**CLR 호스팅**

CLR을 호스팅한다… 무슨말일까?

이 말인 즉, CLR이라는 기능이 많은 사무실을 작업공간으로 빌려 쓴다는 말로 이해하면 될 듯하다. 즉, 내가 CLR의 기능을 다 구현해서 쓰는 것이 아니고 CLR의 기능을 가져와서 사용한다는 것이다.

이것이 가능한 이유는 MS에서 CLR을 DLL에 포함된 일종의 COM 서버 형태로 구현했기 때문이다.

MS는 CLR을 DLL에 포함된 일종의 COM 서버 형태로 구현했다. 사용자가 닷넷 프레임워크를 설치하면 설치과정에서 CLR을 표현하는 COM 서버가 Windows 레지스트리에 등록된다. 어느 윈도우 프로그램이라도 CLR을 호스트할 수 있다. 하지만 CLR의 COM서버 인스턴스를 생성하기 위해서 관리되지 않는 어플리케이션은 인스턴스를 생성하기 위해서 CorBindToRuntimeEx나 MSCorEE.h에 정의되어 잇는 이와 유사한 함수를 사용해야 한다. CorBindToRuntimeEx는 MSCorEE.dll에 구현되어 있다. MSCorEE.dll은 심(shim)이라고 불리는데 이것이 하는 일은

- MSCorEE.dll 런타임 실행 엔진:

필요에 의해서 라이브러리가 참조될 때, 이를 자동적으로 인식해서 해당 어셈블리를 메모리에 로드하는 역할, 이밖에 주어진 어셈블리의 메타데이터를 읽어서 어떤 타입들(class, interface, struct 같은)이 필요한가를 찾아주고 어셈블리의 위치도 결정해 주는 일과같은 중요한 역할을 한다. 또한 IL을 컴파일하여 특정 플랫폼에 맞는 명령어 집합으로 만들어서 얼마만큼의 일을 해야 하고, 보안문제는 어떤가 체크해 주는 일도 처리 (IL을 컴파일한다? 그렇다! .net의 런타임이 이것이다. 그러므로 닷넷에서는 이 dll은 반드시 로드 해야 한다.)

CLR 자체는 심에 구현되어 있지 않고 MSCorWks.dll이라는 파일에 구현되어 있다. (이녀석은 프레임워크 버전마다 설치된다.) CorBindToRunTimeEx함수를 호출할 때 어느 버전의 CLR을 생성할지를 다른 몇 가지 필요한 설정과 함께 인자로 전달하면 된다. MSCorEE.dll 이 이 dll을 로드하는 역할을 한다.

Windows프로세스가 항상 CLR을 로드해야하는 것은 아니다. 관리되는 코드가 특정프로세스에서 실행되기를 원할 때만 해당 프로세스가 CLR을 로드하면 된다. 프로세스는 한가지버전의 CLR만 로드할수 있다. 한번 로드된 CLR은 프로세스 종료만이 답이다.

**AppDomain**

OS에서는 프로세스가 어플리케이션의 경계가 되었고 닷넷에서는 실행공간을 나누기 위해서 AppDomain개념을 사용하고 있다. 그리고 이런 AppDomain은 하나의 프로세스에 여러 개 있을 수 있다.

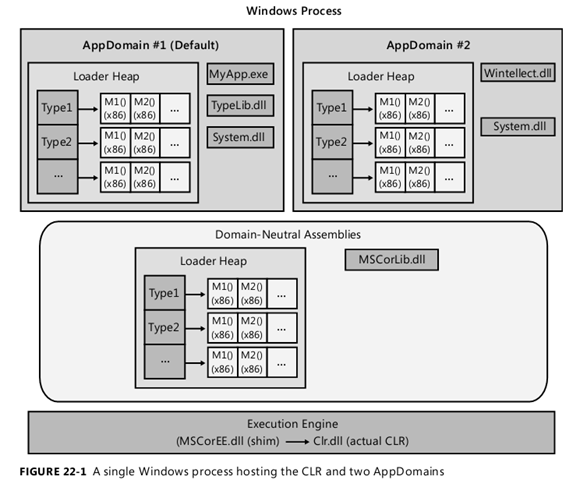
닷넷 어플리케이션이 시작되면, CLR은 기본적으로 default AppDomain을 생성하고 그 안에 어플리케이션을 실행시킨다. 두 어플리케이션이 동일한 어셈블리를 참조하고 있다면 어플리케이션은 각자의 어플리케이션 도메인으로 어셈블리를 로딩하고 객체도 각각의 도메인 안에서 따로 생성한다. 그래서 그림처럼 각 어플리케이션 도메인마다 동일한 타입의 객체들이 독립적으로 존재하게 된다. 어플리케이션 도메인간에는 객체의 인스턴스 멤버뿐만 아니라 정적 멤버도 공유가 되지 않는다. 독립된 실행공간이다.

