# GlobalExceptionHandler 튜토리얼

> 이 프로젝트의 예외 처리 구조를 분석하고, 부족한 부분을 개선하는 가이드

---

## 1. 예외 처리 아키텍처

### 전체 구조

```
_core/
├── errors/
│   ├── GlobalExceptionHandler.java     # 모든 예외를 한 곳에서 처리
│   └── ex/
│       ├── Exception400.java           # 400 Bad Request (유효성/중복)
│       ├── Exception401.java           # 401 Unauthorized (인증 실패)
│       ├── Exception403.java           # 403 Forbidden (권한 없음)
│       ├── Exception404.java           # 404 Not Found (자원 없음)
│       └── Exception500.java           # 500 Internal Server Error (서버 에러)
```

### 동작 원리

```
[어디서든 예외 발생]
throw new Exception401("인증되지 않았습니다.")
    ↓
[Spring이 자동 가로챔]
@RestControllerAdvice → GlobalExceptionHandler
    ↓
[@ExceptionHandler가 예외 타입별로 분기]
Exception401 → ex401() 메서드 실행
    ↓
[script 응답]
alert('인증되지 않았습니다.') + location.href='/login-form'
```

### 핵심 어노테이션

| 어노테이션 | 역할 |
|---|---|
| `@RestControllerAdvice` | 모든 Controller에서 발생하는 예외를 가로채는 전역 핸들러. 리턴 값이 **응답 body**로 직접 나감 |
| `@ExceptionHandler` | 특정 예외 타입을 처리할 메서드 지정 |

> `@ControllerAdvice` vs `@RestControllerAdvice`: 후자는 `@ResponseBody`가 내장되어 있어서 리턴 값이 HTML/JSON 문자열로 바로 응답된다.

---

# 2. 커스텀 예외 클래스 설계

## HTTP 상태 코드 기반 설계

모든 커스텀 예외는 `RuntimeException`을 상속한다:

```
// Exception400.java - 유효성 검사 실패 / 중복
public class Exception400 extends RuntimeException {
    public Exception400(String message) {
        super(message);
    }
}

// Exception401.java - 인증 실패
public class Exception401 extends RuntimeException {
    public Exception401(String message) {
        super(message);
    }
}

// Exception403.java - 권한 없음
public class Exception403 extends RuntimeException {
    public Exception403(String message) {
        super(message);
    }
}

// Exception404.java - 자원 못 찾음
public class Exception404 extends RuntimeException {
    public Exception404(String message) {
        super(message);
    }
}

// Exception500.java - 서버 에러
public class Exception500 extends RuntimeException {
    public Exception500(String message) {
        super(message);
    }
}
```

## 왜 RuntimeException인가?

```
Exception (Checked)
├── IOException, SQLException...          ← try-catch 강제 (컴파일 에러)
└── RuntimeException (Unchecked)
    ├── Exception400, Exception401...     ← try-catch 강제 안 함
    └── NullPointerException...           ← Spring이 자동 처리 가능
```

- `RuntimeException`은 **Unchecked Exception**이라 `throws` 선언 없이 어디서든 throw 가능
- Spring의 `@Transactional`은 RuntimeException 발생 시 **자동 rollback**
- `@ExceptionHandler`가 자동으로 가로채므로 try-catch 불필요

---

## 3. GlobalExceptionHandler 상세

### 현재 코드

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(exception = Exception400.class)
    public String ex400(Exception400 e) {
        String html = String.format("""
                <script>
                    alert('%s');
                    history.back();
                </script>
                """, e.getMessage());
        return html;
    }

    @ExceptionHandler(exception = Exception401.class)
    public String ex401(Exception401 e) {
        String html = String.format("""
                <script>
                    alert('%s');
                    location.href = '/login-form';
                </script>
                """, e.getMessage());
        return html;
    }

    @ExceptionHandler(exception = Exception403.class)
    public String ex403(Exception403 e) {
        String html = String.format("""
                <script>
                    alert('%s');
                    history.back();
                </script>
                """, e.getMessage());
        return html;
    }

    @ExceptionHandler(exception = Exception404.class)
    public String ex404(Exception404 e) {
        String html = String.format("""
                <script>
                    alert('%s');
                    history.back();
                </script>
```

