# Orleans Steam範例專案實作 - 隱式訂閱與Client端訂閱

## 隱式訂閱

隱式訂閱的寫法相較於顯式訂閱的寫法，就變得很簡單，只需在Grain Identity是GUID的Grain類別宣告上加掛 [ImplicitStreamSubscriptionAttribute](https://learn.microsoft.com/dotnet/api/orleans.implicitstreamsubscriptionattribute) 屬性 ，不需要自行呼叫 SubscribeAsync()，並且在萬一Silo故障Grain在別的機器上重啟時，也不需要在 OnActivateAsync() 的Grain生命週期反應函式裡呼叫 ResumeAsync() 恢復訂閱的繁瑣事項。

[ImplicitStreamSubscription("demo-streaming-namespace")]  
public class ConsumerGrain : Grain , ...  
{  
 // Grain implementation...  
}

不過上面這種寫法有一個缺點，就是只能訂閱一個namespace是 demo-streaming-namespace 的事件流，如果要動態訂閱或訂閱多個事件流，屬性可用另一種接受實作 [IStreamNamespacePredicate](https://learn.microsoft.com/en-us/dotnet/api/orleans.streams.istreamnamespacepredicate) 介面的任意自訂類別型態之參數，該介面有一個 bool IsMatch (string streamNamespace)需要開發者自行實作的方法，就可以在此方法內動態判斷。

而在實際使用時，通常Grain實作類別還會多實作一個 [IStreamSubscriptionObserver](https://learn.microsoft.com/en-us/dotnet/api/orleans.streams.core.istreamsubscriptionobserver) 介面，這樣可以在事件流訂閱時，Orleans Runtime自動會呼叫該介面的 OnSubscribed () 方法，如此可以在此方法的實作內，取得訂閱事件的handle以便呼叫 ResumeAsync() 來恢復此Grain的事件流訂閱，並且指派事件收到時真正執行消化事件資料進行處理的實作 [IAsyncObserver<T>](https://learn.microsoft.com/en-us/dotnet/api/orleans.streams.iasyncobserver-1) 介面的物件。

[ImplicitStreamSubscription("demo-streaming-namespace")]  
public class ConsumerGrain : Grain , IStreamSubscriptionObserver  
{  
 private readonly LoggerObserver \_observer;  
  
 // other Grain methods...  
  
 public async Task OnSubscribed(IStreamSubscriptionHandleFactory handleFactory)  
 {  
 var handle = handleFactory.Create<int>();  
 await handle.ResumeAsync(\_observer);  
 }  
}  
  
internal class LoggerObserver : IAsyncObserver<string>  
{  
 // class implementation...  
}

### 隱式訂閱的實作

1. 在[昨天進度的git專案](https://github.com/windperson/OrleansRpcDemo/tree/day20) *src/Shard* 目錄內的 **RpcDemo.Interfaces.EventStreams** 專案內，新增一個 **IConsumerGrain.cs** 介面檔案，內容如下：

* using Orleans;  
    
  namespace RpcDemo.Interfaces.EventStreams;  
    
  public interface IConsumerGrain : IGrainWithGuidKey, IStreamSubscriptionObserver  
  {  
  }
* 此介面由於之後要套用的Grain實作專案沒有其他的RPC方法，所以內部宣告是空的。

1. 更新 **RpcDemo.Interfaces.EventStreams** 專案內的 **StreamDto.cs** 檔案內容為：

* namespace RpcDemo.Interfaces.EventStreams;  
    
  [Serializable]  
  public record struct StreamDto(int Serial, string Message, DateTimeOffset Timestamp);  
    
  public static class StreamConstant  
  {  
   public const string DefaultStreamProviderName = "MyDefaultStreamProvider";  
   public const string ImplicitSubscribeStreamNamespace = "event-streaming-02";  
  }

1. 在 **RpcDemo.Grains.EventStreams** 專案內，新增一個 **ConsumerGrain.cs** 檔案，內容如下：

* using Microsoft.Extensions.Logging;  
  using Orleans;  
  using Orleans.Streams;  
  using Orleans.Streams.Core;  
  using RpcDemo.Interfaces.EventStreams;  
    
  namespace RpcDemo.Grains.EventStreams;  
    
