AddSingleton vs AddScoped vs AddTransient

Suggested Videos

Part 41 - ASP.NET Core Model Binding | Text | Slides
Part 42 - ASP.NET Core model validation | Text | Slides
Part 43 - Select list validation in asp.net core | Text | Slides

In this video we will discuss the **differences between AddSingleton(), AddScoped() and AddTransient()** methods in ASP.NET Core with an example.

IEmployeeRepository interface

Consider the following IEmployeeRepository interface. *Add()* method adds a new employee to the repository. *GetAllEmployees()* method returns all the Employees in the repository.

```
public interface IEmployeeRepository
{
    IEnumerable<Employee> GetAllEmployees();
    Employee Add(Employee employee);
}
```

PRAGIM TECHNOLOGIES Training + Placements

Best software training and placements in marathahalli, bangalore. For further details please call 09945699393.

Complete Tutorials

JavaScript tutorial

Bootstrap tutorial

Angular tutorial for beginners

Angular 5 Tutorial for beginners

Important Videos

The Gift of Education

Web application for your business

Employee Class

```
public class Employee
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

MockEmployeeRepository

MockEmployeeRepository implements IEmployeeRepository. To keep the example simple we are storing the list of employees in-memory in a private field _employeeList.

```
public class MockEmployeeRepository : IEmployeeRepository
{
    private List<Employee> _employeeList;

    public MockEmployeeRepository()
    {
        _employeeList = new List<Employee>()
    {
        new Employee() { Id = 1, Name = "Mary" },
        new Employee() { Id = 2, Name = "John" },
        new Employee() { Id = 3, Name = "Sam" },
    };
    }

    public Employee Add(Employee employee)
    {
        employee.Id = _employeeList.Max(e => e.Id) + 1;
        _employeeList.Add(employee);
        return employee;
    }

    public IEnumerable<Employee> GetAllEmployees()
    {
        return _employeeList;
    }
}
```

How to become .NET developer

Resources available to help you

Dot Net Video Tutorials

ASP.NET Core Tutorial

Angular 6 Tutorial

Angular CRUD Tutorial

Angular CLI Tutorial

Angular 2 Tutorial

Design Patterns

SOLID Principles

ASP.NET Web API

Bootstrap

AngularJS Tutorial

jQuery Tutorial

JavaScript with ASP.NET Tutorial

JavaScript Tutorial

Charts Tutorial

LINQ

LINQ to SQL

LINQ to XML

Entity Framework

HomeController

IEmployeeRepository is injected in to the HomeController. The *Create()* action method that responds to the *POST* request, uses the injected instance to add the employee object to the repository.

```
public class HomeController : Controller
{
    private IEmployeeRepository _employeeRepository;

    public HomeController(IEmployeeRepository employeeRepository)
    {
        _employeeRepository = employeeRepository;
    }

    [HttpGet]
    public ViewResult Create()
    {
        return View();
    }

    [HttpPost]
    public IActionResult Create(Employee employee)
    {
        if (ModelState.IsValid)
        {
            Employee newEmployee = _employeeRepository.Add(employee);
        }

        return View();
    }
}
```

Create View

We are injecting IEmployeeRepository service in to the *Create* view using @inject directive. We are using the injected service to display the total number of employees in the repository.

WCF

ASP.NET Web Services

Dot Net Basics

C#

SQL Server

ADO.NET

ASP.NET

GridView

ASP.NET MVC

Visual Studio Tips and Tricks

Dot Net Interview Questions

Slides

Entity Framework

WCF

ASP.NET Web Services

Dot Net Basics

C#

SQL Server

ADO.NET

ASP.NET

```
@model Employee
@inject IEmployeeRepository empRepository

<form asp-controller="home" asp-action="create" method="post">
    <div>
        <label asp-for="Name"></label>
        <div>
            <input asp-for="Name">
        </div>
    </div>