**AppDomian의 기능:**

* + 한 AppDomain의 코드에서 생성된 객체는 다른 AppDomain의 코드에서 직접적으로 액세스될 수 없다. : 특정 AppDomain의 코드가 객체를 생성하면 이 객체는 해당 AppDomain에게 소유된다. 이 appdomain을 벗어나서 객체는 존재할 수가 없다. 접근을 하려면 marshal-by-reference 혹은 marshal-by-value의 접근 방법으로만 액세스하는 것이 가능하다. 이러한 격리가 가능하기 때문에 실행되는 다른 AppDomain에 전혀 영향을 주지 않고 쉽게 특정 AppDomain을 언로드할 수 있다.
  + AppDomain은 언로드가 가능하다. : CLR은 AppDomain에서 특정 어셈블리를 언로드 하는 기능은 주지 않았지만, CLR에게 AppDomain을 언로드하게 할 수 있으며, 이로 인해 AppDomain의 모든 어셈블리를 언로드하는 것이 가능하다.
  + 각 AppDomain별로 보안 적용이 가능하다. : 권한이 정해져 있기 때문에 관리되지 않는 코드를 로드하고, 로드된 코드가 호스트 어플리케이션에서 사용하는 자료구조를 손상시키거나 읽을 수 없다는 것을 보장 받을 수 있다.
  + 각 AppDomain별로 설정이 가능하다. : AppDomain이 생성될 때 이 Appdomain과 관련된 많은 설정을 할 수 있다. 이들 설정은 어셈블리 경로검색, 버전 바인딩 리디렉트, 쉐도우카피, 로더최적화등이 있다. (.config파일을 이용해서)

하나의 Windows 어플리케이션이 포함할 수 있는 AppDomain의 개수에는 제한이 없다. 각 AppDomain은 사용되고 있는 타입을 기록하는 자신만의 로더 힙을 가진다. 로더 힙에 있는 각 타입객체는 메서드 테이블을 가지며, 해당 메서드가 한 번이라도 호출되었다면 메서드 테이블의 각 항목은 JIT컴파일된 네이티브 코드의 위치를 가리킨다

.



그림에서 아랫단을 보면 심파일이 CLR을 불러오고 있다. AppDomain을 보면 로더힙에 객체가 뭔가를 가리키고 있다. 이것이 메서드 테이블이다 .

그림에서 같은 프로세스에서 두 개의 도메인이 같은 객체를 가지고 있는데 이 객체는 서로 공유되지 않고 독립적입니다. 물론 프로세스가 다르면 당연히 독립적입니다. 다음 그림을 보시면 기본 AppDomain이 3개의 어셈블리를 포함하고 app2가 두 개의 어셈블리를 포함하고 있습니다. 여기서 system.dll이라는 어셈을 둘다 로드하고 있기 때문에 양쪽에 로더힙에 동일한 타입 객체를 가지지만 이 객체는 AppDomain을 넘어서 공유 될 수 없습니다.

타입객체와 JIT 컴파일된 네이티브 코드의 메모리를 공유하지 않는 것은 비경제적이지만 AppDomain의 원래 목적이 “격리”기능을 제공하기 위해서기 때문에 어쩔 수 없다. 즉, CLR은 다른 AppDomain에 영향을 주지 않고 특정 AppDomain을 언로드하여 해당 AppDomain이 사용한 자원을 모두 해지할 수 있는 기능이 필요했기 때문에 이렇게 만들어 졌다.