  [ImplicitStreamSubscription(StreamConstant.ImplicitSubscribeStreamNamespace)]  
  public class ConsumerGrain : Grain, IConsumerGrain  
  {  
   private readonly LoggerObserver \_observer;  
    
   public ConsumerGrain(ILogger<ConsumerGrain> logger)  
   {  
   \_observer = new LoggerObserver(this.GetPrimaryKeyString(), logger);  
   }  
    
   public async Task OnSubscribed(IStreamSubscriptionHandleFactory handleFactory)  
   {  
   var handle = handleFactory.Create<StreamDto>();  
   await handle.ResumeAsync(\_observer);  
   }  
  }  
    
  internal class LoggerObserver : IAsyncObserver<StreamDto>  
  {  
   private readonly ILogger \_logger;  
   private readonly string \_grainPrimaryKey;  
    
   public LoggerObserver(string grainPrimaryKey, ILogger logger)  
   {  
   \_grainPrimaryKey = grainPrimaryKey;  
   \_logger = logger;  
   }  
    
   public Task OnNextAsync(StreamDto item, StreamSequenceToken? token = null)  
   {  
   \_logger.LogInformation("Grain {0} receive: {1}", \_grainPrimaryKey, item);  
   return Task.CompletedTask;  
   }  
    
   public Task OnCompletedAsync()  
   {  
   \_logger.LogInformation("call OnCompletedAsync()");  
   return Task.CompletedTask;  
   }  
    
   public Task OnErrorAsync(Exception ex)  
   {  
   \_logger.LogError(ex, "call OnErrorAsync()");  
   return Task.CompletedTask;  
   }  
  }
* 此Grain實作類別，宣告了一個 LoggerObserver 類別的成員，在 OnSubscribed() 方法內，呼叫 ResumeAsync() 恢復既有訂閱（抑或起始新訂閱）時，指派 LoggerObserver 類別的物件作為事件收到時的處理者。

ConsumerGrain實作完成，來在測試專案裡加上對應的測試：

在 *src/tests* 的 **EventStreamGrains.Tests** 測試專案中，新增一個 **ConsumerGrainTest.cs** 程式碼檔案，內容如下：

using Moq;  
using Orleans;  
using Orleans.Hosting;  
using Orleans.Providers;  
using Orleans.TestingHost;  
using Orleans.Timers;  
using RpcDemo.Grains.EventStreams;  
using RpcDemo.Interfaces.EventStreams;  
  
namespace EventStreamGrains.Tests;  
  
public class ConsumerGrainTest  
{  
 private static Mock<ILogger<ConsumerGrain>>? \_loggerMock;  
  
 #region Test Silo Setup  
 private class TestSiloAndClientConfigurator : ISiloConfigurator, IClientBuilderConfigurator  
 {  
 public static Func<object, Task>? TimerTick { get; private set; }  
  
 public void Configure(ISiloBuilder siloBuilder)  
 {  
 \_loggerMock = new Mock<ILogger<ConsumerGrain>>();  
 var loggerFactorMock = new Mock<ILoggerFactory>();  
 loggerFactorMock.Setup(x => x.CreateLogger(It.IsAny<string>())).Returns(\_loggerMock.Object);  
  
 var mockTimerRegistry = new Mock<ITimerRegistry>();  
 mockTimerRegistry.Setup(x =>  
 x.RegisterTimer(It.IsAny<Grain>(),  
 It.IsAny<Func<object, Task>>(), It.IsAny<object>(), It.IsAny<TimeSpan>(), It.IsAny<TimeSpan>()))  
 .Returns(new Mock<IDisposable>().Object)  
 .Callback(  
 (Grain targetGrain, Func<object, Task>? timerTick, object \_, TimeSpan \_, TimeSpan \_) =>  
 {  
 // Hook producer's every second message producing timer,  
 // so we can invoke it later in Test method.  
 if (targetGrain is ProducerGrain && timerTick != null)  
 {  
 TimerTick = timerTick;  
 }  
 });  
 siloBuilder  
 .AddMemoryGrainStorage("PubSubStore")  
 .AddMemoryStreams<DefaultMemoryMessageBodySerializer>(StreamConstant.DefaultStreamProviderName)  
 .ConfigureServices(services =>  
 {  
 services.AddSingleton(loggerFactorMock.Object);  
 services.AddSingleton(mockTimerRegistry.Object);  
 });  
 }  
  
 public void Configure(IConfiguration configuration, IClientBuilder clientBuilder)  
 {  
 clientBuilder.AddMemoryStreams<DefaultMemoryMessageBodySerializer>(StreamConstant.DefaultStreamProviderName);  
 }  
 }  
 #endregion  
   
