

C#的两种数据类型：值类型和引用类型

什么是值类型，什么是引用类型

概念：值类型直接存储其值，而引用类型存储对其值的引用。部署：托管堆上部署了所有引用类型。

引用类型：基类为Objcet

值类型：均隐式派生自System.ValueType：

值类型：	byte, short, int, long, float, double, decimal, char, bool 和 struct 统称为值类型。
引用类型：	string 和 class统称为引用类型。

- 值类型变量声明后，不管是否已经赋值，编译器为其分配内存。
- 引用类型当声明一个类时，只在栈中分配一小片内存用于容纳一个地址，而此时并没有为其分配堆上的内存空间。当使用 new 创建一个类的实例时，分配堆上的空间，并把堆上空间的地址保存到栈上分配的小片空间中。
- 值类型的实例通常是在线程栈上分配的（静态分配），但是在某些情形下可以存储在堆中。
- 引用类型的对象总是在进程堆中分配（动态分配）。

我们来看下面一段代码：

```
namespace ConsoleApplication1
{
    3 个引用
    public struct SomeVale
    {
        public int Num;
    }

    3 个引用
    public class SomeClass
    {
        public int Num;
    }

    0 个引用
    class Program
    {
        0 个引用
        static void Main(string[] args)
        {
            SomeClass sc1 = new SomeClass();
            sc1.Num = 10;
            SomeClass sc2 = sc1;
            sc2.Num = 11;

            SomeVale sv1 = new SomeVale();
            sv1.Num = 20;
            SomeVale sv2 = sv1;
            sv2.Num = 22;

            Console.WriteLine("sc1.Num:{0}\t sc2.Num:{1};", sc1.Num,sc2.Num);
            Console.WriteLine("sv1.Num:{0}\t sv2.Num:{1}", sv1.Num,sv2.Num);
        }
    }
}
```

公告

昵称：[进修的stone](#)
园龄：[4年11个月](#)
粉丝：[7](#)
关注：[3](#)
[+加关注](#)

< 2019年3月 >						
日	一	二	三	四	五	六
24	25	26	27	28	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

搜索

找找看

谷歌搜索

常用链接

[我的随笔](#)
[我的评论](#)
[我的参与](#)
[最新评论](#)
[我的标签](#)

随笔分类

[.net\(1\)](#)
[C#](#)
[java](#)
[unity\(1\)](#)

随笔档案

[2016年9月 \(2\)](#)

文章分类

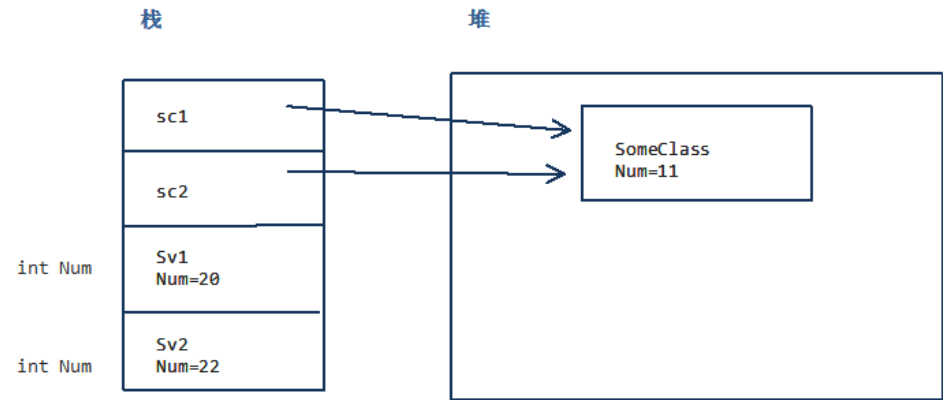
[C#\(1\)](#)
[java](#)
[unity](#)

最新评论

1. Re:C#的两种数据类型：值类型和引用类型
写的非常的详细，但有一点，上文中在C#中，我们用struct/class来声明一个类型为值类型/引用类型。考虑下面的例子：和5. 辨明值类型和引用类型的使用场合这两小节下的内容一致，不知道是为什么。.....
--没有什么大不了
2. Re:C#的两种数据类型：值类型和引用类型
真厉害，一搜博客园 C#资料都是你的。。。
--猪zzZ

输出结果：

```
C:\Windows\system32\cmd.exe
sc1.Num:11      sc2.Num:11;
sv1.Num:20      sv2.Num:22
请按任意键继续. . .
```



值类型在栈内分配空间大小因变量类型而异；
引用类型在栈内的空间大小相同；

1. 通用类型系统

C#中，变量是值还是引用仅取决于其数据类型。
C#的基本数据类型都以平台无关的方式来定义。C#的预定义类型并没有内置于语言中，而是内置于.NET Framework中。.NET使用通用类型系统（CTS）定义了可以在中间语言（IL）中使用的预定义数据类型，所有面向.NET的语言都最终被编译为IL，即编译为基于CTS类型的代码。

