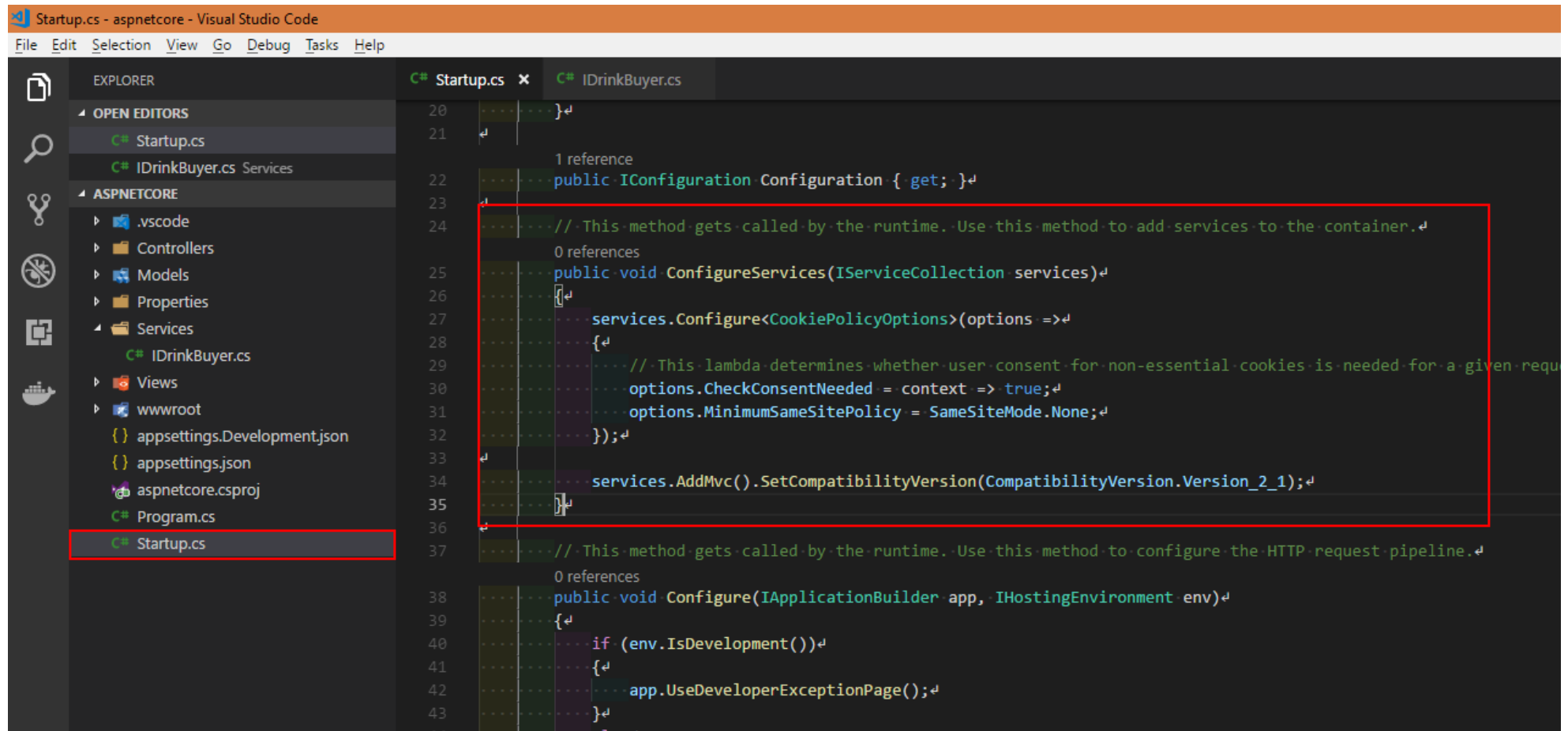
Thế là xong

Khi nhà bạn gần Vinmart, người quản lý sẽ cử thằng nhóc B – VinmartDrinkBuyer canh trước cổng nhà bạn. Bất kể khi nào bạn có nhu cầu mua nước ngọt, bạn sẽ gọi nó. Bạn đâu có biết là thằng nhóc B đó nó chỉ biết mua nước ngọt ở Vinmart thôi. Bạn chỉ quan tâm là bạn gọi nó, và bạn có nước ngọt uống