특정 어셈블리의 경우 여러 AppDomain에서 사용될 용도로 만들어지기도 한다. 대표적인 dll이 MSCorLib.dll이다. 닷넷 프레임워크의 모든 기본적인 타입을 정의한다. 이 어셈블리는 CLR이 초기화될 때 자동으로 로드되며 모든 어셈블가 이 어셈블리에서 정의한 타입을 공유한다. 리소스의 사용을 줄이기 위해서 AppDomain-neutral 형태로 로딩된다. 이것은 CLR이 관리하는 특별한 로더힙으로서 모든 AppDomain이 공유하고자 하는 어셈블리를 위한 것이다. 이런 어셈블리는 프로세스 종료밖에 답이없다.

AppDomain UnLoad

1 CLR은 프로세스 내에서 관리되는 코드를 실행하는 모든 스레드의 동작을 잠시 중지한다.

2. CLR은 모드 스레드의 호출 스택을 검색해서 언로드되려고 하는 AppDomain코드를 실행시키거나 혹은 특정 시점에 AppDomain의 코드로 반환되는 스레드가 있는지를 모두 파악하고, 이 스레드들에게 ThreadAbortException 예외를 발생시킨다. 이렇게 함으로써 스레드들이 자신들이 구현한 Finally블록을 실행하여 스레드 종료에 필요한 작업을 실행할 수 있는 기회를 줄 수 있다. 만일 catch하지 않는다면 이것은 처리되지않는 예외가 되지만 이후 별다는 처리작업은 하지 않고 넘어간다 보통 처리되지 않는 예외가 발생하면 CLR은 프로세스자체를 종료하지만 이 경우에는 스레드는 종료 되지만 프로세스는 계속 실행된다.

3. CLR은 모든 힙을 조사하여 프록시 객체들 중에 언로드된 AppDomain에서 생성된 객체를 원본 객체로서 참조하는 것들이 있나 확인하고 해당 프록시 객체에 특정 플러그 값을 설정한다. 이후에 유효하지 않은 원본을 참조하는 프록시객체의 메서드를 호출하면 Exception이 발생된다,

4. CLRdms 가비지컬렉터를 실행시켜서 언로드된 APPDomain이 생성한 모든 객체의 메모리를 해제한다. 모든 객체들의 finalize 메서드가 호출되어 객체가 소멸되기전에 정리해야 할 작업을 할 기회를 준다.

5. CLR은 남아있는 모든 스레드의 작업을 다시 진행한다. AppDomain.UnLoad 메서드는 동기적으로 호출된다.

스레드가 Unload메서드를 호출하게 되면 CLR은 10초를 기다려주고 후에도 스레드가 반환되지 않으면 exception이 발생하며 , 이 appdomain은 언로드 될 수도 있고 안될 수도 있다.

호스트에서 AppDomain 이용하기

* 콘솔 및 Windows Forms 어플리케이션
* MS 인터넷 익스플로러
* MS ASP.NET의 Web Forms 와 XML 웹서비스 프로그램 어플리케이션

: 이 ASP.NET의 강점으로 웹사이트의 코드가 웹서버의 중단 없이도 바뀔 수 있다는 점이다. 웹사이트의 파일이 하드디스크에서 변경되면 ASP.NET은 이것을 감지하고 이전버전의 어셈블리를 포함한 AppDomain을 언로드한다. 그 후 요청이 처리된 새로운 AppDomain을 생성하고 새 버전의 어셈블리들을 여기에 다시 로드한다. 이 같은 기능은 쉐도우 카피가 있어 가능하다.

* MS SQL Server
* 개발자의 상상

**강력한 호스트 어플리케이션 구현**

호스트는 관리되는 코드에서 오류가 발생하면 CLR이 이를 어떻게 대처해야 하는지 설정할 수 있다. 약한순서부터 강한순서

* 만일 스레드가 실행되는데 혹은 결과가 반환되는데 많은 시간이 걸리면 CLR은 스레드를 취소할 수 있다.
* CLR은 AppDomain을 언로드 할 수 있다.

이렇게 함으로써 해당 AppDomain의 모든 스레드를 abort할 수 있으며 문제의 소지가 있는 코드를 언로드할 수 있다.

* CLR은 Disable될 수 있다.

이로써 더 이상 관리되는 코드가 프로세스에서 실행되지 못한다. 하지만 관리되지 않는 코드는 아직 실행이 가능한 상태이다.

* CLR은 Windows 프로세스를 종료할 수 있다.

