05 AOP 기본개념

AOP 개요

- OOP의 문제점
  - 코드가 중복됨
  - 코드가 지저분해짐
  - 생산성이 저하됨
  - 재활용성이 저하됨
  - 변화가 어려움
- AOP의 개념
  - 객체지향 프로그래밍
  - 로그 처리, 보안, 트랜잭션 관리 그리고 예외사항 처리 등의 코드를 단일 모듈로 각각 작성하고 필요한 시점에 핵심코드를 삽입하여 동작하게 하는 것

- AOP의 목적
  - OOP와 같은 모듈화가 뛰어난 방법을 사용하더라도 결코 쉽게 분리된 모듈로 작성하기 힘든 요구사항이 실제 어플리케이션 설계와 개발에서 자주 발견됨
    → AOP에서는 이를 횡단 관심이라 함

횡단 관심(Crosscutting Concerns)	VS	핵심 관심(Core Concerns)
▪ 결코 쉽게 분리된 모듈로 작성하기 힘든 요구사항 ▪ 객체지향의 기본 원칙을 지키면서 이를 분리해서 모듈화하는 것이 매우 어려움		▪ 해당 시스템의 핵심 가치와 목적이 그대로 드러난 관심 영역 ▪ 기존의 객체지향 분석/설계(OOAD)를 통해 쉽게 모듈화와 추상화가 가능함

AOP 기술 및 용어

- AspectJ의 AOP 기능
  - Aspect는 AspectJ의 특별한 컴파일러를 통해 자바 VM에서 사용될 수 있는 코드로 만들어짐 → Weaving 작업 → 핵심 관심 모듈의 사이사이에 Aspect 형태로 만들어진 횡단 관심 코드들이 삽입되어 Aspect가 적용된 최종 바이너리가 만들어짐

- AOP의 용어

Joinpoint	횡단 관심 모듈의 기능이 삽입되어 동작할 수 있는 실행 가능한 특정위치
Pointcut	어떤 클래스의 어느 Joinpoint를 사용할 것인지를 결정하는 선택 기능
Advice	각 Joinpoint에 삽입되어 동작할 수 있는 코드
Weaving	- Pointcut에 의해서 결정된 조인포인트에 지정된 Advice를 삽입하는 과정 - AOP가 기존의 핵심 관심 모듈의 코드에 전혀 영향을 주지 않으면서 필요한 횡단 관심 기능을 추가할 수 있게 해주는 핵심적인 처리 과정
Introduction	기존의 클래스와 인터페이스에 필요한 메소드나 필드를 추가해서 사용할 수 있게 해주는 방법
Aspect	Pointcut(어디에서)과 Advice(무엇을 할 것인지)를 합쳐놓은 것

SpringAOP의 변화

- Spring 1.x AOP를 사용할 때의 한계와 단점
  - Pointcut을 정의하는 것이 어려움
  - XML 설정파일이 매우 복잡함
  - 다양한 자바 클래스에 대한 Aspect를 적용할 수 없음
  - Proxy 기반의 AOP는 약간의 오버헤드를 가지고 있음
  - Proxy 기반의 AOP는 Proxy와 타깃오브젝트가 분리되어 있음
- Spring 2.x AOP
  - Spring 2.0은 Spring 1.x의 코드베이스를 그대로 유지한 채로 많은 부분의 기능을 추가, 개선한 방식을 택함
  - 1.x와 마찬가지로 Proxy 기반의 AOP를 사용함
  - Spring 2.0 AOP는 가장 발달한 Pointcut 언어를 가지고 있는 AspectJ를 그대로 이용해서 Pointcut 언어로 사용함
- @AspectJ Annotation을 이용한 방법
  - @AspectJ Annotation을 이용한 Aspect, Pointcut과 Advice의 정의는 AspectJ 5에 처음 소개된 기능으로 SpringAOP는 이것을 그대로 사용할 수 있음
  - @AspectJ를 사용하려면 먼저 Spring 설정파일에 AspectJ autoproxy 설정을 해야 함
- Schema(schema) 기반의 AOP 설정을 이용하는 방법
  - Spring 설정파일 안에 Pointcut 정의하는 것이 가능함
  - Advice는 그대로 자바클래스로 만들어서 사용 가능하며 이 때 Spring 설정용 XML은 반드시 Schema 방식이어야 함

SpringAOP의 특징

- SpringAOP의 특징

1. 표준 자바 클래스로 작성됨
2. Runtime 시점에서 Advice 적용됨
3. AOP 연맹의 표준을 준수함
4. 메소드 단위 Joinpoint만을 제공함

Schema-based AOP 설정 기본

- XML Schema
  - XML 문서에서 사용할 수 있는 엘리먼트와 속성의 종류를 정의하기 위해 사용됨

- 한 개의 XML 문서를 여러 가지 Schema 문서를 혼용해서 작성할 수도 있음

- 네임스페이스
  - 엘리먼트의 이름 충돌을 피하는 방법으로 사용됨
  - 2개 이상의 XML Schema를 하나의 XML 문서에서 이용하고자 하는 경우에 사용됨

- 네임스페이스 설정 방법
  - Spring Schema 설정 방식에서는 네임스페이스를 이용하여 태그를 확장할 수 있음
  - Schema 방식을 이용해서 Aspect를 정의할 때는 http://www.springframework.org/schema/aop 네임스페이스를 사용함