4. Register

Như vậy, khai báo như thế nào trong ứng dụng của bạn?

khi tạo mới một ứng dụng asp.net core 2, đã có sẵn 1 số phương thức giúp bạn bắt đầu ngay và luôn



bằng cách thêm vào dòng code sau

```
1 | services.AddScoped();
```

bạn đã khai báo cho tất cả các class có dùng phương thức BuyDrink sẽ mua nước ngọt ở CircleK

4.1. Lifetime

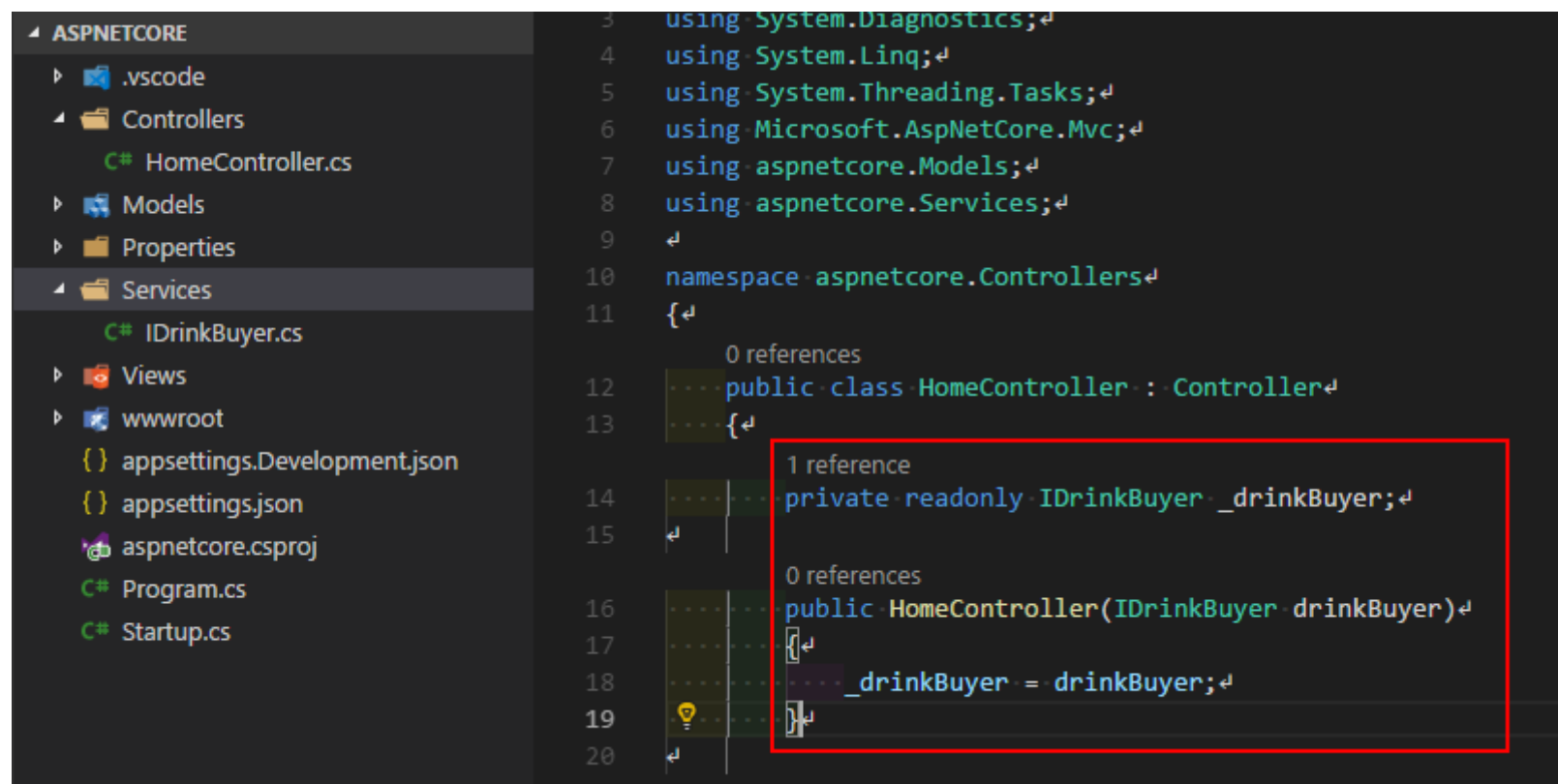
AddScoped dùng để chỉ định *thời gian sống* của service này, có 3 loại:

- Transient: mỗi lần gọi tới nó sẽ là 1 biến thể mới hoàn toàn. Cái này chỉ phù hợp cho các loại service siêu nhẹ nhàng và không có lưu trữ trạng thái
- Scoped: Trong cùng 1 request, nó sẽ chỉ được tạo 1 lần duy nhất
- Singleton: Nó sẽ được tạo lần đầu tiên khi được gọi, và sẽ sống mãi cho tới khi app bạn chết

4.2. Contructor Injection

Sau khi đã register service của bạn với 1 lifetime phù hợp, việc tiếp theo chính là *tiêm – inject* nó vào chỗ cần dùng (thường sẽ là controller)

ASP.NET Core 2 hỗ trợ bạn inject service vào controller thông qua constructor của controller đó



The screenshot shows the Visual Studio IDE with the ASP.NET Core project structure on the left and the code for HomeController.cs on the right. The project structure includes folders for .vscode, Controllers, Models, Properties, Services, Views, and wwwroot. The Services folder contains IDrinkBuyer.cs. The code for HomeController.cs is as follows:

```
3 using System.Diagnostics;
4 using System.Linq;
5 using System.Threading.Tasks;
6 using Microsoft.AspNetCore.Mvc;
7 using aspnetcore.Models;
8 using aspnetcore.Services;
9
10 namespace aspnetcore.Controllers
11 {
12     0 references
13     ... public class HomeController : Controller
14     ... {
15
16     1 reference
17     ... private readonly IDrinkBuyer _drinkBuyer;
18
19     0 references
20     ... public HomeController(IDrinkBuyer drinkBuyer)
21     {
22         ... _drinkBuyer = drinkBuyer;
23     }
24 }
```

4.3. Action Injection

Đôi khi bạn chỉ cần cái service này trong 1 action cụ thể nào đó của controller thôi, thì sẽ có cách hơi khác

```
1 public IActionResult About([FromServices] IDrinkBuyer drinkBuyer)
2 {
3     drinkBuyer.BuyDrink("coxa");
4     return View();
5 }
```

4.4. Service trong Service

Nested services, service con, vân vân và mây mây

Đôi khi bạn cần phải có 1 service con để service cha chạy được, thì inject như nào

Rất đơn giản, cũng dùng constructor injection thôi

Giả sử Service DrinkBuyer cần có service Trasport để chạy

```
1 public interface ITransport
2 {
3     void Transport();
4 }
5
6 public class BikeTransport : ITransport
7 {
8     public void Transport()
9     {
10         // transport by bike
11     }
12 }
13
14 public class CarTransport : ITransport
15 {
16     public void Transport()
17     {
18         // transport by car
19     }
20 }
```

đầu tiên đăng ký nó trong Startup.cs

```
1 services.AddScoped();
```

sau đó bạn inject nó vào DrinkBuyer như sau


```
1 public class CircleKDrinkBuyer : IDrinkBuyer
2 {
3     private readonly ITransport _transport;
4
5     public CircleKDrinkBuyer(ITransport transport)
6     {
7         _transport = transport;
8     }
9
10    public void BuyDrink(string name)
11    {
12        // Go to circle k
13        // Find the -name- drink
14        // Buy it
15        // Deliver
16        _transport.Transport();
17    }
18 }
```

thế này xong 😊

TAGGED ASP.NET, DEPENDENCY INJECTION, DESIGN PATTERN



Published by hunter.tran

Former Senior Software Engineer at NashTech Vietnam [View all posts by hunter.tran](#)

2 thoughts on “[Design Pattern] – Dependency Injection trong ASP.NET Core”

1. Pingback: [Repository và Unit of Work Pattern | Tuan Tran's Blog](#)
2. **Võ Phan Hồng Dũng** says:
[13/06/2019 AT 9:56 PM](#)

Bài viết hay và thực tiễn !

REPLY

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

POWERED BY WORDPRESS.COM.