.NET Language특성

김동근* · 김태석**

1. Delegate와 Event

C# 언어의 혁명 중 하나로 C++의 함수 포인터와 같은 용도를 제공하는 Delegate라고 불리는 것이 있다. 그러나 Delegate는 Type 안정적이며, 보안관리 되는 객체이다. 이 말은 Delegate가 사용가능한 메소드, 즉 위험요소 없는 함수 포인터라는 것이다. Microsoft 윈도우 프로그래밍에서 Callback 메소드는 어떤 함수를 호출하고자 하는 함수에게 그 함수의 함수 포인터를 넘겨줄 필요가 있을 때 사용된다. Callback이 가진 목적은 많지만 가장 일반적인 목적은 다음과 같다.

• 비동기적인 처리: Callback 메소드는 어떤 요청을 처리할 적시에 코드가 실행되어야 하는 경우 비동기적으로 사용된다. 비동기 호출의 경우 오랫동안 작업이 Block되지 않고 Client가 작업을 계속 수행할 수 있도록 함으로써 확실한 이익을 줄 수 있다.

• Class의 코드 경로로 Custom Code를 삽입: Callback 메소드의 또 다른 일반적인 사용처로 Client가 커스텀한 프로세싱을 수행하기 위해 호출되는 메소드를 지정하도록 Class가 허용할 때이다.

*㈜웹타임 교육센터, Microsoft Certified Trainer(MCT)
**동의대학교 소프트웨어공학과 교수

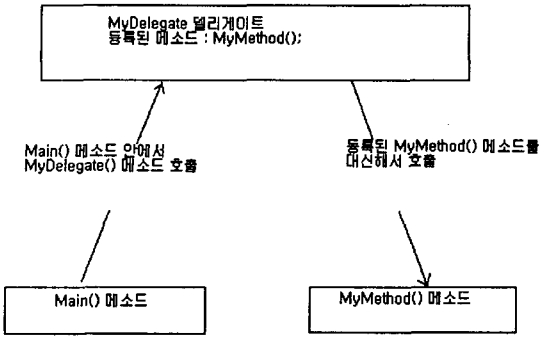
Delegate는 .NET 전반에 걸쳐서 사용되는데, 비 동기 프로그래밍이나 쓰레드 프로그래밍, 각종 Notification을 예로 들 수 있다.

C#에서는 Delegate를 "delegate"라는 키워드로 표현하고, 이것은 메소드들에 대한 참조를 가질 수 있는 하나의 타입이다. Delegate를 이용하는 것 중에 가장 핵심적인 것은 이벤트이다. 이벤트는 GUI프로그램에서 많이 쓰이는 개념이지만 C#에서는 Delegate로 이벤트를 표현한다. C#에서는 이벤트를 "event"라는 키워드로 표현하고 이것은 Class의 멤버이다.

우선 Delegate를 좀 더 살펴보면 사전적 의미는 "위임하다"라는 뜻이다. 그럼 Delegate는 무엇을 위임하는 것일까? 바로 메소드를 대신해서 호출을 받아서 우리가 원하는 메소드를 대신해서 호출한다. 즉 Delegate는 호출하는 부분과 호출받는 부분의 가운데서 중재자 역할을 한다고 볼 수 있다. 사용 예는

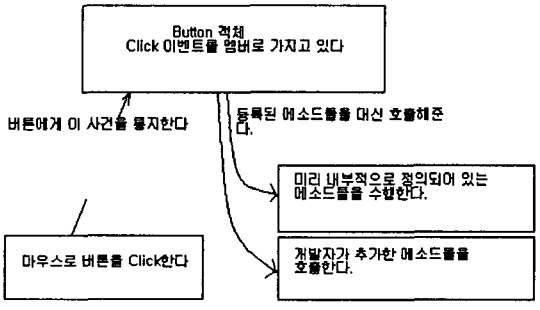
delegate [리턴타입][Delegate이름][인자리스트]

이는 보통의 사용자 정의 타입을 만들 때와는 매우 다르다. 이는 Delegate를 만들 때 위와 같이 메소드의 시그너처를 써주어야 한다. Delegate는 메소드를 대신해서 호출하는 것이기 때문에 호출할 메소드의 시그너처를 미리 정의하는 것이다. 이벤트는 사전적 의미는 "사건"이다. 특정한



