스프링 데이터 JPA

JPA(Java Persistence API)를 보다 쉽게 사용할 수 있도록 여러 기능을 제공하는 **스프링 데이터 JPA**에 대해 학습합니다.

왜 JPA를 학습해야 하는가?

- 도메인 주도 개발이 가능합니다.
    - 애플리케이션의 코드가 SQL 데이터베이스 관련 코드에 잠식 당하는 것을 방지하고 도메인 기반의 프로그래밍으로 비즈니스 로직을 구현하는데 집중할 수 있습니다.
- 그리고 개발 생산성에 좋으며, 데이터베이스에 독립적인 프로그래밍이 가능하고, 타입 세이프한 쿼리 작성 그리고 Persistent Context가 제공하는 캐시 기능으로 성능 최적화까지 가능합니다.

이러한 여러 장점을 지닌 JPA의 한가지 단점. 높은 학습 비용

학습 목표

- ORM(Object-Relation Mapping)에 대해 이해합니다.
- JPA를 사용할 때 반드시 알아야 하는 특징을 이해합니다.
- 스프링 데이터 JPA의 구동 원리를 이해합니다.
- 스프링 데이터 JPA를 사용하여 다양한 방법으로 Repository를 구현할 수 있습니다.
- 스프링 데이터 JPA를 사용하여 다양한 방법으로 쿼리를 만들고 실행할 수 있습니다.

학습 목차

- 핵심 개념 이해
    - 관계형 데이터베이스와 자바
    - ORM이 해결하려는 문제 (패러다임 불일치)
    - JPA의 특징 이해

    - 스프링 데이터 JPA 원리 이해
- 스프링 데이터 JPA 활용
    - 스프링 부트 연동
    - 리포지토리
    - 엔티티 저장
    - 쿼리
    - 스토어드 프로시져
    - 트랜잭션
    - Auditing

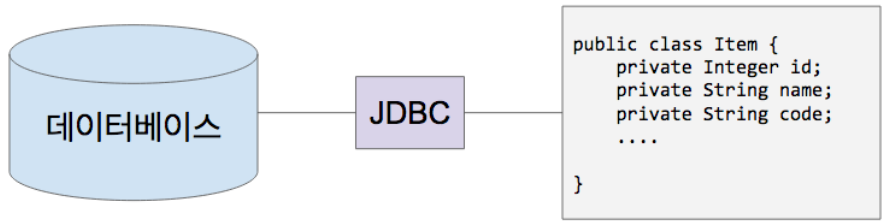
1부: 핵심 개념 이해

본격적인 스프링 데이터 JPA 활용법을 학습하기에 앞서, ORM과 JPA에 대한 이론적인 배경을 학습합니다.

1. 관계형 데이터베이스와 자바

JDBC

- (관계형) 데이터베이스와 자바의 연결 고리



JDBC

- DataSource / DriverManager
- Connection
- PreparedStatement

SQL

- DDL
- DML

무엇이 문제인가?

- SQL을 실행하는 비용이 비싸다.
- SQL이 데이터베이스 마다 다르다.
- 스키마를 바꿨더니 코드가 너무 많이 바뀌네...
- 반복적인 코드가 너무 많아.
- 당장은 필요가 없는데 언제 쓸 줄 모르니까 미리 다 읽어와야 하나...

2. ORM: Object-Relation Mapping

JDBC 사용

```
        try(Connection connection = DriverManager.getConnection(url, username, password))
{
            System.out.println("Connection created: " +  connection);
            String sql = "INSERT INTO ACCOUNT VALUES(1, 'id', 'pass');";
            try(PreparedStatement statement = connection.prepareStatement(sql)) {
                statement.execute();
            }
        }
```

도메인 모델 사용

```
Account account = new Account(“id”, “pass”);
accountRepository.save(account);
```

JDBC 대신 도메인 모델 사용하려는 이유?

- 객체 지향 프로그래밍의 장점을 활용하기 좋으니까.
- 각종 디자인 패턴
- 코드 재사용
- 비즈니스 로직 구현 및 테스트 편함.

ORM은 애플리케이션의 클래스와 SQL 데이터베이스의 테이블 사이의 **맵핑 정보를 기술한 메타데이터**를 사용하여, 자바 애플리케이션의 객체를 SQL 데이터베이스의 테이블에 **자동으로 (또 깨끗하게) 영속화** 해주는 기술입니다.

> In a nutshell, object/relational mapping is the automated (and transparent) persistence of objects in a Java application to the tables in an SQL database, using metadata that describes the mapping between the classes of the application and the schema of the SQL database.
>
> - Java Persistence with Hibernate, Second Edition

장점	단점
생산성 유지보수성 성능 밴더 독립성	학습비용

3. ORM: 패러다임 불일치

객체를 릴레이션에 맵핑하려니 발생하는 문제들과 해결책

밀도(Granularity) 문제

객체	릴레이션
다양한 크기의 객체를 만들 수 있음. 커스텀한 타입 만들기 쉬움.	테이블 기본 데이터 타입 (UDT는 비추)

서브타입(Subtype) 문제

객체	릴레이션
상속 구조 만들기 쉬움. 다형성.	테이블 상속이라는게 없음. 상속 기능을 구현했다 하더라도 표준 기술이 아님. 다형적인 관계를 표현할 방법이 없음.

