# Orleans Steam範例專案實作

## RPC介面與Grain實作

1. 在[前天進度的git專案](https://github.com/windperson/OrleansRpcDemo/tree/day18)中，分別建立新的RPC介面專案和Grain實作專案：

| * 路徑 | * 專案名稱 | 專案類型 | |
| --- | --- | --- |
| * src/Shared | * **RpcDemo.Interfaces.EventStreams** | * .NET 6 類別庫(class library) |
| * src/Grains | * **RpcDemo.Grains.EventStreams** | * .NET 6 類別庫(class library) |

* 將這兩個專案各自加入根目錄的Orleans.sln方案的Shared以及Grains方案資料夾(Solution Folder)中。

1. 將 **RpcDemo.Interfaces.EventStreams** 加入至 **RpcDemo.Grains.EventStreams** 專案的專案對專案參考(project-to-project reference)中。
2. 各專案要安裝的Nuget套件：

| * 專案名稱 | * 需安裝Nuget套件 |
| --- | --- |
| * **RpcDemo.Interfaces.EventStreams** | * + [Microsoft.Orleans.Core.Abstractions](https://www.nuget.org/packages/Microsoft.Orleans.Core.Abstractions)   + [Microsoft.Orleans.CodeGenerator.MSBuild](https://www.nuget.org/packages/Microsoft.Orleans.CodeGenerator.MSBuild) |
| * **RpcDemo.Grains.EventStreams** | * + [Microsoft.Extensions.Logging.Abstractions](https://www.nuget.org/packages/Microsoft.Extensions.Logging.Abstractions)   + [Microsoft.Orleans.Runtime.Abstractions](https://www.nuget.org/packages/Microsoft.Orleans.Runtime.Abstractions)   + [Microsoft.Orleans.Core](https://www.nuget.org/packages/Microsoft.Orleans.Core)   + [Microsoft.Orleans.CodeGenerator.MSBuild](https://www.nuget.org/packages/Microsoft.Orleans.CodeGenerator.MSBuild) |

1. 將 **RpcDemo.Interfaces.EventStreams** 專案中，移除預設專案範本產的 *Class1.cs*，新增C#程式碼檔案：  
   <<**IProducerGrain.cs**>>

* using Orleans;  
    
  namespace RpcDemo.Interfaces.EventStreams;  
    
  public interface IProducerGrain : IGrainWithStringKey  
  {  
   Task StartProducing(string streamNameSpace, Guid key);  
    
   Task StopProducing();  
  }
* 這都是自定義的RPC方法，StartProducing()方法會開始產生訊息寫進事件流，StopProducing()方法會停止產生訊息。  
  <<**IManualConsumerGrain.cs**>>
* using Orleans;  
    
  namespace OrleansStreamingDemo.Grains.Interfaces;  
    
  public interface IManualConsumerGrain : IGrainWithStringKey  
  {  
   Task Subscribe(string streamNameSpace, Guid key);  
    
   Task UnSubscribe();  
  }
* 這裡定義顯式訂閱訊息的RPC方法，Subscribe()方法會訂閱指定的事件流，UnSubscribe()方法會取消訂閱。  
  <<**StreamDto.cs**>>
* namespace RpcDemo.Interfaces.EventStreams;  
    
  [Serializable]  
  public record struct StreamDto(int Serial, string Message, DateTimeOffset Timestamp);  
    
  public static class StreamConstant  
  {  
   public const string DefaultStreamProviderName = "MyDefaultStreamProvider";  
  }
* 此為定義事件流的資料型別，並定義預設的事件流提供者(Stream Provider)名稱。

1. 建立事件流的Producer實作：  
   將 **RpcDemo.Grains.EventStreams** 專案中，移除預設專案範本產的 *Class1.cs*，新增C#程式碼檔案：  
   <<**ProducerGrain.cs**>>

* using Microsoft.Extensions.Logging;  
  using Orleans;  
  using Orleans.Streams;  
  using RpcDemo.Interfaces.EventStreams;  
    
  namespace RpcDemo.Grains.EventStreams;  
    
  public class ProducerGrain : Grain, IProducerGrain  
  {  
   private readonly ILogger<ProducerGrain> \_logger;  
   private IDisposable? \_timer;  
   private IAsyncStream<StreamDto>? \_stream;  
   private int \_counter;  
    
   public ProducerGrain(ILogger<ProducerGrain> logger)  
   {  
   \_logger = logger;  
   }  
    
   public Task StartProducing(string streamNameSpace, Guid key)  
   {  
   if (\_timer is not null)  
   {  
   throw new Exception("This grain is already producing events");  
   }  
    
