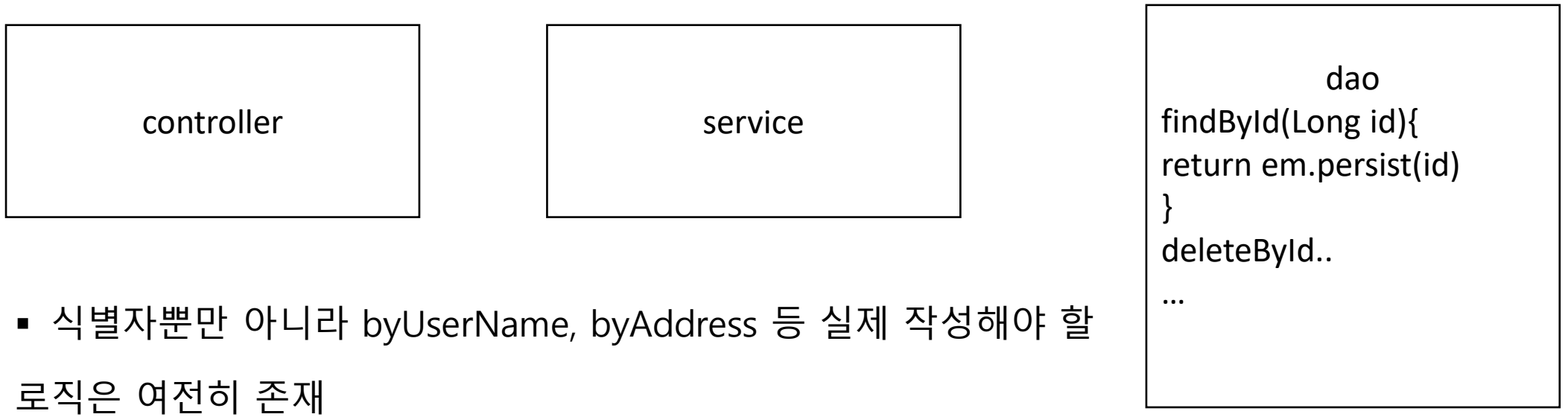
Spring Data JPA 1

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

Spring Data JPA

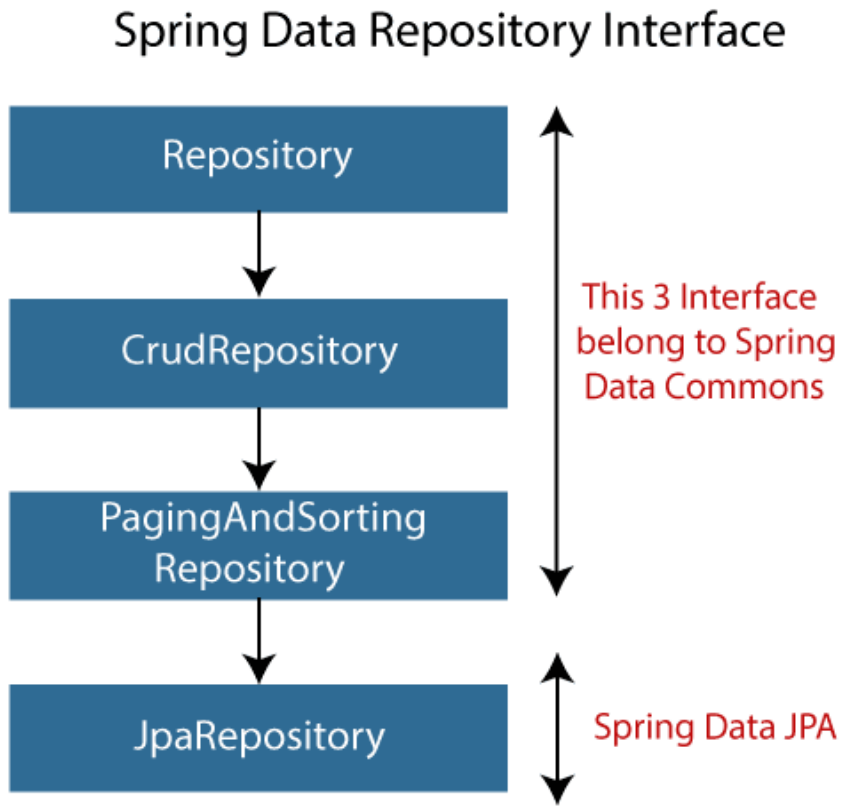
- **개요**
  - JPA를 통해 엔티티를 생성하면 기본 CRUD는 제공해주지만 이를 곧바로 service 계층에서 사용하지 않고 이를 조합하여 DAO에서 각각의 메소드를 생성해야 함