例如，在C#中声明一个int变量时，声明的实际上是CTS中System.Int32的一个实例。这具有重要的意义：

- 确保IL上的强制类型安全；
- 实现了不同.NET语言的互操作性；
- 所有的数据类型都是对象。它们可以有方法，属性，等。例如：

```
int i;

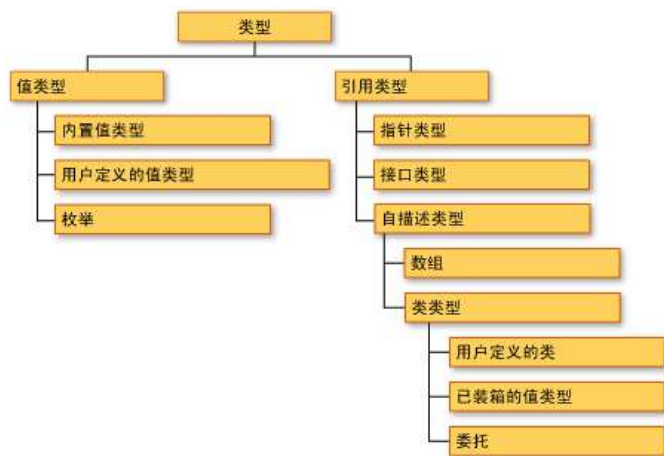
i = 1;

string s;

s = i.ToString();
```

以下关系图（来自MSDN）说明了这几种类型是如何相关的。注意，类型的实例可以只是值类型或自描述类型，即使这些类型有子类别也是如此。

类型类别：



2.值类型

C#的所有值类型均隐式派生自System.ValueType:

结构体: struct (直接派生于System.ValueType) ;

数值类型:

整型: sbyte (System.SByte的别名) , short (System.Int16) , int (System.Int32) , long (System.Int64) , byte (System.Byte) , ushort (System.UInt16) , uint (System.UInt32) , ulong (System.UInt64) , char (System.Char) ;

浮点型: float (System.Single) , double (System.Double) ;

用于财务计算的高精度decimal型: decimal (System.Decimal) 。

bool型: bool (System.Boolean的别名) ;

用户定义的结构体 (派生于System.ValueType) 。

枚举: enum (派生于System.Enum) ;

可空类型 (派生于System.Nullable<T>泛型结构体, T?实际上是System.Nullable<T>的别名) 。

每种值类型均有一个隐式的默认构造函数来初始化该类型的默认值。例如:

```
int i = new int();
```

等价于:

```
Int32 i = new Int32();
```

等价于:

```
int i = 0;
```

等价于:

```
Int32 i = 0;
```

引用类型和值类型都继承自System.Object类。不同的是,几乎所有的引用类型都直接从System.Object继承,而值类型则继承其子类,即直接继承System.ValueType。System.ValueType直接派生于System.Object。即System.ValueType本身是一个类类型,而不是值类型。其关键在于ValueType重写了Equals()方法,从而对值类型按照实例的值来比较,而不是引用地址来比较。

可以用Type.IsValueType属性来判断一个类型是否为值类型:

```
TestType testType = new TestType ();
if (testType.GetType().IsValueType)
{
```

```
Console.WriteLine("{0} is value type.", testType.ToString());  
}
```

3. 引用类型

C#有以下一些引用类型：

数组（派生于System.Array）

用户定义的以下类型：

类：class（派生于System.Object）；

接口：interface（接口不是一个“东西”，所以不存在派生于何处的问题。Anders在《C# Programming Language》中说，接口只是表示一种约定[contract]）；

委托：delegate（派生于System.Delegate）。

object（System.Object的别名）；

字符串：string（System.String的别名）。

可以看出：

引用类型与值类型相同的是，结构体也可以实现接口；

引用类型可以派生出新的类型，而值类型不能；

引用类型可以包含null值，值类型不能（可空类型功能允许将 null 赋给值类型）；

引用类型变量的赋值只复制对对象的引用，而不复制对象本身。而将一个值类型变量赋给另一个值类型变量时，将复制包含的值。

对于最后一条，经常混淆的是string。我曾经在一本书的一个早期版本上看到String变量比string变量效率高；我还经常听说String是引用类型，string是值类型，等等。例如：

```
string s1 = "Hello, ";  
string s2 = "world!";  
string s3 = s1 + s2;//s3 is "Hello, world!"
```

这确实看起来像一个值类型的赋值。再如：

```
string s1 = "a";  
string s2 = s1;  
s1 = "b";//s2 is still "a"
```