   //Get a reference to the stream  
   \_stream = GetStreamProvider(StreamConstant.DefaultStreamProviderName).GetStream<StreamDto>(key, streamNameSpace);  
    
   //Create a timer that will send a message every second  
   var period = TimeSpan.FromSeconds(1);  
   \_timer = RegisterTimer(TimerTick, null, period, period);  
    
   \_logger.LogInformation("Started producing events for stream {StreamNameSpace}/{Key} every {period}", streamNameSpace,  
   key, period);  
   return Task.CompletedTask;  
   }  
    
   private async Task TimerTick(object \_)  
   {  
   \_counter++;  
   if (\_stream is not null)  
   {  
   var data = new StreamDto  
   {  
   Serial = \_counter,  
   Message = $"#{\_counter:0000} from {nameof(ProducerGrain)}:{this.GetPrimaryKey()}",  
   Timestamp = DateTime.UtcNow  
   };  
   \_logger.LogInformation("Sending event {Event}", data);  
   await \_stream.OnNextAsync(data);  
   }  
   }  
    
   public async Task StopProducing()  
   {  
   if (\_timer is not null)  
   {  
   \_timer.Dispose();  
   \_timer = null;  
   }  
    
   if (\_stream is not null)  
   {  
   try  
   {  
   await \_stream.OnCompletedAsync();  
   }  
   catch (Exception e)  
   {  
   \_logger.LogWarning("Stream does not support OnCompletedAsync()");  
   }  
    
   \_stream = null;  
   }  
    
   \_logger.LogInformation("Stopped producing events");  
   }  
  }
* 在開始製造事件流訊息的RPC方法 StartProducing() 實作中，首先利用 GetStreamProvider(..).GetStream<StreamDto>(...) 的API取得Stream的參考之後，使用Grain的Timer機制，每秒產生一筆訊息，並寫進事件流中。  
  而在停止製造事件流訊息的RPC方法 StopProducing() 實作中，則是針對Timer的參考呼叫 Dispose() 方法以便停止Timer，然後呼叫 OnCompletedAsync() 方法，來結束事件流的寫入。但[由於歷史因素，新的Stream Provider其實已經不需要呼叫 OnCompletedAsync()方法](https://github.com/dotnet/orleans/issues/7059#issuecomment-823645203)，為了此Grain在執行時保持對於不同種Stream Provider的相容性，我們這邊還是呼叫這個方法，只是用 try..catch 包起來。

1. 建立事件流的顯式訂閱Consumer實作：  
   將 **RpcDemo.Grains.EventStreams** 專案，新增C#程式碼檔案：  
   <<**ManualConsumerGrain.cs**>>

* using Microsoft.Extensions.Logging;  
  using Orleans;  
  using Orleans.Runtime;  
  using Orleans.Streams;  
  using RpcDemo.Interfaces.EventStreams;  
    
  namespace RpcDemo.Grains.EventStreams;  
    
  public record struct StreamInfo(Guid StreamId, string StreamNamespace);  
    
  public class ManualConsumerGrain : Grain, IManualConsumerGrain  
  {  
   private readonly ILogger<ManualConsumerGrain> \_logger;  
   private StreamSubscriptionHandle<StreamDto>? \_handle;  
   private readonly IPersistentState<StreamInfo> \_streamInfo;  
    
   public ManualConsumerGrain([PersistentState("subscribe\_stream", "consumer\_grain")] IPersistentState<StreamInfo> streamInfo,  
   ILogger<ManualConsumerGrain> logger)  
   {  
   \_streamInfo = streamInfo;  
   \_logger = logger;  
   }  
    
   public override async Task OnActivateAsync()  
   {  
   await base.OnActivateAsync();  
   if (\_streamInfo.RecordExists)  
   {  
   var stream = GetStreamProvider(StreamConstant.DefaultStreamProviderName)  
   .GetStream<StreamDto>(\_streamInfo.State.StreamId, \_streamInfo.State.StreamNamespace);  
   var allHandles = await stream.GetAllSubscriptionHandles();  
   if (allHandles is not null)  
   {  
   \_handle = allHandles.FirstOrDefault();  
   if (\_handle is not null)  
   {  
   \_handle = await \_handle.ResumeAsync(\_onNext);  
   }  
   }  
   }  
   }  
    