식별성(Identity) 문제

객체	릴레이션
레퍼런스 동일성 (==) 인스턴스 동일성 (equals() 메소드)	주키 (primary key)

관계(Association) 문제

객체	릴레이션
객체 레퍼런스로 관계 표현. 근본적으로 ‘방향'이 존재 한다. 다대다 관계를 가질 수 있음	외래키(foreign key)로 관계 표현. ‘방향'이라는 의미가 없음. 그냥 Join으로 아무거나 묶을 수 있음. 태생적으로 다대다 관계를 못만들고, 조인 테이블 또는 링크 테이블을 사용해서 두개의 1대다 관계로 풀어야 함.

데이터 네비게이션(Navigation)의 문제

객체	릴레이션
레퍼런스를 이용해서 다른 객체로 이동 가능. 콜렉션을 순회할 수도 있음.	하지만 그런 방식은 릴레이션에서 데이터를 조회하는데 있어서 매우 비효율적이다. 데이터베이스에 요청을 적게 할 수록 성능이 좋다. 따라서 Join을 쓴다. 하지만, 너무 많이 한번에 가져오려고 해도 문제다. 그렇다고 lazy loading을 하자니 그것도 문제다. (n+1 select)

4. JPA 프로그래밍: 프로젝트 세팅

데이터베이스 실행

- oracle
-

스프링 부트

- 스프링 부트 v3.*
- 스프링 프레임워크 v5.*

스프링 부트 스타터 JPA

- JPA 프로그래밍에 필요한 의존성 추가
    - JPA v2.*
    - Hibernate v5.*
- 자동 설정: HibernateJpaAutoConfiguration
    - 컨테이너가 관리하는 EntityManager (프록시) 빈 설정
    - PlatformTransactionManager 빈 설정

JDBC 설정

- jdbc:oracle:thin:@localhost:1521:xe

application.properties

- spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
- spring.jpa.hibernate.ddl-auto=create

5. JPA 프로그래밍: 엔티티 맵핑

@Entity

- “엔티티”는 객체 세상에서 부르는 이름.
- 보통 클래스와 같은 이름을 사용하기 때문에 값을 변경하지 않음.
- 엔티티의 이름은 JQL에서 쓰임.

@Table

- “릴레이션" 세상에서 부르는 이름.
- @Entity의 이름이 기본값.
- 테이블의 이름은 SQL에서 쓰임.

@Id

- 엔티티의 주키를 맵핑할 때 사용.
- 자바의 모든 primitive 타입과 그 랩퍼 타입을 사용할 수 있음
    - Date랑 BigDecimal, BigInteger도 사용 가능.
- 복합키를 만드는 맵핑하는 방법도 있지만 그건 논외로..

@GeneratedValue

- 주키의 생성 방법을 맵핑하는 애노테이션
- 생성 전략과 생성기를 설정할 수 있다.
    - 기본 전략은 AUTO: 사용하는 DB에 따라 적절한 전략 선택
    - TABLE, SEQUENCE, IDENTITY 중 하나.

@Column

- unique
- nullable
- length
- columnDefinition
- ...

@Temporal

- 현재 JPA 2.1까지는 Date와 Calendar만 지원.

@Transient

- 컬럼으로 맵핑하고 싶지 않은 멤버 변수에 사용.

application.properties

- spring.jpa.show-sql=true
- spring.jpa.properties.hibernate.format_sql=true

6. JPA 프로그래밍: Value 타입 맵핑

엔티티 타입과 Value 타입 구분

- 식별자가 있어야 하는가.
- 독립적으로 존재해야 하는가.

Value 타입 종류

- 기본 타입 (String, Date, Boolean, ...)
- Composite Value 타입
- Collection Value 타입
    - 기본 타입의 콜렉션
    - 컴포짓 타입의 콜렉션

Composite Value 타입 맵핑

- @Embeddable
- @Embedded
- @AttributeOverrides
- @AttributeOverride

```
@Embeddable
public class Address {

    private String street;

    private String city;

    private String state;

    private String zipCode;

}
```

```
    // Account

    @Embedded
    @AttributeOverrides({
        @AttributeOverride(name = "street", column =
@Column(name = "home_street"))
    })
    private Address address;
```

7.JPA 프로그래밍: 1대다 맵핑

관계에는 항상 두 엔티티가 존재 합니다.

- 둘 중 하나는 그 관계의 주인(owning)이고
- 다른 쪽은 종속된(non-owning) 쪽입니다.
- 해당 관계의 반대쪽 레퍼런스를 가지고 있는 쪽이 주인.

단방향에서의 관계의 주인은 명확합니다.

- 관계를 정의한 쪽이 그 관계의 주인입니다.

단방향 @ManyToOne

- 기본값은 FK 생성

단방향 @OneToMany

- 기본값은 조인 테이블 생성

양방향

- FK 가지고 있는 쪽이 오너 따라서 기본값은 @ManyToOne 가지고 있는 쪽이 주인.
- 주인이 아닌쪽(@OneToMany쪽)에서 mappedBy 사용해서 관계를 맺고 있는 필드를 설정해야 합니다.

