(스터디)12장 스프링 데이터 JPA

12.2 스프링 데이터 JPA 설정 p.541

- 필요 라이브러리

xml

```
<groupId>org.springframework.data</groupId>
<artifactId>spring-data-jpa</artifactId>
<version>1.8.0.RELEASE</version>
</dependency>
```

gradle

```
dependencies {
   implementation 'org.springframework.boot:spring-boo
t-starter-data-jpa'
```

- 환경설정
  - 스프링 설정에 javaConfig를 사용하면

```
@Configuration
@EnableJpaRepositories(basepackages = "리포지토리를 검색할 패키지 위치")
public class AppConfig {}
```

- 해당 패키지와 그 하위 패키지를 검색한다.

- 스프링 데이터 JPA 는 애플리케이션을 실행할 때 basePackage에 있는 리포지토리 인터페이스들을 찾아서 해당 인터페이스를 구현한 클래스를동적으로 생성한 다음 스프링 빈으로 등록한다.
- 따라서 개발자가 직접 구현 클래스를 만들지 않아도 된다.

12.3 공통 인터페이스 기능 p.542

JpaRepository 공통 기능 인터페이스

```
public interface JpaRepository<T, ID extends Serializable> extends PagingAndSortingRepository<T, ID> {
    ...
}
```

JpaRepsository를 사용하는 인터페이스

```
public interface MemberRepsository extends JpaRepository<Member, Long> {
}
```

- 상속받은 JpaRepository<Member, Long> 부분에서 회원 엔티티와 식별자 타입을 지정.
- 이제부터 회원 레포지토리는 JpaRepository 인터페이스가 제공하는 다양한 기능을 사용할 수 있다.
- JpaRepository 인터페이스를 상속받으면 추가로 JPA에 특화된 기능을 제공한다.

12.4 쿼리 메소드 기능 p.545

- 메소드 이름만으로 쿼리를 생성하는 기능이 있는데 인터페이스에 메소드만 선언하면 해당 메소드의 이름으로 JPQL 쿼리를 생성해서 실행한다.

12.4.1 메소드 이름으로 쿼리 생성 p.545

- 정해진 규칙에 따라서 메소드 이름을 지어야 한다.
- 쿼리 생성 기능 참고

- 이메일과 이름으로 회원 조회

```
public interface MemberRepository extends Repository<Member, Long
> {
    List<Member> findByEmailAndName(String email, String name);
}
```

- 실행된 JPQL

```
select m from Member m where m.ema
il = ?1 and m.name = ?2
```

- 위의 메소드를 실행하면 스프링 데이터 JPA는 메소드 이름을 분석해서 JPQL을 생성하고 실행한다.
- 엔티티의 필드명이 변경되면 인터페이스에 정의한 메소드 이름도 꼭 함께 변경해야 한다.
- 그렇지 않으면 애플리케이션 시작 시점에 오류가 발생.

12.4.2 메소드 이름으로 JPA NamedQuery 호출 p.547

- 10.2.15절 참고
- @NamedQuery 어노테이션으로 Named 쿼리 정의

```
@Entity
@NamedQuery(
    name = "Member.findByUsername",
    query = "select m from Member m where m.username = :username")
)
public class Member {
    ...
}
```

- 스프링 데이터 JPA를 사용하면 아래와 같이 메소드 이름만으로 Named 쿼리를 호출할 수 있다.

```
public interface MemberRepository extends JpaRepository<Member, Long> {
    List<Member> findByUsername(@Param("username") String username);
}
```

- `@Param` : 이름기반 파라미터를 바인딩 할때 사용하는 어노테이션

- 스프링 데이터 JPA 는 선언한 “도메인 클래스 + .(점) + 메소드 이름” 으로 Named 쿼리를 찾아서 실행한다.
- 만약 실행할 Named 쿼리가 없으면 메소드 이름으로 쿼리 생성 전략을 사용한다.

12.4.3 @Query 어노테이션을 사용해서 리포지토리 인터페이스에 직접 정의 p.549

- 리포지토리 메소드에 직접 쿼리를 정의하려면 @Query 어노테이션을 사용한다.
- 실행할 메소드에 정적 쿼리를 직접 작성하므로 이름 없는 Named 쿼리라고 할 수 있다.
- 애플리케이션 실행 시점에 문법 오류를 발견할 수 있는 장점이 있다.
- 메소드에 JPQL 쿼리 작성

```
public interface MemberRepository extends JpaRepository<Member, Long> {
    //
    @Query("select m from Member m where m.username = ?1") // 직접 정적 쿼리를 작성
    Member findByUsername(String username); // 이름 없는 Named 쿼리 작성법
}
```

- JPQL은 위치 기반 파라미터를 1부터 시작하지만 네이비트 SQL은 0부터 시작한다.