모든 스레드를 abort시키고 모든 AppDomain을 언로드하여 모든 작업을 종료한다. 그런후 프로세스 자체를 종료시킨다.

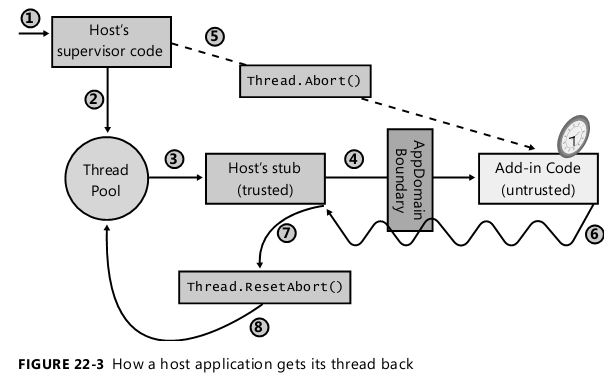
호스트는 에스켈레이션 정책이라고 불리는 정책 값을 설정하여 CLR이 관리되는 코드의 에외를 어떻게 다루어야 하는지 정의해 줄 수 있다.

중요한 영역에서는 정책이 달라진다.

중요한 영역에 있는 스레드가 예상치못한 예외를 만나게되면 CLR은 예외에 대한 처리를 (스레드수준이 아닌) AppDomain에 대해서 하게 된다. 우선 안전하게 AppDomain을 언로드하여 사용되는 모든 스레드와 객체를 제거하려 시도한다. 만약 특정시간이 지나도 언로드 되지 않는다면 CLR은 일방적으로 언로드 하게 된다.

**원래의 스레드로 복귀하는 방법**

호스트 어플리케이션에서 스레드가 무한루프에 빠졌을 경우 다시 돌아오는 방법



1. 클라이언트가 서버에 요청을 보낸다.
2. 요청을 서버의 스레드가 확인하고 스레드 풀 중에서 실제 작업을 담당할 스레드 하나를 선택한 후 선택된 스레드가 작업을 진행한다.
3. 스레드 풀에 있는 스레드는 클라이언트의 요청을 전달받고, 해당 호스트 어플리케이션을 개발하며, 테스트한 회사에서 구현한 신뢰되는 코드를 실행한다.
4. 그리고 이 신뢰되는 코드는 Try 블록에 진입한후 Try블록안에서 AppDomain경계를 넘는 메서드 호출을 수행한다. 이 시점에서 서버는 스레드의 제어를 신뢰되지 않는 코드에게 일임했다. 그러므로 서버 입장에서는 약간 불안해할 수도 있다.
5. 최초 호스트 프로그램이 클라이언트의 요청을 접수하게 될 때 호스트는 시간을 기록해 놓는다 .그리고 만일 신뢰되지 않는 코드가 일정 시간동안(관리자가 설정해놓은) 응답하지 않으면 호스트는 ThreadAbortException예외를 발생시켜서 해당 스레드를 종료할 수 있도록 CLR에게 스레드의 Abort 메서드를 실행하도록 요청한다.
6. 이제 작업 스레드의 종료 과정이 시작된다. Finally 블록이 실행되어 클린업 코드가 진행된다. 그리고 이 스레드는 결과적으로 AppDomain의 경계를 다시 넘어서 원래의 AppDomain으로 되돌아오게 된다. 이것이 가능한 이유는 신뢰되지 않는 코드를 실행하는 호스트의 스텁코드의 Try블록 내부에는 catch블록이 있어서 이 catch블록에서 TrreadAbortException예외를 catch하기 때문이다.
7. ThreadAbortExcetion 예외를 처리하는 Catch 블록에서 호스트는 스레드의 ResetAbort 메서드를 호출한다.
8. 호스트 프로그램은 ThreadAbortException 예외를 처리 완료했다. 이제 호스트 프로그램은 요청 작업이 실패했다는 것을 어떤 방법으로든 클라이언트에게 알려줄 것이다, 그리고 반환된 작업 스레드를 다시 스레드 풀로 복귀시킴으로써 이후에 있을 새로운 요청을 처리할 수 있도록 한다.

