سفری به اعماق کد و معماری سیستم

نام استاد	نام درس
فرشته طلایی	طراحی زبان های برنامه سازی

نسخه	تدوین‌گر	شرح	تاریخ
1.0	فراز جهان منش	از دات‌نت تا اسمبلی: سفری به اعماق کد و معماری سیستم	1403-09-19 2024-12-09

- SampleSource
- Linkedin
- GitHub

1. نحوه کامپایل شدن و توانایی زبان های خانواده دات نت برای ارتباط با یکدیگر :

: ترکیب و تعامل زبان‌های دات نت و کامپایل مشترک

پلتفرم **دات نت (.NET)** یک چارچوب چند زبانه است که به برنامه نویسان اجازه می‌دهد تا برنامه هایی را با استفاده از زبان‌های مختلفی مانند **C#, VB.NET, F#,** و حتی زبان‌های ثالثی مانند **Python (IronPython)** و **JavaScript (JScript.NET)** بنویسند . این قابلیت به دلیل معماری منحصر به فرد **دات نت** فراهم شده است که در ادامه با جزئیات کامل توضیح داده خواهد شد.

1. معماری دات نت و نقش CLR :

معماری **دات نت** حول **Common Language Runtime (CLR)** طراحی شده است که وظیفه ی اجرای کدهای برنامه نویسی را بر عهده دارد . زبان‌های **دات نت** به جای کامپایل مستقیم به کد ماشین ، ابتدا به یک زبان میانی به نام **Microsoft Intermediate Language (MSIL)** یا به‌اختصار **IL** کامپایل می‌شوند.

: گام‌های کامپایل

1. **کامپایل زبان به IL** : هر زبان **دات نت** توسط کامپایلر خاص خود به IL تبدیل می‌شود IL یک زبان مستقل از زبان مبدا است که تمام اطلاعات مورد نیاز برای اجرای برنامه را در خود دارد.
2. **ذخیره در فایل اسمبلی** IL : تولید شده در فایل‌های **dll.** یا **exe.** همراه با متا داده هایی که ساختار و مشخصات کد را توضیح می‌دهند، ذخیره می‌شود.
3. **اجرای IL توسط CLR** : هنگام اجرا، **CLR** این IL را با استفاده از یک کامپایلر **JIT (Just-In-Time)** به کد ماشین تبدیل کرده و اجرا می‌کند.

4. **دلیل امکان تعامل :** از آنجا که تمام زبان‌های دات‌نت به **IL** یکسان کامپایل می‌شوند و توسط **CLR** اجرا می‌شوند، برنامه‌ها می‌توانند به‌راحتی از کد نوشته‌شده در سایر زبان‌ها استفاده کنند.

.2 مشترکات زبان‌های دات‌نت CTS & CLS :

برای هماهنگی میان زبان‌ها، **دات نت** دو مفهوم کلیدی معرفی کرده است:

2.1. Common Type System (CTS):

CTS سیستمی است که تعریف می‌کند تمام زبان‌های دات نت باید از یک مجموعه انواع داده‌ای مشترک استفاده کنند. به عنوان مثال:

- نوع **int** در **C#** معادل **Integer** در **VB.NET** است، و هر دو به **System.Int32** نگاشت می‌شوند.
- این سیستم تضمین می‌کند که انواع داده ای در تمام زبان‌ها یکسان عمل کنند.

2.2. Common Language Specification (CLS):

CLS مجموعه‌ای از قوانین است که باید توسط تمام زبان‌های دات نت رعایت شود تا قابلیت تعامل با یکدیگر داشته باشند. این قوانین شامل موارد زیر است:

- استفاده از انواع داده‌ای سازگار با **CTS**
- پرهیز از ویژگی‌هایی که مختص یک زبان هستند (مانند سربارگذاری اپراتورها که در برخی زبان‌ها پشتیبانی نمی‌شود).

.3 تعامل زبان‌های مختلف در پروژه‌های دات نت

به لطف **CLR**، **CLS**، و**CTS** ، زبان‌های **دات نت** می‌توانند با یکدیگر تعامل داشته باشند. این تعامل در عمل به صورت زیر صورت می‌گیرد:

3.1. ارجاع به اسمبلی‌ها :

یک پروژه می‌تواند به اسمبلی‌هایی که در زبان دیگری نوشته شده‌اند ارجاع دهد. به عنوان مثال:

- یک کلاس در **VB.NET** می‌تواند به متدی که در **C#** نوشته شده است دسترسی داشته باشد.
- این امکان از آنجا فراهم می‌شود که متاداده در فایل‌های اسمبلی (**DLL/EXE**) اطلاعات لازم برای استفاده از کلاس‌ها و متدها را ذخیره می‌کند.

3.2.ایجاد و استفاده از کد ترکیبی :

شما می‌توانید پروژه‌هایی ایجاد کنید که کد آن‌ها در زبان‌های مختلف نوشته شده است:

- **در یک پروژه** : با افزودن فایل‌هایی به زبان‌های مختلف.
- **در چند پروژه** : با ارجاع متقابل بین پروژه‌های جداگانه.

3.3. مشترکات کارکردی :

تمام زبان‌ها به یک مجموعه از کتابخانه‌های استاندارد داتنت (BCL یا Base Class Library) دسترسی دارند، به عنوان مثال:

- کار با فایل‌ها(**System.IO**)
- کار با داده‌ها(**System.Data**)
- پردازش رشته‌ها(**System.Text**)

4. مثال: تعامل VB.NET و C#

فرض کنید یک کلاس در C# داریم:

کد در C#:

```
public class Calculator
{
    public int Add(int a, int b)
    {
        return a + b;
    }
}
```

اکنون می‌توانیم این کلاس را در VB.NET استفاده کنیم:

کد در VB.NET:

```
Imports MyCSharpProjectNamespace

Module Program
    Sub Main()
        Dim calc As New Calculator()
        Dim result As Integer = calc.Add(10, 20)
        Console.WriteLine($"Result: {result}")
    End Sub
End Module
```

این مثال نشان می‌دهد که کد نوشته شده در یک زبان به راحتی می تواند توسط زبان دیگری مصرف شود.

5.مزایا و چالش‌ها

مزایا:

1. **افزایش بهره‌وری تیمی** : تیم‌ها می‌توانند از زبان‌های مختلفی که با آن‌ها راحت ترند، استفاده کنند.
2. **انعطاف‌پذیری** : امکان استفاده از ویژگی‌های منحصر به فرد زبان‌ها برای بخش‌های مختلف پروژه وجود دارد.
3. **استفاده مجدد از کد** : اسمبلی‌های تولید شده توسط یک زبان می‌توانند توسط سایر زبان‌ها استفاده شوند.

چالش‌ها:

1. **پیروی از : CLS** اگر کدی از CLS تبعیت نکند، ممکن است در زبان‌های دیگر سازگاری نداشته باشد.
2. **خوانایی و نگهداری** : ترکیب زبان‌ها در یک پروژه ممکن است نگهداری کد را دشوار کند.
3. **محدودیت در ویژگی‌ها** : ممکن است ویژگی‌هایی که مختص یک زبان هستند در زبان دیگر پشتیبانی نشوند(مانند Tuple در #F و #C)

---

6. نتیجه گیری

دات نت یک پلتفرم واقعاً چندزبانه است که به توسعه دهندگان امکان می‌دهد به زبان‌های مختلفی برنامه بنویسند و از آن‌ها در یک پروژه مشترک استفاده کنند. این قابلیت به لطف ویژگی‌هایی مانند CLR ، CTS، CLS، و IL امکان‌پذیر شده است. با رعایت قوانین CLS و استفاده از انواع داده‌ای سازگار با CTS ، زبان‌های دات نت می‌توانند به صورت یکپارچه با یکدیگر کار کنند، و این ویژگی یکی از نقاط قوت اصلی این پلتفرم محسوب می‌شود.