델리게이트의 기능

사건이 객체에 일어 났을 때, 특정한 사건을 미리 등록되어 있는 다른 객체에게 알려 주는 기능이 바로 이벤트이다. C#에서 이벤트는 Delegate로 만든다. 이벤트가 가장 많이 쓰이는 부분은 바로 GUI(Graphic User Interface)프로그래밍이다. 이 말은 사용자가 마우스로 버튼을 Click하는 "사건"이 발생하면 개발자가 미리 등록해 놓은 메소드를 호출할 것이다. 이것처럼 이벤트는 통지의 기능으로 많이 쓰인다.



이벤트 멤버의 기능

이벤트는 대개 마우스 단추를 누르거나, 키를 누르거나, 데이터를 변경하거나, 문서 또는 폼을 여는 등의 사용자 동작에 의해 수행되지만 프로그램 코드나 시스템에 의해서도 수행될 수 있다. 이벤트 프로시저를 작성하여 아래와 같은 두 가지 수준에서 그러한 동작에 응답할 수 있다.

• **문서 수준 또는 하위 문서 수준 이벤트** : 이 이벤트는 열려 있는 문서에 대해 발생하며 때로는 열려 있는 문서의 개체에 대해서도 발생한다. 예를 들어, Word Document 개체는 Open, New, Close 이벤트에 응답할 수 있고, Excel Workbook 개체는 Open, BeforeClose, BeforeSave 등과 같은 이벤트에 응답할 수 있으며, Excel Worksheet 개체는 Activate, Calculate 등과 같은 이벤트에 응답할 수 있다. Microsoft® PowerPoint®는 응용 프로그램 수준 이벤트만 지원한다.

• **응용 프로그램 수준 이벤트** : 이 이벤트는 응용 프로그램 수준에서 발생한다. 예를 들어, Microsoft® Word 문서, Microsoft® Excel 통합 문서 또는 PowerPoint 프레젠테이션을 만들 때 각각 NewDocument, NewWorkbook, NewPresentation 등의 이벤트가 발생한다.

ListBox 및 TextBox 개체 같이 이들 개체에 포함된 대부분의 컨트롤에 대한 이벤트에 응답하는 다른 모델을 제공한다.

COM+에서 제공하는 Loosely Coupled Event (느슨하게 결합된 이벤트) 모델은 게시자 및 가입자와 이벤트 시스템 간에 런타임에 바인딩된 이벤트 또는 메서드 호출을 지원한다. 서버를 반복적으로 폴링하는 대신 이벤트 시스템에서는 정보를 사용할 수 있게 되었을 때 이를 이해 관계자에게 알린다. 이 서비스를 사용하려면 이벤트 클래스와 이벤트 싱크가 System.EnterpriseServices. ServicedComponent 클래스에서 직접 또는 간접적으로 파생되어야 한다. 싱크와 이벤트 클래스 간에 영구 또는 임시 가입을 만들려면 COM+ 관리 개체를 스크립트 또는 관리되는 코드와 함께 사용한다. COM+ Explorer를 사용하여 임시 가입을 만들 수도 있다.

2. Application Domain

.NET에서는 하나의 프로세스를 AppDomain 이라고 하는 Application Domain 단위로 나눌 수가 있다. 또한 Application Domain은 쓰레드 단위로 다시 나눌 수가 있다. 여기서 Application Domain의 개념은 .NET에서 처음 나오는 개념이다. 우선 프로세스와 쓰레드에 대한 개념을 조금 이해하고 있어야 하는데 개괄적으로 살펴보면 우선 프로그램이 실행되지 않고 디스크에 그냥 있다면 그건 단지 파일일 뿐이다. 그러나, 프로그램을 실행 시키면 해당 파일이 메모리에 로딩되면서 프로세스가 된다. 이 프로세스는 모든 프로그램의 실행 단위가 된다. 그리고 대부분의 운영체제에서 하나 이상의 프로세스를 아주 작은 단위로 나누어서 교대로 프로세스를 실행 시키게 되는데, 프로세스를 작은 단위로 자른 것이 쓰레드가 되며, 이 쓰레드를 운영체제 수준에서 관리하게 되는데 프로세스 안에는 최소한 하나의 쓰레드가 존재하며, 다중의 쓰레드가 존재할 수 있다. 이렇게 운영체제가 관리하는 기본 단위로서 프로세스는 운영체제가 교대로 실행 시키게 되는데, 그 속도는 사용자가 인지할 수 없을 만큼 빠르기 때문에 사용자는 동시에 여러 프로그램을 실행하고 있는 것처럼 느껴지게 된다. 이를 멀티태스킹이라고 한다.

