**목차**

[닷넷 2](#_Toc123520406)

[닷넷 2](#_Toc123520407)

[공통 중간 언어(CIL, Common Intermediate Language) 2](#_Toc123520408)

[공용 타입 시스템과 공용 언어 사양, 공용 언어 기반구조 2](#_Toc123520409)

[메타데이터, 어셈블리, 모듈, 매니페스트 3](#_Toc123520410)

[공용 언어 런타임과 닷넷 프레임워크 4](#_Toc123520411)

[C#과 닷넷 프레임워크, 닷넷 코어와 닷넷 표준, 닷넷5, 6 4](#_Toc123520412)

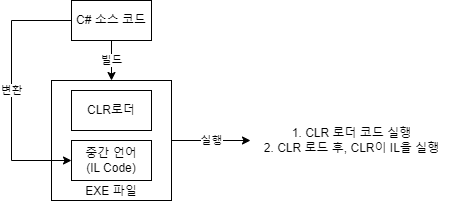
닷넷

## 닷넷

닷넷은 마이크로소프트에서 개발한 응용 프로그램 개발 환경이다. 일반적인 네이티브 언어와 다르게 닷넷 기반 프로그램은 닷넷이 설치된 환경에서만 실행된다.

닷넷을 설치하면 가상 머신 역할을 하는 CLR(Common Language Runtime)구성요소가 실행될 수 있는 환경이 마련된다. 닷넷 프로그램은 EXE/DLL로 기존 프로그램과 동일한 구조로 생겼지만, 내부적으로 CLR이 먼저 로드되어 실행되고 그 CLR이 함께 저장된 닷넷 코드(중간 언어)를 실행한다. 그러므로 C#을 C#컴파일러를 이용해 프로그램을 만들면 IL(Intermediate Language)이라 불리우는 중간 언어를 EXE/DLL 파일 내부에 생성하고, 프로그램이 시작하자마자 CLR을 로드하는 코드를 EXE파일에 추가한다.

CLR에 의해서 실행될 수 있는 중간 언어(IL, Intermediate Language)는 하나의 프로그래밍 언어에 종속된 것이 아니다. 그러므로 어떤 언어의 컴파일러이든지 결과물을 중간 언어로 생성한다면 CLR로 실행이 가능하다. 이처럼 중간 언어로 변역이 가능한 언어를 닷넷 호환 언어라고 하고, 이 중 마이크로소프트에서 공식적으로 제공되는 것은 C#, Visual Basic .Net, F#, C++/CLI가 있다. 또한 COBOL, Lisp, Python, PHP, Ruby 등의 언어도 중간 언어를 만들어내는 버전들이 있다. 닷넷 호환 언어들은 중간 코드 결과물을 공유하므로 상호간에 호출이 가능하다.



## 공통 중간 언어(CIL, Common Intermediate Language)

자바 VM에서의 중간 언어를 바이트코드(Bytecode)라고 하듯이, 닷넷의 CLR에서는 이를 CIL, IL, MSIL이라고 부른다. 모든 닷넷 호환 언어는 소스코드를 IL코드로 컴파일하고, CLR이 실행될 때 IL코드를 기계어로 최종 번역해 사용한다. 그러므로 닷넷 호환 언어는 CPU에 독립적인 결과를 낼 수 있다.

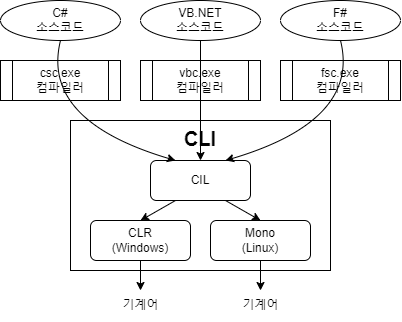
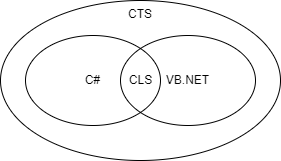
기계어와 어셈블리 언어가 대응하듯이, IL코드도 대응되는 IL언어가 있고, 이는 닷넷의 ilasm.exe를 통해서 컴파일할 수 있다. 그렇기 때문에 닷넷 호환 언어는 ‘언어 -> IL코드’ 또는 ‘언어 -> IL언어 -> ilasm.exe컴파일’와 같은 두 가지 방법을 이용해 주로 구현된다.

