IL执行详解学习总结

> 1753837
>
> 陈柄畅

什么是IL?

IL是.NET框架中中间语言（**Intermediate Language**）的缩写。使用.NET框架提供的编译器可以直接将源程序编译为.exe或.dll文件，但此时编译出来的程序代码并不是CPU能直接执行的机器代码，而是 IL 的代码。

查看IL代码

1. 使用csc命令编译程序。

```
csc /out:Program.exe /t:exe /r:mscorlib.dll Program.cs
```

2. 使用反编译程序 ildasm.exe 查看IL代码。

基本概念

1. Managed Heap: 托管堆，存储引用类型的值，由 GC 在运行时自动管理，整个进程共享一个
2. Evaluation Stack: 线程栈，每个线程都有自己专属的，由 .NET CLR 在运行时自动管理
3. Call Stack: 函数调用栈，每个线程都有自己专属的，由 .NET CLR 在运行时自动管理，其中，每一

项是一个 Record Frame，为一个局部变量表

常见IL指令

1. .maxstack: Evaluation Stack 可容纳数据项的最大个数
2. .locals init：定义变量并存入Call Stack中
3. nop: 没有任何操作
4. ldstr.: 把字符串加压入Evaluation Stack中
5. stloc.: 把 Evaluation Stack 中的值弹出赋值到 Call Stack 中的 Record Frame 中
6. ldloc.: 把 Call Stack 中的 Record Frame 中指定位置的值复制存入 Evaluation Stack中
7. call: 调用指定的方法
8. ret: 标记返回
9. dup: 复制栈顶值
10. ldc.i4.x: 将数字 x 放入Evaluation Stack中

IL代码阅读

简单加法

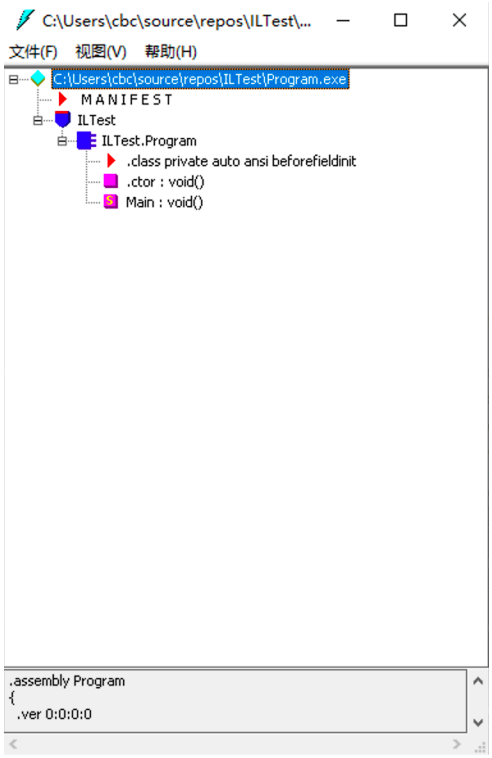
在本示例中，简单地将三个 int 类型的变量相加。

C#程序

```
using System;

namespace ILTest
{
    class Program
    {
        static void Main()
        {
            int i = 1;
            int j = 2;
            int k = 3;
            int answer = i + j + k;
            Console.WriteLine("i+j+k=" + answer);
        }
    }
}
```

ildasm运行截图



IL代码

```
.method private hidebysig static void  Main() cil managed
{
  .entrypoint
  // 代码大小       37 (0x25)
  .maxstack  2  // Evaluation Stack 最大容纳两项
  .locals init (int32 V_0,
           int32 V_1,
           int32 V_2,
           int32 V_3) // 初始化方法，并且将三个局部变量放入 Record Frame
  IL_0000:  nop        // 无操作
  IL_0001:  ldc.i4.1   // 将 1 放入 Evaluation Stack 中
  IL_0002:  stloc.0    // 将 1 弹出，并将其复制到 Call Stack 中 Record Frame 的 V_0
位置
  IL_0003:  ldc.i4.2   // 将 2 放入 Evaluation Stack 中
  IL_0004:  stloc.1    // 将 2 弹出，并将其复制到 Call Stack 中 Record Frame 的 V_1
位置
  IL_0005:  ldc.i4.3   // 将 3 放入 Evaluation Stack 中
  IL_0006:  stloc.2    // 将 3 弹出，并将其复制到 Call Stack 中 Record Frame 的 V_1
位置
  IL_0007:  ldloc.0    // 将 V_0 取出，并放入 Evaluation Stack 中
  IL_0008:  ldloc.1    // 将 V_1 取出，并放入 Evaluation Stack 中
  IL_0009:  add        // 相加
  IL_000a:  ldloc.2    // 将 V_2取出，并放入 Evaluation Stack 中
  IL_000b:  add        // 相加
  IL_000c:  stloc.3    // 将相加的结果弹出，并将其复制到 Call Stack 中 Record Frame
的 V_3 位置
```

```
  IL_000d:  ldstr      "i+j+k="    // 将字符串"i+j+k="加载到栈中
  IL_0012:  ldloca.s   V_3         // 将变量 V_3 放入 Evaluation Stack 中
  IL_0014:  call       instance string [mscorlib]System.Int32::ToString()   //
调用 Int32 的 ToString 函数，将栈顶变量 V_3 转为字符串
  IL_0019:  call       string [mscorlib]System.String::Concat(string,
                                                              string) // 调用
String 的 Concat 函数，拼接字符串
  IL_001e:  call       void [mscorlib]System.Console::WriteLine(string) // 调用
WriteLine 函数
  IL_0023:  nop    // 无操作
  IL_0024:  ret    // 返回
} // end of method Program::Main
```