여기서 abort 메서드의 진행방식은 비동기이다. abort메서드를 호출하면 대상 스레드의 AbortRequest플러그만 설정하고 바로 반환된다. 그리고 스레드가 중단되어야 한다는 것을 런타임임이 알게 되면 런타임은 해당 스레드를 안전한영력으로 이동시킨다. (안전한 영역이란 관리되는 블록에서 sleep이나 wait상태의 스레드를 말한다.) 안전한영역에 있지못한 클래스란 타입의 클래스 생성자, catch나 finally블록 내부의 코드, 중요 영역의 코드 혹은 관리되지 않는 코드를 실행하고 있는 스레드를 의미한다.

**GAC(Global Assembly Cache)**

GAC(Global Assembly Cache)은 머신 차원의 공용 저장소로 이곳에 등록된 어셈블리는 머신에 설치된 모든 애플리케이션에서 같이 사용할 수 있다. 여러 애플리케이션에서 어셈블리에 접근하려면 그 어셈블리는 CLR이 인식할 수 있는 디렉토리에 있어야 한다. 참조하는 어셈블리를 애플리케이션이 로딩하려고 하면 CLR은 자동적으로 미리 정해진 그 디렉토리 구조를 따라가며 검색할 것이다.

GAC는 CLR이 이해할 수 있는 디렉토리 구조를 갖는다. GAC은 그러나 단순한 디렉토리가 아니다. 어셈블리의 버전닝 정책 즉 파일명은 같지만 버전번호가 다른 어셈블리가 동시에 존재할 수 있는 디렉토리 구조이며, 그리고 우연히 두 회사에서 출시한 어셈블리의 파일명이 같더라도 회사의 기밀 사항인 공개키/전용키 값이 다른 어셈블리가 동시에 존재할 수 있는 구조이다. 또한 어셈블리의 충돌을 피하기 위한 수단으로 GAC에 등록하려는 어셈블리는 반드시 디지털 사인과 공개키를 갖고 있어야 한다.

**IIS (Internet Information Service) 인터넷 정보 서비스**

MS Windows에서 제공되는 인터넷정보 서비스의 약자이다. 서버를 운영시 사용되는 MS window Server 등에서 서버의 시작과 끝, 또는 여러가지 서버 관련한 사항을 설정할 수 있는 서비스이다.

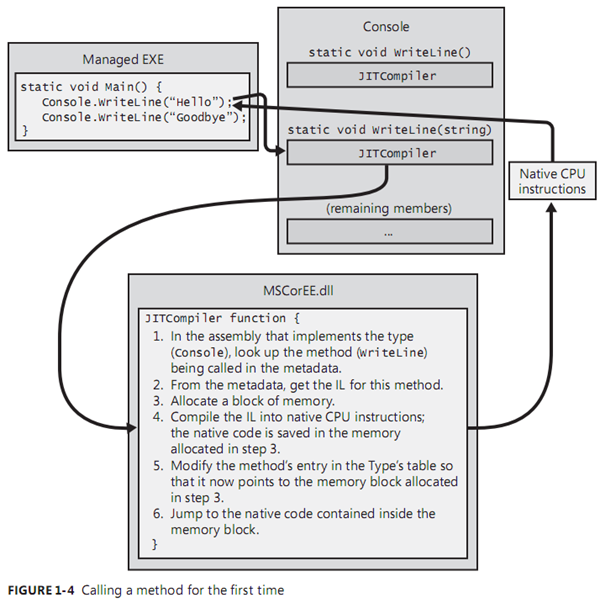
자바의 경우 제우스나 WebSphere등의 웹 서버를 사용합니다. 주로 대기업에서 고사양 고효율을 위해서 설계되어 있어서 많이 사용합니다. IIS의 경우는 주로 중소기업에서 사용합니다.

**서드파티**

공식적으로 하드웨어나 소프트웨어를 개발하는 업체 외에 중소규모의 개발자들이 주어진 규격에 맞추어 제품을 생산하는 경우를 말한다. 생산자와 사용자 사이에서 중개 역할을 하는 업체를 일컫기도 한다. 예를 들어 MS사의 VS2010을 사용해 프로그램을 만들면 우리회사가 서드파티가 된다.

