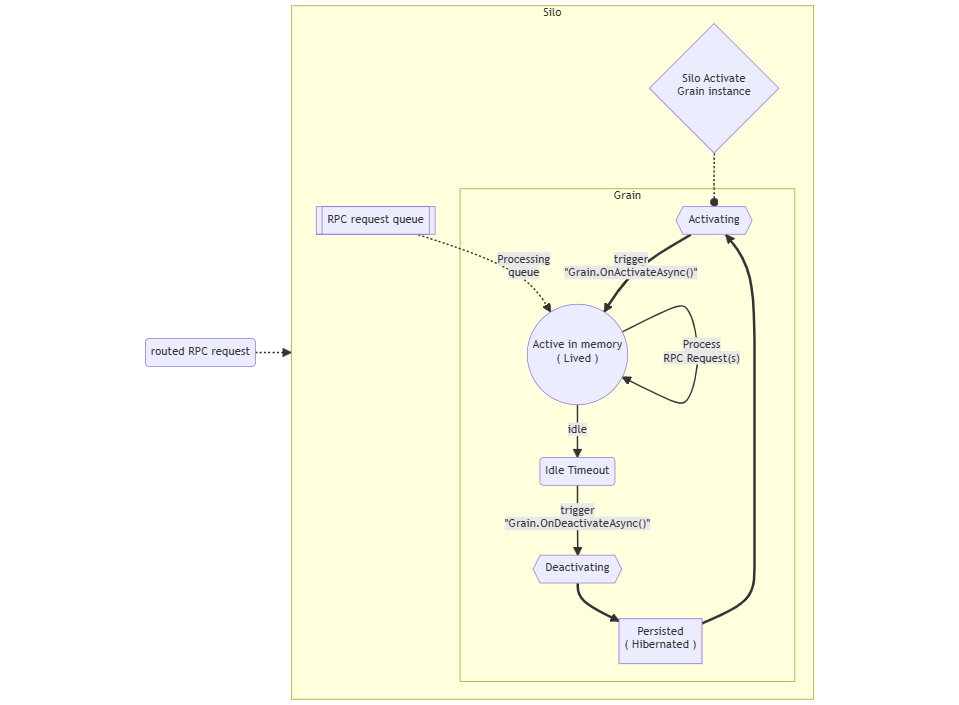
# Orleans的Grain生命週期與State狀態資料存取

Grain的State狀態資料存取，是Orleans的核心功能之一，提供讓開發者可以將Grain的狀態資料存取到外部資料庫上的功能。不過要理解其狀態資料存取的機制，需要先了解Orleans的Grain的生命週期。

## Grain的生命週期

Grain在被Silo啟動，接受RPC呼叫之後，會停留在記憶體裡一段時間，以便下一次RPC呼叫時可以非常快的速度立即反應，但當閒置一段時間沒有人呼叫該Grain之後，Grain佔用的記憶體就會被回收以節省系統資源，等下一次又有人呼叫該Grain RPC方法時，再從頭開始，這整段流程稱為[Grain的生命週期](https://learn.microsoft.com/en-us/dotnet/orleans/grains/grain-lifecycle)。



## Grain的狀態資料存取

Grain的狀態資料存取API有三個：

* 讀取：[ReadStateAsync()](https://learn.microsoft.com/en-us/dotnet/api/orleans.core.istorage.readstateasync)
* 寫入：[WriteStateAsync()](https://learn.microsoft.com/en-us/dotnet/api/orleans.core.istorage.writestateasync)
* 重設：[ClearStateAsync()](https://learn.microsoft.com/en-us/dotnet/api/orleans.core.istorage.clearstateasync)

