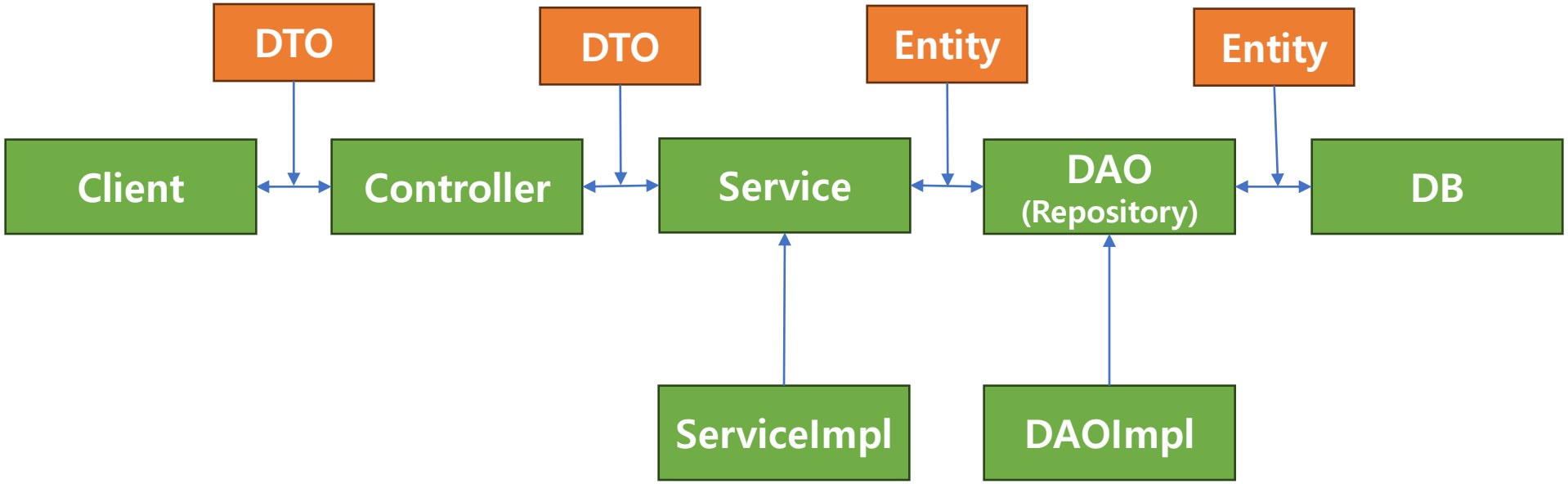
05

SpringBoot 계층(Layer)

Spring Boot 서비스 구조



01. Controller Layer

- Client로 부터 요청(Request)을 전달 받아 여러 다른 Layer 을 사용하여 응답(Response)객체를 만들어 다시 Client로 전달하는 Layer
- 전체적인 흐름을 제어(Control)하는 Layer

02. Entity Layer

- 데이터베이스의 Table과 1:1로 mapping
- 이 클래스의 멤버변수는 Table의 필드(Column)을 의미

```
@Entity
@Table(name = "producttbl")
@Setter
@Getter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String title;
    private String imgsrc;
    private Integer price;
}
```

03. Repository Layer

- DB에 접근하는 메소드를 정의한 인터페이스
- Service와 DB를 연결하는 고리역할
- DB에 적용하고자 하는 CRUD를 정의하는 영역
- JpaRepository Interface를 상속받으면 기본 CRUD 메소드를 사용할 수 있음
- 현재 Interface에 추가 메소드를 정의할 수 있음

```
package com.react_spring_exam_project.repository;

import com.react_spring_exam_project.data.entity.Product;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProductRepository extends JpaRepository<Product, Integer> {

}
```

03. Repository Layer

JpaRepository Interface를 상속받으면 사용 가능한 기본 메소드

- T save(T entity):
  - 엔티티를 저장하거나, 이미 존재하는 경우 업데이트
- List<T> saveAll(Iterable<T> entities):
  - 여러 엔티티를 저장.
- Optional<T> findById(ID id)
  - 주어진 ID에 해당하는 엔티티를 조회. 없으면 Optional.empty()를 반환
- boolean existsById(ID id)
  - 주어진 ID에 해당하는 엔티티가 존재하는지 확인.
- List<T> findAll()
  - 모든 엔티티를 조회.