양방향

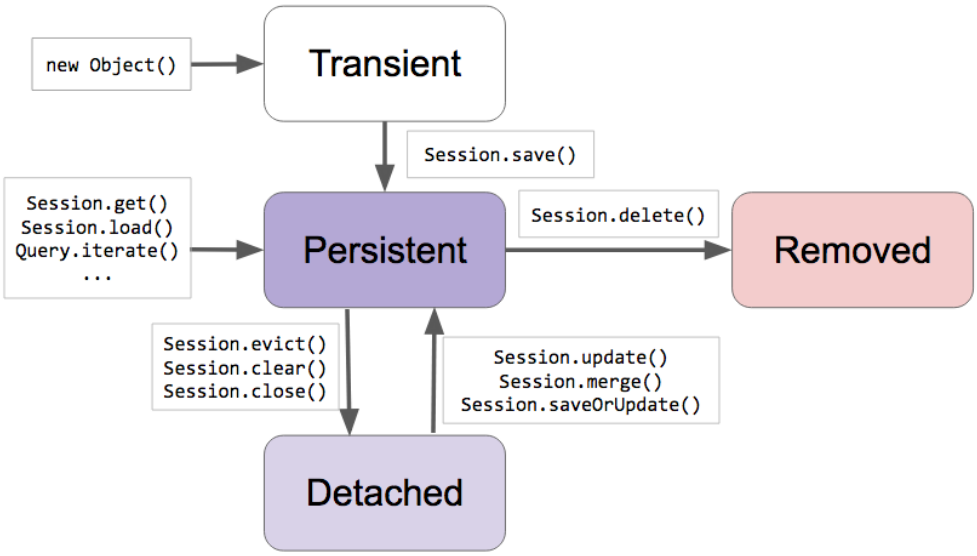
- @ManyToOne (이쪽이 주인)
- @OneToMany(mappedBy)
- 주인한테 관계를 설정해야 DB에 반영이 됩니다.

8. JPA 프로그래밍: Cascade

엔티티의 상태 변화를 전파 시키는 옵션.

잠깐? **엔티티의 상태**가 뭐지?

- Transient: JPA가 모르는 상태
- Persistent: JPA가 관리중인 상태 (1차 캐시, Dirty Checking, Write Behind, ...)
- Detached: JPA가 더이상 관리하지 않는 상태.
- Removed: JPA가 관리하긴 하지만 삭제하기로 한 상태.



9. JPA 프로그래밍: Fetch

연관 관계의 엔티티를 어떻게 가져올 것이냐... 지금 (Eager)? 나중에(Lazy)?

- @OneToMany의 기본값은 Lazy
- @ManyToOne의 기본값은 Eager

10. JPA 프로그래밍: Query

JPQL (HQL)

- Java Persistence Query Language / Hibernate Query Language
- 데이터베이스 테이블이 아닌, 엔티티 객체 모델 기반으로 쿼리 작성.

- JPA 또는 하이버네이트가 해당 쿼리를 SQL로 변환해서 실행함.
- https://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html#hql

```
    TypedQuery<Post> query = entityManager.createQuery("SELECT p FROM Post As p", Post.class);
    List<Post> posts = query.getResultList();
```

Criteria

- 타입 세이프 쿼리
- https://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html#criteria

```
    CriteriaBuilder builder = entityManager.getCriteriaBuilder();
    CriteriaQuery<Post> criteria = builder.createQuery(Post.class);
    Root<Post> root = criteria.from(Post.class);
    criteria.select(root);
    List<Post> posts = entityManager.createQuery(criteria).getResultList();
```

Native Query

- SQL 쿼리 실행하기
- https://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html#sql

```
List<Post> posts = entityManager
            .createNativeQuery("SELECT * FROM Post", Post.class)
            .getResultList();
```

11. 스프링 데이터 JPA 소개 및 원리

JpaRepository<Entity, Id> 인터페이스

- 매직 인터페이스
- @Repository가 없어도 빈으로 등록해 줌.

@EnableJpaRepositories

- 매직의 시작은 여기서 부터

매직은 어떻게 이뤄지나?

- 시작은 @Import(**JpaRepositoriesRegistrar.class**)
- 핵심은 **ImportBeanDefinitionRegistrar** 인터페이스

12. 핵심 개념 이해 정리

데이터베이스와 자바

패러다임 불일치

ORM이란?

JPA 사용법 (엔티티, 벨류 타입, 관계 맵핑)

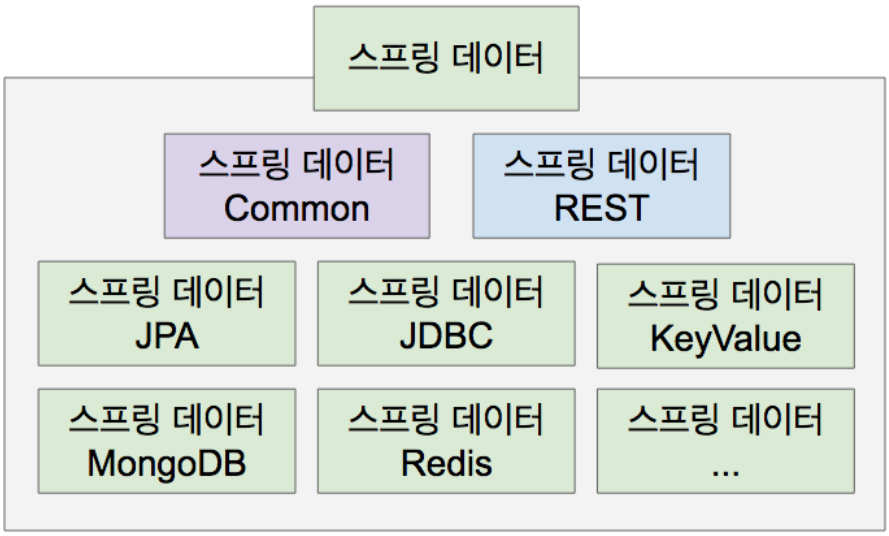
JPA 특징 (엔티티 상태 변화, Cascade, Fetch, 1차 캐시, ...)