 [Fact]  
 public async Task Test\_ConsumerGrain\_Receive()  
 {  
 // Arrange  
 var builder = new TestClusterBuilder();  
 builder.AddSiloBuilderConfigurator<TestSiloAndClientConfigurator>();  
 var testCluster = builder.Build();  
 await testCluster.DeployAsync();  
 var key = Guid.NewGuid();  
 var producer = testCluster.GrainFactory.GetGrain<IProducerGrain>("sender1");  
 var consumer = testCluster.GrainFactory.GetGrain<IConsumerGrain>("receiver1");  
  
 // Act  
 await producer.StartProducing(StreamConstant.ImplicitSubscribeStreamNamespace, key);  
 //Manual Invoke Timer to force produce message to consumer  
 var timerTick = TestSiloAndClientConfigurator.TimerTick;  
 Assert.NotNull(timerTick);  
 await timerTick.Invoke(new object());  
 await timerTick.Invoke(new object());  
 await producer.StopProducing();  
 //Give some time for stream to deliver message  
 await Task.Delay(TimeSpan.FromSeconds(0.3));  
 await testCluster.StopAllSilosAsync();  
  
 // Assert  
 Assert.NotNull(\_loggerMock);  
 \_loggerMock.VerifyLog(logger =>  
 logger.LogInformation("Grain {0} receive: {1}",  
 It.IsAny<string>(), It.IsAny<StreamDto>()), Times.Exactly(2));  
 }  
}

這個和昨天測試顯式訂閱Grain的測試程式碼幾乎一樣，除了Silo配置時可以省略掉先前顯式訂閱時需儲存的訂閱狀態資料之外，在測試程式碼Act的階段也省掉了呼叫接收端Grain訂閱事件流RPC方法的動作。

### 整合至Silo端實際執行

1. 將 *src/Hosting/Server* 目錄下 **RpcDemo.Hosting.Worker** 專案的 Program.cs 檔案內配置SiloBuilder ApplicationParts的配置程式碼新增 ConsumerGrain 的註冊：

* siloBuilder.ConfigureApplicationParts(parts =>  
  {  
   parts.AddApplicationPart(typeof(ManualConsumerGrain).Assembly).WithReferences();  
   parts.AddApplicationPart(typeof(ProducerGrain).Assembly).WithReferences();  
   parts.AddApplicationPart(typeof(ConsumerGrain).Assembly).WithReferences();  
  });

1. 修改 *src/Hosting/Client* 目錄下 **RpcDemo.Client.StreamConsole** 專案的 Program.cs 檔案：
   * 配置ClientBuilder ApplicationParts的配置程式碼新增 IConsumerGrain 的註冊：
   * .ConfigureApplicationParts(parts =>  
     {  
      parts.AddApplicationPart(typeof(IProducerGrain).Assembly).WithReferences();  
      parts.AddApplicationPart(typeof(IManualConsumerGrain).Assembly).WithReferences();  
      parts.AddApplicationPart(typeof(IConsumerGrain).Assembly).WithReferences();  
     })
   * 在最後呼叫 await receiver2.UnSubscribe(); 修改增加另外測試隱式訂閱的訊息發送程式碼：
   * Log.Logger.Information("\r\nPress any key to demo implicit stream subscription\r\n");  
     Console.ReadKey();  
     await producer.StartProducing(StreamConstant.ImplicitSubscribeStreamNamespace, key);  
       
     Log.Logger.Information("Stopped streaming in Producer Grain, press any key to disconnect from Silo and exit");  
     Console.ReadKey();  
     await producer.StopProducing();  
     await client.Close();

將Silo和Client端都執行起來之後，執行起來到最後的步驟，應該可以看到Silo產生的訊息接收log類似如下：

[15:24:18 INF] Grain dbd4a9d2-ebc7-4114-a2a5-abcc61c24ddd receive: StreamDto { Serial = 49, Message = #0049 from ProducerGrain:sender1, Timestamp = 10/6/2022 7:24:18 AM +00:00 }

## Client端訂閱

Orleans RPC Client端訂閱事件流的程式碼寫法，就是在Client端呼叫 GetStreamProvider("A Stream Provider Name").GetStream<T>(a\_GUID, "a-stream-namespace") 來取得事件流，並且呼叫 [SubscribeAsync()](https://learn.microsoft.com/dotnet/api/orleans.streams.asyncobservableextensions.subscribeasync) 來訂閱事件流，指派事件收到時真正執行消化事件資料進行處理的實作 [IAsyncObserver<T>](https://learn.microsoft.com/dotnet/api/orleans.streams.iasyncobserver-1) 介面的物件。

而在運營(ops)方面，Client端要能夠訂閱事件流，在Silo端使用的 Stream Provider，也必須要在Client端配置，以使Client端能夠連接到和Silo端同樣的底層訊息佇列系統，才能使用。