**샌드박스**

보호된 영역 내에서 프로그램을 동작시키는 것으로, 외부 요인에 의해 악영향이 미치는 것을 방지하는 보안 모델. ‘아이를 모래밭(샌드 박스)의 밖에서 놀리지 않는다’라고 하는 말이 어원이라고 알려져 있다. 이 모델에서는 외부로부터 받은 프로그램을 보호된 영역, 즉 ‘상자’ 안에 가두고 나서 동작시킨다. ‘상자’는 다른 파일이나 프로세스로부터는 격리되어 내부에서 외부를 조작하는 것은 금지되고 있다.



메서드가 최초 호출될 때 일어나는 작업

1. 타입을 구현한 어셈블리의 메타데이터에서 해당 메서드를 찾는다.
2. 메타데이터로부터 이 메서드의 IL코드를 찾는다.
3. 메모리 블록을 할당한다.
4. IL코드르 네이티브 CPU지시어로 컴파일한다.
5. 3번에서 할당된 메모리 블록을 가르키도록 타입 테이블 내의 메서드 엔트리를 수정한다,
6. 메모리 블록내부에 포함된 네이티브 코드로 이동한다.

이 작업을 MSCorEE.dll 이 해준다.

**호스트**

자신보다 작거나 능력이 떨어지는 장치 또는 프로그램에게 서비스를 제공하는 장치나 프로그램을 의미한다.

**GUID(Globally Unique Identifier)**

전역 고유 식별자(Globally Unique Identifier, GUID)는 [응용 소프트웨어](http://ko.wikipedia.org/wiki/%EC%9D%91%EC%9A%A9_%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4)에서 사용되는 [유사 난수](http://ko.wikipedia.org/w/index.php?title=%EC%9C%A0%EC%82%AC_%EB%82%9C%EC%88%98&action=edit&redlink=1)이다. GUID는 생성할 때 항상 유일한 값이 만들어진다는 보장은 없지만, 사용할 수 있는 모든 값의 수가 2128 = 3.4028×1038개로 매우 크기 때문에, 적절한 [알고리즘](http://ko.wikipedia.org/wiki/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)이 있다면 같은 숫자를 두 번 생성할 가능성은 매우 적다.

GUID는 [오라클](http://ko.wikipedia.org/wiki/%EC%98%A4%EB%9D%BC%ED%81%B4_DBMS) [데이터베이스](http://ko.wikipedia.org/wiki/%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0%EC%9D%B4%EC%8A%A4) 등 많은 곳에서 쓰이지만, 가장 눈에 띄는 구현은 아마도 [마이크로소프트](http://ko.wikipedia.org/wiki/%EB%A7%88%EC%9D%B4%ED%81%AC%EB%A1%9C%EC%86%8C%ED%94%84%ED%8A%B8)의 구현일 것이다. 표준으로는 [오픈 소프트웨어 파운데이션](http://ko.wikipedia.org/wiki/%EC%98%A4%ED%94%88_%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4_%ED%8C%8C%EC%9A%B4%EB%8D%B0%EC%9D%B4%EC%85%98)(Open Software Foundation, OSF)이 지정한 [범용 고유 식별자](http://ko.wikipedia.org/w/index.php?title=%EB%B2%94%EC%9A%A9_%EA%B3%A0%EC%9C%A0_%EC%8B%9D%EB%B3%84%EC%9E%90&action=edit&redlink=1)(Universally Unique Identifier, UUID)가 있다.

GUID는 '그위드'라 발음하는데, 이는 특히 마이크로소프트에 의해 사용된다. 다른 발음으로는 '구이드'가 있다.

**마샬링**

컴퓨터 프로그래밍에서, 마샬링은 하나 이상의 [프로그램](http://terms.co.kr/program.htm) 또는 연속되어 있지 않은 저장 공간으로부터 [데이터](http://terms.co.kr/data.htm)를 모은 다음, 데이터들을 메시지 [버퍼](http://terms.co.kr/buffer.htm)에 집어넣고, 특정 수신기나 프로그래밍 인터페이스에 맞도록 그 데이터를 조직화하거나, 미리 정해진 다른 형식으로 변환하는 과정을 말한다.

마샬링은 대체로, 어떤 한 언어로 작성된 프로그램의 출력 [매개변수](http://terms.co.kr/parameter.htm)들을, 다른 언어로 작성된 프로그램의 입력으로 전달해야 하는 경우에 필요하다.

