가비아 인턴 개인 과제 수행 회고(4)

👨‍💻 *Endo(안태욱)*
2023-01-13

🛠️ **예외 처리 및 리팩토링**

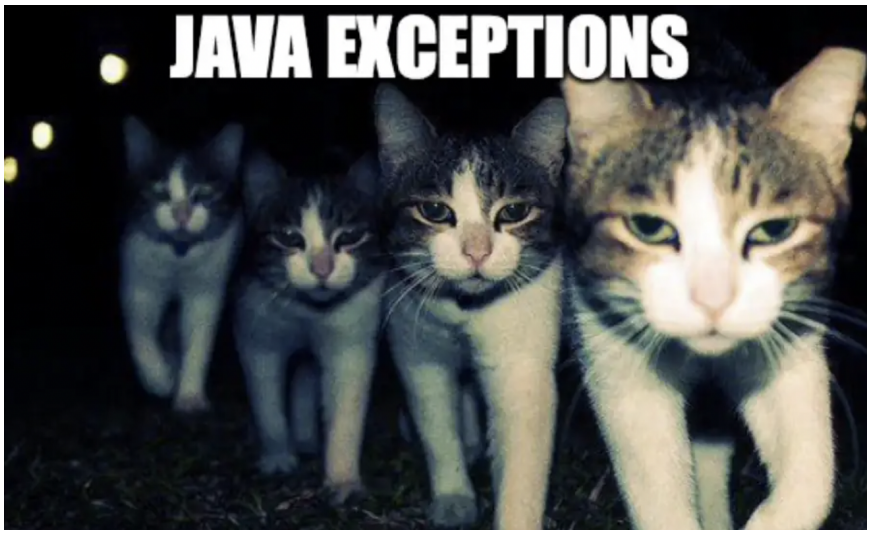
가비아 인턴 과제를 마무리하는 마지막 날로,

필요한 기능 구현은 완료하였고, 여기 저기서 발생하는 **서버 내의 예외 처리**와

성능 개선 및 가독성을 위한 **리팩토링**을 수행하였습니다.

(+ERD cloud를 활용한 ERD 수정)

마지막 회고도 마찬가지로 작성한 코드를 되짚어 보는 방식으로 진행하겠습니다.



예외 처리

사실 저는 서버 개발에 있어 예외 처리를 해본 경험이 없었습니다..

만약 예외나 에러가 발생했을 경우,

해당 *throwable* 이 논리적으로 클라이언트의 잘못인지, 서버의 잘못인지를 구별하며

처리하지 않았고,

응답 또한 클라이언트가 단순히 서버의 문제로 생각할 수 있도록 **마감을 확실하게** 하지 않았습니다.

그래서 이번 **예외 처리** 구현 시, **여러 방식과 의미도 깊게 생각**해 보고자 많은 시간을 들였습니다.

처리 방식

제가 기존에 알고 있던 예외 처리 방식은 *try~ catch~* 를 통한 직접 처리 방식과,

throw~ 로 상위 계층으로 전달 정도만 알고 있었고,

서버에서 발생 예외를 잡아 클라이언트에게 응답하기 위해

컨트롤러에서 예외 처리를 할 수 있다는 *@ControllerAdvice* 만 알고 있었는데,

Aiden 님 덕분에 스프링 서버 **전역에서 발생하는 예외를 잡아서 클라이언트에게 응답**을 줄 수 있는

글로벌 예외 처리 방식이 있다는 것을 알게 되었습니다.

다음은 해당 방식의 순서입니다.