2. دات نت چیست شامل چه زبان هایی میشود چرایی وجود دات نت اهداف ان و اجزا:

دات نت (.NET) چیست؟

دات نت (.NET) یک پلتفرم نرم افزاری چند منظوره است که توسط شرکت **مایکروسافت** ایجاد شده و برای توسعه و اجرای برنامه های کاربردی در سیستم های مختلف طراحی شده است. این پلتفرم شامل مجموعه‌ای از ابزارها، کتابخانه‌ها، و چارچوب‌ها است که برنامه نویسان را قادر می‌سازد تا برنامه هایی را در انواع زبان‌های برنامه نویسی توسعه دهند.

معنای".NET"

واژه "NET" به معنای "شبکه (Network) "است و مایکروسافت از این نام استفاده کرده است تا هدف اصلی این پلتفرم، یعنی تسهیل ایجاد برنامه‌های کاربردی مبتنی بر شبکه و اینترنت، را نشان دهد. دات نت در ابتدا برای یکپارچه سازی فناوری‌های مایکروسافت و ساده تر کردن توسعه برنامه ها در محیط وب، دسکتاپ و موبایل ایجاد شد.

چرا داتنت به وجود آمد؟

1. نیاز به یک چارچوب جامع و مدرن

در دهه 1990، توسعه نرم‌افزارها با زبان‌های قدیمی‌تر مانند C++ و Visual Basic بسیار پیچیده و وقت گیر بود. مایکروسافت برای رفع این مشکلات:

- **ابزاری قدرتمندتر** برای توسعه نرم‌افزار ایجاد کرد.
- نیاز به یک چارچوبی که در عین انعطاف پذیری، چند منظوره باشد و توسعه دهندگان بتوانند از زبان های مختلف استفاده کنند، احساس شد.

2. ساده سازی فرایند توسعه

قبل از دات نت، برنامه نویسان مجبور بودند برای هر پلتفرم یا سیستم‌عاملی، کدهای متفاوتی بنویسند. دات نت ایجاد شد تا:

- برنامه نویسان بتوانند کدی بنویسند که روی سیستم عامل های مختلف (مانند ویندوز، لینوکس، و مک) اجرا شود.
- ابزارهایی مانند Visual Studio ، فرایند نوشتن، اشکال زدایی و اجرای برنامه‌ها را تسهیل کنند.

3. پاسخ به نیاز به یکپارچگی میان زبان‌ها

مایکروسافت با معرفی **CLR (Common Language Runtime)** و **CTS (Common Type System)**، دات نت را به عنوان یک پلتفرم چند زبانه معرفی کرد که امکان تعامل بین زبان‌های مختلف مانند C# ,VB.NET ، و **F#** را فراهم می کند.

4. گسترش توسعه برنامه‌های وب

در اوایل دهه 2000، توسعه وب در حال رشد سریع بود. مایکروسافت دات‌نت را طراحی کرد تا ابزارهایی مانند ASP.NET را برای توسعه سریع و کارآمد برنامه‌های وب ارائه دهد.

5. رقابت با سایر فناوری‌ها

دات نت به عنوان پاسخی به فناوری‌های مشابه مانند جاوا (Java) از شرکت سان مایکرو سیستمز ایجاد شد. جاوا در آن زمان یکی از محبوب ترین پلتفرم‌ها برای توسعه برنامه های چند پلتفرمی بود.

اهداف اصلی دات نت

1. **چند منظوره بودن** : دات نت برای توسعه برنامه های دسکتاپ، وب، موبایل، بازی، اینترنت اشیا (IoT) ، و سرویس‌های ابری طراحی شده است.
2. **چند پلتفرمی بودن** : با نسخه‌های جدید دات نت مانند .NET Core و .NET 5/6/7 ، امکان اجرای برنامه‌ها روی سیستم عامل‌های مختلف فراهم شده است.
3. **ساده‌سازی توسعه** : با ارائه ابزارهایی مثل Visual Studio و کتابخانه‌های گسترده(BCL - Base Class Library) ، توسعه برنامه‌ها را ساده‌تر و سریع‌تر می‌کند.
4. **پشتیبانی از چندین زبان** : دات نت به توسعه دهندگان اجازه می‌دهد زبان برنامه نویسی مورد علاقه خود را انتخاب کنند.
5. **امنیت و پایداری** : دات نت با ویژگی هایی مانند مدیریت حافظه خودکار (Garbage Collection) و سیستم تایپ قوی، برنامه‌های پایدارتر و امن‌تری ارائه می‌دهد.

اجزای اصلی دات نت

1. **CLR (Common Language Runtime)**: موتور زمان اجرا که وظیفه اجرای کدهای دات نت را بر عهده دارد CLR . کدها را از IL (Intermediate Language) به کد ماشین تبدیل کرده و اجرا می‌کند.
2. **BCL (Base Class Library)**: مجموعه‌ای از کتابخانه‌های استاندارد که قابلیت‌هایی مانند مدیریت فایل، شبکه، رشته‌ها، و غیره را فراهم می‌کند.
3. **C#، VB.NET، F#و زبان‌های دیگر** : زبان‌هایی که برای توسعه برنامه‌های دات‌نت استفاده می‌شوند.
4. **ASP.NET** : چارچوبی برای توسعه برنامه‌های وب و سرویس‌های وب.
5. **ADO.NET** : ابزارهایی برای دسترسی به پایگاه داده و کار با داده‌ها.
6. **Visual Studio و Visual Studio Code** : محیط های توسعه یکپارچه (IDE) برای توسعه برنامه‌های دات نت.
7. **.NET Core و.NET 5/6/7** : نسخه های چند پلتفرمی و بهینه شده دات نت برای توسعه برنامه‌های مدرن.

تحولات دات نت

1. **دات نت فریم ورک :(2002)** اولین نسخه دات نت که تنها روی ویندوز اجرا می‌شد.
2. **.NET Core (2016)** : یک نسخه متن باز و چند پلتفرمی از دات نت.

نتیجه‌گیری

دات نت یک پلتفرم جامع، انعطاف پذیر و چند منظوره است که به توسعه دهندگان اجازه می دهد برنامه هایی پایدار، امن و چند پلتفرمی ایجاد کنند. هدف اصلی از ایجاد دات نت ساده سازی توسعه نرم افزار، افزایش بهره وری، و ارائه ابزارها و چارچوب هایی است که توسعه برنامه ها را سریع‌تر و کارآمدتر می‌کنند. این پلتفرم از زمان معرفی تا امروز تحول بسیاری داشته و اکنون به یکی از محبوب ترین گزینه‌ها برای توسعه نرم افزار در مقیاس های مختلف تبدیل شده است.