주의할 점

- 반드시 발생하는 SQL을 확인할 것.
- 팁: “?”에 들어있는 값 출력하기
    - logging.level.org.hibernate.SQL=debug
    - logging.level.org.hibernate.type.descriptor.sql=trace

2부: 스프링 데이터 JPA 활용

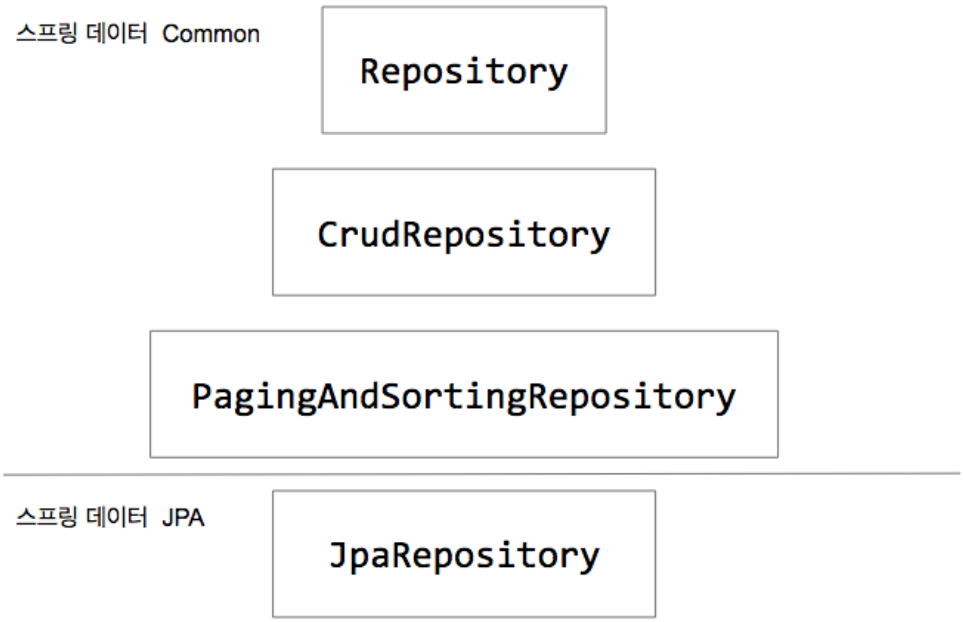
13. 스프링 데이터 JPA 활용 파트 소개



스프링 데이터	SQL & NoSQL 저장소 지원 프로젝트의 묶음.
스프링 데이터 Common	여러 저장소 지원 프로젝트의 공통 기능 제공.
스프링 데이터 REST	저장소의 데이터를 하이퍼미디어 기반 HTTP 리소스로(REST API로) 제공하는 프로젝트.
스프링 데이터 JPA	스프링 데이터 Common이 제공하는 기능에 **JPA 관련 기능 추가**.

http://projects.spring.io/spring-data/

14. 스프링 데이터 Common: Repository



15. 스프링 데이터 Common: Repository 인터페이스 정의하기

Repository 인터페이스로 공개할 메소드를 직접 일일히 정의하고 싶다면

특정 리포지토리 당

- @RepositoryDefinition

```
@RepositoryDefinition(domainClass = Comment.class, idClass = Long.class)
public interface CommentRepository {

    Comment save(Comment comment);

    List<Comment> findAll();

}
```

공통 인터페이스 정의

- @NoRepositoryBean

```
@NoRepositoryBean
public interface MyRepository<T, ID extends Serializable> extends Repository<T, ID> {

    <E extends T> E save(E entity);

    List<T> findAll();

}
```

16. 스프링 데이터 Common: Null 처리하기

스프링 데이터 2.0 부터 자바 8의 Optional 지원.

- Optional<Post> findById(Long id);

콜렉션은 Null을 리턴하지 않고, 비어있는 콜렉션을 리턴합니다.

스프링 프레임워크 5.0부터 지원하는 Null 애노테이션 지원.

- @NonNullApi, @NonNull, @Nullable.
- 런타임 체크 지원 함.
- JSR 305 애노테이션을 메타 애노테이션으로 가지고 있음. (IDE 및 빌드 툴 지원)

17. 스프링 데이터 Common: 쿼리 만들기 개요

스프링 데이터 저장소의 메소드 이름으로 쿼리 만드는 방법

- 메소드 이름을 분석해서 **쿼리 만들기** (CREATE)
- 미리 정의해 둔 **쿼리 찾아 사용하기** (USE_DECLARED_QUERY)
- **미리 정의한 쿼리 찾아보고 없으면 만들기 (CREATE_IF_NOT_FOUND)**

쿼리 만드는 방법

- 리턴타입 {접두어}{도입부}By{프로퍼티 표현식}(조건식)[(And|Or){프로퍼티 표현식}(조건식)]{정렬 조건} (매개변수)

접두어	Find, Get, Query, Count, ...
도입부	Distinct, First(N), Top(N)
프로퍼티 표현식	Person.Address.ZipCode => find(Person)ByAddress_ZipCode(...)
조건식	IgnoreCase, Between, LessThan, GreaterThan, Like, Contains, ...
정렬 조건	OrderBy{프로퍼티}Asc|Desc
리턴 타입	E, Optional<E>, List<E>, Page<E>, Slice<E>, Stream<E>
매개변수	Pageable, Sort

쿼리 찾는 방법

- 메소드 이름으로 쿼리를 표현하기 힘든 경우에 사용.
- 저장소 기술에 따라 다름.
- JPA: @Query @NamedQuery

18. 스프링 데이터 Common: 쿼리 만들기

기본 예제

```
List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);
// distinct
List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);
// ignoring case
List<Person> findByLastnameIgnoreCase(String lastname);
// ignoring case
List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);
```