…

1. 상태코드와 메시지를 포함한 에러코드

제목 그대로 예외가 발생했을 경우 전달할 코드와 메시지를 **Enum type** 으로 정의하는데,

우선 **에러 코드 인터페이스**를 정의하고,

보편적인 예외에 따른 코드인지, 사용자가 정의한 예외에 따른 코드인지를 구별해서 구현하였습니다.

```
public interface ErrorCode {

    HttpStatus getHttpStatus();
    String getMessage();
}
```

```
@Getter
@RequiredArgsConstructor
public enum CommonErrorCode implements ErrorCode {

    INVALID_PARAMETER(HttpStatus.BAD_REQUEST, "Invalid parameter included"),
    RESOURCE_NOT_FOUND(HttpStatus.NOT_FOUND, "Resource not exists"),
    NO_ELEMENT(HttpStatus.NOT_FOUND, "Element not exists"),
    ;

    private final HttpStatus httpStatus;
    private final String message;
}
```

```
@Getter
@RequiredArgsConstructor
public enum UserErrorCode implements ErrorCode {

    EMPTY_CLAIM(HttpStatus.UNAUTHORIZED, "No authentication information in claim"),
    DUPLICATED_ID(HttpStatus.CONFLICT, "ID already exists"),
    WRONG_PERIOD(HttpStatus.BAD_REQUEST, "Not voting period"),
    EXCEED_VOTE(HttpStatus.BAD_REQUEST, "Exceeding the number of possible votes"),
    DUPLICATED_VOTE(HttpStatus.BAD_REQUEST, "Duplicate voting is not allowed")
    ;

    private final HttpStatus httpStatus;
    private final String message;
}
```

공통적인 예외로는

- 스프링 유효성 **검증 단계**에서 (Spring validation) 발생시키는 예외
- 잘못된 ID 를 통한 **자원 조회**에 대해 Optional 에서 기본적으로 발생시키는 예외

두 가지 예외를 다루도록 설계 하였고,

사용자 정의 예외로는

- 토큰에 **클레임 정보가 없을 때** 예외
- **중복된 ID** 로 회원 가입을 요청할 때 예외
- **올바르지 못한 투표 요청**을 보냈을 때 예외

세 가지 예외를 다루도록 설계 하였습니다.

2. 예외 정의

```
@Getter
@RequiredArgsConstructor
public class RestApiException extends RuntimeException {

    private final ErrorCode errorCode;
}
```

```
public class InvalidVoteException extends RestApiException {

    public InvalidVoteException(ErrorCode errorCode) {
        super(errorCode);
    }
}
```

```
public class DuplicateUserException extends RestApiException {

    public DuplicateUserException(ErrorCode errorCode) {
        super(errorCode);
    }
}
```

우선 예외 내 파라미터로 넣을 에러코드를 *RestApiException* 으로 정의하고,
구체적인 예외를 **해당 예외 상속 받는 구조**로 정의하였습니다.

위에서 정의한 에러코드를 파리미터로 전달하면, 예외 객체 내 에러 코드를 가지게 되고,
해당 예외를 처리할 때 **메시지와 상태코드**를 확인할 수 있습니다.

3. 글로벌 예외 처리 핸들러

```
@RestControllerAdvice
public class GlobalExceptionHandler extends ResponseEntityExceptionHandler {

    @ExceptionHandler(RestApiException.class)
    public ResponseEntity<Object> handleCustomException(RestApiException e) {
        ErrorCode errorCode = e.getErrorCode();
        return new ResponseEntity<>(errorCode.getMessage(), errorCode.getHttpStatus());
    }

    @ExceptionHandler(EntityNotFoundException.class)
    public ResponseEntity<Object> handleEntityNotFound(EntityNotFoundException e) {
        ErrorCode errorCode = CommonErrorCode.RESOURCE_NOT_FOUND;
        return new ResponseEntity<>(errorCode.getMessage(), errorCode.getHttpStatus());
    }

    @ExceptionHandler(NoSuchElementException.class)
    public ResponseEntity<Object> handleNoSuchElement(NoSuchElementException e) {
        ErrorCode errorCode = CommonErrorCode.NO_ELEMENT;
        return new ResponseEntity<>(errorCode.getMessage(), errorCode.getHttpStatus());
    }

    @ExceptionHandler(IllegalArgumentException.class)
    public ResponseEntity<Object> handleIllegalArgument(IllegalArgumentException e) {
        ErrorCode errorCode = CommonErrorCode.INVALID_PARAMETER;
        return new ResponseEntity<>(errorCode.getMessage(), errorCode.getHttpStatus());
    }
}
```

최종적으로 서버에서 발생하는 예외를 받아서 처리하는 핸들러 클래스를 작성합니다.