可以看到，在程序开始时，将 Evaluation Stack 最大容纳设为了两项，而在程序执行过程中，的确两项就已经足够，在命令 IL_000d - IL_0014 中，栈中同时保存 "i+j+k" 和 V_3 两个变量，其余时间栈中都为一项或者没有变量。

Sum函数

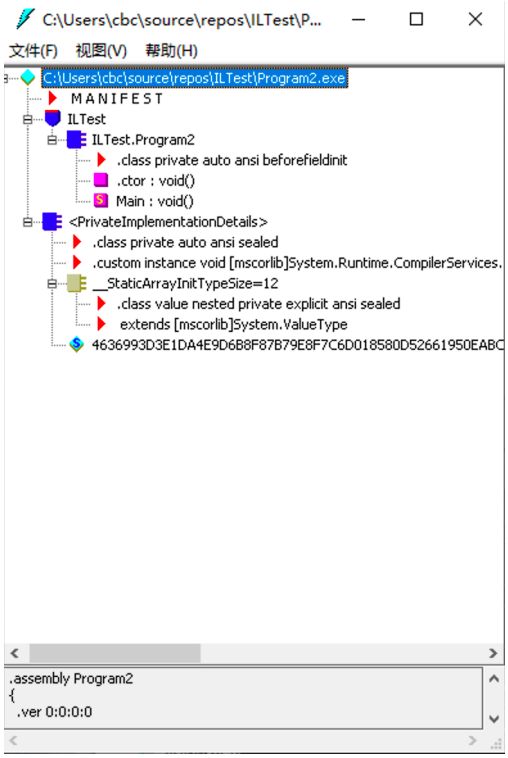
除了简单的加法外，C#还提供了对 array 进行相加的 sum 函数。本示例对 sum 函数的 IL 代码进行解读，了解C#数组的创建等操作的IL代码。

C#程序

```
using System;
using System.Linq;

namespace ILTest
{
    class Program2
    {
        static void Main()
        {
            int[] array = { 1, 2, 3 };
            int arraySum = array.Sum();
            Console.WriteLine("array sum="+arraySum);
        }
    }
}
```

ildasm运行截图



IL代码

```
.method private hidebysig static void  Main() cil managed
{
  .entrypoint
  // 代码大小       50 (0x32)
  .maxstack  3  // Evaluation Stack 最大容纳三项
  .locals init (int32[] V_0,
           int32 V_1) // 初始化方法，并且将两个局部变量放入 Record Frame
  IL_0000:  nop       // 无操作
  IL_0001:  ldc.i4.3  // 将 3 放入 Evaluation Stack 中，为创建数组的长度
  IL_0002:  newarr     [mscorlib]System.Int32   // 创建数组，将一个指向一个在堆中的
数组的引用压入 Evaluation Stack
  IL_0007:  dup   // 复制栈顶的变量
  IL_0008:  ldtoken    field valuetype
'<PrivateImplementationDetails>'/'__StaticArrayInitTypeSize=12'
'<PrivateImplementationDetails>'::'4636993D3E1DA4E9D6B8F87B79E8F7C6D018580D526
61950EABC3845C5897A4D'    // 将数组的元数据token转为运行时表达
  IL_000d:  call       void
[mscorlib]System.Runtime.CompilerServices.RuntimeHelpers::InitializeArray(clas
s [mscorlib]System.Array,

                      valuetype [mscorlib]System.RuntimeFieldHandle)    // 初
始化数组
  IL_0012:  stloc.0   // 将数组引用弹出，并将其复制到 Call Stack 中 Record Frame 的
V_0 位置
  IL_0013:  ldloc.0   // 将 V_0 取出，并放入 Evaluation Stack 中
  IL_0014:  call       int32    [System.Core]System.Linq.Enumerable::Sum(class
[mscorlib]System.Collections.Generic.IEnumerable`1<int32>)   // 调用sum函数
```

```
  IL_0019:  stloc.1   // 将sum函数的结果弹出，并将其复制到 Call Stack 中 Record
Frame 的 V_1 位置
  IL_001a:  ldstr      "array sum="   // 将字符串"array sum="加载到栈中
  IL_001f:  ldloca.s   V_1    // 将 V_0 取出，放入栈中
  IL_0021:  call       instance string  [mscorlib]System.Int32::ToString()
// 将 V_0 转为字符串
  IL_0026:  call       string [mscorlib]System.String::Concat(string,
                                                              string)   // 调用
String 的 Concat 函数，拼接字符串
  IL_002b:  call       void [mscorlib]System.Console::WriteLine(string)   // 调
用 WriteLine 函数
  IL_0030:  nop   // 无操作
  IL_0031:  ret   // 返回
} // end of method Program2::Main
```

通过 IL 代码我们可以看到，整数类型的数组通过命令 newArr 在堆上创建，而栈中保存的是引用对象。