03. Repository Layer

JpaRepository Interface를 상속받으면 사용 가능한 기본 메소드

- List<T> findAllById(Iterable<ID> ids)
  - 주어진 ID 리스트에 해당하는 엔티티들을 조회.
- long count()
  - 총 엔티티의 개수를 반환.
- void deleteById(ID id)
  - 주어진 ID에 해당하는 엔티티를 삭제.
- void delete(T entity)
  - 해당 엔티티를 삭제.
- void deleteAll(Iterable<? extends T> entities)
  - 주어진 엔티티 리스트를 삭제.
- void deleteAll()
  - 모든 엔티티를 삭제.

03. Repository Layer

- 현재 Interface에 추가 메소드를 정의할 때는 Spring Data JPA 메서드 네이밍 규칙을 따라 작성하면 메소드를 직접 구현하지 않아도 Spring이 동적으로 생성해 줌

  - 주제 키워드
    - findBy: 특정 조건을 만족하는 엔터티를 조회.
    - readBy: findBy와 동일하게 조회 작업을 수행.
    - queryBy: findBy와 동일한 기능을 제공, 읽기 작업에 사용
    - getBy: findBy와 비슷하게 사용.
    - countBy: 특정 조건을 만족하는 엔터티의 개수를 반환.
    - existsBy: 특정 조건을 만족하는 엔터티가 존재하는지 여부를 반환
    - deleteBy / removeBy: 특정 조건을 만족하는 엔터티를 삭제

03. Repository Layer

- 조건 키워드
  - Is / Equals: 특정 값과 일치하는지 확인
    - (findByNameIs, findByNameEquals)
  - IsNot / Not: 특정 값과 일치하지 않는지 확인
    - (findByNameIsNot, findByNameNot)
  - Between: 특정 범위 내의 값 찾기
    - (findByAgeBetween)
  - LessThan / LessThanEqual: 특정 값보다 작은지 확인
    - (findByAgeLessThan, findByAgeLessThanEqual)
  - GreaterThan / GreaterThanEqual: 특정 값보다 큰지 확인
    - (findByAgeGreaterThan, findByAgeGreaterThanEqual)
  - After: 특정 날짜 이후인지 확인
    - (findByCreatedAtAfter)
  - Before: 특정 날짜 이전인지 확인
    - (findByCreatedAtBefore)

03. Repository Layer

- 조건 키워드(계속)
  - IsNull / IsNotNull: 값이 null인지 또는 null이 아닌지 확인
    - (findByAddressIsNull, findByAddressIsNotNull)
  - Like: 문자열 패턴을 이용한 검색
    - (findByNameLike)
  - StartingWith: 특정 문자열로 시작하는지 확인
    - (findByNameStartingWith)
  - EndingWith: 특정 문자열로 끝나는지 확인
    - (findByNameEndingWith)
  - Containing: 특정 문자열을 포함하는지 확인
    - (findByNameContaining)
  - In: 특정 컬렉션에 포함되는지 확인
    - (findByAgeIn)
  - NotIn: 특정 컬렉션에 포함되지 않는지 확인
    - (findByAgeNotIn)

03. Repository Layer

- **논리 연산자 사용**
  - **And: 두 조건을 모두 만족하는 경우**
    - (findByNameAndAge)
  - **Or: 두 조건 중 하나만 만족해도 되는 경우**
    - (findByNameOrAge)
    - findByNameAndAgeGreaterThan : name이 일치하고 age가 지정한 값보다 큰 엔터티를 조회.
- **정렬 키워드**
  - **메서드 이름 끝에 OrderBy와 필드명, 정렬 방향(Asc, Desc)을 추가하여 사용**
  - **OrderBy: 지정된 필드로 정렬하여 결과 반환**
    - (findByNameOrderByAgeAsc)
    - findByNameOrderByAgeDesc : name이 일치하는 엔터티를 age 기준으로 내림차순 정렬하여 반환