  - 식별자뿐만 아니라 byUserName, byAddress 등 실제 작성해야 할 로직은 여전히 존재
  - + 엔티티 매니저, 엔티티 매니저 팩토리, 트랜잭션 관리 등, JPA를 사용하기 위한 공통적인 절차가 필요
  - Spring Data JPA는 JPA를 쉽게 사용할 수 있도록 돕는 모듈

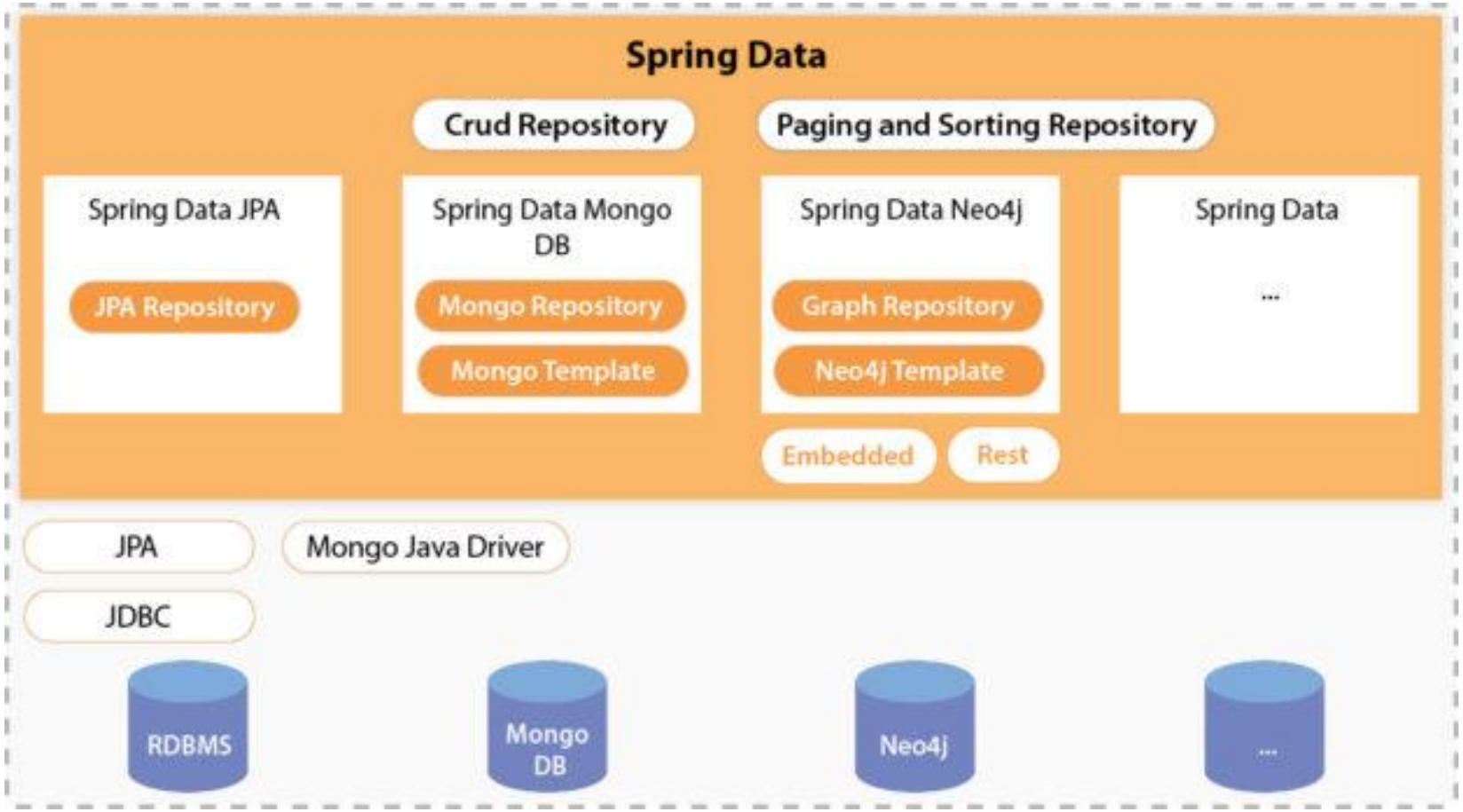
Repository

- **개요**
  - Spring Data JPA를 학습하면 CrudRepository와 JpaRepository를 사용하는 예시들이 있음
  - Repository: 최상위 인터페이스
  - CrudRepositry: CRUD 관련 기능 명세
  - PagingAndSortingRepository: 페이징 및 sorting 관련
  - JpaRepositry: JPA 특화 기능



Repository

- 개요



spring-data에 포함된 다양한 모듈: https://spring.io/projects/spring-data

https://innovationm.co/spring-data-jpa/

CrudRepository

- **CrudRepository Interface**
  - CRUD와 관련된 다양한 기능 명세

```
public interface CrudRepository<T, ID> extends Repository<T, ID> {

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

  Optional<T> findById(ID primaryKey);

  Iterable<T> findAll();

  long count();

  void delete(T entity);

  boolean existsById(ID primaryKey);

  // … more functionality omitted.
}
```

CrudRepository

- **Example**
  - 인터페이스만 정의하면 데이터 JPA가 필요한 구현체를 만들어서 빈으로 등록
  - 데이터 JPA가 기본적으로 제공하지 않는 쿼리에 대해서만 추가적으로 멤버함수로 정의

```
interface UserRepository extends CrudRepository<User, Long> {

   long countByLastname(String lastname);
}
```

```
interface UserRepository extends CrudRepository<User, Long> {

   long deleteByLastname(String lastname);

   List<User> removeByLastname(String lastname);
}
```

  - 위에서 User는 엔티티, Long은 엔티티의 식별자 타입
  - countByLastname, deleteByLastname → 쿼리 메소드

PagingAndSortingRepository

- **PagingAndSortingRepository Interface**
  - 페이징과 정렬 기능 관련

```
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {

  Iterable<T> findAll(Sort sort);

  Page<T> findAll(Pageable pageable);
}
```

```
PagingAndSortingRepository<User, Long> repository = // … get access to a bean
Page<User> users = repository.findAll(PageRequest.of(1, 20));
```

Query Creation

- 인터페이스에 쿼리 메소드를 정의하여 비즈니스 필요한 쿼리 생성

```
public interface UserRepository extends Repository<User, Long> {

   List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);
}
```

```
@Autowired
private UserRepository userRepository;
..
userRepository. findById()
..
userRepository. findByEmailAddressAndLastname
```

```
select u from User u where u.emailAddress = ?1 and u.lastname = ?2
```