## 공용 타입 시스템과 공용 언어 사양, 공용 언어 기반구조

닷넷 호환 언어가 지켜야 하는 타입의 표준 규격을 정의한 것이 공용 타입 시스템(CTS, Common Type System)이다. 만약 새로운 언어를 이용해 닷넷 상에서 실행하고 싶다면 CTS 규약을 만족하는 내에서만 구현할 수 있다. 예를 들자면 CTS에서는 클래스의 단일 상속만을 하도록 정의하므로 언어가 클래스의 다중 상속을 지원하면 안 된다. 또한 이와 관련된 IL코드 또한 존재하지 않는다. 하지만 CTS에 정의된 모든 규격을 구현할 필요 또한 없으므로 필요하다고 판단되는 부분만 구현해도 괜찮다.

하지만 닷넷 호환 언어는 IL을 통해서 서로 호환이 가능해야 하므로 공용 언어 사양(CLS, Common Language Specification)에 정의된 내용만큼은 반드시 구현해야 한다.

공용 언어 기반구조(CLI, Common Language Infrastructure)는 마이크로소프트에서 ECMA 표준으로 제출한 공개 규약이다. CLI는 CTS를 포함하고, IL에 대한 정의, 메타데이터와 이들을 포함하는 이진 파일(binary file)의 구조까지 표준 사양으로 기술한다. CLI는 누구나 가져가서 임의로 구현할 수 있으며, CLI를 준수한 구현체에서 동작한 닷넷 파일은 또 다른 구현체에서도 실행하는 것이 가능하다. CLI를 마이크로소프트에서 구현한 실체가 바로 CLR이고, 그 밖에도 모노(Mono) 프레임워크, 닷넷코어(.Net Core)등이 오픈소스로 꾸준히 개발되고 있다.



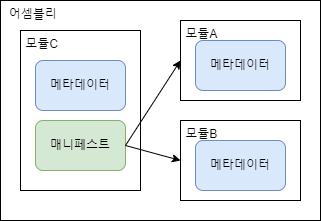
## 메타데이터, 어셈블리, 모듈, 매니페스트

메타데이터(Metadata)는 데이터를 기술하는 데이터이다. 예를 들면 그림 파일은 사진을 이루는 위치에 대한 컬러 값이 사진의 데이터이다. 반면에 이 데이터를 기술하기 위해서 너비, 높이, 해상도 등의 데이터를 표현할 수 있는데 이것이 메타데이터이다.

네이티브 언어들에서는 내가 작성한 코드에 대해서 해당 코드가 어떤 정보를 담고 있는지 코드 자체로는 알 수가 없다. 반면에 CLR에서 동작하는 실행 파일은 코드가 어떤 정보를 담고 있는지 자기 서술적인 메타데이터를 제공하고, 외부에서는 이를 리플렉션(Reflection)이라는 기술을 통해 사용할 수 있다. 그러므로 닷넷 호환 언어는 IL코드뿐 아니라 메타데이터도 함께 생성되게끔 해야 한다.

닷넷에서는 DLL, EXE파일과 같은 실행파일을 어셈블리(Assembly)라고 부른다. 이는 1개 이상의 모듈(Module)로 구성되고, 각 모듈들은 한 개의 파일이 대응된다. 만약 여러 개의 파일이 하나의 어셈블리를 구성하고 있으면, 이를 관리하는 데이터가 필요하고 모듈 중 하나는 반드시 해당 데이터를 가지고 있어야 한다. 이때의 데이터를 매니페스트(Manifest) 데이터라고 한다.

매니페스트를 포함하지 않는 모듈은 확장자가 netmodule이고, 매니페스트를 포함하는 모듈의 경우는 DLL, EXE이다. 어셈블리는 ‘참조 단위’이기도 하고 ‘배포 단위’이기도 하므로 다른 사람이 만든 어셈블리에 구현된 코드를 사용하고 싶다면 매니페스트가 포함된 모듈과 그와 관련된 모든 모듈을 가지고 있어야 한다.