여기서 교대로 프로세스를 실행하는 것은 매우 복잡한 메커니즘으로 메모리를 이용하던 프로세스의 행적을 메모리 어딘가에 저장하고, 다음 프로세스를 메모리에 다시 올리는 작업을 컨텍스트 스위칭(Context Switching) 이라고 한다. 다시 말해서 과도한 컨텍스트 스위칭은 CPU나 운영체제에 쓸데없는 작업을 늘리는 결과를 일으킬 수가 있다. 그래서 컨텍스트 스위칭에 대한 오버헤드를 줄이고, 여러 작업을 동시에 하기 위해 쓰레드라는 개념을 도입하고 프로세스를 나누었다. 그래서 하나의 프로세스에 여러 개의 쓰레드가 존재해서 실행되어지는 환경을 멀티쓰레드라고 한다. 쓰레드는 운영체제뿐만 아니라 프로그램 언어에서도 지원되고 있는데, 물론 .NET에서도 완벽하게 지원되고 있다. 그러나 .NET에서는 쓰레드를 운영체제가 관리하는 프로세스가 아닌 AppDomain이라고 하는 Application Domain에 넣어서 관리하고 있다.

하나 이상의 쓰레드는 Application Domain 안에서 수행 되어진다. 다시 말해서 Application Domain은 물리적인 프로세스 안에 있는 논리적인 프로세스라고 할 수 있다. 물리적인 프로세스는 운영체제에서 관리하지만 Application Domain은 .NET의 CLR에서 관리한다.

프로세스는 하나 이상의 Application Domain을 소유할 수 있고, Application Domain은 하나 이상의 쓰레드를 소유할 수 있다. 물론 프로세스는 최소 하나의 Application Domain을 가지고 Application Domain은 최소 하나의 쓰레드를 가진다. 프로세스는 운영체제가 가상의 메모리 주소공간을 할당해 주지만, Application Domain은 프로세스 안에서 하나의 가상 메모리 주소공간을 서로 공유한다. 그러나, CLR의 메모리 관리를 통해서 하나의 프로세스 안에 있는 여러 Application Domain들은 각자가 엄격히 분리되어 실행되어 진다.

Application Domain을 다시 정리해 보면

◦ Application Domain은 CLR에서 프로세스의 역할을 한다.

◦ Application Domain은 하나의 프로세스에서 유효하다.

◦ 하나의 프로세스는 하나 이상의 Application Domain을 가질 수 있다.

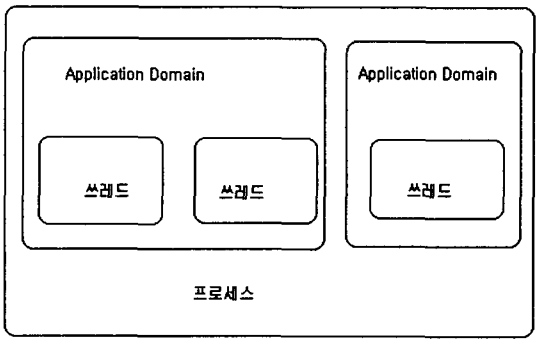
◦ 하나의 프로세스에 여러 Application Domain

이 존재하면 프로세스에 할당된 가상 메모리 주소 공간을 모든 Application Domain들이 나누어 사용한다.

• Application Domain의 메모리 관리는 프로세스의 메모리 관리비용보다 작게 든다.

• 프로세스 간의 통신 비용보다 Application Domain끼리의 통신 비용이 훨씬 작게 든다.

