CQS(Command Query Separation)

개요 : 디자인 패턴

- 소프트웨어 구조를 선택할 땐 많은 조건을 고려하여 선택

- 시스템의 복잡성 , 사용성 , 확장성 , 성능 등등이 고려됨

- 여러 조건들을 따지며 현재까지 많은 사람들이 고민하고 개발한 결과로 디자인 패턴 등장

CQS(Command Query Separation) Pattern

- 소프트웨어 디자인 패턴 중 하나로, 모든 객체의 메서드를 두 가지로 분류하는 패턴임

- ① 작업을 수행하는 command 메서드(쓰기) , ② 데이터를 반환하는 query 메서드(조회)

- 서로 완벽히 분리되어서 command 이면서 query일 수 없고, command와 query 둘 중 하나에만 속해야됨

Command

- 객체의 내부 상태를 바꾸지만 값을 반환하지는 않음(즉 조회 기능은 없음)

- 예를 들어 설정자(setter)가 바로 command임

➔ 값을 변경하지만, 조회하지는 않으므로

Query

- 객체 내부 상태를 바꾸지 않고 객체의 값만 반환(즉 변경 기능은 없음)

- 예를 들어 접근자(getter)이 바로 query임

➔ 값을 조회만 하고, 변경하진 않으므로

CQS 패턴 위배 코드

```
public String setAndGetName(String name){
    this.name = name;
    return name;
}
```

- 한 메서드에서 변경과 조회 둘 다 허용하니 CQS 패턴도 위배했을 뿐더러 SOLID 원칙도 위배됨

- SOLID가 위배되면 예상치 못한 side effect가 발생하므로 CQS 패턴을 지킨다면 SOLID 원칙에 위배되지 않게 코드를 작성할 수 있음

장점

- 읽기와 쓰기를 명확하게 분리할 수 있음

- 즉 읽기와 쓰기가 동시에 일어나지 않으므로 성능을 최적화하는데 도움을 줄 수 있음

- 또한 간편하게 구현할 수 있으며, 읽기 편함

➔ 설정은 설정 코드만 , 조회는 조회 코드만 구현하면 되므로 ! 다른 기능 생각할 거 없이

CQS 예외 case : Stack의 pop()

- Stack의 pop은 가장 최근에 push 된 값을 반환하고 static에서는 제거하니 CQS가 위배됨

➔ command + query

- 마틴 파울러는 할 수 있는 한 CQS 원칙을 지키려 했지만, stack의 pop과 같이 매우 유

용한 상황에서는 얼마든지 CQS 원칙을 깨뜨릴 준비가 되어있음

- 이를 토대로 어떤 패턴이든 필연적으로 trade off가 있으므로 유연한 사고로 설계하고 구현하자

JPA 응용

저장

```
public Long save(Member member){
    em.persist(member);
}
```

상태 변경(Command) 및 id의 조회(Query)를 하므로 필요에 의한 CQS 원칙을 깬 것으로 볼 수 있음

수정

```
public void update(Long memberId , MemberSaveDto member){
    Member findMember = em.find(Member.class , memberId);
    findMember.setUsername(member.getUsername());
    findMember.setAge(member.getAge());
}
```

Command : 상태 변경만 되고 조회는 일어나지 않음

조회

```
public Member findMemberById(Long memberId){
    return em.find(Member.class , memberId);
}
```

Query : 조회만 되고 상태 변경은 되지 않음

실무

- 개발 전반 기본 개념이라고 생각하고 깔고 가자

- 특정 메서드를 호출했을 때 내부에서 변경(Side Effect)이 일어나는 메서드인지, 아니면 내부에서 변경이 전혀 일어나지 않는 메서드인지 명확히 분리

- 그러면 만약 데이터 변경 관련 이슈가 발생했을 때 변경이 일어나는 메서드만 찾아보면 됨 , 대부분 크리티컬한 이슈들은 데이터를 변경하는 곳에서 발생하므로 ㅇㅇ

- 실무에선 insert는 id만 반환하고(아무것도 반환 안 하면 조회가 안 되니) , update는 void , 조회는 내부의 변경이 없는 메서드로 설계하면 좋음

- 결론 : 변경 메서드(command)와 조회 메서드(query)를 분리하여 선택과 집중을 하게 하여 데이터 변경 관련 이슈가 발생하였을 때 변경 메서드만 유지보수하며 성능 상 이점 가져갈 수 있음

프로그래밍에서의 Side effect

- 부작용이라고 하면 뭔가 꺼림직하겠지만 프로그래밍 관점에선 꼭 부정적이진 않음

- 사이드 이펙트 : 데이터의 변경을 뜻함

- 즉 변경이 일어나지 않으면 사이드 이펙트가 발생하지 않는다

※ 참조

CQS

https://medium.com/@su_bak/cqs-command-query-separation-pattern-%E1%84%8B%E1%85%B5%E1%84%85%E1%85%A1%E1%86%AB-f701eabf8754

https://hardlearner.tistory.com/383

Side Effect

https://hyeonk-lab.tistory.com/43