라이브 세션-2022.12.21(수)-예외 처리

✅ 들어온 질문 공유

1️⃣ 명시적으로 처리하지 않는 모든 예외를 `Exception` 으로 한 번에 처리하는게 마음에 걸립니다.

- 개발하면서 자주 발생하는 예외는 미리 처리 로직을 짜면 된다.
    - 예상치 않는 예외가 발생할 수도 있다. 그때 추가 처리 로직을 구성하자.
- 그 외에는 Exception으로 받자. 다른 방법이 없음.

✅ 지난 시간 학습 리뷰

1️⃣ Controller 리뷰

2️⃣ DTO 리뷰

3️⃣ DI와 Mapper를 이용한 Controller와 Service 클래스의 연동 리뷰

✅ 이번 시간 이야기 나눌 내용

1️⃣ 체크 예외(Checked Exception)와 언체크 예외(Unchecked Exception)

- 체크 예외
  - 반드시 체크 해야 하는 예외
    - `SQLException`, `ClassNotFoundException` 등
- 언체크 예외
  - 캐치(catch)할 필요 없는 예외
    - `RuntimeException`

2️⃣ Controller에서 @ExceptionHandler 사용의 장/단점

- 모든 Controller마다 @ExceptionHandler를 사용해서 예외를 catch하게 되면?
  - 사용하기는 쉽다.
  - 하지만 똑같은 타입의 Exception을 Controller 마다 catch해야 하므로 중복 코드가 발생한다.

3️⃣ @RestControllerAdvice를 이용한 예외처리

- Controller에서 중복되는 예외 처리 코드를 공통화 할 수 있도록 해준다.
- `@RestControllerAdvice`를 여러개 정의해서 비슷한 유형의 Exception을 그룹화할 수 있다.
- 단, 너무 많은 `@RestControllerAdvice`를 사용하기 보다는 적절한 개수를 잘 정해서 사용하자.
- HTTP Status가 동적으로 변할 수 있는 경우에는 응답으로 ResponseEntity를 이용할 수 있다.
- HTTP Status가 항상 고정되어 있는 경우에는 `@ResponseStatus` 애너테이션을 사용할 수 있다.

4️⃣ Error 전용 Response를 사용하는 이유

- 고객(Client)에게 조금 더 친절한 Server가 되는 것이 좋겠죠?
- Exception 클래스 자체에서 가지고 있는 에러 내용을 가공처리 해 클라이언트에게 제공하기 위함
  - 조금 더 친절하게
  - 조금 더 일관성 있게

5️⃣ of() 등의 정적 팩토리 메서드(static factory method)

- **정적 팩터리 메서드란?**
  - new를 직접적으로 사용하지 않고, 클래스의 인스턴스를 생성하는 방법

- 정적 팩토리 메서드의 장점
  - **이름을 통해 객체의 의미를 쉽게 알 수 있다.**

    예)
    - Stream.of(1, 2, 3)
      - a stream that is made **of** the numbers 1, 2 and 3
    - List.of(1, 2, 3, 4)
      - a list that is made **of** the numbers 1, 2, 3, and 4
    - `ErrorResponse of(BindingResult bindingResult)`
      - a ErrorResponse instance is made **of** BindingResult object
    - `ErrorResponse of(Set<ConstraintViolation<?>> violations)`
      - a ErrorResponse instance is made **of** BindingResult object

  - **매 번 새로운 객체를 생성할 수 도 있고, 아닐 수 도 있다.**
    - `Boolean.TRUE`
    - 새로운 객체를 생성할 때 제약 조건을 걸 수도 있다.

- **하위 타입 객체 반환 가능**

```
public class Foo {

    private Foo() {

    }

    public static Foo foo() {
        return new Foo();
    }

    public static Foo fooBar() {
        return new Bar();
    }

    private static class Bar extends Foo {

        public Bar() {

        }
    };
};
```

- 정적 팩터리 메서드 작성 시점에 반환될 객체의 클래스가 존재하지 않아도 됨
  - SPI(Service Provider Interface)에서 많이 사용
  - 예)
    - Class.forName(”com.mysql.jdbc.Driver”)

5️⃣ 비즈니스적인 예외 throw/catch

- **백엔드 서버와 외부 시스템과의 연동에서 발생하는 에러 처리**
- **시스템 내부에서 조회하려는 리소스(자원, Resource)가 없는 경우 등의 의도적인 Exception throw**

6️⃣ Custom Exception 사용

- 우리가 작성한 `BusinessLogicException` 같은 예외 클래스
- 목적에 맞는 Custom Exception 클래스가 늘어날 수는 있다.
    - 수정하는데 비용이 많이 들지 않는다면 클래스 개수가 늘어나는 것에 부담은 갖지 마세요.

7️⃣ 비즈니스적인 예외 처리에 대한 Worse case 예제 코드 리뷰

✅ 실습 과제 설명

- 과제 1
- 과제 2
- 과제 3