VO, DAO, DTO, CRUD란 무엇인가?

진탄
2015. 8. 26. 11:51

VO(Value Object)

개념

Value Object는 DTO와 동일한 개념이나 차이점은 read only 속성을 갖는다.
Value Object는 관계 데이터 베이스의 레코드에 대응되는 자바 클래스이다.
형태는 db 레코드를 구성하는 필드들을 Value Object의 Attribute로 하고 해당 변수에 접근할 수 있는 Getter, Setter 메소드의 조합으로 클래스가 형성되어진 클래스이다.
특성은 대체로 불변성이고 equlas()로 비교할 때 객체의 모든 값을 비교해야 한다.

프로그램의 사용자가 화면에서 어떤 데이터를 입력하거나 조회 요청이 왔을 때 입력된 데이터나 조회하는 조건을 VO에 담아서 DAO에 요청하면 DAO는 저장소(일반적으로 Database)로부터 데이터를 입력하거나 조회한 후 그 결과를 돌려주게 된다.

VO는 간단한 독립체(Entity)를 의미하는 작은 객체를 의미한다.
VO의 같음은 그 정체성에 의해 결정되지 않는데, 그 뜻은 두 VO 들은 그 두 가지가 같은 값을 갖고 있을 때 같은 것이지 같은 객체라고 해서 같지 않다는 것이다.

작기 때문에, 같은 독립체를 대변하는 복수의 같은 VO들이 존재할 수 있다. 가끔 하나의 인스턴스에 의존하고 그에 기반한 레퍼런스를 사용하기 보다는 새 객체를 생성하는 것이 더 간편하다.

필요성

Network traffic을 줄임으로 인해 효과적이다.

기대효과

Network traffic이 줄어든다.

장단점

장점은 비 서버 측 클라이언트도 네트워크 오버헤드 없이 영속성 데이터에 액세스 할 수 있다는 점이다.
데티어 전달을 위해 가장 효율적인 방법이지만, 클래스의 선언을 위해 많은 코드가 필요하다.
즉 파일수가 많아지게 되고 관리도 힘들어진다.

DTO(Data Transfer Object : 데이터 전송 객체)

개념

데이터가 포함된 객체를 한 시스템에서 다른 시스템으로 전달하는 작업을 처리하는 개체이다.
Data에 접속하는 객체이다. 여기서 Data란 일반적인 Database도 될 수 있고, 파일도 될 수 있으며, 메모리도 될 수 있고, 기타 다른 저장소도 될 수 있다.

DTO는 프로세스 사이에서 데이터를 전송하는 객체를 의미한다. 이것은 이용하는 이유는 프로세스 간의 커뮤니케이션이 주로 개별 호출이 부담스러운 작업일 경우가 많은 원격 인터페이스(예:웹 서비스)에 의해 이루어지기 때문이다.
대부분의 개별 호출이 클라이언트와 서버 간의 왕복 시간을 소모하기 때문에, 호출 횟수를 줄이는 방법 중 하나는 몇 번의 호출에 의해 전송될 데이터를 모으는 DTO를 이용해서 한번만 호출하게 하는 것이기 때문이다.

VO와 DTO의 비교

DTO의 나머지 속성은 VO와 똑같다고 생각된다.
Core J2EE Patterns라는 책에서는 Value Object와 Transfer Object를 동일한 뜻으로 사용하지만, 반대로 Martin Fowler는 저서 Patterns of Enterprise Applicatino Architecture에서 약간 다른 의미로 이야기 한다.
DTO는 메소드 호출 횟수를 줄이기 위해 데이터를 담고 있는 녀석으로, VO는 값이 같으면 동일 오브젝트라고 볼 수 있는 것으로 표현하고 있다.

DTO a = new DTO(1);
DTO b = new DTO(1);

이라고 했을 때, a != b 이지만,

VO a = VO(1);
VO b = VO(1);

이라고 했을 때는 a == b 라고 정의하는 형태이다.

대부분의 검색에서 사람들은 VO와 DTO를 같은 개념으로 이야기 하고 있어서, 아직도 VO와 DTO가 "이런거다"라기 보다 거의 똑같은 개념으로 생각하고 있다.

DAO(Data Access Object : 데이터 접근 객체)

개념

데이터 접근을 목적하는 객체이다. 커넥션 같은 것을 하나만 두고 여러 사용자가 DAO의 인터페이스를 사용하여 필요한 자료에 접근 하도록 하는 것이 DAO의 개념이다.
DB에 대한 insert, update, delete, select를 처리한다.

DAO는 특정 타입의 데이터베이스나 다른 지속적인 메커니즘(Persistence Mechanism)에 추상 인터페이스를 제공하는 객체이다. 어플리케이션 호출을 데이터 저장 부분(Persistence Layer)에 매핑함으로써 DAO는 데이터베이스의 세부 내용을 노출하지 않고 특정 데이터 조작 기능을 제공한다. 이 고립성은 단일 책임 원칙(Single Responsibilty Principle)에 기반한다.

이 패턴은 대부분의 프로그래밍 언어에, 영속성이 필요한 대부분의 소프트웨어에, 대부분의 데이터베이스에 적용될 수 있다.

단일 책임 원칙 : 객체 지향 프로그래밍에서 모든 컨텍스트(클래스, 기능, 변수 등)은 하나의 책임만 가져야 한다는 것이며, 이는 컨텍스트에 의해 완전히 캡슐화 되어야 한다는 것이다. 그리고 모든 서비스들은 해당 책임에 맞춰 조정되어야 한다.
다른 말로는 "클래스를 수정해야 할 이유는 오직 하나여야 한다"고 한다.

필요성

모든 데이터베이스에 공통적으로 접속 할 수 있는 ODBC가 나왔지만 여전히 로우 레벨의 API를 포함하고 있었기 때문에 개발 장벽이 높았다.
이런 이유 때문에 개발자들은 정작 데이터베이스에 들어 있는 데이터를 어떻게 이용할지에 초점을 맞추기 보다, 어떻게 데이터베이스에 접속해서 데이터베이스와 교류하는지에 더 초점을 기울였다.
즉, 데이터를 활용하는 논리적 고민보다 기술적 고민에 더 많은 신경을 썼었다.
이런 이유로 DAO란 대안이 나오게 되었다.

기대 효과

사용자는 자신이 필요한 interface를 DAO에게 던지고 DAO는 이 인터페이스를 구현한 객체를 사용자에게 편리하게 사용 할 수 있도록 반환해준다.

장단점

DB에 대한 접근을 DAO가 담당하도록 하여 데이터베이스 액세스를 DAO에서만 하게 되면 다수의 원격 호출을 통한 오버헤더를 VO나 DTO를 통해 줄일 수 있고, 다수의 DB 호출 문제를 해결할 수 있다. 또한 단순히 읽기만 하는 연산이므로 트랜젝션 간의 오버헤드를 감소할 수 있다.
그러나 Persistent Storage를 너무 밀접하게 결합해서 작성을 하게 되면 Persistent Storage를 다시 작성할 경우가 생기는데 이러한 경우 유지 보수의 문제가 생길 수도 있다.

CRUD

CREATE(INSERT), READ(SELECT), UPDATE, DELETE