改变s1的值对s2没有影响。这更使string看起来像值类型。实际上，这是运算符重载的结果，当s1被改变时，.NET在托管堆上为s1重新分配了内存。这样的目的，是为了将做为引用类型的string实现为通常语义下的字符串。

4. 值类型和引用类型在内存中的部署

经常听说，并且经常在书上看到：值类型部署在栈上，引用类型部署在托管堆上。实际上并没有这么简单。

MSDN上说：托管堆上部署了所有引用类型。这很容易理解。当创建一个应用类型变量时：

```
object reference = new object();
```

关键字new将在托管堆上分配内存空间，并返回一个该内存空间的地址。左边的reference位于栈上，是一个引用，存储着一个内存地址；而这个地址指向的内存（位于托管堆）里存储着其内容（一个System.Object的实例）。下面为了方便，简称引用类型部署在托管堆上。

再来看值类型。《C#语言规范》上的措辞是“结构体不要求在堆上分配内存（However, unlike classes, structs are value types and do not require heap allocation）”而不是“结构体在栈上分配内存”。这不可避免容易让人感到困惑：值类型究竟部署在什么地方？

4.1 数组

考虑数组：

```
int[] reference = new int[100];
```

根据定义，数组都是引用类型，所以int数组当然是引用类型（即`reference.GetType().IsValueType`为false）。

而int数组的元素都是int，根据定义，int是值类型（即`reference[i].GetType().IsValueType`为true）。那么引用类型数组中的值类型元素究竟位于栈还是堆？

如果用WinDbg去看`reference[i]`在内存中的具体位置，就会发现它们并不在栈上，而是在托管堆上。

实际上，对于数组：

```
TestType[] testTypes = new TestType[100];
```

如果TestType是值类型，则会一次在托管堆上为100个值类型的元素分配存储空间，并自动初始化这100个元素，将这100个元素存储到这块内存里。

如果TestType是引用类型，则会先在托管堆为testTypes分配一次空间，并且这时不会自动初始化任何元素（即`testTypes[i]`均为null）。等到以后有代码初始化某个元素的时候，这个引用类型元素的存储空间才会被分配在托管堆上。

4.2 类型嵌套

```
public class MyClass
{
    private int _value1;
    1 个引用
    public MyClass()
    {
        _value1 = 0;
    }
    1 个引用
    public void Method()
    {
        int _value2 = 0;
    }
}
2 个引用
public struct MyStruct
{
    private object _object1;
    1 个引用
    public void Method()
    {
        _object1 = new object();
        object _object2 = new object();
    }
}
0 个引用
class Program
{
    0 个引用
    static void Main(string[] args)
    {
        MyClass mc = new MyClass();
        mc.Method();
        MyStruct ms = new MyStruct();
        ms.Method();
    }
}
```

引用类型部署在托管堆上；

值类型总是分配在它声明的地方：作为字段时，跟随其所属的变量（实例）存储；作为局部变量时，存储在栈上。

从上下文看，**mc**是一个局部变量，所以部署在托管堆上，并被栈上的一个引用所持有；

值类型字段 **value1**属于引用类型实例**mc**的一部分，所以跟随引用类型实例**mc**部署在托管堆上（有点类似于数组的情形）；

value2是值类型局部变量，所以部署在栈上。

而对于值类型实例，即**MyStruct**：

根据上下文，值类型实例**ms**本身是一个局部变量而不是字段，所以位于栈上；

其引用类型字段 **object1**不存在跟随的问题，必然部署在托管堆上，并被一个引用所持有（该引用是**ms**的一部分，位于栈）；

其引用类型局部变量 **object2**显然部署在托管堆上，并被一个位于栈的引用所持有。

所以，简单地说“值类型存储在栈上，引用类型存储在托管堆上”是不对的。必须具体情况具体分析

在C#中，我们用struct/class来声明一个类型为值类型/引用类型。考虑下面的例子：

```
SomeType[] oneTypes = new SomeType[100];
```

如果SomeType是值类型，则只需要一次分配，大小为SomeType的100倍。而如果SomeType是引用类型，刚开始需要100次分配，分配后数组的各元素值为null，然后再初始化100个元素，结果总共需要进行101次分配。这将消耗更多的时间，造成更多的内存碎片。所以，如果类型的职责主要是存储数据，值类型比较合适。

一般来说，值类型（不支持多态）适合存储供C#应用程序操作的数据，而引用类型（支持多态）应该用于定义应用程序的行为。通常我们创建的引用类型总是多于值类型。如果满足下面情况，那么我们就应该创建为值类型：

该类型的主要职责用于数据存储。

该类型的共有接口完全由一些数据成员存取属性定义。

该类型永远不可能有子类。

该类型不具有多态行为。

5. 辨明值类型和引用类型的使用场合

在C#中，我们用struct/class来声明一个类型为值类型/引用类型。考虑下面的例子：

```
SomeType[] oneType = new SomeType[100];
```