12.4.4 파라미터 바인딩 p.549

- **위치 기반 파라미터 바인딩**
  - 기본값.
  - 코드 가독성과 유지보수가 좋다.
  - `select m from Member m where m.username =?1`
- **이름 기반 파라미터 바인딩**
  - `@Param(파라미터 이름)` 어노테이션을 사용
  - `select m from Member m where m.username = :name`

12.4.5 벌크성 수정 쿼리 p.550

- JPA를 사용한 벌크성 수정 쿼리

```
int bulkPriceUp(String stockAmount) {
    ...
    String qlString = "update Product p set p.price = p.price * 1.1 where p.stockAmount < :stockAmount";

    int resultCount = em.createQuery(qlString)
                        .setParamter("stockAmount", stockAmount)
                        .executeUpdate();
}
```

- 스프링 데이터 JPA를 사용한 벌크성 수정 쿼리

```
@Modifying
@Query("update Product p set p.price = p.price * 1.1 where p.stockAmount < :stockAmount")
int bulkPriceUp(@Param("stockAmount") String stockAmount);
```

- 벌크성 수정, 삭제 쿼리는 `@Modifying` 어노테이션을 사용하면 된다.

- 벌크성 쿼리를 실행하고 나서 영속성 컨텍스트를 초기화하고 싶으면 clearAutomatically 옵션을 true로 설정하면 된다.

12.4.6 반환 타입 p.551

- 결과가 한 건 이상이면 컬렉션 인터페이스 사용
  - `List<Member> findByName(String name);`
  - 조회 결과가 없으면 컬렉션은 빈 컬렉션을 반환
- 결과가 단건이면 반환 타입을 지정
  - `Member findByEmail(String email)`
  - 조회 결과가 없으면 단건은 null을 반환
- 단건으로 지정한 메소드를 호출하면 스프링 데이터 JPA 는 내부에서 JPQL의 Query.getSingleResult()를 호출한다.
- 이 메소드를 호출했는데 조회 결과가 없으면 NoResultException 예외가 발생하는데, 스프링 데이터 JPA는 단건을 조회 할 때 이 예외가 발생하면 예외를 무시하고 대신 null을 반환한다.

12.4.7 페이징과 정렬 p.551

- 페이징과 정렬 기능을 위한 파라미터 제공
  - org.springframework.data.domain.Sort : 정렬기능
  - org.springframework.data.domain.Pageable : 페이징 기능(내부에 Sort 포함)
- 파라미터에 `Pageable`을 사용하면 반환타입은 `List` 나 `org.springframework.data.domain.Page`를 사용 가능
- 반환타입으로 Page를 사용하면 전체 데이터 건수를 조회하는 count 쿼리를 추가로 호출한다.

```
// count 쿼리 사용
Page<Member> findByName(String name, Pageable pageable);

// count 쿼리 사용 안 함
List<Member> findByName(String name, Pageable pageable);

List<Member> findByName(String name, Sort sort);
```

12.4.8 힌트 p.553

- JPA 쿼리 힌트는 SQL 힌트가 아니라 JPA 구현체에게 제공하는 힌트다.
- forCounting 속성은 반환 타입으로 Page 인터페이스를 적하면 추가로 호출 하는 페이징을 위한 count 쿼리에도 힌트를 적용할지를 설정하는 옵션이다.

```
@QueryHints(value = {@QueryHint(name="org.hibernate.readOnly", value="true")},
             forCounting=true)
Page<Member> findByName(String name, Pageable pageable);
```

  - 기본값은 true

12.4.9 Lock p.554

- 쿼리 시 락을 걸려면 `org.springframework.data.jpa.repository.Lock` 어노테이션을 사용한다.

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
List<Member> findByName(String name);
```

- 16.1절 참고

12.5 명세 p.554

12.6 사용자 정의 리포지토리 구현 p.554

- 스프링 데이터 JPA는 필요한 메소드만 구현할 수 있는 방법을 제공한다.
- 먼저 직접 구현할 메소드를 위한 사용자 정의 인터페이스를 작성한다.
  - 사용자 정의 인터페이스

```
public interface MemberRepositoryCustom{
    public List<Member> findMemberCustom();
}
```

- 다음으로 사용자 정의 인터페이스를 구현한 클래스를 작성한다.

