dotnet 应用程序级别的事件发布和订阅

作者：loogn
原文链接：https://ld246.com/article/1562570151581
来源网站：链滴

我一直都认为，字典这种数据结构是个很好东西，简单而且强大，利用好了真是事半功倍。说到事件时候，我们一定也要想到委托，在net的事件里，事件就是多播委托，这篇文章的角色主要就是字典委托了。

在一个小项目中，对于数据的精准性可能要求不太严格，但是也会存在这样的逻辑，比如下单之后，录日志，增加用户订单量，等等其他一些不必再事务中处理的操作（事务中处理的步骤越少越好），些逻辑操作就算失败一两个也没什么关系，这个时候就没必要使用外部队列来确保执行了，这种情况使用这篇文章说的应用程序内的事件发布和订阅就比较合适。

我们先定义事件管理器：

```
public class AppEventService
    {
        private readonly ConcurrentDictionary<string, Func<object[], object>> _eventHandlerDi
t =
            new ConcurrentDictionary<string, Func<object[], object>>();
    }
```

其中的字典便是存储委托的容器，再提供一个添加订阅和发布的方法如下：

```
        /// <summary>
        /// 添加事件处理程序
        /// </summary>
        /// <param name="eventKey"></param>
        /// <param name="func"></param>
        public void AddHandler(string eventKey, Func<object[], object> func)
        {
            _eventHandlerDict.AddOrUpdate(eventKey, func, (key, oldFun) => { return oldFun +=
unc; });
        }

        /// <summary>
        /// 触发事件
        /// </summary>
        /// <param name="eventKey"></param>
        /// <param name="ps"></param>
        /// <returns></returns>
        public Task Fire(string eventKey, params object[] ps)
        {
            return _eventHandlerDict.TryGetValue(eventKey, out var func)
                ? Task.Run(() => func(ps))
                : Task.CompletedTask;
        }
```

基本功能是实现了，但是由于事件处理的委托需要一个一个添加，也是挺麻烦的，所以我们定义一个性类，来标识应用程序内的事件处理程序：

```
    /// <summary>
    /// 标记方法是appEvent的事件处理程序
    /// </summary>
    [AttributeUsage(AttributeTargets.Method, AllowMultiple = false, Inherited = false)]
    public class AppEventHandlerAttribute : Attribute
    {
        public AppEventHandlerAttribute(string eventKey)
        {
```

```
            EventKey = eventKey;
        }
        /// <summary>
        /// key
        /// </summary>
        public string EventKey { get; set; }
    }
```

有了标识的特性类，我们就可以在AppEventService添加一个批量扫描的方法：

```
/// <summary>
        /// 扫码程序集，注册事件处理程序
        /// </summary>
        /// <param name="assembly"></param>
        public void ScanEventHandler(Assembly assembly)
        {
            foreach (var type in assembly.GetTypes())
            {
                var methodInfos = type.GetMethods(BindingFlags.Static | BindingFlags.Instance | Bi
dingFlags.Public |
                                                  BindingFlags.NonPublic ^ BindingFlags.GetProperty ^
                                                  BindingFlags.SetProperty);

                foreach (var methodInfo in methodInfos)
                {
                    var mehAttr = methodInfo.GetCustomAttribute<AppEventHandlerAttribute>();
                    if (mehAttr == null) continue;
                    var fun = DynamicMethodHelper.GetExecuteDelegate(methodInfo);

                    AddHandler(mehAttr.EventKey,
                        (args) => fun(methodInfo.IsStatic ? null : Activator.CreateInstance(type), args));
                }
            }
        }
```

在asp.netcore中，框架自带了DI，我们再添加的代码来实现从容器中加载：

```
        private IServiceProvider _serviceProvider;
        /// <summary>
        /// 实例化AppEventService
        /// </summary>
        public AppEventService()
        {
        }

        public void SetServiceProvider(IServiceProvider serviceProvider)
        {
            _serviceProvider = serviceProvider;
        }

        /// <summary>
        /// 扫描容器中的服务，注册时间处理程序
        /// </summary>
        /// <param name="services"></param>
```

```
    public void ScanEventHandler(IServiceCollection services)
    {
        foreach (var service in services)
        {
            var methodInfos = service.ServiceType.GetMethods(
                BindingFlags.Static | BindingFlags.Instance | BindingFlags.Public | BindingFlags.N
nPublic ^
                BindingFlags.GetProperty ^ BindingFlags.SetProperty);
            foreach (var methodInfo in methodInfos)
            {
                var mehAttr = methodInfo.GetCustomAttribute<AppEventHandlerAttribute>();
                if (mehAttr == null) continue;
                var fun = DynamicMethodHelper.GetExecuteDelegate(methodInfo);

                AddHandler(mehAttr.EventKey,
                    (args) => fun(methodInfo.IsStatic ? null : _serviceProvider.GetService(service.Se
viceType),
                        args));
            }
        }
    }
```

然后安装常规约定，我们提供一个扩展方法来添加AppEventService：

```
public static class AppEventExtensions
{
    /// <summary>
    /// 注册应用程序域中所有有AppService特性的类
    /// </summary>
    /// <param name="services"></param>
    public static void AddAppEvents(this IServiceCollection services)
    {
        AppEventService appEventService = new AppEventService();
        appEventService.ScanEventHandler(services);
        services.AddSingleton(appEventService);
    }
}
```