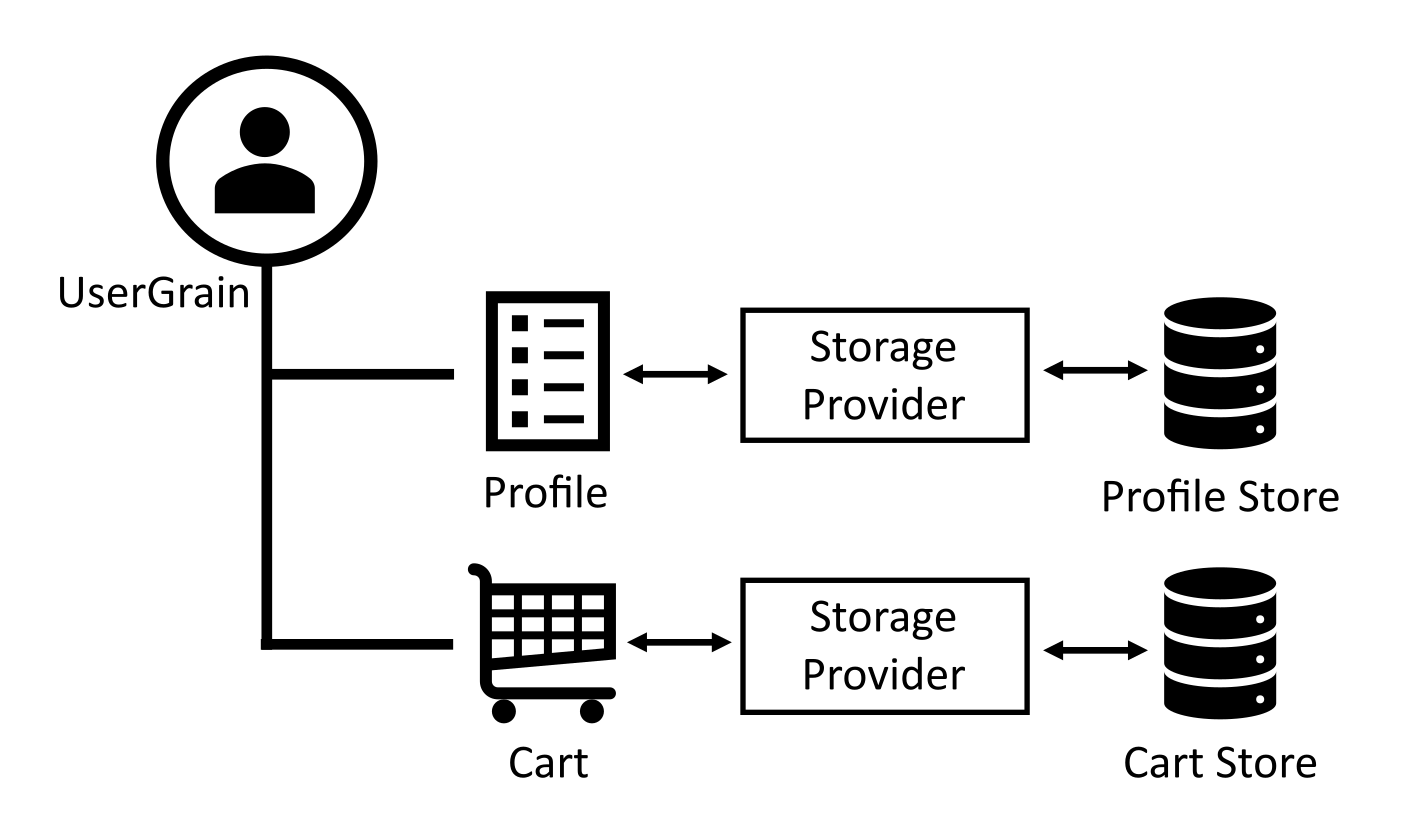
這三個API都是非同步的，建議在呼叫時使用await寫法等待其執行完成。

重設狀態資料的API，實際行為會根據Orleans的SiloBuilder配置程式碼中使用的 *Persistance Storage Provider* 種類和其實作，有些呼叫重設API的行為不會實際刪除目標資料庫的資料，而是將資料標記為已刪除，這樣就可以讓Grain在下次啟動時，不再讀取到這些資料。

Orleans提供的Grain State狀態資料存取在Grain生命週期的 ***Activating*** 階段，會觸發Grain的 OnActivateAsync() 方法，此時狀態資料Orleans框架會自動從外部資料庫讀出，將其載入到Grain的State狀態資料中，也就是說，**Grain會在其啟動時時自動讀取狀態資料**；當然，也可以在RPC方法的實作程式碼內需要狀態資料時再度呼叫 ReadStateAsync() ，以便確保狀態資料是最新版本。

當Grain的生命週期進入 ***Deactivating*** 階段時，會觸發Grain的 OnDeactivateAsync() 方法，因此也可在這個方法中，呼叫[WriteStateAsync()](https://learn.microsoft.com/en-us/dotnet/api/orleans.core.istorage.writestateasync) API方法將Grain的狀態資料存回外部資料庫中，但由於有可能因為Silo主機故障等等外部因素讓Silo突然停止，來不及讓Orleans框架呼叫此事件處理程式碼，因此建議 **當Grain State資料狀態需要保存時就立即呼叫寫入API**，以避免資料遺失。