```
public class MemberRepositoryImpl implements MemberRepositoryCustom {
    @Override
    public List<Member> findMemberCustom() {
        ... // 사용자 정의 구현
    }
}
```

- 이때 클래스 이름을 짓는 규칙이 있는데 레포짓토리 인터페이스 이름 + Impl로 지어야 한다.
- 이렇게 하면 스프링 데이터 JPA가 사용자 정의 구현 클래스로 인식한다.

- 마지막으로 레포지토리 인터페이스에서 사용자 정의 인터페이스를 상속 받으면 된다.
  - 사용자 정의 인터페이스 상속

```
public interface MemberRepository extends JpaRepository<Member, Long>, MemberRepositoryCustom {

}
```

- 만약 사용자 정의 구현 클래스 이름 끝에 Impl 대신 다른 이름을 붙이고 싶으면 repository-impl-postfix 속성을 변경하면 된다.

12.7 Web 확장 p.558

- 스프링 데이터 프로젝트는 스프링 MVC에서 사용할 수 있는 편리한 기능을 제공한다.

12.7.1 설정

- 스프링 데이터가 제공하는 Web 확장 기능을 활성화하려면 SpringDataWebConfiguration을 스프링 빈으로 등록하면 된다.

```
<bean class="org.springframework.data.web.config.SpringDataWebConfiguration" />
```

- javaConfig를 사용하면 `@EnableSpringDataWebSupport` 어노테이션을 사용하면 된다.

```
@Configuration
@EnableWebMvc
@EnableSpringDataWebSupport
public class WebAppConfig {
    ...
}
```

  - 설정을 완료하면 도메인 클래스 컨버터와 페이징과 정렬을 위한 `HandlerMethodArgumentResolver`가 스프링 빈으로 등록된다.

12.7.2 도메인 클래스 컨버터 기능

- 도메인 클래스 컨버터는 HTTP 파라미터로 넘어온 엔티티의 아이디로 엔티티 객체를 찾아서 바인딩해준다.
- 도메인 클래스 컨버터 적용

```
@Controller
public class MemberController {
    @AutoWried MemberRepository memberRepositry;

    @RequestMapping("member/memberUpdateForm")
    public String memberUpdateForm(@RequestParam("id") Member member, Model model) {
        model.addAttribute("member", member);
        return "member/memberSaveForm";
    }
}
```

- 도메인 클래스 컨버터가 중간에 동작해서 아이디를 회원 엔티티 객체로 변환해서 넘겨준다.
- 도메인 클래스 컨버터는 해당 엔티티와 관련된 레포지토리를 사용해서 엔티티를 찾는다.

> 💡 도메인 클래스 컨버터를 통해 넘어온 회원 엔티티를 컨트롤러에서 직접 수정해도 실제 데이터 베이스에는 반영되지 않는다.

12.7.3 페이징과 정렬 기능 p.561

- 스프링 MVC 에서 스프링 데이터가 제공하는 페이징과 정렬 기능을 편리하게 사용할 수 있도록 `HandlerMethodArgumentResolver`를 제공한다.
- 페이징 기능 : `PageableHandlerMethodArgumentResolver`
- 정렬 기능 : `PageableHandlerMethodArgumentResolver`

페이징과 정렬 예제

```
@RequestMapping(value = "/members", method = RequestMethod.GET)
public String list(Pageable pageable, Model model){ // 파라미터로 Pageable을 받는다. Pageable은 요청 파라미터 정보로 만들어진다.
    Page<Member> page = memberService.findMembers(pageable);
    model.addAttribute("members", page.getContent());
    return "members/memberList";
}
```

- ex) `/members?page=0&size=20&sort=name, desc&sort=address.city`

접두사

- 사용해야 할 페이징 정보가 둘 이상이면 접두사를 사용해 구분 가능
- `@Qualifier` 사용.
- `접두사명_` 으로 구분

```
public String list(
    @Qualifier("member") Pageable memberPageable,
    @Qualifier("order") Pageable orderPageable, ...
)
```

- ex) `/members?member_page=0&order_page=1`

기본값

- `Pageable`의 기본값은 page=0, size=20 이다.
- 기본값 변경 하려면 `@PageableDefault` 어노테이션을 사용한다.