3.زبان های خانواده دات نت

خانواده دات نت شامل زبان‌های برنامه نویسی متنوعی است که برای توسعه برنامه‌های مبتنی بر پلتفرم NET. طراحی شده‌اند. این زبان‌ها می‌توانند با یکدیگر تعامل داشته باشند، زیرا همگی از ویژگی‌های مشترک دات نت مانند

CLR (Common Language Runtime) و **CTS (Common Type System)** استفاده می‌کنند. در ادامه به زبان‌های اصلی و مهم این خانواده اشاره می‌کنیم:

.1 سی‌شارپ

- **معرفی :**محبوب ترین زبان در خانواده دات نت و یکی از زبان های اصلی که مایکروسافت برای این پلتفرم طراحی کرده است.
- **ویژگی‌ها:**
  - ساده، مدرن و شیءگرا.
  - قابلیت پشتیبانی از برنامه‌نویسی تابعی و عمومی.(Generic Programming)
  - مناسب برای توسعه وب، دسکتاپ، بازی و برنامه‌های ابری.

.2 ویژوال بیسیک دات‌نت

- **معرفی :** نسخه ای مدرن از زبان ویژوال بیسیک که برای استفاده در دات نت طراحی شده است.
- **ویژگی‌ها:**
  - یادگیری آسان و مناسب برای مبتدیان.
  - قابلیت‌های شیءگرایی و مجتمع شدن با سایر زبان‌های دات نت.
  - اغلب در پروژه‌های دسکتاپ و برنامه های تجاری استفاده می‌شود.

---

3. اف شارپ

- **معرفی :**یک زبان برنامه نویسی تابعی و چندالگویی که برای دات نت طراحی شده است.
- **ویژگی‌ها:**
  - مناسب برای محاسبات ریاضی، تحلیل داده و پردازش هم زمان.
  - ترکیبی از قابلیت های برنامه نویسی شیءگرا و تابعی.
  - قدرتمند در نوشتن کدهای کوتاه، خوانا و قابل اطمینان.

---

4. سی پلاس پلاس

- **معرفی :** یک نسخه از C++ که برای کار با دات نت طراحی شده است.
- **ویژگی‌ها :**
  - امکان ترکیب کدهای دات نت با کدهای اصلی.(Native Code)
  - استفاده در سناریوهایی که نیاز به کارایی بالا یا تعامل با کتابخانه‌های غیرمدیریتی دارند.

---

5. PowerShell

- **معرفی :**زبان اسکریپت نویسی برای مدیریت سیستم و خودکارسازی وظایف.
- **ویژگی‌ها:**
  - مبتنی بر دات نت فریم ورک.
  - قابلیت تعامل با اشیای دات نت و مدیریت سرویس های ویندوز.

---

6. IronPython

- **معرفی :** یک نسخه از Python که برای اجرا در دات نت طراحی شده است.
- **ویژگی‌ها:**
  - امکان استفاده از کتابخانه‌ها و ابزارهای دات نت در برنامه های پایتون.
  - مناسب برای توسعه سریع نمونه‌های اولیه(Prototyping) .

---

7. IronRuby

- **معرفی :**یک نسخه از Ruby که با پلتفرم دات نت سازگار است.
- **ویژگی‌ها:**
  - ادغام ویژگی‌های Ruby با قدرت و امکانات دات نت.
  - مناسب برای اسکریپت نویسی و توسعه سریع برنامه‌ها.

---

8. JScript.NET

- **معرفی :** نسخه‌ای از JavaScript که برای استفاده در دات نت طراحی شده است.
- **ویژگی‌ها:**
  - امکان استفاده از ویژگی‌های شیءگرایی دات نت در کدهای جاوا اسکریپت.
  - کمتر محبوب است و بیشتر در پروژه‌های قدیمی دیده می‌شود.

---

9. Kotlin/Native

- **معرفی :** اگرچه Kotlin به صورت مستقیم بخشی از دات نت نیست، اما ابزارهایی مانند **Kotlin/Native** و **IKVM.NET** امکان اجرای برنامه‌های Kotlin روی دات نت را فراهم می‌کنند.

10. زبان‌های دیگر قابل تعامل

پلتفرم دات نت از طریق ویژگی‌های **CLR** و **CTS** از زبان‌های دیگر نیز پشتیبانی می‌کند که توسط جامعه متن باز یا ابزارهای جانبی توسعه داده شده‌اند:

- **Mercury :** زبانی جدید که جایگزینی مدرن برای VB.NET محسوب می‌شود.
- **Boo :** زبان شیءگرایی که ترکیبی از Python و #C است.
- **Oxygene :** زبانی مبتنی بر پاسکال برای دات نت.

چرا این زبان‌ها با هم کار می‌کنند؟

1. **CLR (Common Language Runtime):**
   تمام زبان‌های دات نت توسط CLR اجرا می‌شوند. این موتور کدهای زبان های مختلف را به یک زبان میانی مشترک به نام **IL (Intermediate Language)** تبدیل می‌کند.
2. **CTS (Common Type System):**
   همه زبان‌های دات نت از یک سیستم نوع (Type System) مشترک استفاده می‌کنند. این باعث می‌شود که کلاس‌ها، ساختارها، و متدهای نوشته شده در یک زبان به راحتی در زبان‌های دیگر استفاده شوند.
3. **CLS (Common Language Specification):**
   مجموعه ای از قوانین و استاندارد ها که تضمین می‌کند کدهای نوشته شده در یک زبان با سایر زبان‌ها سازگار باشند.

4.اسامی زبان های خانواده دات نت

1. **C#**
2. **VB.NET**
3. **F#**
4. **C++/CLI**
5. **PowerShell**
6. **IronPython**
7. **IronRuby**
8. **JScript.NET**
9. **Kotlin/Native**
10. **Mercury**
11. **Boo**
12. **Oxygene**

5.چرا زبان C در این دسته نیست

زبان **C** به طور مستقیم بخشی از خانواده دات نت نیست. دلیل آن این است که زبان C یک زبان **Native** (غیرمدیریتی) است و مستقیماً تحت **CLR (Common Language Runtime)** اجرا نمی‌شود. دات نت بیشتر برای زبان هایی طراحی شده است که بر روی محیط مدیریت شده (Managed Environment) اجرا می شوند، مانند **C#** یا **VB.NET**

با این حال، می‌توان از زبان C در کنار دات نت در سناریوهایی خاص استفاده کرد:

- **از طریق P/Invoke (Platform Invocation) :** برای فراخوانی کتابخانه‌های Native (مانند DLLهای نوشته‌شده با C) از برنامه‌های دات نت.
- **از طریق زبان C++/CLI : که نسخه ای از C++ است و قابلیت تعامل با کدهای داتنت و زبان C را فراهم می‌کند.**

بنابراین، زبان C به صورت مستقیم عضو خانواده دات نت محسوب نمی شود، اما می تواند در تعامل با آن استفاده شود.