先前的文章中有稍微提及Grain State的宣告是寫在在Grain的建構子(Constructor)程式碼宣告中，類似 .NET Core/.NET5+的依賴注入物件的寫法；因此， **一個Grain可以有一個以上的狀態資料可供存取使用**，例如設計一個購物網站顧客的UserGrain時，該Grain有兩個狀態資料：顧客基本資料與當前購物車狀態資料；然後在SiloBuilder的配置程式碼中再使用Orleans運營面向(Ops)的API設定使用不同的Storage Provider，將這兩個狀態資料Provider分別設定是存放在不同的資料庫中，例如顧客基本資料存放在SQL Server資料庫中，而購物車資料則存放在No-SQL資料庫像是Azure Table Storage雲端服務中。

  
*圖片來源：*[*舊Orleans說明文件*](https://github.com/dotnet/orleans-docs/blob/main/src/images/grain_state_1.png)

以下以一個記數器Grain為例，來說明Grain的狀態資料存取的程式碼實作：

## 記數器Grain實作

### CounterGrain程式碼撰寫

1. 在[昨天進度的原始碼git專案](https://github.com/windperson/OrleansRpcDemo/tree/day09)，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.Counter** | * .NET 6 類別庫(class library) |
| * src/Grains | * **RpcDemo.Grains.Counters** | * .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.Counter** 加入至 **RpcDemo.Grains.Counters** 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.Counter** | * + [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.Counters** | * + [Microsoft.Extensions.Logging.Abstractions](https://www.nuget.org/packages/Microsoft.Extensions.Logging.Abstractions)   + [Microsoft.Orleans.Core](https://www.nuget.org/packages/Microsoft.Orleans.Core)   + [Microsoft.Orleans.Runtime.Abstractions](https://www.nuget.org/packages/Microsoft.Orleans.Runtime.Abstractions)   + [Microsoft.Orleans.CodeGenerator.MSBuild](https://www.nuget.org/packages/Microsoft.Orleans.CodeGenerator.MSBuild) |

* 其中 **Microsoft.Orleans.Runtime.Abstractions** 套件是用於提供Grain State宣告需要用的 [PersistentStateAttribute](https://learn.microsoft.com/en-us/dotnet/api/orleans.runtime.persistentstateattribute) 屬性(Attribute)和 [IPersistentState<TState>](https://learn.microsoft.com/en-us/dotnet/api/orleans.runtime.ipersistentstate-1) 用於依賴注入的介面。

1. 定義記數器的RPC介面，在 **RpcDemo.Interfaces.Counter** 專案中，移除預設專案範本產的 *Class1.cs*，新增一個C#程式碼檔案 **ICounterGrain.cs**：

* using Orleans;  
    
  namespace RpcDemo.Interfaces.Counter;  
    
  public interface ICounterGrain : IGrainWithGuidKey  
  {  
   Task<int> GetCountAsync();  
   Task IncrementAsync();  
   Task ResetAsync();  
  }

1. 記數器Grain的實作，在 **RpcDemo.Grains.Counters** 專案中，移除預設專案範本產的 *Class1.cs*，新增一個C#程式碼檔案 **CounterGrain.cs**：

* using Microsoft.Extensions.Logging;  
  using Orleans;  
  using Orleans.Runtime;  
  using RpcDemo.Interfaces.Counter;  
    
  namespace RpcDemo.Grains.Counters;  
    
  public class CounterGrain : Grain, ICounterGrain  
  {  
   private readonly ILogger<CounterGrain> \_logger;  
   private readonly IPersistentState<CounterState> \_counterStore;  
    
   public CounterGrain(  
   [PersistentState("counter\_grain", "demo\_counters")] IPersistentState<CounterState> counterStore,  
   ILogger<CounterGrain> logger)  
   {  
   \_logger = logger;  
   \_counterStore = counterStore;  
   logger.LogInformation("CounterGrain created");  
   }  
    
   public override async Task OnActivateAsync()  
   {  
   await base.OnActivateAsync();  
   \_logger.LogInformation("CounterGrain activated");  
   \_logger.LogInformation("Current Count={Count}", \_counterStore.State.Count);  
   }  
    
   public override async Task OnDeactivateAsync()  
   {  
   //NOTE: This is not always being called  
   await \_counterStore.WriteStateAsync();  
   await base.OnDeactivateAsync();  
   \_logger.LogInformation("CounterGrain deactivated");  
   }  
    