정렬

```
List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
```

페이징

```
Page<User> findByLastname(String lastname, Pageable pageable);
Slice<User> findByLastname(String lastname, Pageable pageable);
List<User> findByLastname(String lastname, Sort sort);
List<User> findByLastname(String lastname, Pageable pageable);
```

스트리밍

```
Stream<User> readAllByFirstnameNotNull();
```

- try-with-resource 사용할 것. (Stream을 다 쓴다음에 close() 해야 함)

19. 스프링 데이터 Common: 비동기 쿼리

비동기 쿼리

```
@Async Future<User> findByFirstname(String firstname);
@Async CompletableFuture<User> findOneByFirstname(String firstname);
@Async ListenableFuture<User> findOneByLastname(String lastname);
```

- 해당 메소드를 스프링 TaskExecutor에 전달해서 별도의 쓰레드에서 실행함.
- Reactive랑은 다른 것임

권장하지 않는 이유

- 테스트 코드 작성이 어려움.
- 코드 복잡도 증가.
- 성능상 이득이 없음.
    - DB 부하는 결국 같고.
    - 메인 쓰레드 대신 백드라운드 쓰레드가 일하는 정도의 차이.
    - 단, 백그라운드로 실행하고 결과를 받을 필요가 없는 작업이라면 @Async를 사용해서 응답 속도를 향상 시킬 수는 있다.

20. 스프링 데이터 Common: 커스텀 리포지토리

쿼리 메소드(쿼리 생성과 쿼리 찾아쓰기)로 해결이 되지 않는 경우 직접 코딩으로 구현 가능.

- 스프링 데이터 리포지토리 **인터페이스에 기능 추가**.
- 스프링 데이터 리포지토리 **기본 기능 덮어쓰기 가능**.
- 구현 방법
    a. 커스텀 리포지토리 인터페이스 정의
    b. 인터페이스 구현 클래스 만들기 (기본 접미어는 Impl)
    c. 엔티티 리포지토리에 커스텀 리포지토리 인터페이스 추가

기능 추가하기

기본 기능 덮어쓰기

접미어 설정하기

21. 스프링 데이터 Common: 기본 리포지토리 커스터마이징

모든 리포지토리에 공통적으로 추가하고 싶은 기능이 있거나 덮어쓰고 싶은 기본 기능이 있다면

1. JpaRepository를 상속 받는 인터페이스 정의
    - @NoRepositoryBean
2. 기본 구현체를 상속 받는 커스텀 구현체 만들기
3. @EnableJpaRepositories에 설정
    - repositoryBaseClass

```
@NoRepositoryBean
public interface MyRepository<T, ID extends Serializable> extends JpaRepository<T, ID> {

    boolean contains(T entity);

}
```

```
public class SimpleMyRepository<T, ID extends Serializable> extends SimpleJpaRepository<T, ID>
implements MyRepository<T, ID> {

    private EntityManager entityManager;

    public SimpleMyRepository(JpaEntityInformation<T, ?> entityInformation, EntityManager
entityManager) {
        super(entityInformation, entityManager);
        this.entityManager = entityManager;
    }

    @Override
    public boolean contains(T entity) {
        return entityManager.contains(entity);
    }
}
```

```
@EnableJpaRepositories(repositoryBaseClass = SimpleMyRepository.class)
```

```
public interface PostRepository extends MyRepository<Post, Long> {
}
```

22. 스프링 데이터 Common: 도메인 이벤트

도메인 관련 이벤트를 발생시키기

스프링 프레임워크의 이벤트 관련 기능

- https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#context-functionality-events
- ApplicationContext extends **ApplicationEventPublisher**
- 이벤트: extends ApplicationEvent
- 리스너
    - implements ApplicationListener<E extends ApplicationEvent>
    - @EventListener

스프링 데이터의 도메인 이벤트 Publisher

- @DomainEvents
- @AfterDomainEventPublication
- extends AbstractAggregateRoot<E>
- 현재는 save() 할 때만 발생 합니다.

23. 스프링 데이터 Common: QueryDSL

findByFirstNameIngoreCaseAndLastNameStartsWithIgnoreCase(String firstName, String lastName)
이게 이게 뭐냐... @_@ 어지러우시죠?? 이 정도 되면 그냥 한글로 주석을 달아 두시는게...

여러 쿼리 메소드는 대부분 두 가지 중 하나.

- Optional<T> findOne(**Predicate**): **이런 저런 조건**으로 무언가 하나를 찾는다.

- List<T>|Page<T>|.. findAll(**Predicate**): **이런 저런 조건**으로 무언가 여러개를 찾는다.
- QuerydslPredicateExecutor 인터페이스

QueryDSL

- http://www.querydsl.com/
- 타입 세이프한 쿼리 만들 수 있게 도와주는 라이브러리
- JPA, SQL, MongoDB, JDO, Lucene, Collection 지원
- QueryDSL JPA 연동 가이드

스프링 데이터 JPA + QueryDSL

- 인터페이스: QuerydslPredicateExecutor<T>
- 구현체: QuerydslPredicateExecutor<T>

연동 방법

- 기본 리포지토리 커스터마이징 안 했을 때.
- 기본 리포지토리 커스타마이징 했을 때.

