عنوان: **بررسی مقدماتی مراحل کامپایل یک قطعه کد سی‌شارپ و آشنایی با OpCodes**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۳ ۱۳:۸
آدرس: www.dotnettips.info
گروه‌ها: C#, CIL, CLR, IL, MSIL, Reflection

کامپایلر سی‌شارپ چگونه عمل می‌کند؟

کار یک کامپایلر ترجمه قطعه‌ای از اطلاعات به چیز دیگری است. کامپایلر سی‌شارپ، machine code معادل دستورات دات نتی را تهیه نمی‌کند. Machine code، کدی است که مستقیما بر روی CPU قابل اجرا است. در دات نت این مرحله به CLR یا Common language runtime واگذار شده است تا کار اجرای نهایی کدهای تهیه شده توسط کامپایلر سی‌شارپ را انجام دهد.
بنابراین زمانیکه در VS.NET سعی در اجرای یک قطعه کد می‌نمائیم، مراحل ذیل رخ می‌دهند:
الف) فایل‌های سی‌شارپ پروژه، توسط کامپایلر بارگذاری می‌شوند.
ب) کامپایلر کدهای این فایل‌ها را پردازش می‌کند.
ج) سپس چیزی را به نام MSIL تولید می‌کند.
د) در ادامه فایل خروجی نهایی، با افزودن PE Headers تولید می‌شود. توسط PE headers مشخص می‌شود که فایل تولیدی نهایی آیا اجرایی است، یا یک DLL می‌باشد و امثال آن.
ه) و در آخر، فایل اجرایی تولیدی توسط CLR بارگذاری و اجرا می‌شود.

MSIL چیست؟

MSIL مخفف Microsoft intermediate language است. به آن CIL یا Common intermediate language هم گفته می‌شود و این دقیقا همان کدی است که توسط CLR خوانده و اجرا می‌شود. MSIL یک زبان طراحی شده مبتنی بر پشته‌ها است و بسیار شبیه به سایر زبان‌های اسمبلی موجود می‌باشد.

یک سؤال: آیا قطعه کدهای ذیل، کدهای IL یکسانی را تولید می‌کنند؟

```
namespace FastReflectionTests
{
    public class Test
    {
        public void Method1()
        {
            var x = 10;
            var y = 20;
            if (x == 10)
            {
                if (y == 20)
                {

                }
            }
        }

        public void Method2()
        {
            var x = 10;
            var y = 20;
            if (x == 10 && y == 20)
            {

            }
        }
    }
}
```

برای یافتن کدهای MSIL یا IL یک برنامه کامپایل شده می‌توان از ابزارهایی مانند Reflector یا ILSpy استفاده کرد. برای نمونه اگر از برنامه ILSpy استفاده کنیم چنین خروجی IL معادلی را می‌توان مشاهده کرد:

```
.class public auto ansi beforefieldinit FastReflectionTests.Test
extends [mscorlib]System.Object
{
// Methods
.method public hidebysig
instance void Method1 () cil managed
{
// Method begins at RVA 0x3bd0
// Code size 17 (0x11)
.maxstack 2
.locals init (
[0] int32 x,
[1] int32 y
)

IL_0000: ldc.i4.s 10
IL_0002: stloc.0
IL_0003: ldc.i4.s 20
IL_0005: stloc.1
IL_0006: ldloc.0
IL_0007: ldc.i4.s 10
IL_0009: bne.un.s IL_0010

IL_000b: ldloc.1
IL_000c: ldc.i4.s 20
IL_000e: pop
IL_000f: pop

IL_0010: ret
} // end of method Test::Method1

.method public hidebysig
instance void Method2 () cil managed
{
// Method begins at RVA 0x3bf0
// Code size 17 (0x11)
.maxstack 2
.locals init (
[0] int32 x,
[1] int32 y
)

IL_0000: ldc.i4.s 10
IL_0002: stloc.0
IL_0003: ldc.i4.s 20
IL_0005: stloc.1
IL_0006: ldloc.0
IL_0007: ldc.i4.s 10
IL_0009: bne.un.s IL_0010

IL_000b: ldloc.1
IL_000c: ldc.i4.s 20
IL_000e: pop
IL_000f: pop

IL_0010: ret
} // end of method Test::Method2

.method public hidebysig specialname rtspecialname
instance void .ctor () cil managed
{
// Method begins at RVA 0x3c0d
// Code size 7 (0x7)
.maxstack 8

IL_0000: ldarg.0
IL_0001: call instance void [mscorlib]System.Object::.ctor()
IL_0006: ret
} // end of method Test::.ctor

} // end of class FastReflectionTests.Test
```

همانطور که مشاهده می‌کنید، کدهای IL با یک برچسب شروع می‌شوند مانند IL_0000. پس از آن OpCodes یا Operation codes قرار دارند. برای مثال ldc کار load constant را انجام می‌دهد. به این ترتیب مقدار ثابت 10 بارگذاری شده و بر روی پشته ارزیابی قرار داده می‌شود و نهایتا در سمت راست، مقادیر را ملاحظه می‌کنید؛ برای مثال مقادیری مانند 10 و 20.

این کدها در حالت کامپایل Release تهیه شده‌اند و در این حالت، کامپایلر یک سری بهینه سازی‌هایی را جهت بهبود سرعت و

کاهش تعداد OpCodes مورد نیاز برای اجرا برنامه، اعمال می‌کند.