   #region ICounterGrain implementation  
    
   public Task<int> GetCountAsync()  
   {  
   \_logger.LogInformation("GetCountAsync called");  
   return Task.FromResult(\_counterStore.State.Count);  
   }  
    
   public async Task IncrementAsync()  
   {  
   \_counterStore.State.Count++;  
   \_logger.LogInformation("IncrementAsync called, new count: {Count}", \_counterStore.State.Count);  
   await \_counterStore.WriteStateAsync();  
   }  
    
   public async Task ResetAsync()  
   {  
   \_logger.LogInformation("ResetAsync called");  
   await \_counterStore.ClearStateAsync();  
   }  
    
   #endregion  
  }  
    
  [Serializable]  
  public class CounterState  
  {  
   public int Count { get; set; } = 0;  
  }
* Grain State的宣告需要一個定義狀態存什麼內容的Data Object型別，在此範例是 CounterState 這個可序列化的類別(Class)，到時候注入的Grain State變數，其 State 屬性(Property)，就會是該型別；該依賴注入變數需要加掛 [PersistentStateAttribute](https://learn.microsoft.com/en-us/dotnet/api/orleans.runtime.persistentstateattribute) 這個屬性(Attribute)來定義依賴注入的是Grain State變數，而非一般依賴注入的參數；在此範例中，該屬性第一個參數為 counter\_grain 的字串，用來定義到時候實際儲存的『狀態名稱』（可以先假想比喻為資料庫儲存的資料表名稱），而第二個 demo\_counters，是該Grain State使用的Storage Provider名稱，到時候在SiloBuilder的配置設定程式碼中會指定該名稱實際對應到使用哪種儲存機制提供者。

接下來我們寫一個簡單的單元測試，來驗證一下這些RPC方法是否正常運作。

### CounterGrain的單元測試

1. 在程式碼專案根目錄的 *tests* 目錄之下建立一個 **CounterGrains.Tests** 的xUnit單元測試專案：

* dotnet new xunit --no-restore --name CounterGrains.Tests

1. 將 [**Microsoft.Orleans.TestingHost**](https://www.nuget.org/packages/Microsoft.Orleans.TestingHost) Nuget套件安裝到測試專案中。
2. 將 **RpcDemo.Grains.Counters** 專案加入至此測試專案的專案對專案參考(project-to-project reference)中。
3. 將 **CounterGrains.Tests** 測試專案的預設 *UnitTest1.cs* 檔刪除，新增 **CounterGrainTest.cs** 檔案為以下內容：

* using Orleans.Hosting;  
  using Orleans.TestingHost;  
  using RpcDemo.Interfaces.Counter;  
    
  namespace CounterGrains.Tests;  
    
  public class CounterGrainTest  
  {  
   private class TestSiloConfigurator : ISiloConfigurator  
   {  
   public void Configure(ISiloBuilder siloBuilder)  
   {  
   siloBuilder.AddMemoryGrainStorage("demo\_counters");  
   }  
   }  
    
   [Fact]  
   public async Task CounterGrainTest1()  
   {  
   //Arrange  
   var host = new TestClusterBuilder()  
   .AddSiloBuilderConfigurator<TestSiloConfigurator>()  
   .Build();  
   await host.DeployAsync();  
    
   //Act & Assert  
   var counterGrain = host.GrainFactory.GetGrain<ICounterGrain>(Guid.NewGuid());  
   await counterGrain.IncrementAsync();  
   Assert.Equal(1, await counterGrain.GetCountAsync());  
    
   await counterGrain.IncrementAsync();  
   await counterGrain.IncrementAsync();  
   Assert.Equal(3, await counterGrain.GetCountAsync());  
    
   await counterGrain.ResetAsync();  
   Assert.Equal(0, await counterGrain.GetCountAsync());  
    
   await host.StopAllSilosAsync();  
   }  
  }
* 在這個單元測試的程式碼中，測試用Silo的配置程式碼，有使用一個 [AddMemoryGrainStorage()](https://learn.microsoft.com/en-us/dotnet/api/orleans.hosting.memorygrainstoragesilobuilderextensions.addmemorygrainstorage) 的擴充方法，來指定使用 Orleans 框架內建的測試用In-Memory Storage Provider，將狀態資料儲存在RAM，關閉程式即消失；而呼叫的參數 demo\_counters 字串，就是在 **RpcDemo.Grains.Counters** 專案中，CounterGrain的Grain State宣告時，所指定的StorageProvider名稱。

整個完成的範例程式GitHub專案在：https://github.com/windperson/OrleansRpcDemo/tree/day10

明天繼續介紹另外兩種官方提供的Storage Provider：Azure Blob/Table Storage使用方法，以及其他Grain State在程式碼撰寫時的注意事項。