03. Repository Layer

- **제한 (Top, First)**
  - **Top 또는 First 키워드를 사용하여 결과 개수를 제한.**
  - **Top: 지정한 상위 개수만 반환**
    - (findTop3ByNameOrderByAgeDesc)
  - **First: 지정한 상위 개수 중 첫 번째 항목 반환**
    - (findFirstByNameOrderByAgeAsc)
    - findTop3ByNameOrderByAgeDesc : name이 일치하는 항목을 age 기준으로 내림차순 정렬한 후, 상위 3개의 결과만 반환

03. Repository Layer

```
List<Product> findByNameAndCategory(String name, String category);
// name과 category 조건 모두 만족

List<Product> findByPriceBetween(Double minPrice, Double maxPrice);
// price가 minPrice와 maxPrice 사이인 제품

List<Product> findByCreatedAtBefore(Date date);
// 특정 날짜 이전에 생성된 제품

List<Product> findByNameContaining(String keyword);
// name에 특정 키워드 포함

List<Product> findTop5ByCategoryOrderByPriceAsc(String category);
// category로 상위 5개의 제한
```

04. DAO(Data Access Object) Layer

- DB에 접근하는 객체를 의미
- Service가 DB에 연결할 수 있게 해 주는 역할
- DB를 사용하여 데이터를 조작하는 기능을 전담
- Repository를 다루는 기능을 감싸는 객체
- Repository자체를 DAO로 사용하기도 함

04. DAO(Data Access Object) Layer

```
@Service
@RequiredArgsConstructor
public class ProductDAO {
    private final ProductRepository productRepository;

    public List<Product> getProductList() {
        List<Product> productList = this.productRepository.findAll();
        return productList;
    }

    public Product saveProduct(String title, String imgsrc, Integer price) {
        Product product = Product.builder()
                .title(title)
                .price(price)
                .imgsrc(imgsrc).build();
        this.productRepository.save(product);
        return product;
    }
}
```

05. DTO (Data Transfer Object) Layer

- **DTO : 데이터 전송 객체**
- **계층 간 데이터를 교환할 때 사용되는 객체**
  - 프론트엔드와 백엔드 간 데이터 교환 또는 서비스 간 데이터 전송에 사용

```
@Setter
@Getter
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class ProductDTO {
    private Integer id;
    private String title;
    private String imgsrc;
    private int price;
}
```

06. Service Layer

- 애플리케이션의 비즈니스 로직을 처리하는 핵심 계층
- 일반적으로 Controller와 DAO(Repository) 사이에 위치하며, 애플리케이션의 주요 기능과 규칙을 수행
  - DAO(Repository) Layer와 상호작용하며, 데이터를 CRUD하거나 가공하여 필요한 정보를 제공
  - Controller는 DB의 세부적인 접근 방식에 구애받지 않고, Service 메서드를 통해 필요한 작업을 요청
  - Service Layer를 두어 Controller와 데이터 접근 코드를 분리함으로써 코드의 유지보수성과 재사용성을 향상

- @Service를 사용
  - Spring 컨테이너가 해당 클래스를 빈(bean)으로 자동 등록
  - 의존성 주입을 통해 사용할 수 있게 함

06. Service Layer

```
@Service
@RequiredArgsConstructor
public class ProductService {
    private final ProductRepository productRepository;

    public void save(String title, Integer price) {
        Product product = new Product();
                product.setTitle(title);
                product.setPrice(price);
        this.productRepository.save(product);
    }
}
```

```
@PostMapping("/new-product")
String writePost(@RequestParam String title,
                 @RequestParam Integer price) {
    this.productService.save(title, price);
    return "redirect:/list";
}
```