• 운영체제는 하나의 프로세스가 오류를 발생시켜 종료 되어도 다른 프로세스에 영향을 미치지 않게 하고, .NET의 CLR은 프로세스 안에 있는 하나의 Application Domain이 오류를 일으켜 종료되어도 같은 프로세스에 존재하는 다른 Application Domain에 영향을 주지 않도록 한다.



3. Attribute

.NET에서 Attribute가 제공하는 기능은 새로운 것이라고 할 수 있다. Attribute를 디자인 Time의 정보, Runtime의 정보, 혹은 실행 시점의 실행 특징 등을 정의하는데 사용할 수 있다. 예를 들면, Field에 대한 데이터베이스 컬럼, 문서 정보, 멤버에 대한 트랜잭션 정보 등을 들 수가 있다.

C#에서 가장 특징적이고 낯 설은 개념 중 하나라고 할 수 있다. 런타임에 프로그램 Logic과 관계없는 부가적인 정보를 전달하기 위해 사용되는 것이다. 다시 말해서 해당 프로그램에 존재하는 타입이나 메소드 프로퍼티 등의 행동을 런타임에 변경할 수 있게 해주고, 개발자가 직접 Attribute를 만들어서 자신의 이름 등의 정보를 런타임에 알 수 있게 해준다. 런타임에 알 수 있는 방법은 Reflection 개념을 이용한다.

Attribute를 사용하는 방법은 크게 두 가지로 나눌 수 있다.

첫째: 기존에 .NET에 존재하는 각종 Attribute들을 사용해서 프로그래밍 하는 것이고

둘째: 개발자가 직접 자기만의 사용자 정의 Attribute를 만들어서 프로그램에 들어가기를 원하는 정보를 넣는 것이다.

Attribute는 내부적으로는 하나의 Class이다. 즉 모든 Attribute는 Attribute Class에서 파생된다. 이는 델리게이트에서 코드상의 표현 방식이 Class가 아닌 것처럼 보이는 것과 마찬가지이다.

.NET에는 기본 Class로 Attribute와 관련해서 미리 정의되어 있는 많은 Attribute Class들이 있다. 사용하는 방법은

[Attribute 이름(위치 인자, 이름 인자=값,·)]처럼 사용한다. .NET에서 제공하는 주요 Attribute들을 살펴보면

Attribute클래스에는 사용자 지정 특성에 액세스하고 해당 Attribute를 테스트하는 데 편리한 메서드가 들어 있다. 모든 사용자 정의 형식은 Attribute로 사용할 수 있지만 대부분의 Attribute는 **Attribute**에서 파생된 형식의 인스턴스가 되어야 한다.

모든 특성은 **Attribute** 클래스에서 직접 또는 간접으로 파생된다. 특성은 모든 대상 요소(AttributeTargets 참조)에 적용할 수 있다. 특성의 여러 인스턴스는 같은 대상 요소에 적용할 수 있으며 특성은 대상 요소에서 파생된 요소에서 상속 받을 수 있다. 컴파일러와 다른 개발 도구에

Attribute 이름	사용위치	의 미
Conditional	메소드	조건부 컴파일을 할 수 있다.
DllImport	메소드	비관리형 코드로 만들어진 .dll 파일을 사용할 수 있도록 해준다.
Obsolete	Attribute, 타입, 메소드	해당 메소드의 기능이 개선된 메소드가 있으면, 그 메소드를 사용하는 곳에 알리기 위해 사용된다. 이름인자에는 경고나 에러메세지에 나타나게 할 문자열과 경고/에러 메시지를 선택하는 bool 타입이 올 수 있다.
Serializable	Class	해당 클래스에 대한 객체의 값들을 직렬화할 수 있게 해준다.
Transaction	Class	해당 클래스에 대한 객체가 이용할 수 있는 트랜잭션을 명시한다. 이름인자에는 TransactionOption 열거 타입의 멤버가 들어간다.
WebMethod	메소드	해당 메소드를 웹 서비스를 위한 웹 메소드임을 나타낸다.
WebService	Class	해당 클래스가 웹 서비스를 위한 것임을 나타낸다.
CLSCompliant	Assembly, 타입, 멤버	CLS에 맞는지를 알려준다.
AssemblyTitle	Assembly	해당 Assembly의 제목을 명시할 수 있다.
AssemblyCompany	Assembly	해당 Assembly의 제조회사 이름을 명시할 수 있다.
AssemblyVersion	Assembly	해당 Assembly의 버전을 명시할 수 있다.