의존성 추가

```
<dependency>
   <groupId>com.querydsl</groupId>
   <artifactId>querydsl-apt</artifactId>
</dependency>
<dependency>
   <groupId>com.querydsl</groupId>
   <artifactId>querydsl-jpa</artifactId>
</dependency>
```

```
<plugin>
   <groupId>com.mysema.maven</groupId>
   <artifactId>apt-maven-plugin</artifactId>
   <version>1.1.3</version>
   <executions>
      <execution>
         <goals>
            <goal>process</goal>
         </goals>
         <configuration>
            <outputDirectory>target/generated-sources/java</outputDirectory>
            <processor>com.querydsl.apt.jpa.JPAAnnotationProcessor</processor>
         </configuration>
```

```
            </execution>
          </executions>
        </plugin>
```

```
public interface AccountRepository extends JpaRepository<Account, Long>,
QuerydslPredicateExecutor<Account> {
}
```

24. 스프링 데이터 Common: Web 1부: 웹 지원 기능 소개

스프링 데이터 웹 지원 기능 설정

- 스프링 부트를 사용하는 경우에.. 설정할 것이 없음. (자동 설정)
- 스프링 부트 사용하지 않는 경우?

```
@Configuration
@EnableWebMvc
@EnableSpringDataWebSupport
class WebConfiguration {}
```

제공하는 기능

- 도메인 클래스 컨버터
- @RequestHandler 메소드에서 Pageable과 Sort 매개변수 사용
- Page 관련 HATEOAS 기능 제공
    - PagedResourcesAssembler
    - PagedResoure
- Payload 프로젝션
    - 요청으로 들어오는 데이터 중 일부만 바인딩 받아오기
    - @ProjectedPayload, @XBRead, @JsonPath
- 요청 쿼리 매개변수를 QueryDSLdml Predicate로 받아오기
    - ?firstname=Mr&lastname=White => Predicate

25. 스프링 데이터 Common: Web 2부: DomainClassConverter

스프링 Converter

- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/core/convert/converter/Converter.html
- Formatter도 들어 본 것 같은데...

```
@GetMapping("/posts/{id}")
public String getAPost(@PathVariable Long id) {
    Optional<Post> byId = postRepository.findById(id);
    Post post = byId.get();
    return post.getTitle();
}
```

```
@GetMapping("/posts/{id}")
public String getAPost(@PathVariable(“id”) Post post) {
    return post.getTitle();
}
```

26. 스프링 데이터 Common: Web 3부: Pageable과 Sort 매개변수

스프링 MVC HandlerMethodArgumentResolver

- 스프링 MVC 핸들러 메소드의 매개변수로 받을 수 있는 객체를 확장하고 싶을 때 사용하는 인터페이스
- https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/method/support/HandlerMethodArgumentResolver.html

페이징과 정렬 관련 매개변수

- page: 0부터 시작.
- size: 기본값 20.
- sort: property,property(,ASC|DESC)
- 예) sort=created,desc&sort=title (asc가 기본값)

27. 스프링 데이터 Common: Web 4부: HATEOAS

Page를 PagedResource로 변환하기

- 일단 HATEOAS 의존성 추가 (starter-hateoas)
- 핸들러 매개변수로 PagedResourcesAssembler

리소스로 변환하기 전


```
{
  "content":[
...
    {
      "id":111,
      "title":"jpa",
      "created":null
    }
  ],
  "pageable":{
    "sort":{
      "sorted":true,
      "unsorted":false
    },
    "offset":20,
    "pageSize":10,
    "pageNumber":2,
    "unpaged":false,
    "paged":true
  },
  "totalElements":200,
  "totalPages":20,
  "last":false,
  "size":10,
  "number":2,
  "first":false,
```

```
  "numberOfElements":10,
  "sort":{
    "sorted":true,
    "unsorted":false
  }
}
```


리소스로 변환한 뒤


```
{
  "_embedded":{
    "postList":[
      {
        "id":140,
        "title":"jpa",
        "created":null
      },
...
      {
        "id":109,
        "title":"jpa",
        "created":null
      }
    ]
  },
  "_links":{
    "first":{
      "href":"http://localhost/posts?page=0&size=10&sort=created,desc&sort=title,asc"
    },
    "prev":{
      "href":"http://localhost/posts?page=1&size=10&sort=created,desc&sort=title,asc"
    },
    "self":{
      "href":"http://localhost/posts?page=2&size=10&sort=created,desc&sort=title,asc"
    },
    "next":{
      "href":"http://localhost/posts?page=3&size=10&sort=created,desc&sort=title,asc"
    },
    "last":{
      "href":"http://localhost/posts?page=19&size=10&sort=created,desc&sort=title,asc"
    }
  },
  "page":{
    "size":10,
    "totalElements":200,
    "totalPages":20,
    "number":2
  }
}
```

28. 스프링 데이터 Common: 마무리

지금까지 살펴본 내용

- 스프링 데이터 Repository
- 쿼리 메소드
    - 메소드 이름 보고 만들기
    - 메소드 이름 보고 찾기
- Repository 정의하기
    - 내가 쓰고 싶은 메소드만 골라서 만들기
    - Null 처리
- 쿼리 메소드 정의하는 방법
- 리포지토리 커스터마이징
    - 리포지토리 하나 커스터마이징
    - 모든 리포지토리의 베이스 커스터마이징
- 도메인 이벤트 Publish
- 스프링 데이터 확장 기능
    - QueryDSL 연동
    - 웹 지원

29. 스프링 데이터 JPA: JPA Repository

@EnableJpaRepositories

- 스프링 부트 사용할 때는 사용하지 않아도 자동 설정 됨.
- 스프링 부트 사용하지 않을 때는 @Configuration과 같이 사용.

@Repository 애노테이션을 붙여야 하나 말아야 하나...

- 안붙여도 됩니다.
- 이미 붙어 있어요. 또 붙인다고 별일이 생기는건 아니지만 중복일 뿐입니다.

스프링 @Repository

- SQLExcpetion 또는 JPA 관련 예외를 스프링의 DataAccessException으로 변환 해준다.

