

# 第09期-使用MediatR实现中介者发布订阅

2019年11月25日 11:19

## C#.NET设计模式系列视频教程

来自 <<https://www.xcode.me/Video/List/1000060645>>

### 第17期-Mediator中介者模式

在软件构建过程中，经常会出现多个对象互相关联交互的情况，对象之间常常会维持一种复杂的引用关系，如果遇到一些需求的更改，这种直接的引用关系将面临不断的变化。

在这种情况下，我们可使用一个中介对象来管理对象间的关联关系，避免相互交互的对象之间的紧耦合引用关系，从而更好地抵御变化。用一个中介对象来封装一系列的对象交互。

中介者使各对象不需要显式的相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。学习 .NET 架构中的中介者模式。

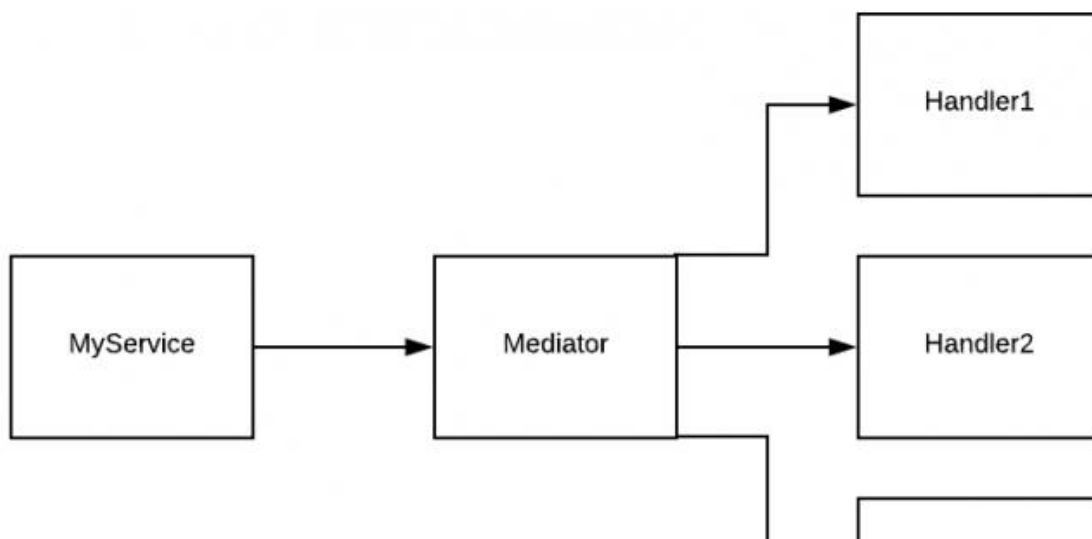
### MediatR 开源库

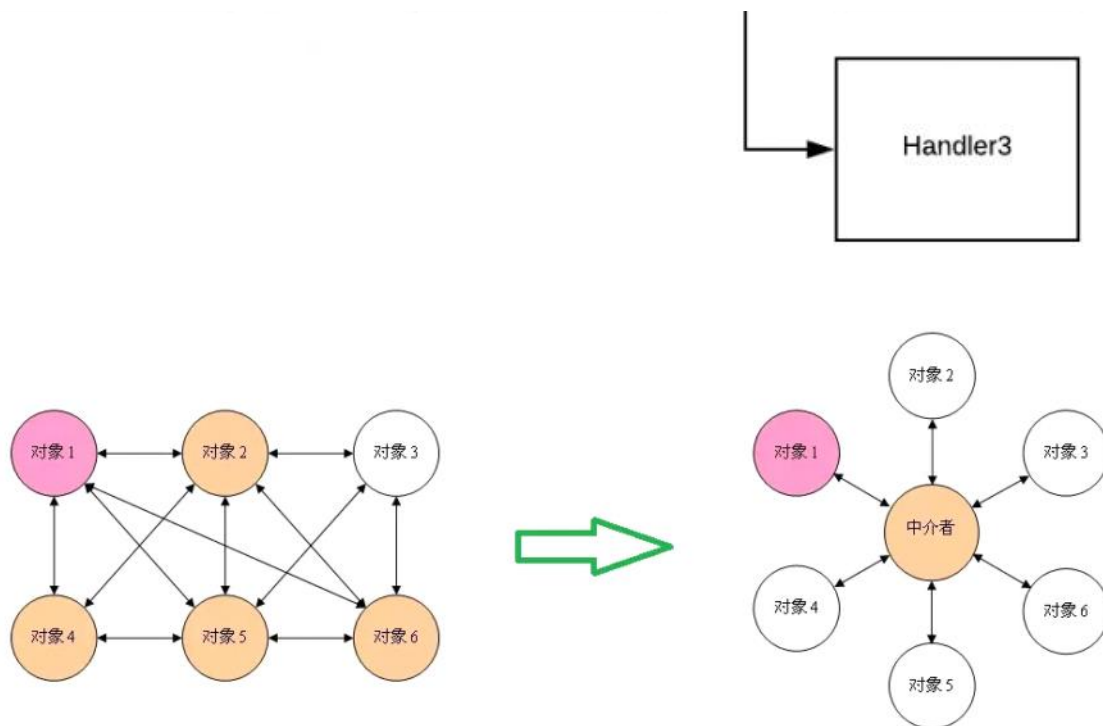
开源地址：<https://github.com/jbogard/MediatR>

