(T8)比較LazyLoading延遲執行(Select、Where、Take、Skip)、EagerLoading立刻執行(aggregate、ToList)  
CourseGUID: 5ba9a6fe-7475-4b0c-8b99-bbcf7f5e2e1c  
=======================================================================  
(T8)比較LazyLoading延遲執行(Select、Where、Take、Skip)、EagerLoading立刻執行(aggregate、ToList)  
=======================================================================  
0. Summary

-----------

1. New Project

1.1. Create New Project : Sample

-----------

2. Sample : Program.cs  
=======================================================================

0. Summary

Based on the behavior of query execution, Linq can be classified into 2 categories.

1. Deferred Operators/Lazy Operators/Lazy Loading use deferred execution.

E.g.  select, where, Take, Skip ...

2. Immediate Operators/Greedy Operators/Eager Loading use immediate execution.

E.g.  count, average, min, max, ToList ...

=============================================

1. New Project

1.1. Create New Project : Sample

File --> New --> Project... -->

Visual C# -->  **Console App** **(.Net Framework)** -->

Name: **Sample**

Graphical user interface, application, email

Description automatically generated



Graphical user interface, text, application, Excel

Description automatically generated

=============================================

2. Sample : Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using OnLineGame;

namespace Sample

{

    class Program

    {

        static void Main(string[] args)

        {

            //1. =============================================

            //LinqDeferredExecutionExample();

            Console.WriteLine("1. LinqDeferredExecutionExample(); ============= ");

            LinqDeferredExecutionExample();

            //2. =============================================

            //LinqImmediateExecutionExample();

            Console.WriteLine("2. LinqImmediateExecutionExample(); ============= ");

            LinqImmediateExecutionExample();

            //3. =============================================

            //LinqImmediateExecutionExample2();

            Console.WriteLine("3. LinqImmediateExecutionExample2(); ============= ");

            LinqImmediateExecutionExample2();

            Console.ReadLine();

        }

        //1. =============================================

        //LinqDeferredExecutionExample();

        static void LinqDeferredExecutionExample()

        {

            List<Gamer> gamersList = GamerHelper.GetSampleGamers();

            //1.

            //Deferred /Lazy Operators use deferred execution.

            //E.g.select, where, Take, Skip...

            //LINQ Query has been defined but not executed yet at this point.

            //If the Linq query has been executed at this point,

            //the result should not include

            //new Gamer { Id = 4, Name = "Name4", Score = 100 }

            IEnumerable<Gamer> gamerScoreEqualTo100 =

              from gamer in gamersList

              where gamer.Score == 100

              select gamer;

            // Add a new Gamer object with Score=100 to the source list.

            gamersList.Add(new Gamer { Id = 4, Name = "Name4", Score = 100 });

            //The above LINQ Query has been actually executed here

            //when using foreach loop.

            //the result includes

            //new Gamer { Id = 4, Name = "Name4", Score = 100 }

            foreach (var gamerScoreEqualTo100Item in gamerScoreEqualTo100)

            {

                Console.WriteLine(gamerScoreEqualTo100Item);

            }

        }

        //Id==1,Name==Name1,Score==100

        //Id==2,Name==Name2,Score==100

        //Id==4,Name==Name4,Score==100

       //2. =============================================

        //LinqImmediateExecutionExample();

        private static void LinqImmediateExecutionExample()

        {

            List<Gamer> gamersList = GamerHelper.GetSampleGamers();

            //2.

            //Immediate/Greedy Operators use immediate execution.

            //E.g.count, average, min, max, ToList...

            //ToList() which is a Immediate/Greedy Operator,

            //so LINQ Query has been executed immediately at this point.

            //the LINQ Query is executed immediately at this point.

            //the result does not include

            //new Gamer { Id = 4, Name = "Name4", Score = 100 }

            List<Gamer> gamerScoreEqualTo100 =

                (from gamer in gamersList

                 where gamer.Score == 100

                 select gamer).ToList();

            //Add a new Gamer object with Score=100 to the source list.

            //This will not affect on the result

            //because the Linq query has been already executed.

            gamersList.Add(new Gamer { Id = 4, Name = "Name4", Score = 100 });

            //The above LINQ Query has been actually executed

            //when using .ToList()

            //the result will not include

            //new Gamer { Id = 4, Name = "Name4", Score = 100 }

            foreach (var gamerScoreEqualTo100Item in gamerScoreEqualTo100)

            {

                Console.WriteLine(gamerScoreEqualTo100Item);

            }

        }

        //Id==1,Name==Name1,Score==100

        //Id==2,Name==Name2,Score==100

       //3. =============================================

        //LinqImmediateExecutionExample2();

        static void LinqImmediateExecutionExample2()

        {

            List<Gamer> gamersList = GamerHelper.GetSampleGamers();

            //2.

            //Immediate/Greedy Operators use immediate execution.

            //E.g.count, average, min, max, ToList...

            //Count() which is a Immediate/Greedy Operator,

            //so LINQ Query has been executed immediately at this point.

            //the LINQ Query is executed immediately at this point.

            //the result does not include

            //new Gamer { Id = 4, Name = "Name4", Score = 100 }

            int gamerScoreEqualTo100Count =     //2

                (from gamer in gamersList

                 where gamer.Score == 100

                 select gamer).Count();

            //Add a new Gamer object with Score=100 to the source list.

            //This will not affect on the result

            //because the Linq query has been already executed.

            gamersList.Add(new Gamer { Id = 4, Name = "Name4", Score = 100 });

            //The above LINQ Query has been actually executed

            //when using .Count()

            //the result will not include

            //new Gamer { Id = 4, Name = "Name4", Score = 100 }

            Console.WriteLine($"gamerScoreEqualTo100Count=={gamerScoreEqualTo100Count}");

        }

        //gamerScoreEqualTo100Count==2

    }

}

namespace OnLineGame

{

    public class Gamer

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public int Score { get; set; }

        public override string ToString()

        {

            return $"Id=={Id},Name=={Name},Score=={Score}";

        }

    }

    public class GamerHelper

    {

        public static List<Gamer> GetSampleGamers()

        {

            return new List<Gamer>

            {

                new Gamer { Id = 1, Name = "Name1", Score =100 },

                new Gamer { Id = 2, Name = "Name2", Score =100 },

                new Gamer { Id = 3, Name = "Name3", Score =200 }

            };

        }

        // Create a List<Gamer> which contains numberOfGamers gamers.

        public static List<Gamer> GetSampleGamers(int numberOfGamers)

        {

            //int numberOfGamers = 10;

            List<Gamer> gamerList = new List<Gamer>();

            for (int i = 1; i <= numberOfGamers; i++)

            {

                Random rnd = new Random();

                int rndScore = rnd.Next(1000, 6000); // creates a number between 1000 and 6000

                gamerList.Add(new Gamer { Id = i, Name = $"Name{i}", Score = rndScore });

            }

            return gamerList;

        }

    }

}