30. 스프링 데이터 JPA: 엔티티 저장하기

JpaRepository의 save()는 단순히 새 엔티티를 추가하는 메소드가 아닙니다.

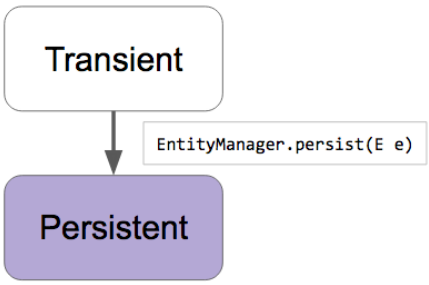
- Transient 상태의 객체라면 EntityManager.persist()
- Detached 상태의 객체라면 EntityManager.merge()

Transient인지 Detached 인지 어떻게 판단 하는가?

- 엔티티의 @Id 프로퍼티를 찾는다. 해당 프로퍼티가 null이면 Transient 상태로 판단하고 id가 null이 아니면 Detached 상태로 판단한다.
- 엔티티가 Persistable 인터페이스를 구현하고 있다면 isNew() 메소드에 위임한다.
- JpaRepositoryFactory를 상속받는 클래스를 만들고 getEntityInfomration()을 오버라이딩해서 자신이 원하는 판단 로직을 구현할 수도 있습니다.

EntityManager.persist()

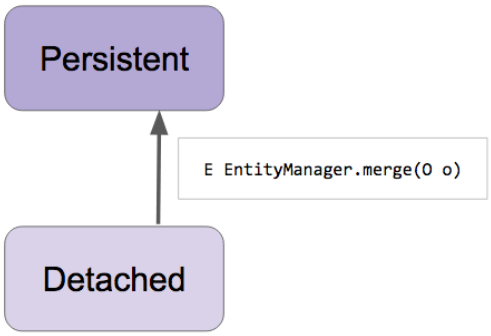
- https://docs.oracle.com/javaee/6/api/javax/persistence/EntityManager.html#persist(java.lang.Object)
- Persist() 메소드에 넘긴 그 엔티티 객체를 Persistent 상태로 변경합니다.



EntityManager.merge()

- https://docs.oracle.com/javaee/6/api/javax/persistence/EntityManager.html#merge(java.lang.Object)

- Merge() 메소드에 넘긴 그 엔티티의 복사본을 만들고, 그 복사본을 다시 Persistent 상태로 변경하고 그 복사본을 반환합니다.



31. 스프링 데이터 JPA: 쿼리 메소드

쿼리 생성하기

- https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation
- And, Or
- Is, Equals
- LessThan, LessThanEqual, GreaterThan, GreaterThanEqual
- After, Before
- IsNull, IsNotNull, NotNull
- Like, NotLike
- StartingWith, EndingWith, Containing
- OrderBy
- Not, In, NotIn
- True, False
- IgnoreCase

쿼리 찾아쓰기

- 엔티티에 정의한 쿼리 찾아 사용하기 JPA Named 쿼리
  - @NamedQuery
  - @NamedNativeQuery
- 리포지토리 메소드에 정의한 쿼리 사용하기

- @Query
- @Query(nativeQuery=true)

32. 스프링 데이터 JPA: 쿼리 메소드 Sort

이전과 마찬가지로 Pageable이나 Sort를 매개변수로 사용할 수 있는데, @Query와 같이 사용할 때 제약 사항이 하나 있습니다.

Order by 절에서 함수를 호출하는 경우에는 Sort를 사용하지 못합니다. 그 경우에는 JpaSort.unsafe()를 사용 해야 합니다.

- Sort는 그 안에서 사용한 **프로퍼티** 또는 **alias**가 엔티티에 없는 경우에는 예외가 발생합니다.
- JpaSort.unsafe()를 사용하면 함수 호출을 할 수 있습니다.
  - JpaSort.unsafe("LENGTH(firstname)");

33. 스프링 데이터 JPA: Named Parameter과 SpEL

Named Parameter

- @Query에서 참조하는 매개변수를 ?1, ?2 이렇게 채번으로 참조하는게 아니라 이름으로 :title 이렇게 참조하는 방법은 다음과 같습니다.

```
@Query("SELECT p FROM Post AS p WHERE p.title = :title")
List<Post> findByTitle(@Param("title") String title, Sort sort);
```

SpEL

- 스프링 표현 언어
- https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#expressions
- @Query에서 엔티티 이름을 #{#entityName} 으로 표현할 수 있습니다.

```
@Query("SELECT p FROM #{#entityName} AS p WHERE p.title = :title")
List<Post> findByTitle(@Param("title") String title, Sort sort);
```

34. 스프링 데이터 JPA: Update 쿼리 메소드

쿼리 생성하기

- find...
- count...
- delete...
- 흠.. update는 어떻게 하지?

Update 또는 Delete 쿼리 직접 정의하기

- @Modifying @Query
- 추천하진 않습니다.

```
@Modifying(clearAutomatically = true, flushAutomatically = true)
@Query("UPDATE Post p SET p.title = ?2 WHERE p.id = ?1")
int updateTitle(Long id, String title);
```

35. 스프링 데이터 JPA: EntityGraph

쿼리 메소드 마다 연관 관계의 Fetch 모드를 설정 할 수 있습니다.

@NamedEntityGraph

- @Entity에서 재사용할 여러 엔티티 그룹을 정의할 때 사용.

@EntityGraph

- @NamedEntityGraph에 정의되어 있는 엔티티 그룹을 사용 함.
- 그래프 타입 설정 가능
    - (기본값) FETCH: 설정한 엔티티 애트리뷰트는 EAGER 패치 나머지는 LAZY 패치.
    - LOAD: 설정한 엔티티 애트리뷰트는 EAGER 패치 나머지는 기본 패치 전략 따름.