6.اولین گام پس از کامپایل شدن کد ما

پس از اینکه IDE (مانند Visual Studio) کد شما را کامپایل می‌کند، اولین گام در فرآیند اجرای کد بستگی به نوع زبان برنامه نویسی و محیط اجرای آن دارد. اگر در مورد برنامه های دات نت صحبت می کنیم، فرآیند به این شکل است:

.1تولید زبان میانی(Intermediate Language - IL)

- **چه اتفاقی می‌افتد؟**
  کد منبع شما (مانند کد C# یا VB.NET) به یک زبان میانی به نام (Intermediate Language) IL کامپایل می‌شود. این زبان میانی مستقل از معماری سخت‌افزار است و توسط کامپایلرهای داتنت (مانند Roslyn برای C#) تولید می‌شود.
- **فایل خروجی:**
  یک فایل اسمبلی تولید می‌شود که معمولاً دارای فرمت‌های زیر است:
  - **DLL (Dynamic Link Library)** : برای کتابخانه ها.
  - **EXE (Executable)** : برای برنامه های اجرایی.

.2 تولید متاداده(Metadata)

- **چه اتفاقی می‌افتد؟**
  متا داده ای که توصیف کننده انواع (Types) ، متدها، کلاس‌ها و سایر اطلاعات کد است به فایل اسمبلی اضافه می‌شود. این متاداده به CLR کمک می‌کند تا در زمان اجرا اطلاعات مورد نیاز برای مدیریت حافظه، امنیت و نوع‌دهی (Typing) را داشته باشد.

.3بارگذاری فایل توسط CLR (در زمان اجرا)

هنگامی که برنامه اجرا می‌شود، **CLR (Common Language Runtime)** وارد عمل می‌شود:

- **Assembly Loader : فایل کامپایل شده (DLL یا EXE) توسط CLR بارگذاری می‌شود.**
- **تأیید ایمنی کد (Code Verification) : بررسی می‌شود که کد از قوانین امنیتی و سازگاری پیروی می‌کند.**

.4کامپایل Just-In-Time (JIT)

- **چه اتفاقی می‌افتد؟**
  IL (زبان میانی) به کد ماشین (Machine Code) تبدیل می‌شود. این کار توسط کامپایلر **JIT (Just-In-Time)** انجام می‌شود و نتیجه آن کدی است که مستقیماً توسط پردازنده اجرا می‌شود.

.5اجرای کد توسط پردازنده

پس از کامپایل توسط JIT ، کد نهایی مستقیماً روی سخت افزار اجرا می‌شود.

فرآیند کلی:

1. کد منبع → **کامپایل بهIL**
2. تولید متاداده → **ساخت فایل اسمبلی**
3. اجرای فایل توسط CLR → **کامپایلJIT**
4. اجرای کد روی پردازنده

این فرآیند در محیط های **Managed**، مانند دات نت، انجام می‌شود و شامل ویژگی هایی مانند مدیریت حافظه، Garbage Collection، و امنیت کد است.

7.دو نمونه کد و ترجمه ان ها به il

در اینجا، دو متد ساده برای جمع دو عدد در زبان‌های C# و VB.NET ارائه شده و سپس ترجمه آنها به زبان میانی (IL) و نحوه انجام ترجمه توضیح داده می‌شود.

کد جمع دو عدد در#C

```
public class Calculator
{
   public int Add(int a, int b)
   {
      return a + b;
   }
}
```

کد جمع دو عدد در VB.NET

```
Public Class Calculator
   Public Function Add(a As Integer, b As Integer) As Integer
      Return a + b
   End Function
End Class
```

ترجمه کد به زبان میانی (IL)

هنگامی که این کدها کامپایل می‌شوند، به زبان میانی **IL (Intermediate Language)** تبدیل می‌شوند. برای مشاهده کد IL ، می‌توانید از ابزارهایی مانند **ILDasm** (Intermediate Language Disassembler) استفاده کنید.

کد IL برای C#:

```
.class public auto ansi beforefieldinit Calculator
      extends [mscorlib]System.Object
{
   .method public hidebysig
         instance int32 Add (
            int32 a,
            int32 b
         ) cil managed
   {
      .maxstack 2
      .locals init (
         [0] int32 V_0
      )
      IL_0000: nop
      IL_0001: ldarg.1      // بارگذاری مقدار 'a' در استک
      IL_0002: ldarg.2      // بارگذاری مقدار 'b' در استک
      IL_0003: add          // جمع مقادیر در استک
      IL_0004: stloc.0      // ذخیره نتیجه در متغیر محلی V_0
      IL_0005: br.s IL_0007
      IL_0007: ldloc.0      // بارگذاری نتیجه از V_0 در استک
      IL_0008: ret          // بازگرداندن مقدار
   }
}
```

کد IL برای VB.NET:

```
.class public auto ansi beforefieldinit Calculator
      extends [mscorlib]System.Object
{
   .method public instance int32 Add (
      int32 a,
      int32 b
   ) cil managed
   {
      .maxstack 2
      .locals init (
         [0] int32 V_0
      )
      IL_0000: ldarg.1     // بارگذاری مقدار 'a' در استک
      IL_0001: ldarg.2     // بارگذاری مقدار 'b' در استک
      IL_0002: add         // جمع مقادیر در استک
      IL_0003: stloc.0     // ذخیره نتیجه در V_0
      IL_0004: ldloc.0     // بارگذاری V_0 در استک
      IL_0005: ret         // بازگرداندن مقدار
   }
}
```

نحوه ترجمه توسط کامپایلر

1. **کد منبع → کامپایل به:IL**
   - **C# و VB.NET توسط کامپایلرهای مخصوص (مانند Roslyn) به IL تبدیل می‌شوند.**
   - IL زبانی مستقل از پلتفرم است که توسط CLR قابل فهم است.
2. **ترجمه دستورات:**

- دستور return a + b; در C# یا Return a + b در VB.NET به دنباله‌ای از دستورات IL ترجمه می‌شود :
  - **ldarg.1 و ldarg.2 : بارگذاری مقادیر ورودی در استک.**
  - **Add : اجرای عملیات جمع روی مقادیر موجود در استک**.
  - **stloc.0: ذخیره نتیجه در متغیر محلی**.
  - **Ret** : بازگرداندن مقدار نهایی.

3. **بهینه‌سازی:**
   - کامپایلر، دستورات غیرضروری (مانند nop) را در مواردی حذف یا بهینه می‌کند.

چرا IL یکسان است؟

در این مثال، ILتولیدشده برای C# و VB.NET تقریباً یکسان است، زیرا هر دو زبان از یک سیستم مشترک (**CTS - Common Type System**) و کامپایلرهای استاندارد استفاده می‌کنند که هدفشان تولید IL یکپارچه است.

مشاهده و اجرای کد:IL

برای مشاهده:IL

1. کد C# یا VB.NET را کامپایل کنید تا فایل DLL یا EXE ایجاد شود.
2. از ابزار **ILDasm** برای باز کردن فایل خروجی استفاده کنید:
3. ildasm YourAssembly.dll
4. ILتولیدشده را بررسی کنید و دستورات مربوط به متد Addرا مشاهده کنید.

8.نگارش به نحوه ای که il های کاملا یکسان داشته باشد

برای اینکه کدهای C# و VB.NET به IL یکسانی ترجمه شوند، باید اطمینان حاصل کنیم که از ساختار و روش‌هایی استفاده کنیم که دقیقاً مشابه عمل می‌کنند. برخی از تفاوت‌های جزئی در نحو زبان ممکن است باعث تفاوت‌های کوچک در IL شوند، بنابراین با استفاده از روش‌های زیر می‌توانیم این هماهنگی را ایجاد کنیم.

