cover story 6

더 세련된 객체지향 활용

AOP와 애플리케이션 다이어트

관점지향 프로그래밍(Aspect Oriented Programming)은 디자인 패턴이나 객체지향적인 소프트웨어 아키텍처로 해결할 수 없는 유연함을 제공하며 이런 장점을 잘 지원하는 AspectJ, SpringAOP 등의 관점지향 프로그래밍 프레임워크를 다수 보유하고 있다. 하지만 관점지향 프로그래밍은 여전히 닷넷 개발자들에게는 낯선 영역인 게 사실. 사용하기 간편하고 무엇보다 비주얼 스튜디오와 연동되어 쉽게 접할 수 있는 Aspect.NET을 통해 닷넷에서의 관점지향 프로그래밍을 활용해 보자.

김용현 drvoss@gmail.com, www.YHKim.com | 이스트소프트에서 알약 개발팀 팀장을 맡고 있으며 Software Architecture와 Design Pattern에 관심이 많다. 어렵게 배운 지식을 쉽게 알려주는 방법을 고민하고 있으며, 최근에는 받은 것을 베풀기 위해 서울과 부산에서 Design Pattern 스터디그룹의 멘토링을 진행하고 있다.

생산성은 어느 한 순간 급격하게 증가하는 형태를 보인다. 프로그래밍 분야에서도 마찬가지인데, 함수가 등장하면서 코드를 재사용할 수 있게 됐고 객체지향 프로그래밍이 등장하면서 자신의 상태를 저장할 수 있는 객체 재사용으로 인해 다시 한 번 생산성이 급격히 증가했다. 객체지향 프로그래밍의 최대 장점은 함수나 객체로 만들어진 신뢰할 만한 코드를 수정 없이 다시 한 번 활용하는 것이고, 재활용의 빈도가 높으면 높을수록 프로그래밍 분야의 생산성은 급격히 높아진다.

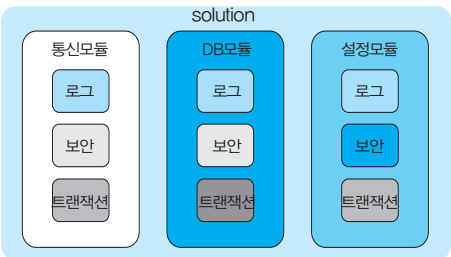
하지만 생각과 달리, 객체를 재사용할 수 있는 환경이 만들어진 오늘날의 C++나 C# 코드에서 객체를 그대로 재사용하는 빈도는 그렇게 높지 않다. 비즈니스 애플리케이션을 만들 때 처음에는 코드를 재사용할 수 있도록 구조를 잘 잡아서 만들다가도 각종 고객의 요청사항이나 디버깅 로직들이 들어가게 되면 점점 그대로 재사용할 수 없는 코드로 변질되기 시작한다.

관점의 분리

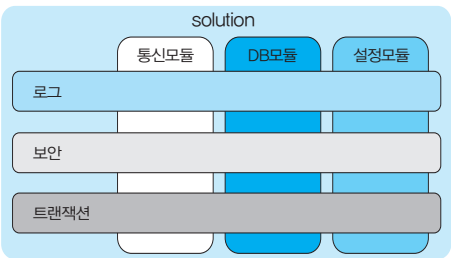
코드의 유연한 재사용이 가능하려면 코드가 지향하는 한 가지 목적을 가지고 목적에 맞는 코드만으로 구성되어야 한다. 하지만 비즈니스 애플리케이션에서는 순수한 클래스 본연의 기능만 들어가기가 어렵다. 프로젝트가 진행되면 진행될수록 도메인에 특화된 보안 정책이나 로그 정책, 혹은 디버깅을 위한 다른 로직들이 클래스 본연의 기능에 하드 코딩되면서 해당 클래스를 공통으

로 사용하기 어려워지기 때문이다. 〈그림 1〉의 경우 통신 모듈에서 로그 기능, 보안 기능, 트랜잭션 기능을 분리할 수 있다면 통신 모듈 자체의 기능에 집중할 수 있고 통신 모듈을 다른 솔루션에서 재활용하기도 수월할 것이다.

관점지향 프로그래밍은 이러한 통신 모듈의 본연의 기능인 핵심관심 사항(Core Concern)과 클래스의 재활용에 방해가 되었던 로그, 보안, 트랜잭션 같은 기능을 〈그림 2〉와 같이 공통관심 사항(Cross Cutting Concern)으로 분리하는 것이다. 핵심관심 사항과 공통관심 사항이 서로 분리되어 있으므로 공통 관심사항이나 핵심 관심사항의 유지보수가 다른 쪽에 영향을 미치지 않으며, 공통관심 사항을 쉽게 추가하거나 뺄 수 있고 공통관심 사항에 선택적으로 적용시킬 수 있다.