   public async Task Subscribe(string streamNameSpace, Guid key)  
   {  
   if (\_handle is not null)  
   {  
   throw new Exception("Already subscribed");  
   }  
    
   var stream = GetStreamProvider(StreamConstant.DefaultStreamProviderName).GetStream<StreamDto>(key, streamNameSpace);  
   \_handle = await stream.SubscribeAsync(\_onNext);  
   \_streamInfo.State = new StreamInfo(key, streamNameSpace);  
   await \_streamInfo.WriteStateAsync();  
   }  
    
   private Func<StreamDto, StreamSequenceToken, Task> \_onNext => (dto, \_) =>  
   {  
   \_logger.LogInformation("Grain {0} receive: {1}", this.GetPrimaryKeyString(), dto);  
   return Task.CompletedTask;  
   };  
    
   public async Task UnSubscribe()  
   {  
   if (\_handle is not null)  
   {  
   await \_handle.UnsubscribeAsync();  
   \_handle = null;  
   await \_streamInfo.ClearStateAsync();  
   }  
   }  
  }
* 在這裡由於是顯式訂閱的寫法，因此需要在 OnActivateAsync() 方法中，將之前訂閱stream時的自訂邏輯常式恢復回來，所以當此Grain有訂閱事件流時，將取得stream用的資訊以 StreamInfo 資料結構存放在Grain狀態中，並且在 OnActivateAsync() 方法中，假如有存在這些資訊，就依此取得stream，並且呼叫 GetAllSubscriptionHandles() 方法，取得此Grain對於該stream所有的訂閱handle，最後呼叫 ResumeAsync() 方法，恢復訂閱事件流的動作。

## 測試專案驗證Grain-to-Grain事件流功能

1. 在 *tests* 目錄下建立用xUnit的測試專案 **EventStream.Tests**，安裝或更新下列Nuget套件：
   * [coverlet.collector](https://www.nuget.org/packages/coverlet.collector)
   * [ILogger.Moq](https://www.nuget.org/packages/ILogger.Moq)
   * [Microsoft.NET.Test.Sdk](https://www.nuget.org/packages/Microsoft.NET.Test.Sdk/)
   * [Microsoft.Orleans.TestingHost](https://www.nuget.org/packages/Microsoft.Orleans.TestingHost)
   * [Moq](https://www.nuget.org/packages/Moq)
   * [xunit](https://www.nuget.org/packages/xunit)
   * [xunit.runner.visualstudio](https://www.nuget.org/packages/xunit.runner.visualstudio)
   * [xunit.runner.console](https://www.nuget.org/packages/xunit.runner.console)
2. 將此專案加入根目錄的Orleans.sln方案的 *tests* 方案資料夾(Solution Folder)中。
3. 將 **RpcDemo.Grains.EventStreams** 專案加入至此測試專案的專案對專案參考(project-to-project reference)中。
4. 將專案中預設產生的 *Using.cs* 檔案，修改如下：

* global using Xunit;  
  global using Microsoft.Extensions.Configuration;  
  global using Microsoft.Extensions.DependencyInjection;  
  global using Microsoft.Extensions.Logging;
* 先將一些一定會用到的namespace引用加入到全域範圍中，以方便測試程式撰寫。

1. 將此測試專案中的 *UnitTest1.cs* 刪除，新增一個 *ManualConsumerGrainTest.cs*，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 ManualConsumerGrainTest  
  {  
   private static Mock<ILogger<ManualConsumerGrain>>? \_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<ManualConsumerGrain>>();  
   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("consumer\_grain")  
   .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\_ManualConsumerGrain\_Receive()  
   {  
   // Arrange  
   var builder = new TestClusterBuilder();  
   builder.AddSiloBuilderConfigurator<TestSiloAndClientConfigurator>();  
   var testCluster = builder.Build();  
   await testCluster.DeployAsync();  
   var key = Guid.NewGuid();  
   const string streamNamespace = "TestNamespace";  
   var producer = testCluster.GrainFactory.GetGrain<IProducerGrain>("sender1");  
   var consumer = testCluster.GrainFactory.GetGrain<IManualConsumerGrain>("receiver1");  
    