کد C#

```
public class Calculator
{
   public int Add(int a, int b)
   {
      return a + b; // عمل جمع و بازگشت نتیجه
   }
}
```

کد VB.NET

```
Public Class Calculator
   Public Function Add(a As Integer, b As Integer) As Integer
      Return a + b ' عمل جمع و بازگشت نتیجه
   End Function
End Class
```

IL یکسان برای هر دو زبان

برای کدهای بالا، IL تولید شده کاملاً یکسان خواهد بود، زیرا هر دو:

1. از ساختار مشابهی برای تعریف کلاس و متد استفاده می‌کنند.
2. از عملگر + برای جمع دو مقدار استفاده می‌کنند.
3. یک متد ساده برای بازگرداندن مقدار نهایی دارند.

IL تولیدشده :

```
.class public auto ansi beforefieldinit Calculator
      extends [mscorlib]System.Object
{
   .method public hidebysig instance int32 Add (
      int32 a,
      int32 b
   ) cil managed
   {
      .maxstack 2              // حداکثر تعداد آیتم‌های موجود در استک
      .locals init (           // تعریف متغیرهای محلی
         [0] int32 V_0
      )
      IL_0000: nop             // هیچ عملی انجام نمی‌دهد (فقط برای تنظیم مکان کد)
      IL_0001: ldarg.1         // مقدار 'a' را به استک می‌فرستد
      IL_0002: ldarg.2         // مقدار 'b' را به استک می‌فرستد
      IL_0003: add             // جمع مقادیر در استک
      IL_0004: stloc.0         // ذخیره نتیجه در متغیر محلی V_0
      IL_0005: ldloc.0         // مقدار V_0 را دوباره به استک می‌فرستد
      IL_0006: ret             // بازگرداندن مقدار از استک
   }
}
```

چرا IL یکسان است؟

- هر دو زبان از **CTS (Common Type System)** و کامپایلرهای استاندارد دات نت برای تولید IL استفاده می‌کنند.
- هیچ تفاوتی در تعریف کلاس یا نحوه استفاده از عملگر + وجود ندارد.
- روش بازگرداندن مقدار (Return) در هر دو زبان به دستورات **IL** یکسانی ترجمه می‌شود.

مشاهده‌IL

1. کد #C یا VB.NET را کامپایل کنید:
   - **C#:**

     bash

     csc Calculator.cs
   - **VB.NET:**

     bash

     vbc Calculator.vb
2. از ابزار **ILDasm** برای مشاهده فایل کامپایل‌شده استفاده کنید:

bash

ildasm Calculator.dll

3. متد Add را باز کنید و بررسی کنید که IL تولیدشده یکسان است.

نکته

اطمینان حاصل کنید که:

- در هر دو زبان از ساختارهای پایه‌ای استفاده کنید.
- از ویژگی‌ها یا قابلیت‌های خاص هر زبان (مانند Optional Parameters در VB.NET یا Expression-Bodied Members در C#) که ممکن است به IL متفاوتی ترجمه شوند، اجتناب کنید.

8.1 متا دیتا چیست و نجوه تولید ان وظایف و اهمیت ان

متاداده چیست؟

در دات‌نت، **متاداده (Metadata)** به اطلاعاتی گفته می‌شود که درباره ساختار کد، انواع داده‌ها، متدها، خصوصیات، رویدادها و سایر المان‌های برنامه ارائه می‌شود. متاداده به همراه کد میانی (IL) در فایل‌های اسمبلی (DLL یا EXE) ذخیره می‌شود و توسط CLR (Common Language Runtime)در زمان اجرا برای بارگذاری، مدیریت و اجرای کد استفاده می‌شود.

وظایف و اهمیت متاداده

1. **توصیف کامل برنامه :**
   - شامل اطلاعات درباره کلاس‌ها، متدها، انواع داده‌ها، و پارامترها.
   - ارائه اطلاعات مانند نام، نوع، دامنه دسترسی، و وابستگی‌ها.
2. **قابلیت بازتاب (Reflection) :**
   - امکان خواندن و تعامل با متاداده در زمان اجرا (با استفاده از کلاس‌های System.Reflection).
3. **عدم نیاز به فایل‌های جداگانه :**
   - متاداده بخشی از فایل اسمبلی است؛ بنابراین نیازی به فایل هدر یا اطلاعات جداگانه (مانند فایل‌های .h در C++) نیست.
4. **پشتیبانی از چندزبانگی :**
   - متاداده به CLR کمک می‌کند تا انواع مختلف زبان‌های دات نت را در یک محیط یکپارچه مدیریت کند.
5. **ابزارهای توسعه و دیباگینگ:**
   - ابزارهایی مانند Visual Studio و ILDasm از متاداده برای نمایش ساختار کد استفاده می‌کنند.

متاداده‌ای که برای متدهای ساده جمع تولید می‌شود

کد C# و VB.NET:

```
public class Calculator
{
   public int Add(int a, int b)
   {
      return a + b;
   }
}
```

```
Public Class Calculator
   Public Function Add(a As Integer, b As Integer) As Integer
      Return a + b
   End Function
End Class
```

متاداده تولید شده :

با استفاده از ابزار **ILDasm** می‌توانید متاداده مربوط به کلاس Calculator و متد Add را مشاهده کنید. متاداده برای هر زبان یکسان است، زیرا به CLR یک زبان مشترک ارائه می‌دهند.

مثال:

```
.class public auto ansi beforefieldinit Calculator
      extends [mscorlib]System.Object
{
   .method public hidebysig instance int32
         Add(int32 a, int32 b) cil managed
   {
      .maxstack 2
      .locals init ([0] int32 V_0)
      IL_0000: nop
      IL_0001: ldarg.1
      IL_0002: ldarg.2
      IL_0003: add
      IL_0004: stloc.0
      IL_0005: ldloc.0
      IL_0006: ret
   }
}
```

جزئیات متاداده برای متد Add

متاداده‌ای که برای این متد تولید می‌شود شامل اطلاعات زیر است:

1. **نام متد:**
   - Add در هر دو زبان یکسان است.
2. امضا (Signature):
   - نوع بازگشتیint32 :
   - پارامترها :
     - int32 a
     - int32 b
3. دسترسی (Access):
   - public: این متد به صورت عمومی در دسترس است.
4. **توضیحات رفتار:**
   - cil managed : نشان می دهد که این متد در محیط مدیریت شده (Managed Environment) اجرا می‌شود و از کد IL استفاده می‌کند.
5. **اطلاعات کلاس:**
   - متاداده کلاس Calculator شامل موارد زیر است:
     - نام کلاس.
     - پایه کلاس.(System.Object)
     - مشخصات دسترسی.(public)
6. **سایر اطلاعات:**
   - تعداد پارامترها.
   - انواع پارامترها.
   - اطلاعات مربوط به حافظه محلی (مانند .(locals

نحوه تولید متاداده توسط کامپایلر

1. **تجزیه و تحلیل کد منبع:**
   - کامپایلر کد شما را اسکن می‌کند تا تمامی کلاس‌ها، متدها، پارامترها، و سایر المان‌ها را شناسایی کند.
2. **ساخت جدول متاداده:**
   - کامپایلر جدولی از اطلاعات مرتبط با کد شما تولید می‌کند. این جدول شامل انواع داده‌ها، متدها، مقادیر بازگشتی، و ویژگی‌های مرتبط است.
3. **ذخیره در اسمبلی:**
   - این متاداده همراه با IL در فایل اسمبلی ذخیره می‌شود.