بررسی OpCodes مقدماتی

الف) OpCodes ریاضی
مانند Add، Sub، Mul و Div

ب) OpCodes کنترل جریان برنامه
مانند Jmp، Beq، Bge، Ble، Bne، Call و Ret
برای پرش به یک برچسب، بررسی تساوی و بزرگتر یا کوچک بودن، فراخوانی متدها و بازگشت دادن مقادیر

ج) OpCodes مدیریت آرگومان‌ها
مانند Ldarg_0، Ldarg تا Ldc_I4 ، Ldarg_3 و Ldc_I4_1 تا Ldc_I4_8
برای بارگذاری آرگومان‌ها و همچنین بارگذاری مقادیر قرار گرفته شده بر روی پشته ارزیابی.

برای توضیحات بهتر این موارد می‌توان کدهای IL فوق را بررسی کرد:

```
IL_0000: ldc.i4.s 10
IL_0002: stloc.0
IL_0003: ldc.i4.s 20
IL_0005: stloc.1
IL_0006: ldloc.0
IL_0007: ldc.i4.s 10
IL_0009: bne.un.s IL_0010
IL_000b: ldloc.1
IL_000c: ldc.i4.s 20
IL_000e: pop
IL_000f: pop
```

در اینجا تعدادی مقدار بر روی پشته ارزیابی بارگذاری می‌شوند. تساوی آن‌ها بررسی شده و نهایتا متد خاتمه می‌یابد.

Stack چیست و MSIL چگونه عمل می‌کنید؟

Stack یکی از انواع بسیار متداول ساختار داده‌ها است و اگر بخواهیم خارج از دنیای رایانه‌ها مثالی را برای آن ارائه دهیم می‌توان به تعدادی برگه کاغذ که بر روی یکدیگر قرار گرفته‌اند، اشاره کرد. زمانیکه نیاز باشد تا برگه‌ای از این پشته برداشته شود، باید از بالاترین سطح آن شروع کرد که به آن LIFO یا Last in First out نیز گفته می‌شود. چیزی که آخر از همه بر روی پشته قرار می‌گیرد، در ابتدا برداشته و خارج خواهد شد.
در دات نت، برای قرار دادن اطلاعات بر روی Stack از متد Push و برای بازیابی از متد Pop استفاده می‌شود. استفاده از متد Pop، سبب خذف آن شیء از پشته نیز می‌گردد.
MSIL نیز یک Stack based language است. MSIL برای مدیریت یک سری از موارد از Stack استفاده می‌کند؛ مانند: پارامترهای متدها، مقادیر بازگشتی و انجام محاسبات در متدها. OpCodes کار قرار دادن و بازیابی مقادیر را از Stack به عهده دارند. به تمام این‌ها در MSIL، پشته ارزیابی یا Evaluation stack نیز می‌گویند.

یک مثال: فرض کنید می‌خواهید جمع 5+10 را توسط MSIL شبیه سازی کنیم.
الف) مقدار 5 بر روی پشته ارزیابی قرار داده می‌شود.
ب) مقدار 10 بر روی پشته ارزیابی قرار داده می‌شود. این مورد سبب می‌شود که 5 یک سطح به عقب رانده شود. به این ترتیب اکنون 10 بر روی پشته است و پس از آن 5 قرار خواهد داشت.
ج) سپس OpCode ایی مساوی Add فراخوانی می‌شود.
د) این OpCode سبب می‌شود تا دو مقدار موجود در پشته Pop شوند.
ه) سپس Add، حاصل عملیات را مجددا بر روی پشته قرار می‌دهد.

یک استثناء

در MSIL برای مدیریت متغیرهای محلی تعریف شده در سطح یک تابع، از Stack استفاده نمی‌شود. این مورد شبیه سایر زبان‌های اسمبلی است که در آن‌ها می‌توان مقادیر را در برچسب‌ها یا رجیسترهای خاصی نیز ذخیره کرد.

نظرات خوانندگان

نویسنده: میثم هوشمند
تاریخ: ۱۳۹۲/۰۵/۱۴ ۰:۳۱

سلام جناب نصیری
ممنون. این تیپ مقالات خیلی جذابند و البته عمق خوبی به بینش برنامه نویس می‌دهد .
یک سوال
یادم هست در فروم برنامه نویس چشمم به مطلب خورده که نوشته بود امکان دارد که کدهای دات نت تبدیل به کدهای ماشین کرد که دیگر نیازی به نصب دات نت فریم ورک بر روی سیستم مقصد نباشد
یعنی میتوان تمام نیازمندیهای برنامه را از دل فریم ورک بیرون کشید و به برنامه اضافه کرد و در نهایت یک فایل اجرایی قبال اجرا بدون نیاز به فریم

ممکنه توضیح بدهید در این خصوص؟
ممنون و متشکرم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۴ ۰:۵۹

یک نمونه از این پروژه‌ها، پروژه `Code Refractor` است. خلاصه کاری که انجام می‌دهد شامل مراحل زیر است:
- اسمبلی دات نتی را می‌خواند و `bytecodes/operations` آن‌را استخراج می‌کند.
- پس از آن، نتیجه را تبدیل به یک کد میانی خاص خودش می‌کند.
- این کد میانی خاص خودش را به `C++` ترجمه می‌کند.
- نهایتا از یک کامپایلر `C++` برای تولید فایل اجرایی نهایی استفاده خواهد کرد.
اطلاعات بیشتر

نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۲/۰۶/۰۲ ۱۲:۹

از اینکه این دوره را برای کسانی که در ۳۰ روز گذشته پستی نداشتند آزاد کردید تشکر می‌کنم.