# Orleans Grain的 RPC Transaction範例實作

這裡建立一個銀行ATM轉帳的範例專案，來說明Orleans Grain的RPC Transaction的實作。

## 銀行ATM帳號轉帳範例

### 專案檔案架構建立

建立資料夾結構（以PowerShell為例）：

New-Item -ItemType Directory "./OrleansTransactionDemo/src/Shared"  
New-Item -ItemType Directory "./OrleansTransactionDemo/src/Grains"  
New-Item -ItemType Directory "./OrleansTransactionDemo/src/Hosting/Client"  
New-Item -ItemType Directory "./OrleansTransactionDemo/src/Hosting/Server"

然後PowerShell將工作目錄切入在 **OrleansTransactionDemo** 資料夾，建立一個 global.json 檔案，鎖定使用的SDK版本範圍，以確保專案使用的SDK版本相容性，避免開發環境的SDK版本不同造成的問題：

dotnet new globaljson --sdk-version 6.0.400 --roll-forward latestMinor

在 **OrleansTransactionDemo** 根目錄，使用dotnet CLI建立一個OrleansTransactionDemo.sln的解決方案檔案。

dotnet new sln

建立下列專案，並加入到解決方案中：

| 專案路徑名稱 | 專案類型 | 目標框架 |
| --- | --- | --- |
| src/Shared/Bank.Interfaces | Class Library | net6.0 |
| src/Grains/Bank.Grains | Class Library | net6.0 |
| src/Hosting/Client/Bank.Client.Console | Console App | net6.0 |
| src/Hosting/Server/Bank.Silo | Worker Service | net6.0 |

dotnet new classlib -f net6.0 --no-restore -n Bank.Interfaces -o src/Shared/Bank.Interfaces  
dotnet sln add ./src/Shared/Bank.Interfaces/Bank.Interfaces.csproj --solution-folder Shared

dotnet new classlib -f net6.0 --no-restore -n Bank.Grains -o src/Grains/Bank.Grains  
dotnet sln add ./src/Grains/Bank.Grains/Bank.Grains.csproj --solution-folder Grains

dotnet new console -f net6.0 --no-restore -n Bank.Client.Console -o src/Hosting/Client/Bank.Client.Console  
dotnet sln add ./src/Hosting/Client/Bank.Client.Console/Bank.Client.Console.csproj --solution-folder Hosting/Client

dotnet new worker -f net6.0 --no-restore -n Bank.Silo -o src/Hosting/Server/Bank.Silo  
dotnet sln add ./src/Hosting/Server/Bank.Silo/Bank.Silo.csproj --solution-folder Hosting/Server

### 專案參考設定

在 **Bank.Grains** 專案中，加入對專案參考到 **Bank.Interfaces** 專案：

dotnet add ./src/Grains/Bank.Grains/Bank.Grains.csproj reference ./src/Shared/Bank.Interfaces/Bank.Interfaces.csproj

在 **Bank.Client.Console** 專案中，加入對專案參考到 **Bank.Interfaces** 專案：

dotnet add ./src/Hosting/Client/Bank.Client.Console/Bank.Client.Console.csproj reference ./src/Shared/Bank.Interfaces/Bank.Interfaces.csproj

在 **Bank.Silo** 專案中，加入對專案參考到 **Bank.Grains** 專案：

dotnet add ./src/Hosting/Server/Bank.Silo/Bank.Silo.csproj reference ./src/Grains/Bank.Grains/Bank.Grains.csproj

各專案加入下列NuGet套件參考：

| 專案名稱 | 套件名稱 | 套件版本 |
| --- | --- | --- |
| Bank.Interfaces | * [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) | 3.6.5 3.6.5 |
| Bank.Grains | * [Microsoft.Orleans.Core](https://www.nuget.org/packages/Microsoft.Orleans.Core) * [Microsoft.Orleans.CodeGenerator.MSBuild](https://www.nuget.org/packages/Microsoft.Orleans.CodeGenerator.MSBuild) * [Microsoft.Orleans.Transactions](https://www.nuget.org/packages/Microsoft.Orleans.Transactions) | 3.6.5 3.6.5 3.6.5 |
| Bank.Client.Console | * [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) | 3.6.5 2.12.0 5.0.1 4.1.0 |
| Bank.Silo | * [Microsoft.Extensions.Hosting](https://www.nuget.org/packages/Microsoft.Extensions.Hosting) * [Microsoft.Orleans.Server](https://www.nuget.org/packages/Microsoft.Orleans.Server) * [Microsoft.Orleans.Transactions.AzureStorage](https://www.nuget.org/packages/Microsoft.Orleans.Transactions.AzureStorage) * [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) * [Serilog.Sinks.Debug](https://www.nuget.org/packages/Serilog.Sinks.Debug) | 6.0.1 3.6.5 3.6.5 2.12.0 5.0.1 4.1.0 2.0.0 |