36. 스프링 데이터 JPA: Projection

엔티티의 일부 데이터만 가져오기.

인터페이스 기반 프로젝션

- Nested 프로젝션 가능.
- **Closed 프로젝션**
    - 쿼리를 최적화 할 수 있다. 가져오려는 애트리뷰트가 뭔지 알고 있으니까.
    - Java 8의 디폴트 메소드를 사용해서 연산을 할 수 있다.
- Open 프로젝션
    - @Value(SpEL)을 사용해서 연산을 할 수 있다. 스프링 빈의 메소드도 호출 가능.
    - 쿼리 최적화를 할 수 없다. SpEL을 엔티티 대상으로 사용하기 때문에.

클래스 기반 프로젝션

- DTO
- 롬복 @Value로 코드 줄일 수 있음

다이나믹 프로젝션

- 프로젝션 용 메소드 하나만 정의하고 실제 프로젝션 타입은 타입 인자로 전달하기.

```
<T> List<T> findByPost_Id(Long id, Class<T> type);
```

37. 스프링 데이터 JPA: Specifications

에릭 에반스의 책 DDD에서 언급하는 Specification 개념을 차용 한 것으로 QueryDSL의 Predicate와 비슷합니다.

설정 하는 방법

- https://docs.jboss.org/hibernate/stable/jpamodelgen/reference/en-US/html_single/
- 의존성 설정
- 플러그인 설정
- IDE에 애노테이션 처리기 설정
- 코딩 시작

```
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-jpamodelgen</artifactId>
    </dependency>
```

```
        <plugin>
          <groupId>org.bsc.maven</groupId>
          <artifactId>maven-processor-plugin</artifactId>
          <version>2.0.5</version>
          <executions>
            <execution>
              <id>process</id>
              <goals>
                <goal>process</goal>
              </goals>
              <phase>generate-sources</phase>
              <configuration>
                <processors>

<processor>org.hibernate.jpamodelgen.JPAMetaModelEntityProcessor</processor>
                </processors>
              </configuration>
            </execution>
          </executions>
          <dependencies>
            <dependency>
```

```
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-jpamodelgen</artifactId>
                <version>${hibernate.version}</version>
            </dependency>
        </dependencies>
    </plugin>
```

org.hibernate.jpamodelgen.JPAMetaModelEntityProcessor

```
public interface CommentRepository extends JpaRepository<Comment, Long>,
JpaSpecificationExecutor<Comment> {

}
```

38. 스프링 데이터 JPA: Query by Example

QBE는 *필드 이름을 작성할 필요 없이(뻥)* 단순한 인터페이스를 통해 동적으로 쿼리를 만드는 기능을 제공하는 사용자 친화적인 쿼리 기술입니다. (감이 1도 안잡히는거 이해합니다.. 코드를 봐야 이해하실꺼에요.)

Example = Probe + ExampleMatcher

- Probe는 필드에 어떤 값들을 가지고 있는 도메인 객체.
- ExampleMatcher는 Prove에 들어있는 그 필드의 값들을 어떻게 쿼리할 데이터와 비교할지 정의한 것.
- Example은 그 둘을 하나로 합친 것. 이걸로 쿼리를 함.

장점

- 별다른 코드 생성기나 애노테이션 처리기 필요 없음.
- *도메인 객체 리팩토링 해도 기존 쿼리가 깨질 걱정하지 않아도 됨.(뻥)*
- 데이터 기술에 독립적인 API

단점

- **nested 또는 프로퍼티 그룹 제약 조건을 못 만든다.**

- **조건이 제한적이다.** 문자열은 starts/contains/ends/regex 가 가능하고 그밖에 property는 값이 정확히 일치해야 한다.

QueryByExampleExecutor

https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#query-by-example

39. 스프링 데이터 JPA: 트랜잭션

스프링 데이터 JPA가 제공하는 Repository의 모든 메소드에는 기본적으로 @Transaction이 적용되어 있습니다.

스프링 @Transactional

- 클래스, 인터페이스, 메소드에 사용할 수 있으며, 메소드에 가장 가까운 애노테이션이 우선 순위가 높다.
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/annotation/Transactional.html (반드시 읽어볼 것.)

JPA 구현체로 Hibernate를 사용할 때 트랜잭션을 readOnly를 사용하면 좋은 점

- Flush 모드를 NEVER로 설정하여, Dirty checking을 하지 않도록 한다.

40. 스프링 데이터 JPA: Auditing

스프링 데이터 JPA의 Auditing

```
@CreatedDate
private Date created;
```

```
@LastModifiedDate
private Date updated;

@CreatedBy
@ManyToOne
private Account createdBy;

@LastModifiedBy
@ManyToOne
private Account updatedBy;
```

엔티티의 변경 시점에 언제, 누가 변경했는지에 대한 정보를 기록하는 기능.

아쉽지만 이 기능은 스프링 부트가 자동 설정 해주지 않습니다.

1. 메인 애플리케이션 위에 @EnableJpaAuditing 추가
2. 엔티티 클래스 위에 @EntityListeners(AuditingEntityListener.class) 추가
3. AuditorAware 구현체 만들기
4. @EnableJpaAuditing에 AuditorAware 빈 이름 설정하기.

JPA의 라이프 사이클 이벤트

- https://docs.jboss.org/hibernate/orm/4.0/hem/en-US/html/listeners.html
- @PrePersist
- @PreUpdate
- ...