ابزارها برای مشاهده متاداده

1. **ILDasm (Intermediate Language Disassembler):**
   - می‌توانید متاداده و IL را برای یک فایل اسمبلی مشاهده کنید:

bash

ildasm Calculator.dll

2. **dotPeek (JetBrains):**
   - یک ابزار دیگر برای مشاهده متاداده و بازسازی کد اصلی.
3. **System.Reflection:**
   - از کد C# یا VB.NET می‌توانید متاداده را به صورت برنامه‌ریزی‌شده بخوانید :

```
var methodInfo = typeof(Calculator).GetMethod("Add");

Console.WriteLine(methodInfo.Name);  // Add

Console.WriteLine(methodInfo.ReturnType);  // System.Int32
```

نحوه استفاده از متاداده توسط CLR

1. **بارگذاری اسمبلی:**
   - CLRمتاداده را برای درک ساختار برنامه و وابستگی‌های آن بارگذاری می‌کند.
2. **ایجاد اشیا:**
   - متاداده به CLR اجازه می‌دهد کلاس‌ها را بسازد و متدها را اجرا کند.
3. **مدیریت امنیت:**
   - از متاداده برای بررسی دسترسی و امنیت استفاده می‌شود.
4. **دیباگ و بهینه‌سازی:**
   - متاداده اطلاعات لازم برای دیباگ و بهینه‌سازی کد را فراهم می‌کند.

خلاصه

متاداده در دات نت اطلاعاتی غنی درباره ساختار و رفتار کد ارائه می‌دهد. برای متد Add، متاداده شامل نام، نوع بازگشتی، نوع پارامترها و دسترسی متد است. این اطلاعات توسط CLR برای اجرای کد، و توسط ابزارهایی مانند ILDasm برای تجزیه و تحلیل کد استفاده می‌شود.

9.اجرای فایل ها توسط CLR و نحوه کار و کاربرد JIT و AOT

اجرای فایل توسط CLR و نحوه کار با آن

CLR (Common Language Runtime) هسته اصلی دات نت است که مسئول اجرای کدهای میانی (IL) و مدیریت منابع در برنامه‌های دات نت می‌باشد CLR .یک محیط مدیریت شده را فراهم می‌کند که از برنامه‌ها در برابر مشکلاتی مانند مدیریت حافظه، جمع‌آوری زباله (Garbage Collection) ، و امنیت محافظت می کند. در اینجا مراحل اصلی اجرای فایل‌های دات نت توسط CLR آورده شده است:

1. **کامپایل کد منبع:**
   - کد منبع (C#, VB.NET و ...) ابتدا توسط کامپایلر به IL (Intermediate Language) تبدیل می شود. فایل خروجی معمولاً یک DLL یا EXE است.
2. **بارگذاری اسمبلی:**
   - وقتی یک برنامه دات‌نت اجرا می‌شود، CLR فایل اسمبلی (DLL یا EXE) را بارگذاری می کند. این کار شامل خواندن متا داده و کد IL است.
3. **مدیریت متاداده:**
   - CLR اطلاعات متاداده را تجزیه و تحلیل می‌کند تا بفهمد چه کلاس‌ها، متدها و ویژگی‌هایی وجود دارد و چگونه باید آن‌ها را مدیریت کند.
4. **ایجاد اشیا:**
   - با استفاده از متاداده، CLR اشیای لازم را از کلاس‌ها ایجاد می‌کند.
5. **اجرا:**
   - کد IL به کد ماشین توسط **JIT (Just-In-Time)** یا **AOT (Ahead-Of-Time)** کامپایل می‌شود. این کد ماشین بر اساس معماری سیستم (x86, x64) تولید می‌شود و در حافظه بارگذاری می‌گردد.

استفاده از JIT وAOT

1. JIT (Just-In-Time) Compilation:

- **تعریف:**
  - JIT یک تکنیک کامپایل است که کد IL را در زمان اجرا (runtime) به کد ماشین تبدیل می‌کند. این کار در اولین بار که متد فراخوانی می‌شود، انجام می‌شود.
- **مراحل کار:**
  - زمانی که یک متد از کد IL باید اجرا شود، JIT آن را به کد ماشین تبدیل می‌کند و کد تولید شده در حافظه کش (cache) می‌شود.
  - در دفعات بعدی که آن متد فراخوانی می‌شود، JIT از نسخه کش شده استفاده می‌کند.
- **مزایا:**
  - **بهینه‌سازیJIT :** می‌تواند بهینه سازی‌هایی انجام دهد که به شرایط runtime بستگی دارد.
  - **انعطاف‌پذیریJIT :** می‌تواند در هنگام اجرای برنامه تصمیمات بهینه سازی را بگیرد.
- **معایب:**
  - **زمان تاخیر:** در اولین بار فراخوانی متد، زمان تاخیر وجود دارد زیرا JIT باید کد را کامپایل کند.

2. AOT (Ahead-Of-Time) Compilation:

- **تعریف:**
  - AOT یک تکنیک است که در آن کد IL قبل از اجرای برنامه به کد ماشین کامپایل می‌شود. این کار معمولاً در زمان بیلد (build) انجام می‌شود.
- **مراحل کار:**
  - برنامه به صورت کامل به کد ماشین کامپایل می‌شود و یک فایل اجرایی مستقل تولید می‌شود.
  - این فایل می‌تواند بدون نیاز به CLR اجرا شود.
- **مزایا:**
  - **زمان بارگذاری سریع‌تر :** با AOT ، زمان بارگذاری برنامه به شدت کاهش می یابد زیرا دیگر نیازی به کامپایل JIT در زمان اجرا نیست.
  - **بدون نیاز به AOT: CLR می‌تواند برنامه را به گونه‌ای کامپایل کند که در محیط‌هایی که CLR موجود نیست، اجرا شود.**
- **معایب:**
  - **عدم بهینه سازی در زمان اجرا AOT :** نمی‌تواند بهینه سازی‌هایی بر اساس شرایط runtime انجام دهد، زیرا همه چیز از قبل مشخص است.
  - **حجم بیشتر: برنامه‌های AOT معمولاً حجم بیشتری دارند، زیرا شامل تمام کد ماشین و اطلاعات مورد نیاز برای اجرا هستند.**

تفاوت‌های JIT وAOT

AOT	JIT	ویژگی
قبل از اجرا (در زمان بیلد)	در زمان اجرا(runtime)	**زمان کامپایل**
بهینه‌سازی ثابت (بر اساس کد IL)	امکان بهینه‌سازی بر اساس شرایطruntime	**بهینه‌سازی**
معمولاً بزرگتر (کد ماشین مستقل)	معمولاً کوچکتر (کد IL همراه)	**حجم خروجی**
می‌تواند بدون CLR اجرا شود	نیاز بهCLR	**وابستگی بهCLR**
زمان بارگذاری سریع‌تر	زمان بارگذاری بیشتر (در اولین فراخوانی متد)	**زمان بارگذاری**

نحوه کار با JIT وAOT

کار با:JIT

- در اکثر موارد، شما نیازی به تنظیم خاصی برای JIT ندارید؛ CLR به طور خودکار از JIT استفاده می‌کند.
- می‌توانید از ابزارهای پروفایل سازی (profiling tools) برای بهینه سازی عملکرد استفاده کنید، زیرا JIT می‌تواند بر اساس اطلاعات runtime بهینه سازی کند.