**프록시 개체**

클라이언트에서 원격 개체를 참조하기 위해 만든 가상의 개체 클라이언트 영역에서 원격 개체처럼 행동한다.

**스토어드 프로시저**

Stored procedure[스토어드 프로시저]는 [DBMS](http://www.terms.co.kr/DBMS.htm)에서 데이터베이스 서버와 함께 저장되어 있는 연산을 말하며, 스토어드 프로시저는 연속된 SQL문들을 하나로 모아 SQL 서버에 미리 컴파일해서 저장해 놓은 것을 말하는데, 클라이언트로부터 호출문을 통해 복잡한 SQL의 일괄 작업을 수행하는 데 적합하다. 미리 저장이 되어있기 때문에 실행이 빠르다.

**COM (Component object Method)이란**?

COM은 소프트웨어 컴포넌트들의 통합을 위한 인터페이스 표준이다. COM은 COM 규약대로 따라 만들면 언어에 구애받지 않고 작성된 프로그램뿐만 아니라 파일, DB처럼 다양한 데이터 저장공간으로부터 정보를 얻는 것도 가능하다.  이런 다른종류의 프로그래밍 언어, 서로 다른 프로그램. 서로 다른 OS의 머신 간의 통신을 위한 일종의 약속된 표준 인터페이스가 COM이라고 할 수 있다.

프로그램을 작성하는 ‘틀’과 ‘방법’을 제공해주는 일종의 도서관 같은 놈이다.

COM 객체는 QueryInterface 라는 함수를 통해서 자신이 지원하는 기능이면 들어가는 문(포인터)를 알려주고 만약 지원이 안된다면 null을 리턴한다.

**interpreter : 인터프리터**

|  |
| --- |
| 인터프리터는 [고급언어](http://terms.co.kr/high-levellanguage.htm)로 작성된 [원시코드](http://terms.co.kr/sourcecode.htm) 명령어들을 한번에 한 줄씩 읽어들여서 실행하는 [프로그램](http://terms.co.kr/program.htm)이다. 고급언어로 작성된 프로그램들을 실행하는 데에는 두 가지 방법이 있다. 가장 일반적인 방법은 프로그램을 컴파일 하는 것이고, 다른 하나는 프로그램을 인터프리터에 통과시키는 방법이다. 인터프리터는 고급 [명령어](http://terms.co.kr/instruction.htm)들을 중간 형태로 번역한 다음, 그것을 실행한다. 이와는 대조적으로, [컴파일러](http://terms.co.kr/compiler.htm)는 고급 명령어들을 직접 기계어로 번역한다.  컴파일된 프로그램들은 일반적으로 인터프리터를 이용해 실행시키는 것보다 더 빠르게 실행된다. 그러나 인터프리터의 장점은 [기계어](http://terms.co.kr/machinecode.htm) 명령어들이 만들어지는 컴파일 단계를 거칠 필요가 없다는데 있다. 컴파일 과정은 만약 원시 프로그램의 크기가 크다면, 상당한 시간이 걸릴 수 있다. 이와는 달리 인터프리터는 고급 프로그램을 즉시 실행시킬 수 있다. 이런 이유 때문에, 인터프리터는 종종 프로그램의 개발단계에서 사용되는데, 그것은 프로그래머가 한번에 적은 량의 내용을 추가하고 그것을 빠르게 테스트 해보길 원하기 때문이다.  **쉐도우 카피**  일반적으로. 서버측 환경(IIS, COM+)에서 DLL이 한번 로딩되면 그 어셈블리에 다른 수정을 가하지 못하다록 읽기 락(read lock)이 걸린다. 따라서 DLL이 IIS, COM+환경하에서 로딩되고 나면 새로운 버전으로 그것을 덮어쓸 수가 없게 된다. 결국 새로운 버전을 배포하려면 애플리케이션의 실행을 중지시키고 해당 DLL의 락을 해제한 후에나 가능하게 된다. 섀도우 카피는 이런 문제를 해결하기위해 사용될 수 있는 개념이다. CLR이 섀도우 카피를 이용하여 어셈블리를 로딩할때는 일단 임시 디렉토리로 어셈블리를 복사하고 나서 원본 대신에 그 복사본을 로딩하는 것이다. 따라서 복사본이 로딩되어 있는 동안에도 원래의 디렉토리로 배포가 가능하다. |