```
@RequestMapping(value = "/members_page", method = RequestMethod.GET)
public String list(@PageableDefault(size=12, sort="name",
  direction=Sort.Direction.DESC) Pageable pageable) {
    ...
}
```

12.8 스프링 데이터 JPA 가 사용하는 구현체 p.562

- 스프링 데이터 JPA가 제공하는 공통 인터페이스는 `org.springframework.data.jpa.repository.support.SimpleJpqRepository` 클래스가 구현한다.
- **@Repository 적용**
  - JPA 예외를 스프링이 추상화한 예외로 변환한다.
- **@Transactional 적용**

- JPA의 모든 변경은 트랜잭션 안에서 이루어져야 한다.
- 스프링 데이터 JPA가 제공하는 공통 인터페이스를 사용하면 데이터 변경 메소드에 `@Transactional` 로 처리되어 있다.

- **@Transactional(readOnly = true)**
  - 데이터를 조회하는 메소드에는 `readOnly = true` 옵션이 적용되어 있다.
- **save() 메소드**
  - 저장할 엔티티가 새로운 엔티티면 저장(persist), 이미 있는 엔티티면 병합(merge)
  - **새로운 엔티티 판단 기본 전략은 엔티티의 식별자로 판단**
    - 식별자가 객체일 때, null 이면 새로운 엔티티로 판단.
    - 식별자가 자바 기본 타입일 때, 숫자 0 값이면 새로운 엔티티로 판단.

12.9 JPA 샵에 적용 p.564

12.10 스프링 데이터 JPA 와 QueryDSL 통합 p.572

- 스프링 데이터 JPA는 2가지 방법으로 QueryDSL 을 지원한다.
  - org.springframwork.data.querydsl.QueryDslPredicateExecutor
  - org.springframwork.data.querydsl.QueryDslRepositorySupport

12.10.1 QueryDslPredicateExecutor 사용 p.572

- 다음처럼 리포지토리에서 `QueryDslPredicateExecutor` 를 상속받으면 된다.

```
public interfavce ItemRepository extends JpaRepository<Item, Long>, QueryDslPredicateExecutor<Item> {

}
```

- QueryDSL 사용 예제

```
QItem item = QItem.item;
Iterable<Item> result = itemRepository.findAll(item.name.contains("장난감").and(item.price.between(1000,2000)));
```

- `QueryDslPredicateExecutor` 인터페이를 보면 QueryDSL을 검색조건으로 사용하면서 스프링 데이터 JPA가 제공하는 페이징 정렬 기능도 함께 사용할 수 있다.

- QueryDslPredicateExecutor 인터페이스

```
public interface QueryDslPredicateExecutor<T> {
    T findOne(Predicate predicate);
    Iterable<T> findAll(Predicate predicate);
    Iterable<T> findAll(Predicate predicate, OrderSpecifier<?>... order);
    Page<T> findAll(Predicate predicate, Pageable pageable);
```

```
    long count(Predicate predicate);
}
```

- `QueryDslPredicateExecutor`는 스프링 데이터 JPA에서 편리하게 QueryDSL을 사용할 수 있지만 기능에 한계가 있는데, join, fetch를 사용할 수가 없다. (JPQL에서 이야기하는 묵시적 조인은 가능)
- QueryDSL이 제공하는 다양한 기능을 사용하려면 JPAQuery를 직접 사용하거나 스프링 데이터 JPA가 제공하는 `QueryDslRepositorySupport`를 사용해야 한다.

12.10.2 QueryDslRepositorySupport 사용 p.573

- QueryDSL의 모든 기능을 사용하려면 JPAQuery 객체를 직접 생성해서 사용하면 된다.
- `QueryDslRepositorySupport`를 상속 받아 사용하면 조금 더 편리하게 QueryDSL를 사용할 수 있다.

- CustomOrderRepository 사용자 정의 레파지토리

```
public interface CustomOrderRepository {
    public List<Order> search(OrderSearch orderSearch);
}
```

- 스프링 데이터 JPA가 제공하는 공통 인터페이스를 직접 구현할 수가 없기 때문에 `CustomOrderRepository`라는 사용자 정의 레포지토리를 만들었다.

- `QueryDslRepositorySupport`를 사용해서 QueryDSl로 구현한 예제 p.574
- 검색 조건에 따라 동적으로 쿼리를 생성한다.
- 참고로 생성자에서 QueryDslRepositorySupport에 엔티티 클래스 정보를 넘겨주어야 한다.

- QueryDslRepositorySupport 의 핵심 기능 p.575