如果SomeType是值类型，则只需要一次分配，大小为SomeType的100倍。而如果SomeType是引用类型，刚开始需要100次分配，分配后数组的各元素值为null，然后再初始化100个元素，结果总共需要进行101次分配。这将消耗更多的时间，造成更多的内存碎片。所以，如果类型的职责主要是存储数据，值类型比较合适。

一般来说，值类型（不支持多态）适合存储供C#应用程序操作的数据，而引用类型（支持多态）应该用于定义应用程序的行为。通常我们创建的引用类型总是多于值类型。如果满足下面情况，那么我们就应该创建为值类型：

该类型的主要职责用于数据存储。

该类型的共有接口完全由一些数据成员存取属性定义。

该类型永远不可能有子类。

该类型不具有多态行为。

值类型和引用类型的区别（小结）

相同点：

引用类型可以实现接口，值类型当中的结构体也可以实现接口；

引用类型和值类型都继承自System.Object类。

1) 范围方面

C#的值类型包括：结构体（数值类型、bool型、用户定义的结构体），枚举，可空类型。

C#的引用类型包括：数组，用户定义的类、接口、委托，object，字符串。

2) 内存分配方面：

数组的元素不管是引用类型还是值类型，都存储在托管堆上。

引用类型在栈中存储一个引用，其实际的存储位置位于托管堆。简称引用类型部署在托管堆上。而值类型总是分配在它声明的地方：作为字段时，跟随其所属的变量（实例）存储；作为局部变量时，存储在栈上。（栈的内存是自动释放的，堆内存是.NET中会由GC来自动释放）

3) 适用场合

值类型在内存管理方面具有更好的效率，并且不支持多态，适合用做存储数据的载体；引用类型支持多态，适用于定义应用程序的行为。

引用类型可以派生出新的类型，而值类型不能，因为所有的值类型都是密封（seal）的；

引用类型可以包含null值，值类型不能（可空类型功能允许将 null 赋给值类型，如 `int? a = null;`）；

引用类型变量的赋值只复制对对象的引用，而不复制对象本身。而将一个值类型变量赋给另一个值类型变量时，将复制包含的值。

值得注意的是，引用类型和值类型都继承自System.Object类。不同的是，几乎所有的引用类型都直接从System.Object继承，而值类型则继承其子类，即直接继承System.ValueType。即System.ValueType本身是一个类类型，而不是值类型。其关键在于ValueType重写了Equals()方法，从而对值类型按照实例的值来比较，而不是引用地址来比较。

内容参考来自文章：

[值类型和引用类型，栈和堆的含义](#)

[通用类型系统概述](#)

[C#详解值类型和引用类型区别](#)

分类: C#

好文要顶

关注我

收藏该文



进修的stone

关注 - 3

粉丝 - 7



8

0

+加关注

posted @ 2016-09-17 14:16 [进修的stone](#) 阅读(23507) 评论(2) [编辑](#) [收藏](#)

评论列表

[回复](#) [引用](#)

#1楼 2018-03-09 15:58 [cocoyoona](#)

真厉害，一搜博客园 C#资料都是你的。。。

[支持\(0\)](#) [反对\(0\)](#)

[回复](#) [引用](#)

#2楼 2019-02-24 23:26 [没有什么大不了](#)

写的非常的详细，
但有一点，
上文中
在C#中，我们用struct/class来声明一个类型为值类型/引用类型。考虑下面的例子：
和
5. 辨明值类型和引用类型的使用场合

这两小节下的内容一致，不知道是为什么。







[支持\(0\)](#) [反对\(0\)](#)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

发表评论

昵称：

评论内容：



提交评论

[退出](#) [订阅评论](#)

- [Ctrl+Enter快捷键提交]
- 【幸运】99%的人不知道我们有可以帮你薪资翻倍的秘笈！
 - 【推荐】超50万C++/C#源码: 大型实时仿真组态图形源码
 - 【推荐】百度云“猪”你开年行大运，红包疯狂拿
 - 【推荐】55K刚面完Java架构师岗，这些技术你必须掌握

相关博文：

- [值类型和引用类型](#)
- [C# 值类型和引用类型](#)
- [C# 值类型和引用类型、堆和栈值类型和引用类型](#)
- [C# 值类型和引用类型](#)
- [c#——值类型和引用类型](#)

最新新闻：

- [基石资本张维：为何没有任何一家造车新势力值得投资](#)
- [苹果发布新款iMac 首配八核处理器表现提升2.4倍](#)
- [小米去年营收低于格力 雷军输掉与董明珠赌局](#)
- [携程最新股权曝光：梁建章持股增至2.1% 百度降至19%](#)
- [美团的亏损，为什么停不下来？](#)
- » [更多新闻...](#)