کار با:AOT

- برای استفاده ازAOT ، معمولاً از ابزارهایی مانند **NGen (Native Image Generator)** یا **ReadyToRun** استفاده می‌شود.
- به عنوان مثال، برای کامپایل برنامه به کد ماشین پیش از اجرا:

bash

```
ngen install YourAssembly.dll
```

نتیجه‌گیری

در دات نت، CLR مسئول اجرای کدهای IL است و از تکنیک‌های JIT و AOT برای تبدیل IL به کد ماشین استفاده می‌کند JIT .کد را در زمان اجرا کامپایل می‌کند، در حالی که AOT این کار را قبل از اجرا انجام می‌دهد. هر یک از این روش‌ها مزایا و معایب خاص خود را دارند و بسته به نیازهای برنامه می‌توانید یکی از آن‌ها را انتخاب کنید.

9.1.روش های دیگه جز AOT و JIT

بله، علاوه بر **AOT (Ahead-Of-Time)** و **JIT (Just-In-Time)**، روش‌های دیگری نیز برای اجرای کد در دات نت وجود دارند. این روش‌ها به منظور بهبود عملکرد، کاهش زمان بارگذاری یا ارائه انعطاف پذیری بیشتر در نحوه اجرای کد طراحی شده‌اند. در اینجا به چند مورد دیگر اشاره می‌کنیم

1. Interpreted Execution

- **تعریف :**در این روش، کد به صورت خط به خط تفسیر و اجرا می‌شود، به جای اینکه به کد ماشین کامپایل شود.
- **ویژگی‌ها:**
  - معمولاً سرعت اجرای کمتری نسبت به JIT و AOT دارد.
  - تغییرات در کد می تواند به سرعت منعکس شود، زیرا نیازی به کامپایل مجدد ندارد.
- **استفاده :**بیشتر در زبان‌های اسکریپتی مثل **Python** و **Ruby** کاربرد دارد، اما در دات نت معمولاً از این روش استفاده نمی‌شود.

2. Native Image Generation (NGen)

- **تعریف** NGen **:**یک ابزار برای تولید تصاویر بومی (native images) از اسمبلی‌های دات نت است.
- **ویژگی‌ها:**
  - کد IL به کد بومی قبل از اجرای برنامه کامپایل می‌شود، مشابه AOT، اما می‌تواند با استفاده از CLR در محیط های مختلف اجرا شود.
  - Ngen می‌تواند به کاهش زمان بارگذاری و بهبود عملکرد برنامه کمک کند.
- **محدودیت‌ها** NGen **:**فقط برای اسمبلی‌هایی که به صورت ایستا (static) بارگذاری می‌شوند، عمل می‌کند.

3. ReadyToRun (R2R)

- **تعریف** ReadyToRun **:**یک تکنیک جدیدتر برای کامپایل کد IL به کد بومی است که به همراه اسمبلی دات نت می‌آید.
- **ویژگی‌ها:**
  - R2R ترکیبی از JIT و AOT است. زمانی که برنامه اجرا می‌شود، برخی از متدها از پیش به کد بومی تبدیل شده‌اند و برخی دیگر همچنان در زمان اجرا با JIT کامپایل می‌شوند.
  - این روش بهبود عملکرد را در زمان بارگذاری و اجرای متدها به همراه دارد.
- **مزایا :**به برنامه‌ها این امکان را می‌دهد که سریع‌تر اجرا شوند، بدون اینکه تمام کد به طور کامل از قبل کامپایل شود.

4. .NET Native

- **تعریف** .NET Native **:**یک فناوری است که برای برنامه‌های ویندوز و UWP (Universal Windows Platform) طراحی شده است.
- **ویژگی‌ها:**
  - به طور پیشرفته تری نسبت به AOT، کد IL را به کد بومی کامپایل می‌کند و از بهینه سازی‌های خاصی برای محیط های خاص استفاده می‌کند.
  - باعث می‌شود برنامه‌ها سریع‌تر بارگذاری و اجرا شوند.
- **محدودیت‌ها :**این روش بیشتر برای اپلیکیشن‌های ویندوز و UWP کاربرد دارد.

5. Crossgen

- **تعریف** Crossgen **:**یک ابزار برای تولید کد بومی از اسمبلی‌های دات نت است.
- **ویژگی‌ها:**
  - مشابهNGen ، اما با قابلیت‌های بیشتری برای بهینه‌سازی و تولید کد برای انواع مختلف پلتفرم‌ها.
  - می‌تواند برای تولید نسخه‌های بومی از اسمبلی‌های دات نت برای استفاده در حالت‌های مختلف (شامل موبایل و دسکتاپ) مورد استفاده قرار گیرد.

نتیجه‌گیری

علاوه بر AOT وJIT ، دات نت چندین روش دیگر برای اجرای کد دارد که هر یک به نوبه خود دارای ویژگی‌ها و مزایای خاصی هستند. بسته به نیازهای خاص یک برنامه و نوع پلتفرمی که بر روی آن اجرا می‌شود، می‌توان از این تکنیک‌ها استفاده کرد تا عملکرد و کارایی برنامه‌ها بهبود یابد.

9.2.رفتار clr با کد های زبان میانی

در دات نت، (Common Language Runtime) CLR نقش کلیدی در مدیریت اجرای کدهای میانی

(Intermediate Language یا IL) ایفا می‌کند. در اینجا، نحوه عملکرد CLR و نحوه تعامل آن با کد های زبان‌های میانی را بررسی می‌کنیم و رفتار آن را با دو متد ذکر شده (جمع دو عدد) توضیح می‌دهیم.

نقش CLR در دات نت

1. **اجرای کد میانی:(IL)**
   - کد منبع (به زبان‌های مختلف مانند #C یا VB.NET) توسط کامپایلر به کد میانی (IL) تبدیل می‌شود.
   - CLR مسئول بارگذاری این کد IL و اجرای آن است.
2. **مدیریت منابع:**
   - CLR به مدیریت حافظه، جمع‌آوری زباله (Garbage Collection) و امنیت در زمان اجرا کمک می‌کند.
3. **تجزیه و تحلیل متاداده:**
   - CLR متاداده موجود در اسمبلی را تجزیه و تحلیل می‌کند تا اطلاعات مربوط به کلاس‌ها، متدها و نوع داده‌ها را بفهمد.
4. **کامپایل به کد ماشین:**
   - با استفاده از **JIT (Just-In-Time)**، CLR کد IL را به کد ماشین (native code) تبدیل می‌کند که به CPU فهمیده می‌شود.
   - اگر از **AOT (Ahead-Of-Time)** استفاده شده باشد، این مرحله قبل از اجرا انجام شده و CLR از کد ماشین تولیدشده استفاده می‌کند.