서는 이 정보를 사용하여 사용자 지정 특성을 식별한다.

사용자 지정 특성은 메타데이터의 모든 요소와 함께 저장할 수 있다. 이 메커니즘을 사용하면 컴파일 타임에 응용 프로그램별 정보를 저장하고 런타임 또는 다른 도구에서 메타데이터를 읽을 때 이 정보에 액세스할 수 있다.

.NET Framework에서는 일부 Attribute 형식을 미리 정의하여 런타임 동작을 제어하는 데 사용할 수 있다. 일부 언어에서는 Attribute 형식을 미리 정의하여 .NET Framework 공용 형식 시스템에서 직접적으로 표현되지 않는 언어 기능을 표현한다. 사용자 또는 다른 도구에서 추가 특성 형식을 정의하고 사용할 수 있다.

윈도우의 레지스트리에 정보를 기록하는 Application이 있다고 가정하면 설계 원칙 중의 하나는 레지스트리의 정보를 저장하는 저장소에 관한 것이다. 대부분의 개발환경에서 이러한 정보는 전통적으로 리소스 파일이나 혹은 레지스트리 API를 호출하는 코드 속에 직접 표시되기도 한다. 이러한 경우 Class의 필수적인 부분이 해당 Class의 나머지 부분과 별도로 저장되는 것이다. Attribute를 이용하면 이러한 정보를 Class의 멤버에 접착시킬 수 있어서 자기자신을 설명할 수 있는 컴포넌트를 가지게 된다. Attribute 데이터 형이나 멤버 앞에 []안에 표시해 주어야 한다.

4. Reflection

.NET의 아주 강력한 기능 중의 하나인 Reflection은 Application의 메타 데이터를 액세스할 수 있는 코드를 만들도록 허용하는 기능이다. 간단히 Reflection은 Runtime에 형 정보를 알아내는 능력이라 할 수 있다. .NET의 Reflection API는 System.Reflection Namespace 안에 정의 된 Class들의 내용 그 자체를 말한다.

Reflection은 개발자가 Assembly 혹은 Assembly에 속한 타입의 멤버들을 런타임에 확인할 수 있게 해준다. 런타임에서 객체의 멤버를 동적

으로 알 수 있는 이유는 CLR때문이다. CLR은 Application Domain이 운영체제가 관리하는 메모리에서 제대로 수행될 수 있도록 관리해준다. 특히 Application Domain에 로드 되어 있는 Assembly를 관리하고 해당 Assembly에 들어 있는 타입들을 계층적으로 관리하게 되는데, 결국 CLR에서 현재 각 Application Domain에서 수행되고 있는 모든 타입들을 관리하기 때문에 CLR에서 작동하는 프로그램에서 동적으로 타입들을 알아내고 조작할 수 있게 되는 것이다. 예를 들어서 개발자가 프로그램에서 다른 Assembly에 들어 있는 타입의 정보를 얻을 수 있고, 그 속성에서 알아 본 것처럼 개발자가 직접 정보를 넣은 것도 얻을 수 있다. ILDASM 도구도 이러한 Reflection을 이용하는 것이고, Visual Studio.NET의 자동 완성 기능도 Reflection을 이용하는 것이다.

Reflection은 단지 타입에 관한 정보뿐만 아니라 해당 타입에 대한 객체를 런타임에 생성하고 객체의 메소드를 런타임에 호출할 수 있다.

CLR 로더는 Application Domain을 관리하는데, 각 어셈블리를 적절한 Application Domain으로 로드하고 각 Assembly내에서 형식 계층의 메모리 레이아웃을 제어하는 작업 등이 여기에 포함된다.

Assembly는 모듈을 포함하고 모듈은 형식을 포함하며 형식은 멤버를 포함한다. Reflection은 Assembly, 모듈, 형식을 캡슐화하는 개체를 제공한다. Reflection을 사용하면 형식 인스턴스를 동적으로 만들고 형식을 기존 개체에 바인딩하거나 기존 개체에서 형식을 가져올 수 있다. 그런 다음 형식의 메서드를 호출하거나 해당 필드 및 속성에 액세스할 수 있다.