```
                """, e.getMessage());
        return html;
    }

    @ExceptionHandler(exception = Exception500.class)
    public String ex500(Exception500 e) {
        String html = String.format("""
                <script>
                    alert('%s');
                    history.back();
                </script>
                """, "관리자에게 문의하세요");
        System.out.println("에러 : " + e.getMessage());
        return html;
    }
}
```

### 예외별 응답 동작

| 예외 | alert 메시지 | 이동 | 사용처 |
|---|---|---|---|
| Exception400 | 사용자 입력 메시지 | `history.back()` | 유효성 실패, 중복 |
| Exception401 | 사용자 입력 메시지 | `/login-form` 이동 | 인증 실패 → 로그인 유도 |
| Exception403 | 사용자 입력 메시지 | `history.back()` | 권한 없음 |
| Exception404 | 사용자 입력 메시지 | `history.back()` | 자원 없음 |
| Exception500 | "관리자에게 문의하세요" (고정) | `history.back()` | 서버 에러 (메시지 숨김) |

> Exception500은 보안상 실제 에러 메시지를 사용자에게 노출하지 않고, `System.out.println`으로 서버 로그에만 남긴다.

---

## 4. 계층별 예외 throw 현황

### Controller 계층 - 인증 체크

```
// BoardController, ReplyController에서 반복되는 패턴
User sessionUser = (User) session.getAttribute("sessionUser");
if (sessionUser == null)
    throw new Exception401("인증되지 않았습니다.");
```

**사용 위치:**

| Controller | 메서드 | 예외 |
|---|---|---|
| BoardController | save() | Exception401 |
| BoardController | update() | Exception401 |

| Controller | 메서드 | 예외 |
|---|---|---|
| BoardController | saveForm() | Exception401 |
| BoardController | updateForm() | Exception401 |
| BoardController | delete() | Exception401 |
| ReplyController | delete() | Exception401 |
| ReplyController | save() | Exception401 |

### Service 계층 - 비즈니스 로직

**BoardService:**

| 메서드 | 예외 | 상황 |
|---|---|---|
| 수정폼게시글정보() | Exception404 | 게시글 없음 |
| 수정폼게시글정보() | Exception403 | 수정 권한 없음 |
| 상세보기() | Exception404 | 게시글 없음 |
| 게시글수정() | Exception404 | 게시글 없음 |
| 게시글수정() | Exception403 | 수정 권한 없음 |
| 게시글삭제() | Exception404 | 게시글 없음 |
| 게시글삭제() | Exception403 | 삭제 권한 없음 |

**UserService:**

| 메서드 | 예외 | 상황 |
|---|---|---|
| 회원가입() | Exception400 | 유저네임 중복 |
| 로그인() | Exception401 | 유저 없음 |
| 로그인() | Exception401 | 비밀번호 불일치 |

**ReplyService:**

| 메서드 | 예외 | 상황 |
|---|---|---|
| 댓글삭제() | Exception404 | 댓글 없음 |
| 댓글삭제() | Exception403 | 삭제 권한 없음 |

---

## 5. 예외 처리 흐름 예시

### 예시 1: 존재하지 않는 게시글 수정 시도

```
사용자: POST /boards/999/update
    ↓
BoardController.update()
    ├── 인증 체크 → 통과
    └── boardService.게시글수정(999, ...)
        ↓
BoardService.게시글수정()
    └── boardRepository.findById(999) → Optional.empty()
        └── .orElseThrow(() -> new Exception404("수정할 게시글을 찾을 수 없어요"))
            ↓
GlobalExceptionHandler.ex404()
    └── alert('수정할 게시글을 찾을 수 없어요') + history.back()
```

### 예시 2: 다른 사람의 게시글 삭제 시도

```
사용자 (userId=2): POST /boards/1/delete
    ↓
BoardController.delete()
    ├── 인증 체크 → 통과 (sessionUserId = 2)
    └── boardService.게시글삭제(1, 2)
        ↓
BoardService.게시글삭제()
    ├── findById(1) → board (userId = 1)
    └── 2 != 1 → throw new Exception403("삭제할 권한이 없습니다.")
        ↓
GlobalExceptionHandler.ex403()
    └── alert('삭제할 권한이 없습니다.') + history.back()
```

### 예시 3: 로그인 실패