رفتار CLR با دو متد جمع دو عدد

برای مثال، فرض کنید دو متد ساده برای جمع دو عدد به زبان #C و VB.NET داریم:

1.متد:C#

```
public int Add(int a, int b)
{
   return a + b;
}
```

2.متد:VB.NET

```
Public Function Add(a As Integer, b As Integer) As Integer
   Return a + b
End Function
```

9.3 نحوه کار CLR و نحوه کار با متود های بالا

گام 1: کامپایل کد منبع بهIL

- **کامپایلر (مثل Roslyn برای C#) کدهای منبع را به کد IL تبدیل می‌کند. این کد IL شامل دستورات مربوط به جمع دو عدد و نوع بازگشتی است.**
- برای هر زبان، کد IL مشابهی تولید می‌شود که رفتار یکسانی را نشان می‌دهد. در واقع، با نگاهی به IL تولید شده برای هر دو متد، می‌توان دید که فرمت IL مشابهی برای هر دو زبان وجود دارد.

گام 2: بارگذاری اسمبلی

- زمانی که برنامه اجرا می‌شود، CLR اسمبلی (که حاوی کد IL و متاداده است) را بارگذاری می‌کند.
- CLR متاداده را تجزیه و تحلیل می کند تا بفهمد متدها، پارامترها و نوع های داده چه هستند.

گام 3: کامپایل JIT یاAOT

- در زمان اجرا، وقتی متد Add برای اولین بار فراخوانی می‌شود، CLR از **JIT** استفاده می‌کند تا IL را به کد ماشین تبدیل کند. اگر از **AOT** استفاده شده باشد، کد بومی از قبل تولید شده و آماده است.

گام 4: اجرای کد ماشین

- کد ماشین به CPU ارسال می‌شود و عملیات جمع دو عدد انجام می‌شود. نتیجه به صورت بازگشتی به caller برمی‌گردد.

نتیجه گیری

- **CLR** مدیریت اجرای کد IL را بر عهده دارد و با استفاده از JIT یا AOT کد را به کد ماشین تبدیل می‌کند.
- **متدهای #C و VB.NET** با وجود تفاوت‌های ظاهری در نحوه نوشتن، در نهایت به کد IL مشابهی کامپایل می‌شوند که توسط CLR پردازش و اجرا می‌شود.
- به این ترتیب، CLRتضمین می‌کند که برنامه‌ها به طور صحیح و امن اجرا شوند، بدون توجه به زبان برنامه‌نویسی که برای نوشتن آن‌ها استفاده شده است.

10. il چیست و نقش ان و اهمیت ان

کد (Intermediate Language) IL، که به آن (Microsoft Intermediate Language) MSIL یا CIL

(Common Intermediate Language) نیز گفته می‌شود، زبان میانی است که در اکوسیستم دات نت برای توصیف کدها استفاده می‌شود. این کد، پس از کامپایل کد منبع (مانند #C، VB.NET و ...) توسط کامپایلر، تولید می‌شود و به

(Common Language Runtime) CLR ارسال می‌شود تا در زمان اجرا پردازش شود. در اینجا توضیحات جامع تری درباره IL ارائه می‌شود:

1.تعریفIL

- **(Intermediate Language) IL یک زبان میانی است که به کدهای منبع نوشته شده در زبان‌های مختلف داتنت (مانند #C، VB.NET و F#) تبدیل می‌شود.**
- IL به صورت کدهایی نوشته می‌شود که مستقل از معماری سخت افزاری و سیستم عامل هستند.

2.نقش IL در دات نت

- **کامپایل به IL: کد منبع به IL کامپایل می‌شود. این کد IL شامل متدها، کلاس‌ها و اطلاعات متاداده مربوط به اسمبلی است.**
- **اجرای IL: زمانی که برنامه اجرا می‌شود، CLR IL را بارگذاری کرده و آن را به کد ماشین (native code) تبدیل می‌کند.**
- **ایجاد سازگاری: IL به دات نت این امکان را می‌دهد که زبان‌های مختلف به طور یکپارچه با یکدیگر کار کنند، زیرا تمام زبان‌ها به IL کامپایل می‌شوند.**

3.ویژگی‌هایIL

- **زبان مستقل از پلتفرم: IL به گونه ای طراحی شده است که بتواند بر روی هر پلتفرمی که CLR اجرا می‌شود، کار کند.**
- **خود توصیف‌کنندگی: IL شامل متاداده‌ای است که اطلاعاتی درباره ساختار کد، نوع‌ها، متدها و ویژگی‌های دیگر را در خود دارد.**
- **بهینه‌سازی: CLR می‌تواند در زمان اجرا IL را به کد ماشین بهینه تبدیل کند، که به بهبود عملکرد برنامه‌ها کمک می‌کند.**

4.ساختارIL

- IL به صورت مجموعه‌ای از دستورات (opcodes) نوشته می‌شود که شامل عملگرها، پارامترها و اطلاعات دیگر است. برای مثال، برخی از دستورات رایج IL شامل موارد زیر است:
- **ldarg.0: بارگذاری پارامتر اول متد**.
- **add: جمع دو عدد**.
- **ret: بازگشت از متد**.

5.مثالIL

به عنوان مثال، متد جمع دو عدد در C# به IL به شکل زیر تبدیل می‌شود:

```
public int Add(int a, int b)
{
    return a + b;
}
```

کد IL معادل:

```
.method public hidebysig static
    int32 Add(int32 a, int32 b) cil managed
{
    .maxstack 2
    ldarg.0
    ldarg.1
    add
    ret
}
```

در این مثال:

- **ldarg.0 و ldarg.1** پارامترها را بارگذاری می‌کنند.
- **Add** عمل جمع را انجام می‌دهد.
- **Ret** نتیجه را برمی‌گرداند.

6.تجزیه و تحلیلIL

IL به صورت مجموعه‌ای از دستورات در فایل‌های DLL یا EXE دات‌نت ذخیره می‌شود. این فایل‌ها شامل:

- **متاداده:** اطلاعاتی درباره کلاس‌ها، متدها و نوع‌ها.
- **خود IL: کد واقعی که توسط CLR در زمان اجرا استفاده می‌شود.**

7.تبدیل IL به کد ماشین

- در زمان اجرا، CLRاز **JIT (Just-In-Time)**یا **AOT (Ahead-Of-Time)**برای تبدیل IL به کد ماشین استفاده می‌کند.
- **JIT: کد IL را به کد ماشین در زمان اجرا تبدیل می‌کند. این به برنامه‌ها اجازه می دهد تا با توجه به وضعیت موجود و نوع CPU بهینه سازی شوند.**
- **AOT: قبل از زمان اجرا کد IL را به کد ماشین تبدیل می‌کند، که زمان بارگذاری را کاهش می دهد.**

8.مزایایIL

- **سازگاری IL :**به دات نت این امکان را می‌دهد که از زبان‌های مختلف پشتیبانی کند و به برنامه نویسان اجازه دهد از ویژگی های مختلف هر زبان بهره برداری کنند.
- **بهینه‌سازی در زمان اجرا: CLR می‌تواند IL را در زمان اجرا بهینه کند و باعث افزایش کارایی برنامه‌ها شود.**
- **مدیریت خطا: IL به CLR این امکان را می‌دهد که به طور مؤثری خطاها را مدیریت کرده و بهبود یابد.**

9.جمع‌بندی

IL یک جزء اساسی از دات نت است که به برنامه نویسان این امکان را می‌دهد که از زبان‌های مختلف استفاده کنند و در عین حال از مزایای اجرای یکسان و بهینه سازی‌های CLR بهره مند شوند. IL به عنوان زبانی مستقل از پلتفرم و معماری، به دات نت اجازه می‌دهد تا کارایی بالا و قابلیت همکاری را فراهم کند.