# 최근에 느낀 DTO와 관련한 이야기

지난 주 목요일 프로그래머로 산다는 것의 출판 기념 회식에 초대 받아 참석하게 되었다. 다른 분들은 안면식이 있는 분들인데 김성박 님은 온라인으로만 뵙다가 오프라인에서 뵈니까 넘 반가웠다. 첫 만남이 사람을 참 편안하게 만드셔서 쉽게 어울릴 수 있었다.

술잔이 몇 잔 오간 후에 김성박 님이 갑작스럽게 몇 일 전에 있었던 DTO와 관련한 이야기를 했다. 먼저 DTO가 무엇인지 모르는 사람들을 위해 간단히 소개하면 다음과 같다. DTO는 Data Transfer Object의 약자로 각 레이어 사이에 데이터를 전달하기 위한 목적으로 사용하는 Object라고 생각하면 된다. 일반적으로 전달하는 데이터에 대한 속성과 set, get 메소드로 구성되어 있다.

김성박 님의 이야기는 이러했다. 최근에 데이터베이스와 매핑되어 있는 Persistent Object를 JSON으로 변환할 필요가 있었다. 그런데 Spring MVC를 활용해 Persistent Object를 JSON으로 자동 변환할 경우 Persistent Object에 존재하는 모든 속성과 Relation 관계에 있는 객체의 속성까지 JSON으로 자동 변환되는 이슈가 있다. 자동 변환되는 것이 좋은 점도 있다. 하지만 객체가 Bi-Direction 관계일 경우 무한 루프에 빠지는 경우도 있고, 불필요한 속성까지 JSON으로 변환되기 때문에 불만족스러운 경우가 종종 있다. 이 같은 이슈를 어떻게 해결할지 고민하다가 Facebook을 통해 백기선씨에게 질문을 했고 이에 대한 해결 방법을 오프라인 모임에서 나누었다는 것이다. 이 자리에 이일민씨도 참석해 있었는데 지금까지 자신이 생각하지 못했던 해결책을 제시해 주어 너무 좋았다는 경험담이었다.

그 자리에서 나눈 이야기는 Persistent Object에서 JSON으로 변환할 필요가 있는 데이터를 뽑아 DTO로 만드는 것이 좋겠다는 것이었다. DTO를 JSP와 같은 하나의 View로 생각하면 쉽게 해결책을 찾을 수 있다는 것이다. 이에 덧붙여 Layer 간에 데이터를 전달할 때는 DTO 개념을 사용하는 것이 바람직하다는 이야기까지 나누었다고 한다.

위 이야기를 들으면 과거의 악몽이 되살아 난다고 치를 떠는 개발자가 있을 수 있다. Persistent Object를 DTO로 변환하는 것은 나름 수긍할 수 있겠지만, Layer 사이에 DTO를 사용해야 한다고 이야기하면 수긍하지 못하는 개발자가 많을 것이다. 과거에 DTO 개념을 적용하기 위해 비슷한 속성을 가지는 객체를 만들어본 개발자라면 그럴 수 있을 것이다. 이 같은 폐해를 없애기 위해 최근의 개발 흐름은 Object 하나를 만들어 DTO와 Persistent Object 역할을 담당하도록 구현하고 있다. 나 또한 이 같은 방식으로 구현하고 있다. Object 하나가 DTO와 Persistent Object 역할을 담당하는 개발 생산성이나 유지보수 측면에도 좋은 선택이라고 생각한다. 대부분의 경우 이 같은 방식으로 구현해도 무리가 없을 듯 하다.