   // Act  
   await producer.StartProducing(streamNamespace, key);  
   await consumer.Subscribe(streamNamespace, 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));  
   }  
  }
* 這個測試類別的程式碼內容中使用了2個技巧：
  1. 使用 [**ILogger.Moq**](https://github.com/adrianiftode/moq.ilogger) 套件提供的功能，來檢驗 ManualConsumerGrain 是否有正確的寫log，也就是有接收到事件流訊息（這需要包裝成LoggerFactory的形式，然後在Silo依賴注入，如此在Grain的建構子才能產生用此套件的Logger實體）。
  2. 由於原本 ProducerGrain 的stream訊息產生機制是用一秒執行一次的Timer來發送，所以在TestSilo的初始配置程式碼中，塞入一個Mock的 [ITimerRegistry](https://learn.microsoft.com/en-us/dotnet/api/orleans.timers.itimerregistry) 依賴注入服務。  
     （因為當Grain要呼叫RegisterTimer的API新增Timer時，實際上底層都會透過Orleans的底層TimerRegistry來跟Orleans Runtime註冊，讓Timer時間到時，由Orleans Runtime觸發執行）  
     這樣當ProducerGrain的Timer建立時，就能將Timer執行的方法內容用Moq框架提供的callback API，來取得可呼叫的參考實體，以便等下在測試方法的實作程式碼中，手動呼叫Timer的Tick方法來強制產生事件流訊息，如此就能在測試方法中手動控制訊息產生，以此來檢驗Consumer端有正確接收到訊息。

注意：當使用Orleans提供的In-Memory Stream Provider時，**需要額外多配置一個In-Memory的Grain Storage 來儲存訊息，而且一定要將Provider名稱定為 PubSubStore**，否則會出現錯誤訊息，所以需要的配置程式碼如下：

siloBuilder  
 .AddMemoryGrainStorage("PubSubStore")  
 .AddMemoryStreams<DefaultMemoryMessageBodySerializer>("Provider Name configured in Grain");

然後就可以執行測試，如果測試通過，就表示有接收到事件流訊息。

## 建立使用Azure Queue Storage作為Stream Provider的Silo Worker Service程式

### 建立執行Silo的Worker Service專案

1. 在git專案的根目錄使用命令列建立一個新的Worker Service專案 **RpcDemo.Hosting.Worker** 到 *src/Hosting/Server* 目錄內：

* dotnet new worker --no-restore --name RpcDemo.Hosting.Worker --output src/Hosting/Server/RpcDemo.Hosting.Worker
* 將此專案加入根目錄的Orleans.sln方案的 *Hosting/Server* 方案資料夾(Solution Folder)中：
* dotnet sln add ./src/Hosting/Server/RpcDemo.Hosting.Worker/RpcDemo.Hosting.Worker.csproj --solution-folder Hosting/Server

1. 將 **RpcDemo.Grains.EventStreams** 專案加入至此專案的專案對專案參考(project-to-project reference)中。
2. 安裝/更新下列Nuget套件至此專案中：
   * [Microsoft.Extensions.Hosting](https://www.nuget.org/packages/Microsoft.Extensions.Hosting)
   * [Microsoft.Orleans.Persistence.AzureStorage](https://www.nuget.org/packages/Microsoft.Orleans.Persistence.AzureStorage)
   * [Microsoft.Orleans.Streaming.AzureStorage](https://www.nuget.org/packages/Microsoft.Orleans.Streaming.AzureStorage)
   * [Microsoft.Orleans.Server](https://www.nuget.org/packages/Microsoft.Orleans.Server)
   * [Serilog.Extensions.Hosting](https://www.nuget.org/packages/Serilog.Extensions.Hosting/)
   * [Serilog.Sinks.Console](https://www.nuget.org/packages/Serilog.Sinks.Console)
   * [Serilog.Sinks.Debug](https://www.nuget.org/packages/Serilog.Sinks.Debug)
3. 將專案中的 *Worker.cs* 檔刪除，修改 **Program.cs** 檔為下列內容：

* using Microsoft.Extensions.Options;  
  using Orleans;  
  using Orleans.Configuration;  
  using Orleans.Hosting;  
  using RpcDemo.Grains.EventStreams;  
  using RpcDemo.Interfaces.EventStreams;  
  using Serilog;  
  using Serilog.Events;  
    
  Log.Logger = new LoggerConfiguration()  
   .MinimumLevel.Override("Microsoft", LogEventLevel.Information)  
   .Enrich.FromLogContext()  
   .WriteTo.Console()  
   .WriteTo.Debug()  
   .CreateLogger();  
    
