Common Language Runtime

A Common Language Runtime (CLR) provides essential runtime services such as automatic memory management and exception handling. (The word “common” refers to the fact that the same runtime can be shared by other managed program ming languages, such as F#, Visual Basic, and Managed C++.) C# is called a managed language because it compiles source code into managed code, which is represented in Intermediate Language (IL). The CLR converts the IL into the native code of the machine, such as X64 or X86, usually just prior to execution. This is referred to as Just-In-Time (JIT) compilation. Ahead-of-time compilation is also available to improve startup time with large assemblies or resource-constrained devices (and to satisfy iOS app store rules when developing mobile apps). The container for managed code is called an assembly. An assembly contains not only IL but also type information (metadata). The presence of metadata allows assemblies to reference types in other assemblies without needing additional files.

ترجمه پاراگراف

Common Language Runtime (CLR)
یک CLRخدمات ضروری زمان اجرا را فراهم می‌کند، مثل مدیریت خودکار حافظه و مدیریت استثناها (exception handling). واژه‌ی "Common" به این معناست که یک runtime می‌تواند بین زبان‌های مدیریت‌شده‌ی دیگر مثل F# ، Visual Basicو C++ Managed مشترک باشد.

C#یک زبان مدیریت‌شده (managed language) نامیده می‌شود چون کد منبع آن به کد مدیریت‌شده (managed code)کامپایل می‌شود که در قالب Intermediate Language (IL)نمایش داده می‌شود . CLRاین IL را به کد بومی (native code) ماشین — مثل X64 یا X86 — تبدیل می‌کند، معمولاً درست پیش از اجرا. این فرآیند را کامپایل در لحظه Just-In-Time Compilation) یا (JIT می‌نامند.

کامپایل پیش از زمان اجرا Ahead-of-Time compilation) یا (AOT هم وجود دارد که زمان شروع (startup time)را برای اسمبلی‌های بزرگ یا دستگاه‌هایی با منابع محدود بهبود می‌دهد) و همین‌طور برای رعایت قوانین App Store در iOS هنگام توسعه‌ی اپ‌های موبایل.(

ظرفی که کد مدیریت‌شده را در بر دارد Assemblyنامیده می‌شود. یک اسمبلی نه تنها IL را شامل می‌شود، بلکه اطلاعات نوع (metadata) را هم دارد. وجود metadata باعث می‌شود اسمبلی‌ها بتوانند به انواع (types) در اسمبلی‌های دیگر ارجاع دهند، بدون اینکه فایل‌های اضافی نیاز باشد.

توضیحات تکمیلی و نکات ارائه

1. CLR = قلب.NET:
   - خدمات مهم:
     - مدیریت حافظه (Garbage Collection)
     - مدیریت استثناها (Exception Handling)
     - امنیت و مدیریتThread ها
2. Managed Code vs Unmanaged Code:
   - Managed Code:کدی که روی CLR اجرا میشه) مثل.(C#, F#, VB.NET
   - Unmanaged Code:کدی که مستقیم روی سیستم‌عامل و سخت‌افزار اجرا میشه) مثل C++ و C.)
   - مزیت :Managed امنیت بیشتر، مدیریت حافظه خودکار، پورتابل بودن.

3. Intermediate Language (IL):
   - وقتی کد #C رو کامپایل می‌کنی، خروجی مستقیم machine code نیست → اول IL تولید میشه.
   - ILمستقل از سخت‌افزاره.
4. JIT Compilation:
   - درست قبل از اجرا، CLRکد IL رو به native code دستگاه تبدیل می‌کنه.
   - مزیت: بهینه‌سازی کد بر اساس سخت‌افزار فعلی.
5. AOT Compilation:
   - کد قبل از اجرا به native code کامل کامپایل میشه.
   - کاربرد:
     - بهبود سرعت اجرا (startup) در پروژه‌های بزرگ.
     - در موبایل) مثل (iOS که JIT ممنوعه.
6. Assembly:
   - فایل خروجی پروژه #C همون اسمبلیه exe.) یا .(dll.
   - محتویات اسمبلی:
     - IL
     -) Metadataاطلاعات درباره‌ی انواع و متدها(
   - مزیت :Metadata اسمبلی‌ها می‌تونن بدون فایل‌های اضافی به هم ارجاع بدن.