〈그림 1〉 담당 팀별 공통관심 사항 모듈이 제각각인 경우



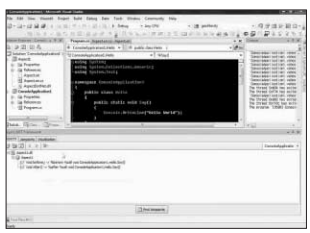
〈그림 2〉 공통관심 사항과 핵심관심 사항이 분리되어 있는 형태

관점지향 프로그래밍은 클래스를 구조화하거나 상속하는 등 일반적인 기존의 객체지향 프로그래밍으로는 구현할 수 없는 공통관심 사항의 유연한 삽입을 가능하게 한다. 이러한 기능의 유연한 삽입은 보안, 안전한 멀티스레드 코드, 특히 디버깅 등의 목적으로 로그 남기기 등에 탁월한 능력을 보여준다.

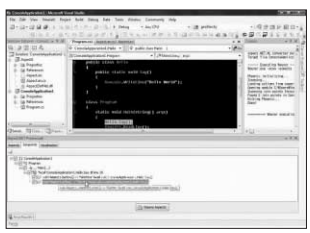
Aspect.NET

풍부한 프레임워크 지원으로 관점지향 프로그래밍은 닷넷 진영보다는 자바 진영에서 주목받고 있다. 닷넷에서도 관점지향 프로그래밍을 지원하는 여러 가지 프레임워크가 있지만 사용법이 번거롭다 보니 실제 환경에서 접하기가 어려운 실정이다. Aspect.NET의 경우, 마이크로소프트의 아카데미 프로그램에서 후원하는 프레임워크이면서 비주얼 스튜디오와의 연동을 통해 닷넷 개발자들이 쉽고 편리하게 사용할 수 있는 장점이 있다. 특히 Aspect.NET의 애스팩트(Aspect)는 닷넷 어셈블리에서 수행되므로, 다양한 닷넷 디버거나 ildasm 등과 같은 툴에서 함께 이용할 수 있다. 현재 Aspect.NET은 2.1 버전이 최신이며 차기 버전에서는 닷넷에서 동작하는 다양한 다른 언어들도 지원할 예정이다. Aspect.NET은 참고문서에 기술된 URL에서 다운로드할 수 있으며 비주얼 스튜디오 2005와 피닉스 RDK가 설치되어 있어야 사용할 수 있다. 만일 VisualAssist와 같은 비주얼 스튜디오에서 동작하는 몇몇 에드인들과 함께 사용하면 비정상적으로 동작하거나 아예 동작하지 않을 수 있다.

애스팩트가 애플리케이션에 새로운 기능을 삽입하는 것은 위빙(Weaving) 동작을 통해 이뤄진다. 문서에 따라 위빙과 크로스커팅이 혼재되어 사용된다. 핵심관심 사항에 공통관심 사항을 삽입하는 것을 위빙이라고 하고, 공통관심 사항 입장에서는 크로스커팅이라고 하므로 두 단어는 같은 동작을 의미한다. 위빙 과정은 프레임워크의 특성에 따라 컴파일 시간, 수행 시간, 클래스 로딩시에 위빙규칙에 기술된 위빙이 이뤄질 상황(Condition)에 맞춰 진행된다.



〈화면 1〉 Aspect.NET 모듈의 로딩



〈화면 2〉 Aspect.NET 모듈의 스캐닝

위빙규칙을 통해 애플리케이션에 새로운 기능을 삽입할 수 있는 조인포인트(JoinPoint)를 찾는다. 〈화면 1〉에서 Find Jointpoint 버튼을 누르면 Aspect.NET은 스캐닝(Scanning)을 통해 기능을 삽입할 수 있는 조인포인트들을 찾아 조인포인트 탭으로 넘겨준다. 〈화면 2〉의 조인포인트 탭에서 프로그래머는 실제로 기능이 삽입될 포인트컷(Pointcut)을 확인할 수 있다. 목록을 더블 클릭하면 실제 소스 중 위빙이 될 소스가 화면에 나타나고 위빙에 영향을 받을 포인트컷이 하이라이트된다. 다른 애스팩트 툴들과 달리 Aspect.NET은 조인포인트 중 실제로 기능을 삽입할 위치의 포인트컷 결정을 GUI 형태로 툴에서 직접하도록 지원한다. 프로그래머가 포인트컷을 결정하고 나면 Weave Aspects 버튼을 통해 애플리케이션에 위빙이 이뤄진다.

참고자료

1. Aspect.NET 2.1
   http://www.academicresourcecenter.net/curriculum/pfv.aspx?ID=6801
2. Microsoft Research - Phoenix Academic Program
   http://msdn.microsoft.com/ko-kr/library/eye126ky.aspx
3. Using Aspect-Oriented Programming for Trustworthy Software Development - Wiley