**安装**

需要安装包

*Volo.Abp.EventBus.RabbitMQ*

然后在Module类上添加 [DependsOn(typeof(AbpEventBusRabbitMqModule))] 特性

**基本配置**

我们可以在代码中配置，也可以在appsettings.json中配置

* 在模块的ConfigureServices方法中配置

配置连接

Configure<AbpRabbitMqOptions>(options =>

{

    options.Connections.Default.UserName = "user";

    options.Connections.Default.Password = "pass";

    options.Connections.Default.HostName = "123.123.123.123";

    options.Connections.Default.Port = 5672;

});

配置交换机名（ExchangeName）和队列名（ClientName）

Configure<AbpRabbitMqEventBusOptions>(options =>

{

    options.ClientName = "TestApp1";

    options.ExchangeName = "TestMessages";

});

* 在appsettings.json中配置
* {
* "RabbitMQ": {
* "Connections": {
* "Default": {
* "HostName": "123.123.123.123",
* "Port": "5672"
* }
* },
* "EventBus": {
* "ClientName": "MyClientName",
* "ExchangeName": "MyExchangeName"
* }
* }
* }

**集群配置**

你可以在HostName中指定多个集群主机

{

    "RabbitMQ": {

        "Connections": {

            "Default": {

                "HostName": "123.123.123.123;234.234.234.234"

            }

        }

    }

}

**发布事件**

配置好RabbitMQ之后，会可以使用其进行事件发布了，如下

1. 定义事件消息

[EventName("MyApp.Product.StockChange")]

public class StockCountChangedEto

{

    public Guid ProductId { get; set; }

    public int NewCount { get; set; }

}

1. 发布事件

我们通过IDistributedEventBus进行事件发布

using System;

using System.Threading.Tasks;

using Volo.Abp.DependencyInjection;

using Volo.Abp.EventBus.Distributed;

namespace AbpDemo

{

    public class MyService : ITransientDependency

    {

        private readonly IDistributedEventBus \_distributedEventBus;

        public MyService(IDistributedEventBus distributedEventBus)

        {

            \_distributedEventBus = distributedEventBus;

        }

        public virtual async Task ChangeStockCountAsync(Guid productId, int newCount)

        {

            await \_distributedEventBus.PublishAsync(

                new StockCountChangedEto

                {

                    ProductId = productId,

                    NewCount = newCount

                }

            );

        }

    }

}

**在聚合中发布领域事件**

using System;

using Volo.Abp.Domain.Entities;

namespace AbpDemo

{

    public class Product : AggregateRoot<Guid>

    {

        public string Name { get; set; }

        public int StockCount { get; private set; }

        private Product() { }

        public Product(Guid id, string name)

            : base(id)

        {

            Name = name;

        }

        public void ChangeStockCount(int newCount)

        {

            StockCount = newCount;

            //ADD an EVENT TO BE PUBLISHED

            AddDistributedEvent(

                new StockCountChangedEto

                {

                    ProductId = Id,

                    NewCount = newCount

                }

            );

        }

    }

}

**定义事件**

Abp通过继承IDistributedEventHandler来进行事件订阅，如下我们订阅了StockCountChangedEto事件类型

using System.Threading.Tasks;

using Volo.Abp.DependencyInjection;

using Volo.Abp.EventBus.Distributed;

namespace AbpDemo

{

    public class MyHandler

        : IDistributedEventHandler<StockCountChangedEto>,

          ITransientDependency

    {

        public async Task HandleEventAsync(StockCountChangedEto eventData)

        {

            var productId = eventData.ProductId;

        }

    }

}

如果事件处理程序成功执行(没有抛出任何异常)，它将向RabbitMQ代理发送确认(ACK).

**Abp提供的事件**

Abp提供了如下的3中事件，其对应实体的创建，更新和删除

* EntityCreatedEto<T> 是实体创建后发布.
* EntityUpdatedEto<T> 是实体更新后发布.
* EntityDeletedEto<T> 是实体删除后发布.

其中T默认是EntityEto，它只有2个属性

* EntityType (string): 实体类的全名（包括命令空间）
* KeysAsString (string): 已更改实体的主键

如下介绍如何使用它

1. 配置

我们在模块的ConfigureServices方法中配置那些实体会触发这些事件

Configure<AbpDistributedEntityEventOptions>(options =>

{

    // 所有实体都触发

    options.AutoEventSelectors.AddAll();

    // IdentityUser 实体会触发

    options.AutoEventSelectors.Add<IdentityUser>();

    // 命名空间 Volo.Abp.Identity 下的实体会触发

    options.AutoEventSelectors.AddNamespace("Volo.Abp.Identity");

    // Custom predicate expression that should return true to select a type

    options.AutoEventSelectors.Add(

        type => type.Namespace.StartsWith("MyProject.")

    );

});

1. 指定Eto

上面介绍发布的T默认是EntityEto，但我们可以配置指定实体对应的Eto

如下我们指定Product的Eto为ProductEto

Configure<AbpDistributedEntityEventOptions>(options =>

{

    options.AutoEventSelectors.Add<Product>();

    options.EtoMappings.Add<Product, ProductEto>();

});

如下是ProductEto的定义

[AutoMap(typeof(Product))]

public class ProductEto : EntityEto

{

    public Guid Id { get; set; }

    public string Name { get; set; }

}

在发布事件时，AutoMapper会自动将Product映射成ProductEto，然后发布事件

**Abp是如何发送消息的**1、交换器和队列和消费者生成

        "EventBus": {

            "ClientName": "MyClientName",

            "ExchangeName": "MyExchangeName"

        }

我们指定配置，在程序运行时会生成交换器MyExchangeName和队列MyClientName（每一个程序都应该有自己的队列名，不要共用）和一个消费者，这个消费者绑定在MyClientName队列

1. 路由绑定到队列

    public class MyHandler

        : IDistributedEventHandler<StockCountChangedEto>,

          ITransientDependency

--------------------

[EventName("MyApp.Product.StockChange")]

public class StockCountChangedEto

如果我们定义了MyHandler，那么程序运行时会将MyApp.Product.StockChange路由key绑定到当前队列，即MyClientName队列

1. 发送消息

当我们发送StockCountChangedEto消息时，其会去EventName作为路由key，即MyApp.Product.StockChange作为路由key，消息到达交换器后会根据路由key分发到队列，即MyClientName队列

4、接受消息

队列收到消息后把消息发送给消费者，即我们的程序，消费者会运行MyHandler处理消息，如果处理过程没有异常，则回复ack