而其實之前第18天講的Grain Observer事件發送機制，也可以讓外界有實作 [IGrainObserver](https://learn.microsoft.com/dotnet/api/orleans.igrainobserver) 的物件訂閱來動作，只是這樣的寫法，在Orleans的RPC Client端呼叫沒有作用，而寫在Silo端的話，要注意事件訂閱物件在取消訂閱後是否有記憶體洩漏的問題，所以在此不示範一般物件訂閱Grain Observer事件發送機制的實作程式。

建議使用事件流訂閱的方式來讓Grain驅動外部程式反應的方法，因為此種方式可以讓外部程式在不需要知道Grain實體參考的情況下，就可以訂閱事件流，並且因為有底層實際訊息佇列系統來分派發送事件流訊息，可靠性較Grain Observer高。

以下示範使用RPC Client端訂閱的寫法：

### Client端訂閱的實作

1. 將 *src/Hosting/Client* 目錄下 **RpcDemo.Client.StreamConsole** 專案，安裝 [Microsoft.Orleans.Streaming.AzureStorage](https://www.nuget.org/packages/Microsoft.Orleans.Streaming.AzureStorage) NuGet 套件。
2. 修改此專案的 Program.cs 檔案，為 clientBuilder 配置程式碼新增Azure queue storage stream provider的設定，修改為：

* var clientBuilder = new ClientBuilder()  
  .UseLocalhostClustering()  
  .Configure<ClusterOptions>(options =>  
  {  
   options.ClusterId = "client1";  
   options.ServiceId = "Stream-Demo";  
  })  
  .AddAzureQueueStreams(StreamConstant.DefaultStreamProviderName,  
   (OptionsBuilder<AzureQueueOptions> optionsBuilder) =>  
   {  
   optionsBuilder.Configure(options => { options.ConfigureQueueServiceClient("UseDevelopmentStorage=true"); });  
   })  
  .ConfigureApplicationParts(parts =>  
  {  
   parts.AddApplicationPart(typeof(IProducerGrain).Assembly).WithReferences();  
   parts.AddApplicationPart(typeof(IManualConsumerGrain).Assembly).WithReferences();  
   parts.AddApplicationPart(typeof(IConsumerGrain).Assembly).WithReferences();  
  })  
  .ConfigureLogging(logging => logging.AddSerilog());
* 最前面的引用命名空間需要增加這些：
* using Microsoft.Extensions.Options;  
  using Orleans.Hosting;  
  using Orleans.Streams;

1. 修改 Program.cs 檔案，從呼叫停止隱式訂閱範例的producer的RPC方法之後，最後段改為：

* Log.Logger.Information("\r\nPress any key to demo client-side stream subscription\r\n");  
  Console.ReadKey();  
  var stream = client.GetStreamProvider(StreamConstant.DefaultStreamProviderName)  
   .GetStream<StreamDto>(key, StreamConstant.ImplicitSubscribeStreamNamespace);  
  await stream.SubscribeAsync((dto, \_) =>  
  {  
   Log.Logger.Information("Received message from stream: {dto}", dto);  
   return Task.CompletedTask;  
  });  
  await producer.StartProducing(StreamConstant.ImplicitSubscribeStreamNamespace, key);  
    
  Log.Logger.Information("\r\nPress any key to stop streaming in Producer Grain\r\n");  
  Console.ReadKey();  
  await producer.StopProducing();  
    
  Log.Logger.Information("Stopped streaming in Producer Grain, press any key to disconnect from Silo and exit");  
  Console.ReadKey();  
  await client.Close();
* 其中那個 await stream.SubscribeAsync() 的訂閱事件流API，除了像在Silo端Grain實作程式碼內定義 Task OnNextAsync(int item, StreamSequenceToken? token = null){ ... } 自訂函式來接收事件訊息並處理之外，也可像上面的範例程式碼一樣，使用Lambda運算式來產生對應 Func<StreamDto, StreamSequenceToken, Task> 型態的事件訊息處理匿名函式。

將Silo和Client端都執行起來到最後的步驟，應該可以看到Client端Console視窗產生的訊息接收log類似如下：

[16:40:28 INF] Received message from stream: StreamDto { Serial = 8, Message = #0008 from ProducerGrain:sender1, Timestamp = 10/6/2022 8:40:28 AM +00:00 }

完整範例的程式碼：  
<https://github.com/windperson/OrleansRpcDemo/tree/day21>

明天來說明Orleans v3.x 版最新提供的功能：Grain RPC的Transaction機制。