  IHost host = Host.CreateDefaultBuilder(args)  
   .UseSerilog()  
   .UseOrleans((ISiloBuilder siloBuilder) =>  
   {  
   siloBuilder.UseLocalhostClustering()  
   .Configure<ClusterOptions>(options =>  
   {  
   options.ClusterId = "silo1";  
   options.ServiceId = "Stream-Demo";  
   })  
   .AddAzureTableGrainStorage("PubSubStore", options =>  
   {  
   options.ConfigureTableServiceClient("UseDevelopmentStorage=true");  
   })  
   .AddAzureQueueStreams(StreamConstant.DefaultStreamProviderName,  
   (OptionsBuilder<AzureQueueOptions> optionsBuilder) =>  
   {  
   optionsBuilder.Configure(options => { options.ConfigureQueueServiceClient("UseDevelopmentStorage=true"); });  
   });  
   siloBuilder.ConfigureApplicationParts(parts =>  
   {  
   parts.AddApplicationPart(typeof(ManualConsumerGrain).Assembly).WithReferences();  
   parts.AddApplicationPart(typeof(ProducerGrain).Assembly).WithReferences();  
   });  
   })  
   .ConfigureServices(services =>  
   {  
   })  
   .Build();  
    
  await host.RunAsync();
* 這邊要注意的是，由於Azure Queue Storage的名稱合法字元規則是英數字大小寫以及『-』符號，而Orleans的Stream Provider會將Silo ClusterOption中設定的ServiceId納入到Azure Queue Storage的名稱中，所以在此範例中，ServiceId設定不能使用到Azure Queue Storage不容許的字元。  
  還有AzureQueueStorage跟In-Memory Storage一樣，也需要有個叫 **PubSubStore** 的Grain Storage Provider也配置才能正常作用，所以這裡也配置了一個Azure Table Storage的Provider供Stream provider使用。

### 建立呼叫的Client端Console專案

1. 在git專案的根目錄建立一個新的Console專案 **RpcDemo.Client.StreamConsole** 在 *src/Hosting/Client* 目錄下：

* dotnet new console --framework net6.0 --no-restore --name RpcDemo.Client.StreamConsole --output src/Hosting/Client/RpcDemo.Client.StreamConsole
* 將此專案加入根目錄的Orleans.sln方案的 *Hosting/Client* 方案資料夾(Solution Folder)中：
* dotnet sln add ./src/Hosting/Client/RpcDemo.Client.StreamConsole/RpcDemo.Client.StreamConsole.csproj --solution-folder Hosting/Client

1. 將 **RpcDemo.Interfaces.EventStreams** 專案加入至此專案的專案對專案參考(project-to-project reference)中。
2. 安裝/更新下列Nuget套件至此專案中：
   * [Microsoft.Orleans.Client](https://www.nuget.org/packages/Microsoft.Orleans.Client)
   * [Serilog](https://www.nuget.org/packages/Serilog)
   * [Serilog.Extensions.Hosting](https://www.nuget.org/packages/Serilog.Extensions.Hosting)
   * [Serilog.Sinks.Console](https://www.nuget.org/packages/Serilog.Sinks.Console)
3. 專案中的 **Program.cs** 的內容修改為如下：

* using Orleans;  
  using Orleans.Configuration;  
  using RpcDemo.Interfaces.EventStreams;  
  using Serilog;  
    
  Log.Logger = new LoggerConfiguration()  
   .WriteTo.Console()  
   .CreateLogger();  
    
  var clientBuilder = new ClientBuilder()  
   .UseLocalhostClustering()  
   .Configure<ClusterOptions>(options =>  
   {  
   options.ClusterId = "client1";  
   options.ServiceId = "Stream-Demo";  
   }).ConfigureApplicationParts(parts =>  
   {  
   parts.AddApplicationPart(typeof(IProducerGrain).Assembly).WithReferences();  
   parts.AddApplicationPart(typeof(IManualConsumerGrain).Assembly).WithReferences();  
   })  
   .ConfigureLogging(logging => logging.AddSerilog());  
    
  var client = clientBuilder.Build();  
    