## 공용 언어 런타임과 닷넷 프레임워크

공용 언어 런타임(CLR, Common Language Runtime)은 CLI사양을 따르는 가장 대표적인 VM이고, 마이크로소프트가 개발해 윈도우 운영체제용으로만 배포된다. CLR의 두 가지 큰 기능은 IL을 JIT 컴파일러를 이용해 기계어로 변환하는 것과 가비지 수집기(GC, Garbage Collector)를 제공해 동적 메모리 할당, 회수를 지원하는 것이다.

CLR은 프로세스 내에서 API에 의해 로드될 수 있다. 그러므로 윈도우 개발 환경에서 동작하는 기존 네이티브 프로그램에서 CLR을 내장해 필요할 때 닷넷을 활용하는 것이 가능하다. 그래서 CLR자체를 관리 환경(Managed Environment)라 하기도 하고, CLR이 로드되는 프로세스를 관리 프로세스(Managed Process)라고 부르기도 한다.

CLR에서 지원하는 기능만으로는 응용 프로그램을 개발하는데 불편함이 많이 생긴다. 그렇기 때문에 마이크로소프트에서는 이를 활용한 여러 가지 구성 요소를 함께 만들어 이를 하나의 패키지로 묶어 배포하는데, 이것이 ‘닷넷 프레임워크’이다. 즉, ‘CLR + 부가 구성 요소 = 닷넷 프레임워크’라고 볼 수 있으며, 부가 구성 요소는 BCL(Base Class Library), C#, VB.NET 컴파일러 및 각종 유틸리티 성격의 실행 파일, GAC(Global Assembly Cache) 등의 기능을 지원한다.

닷넷 응용 프로그램은 기존의 모든 윈도우 응용 프로그램에 대응해 만들 수 있다. 그리고 이를 만드는 과정에서 해야 하는 여러 작업들을 편하게 해주는 많은 기능들이 BCL에 구현되어 있다. 또한 새로운 버전의 닷넷 프레임워크가 출시될 때마다 기능이 추가됨으로써 BCL의 영역은 점점 커지고 있다.

## C#과 닷넷 프레임워크, 닷넷 코어와 닷넷 표준, 닷넷5, 6

위의 내용을 종합해보면 C#은 닷넷 프레임워크를 기반으로 IL을 생성하기 위한 도구이다. 즉, C#을 배우는 것은 닷넷 프레임워크를 배우는 것이나 마찬가지이다. 이때 다른 닷넷 호환 언어를 사용할 수도 있지만 C#은 마이크로소프트가 닷넷 프레임워크를 위해 만든 언어이기 때문에 C#을 사용하는 이점이 크다.

닷넷 코어(.NET Core)는 CLI의 다른 구현체이다. 그리고 이는 윈도우뿐 아니라 다른 플랫폼에서도 실행할 수 있도록 만들어진 것이 강점이다. 기존의 모노 프레임워크가 다중 플랫폼을 지원했지만 다중 플랫폼의 중요도가 올라가면서 마이크로소프트가 직접 나서서 만든 것이 닷넷 코어이다. 닷넷 코어는 CLI구현에 필요한 부분을 닷넷 프레임워크와 공유하는 형태로 개발됐으며, 소스코드가 공개되어 오픈소스로 개발되고 있다.

다중 플랫폼이 가능하지만 다르게 말하면 특정 플랫폼에서만 가능한 기능을 지원할 수 없다는 단점이 있다. 그렇기 때문에 닷넷 프레임워크와 닷넷 코어는 CIL에 해당하는 하부 구조는 동일하지만 이를 이용한 기반 라이브러리 단계부터는 서로 다른 라이브러리를 사용하게 되고 서로 호환되지 않아 라이브러리를 재사용할 수 없는 문제가 발생한다. 이를 해결하기 위해서 만들어진 호환 가능한 표준 규격을 닷넷 표준(.NET Standard) 라이브러리라고 한다.

2019년에 마이크로소프트는 닷넷 프레임워크 4.8이 마지막 버전이라고 발표했고, 이후 닷넷 코어 기반으로 이를 통합하기 시작했다. 그리고 2020년 11월 10일에 기존의 닷넷 프레임워크, 닷넷 코어라는 구분을 모두 없애고 .NET 5버전을 발표했으며 그로부터 1년 뒤 이를 개선한 .NET 6을 릴리스했다.

텍스트이(가) 표시된 사진

자동 생성된 설명