상단에서 정의 했던 *RestApiException* 과 일반적으로 발생할 수 있는 예외에 대해 처리 과정을 작성하면,

정의 했던 에러 코드를 통해 사용자에게 응답을 전달할 수 있게 됩니다.

위 핸들러 구현을 통해 **클라이언트의 잘못된 요청과 서버 내 오류를 구별하여 처리하는 방식으로**

서버를 구성할 수 있습니다.

리팩토링

리팩토링은 크게 두 가지 변경을 통해 진행하였습니다.

1. 투표 시스템 호출 방식 변경

기존의 서비스에서 투표 시스템을 호출하는 방식을 변경하였습니다. (*summer* 님이 발견한 이슈)

```
VotingSystemFactory factory = new VotingSystemFactory(voteRepository);
NormalVotingSystem votingSystem = factory.makeVotingSystem(agenda);
```

기존(위) → 변경(아래)

```
VotingSystem votingSystem = votingSystemFactory.makeVotingSystem(agenda);
```

기존에 서비스에서 **new**로 투표 시스템 팩토리를 생성하고, 팩토리 내에서 투표 시스템도 **new**로

생성하는 방식 이였는데,

사실 투표 팩토리와 시스템 자체가 **무상태(stateless)**로 처리를 하는데,

요청 마다 **new** 로 객체를 생성해주면 자원 낭비로 이어질 수 있기 때문에 투표 시스템과 팩토리를

컴포넌트로 등록하여 스프링 빈에 의해 싱글톤 방식으로 관리될 수 있도록 수정하였습니다.

```
@Component
@Slf4j
@RequiredArgsConstructor
public class NormalVotingSystem implements VotingSystem {

    private final VoteRepository voteRepository;
```

2. 사용자 서비스 수정

기존의 사용자 서비스에서 사용하던 **계층을 통일** 하였습니다. (코드 리뷰)

```
@Transactional
public UserJoinResponseDto create(UserJoinRequestDto userJoinRequestDto) {

    validateUserId(userJoinRequestDto);

    User user = User
            .builder()
            .userId(userJoinRequestDto.getUserId())
            .password(passwordEncoder.encode(userJoinRequestDto.getPassword()))
            .name(userJoinRequestDto.getName())
            .role(userJoinRequestDto.getRole())
            .voteRights(userJoinRequestDto.getVoteRights())
            .build();

    entityManager.persist(user);    // old
    userRepository.directSave(user);     // new

    return UserJoinResponseDto.from(user);
}
```

위 서비스 코드에서 새로 회원 가입 하는 유저 정보를 유저 저장소에 저장하는 과정에서

불필요한 *select* 쿼리를 방지하기 위해 **EntityManager** 를 통해 직접 저장 하였는데,

그렇게 구현 하면 **계층 별 추상화 단계**에 어긋나기 때문에 리포지토리로 해당 메서드를 감싸서

리포지토리 메서드를 사용하도록 수정하였습니다.

결론

4일동안 온라인 투표 시스템을 완성 하였는데,

발표날인 다음주 월요일 전 까지 **API 명세 문서**도 새로 작성하고 **추가적으로 리팩토링** 할 부분을 찾고,

수행할 예정입니다. *(쓰는 시점에도 많은 리팩토링 요소 발견..)*

같은 시스템을 개발하기 위해 여러 분들과 **함께 코드 작성**을 진행하고 **시니어 분들께 리뷰**를 받을 수 있어서

좀 더 좋은 설계와 구현에 대해 고민해 볼 수 있었던 것 같습니다.