Reflection은 일반적으로 다음과 같은 경우에 사용된다.

• Assembly를 사용하여 Assembly를 정의 및 로드하고, 어셈블리 Manifest에 나열된 모듈을 로드하고, 이 Assembly에서 형식을 찾아 형식 인스턴스를 만드는 경우

• Module을 사용하여 모듈 및 해당 모듈의 클래스를 포함하는 Assembly 같은 정보를 검색하는 경우. 또한 모듈에 정의된 모든 전역 메서드 또는 특정한 비 전역 메서드를 모두 가져올 수 있다.

• ConstructorInfo를 사용하여 생성자의 이름, 매개 변수, 액세스 한정자(예: **public** 또는 **private**) 및 구현 정보(예: **abstract** 또는 **virtual**) 등의 정보를 검색하는 경우. Type 개체의 Type.GetConstructors 또는 GetConstructor 메서드를 사용하여 특정 생성자를 호출하는 경우

• MethodInfo를 사용하여 메서드의 이름, 반환 형식, 매개 변수, 액세스 한정자(예: **public** 또는 **private**) 및 구현 정보(예: **abstract** 또는 **virtual**) 등의 정보를 검색하는 경우. Type 개체의 Type.GetMethods 또는 GetMethod 메서드를 사용하여 특정 메서드를 호출하는 경우

• FieldInfo를 사용하여 필드의 이름, 액세스 한정자(예: **public** 또는 **private**) 및 구현 정보(예: **static**) 등의 정보를 검색하고 필드 값을 가져오거나 설정하는 경우

• EventInfo를 사용하여 이벤트의 이름, 이벤트 처리기 데이터 형식, 사용자 지정 특성, 선언 형식 및 리플렉팅된 형식 등의 정보를 검색하고 이벤트 처리기를 추가하거나 제거하는 경우

• PropertyInfo를 사용하여 속성의 이름, 데이터 형식, 선언 형식, 리플렉팅된 형식 및 읽기 전용 또는 쓰기 가능 상태 등의 정보를 검색하고 속성 값을 가져오거나 설정하는 경우

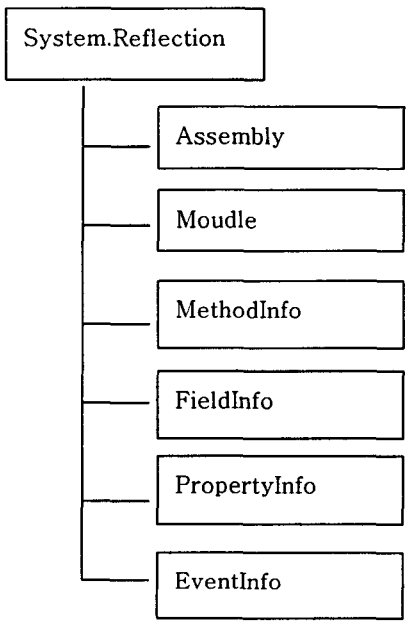
• ParameterInfo를 사용하여 매개 변수의 이

름, 데이터 형식, 입력 매개 변수인지 출력 매개 변수인지 여부 및 메서드 서명에서 매개 변수의 위치 등의 정보를 검색하는 경우

System.Reflection.Emit 네임스페이스의 클래스는 런타임에 형식을 빌드할 수 있는 특정 형식의 Reflection을 제공한다.

또한 Reflection을 사용하면 형식 브라우저라고 하는 응용 프로그램을 만들 수도 있는데 이를 통하여 사용자는 형식을 선택한 다음 이 형식에 대한 정보를 볼 수 있다.

다른 용도로 Reflection을 사용할 수도 있다. JScript 같은 언어의 컴파일러는 Reflection을 사용하여 기호 Table을 구성한다. System.Runtime.Serialization 네임스페이스의 클래스는 Reflection을 사용하여 데이터에 액세스하며 지속할 필드를 결정한다. System.Runtime.Remoting 네임스페이스의 클래스는 serialization을 통해 간접적으로 Reflection을 사용한다.