جمع‌بندی ارائه برای این بخش:
CLR"هسته‌ی اجرایی #C هست که حافظه و خطاها رو مدیریت می‌کنه. کد #C اول به IL تبدیل میشه، بعد توسط CLR در لحظه به کد native سخت‌افزار ترجمه میشه .(JIT) برای بعضی شرایط خاص هم AOTوجود داره. خروجی پروژه‌ها اسمبلیه که علاوه برIL ، متادیتا هم داره و همین باعث میشه اسمبلی‌ها بتونن به همدیگه راحت وصل بشن".

You can examine and disassemble the contents of an assembly with Microsoft's ildasm tool. And with tools such as ILSpy or JetBrain's dotPeek, you can go further and decompile the IL to C#. Because IL is higher level than native machine code, the decompiler can do quite a good job of reconstructing the original C#. A program can query its own metadata (reflection) and even generate new IL at runtime (reflection.emit)

ترجمه پاراگراف

شما می‌توانید محتوای یک اسمبلی را با استفاده از ابزار ildasmمایکروسافت بررسی و جدا (disassemble)کنید. همچنین با ابزارهایی مثل ILSpyیا dotPeekاز JetBrains می‌توانید یک قدم جلوتر بروید و کد IL را دوباره به #C دیکامپایل (decompile) کنید. چون IL سطح بالاتری نسبت به کد ماشین بومی دارد، دیکامپایلر می‌تواند کار نسبتاً خوبی در بازسازی کد اصلی #C انجام دهد.

یک برنامه می‌تواند metadataخودش را پرس‌وجو کند (با استفاده از (reflectionو حتی در زمان اجرا کد ILجدید تولید کند (با استفاده از .(reflection.emit

---

🔻توضیحات تکمیلی و نکات ارائه

1. ildasm (IL Disassembler):
   - ابزاری رسمی از مایکروسافت برای مشاهده‌ی محتویات اسمبلی.
   - میشه IL و متادیتای اسمبلی رو بررسی کرد.
   - بیشتر برای یادگیری و دیباگ سطح پایین استفاده میشه.
2. Decompiler (ILSpy, dotPeek):
   - فراتر از disassemble → دیکامپایل IL به C#.
   - چون IL نزدیک‌تر به سطح بالاست (نسبت به کد ماشین)، خروجی دیکامپایلر شباهت زیادی به کد اصلی داره.
   - کاربرد:
     - بررسی اسمبلی‌های بدون سورس‌کد.
     - یاد گرفتن از کتابخانه‌های ناشناخته.
     - دیباگ کردن وقتی سورس‌کد در دسترس نیست.
3. Reflection:
   - قابلیتی در NET. که برنامه بتونه خودش رو در زمان اجرا بررسی کنه.
   - می‌تونی نوع‌ها، متدها، پراپرتی‌ها و ویژگی‌های کد رو در زمان اجرا بگیری.
4. Reflection.Emit:
   - پا رو فراتر می‌ذاره → می‌تونی در زمان اجرا IL جدید بسازی و اجرا کنی.
   - یعنی برنامه می‌تونه کد بسازه و همون لحظه اجرا کنه.
   - کاربرد:
     - ساخت Dynamic Proxies
     - ORMهایی مثلEntity Framework
     - تولید کد در زمان اجرا برای کارایی یا انعطاف بیشتر

جمع‌بندی ارائه برای این بخش:
"ابزارهایی مثل ildasm اجازه میدن اسمبلی رو باز کنیم و کد IL رو ببینیم. با ابزارهایی مثل ILSpy و dotPeekحتی میشه اون IL رو دوباره به C# تبدیل کرد. به خاطر سطح بالاترIL ، دیکامپایلرها خروجی خوبی میدن. علاوه بر این، با Reflection یک برنامه می‌تونه متادیتای خودش رو بخونه و حتی با Reflection.Emitکد جدید بسازه و در زمان اجرا اجرا کنه".