### 程式碼撰寫

#### Bank.Interfaces專案

將專案範本預設產生的 **Class1.cs** 檔案刪除，並新增一個 **IAtmGrain.cs** 檔案，內容如下：

using Orleans;  
  
namespace Bank.Interfaces;  
  
public interface IAtmGrain : IGrainWithGuidKey  
{  
 [Transaction(TransactionOption.Create)]  
 Task Transfer(IAccountGrain from, IAccountGrain to, decimal amount);  
}

此為ATMGrain的介面，定義一個轉帳的RPC方法 Transfer()，並標記此方法為啟始分散式交易階段的屬性（TransactionOption.Create）。

新增 **IAccountGrain.cs** 檔案，內容如下：

using Orleans;  
  
namespace Bank.Interfaces;  
  
public interface IAccountGrain : IGrainWithStringKey  
{  
 [Transaction(TransactionOption.CreateOrJoin)]  
 Task<decimal> GetBalance();  
  
 [Transaction(TransactionOption.Join)]  
 Task Withdraw(decimal amount);  
   
 [Transaction(TransactionOption.Join)]  
 Task Deposit(decimal amount);  
}

這定義的是一個帳戶Grain的介面，有三個方法，分別是取得目前帳戶餘額（GetBalance）、提款（Withdraw）、存款（Deposit）。由於帳戶Grain在使用上除了取得目前帳戶餘額這個方法可單獨或是跟其他方法聯合使用之外，另外兩個提款和存款得配合AtmGrain才會動作，因此除了取得目前帳戶餘額這個方法用是可啟使或加入分散式交易階段的屬性（TransactionOption.CreateOrJoin）之外，其他兩個方法都標記為加入分散式交易階段屬性（TransactionOption.Join）。

新增 **InsufficientAmountException.cs** 檔案，內容如下：

namespace Bank.Interfaces;  
  
public class InsufficientAmountException : Exception  
{  
 public decimal Amount { get; }  
  
 public decimal Balance { get; }  
  
 public string Account { get; }  
  
 public InsufficientAmountException(string message, string account, decimal amount, decimal balance) : base(message)  
 {  
 Account = account;  
 Amount = amount;  
 Balance = balance;  
 }  
}

此為當餘額不足時會拋出的例外，包含帳戶、提款金額、餘額三個屬性。

#### Bank.Grains專案

將專案範本預設產生的 **Class1.cs** 檔案刪除，並新增一個 **AtmGrain.cs** 檔案，內容如下：

using Bank.Interfaces;  
using Orleans;  
  
namespace Bank.Grains;  
  
public class AtmGrain : Grain, IAtmGrain  
{  
 public Task Transfer(IAccountGrain from, IAccountGrain to, decimal amount)  
 {  
 var fromAccount = from.GetPrimaryKeyString();  
 var toAccount = to.GetPrimaryKeyString();  
 if(fromAccount == toAccount)  
 {  
 throw new ArgumentException("Cannot transfer to the same account");  
 }  
  
 return Task.WhenAll(from.Withdraw(amount), to.Deposit(amount));  
 }  
}

AtmGrain的轉帳方法實作基本上就是呼叫兩個帳戶Grain各自的提款和存款方法，並且使用 Task.WhenAll() 方法將兩個方法的執行結果合併成一個回傳，也就是要等這兩個Grain各自的RPC方法執行成功後，此轉帳方法的執行即為成功。

新增一個 **AccountGrain.cs** 檔案，內容如下：

using Bank.Interfaces;  
using Microsoft.Extensions.Logging;  
using Orleans;  
using Orleans.Transactions.Abstractions;  
  
namespace Bank.Grains;  
  
public record class Balance  
{  
 public decimal Amount { get; set; } = 1\_000;  
}  
  
public class AccountGrain : Grain, IAccountGrain  
{  
 private readonly ITransactionalState<Balance> \_balance;  
 private readonly ILogger<AccountGrain> \_logger;  
  
 public AccountGrain([TransactionalState("balance")] ITransactionalState<Balance> balance, ILogger<AccountGrain> logger)  
 {  
 \_balance = balance;  
 \_logger = logger;  
 }  
   
 public Task<decimal> GetBalance()  
 {  
 return \_balance.PerformRead(balance => balance.Amount);  
 }  
   
 public Task Withdraw(decimal amount)  
 {  
 return \_balance.PerformUpdate(balance =>  
 {  
 if (balance.Amount < amount)  
 {  
 var account = this.GetPrimaryKeyString();  
 var balanceAmount = balance.Amount;  
 var errorMessage =  
 $"Withdrawing \"{amount}\" from account {{{account}}} is not possible. Current Balance is {balanceAmount}";  
 \_logger.LogError(errorMessage);  
 throw new InsufficientAmountException(errorMessage, account, amount, balanceAmount);  
 }  
  