그런데 최근에 SLiPP QnA 버전(이 사이트에 QnA 기능을 구현하고 있다. 조만간 오픈할 계획이다.)을 구현하다가 느끼는 바가 있어 공유하려고 한다. 처음에는 요구사항도 정말 단순해서 Object 하나에 DTO와 Persistent Object 역할까지 담당하도록 구현하는데 큰 무리가 없었다. 시간이 지나면서 요구사항도 증가하고 로직도 증가했다. 로직이 증가하면서 이 로직을 가능하면 Rich Domain Object로 구성하려는 시도를 했다. 이렇게 조금씩 조금씩 로직을 Object로 구현하다보니 물리적인 Object는 하나이지만 논리적으로는 DTO와 Persistent Object 역할을 하도록 구분해야 겠다는 생각에 다다랐다. 이 같은 생각으로 구현한 소스 코드는 다음과 같다.

```
public static Question newQuestion(SocialUser loginUser, Question
questionDto, TagRepository tagRepository) {
  Question newQuestion = new Question();
  newQuestion.writer = loginUser;
  newQuestion.title = questionDto.title;
  newQuestion.contentsHolder = questionDto.contentsHolder;
  newQuestion.processTags(questionDto.plainTags, tagRepository);

  return newQuestion;
}
private void processTags(String plainTags, TagRepository tagRepository)
{
  TagProcessor tagProcessor = new TagProcessor(tagRepository);
  tagProcessor.processTags(this.tags, plainTags);
  this.tags = tagProcessor.getTags();
  this.denormalizedTags = tagProcessor.getDenormalizedTags();
  this.newTags = tagProcessor.getNewTags();
}
```

사용자가 입력한 질문 데이터는 DTO 역할을 담당하는 Question Object를 통해 전달된다. 각각의 역할을 명확히 구분하기 위해 변수명까지 questionDto로 구현했다. Persistent Object 역할을 담당하는 Question Object는 Question DTO에서 전달되는 데이터 이외에도 로그인한 사용자 정보와 사용자가 입력한 태그 정보를 파싱해 데이터베이스와 매핑할 수 있도록 재구성해야 한다. 이 작업을 newQuestion() 메소드에서 구현하고 있다. newQuestion() 메소드는 Persistent Object 역할을 담당하는 새로운 Question Object를 만들고 필요한 데이터를 전달하도록 구현하고 있다. 이와 같이 DTO 역할과 Persistent Object 역할을 담당하는 인스턴스를 명확하게 구분해서 구현했다. 이 단계까지 진행하고 보니 DTO 역할과 Persistent Object 역할을 담당하는 QuestionDto와 Question 클래스를 각각 만들어도 좋겠다는 생각이 들었다. 아직까지는 분리하지 않아도 복잡도가 그리 높지 않아서 하나의 클래스로 감당할 수 있지만 복잡도가 증가한다면 분리하는 것도 좋은 방법이라고 생각한다.

위와 같이 Question 클래스를 구현하면 Service는 다음과 같이 클래스간의 관계를 연결, 상태 저장, Transaction 처리에만 집중하도록 구현했다.

```
@Service("qnaService")
@Transactional
public class QnaService {
  public void createQuestion(SocialUser loginUser, Question questionDto)
{
    Question newQuestion = Question.newQuestion(loginUser, questionDto,
tagRepository);
    Question savedQuestion = questionRepository.save(newQuestion);
    tagService.saveNewTag(loginUser, savedQuestion,
newQuestion.getNewTags());
  }
  ...
}
```

내가 이 글을 통해 이야기하고 싶은 부분은 효율성을 위해 DTO와 Persistent Object를 하나의 클래스로 구현하고 있지만 논리적으로는 각각의 역할을 명확히 구분해서 사용하는 것이 좋겠다는 것이다. 이를 구분하려면 각각의 역할에 해당하는 인스턴스를 별도로 생성한다면 좀 더 명확해 지지 않을까라는 생각이다. EJB가 뜨거운 감자였을 때는 DTO가 정말 중요한 요소로 자리 잡았는데 Spring 프레임워크가 주도하면서 이에 대한 필요성과 개념을 너무 간과하는 것은 아닌가라는 생각이 든다. 논리적으로 Layer만 분리되어 있기 때문에 하나의 클래스를 통해 DTO와 Persistent Object 역할을 같이 담당하도록 구현하고 있지만 각각의 역할을 구분해서 구현하는 연습을 하는 것이 좋겠다. 특히 Rich Domain Object를 지향한다면 업무 복잡도가 증가하는 시점에는 DTO와 Persistent Object를 클래스로도 분리하는 것이 바람직할 것으로 판단된다.