  Log.Logger.Information("Press any key to start connecting to Silo");  
  Console.ReadKey();  
    
  await client.Connect();  
  Log.Logger.Information("\r\nConnected to Silo, press any key to start Orleans stream demo\r\n");  
  Console.ReadKey();  
    
  var producer = client.GetGrain<IProducerGrain>("sender1");  
  var key = Guid.NewGuid();  
  const string streamNamespace = "demo";  
  await producer.StartProducing(streamNamespace, key);  
  Log.Logger.Information("\r\nProducer Grain (sender1) is starting to produce messages in stream every second," +  
   "\r\nPress any key to create Consumer Grain (receiver1) and subscribe the stream\r\n");  
  Console.ReadKey();  
  var receiver1 = client.GetGrain<IManualConsumerGrain>("receiver1");  
  await receiver1.Subscribe(streamNamespace, key);  
  Log.Logger.Information("\r\nConsumer Grain (receiver1) is subscribing the stream," +  
   "\r\nPress any key to creat another Consumer Grain (receiver2) and subscribe the stream\r\n");  
    
  Console.ReadKey();  
  var receiver2 = client.GetGrain<IManualConsumerGrain>("receiver2");  
  await receiver2.Subscribe(streamNamespace, key);  
  Log.Logger.Information("\r\nConsumer Grain (receiver2) is subscribing the stream," +  
   "\r\nPress any key to stop producing messages\r\n");  
    
  Console.ReadKey();  
  await producer.StopProducing();  
  await receiver1.UnSubscribe();  
  await receiver2.UnSubscribe();  
    
  Log.Logger.Information("Stopped streaming in Producer Grain, press any key to disconnect from Silo and exit");  
  Console.ReadKey();  
  await client.Close();

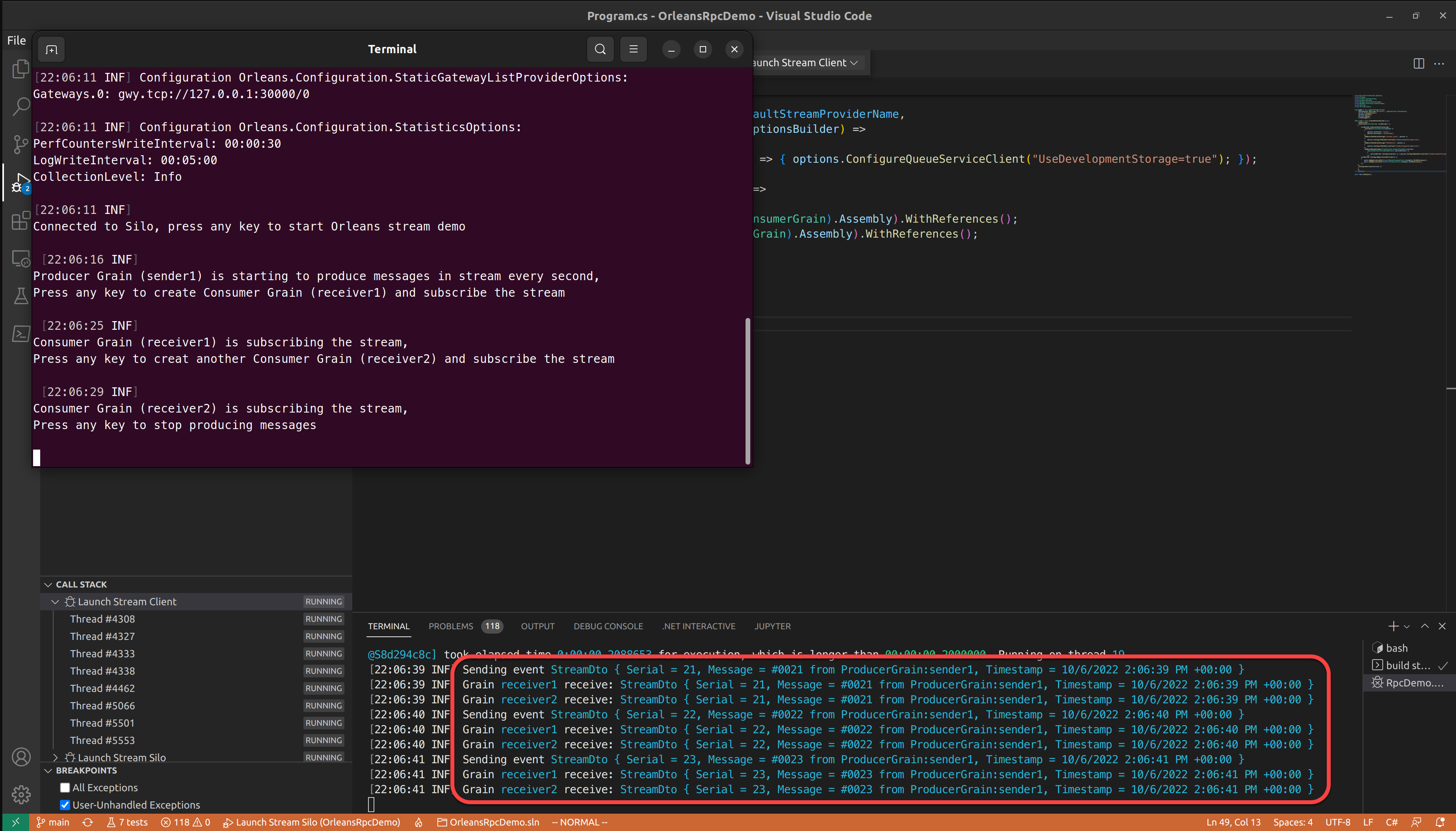
1. 將 **.vscode/tasks.json** 設定檔內新增此Stream範例專案的Server和Client建置設定：

