عنوان: **تزریق وابستگی‌ها در حالتی‌که از یک اینترفیس چندین کلاس مشتق شده‌اند**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۱۷ ۲۳:۱۷
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, Dependency Injection, IoC

حین کار با ASP.NET Identity به اینترفیسی به نام IIdentityMessageService شبیه به اینترفیس ذیل می‌رسیم:

```
namespace SameInterfaceDifferentClasses.Services.Contracts
{
  public interface IMessageService
  {
   void Send(string message);
  }
}
```

فرض کنید از آن دو پیاده سازی در برنامه برای ارسال پیام‌ها توسط ایمیل و همچنین توسط SMS، وجود دارد:

```
public class EmailService : IMessageService
{
  public void Send(string message)
  {
   // ...
  }
}

public class SmsService : IMessageService
{
  public void Send(string message)
  {
   //todo: ...
  }
}
```

اکنون کلاس مدیریت کاربران برنامه، در سازنده‌ی خود نیاز به دو وهله، از این سرویس‌های متفاوت، اما در اصل مشتق شده‌ی از یک اینترفیس دارد:

```
public interface IUsersManagerService
{
  void ValidateUserByEmail(int id);
}

public class UsersManagerService : IUsersManagerService
{
  private readonly IMessageService _emailService;
  private readonly IMessageService _smsService;

  public UsersManagerService(IMessageService emailService, IMessageService smsService)
  {
   _emailService = emailService;
   _smsService = smsService;
  }

  public void ValidateUserByEmail(int id)
  {
   _emailService.Send("Validated.");
  }
}
```

در این حالت صرف تنظیمات ابتدایی انتساب یک اینترفیس، به یک کلاس مشخص کافی نیست:

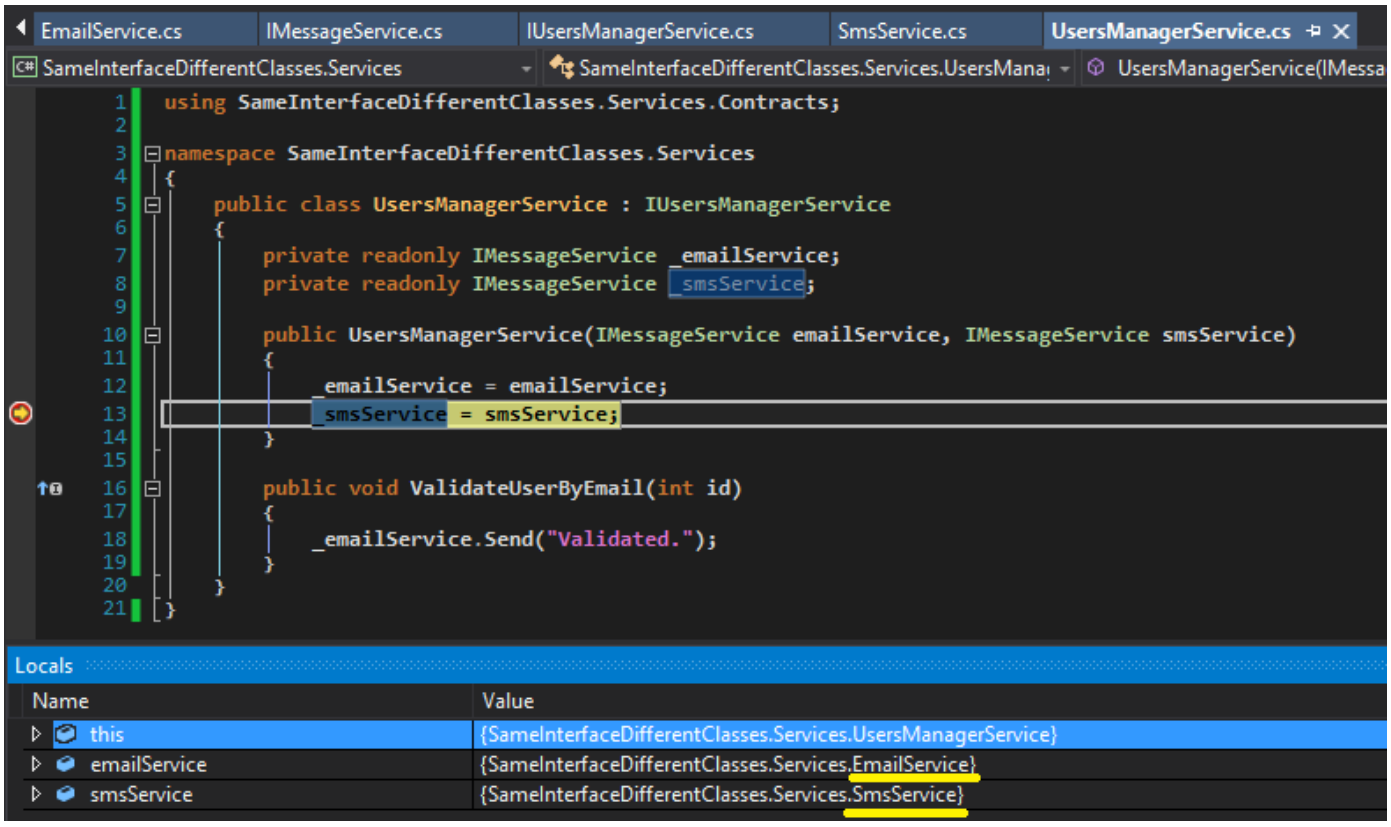
```
ioc.For<IMessageService>().Use<SmsService>();
ioc.For<IMessageService>().Use<EmailService>();
```

از این جهت که در سازنده‌ی کلاس UsersManagerService دقیقا مشخص نیست، پارامتر اول باید سرویس SMS باشد یا ایمیل؟
برای حل این مشکل می‌توان به نحو ذیل عمل کرد:

```
public static class SmObjectFactory
{
  private static readonly Lazy<Container> _containerBuilder =
   new Lazy<Container>(defaultContainer, LazyThreadSafetyMode.ExecutionAndPublication);

  public static IContainer Container
  {
   get { return _containerBuilder.Value; }
  }

  private static Container defaultContainer()
  {
   return new Container(ioc =>
   {
    // map same interface to different concrete classes
    ioc.For<IMessageService>().Use<SmsService>();
    ioc.For<IMessageService>().Use<EmailService>();

    ioc.For<IUsersManagerService>().Use<UsersManagerService>()
     .Ctor<IMessageService>("smsService").Is<SmsService>()
     .Ctor<IMessageService>("emailService").Is<EmailService>();
   });
  }
}
```

در اینجا توسط متد Ctor که مخفف Constructor یا سازنده‌ی کلاس است، مشخص می‌کنیم که اگر به پارامتر smsService رسیدی، از کلاس SmsService استفاده کن و در مورد کلاس سرویس ایمیل نیز به همین ترتیب. اینبار اگر برنامه را اجرا کنیم:

```
var usersManagerService = SmObjectFactory.Container.GetInstance<IUsersManagerService>();
usersManagerService.ValidateUserByEmail(id: 1);
```



همانطور که در تصویر مشخص است، هر کدام از پارامترها، توسط کلاس‌های متفاوتی مقدار دهی شده‌اند؛ هرچند از یک اینترفیس

مشخص استفاده می‌کنند.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید:

Dependency-Injection-Samples/DI09