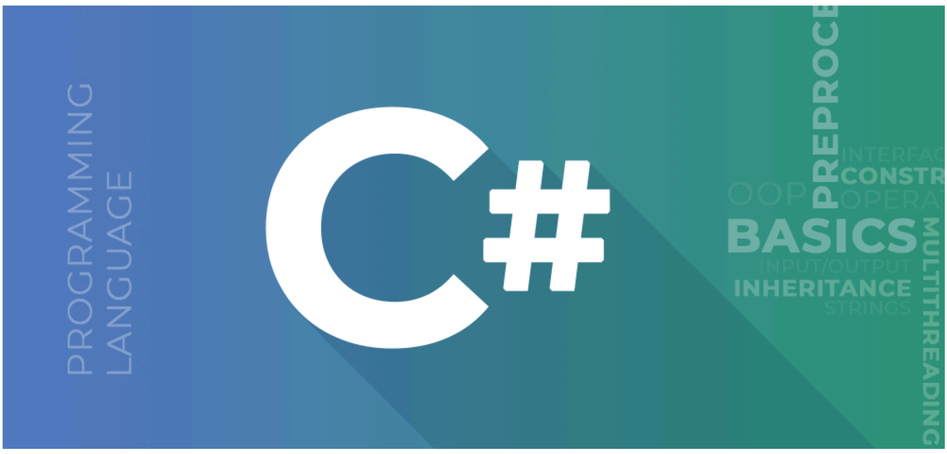
.net中经常看到通过Action来进行Options配置参数的传递，你知道是怎么实现的吗？

作者：微信公众号：【架构师老卢】 11-13 8:58 124



概述：在.NET Core中，使用Action和Options参数方式配置服务并将配置信息对象注册到IServiceCollection的好处在于，它提供了更高级别的可配置性和可扩展性。这种模式允许将配置信息与服务的实现分离，使配置更加模块化和可管理。通过将配置信息对象注册到IServiceCollection，可以轻松将其注入到需要的服务中，从而使配置信息对整个应用程序都可用。以下是如何配置邮件发送服务并将配置信息对象注册到IServiceCollection的示例：首先，让我们创建一个配置信息对象EmailServiceOptions，用于定义邮件发送的配置选项：using System;pu

在.NET Core中，使用`Action`和`Options`参数方式配置服务并将配置信息对象注册到`IServiceCollection`的好处在于，它提供了更高级别的可配置性和可扩展性。这种模式允许将配置信息与服务的实现分离，使配置更加模块化和可管理。通过将配置信息对象注册到`IServiceCollection`，可以轻松将其注入到需要的服务中，从而使配置信息对整个应用程序都可用。

以下是如何配置邮件发送服务并将配置信息对象注册到`IServiceCollection`的示例：

首先，让我们创建一个配置信息对象 `EmailServiceOptions`，用于定义邮件发送的配置选项：

```
using System;

public class EmailServiceOptions
{
    public string SmtpServer { get; set; }
    public int SmtpPort { get; set; }
    public string SenderEmail { get; set; }
    public string SenderPassword { get; set; }
}
```

接下来，我们将创建一个邮件发送服务 `EmailService`，它使用 `EmailServiceOptions` 作为配置参数，并将其注册到 `IServiceCollection`：

```
using System;
using System.Net;
using System.Net.Mail;

public class EmailService
{
    private readonly EmailServiceOptions _options;

    public EmailService(EmailServiceOptions options)
    {
        _options = options;
    }

    public void SendEmail(string to, string subject, string message)
    {
        using (var client = new SmtpClient(_options.SmtpServer, _options.SmtpPort))
        {
            client.Credentials = new NetworkCredential(_options.SenderEmail, _options.SenderPassword);
            client.EnableSsl = true;

            var mail = new MailMessage(_options.SenderEmail, to, subject, message);
            client.Send(mail);
        }
        Console.WriteLine($"已发送邮件给: {to}");
    }
}
```

现在，让我们创建一个.NET Core控制台应用程序来演示如何配置和使用 `EmailService` 服务，并将配置信息对象注册到 `IServiceCollection`：

```
using System;
using Microsoft.Extensions.DependencyInjection;

class Program
{
    static void Main(string[] args)
    {
        // 创建依赖注入容器
        var serviceProvider = new ServiceCollection()
            .AddScoped<EmailService>() // 注册 EmailService 服务
            .Configure<EmailServiceOptions>(options =>
            {
                options.SmtpServer = "smtp.example.com";
                options.SmtpPort = 587;
                options.SenderEmail = "sender@example.com";
                options.SenderPassword = "mypassword";
            })
            .BuildServiceProvider();

        // 获取EmailService服务
        var emailService = serviceProvider.GetRequiredService<EmailService>();

        // 发送邮件
        emailService.SendEmail("recipient@example.com", "Test Email", "This is a test email message.");

        Console.ReadLine();
    }
}
```

在这个示例中，我们首先创建了依赖注入容器，并使用 `.AddScoped<EmailService>()` 注册了 `EmailService` 服务。接下来，使用 `.Configure<EmailServiceOptions>` 配置了 `EmailServiceOptions` 的各个属性。

在 `EmailService` 中，构造函数接受 `EmailServiceOptions` 作为参数，这允许您在服务内部访问配置信息。

当您运行这个控制台应用程序时，它将根据配置的选项发送邮件，并输出发送结果。这个示例演示了如何使用`Action`和`Options`参数配置邮件发送服务，并将配置信息对象注册到`IServiceCollection`，以便在服务内部获取配置信息的值。这种模式提供了更高级别的可配置性和可扩展性，使配置信息与服务的实现分离。