라이브 세션-서비스 계층-2022.08.23(화)

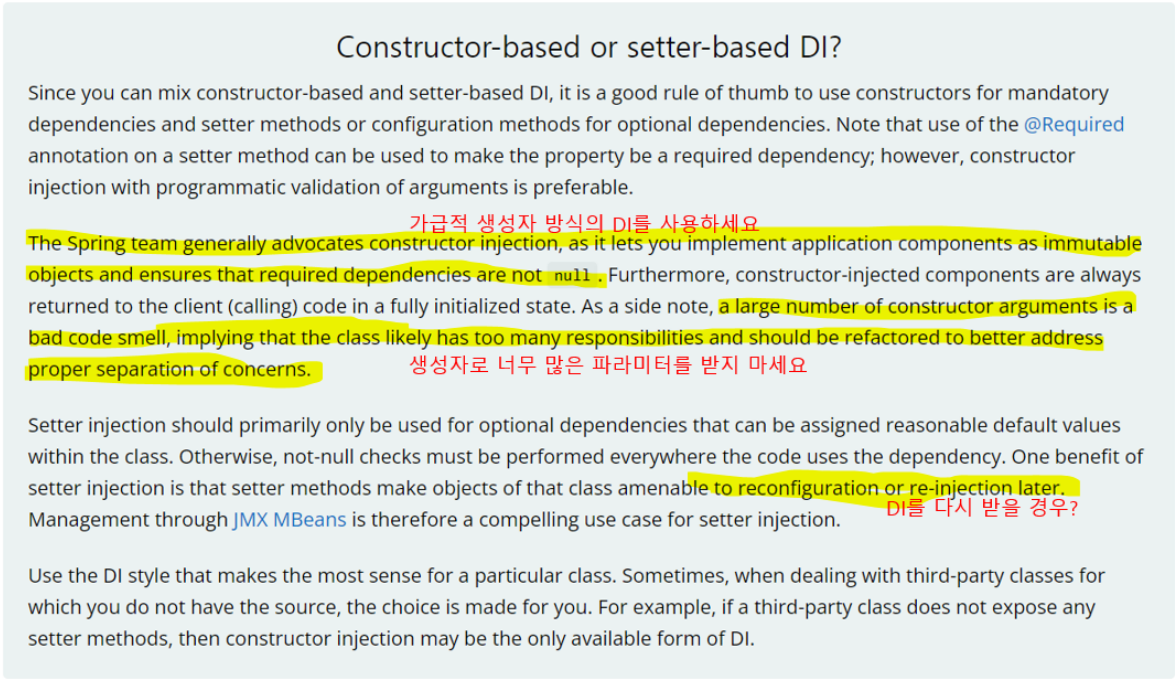
Controller와 DTO 초간단 핵심 리뷰

코드로 설명

DI를 통한 Controller와 Service의 연동 방법

- 서비스 클래스를 Spring Bean에 등록하는 방법
  - @Service
  - @Component

- DI 방법
  - 생성자 기반 DI
    - 생성자가 하나일 경우 @Autowired를 붙일 필요가 없음
    - 생성자가 여러개일 경우 @Autowired 필수
    - 가급적 final 키워드를 사용하자
  - setter 기반 DI
    - setter를 통해 DI 되는 객체를 바꿔야 될 경우
      - 일반적으로 흔하지 않다.



Constructor-based or setter-based DI?

Since you can mix constructor-based and setter-based DI, it is a good rule of thumb to use constructors for mandatory dependencies and setter methods or configuration methods for optional dependencies. Note that use of the @Required annotation on a setter method can be used to make the property be a required dependency; however, constructor injection with programmatic validation of arguments is preferable.

가급적 생성자 방식의 DI를 사용하세요

The Spring team generally advocates constructor injection, as it lets you implement application components as immutable objects and ensures that required dependencies are not `null`. Furthermore, constructor-injected components are always returned to the client (calling) code in a fully initialized state. As a side note, a large number of constructor arguments is a bad code smell, implying that the class likely has too many responsibilities and should be refactored to better address proper separation of concerns.

생성자로 너무 많은 파라미터를 받지 마세요

Setter injection should primarily only be used for optional dependencies that can be assigned reasonable default values within the class. Otherwise, not-null checks must be performed everywhere the code uses the dependency. One benefit of setter injection is that setter methods make objects of that class amenable to reconfiguration or re-injection later. Management through JMX MBeans is therefore a compelling use case for setter injection.

DI를 다시 받을 경우?

Use the DI style that makes the most sense for a particular class. Sometimes, when dealing with third-party classes for which you do not have the source, the choice is made for you. For example, if a third-party class does not expose any setter methods, then constructor injection may be the only available form of DI.

Mapper란?

DTO 클래스와 엔티티(Entity) 클래스를 서로 변환해주는 변환자

Mapper를 사용하는 이유

계층간 역할 분리

DTO 클래스와 Entity 클래스의 역할 분리 이유

- **계층별 관심사의 분리**
- **코드 구성의 단순화**
- **REST API 스펙의 독립성 확보**
  - 패스워드 같은 원하지 않는 데이터 제외 가능
  - API 문서로써의 역할

    - 유지보수 용이

Mapper 종류

- https://github.com/arey/java-object-mapper-benchmark
- https://www.baeldung.com/java-performance-mapping-frameworks

Mapstruct vs ModelMapper

- ModelMapper
    - Runtime에 리플렉션으로 매핑 진행
- Mapstruct
    - 컴파일 타임에 매핑 구현체 모두 생성

Mapstruct

Mapstruct 기본 사용 방법

코드로 설명

Mapstruct가 매핑을 정상적으로 하기 위한 우선 순위 조건 (중요)

1번 부터 우선 순위가 가장 높으며, 우선 순위가 높은 조건에 먼저 일치하면 해당 조건으로 매핑을 진행한다.

1. Builder 패턴이 적용되어 있는 경우
2. 모든 필드를 파라미터로 가지는 생성자가 있는 경우
    a. 단, 기본 생성자가 포함 되어 있을 경우 2의 생성자는 제 역할을 못한다.
3. setter 메서드가 있는 경우

학습 코드 리뷰

코드로 설명

실습 과제 리뷰

과제 내용 확인 및 코드 설명으로 대체

Stream API 리뷰(Optional)