 \_logger.LogInformation("Withdrawing \"{Amount}\" from account {Account)}", amount, this.GetPrimaryKeyString());  
 balance.Amount -= amount;  
 });  
 }  
  
  
 public Task Deposit(decimal amount)  
 {  
 return \_balance.PerformUpdate(balance =>  
 {  
 \_logger.LogInformation("Depositing \"{Amount}\" to account {Account)}", amount, this.GetPrimaryKeyString());  
 balance.Amount += amount;  
 });  
 }  
}

在此帳戶Grain實作建構式中注入了 ITransactionalState<Balance> 這個用來儲存帳戶總金額的交易狀態變數，而在裝飾的屬性 [[TransactionalState]](https://learn.microsoft.com/dotnet/api/orleans.transactions.abstractions.transactionalstateattribute.-ctor#orleans-transactions-abstractions-transactionalstateattribute-ctor(system-string-system-string)) 內指定底層Provider要使用儲存名稱為balance，第二個Transaction Storage Provider名稱不填入的時候，會使用在Silo配置程式碼裡指令的預設Provider。

在GetBalance和Deposit方法中，由於一個是讀取，而另一個是加錢進此帳戶，基本上沒有什麼會出錯的可能，所以直接用 PerformRead() 和 PerformUpdate() 的非同步方法來執行，而在Withdraw方法中，由於是提款，所以要先檢查帳戶餘額是否足夠，如果不夠就拋出 InsufficientAmountException 例外。

#### Bank.Silo專案

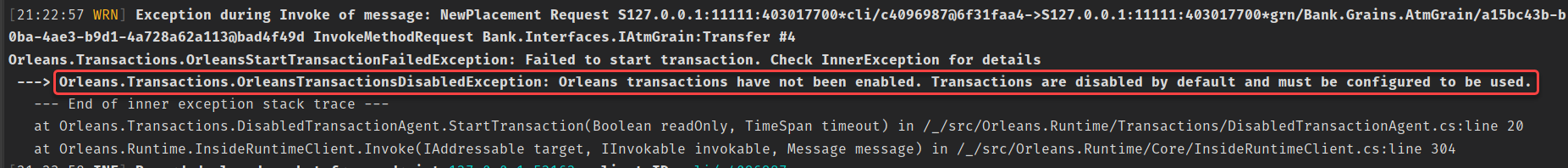
將專案範本預設產生的 **Worker.cs** 檔案刪除，並將原本的 **Program.cs** 檔案修改為如下：

using Bank.Grains;  
using Orleans;  
using Orleans.Configuration;  
using Orleans.Hosting;  
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(siloBuilder =>  
 {  
 siloBuilder.UseLocalhostClustering()  
 .Configure<ClusterOptions>(options =>  
 {  
 options.ClusterId = "silo1";  
 options.ServiceId = "Bank-Transfer-Demo";  
 })  
 .AddAzureTableTransactionalStateStorageAsDefault( options =>  
 {  
 options.ConfigureTableServiceClient("UseDevelopmentStorage=true");  
 })  
 .UseTransactions()  
 .ConfigureLogging(logging => logging.AddSerilog())  
 .ConfigureApplicationParts(parts =>  
 {  
 parts.AddApplicationPart(typeof(AtmGrain).Assembly).WithReferences();  
 parts.AddApplicationPart(typeof(AccountGrain).Assembly).WithReferences();  
 });  
 })  
 .ConfigureServices(services =>  
 {  
 })  
 .Build();  
  
await host.RunAsync();

在 [**Microsoft.Orleans.Transactions.AzureStorage**](https://www.nuget.org/packages/Microsoft.Orleans.Transactions.AzureStorage/) NuGet套件中，有提供一個 [AddAzureTableTransactionalStateStorageAsDefault()](https://learn.microsoft.com/dotnet/api/orleans.hosting.azuretablesilobuilderextensions.addazuretabletransactionalstatestorageasdefault) 的擴充方法，可以直接將Azure Table Storage作為預設的Transaction Storage Provider，而等下我們會使用Azurite來跑地端的Azure Storage Emulator來提供Azure Table Storage服務，所以在 ConfigureTableServiceClient() 方法中，指定 UseDevelopmentStorage=true。

