(T4)比較RAM的Stack、Heap。比較Struct、Class。比較ValueType、ReferenceType  
CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92  
=======================================================================  
(T4)比較RAM的Stack、Heap。比較Struct、Class。比較ValueType、ReferenceType  
=======================================================================  
0. Summary

1. Create New Project

2. Program  
=======================================================================

0. Summary

0.

Struct V.S. Class

RAM: Stack V.S. Heap

Value Type

Reference Type

--------------------------------------------------------

1.

Struct V.S. Class

Value type V.S. Reference type

Reference:

<http://net-informations.com/faq/net/stack-heap.htm>

----------------------------

1.1.

Both Stack and Heap are in the RAM.

Constructor is a special method and will be called when the object is created.

Destructor is a special method and will be called when the object is destroyed.

Sealed types means it can not has sub-class.

----------------------------

1.2.

Value Type

// E.g.  int i = 2;

Value type stores its value directly on stack

which is used for static memory allocation

when the program is compiled.

Value types are destroyed immediately after the scope is lost.

----------------------------

1.3.

Reference Type

// E.g. ClassA a1 = new ClassA()

RHS: Use ClassA as template to create its instance object, this is the "Value".

LHS: declare a1 as "Object Reference Variable" with the type ClassA.

Reference type stores its "Object Reference Variable" in Stack which is static memory allocation,

and stores its "value" in heap at run time which is for dynamic memory allocation.

Thus, heap is a bit slower than stack.

The heap size is only limited by the size of virtual memory.

The "Object Reference Variable" in LHS is destroyed after the scope is lost.

The object "value" in RHS is later destroyed by garbage collector.

----------------------------

1.4.

A struct is a value type.

When copy struct1 into struct2.

The changes in struct2 will not affect struct1,

because the value of struct1 and struct2 are

stored in different memory location in Stack.

struct can't have destructor.

struct can have constructor with parameters,

but can not have constructor without any parameter.

Struct can not extend other Class, but can implement Interface.

Struct is sealed type can not be extended by other Class or other Struct.

E.g.

int (System.lnt32), double(System.Double) etc.

----------------------------

1.5.

A class is a reference type.

When copy Class1 into Class2.

The changes in Class2 will reflect back Class1,

because both "Object Reference Variable" (LHS) of Class1 and Class2 are stored in Stack,

but both "Object Reference Variable" are pointing to the same heap location

which stored its "Object Value" (RHS)

Class can have destructor.

Class can have any kind of constructor.

Class can extend other Class, also can implement Interface.

Sealed keyword can prevent a class from being inherited.

--------------------------------------------------------

2.

Value Type V.S. Reference Type

E.g.

//int i1 = 2;

//int i2 = i1;

//i2++;

...

//PersonClass pc1 = new PersonClass();

//pc1.Id = 1;

//pc1.Name = "Name01";

//PersonClass pc2 = pc1;

//pc2.Id = 2;

//pc2.Name = "Name02";

Chart

Description automatically generated with low confidence

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

1. Create New Project

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

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

Name: **Sample**

Graphical user interface, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, Excel

Description automatically generated

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

2. Program

using System;

namespace Sample

{

    class Program

    {

        static void Main(string[] args)

        {

            int i1 = 2;

            int i2 = i1;

            i2++;

            Console.WriteLine("i1 = {0} , i2 = {1}", i1, i2);

            // i1 = 2 , i2 = 3

            Console.WriteLine("---------------------------------------------");

            PersonClass pc1 = new PersonClass();

            pc1.Id = 1;

            pc1.Name = "Name01";

            Console.WriteLine("pc1.Id = {0} , pc1.Name = {1}", pc1.Id, pc1.Name);

            // pc1.Id = 1 , pc1.Name = Name01

            Console.WriteLine("---------------------------------------------");

            PersonClass pc2 = pc1;

            pc2.Id = 2;

            pc2.Name = "Name02";

            Console.WriteLine("pc1.Id = {0} , pc1.Name = {1}", pc1.Id, pc1.Name);

            Console.WriteLine("pc2.Id = {0} , pc2.Name = {1}", pc2.Id, pc2.Name);

            //pc1.Id = 2 , pc1.Name = Name02

            //pc2.Id = 2 , pc2.Name = Name02

            Console.ReadLine();

        }

    }

}

public interface IPersonA

{

}

public interface IPersonB

{

}

public class ClassA

{

}

public class PersonClass : ClassA, IPersonA, IPersonB

{

    // Private Fields ------------------------

    private int \_id;

    private string \_name;

    // Constructor ------------------------

    public PersonClass(int id, string name)

    {

        \_id = id;

        this.\_name = name;

    }

    public PersonClass()

    {

        \_id = 0;

        \_name = "";

    }

    // Destructor ------------------------

    ~PersonClass()

    {

        \_id = 0;

        \_name = "";

    }

    // Public Properties ------------------------

    public int Id

    {

        get { return \_id;}

        set { \_id = value;}

    }

    public string Name

    {

        get { return \_name; }

        set { \_name = value; }

    }

    // Method ------------------------

    public void PrintName()

    {

        Console.WriteLine("Id = {0} && Name = {1}", \_id, \_name);

    }

}

////Struct can not extend other Class, but can implement Interface.

//public struct PersonStruct : ClassA, IPersonA, IPersonB   //compile error

public struct PersonStruct : IPersonA, IPersonB

{

    // Private Fields ------------------------

    private int \_id;

    private string \_name;

    // Constructor ------------------------

    //struct can have constructor with parameters,

    //but can not have constructor without any parameter.

    public PersonStruct(int id, string name)

    {

        \_id = id;

        \_name = name;

    }

    //public PersonStruct()

    //{

    //    \_id = 0;

    //    \_name = "";

    //}

    // Destructor ------------------------

    //// struct can't have destructor.

    //~PersonStruct()

    //{

    //    \_id = 0;

    //    \_name = "";

    //}

    // Public Properties ------------------------

    public int Id

    {

        get { return \_id; }

        set { \_id = value; }

    }

    public string Name

    {

        get { return \_name; }

        set { \_name = value; }

    }

    // Method ------------------------

    public void PrintName()

    {

        Console.WriteLine("Id = {0} && Name = {1}", \_id, \_name);

    }

}

////Struct is sealed type can not be extended by other Class or other Struct.

//public class ClassB : PersonStruct

//{

//}

public sealed class ClassB : PersonClass

{

}

////Sealed keyword can prevent a class from being inherited.

//public class ClassC : ClassB

//{

//}

Text

Description automatically generated