초격차 패키지 Online.

Spring Data JPA 와 테크닉

PART1 | **@Repository**

Spring Data JPA 의 시작점

PART2 | **@Entity 디자인**

객체를 정의하고 DB 데이터와 연결짓는 테크닉

PART3 | **DataSource, TransactionManager**

Spring Data JPA 기반 기술 정리

PART4 | **DB 바꾸기 (MySQL -> PostgreSQL)**

정말 특정 DB에 종속되지 않았을까?

PART5 | **JPA 테스트**

Repository 테스트 방법 살펴보기

Spring Data JPA 와 테크닉

3 DataSource, TransactionManager

DataSource

물리적인 데이터소스(데이터베이스) 정보를 담는 인터페이스

- 하나의 물리 데이터베이스를 표현
- 다양한 구현체를 사용
  - EmbeddedDatabaseBuilder: HSQL, Derby, H2 등 임베디드 DB 세팅할 때 사용
  - DataSourceBuilder: JDBC DataSource 빌더
  - DriverManagerDataSource: JDBC 드라이버로 세팅하는 DataSource
  - SimpleDriverDataSource: DriverManagerDataSource 를 간편하게 만든 버전
  - HikariDataSource: HikariCP 를 connection pool 로 사용하는 DataSource
  - 기타 등등

TransactionManager

스프링 트랜잭션 관리 기능을 담당하는 인터페이스

- 용도에 따라 다양한 인터페이스와 구현체들
  - PlatformTransactionManager, ReactiveTransactionManager
  - JpaTransactionManager: Spring Data JPA 일반적인 상황에 사용하는 구현체, 단일 EntityManagerFactory 를 사용
  - DataSourceTransactionManager: 단일 JDBC DataSource 를 사용하는 구현체
  - HibernateTransactionManager: 하이버네이트 SessionFactory 를 사용하는 구현체
  - ChainedTransactionManager: 여러 개의 트랜잭션 매니저를 묶어서 사용하는 구현체
    - **@Deprecated !!! (as of Boot 2.5)**
  - 등등

JPA DB 수동 설정 (Java code)

자바 코드로 DataSource, TransactionManager 를 수동 세팅해야 하는 경우가 있다.

- 언제?
  - configuration properties 로 커버되지 않는 세밀한 옵션을 줄 때
  - 다중 DataSource
- 세팅해야 하는 요소
  - DataSource
  - EntityManagerFactory -> LocalContainerEntityManagerFactoryBean
    - 추가적인 예외 처리 기능 때문에, 인터페이스 말고 구현체를 직접 빈으로 등록해야 한다!
  - PlatformTransactionManager
- 세팅 구성: DataSource (DB 설정) -> EntityManagerFactory (JPA 엔티티 관리) -> PlatformTransactionManager (트랜잭션 관리)

@Transactional

스프링이 애노테이션 기반 트랜잭션 관리 기능을 제공

- EntityManager 불러오고 -> 구역 지정하고 -> commit(), rollback() 직접 할 필요 없다!
- 서비스 클래스, 메소드에 적용하는 것으로 간단히 트랜잭션 구역을 설정
  - 동시에 설정하면 메소드가 우선순위
- JpaRepository 는 메소드 단위 @Transactional 이 이미 붙어있음
- 관련 애노테이션
  - 스프링 테스트 지원 애노테이션: @DataJpaTest 와 좋은 궁합
    - @Commit
    - @Rollback
  - javax.transaction.@Transactional: 스프링 패키지가 아님, 기대하는 기능을 주지 않으므로 주의

@Transactional: attributes

@Transactional 이 제공하는 다양한 옵션들

- transactionManager(value): 사용할 트랜잭션 매니저를 이름으로 특정
- label: 트랜잭션 구분 짓고 식별하는 레이블
- propagation: 트랜잭션이 중첩될 경우 동작(트랜잭션 효과의 전파) 규칙 (default: REQUIRED)
- isolation: 트랜잭션 내부 데이터의 격리 레벨 (default: DEFAULT)
- timeout, timeoutString: 시간 제한을 거는 것이 가능
- readOnly: "이 트랜잭션 안에서는 select 만 일어난다" 를 표현
  - 강제성이 없음을 주의 - 힌트로 생각하자
  - 이 옵션을 처리하지 않는 트랜잭션 매니저 구현체를 사용할 경우, 별도의 예외처리를 안 함
- rollbackFor, rollbackForClassName
- noRollbackFor, noRollbackForClassName

@Transactional: Propagation

중첩된 트랜잭션의 동작 규칙

- REQUIRED(default): 현재 있으면 보조, 없으면 새로 만들기
- SUPPORTS: 현재 있으면 보조, 없으면 트랜잭션 없이 실행
- MANDATORY: 있으면 보조, 없으면 예외 처리
- REQUIRES_NEW: 트랜잭션 생성하고 실행, 현재 있던 것은 미룸
- NOT_SUPPORTED: 트랜잭션 없이 실행, 현재 있던 것은 미룸
- NEVER: 트랜잭션 없이 실행, 현재 트랜잭션이 있었으면 예외 처리
- NESTED: 현재 있으면 그 안에서 중첩된 트랜잭션 형성

@Transactional: Isolation

트랜잭션 내부 데이터의 격리 수준

- DEFAULT: 데이버테이스에게 맡김
- READ_UNCOMMITTED: dirty read + non-repeatable read + phantom read
- READ_COMMITTED: non-repeatable read + phantom read
- REPEATABLE_READ: phantom read
- SERIALIZABLE: 완전 직렬 수행

Reference

- https://docs.spring.io/spring-data/jpa/docs/2.5.5/reference/html/#jpa.java-config
- https://github.com/spring-projects/spring-data-commons/issues/2232