    <div>
        <button type="submit">Create</button>
    </div>

    <div>
        Total Employees Count = @empRepository.GetAllEmployees().Count().ToString()
    </div>
</form>
```

Registering Services

ASP.NET core provides the following 3 methods to register services with the dependency injection container. The method that we use determines the lifetime of the registered service.

AddSingleton() - As the name implies, **AddSingleton()** method creates a Singleton service. A Singleton service is created when it is first requested. This same instance is then used by all the subsequent requests. So in general, a Singleton service is created only one time per application and that single instance is used throughout the application life time.

AddTransient() - This method creates a Transient service. A new instance of a Transient service is created each time it is requested.

AddScoped() - This method creates a Scoped service. A new instance of a Scoped service is created once per request within the scope. For example, in a web application it creates 1 instance per each http request but uses the same instance in the other calls within that same web request.

GridView

ASP.NET MVC

Visual Studio Tips and Tricks

Java Video Tutorials

Part 1 : Video | Text | Slides

Part 2 : Video | Text | Slides

Part 3 : Video | Text | Slides

Interview Questions

C#

SQL Server

Written Test

In ASP.NET Core, services are registered in **ConfigureServices()** method of the **Startup.cs** file.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddSingleton<IEmployeeRepository, MockEmployeeRepository>();
}
```

1. At the moment we are using **AddSingleton()** method to register MockEmployeeRepository service.
2. **AddSingleton()** creates a single instance of the service when it is first requested and reuses that same instance in all the places where that service is needed.
3. This means all the requests throughout the life time of the application use that same instance.
4. At the moment, in our example we need MockEmployeeRepository service instance in 2 places - *Create* action method in the HomeController and in the *Create* view.
5. At this point when we navigate to http://localhost:/home/create we see the Total Employees Count is 3
6. To serve this request an instance of the HomeController is created first. HomeController has a dependency on IEmployeeRepository.
7. This is the first time, the instance of the service is requested. So asp.net core creates an instance of the service and injects it into the HomeController.
8. *Create* view also needs an instance of the service to calculate the total number of employees. With singleton, the same service instance is used. So the instance of the service that is already created is provided to the *Create* view also.
9. Now if you provide a Name in the Name textbox and click *Create* button you will see the count goes up every time you click the button.
10. This is because with Singleton, the same object is used, so changes made to the object can be viewed in all the places across all the HTTP requests.

AddScoped

Now, use **AddScoped()** method to register the service.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddScoped<IEmployeeRepository, MockEmployeeRepository>();
}
```

1. Issue a request to http://localhost:/home/create.
2. We see the Total Employees Count is 3.
3. Provide a Name and click the *Create* button.
4. The Total Employees Count increases to 4.
5. When we click the *Create* button again, the Total Employees Count is still 4.

This is because for a scoped service with every http request we get a new instance. However, with in the same http request if the service is required in multiple places like in the view and in the controller then the same instance is provided for the entire scope of that http request.

If we relate this to our example, both the HomeController and the *Create* view will use the same instance of the service for a given http request. This is the reason *Create* view is also able to see the new employee added by the *Create* action method of the HomeController. Hence we see the Total Employees Count as 4.

But every new http request will get a new instance of the service. This is the reason an employee added by one http request cannot be seen in another http request. So this means every time we click the *Create* button we are issuing a new http request and hence the Total Employees Count does not go beyond 4.

AddTransient

Finally let's use **AddTransient()** method to register our service

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddTransient<IEmployeeRepository, MockEmployeeRepository>();
}
```

1. Issue a request to http://localhost:/home/create
2. We see the Total Employees Count is 3
3. Provide a Name and click the *Create* button.
4. Notice, on the *Create* view, we still see the Total Employees Count as 3

This because with a transient service a new instance is provided every time a service instance is requested whether it is in the scope of the same http request or across different http requests.

Since a new instance is provided even within the same scope of a given http request, the *Create* view is not able to see the new employee added by the *Create* action of the HomeController. This is the reason the count of employees is 3 even after adding a new employee.

Scoped Service vs Transient Service vs Singleton Service

The following is the main difference between a scoped service and a transient service.

With a scoped service we get the same instance within the scope of a given http request but a new instance across different http requests

With a transient service a new instance is provided every time an instance is requested whether it is in the scope of the same http request or across different http requests

With a singleton service, there is only a single instance. An instance is created, when the service is first requested and that single instance is used by all http requests throughout the application.

WWW.PRAGIMTECH.COM
CLICK HERE FOR THE FULL ASP.NET CORE TUTORIAL
facebook.com/pragimtech | twitter.com/kudvenkat

1 comment:

Shariyar Shourov April 10, 2019 at 7:41 PM

Outstanding Sir!

Reply

Enter your comment...

Comment as: toilati123vn@g ▼

Publish Preview

Links to this post

Create a Link

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)