케이스 유의

Query Creation

- **Query Creation 생성**

```
interface PersonRepository extends Repository<Person, Long> {

  List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
  List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```

지원하는 키워드 보기 → https://docs.spring.io/spring-data/jpa/docs/1.4.1.RELEASE/reference/html/jpa.repositories.html

Query Creation

- **Query Creation 생성**
  - 쿼리 메소드 이름을 파싱하는 것은 두 단계로 구분
    - find…By, exists…By: 쿼리의 목적 정의
    - 두 번째 파트에 의해 쿼리가 형성

findByEmailAddressAndLastname

Keyword
find…By, read…By, get…By, query…By, search…By, stream…By
exists…By
count…By
delete…By, remove…By
…First<number>…, …Top<number>…
…Distinct…

Property Expressions

- **엔티티의 객체이름으로 쿼리 생성**
  - Query Creation에서 언급한 바와 더불어 중첩된 속성에 대한 쿼리 생성 가능

```
List<Person> findByAddressZipCode(ZipCode zipCode);
```

  - Person은 Address를 가짐. Address는 ZipCode를 가짐 → p.address.zipCode 속성 탐색
    - 최초에는 엔티티에 AddressZipCode가 있는지를 탐색. 있다면 이를 이용하여 쿼리생성
    - 없다면 AddressZipCode를 Address.Zipcode로 해석(카멜 케이스에 따라)
      - 사실 AddressZip 와 Code와 같이 중간 과정에서 가능성이 있는 모든 케이스 검사
  - 표현의 명확성을 위해 아래와 같이 메소드 이름에 하이픈 사용 가능

```
List<Person> findByAddress_ZipCode(ZipCode zipCode);
```

Special parameter handling

- **Pageable, Slice, and Sort**
  - Pageable과 Sort라는 특별한 타입이 존재

```
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);
```

Special parameter handling

- **Pageable 예제**
  - Repository 생성

```
public interface ProductRepository extends PagingAndSortingRepository<Product, Integer> {

    List<Product> findAllByPrice(double price, Pageable pageable);
}
```

  - Repository 사용

```
Pageable firstPageWithTwoElements = PageRequest.of(0, 2); #페이지의 size를 2로 설정했을 때 첫 번째 페이지
Pageable secondPageWithFiveElements = PageRequest.of(1, 5); #페이지의 size를 5로 설정했을 때 두 번째 페이지

Page<Product> allProducts = productRepository.findAll(firstPageWithTwoElements);

List<Product> allTenDollarProducts =
        productRepository.findAllByPrice(10, secondPageWithFiveElements);
```

  - PagingAndSortingRepository의 상속으로 new Pageable(page, size) 는 Deprecated 됨
    - indAll(Pageable pageable), findAll(Sort sort) 사용 가능

https://www.baeldung.com/spring-data-jpa-pagination-sorting

Special parameter handling

- **PageRequest.of()**
  - Pagenation은 Sorting을 함께 제공

of()	설명
PageRequest.of(int page, int size)	페이지 번호(0부터 시작), 페이지당 데이터의수
PageRequest.of (int page, int size, Sort.Direction direction, String ...props)	페이지 번호, 페이지당 데이터의 수, 정렬 방향, 속성(칼럼)들
PageRequest.of(int page, int size, Sort sort)	페이지 번호, 페이지당 데이터의 수, 정렬방향

Special parameter handling

- **반환 타입에 따른 페이징 결과**
  - Page<T>: 총 레코드 수를 포함한 페이징 결과를 받을 때
    - 사용 가능한 데이터의 총 개수(count)와 전체 페이지 수 파악 가능
    - 예를 들어 게시판의 페이징의 경우 전체 페이지 수를 알아야 화면에 표시할 수 있음
  - Slice<T>: Page extends Slice
    - count 쿼리는 지양하는 것이 좋음
    - 예를 들어 테이블에 특정 레코드의 저장 유무를 파악하기 위해 count쿼리의 결과가 1인지 확인하는 것보다 exsists를 사용하는 것이 성능상 좋음
    - 단지 다음 페이지가 존재하는지 유무만 파악: 쇼핑몰에 더보기와 같은 기능
  - List<T>: 주어진 범위에 존재하는 엔티티만 반환