.NET의 System.Reflection 클래스 계층도 일부

5. CodeDOM

소스 코드를 출력하는 일부 응용 프로그램에서 런타임에 소스 코드를 여러 가지 언어로 생성하고 출력하도록 요청 할 수 있다. .NET Framework SDK에서는 렌더링할 코드를 나타내는 단일 모델을 기반으로 런타임에 소스 코드를 여러 가지 프로그래밍 언어로 출력할 수 있도록 하는 CodeDOM(Code Document Object Model)이라는 메커니즘을 제공한다. CodeDOM 요소로 Object 그래프를 Assemble하여 소스 코드 모델의 구조를 빌드하도록 프로그램을 만들 수 있다. 이 Object 그래프는 지원되는 프로그래밍 언어를 사용하여 소스 코드로 렌더링 될 수 있다. 또한, CodeDOM을 사용하면 외부 컴파일러를 사용하여 소스 코드를 이진 실행 파일로 컴파일할 수 있다.

CodeDOM은 일반적으로 다음과 같은 경우에 사용한다.

◦ 템플릿 기반 코드 생성: ASP.NET, XML 기반 웹 서비스의 클라이언트 Proxy, 코드 마법사, 디자이너 또는 코드를 내보내는 기타 메커니즘 등에 대한 코드를 생성한다.

◦ 동적 컴파일링: 코드를 한 가지 또는 여러 가지 언어로 컴파일한다.

CodeDOM은 프로그래밍 언어에 상관 없이, 소스 코드의 구조를 나타내는 데 사용되는 클래스, 인터페이스 및 아키텍처를 제공한다. 또한 CodeDOM에서 지원되는 프로그래밍 언어 중에서 하나를 런타임에 선택하여 코드를 해당 언어의 소스 코드로 출력할 수 있다. 사용하는 언어에 대해 CodeDOM을 개발하는 컴파일러 공급업체에서 지원 가능한 언어를 확장할 수 있다. CodeDOM은 또한 소스 코드를 외부 컴파일러를

사용하여 컴파일하는 작업을 도와주는 클래스를 제공한다.

소스 코드를 나타내기 위해 CodeDOM 요소는 서로 연결되어 CodeDOM 그래프 또는 CodeDOM 트리라고 하는 데이터 구조를 형성하며, 데이터 구조는 소스 코드 문서 또는 문서 세그먼트를 모델링한다. CodeCOM 요소를 사용하면 개체 그래프를 프로그래밍 방식으로 생성할 수 있다. 소스 코드를 생성하는 프로그램의 개발자는 소스 코드로 렌더링될 수 있는 CodeDOM 그래프를 생성할 수 있다. CodeDOM은 또한 CodeCOM에서 지원되는 프로그래밍 언어를 사용하는 어셈블리에 소스 코드를 컴파일하도록 도와주는 클래스를 제공한다. 예를 들어, ASP.NET에서는 CodeCOM을 사용하여, HTML 페이지나 컨트롤을 렌더링하는 어셈블리에 컴파일 할 개체 그래프를 만든다.

김 동 근

- 1997년 숭실대학교 행정학과 학사 졸업
- 현재 : (주)웹타임 교육센터, Microsoft Certified Trainer (MCT)
- 자격사항
  MCAD (Microsoft Certified Application Developer)
  MCSD (Microsoft Certified Solution Developer)
  MCDBA (Microsoft Certified Database Administrator)
  MCSE (Microsoft Certified System Engineer)
  MCT (Microsoft Certified Trainer)

김 태 석

- 1981년 경북대학교 전자공학과 졸업(공학사)
- 1989년 일본 KEIO대학 이공학부 계산기과학전공(공학석사)
- 1993년 일본 KEIO대학 이공학부 계산기과학전공(공학박사)
- 1993년 일본 국제전신전화연구소(KDD) 기술고문
- 1993년 일본 KEIO대학 이공학부 객원연구원
- 1994년~현재 동의대학교 소프트웨어공학과 교수
- 자격증 : 멀티미디어기술사, 인터넷시스템관리사(기술사)
- 저서 : 인터넷비즈니스, 자연언어처리, 자연언어이해 등 다수
- 관심분야 : 정보시스템, 기계번역, 인터넷비즈니스