依赖注入：<https://github.com/jbogard/MediatR.Extensions.Microsoft.DependencyInjection>

MediatR 和 Mediator 之间少一个字母，作者有意为之？还是手误写错？

一个中介者模式的.NET实现，其目的是消息发送和消息处理的解耦。它支持以单播和多播形式使用同步或异步的模式来发布消息，创建和侦听事件。





优点：中介者模式的优点就是减少类间的依赖，把原有的一对多的依赖变成了一对一的依赖，同时类只依赖中介者，减少了依赖，当然同时也降低了类间的耦合。

缺点：中介者模式的缺点就是中介者会膨胀得很大，而且逻辑复杂，原本N个对象直接的相互依赖关系转换为中介者和同事类的依赖关系，同事类越多，中介者的逻辑就越复杂。

## 依赖注入相关服务

```
services.AddMediatR(System.Reflection.Assembly.GetExecutingAssembly());
```

## MediatR会分派两种消息

请求/响应消息，分派给单个处理程序，单播  
通知消息，分派给多个处理程序，多播

## 有返回值的单播消息

### 定义消息

```
public class Ping : IRequest<string>
{
}
```

### 消息处理器

```
public class PingHandler : IRequestHandler<Ping, string>
```

```
{
    public Task<string> Handle(Ping request, CancellationToken cancellationToken)
    {
        Console.WriteLine("doing...");
        return Task.FromResult("ok");
    }
}
```

## 调用中介者发送消息

```
endpoints.MapGet("/test", async context =>
{
    var mediator = context.RequestServices.GetRequiredService<IMediator>();
    Ping ping = new Ping();
    string result = await mediator.Send(ping);
    await context.Response.WriteAsync(result);
});
```

## 无返回值的单播消息

### 定义消息

```
public class OneWay : IRequest { }
```

### 编写消息处理器

```
public class OneWayHandlerWithBaseClass : AsyncRequestHandler<OneWay>
{
    protected override Task Handle(OneWay request, CancellationToken
cancellationToken)
    {
        // Twiddle thumbs
    }
}
```

## 如果消息完全是同步的，不需要异步，可直接从 RequestHandler 继承

```
public class SyncHandler : RequestHandler<Ping, string>
{
    protected override string Handle(Ping request)
    {
        return "Pong";
    }
}
```

IRequest<T> 有返回值单播消息

IRequest 无返回值单播消息

IRequestHandler <T>返回Task <Unit>。

AsyncRequestHandler <T>返回Task。

RequestHandler <T>不返回任何内容。

## 通知多播消息，无返回值，有返回值无意义

### 定义消息

```
public class Ping : INotification { }
```

### 编写一个或者多个通知接受处理器

```
public class Pong1 : INotificationHandler<Ping>
{
    public Task Handle(Ping notification, CancellationTokentoken cancellationTokentoken)
    {
        Debug.WriteLine("Pong 1");
        return Task.CompletedTask;
    }
}
```

```
public class Pong2 : INotificationHandler<Ping>
{
    public Task Handle(Ping notification, CancellationTokentoken cancellationTokentoken)
    {
        Debug.WriteLine("Pong 2");
        return Task.CompletedTask;
    }
}
```

### 多播消息发送通知

```
await mediator.Publish(new Ping());
```

### 多路广播发布策略问题

Publish 方法的默认实现：同步循环每个处理程序，一个失败不影响后边的处理程序，这样可以确保每个处理程序都依次运行，而且按照顺序运行。

根据发布通知的不同需求，您可能需要不同的策略来处理通知，也许您想并行发布所有通知，或者使用您自己的异常处理逻辑包装每个通知处理程序。

在 MediatR.Examples.PublishStrategies 中可以找到一些示例实现， 其实就是使用 PublishStrategy 枚举设置不同的策略，参考以下链接。

<https://github.com/jbogard/MediatR/tree/master/samples/MediatR.Examples.PublishStrategies>

## 类型协变（单播处理器和多播处理器）

例如：您可以实现 `INotificationHandler <INotification>` 来处理所有通知。

[MediatR 知多少](#)

[MediatR Document WIKI](#)

[ASP.NET Core 使用 MediatR 进程内发布与订阅](#)