



스프링 컨벤션

1. 구조 (Structure)

1.1. 기능, 목적, 자원 별로 상위 패키지 하나를 두어 관리한다.

1.2. 목적별 패키지 내용

- 하나의 상위 패키지에는 아래 내용들이 포함 될 수 있다.
 - Controller
 - Service
 - Repository
 - Dto
 - Entity
-

2. 컨트롤러 (Controller)

2.1. 클래스 네이밍 규칙

- 각 컨트롤러의 앞에 상위 패키지 이름을 붙여서 클래스의 이름을 정한다.
 - Ex. `UserController`, `BoardController`

2.2. 메서드 네이밍 규칙

- 각 컨트롤러의 메서드명은 다음과 같은 접미사를 붙인다.
 - `userList()` – 여러건의 정보에 대한 조회
 - `userDetails()` – 한 건의 정보에 대한 조회
 - `userAdd()` – 정보 추가
 - `userModify()` – 정보 수정
 - `userRemove()` – 정보 삭제

2.3. 역할

- 컨트롤러에서는 Service 호출, 예외 처리, 클라이언트와의 연결만을 담당한다.
-

3. 서비스 (Service)

3.1. 클래스 네이밍 규칙

- 도메인명의 서비스 생성은 피한다.
 - Order 라는 도메인이 있을 때 OrderService 로 만드는 것은 피한다.
 - OrderService로 만들 경우 그 안에 도메인과 관련된 여러 기능을 넣을 가능성이 높다.
 - 도메인 관련 기능을 세분화하여 서비스를 만든다.
 - Ex. `OrderRegisterService` , `OrderStatusService`

3.2. 메서드 네이밍 규칙

- 각 서비스의 메서드 명은 다음과 같은 접두사를 붙인다.
 - `findUser()` – 정보에 대한 조회
 - `addUser()` – 정보 추가
 - `modifyUser()` – 정보 수정
 - `removeUser()` – 정보 삭제

3.3. 역할

- 서비스에서는 비즈니스 로직을 모두 담당하며 DB와의 연결을 위해 Repository를 호출한다.
 - DTO ↔ Entity 변환은 서비스에서만 해준다.
-

3. 레포지토리 (Repository)

3.1. 클래스 네이밍 규칙

- 각 레포지토리의 앞에 상위 패키지 이름을 붙여서 클래스의 이름을 정한다.
 - Ex. `UserRepository` , `BoardRepository`

3.2. 메서드 네이밍 규칙

- 메서드 네이밍 규칙은 `JpaRepository` 의 규칙을 따른다.

- Ex. `save`, `findById`

3.3. 역할

- 각 Repository는 `JpaRepository`를 상속받은 인터페이스로 만든다.

4. DTO (Data Transfer Object)

4.1. 클래스 네이밍 규칙

- 각 DTO 앞에 이름을 붙여서 클래스 이름을 정한다.
- Ex. `UserDto`, `BoardDto`

4.2. 역할

- 각 계층간 데이터 교환에서 사용하기 위한 객체이다.
- Entity를 바로 보내는 것이 아닌 Dto로 변환하여 사용하도록한다.

4.3. 조건

- 비즈니스 로직을 갖지 않고 단순 데이터 전달의 역할만 하도록한다.
- 기본적으로 `Getter`를 가지고 빌드패턴을 활용할 수 있도록 모든 파라미터를 이용한 생성자에 `@Builder` 어노테이션을 이용한다.
- 필요에 따라 `Setter`를 생성해도 된다.
- `Entity`로 변환이 필요한 경우 `toEntity` 메소드를 만들어 활용한다.

예시

```
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class CategoryDto {
    private Long id;
    private String name;

    public Category toEntity(){
        return Category.builder()
            .id(id)
            .name(name)
            .build();
    }
}
```

5. Entity

5.1. 클래스 네이밍 규칙

- 특별한 접두사, 접미사 없이 이름을 정한다.
- Ex. `User`, `Board`

5.2. 역할

- DB의 테이블과 1대1 매칭되는 클래스이다

5.3. 조건

- 기본적으로 `Getter` 를 가지고 빌드패턴을 활용할 수 있도록 모든 파라미터를 이용한 생성자에 `@Builder` 어노테이션을 이용한다.
- 새로운 데이터 생성을 위해 `Auto Increase` 속성을 가지는 id 값을 제외한 파라미터를 가지는 생성자를 만들 수 있다.
- `Setter` 는 사용하지 않는다.

예시

```
@NoArgsConstructor
@Getter
@Entity
public class Category {

    @Id
    @GeneratedValue
    private Long id;

    @Column(unique = true)
    private String name;

    @Builder
    public Category(Long id, String name){
        this.id = id;
        this.name = name;
    }

    public Category(String name){
        this.name = name;
    }
}
```


▼ 정해야 하는 것

예외처리 방식 (핸들러??)

에러메시지 Dto, Enum (에러메시지 형식) 만들지

HTTP Response Body에 예외 error message 지정해 보내기

Http Response Body에 Exception error message 지정하기 직접 만든 에러메시지를 Http Response의 "message" :에 넣어 줄 수 있다. 먼저 Exception과 에러메시지를 만든다. DishController.java // ...

 <https://bibi6666667.tistory.com/289>



Springboot Exception Handling(스프링부트 exception 핸들링)

스프링부트에서 exception을 처리하는 방법을 알아보자 순서는 다음과 같다 1. 에러코드 정리 enum 클래스로 작성 2. Exception 발생시 응답하는 에러 정보 클래스 작성 3. 사용자 정의 Exception 클래스 작성 4.

☹ <https://samtao.tistory.com/42>