* {  
   "version": "2.0.0",  
   "tasks": [  
   {   
   // Other tasks...  
   }  
   {  
   "label": "build stream demo",  
   "dependsOn": [  
   "build stream server",  
   "build stream client"  
   ],  
   "dependsOrder": "sequence",  
   "group": "build"  
   },  
   {  
   "label": "build stream client",  
   "command": "dotnet",  
   "type": "process",  
   "args": [  
   "build",  
   "${workspaceFolder}/src/Hosting/Client/RpcDemo.Client.StreamConsole/RpcDemo.Client.StreamConsole.csproj",  
   "/property:GenerateFullPaths=true",  
   "/consoleloggerparameters:NoSummary"  
   ],  
   "problemMatcher": "$msCompile"  
   },  
   {  
   "label": "build stream server",  
   "command": "dotnet",  
   "type": "process",  
   "args": [  
   "build",  
   "${workspaceFolder}/src/Hosting/Server/RpcDemo.Hosting.Worker/RpcDemo.Hosting.Worker.csproj",  
   "/property:GenerateFullPaths=true",  
   "/consoleloggerparameters:NoSummary"  
   ],  
   "problemMatcher": "$msCompile"  
   }  
   ]  
  }

1. 將 **.vscode/launch.json** 設定檔內新增此Stream範例專案的Server和Client除錯設定：

* {  
   "version": "0.2.0",  
   "configurations": [  
   {  
   // Other debug configurations...  
   },  
   {  
   "name": "Launch Stream Silo",  
   "type": "coreclr",  
   "request": "launch",  
   "preLaunchTask": "build stream server",  
   // If you have changed target frameworks, make sure to update the program path.  
   "program": "${workspaceFolder}/src/Hosting/Server/RpcDemo.Hosting.Worker/bin/Debug/net6.0/RpcDemo.Hosting.Worker.dll",  
   "args": [],  
   "cwd": "${workspaceFolder}/src/Hosting/Server/RpcDemo.Hosting.Worker",  
   "console": "integratedTerminal",  
   "stopAtEntry": false  
   },  
   {  
   "name": "Launch Stream Client",  
   "type": "coreclr",  
   "request": "launch",  
   "preLaunchTask": "build stream client",  
   // If you have changed target frameworks, make sure to update the program path.  
   "program": "${workspaceFolder}/src/Hosting/Client/RpcDemo.Client.StreamConsole/bin/Debug/net6.0/RpcDemo.Client.StreamConsole.dll",  
   "args": [],  
   "cwd": "${workspaceFolder}/src/Hosting/Client/RpcDemo.Client.StreamConsole",  
   "console": "externalTerminal",  
   "stopAtEntry": false  
   }  
   ]  
  }

在本機端執行Azurite Local Storage Emulator， 然後在Visual Studio Code依次執行除錯設定 **Launch Stream Silo** 和 **Launch Stream Client**，可以在Visual Studio的 Terminal 和 Debug Console 看到Stream範例的執行結果Log：



完整範例的程式碼：  
<https://github.com/windperson/OrleansRpcDemo/tree/day20>

明天再來繼續介紹事件流Grain的隱式訂閱寫法，以及直接從Client端訂閱事件流的寫法。