而預設Orleans Silo的Transaction功能是停用的，需要呼叫 [UseTransactions()](https://learn.microsoft.com/dotnet/api/orleans.hosting.silobuilderextensions.usetransactions) 這個擴充方法來啟用。

沒啟用的Silo在呼叫到相關的Grain RPC方法時，會拋出 [OrleansTransactionDisabledException](https://learn.microsoft.com/dotnet/api/orleans.transactions.orleanstransactionsdisabledexception) 例外：



#### Bank.Client.Console專案

將專案範本產生的 **Program.cs** 檔案修改為如下：

using Bank.Interfaces;  
using Orleans;  
using Orleans.Configuration;  
using Orleans.Transactions;  
using Serilog;  
  
Log.Logger = new LoggerConfiguration()  
 .WriteTo.Console()  
 .CreateLogger();  
  
var clientBuilder = new ClientBuilder()  
 .UseLocalhostClustering()  
 .Configure<ClusterOptions>(options =>  
 {  
 options.ClusterId = "client1";  
 options.ServiceId = "Bank-Transfer-Demo";  
 })  
 .ConfigureLogging(logging => logging.AddSerilog())  
 .ConfigureApplicationParts(parts =>  
 {  
 parts.AddApplicationPart(typeof(IAtmGrain).Assembly).WithReferences();  
 parts.AddApplicationPart(typeof(IAccountGrain).Assembly).WithReferences();  
 });  
  
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 Bank Transfer demo\r\n");  
Console.ReadKey();  
  
var atmGrain = client.GetGrain<IAtmGrain>(Guid.NewGuid());  
var account1 = client.GetGrain<IAccountGrain>("account1");  
var account2 = client.GetGrain<IAccountGrain>("account2");  
  
try  
{  
 while (true)  
 {  
 Log.Logger.Information("Transfer 100 from account1 to account2");  
 await atmGrain.Transfer(account1, account2, 100);  
  
 var balance1 = await account1.GetBalance();  
 var balance2 = await account2.GetBalance();  
 Log.Logger.Information("After transfer: account1: {Balance1}, account2: {Balance2}", balance1, balance2);  
 await Task.Delay(TimeSpan.FromSeconds(1));  
 }  
}  
catch (OrleansTransactionAbortedException ex)  
{  
 Log.Logger.Error(ex, "Error during transfer");  
  
 var balance1 = await account1.GetBalance();  
 var balance2 = await account2.GetBalance();  
 Log.Logger.Information("Right now account1: {Balance1}, account2: {Balance2}", balance1, balance2);  
}  
catch (Exception ex)  
{  
 Log.Logger.Error(ex, "Other Error");  
}  
  
Log.Logger.Information("Press any key to transfer back");  
Console.ReadKey();  
  
try  
{  
 while (true)  
 {  
 Log.Logger.Information("Transfer 100 from account1 to account2");  
 await atmGrain.Transfer(account2, account1, 100);  
  
 var balance1 = await account1.GetBalance();  
 var balance2 = await account2.GetBalance();  
 Log.Logger.Information("After transfer: account1: {Balance1}, account2: {Balance2}", balance1, balance2);  
 await Task.Delay(TimeSpan.FromSeconds(1));  
 }  
}  
catch (OrleansTransactionAbortedException ex)  
{  
 Log.Logger.Error(ex, "Error during transfer");  
 var balance1 = await account1.GetBalance();  
 var balance2 = await account2.GetBalance();  
 Log.Logger.Information("Right now account1: {Balance1}, account2: {Balance2}", balance1, balance2);  
}  
catch (Exception ex)  
{  
 Log.Logger.Error(ex, "Other Error");  
}  
  
Log.Logger.Information("Bank Transfer Demo complete, press any key to disconnect from Silo and exit");  
Console.ReadKey();  
await client.Close();

這邊使用兩個帳號轉帳過來又轉帳回去的實驗範例來驗證AtmGrain的 Transfer() 這個RPC方法的Orleans交易功能，不會有一邊扣到變0時另一邊的Deposit還會成功。

在配置ClientBuilder時，記得Client這邊的ServiceId要與Silo的ServiceId相同，否則Client無法連線到Silo。

要執行此範例，記得要先啟動自己地端的Azurite，執行Bank.Silo專案後，再執行Bank.Client.Console專案。

完整的範例程式碼： <http://www.github.com/windperson/OrleansTransactionDemo>

明天繼續示範另一個使用Orleans Transaction的範例，以及如何為使用Orleans Transaction的Grain撰寫測試。