```
사용자: POST /login (username="ssar", password="wrong")
    ↓
UserController.login()
    └── userService.로그인("ssar", "wrong")
        ↓
UserService.로그인()
    ├── findByUsername("ssar") → User 찾음
    └── "1234".equals("wrong") → false
        └── throw new Exception401("패스워드가 일치하지 않아요")
            ↓
GlobalExceptionHandler.ex401()
    └── alert('패스워드가 일치하지 않아요') + location.href='/login-form'
```

---

## 6. 부족한 부분 분석 및 개선안

### 문제 1: RuntimeException이 GlobalExceptionHandler에 잡히지 않음

현재 게시글쓰기()에서 user를 세팅하지 않아 DB에 null이 들어가면 JPA가 RuntimeException 계열 예외를 던진다. 하지만 GlobalExceptionHandler에는 일반 RuntimeException을 잡는 핸들러가 없다.

**현상:** 스프링 기본 에러 페이지(Whitelabel Error Page)가 노출됨

**개선: RuntimeException 핸들러 추가**

```
@ExceptionHandler(exception = RuntimeException.class)
public String exUnknown(RuntimeException e) {
    String html = String.format("""
            <script>
                alert('%s');
                history.back();
            </script>
            """, "관리자에게 문의하세요");
    System.out.println("미처리 예외 : " + e.getMessage());
    e.printStackTrace();
    return html;
}
```

> 커스텀 예외(Exception400~500)가 먼저 매칭되고, 나머지 모든 RuntimeException이 이 핸들러로 떨어진다.

---

## 문제 2: 댓글등록(ReplyService)에 예외 처리 부재

```
// 현재 코드 - 존재하지 않는 boardId나 userId로 getReference하면
// 실제 INSERT 시점에 FK 위반 에러 발생 → RuntimeException (잡히지 않음)
public void 댓글등록(Integer boardId, String comment, Integer sessionUserId) {
    Board board = em.getReference(Board.class, boardId);   // ← 검증 없음
    User user = em.getReference(User.class, sessionUserId); // ← 검증 없음
    // ...
    replyRepository.save(reply); // ← FK 위반 시 에러
}
```

**개선: getReference 전에 존재 여부 체크하거나, RuntimeException 핸들러로 커버**

문제 1의 RuntimeException 핸들러를 추가하면 자동으로 커버된다.

---

## 문제 3: 게시글쓰기에서 user 세팅 누락

```
// 현재 코드
public void 게시글쓰기(String title, String content) {
    Board board = new Board();
    board.setTitle(title);
    board.setContent(content);
    // ← board.setUser(???) 가 없음!
```

```
    boardRepository.save(board);
}
```

**개선: sessionUserId를 받아서 user를 세팅**

```
@Transactional
public void 게시글쓰기(String title, String content, int sessionUserId) {
    Board board = new Board();
    board.setTitle(title);
    board.setContent(content);
    board.setUser(em.getReference(User.class, sessionUserId));
    boardRepository.save(board);
}
```

BoardController에서 호출하는 부분도 수정:

```
boardService.게시글쓰기(reqDTO.getTitle(), reqDTO.getContent(),
sessionUser.getId());
```

---

## 문제 4: 예외 응답에 HTTP 상태 코드가 없음

현재 모든 예외 응답이 **HTTP 200 OK**로 내려간다. 브라우저에서는 script가 실행되니 문제없지만, API 호출(`/api/boards/{id}`) 시에는 올바른 상태 코드가 필요하다.

**개선: @ResponseStatus 추가**

```
@ExceptionHandler(exception = Exception400.class)
@ResponseStatus(HttpStatus.BAD_REQUEST) // 400
public String ex400(Exception400 e) { ... }

@ExceptionHandler(exception = Exception401.class)
@ResponseStatus(HttpStatus.UNAUTHORIZED) // 401
public String ex401(Exception401 e) { ... }

@ExceptionHandler(exception = Exception403.class)
@ResponseStatus(HttpStatus.FORBIDDEN) // 403
public String ex403(Exception403 e) { ... }

@ExceptionHandler(exception = Exception404.class)
@ResponseStatus(HttpStatus.NOT_FOUND) // 404
public String ex404(Exception404 e) { ... }

@ExceptionHandler(exception = Exception500.class)
@ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR) // 500
public String ex500(Exception500 e) { ... }
```

## 개선 전/후 비교

**개선된 GlobalExceptionHandler 전체 코드:**

```
package com.example.boardv1._core.errors;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestControllerAdvice;

import com.example.boardv1._core.errors.ex.Exception400;
import com.example.boardv1._core.errors.ex.Exception401;
import com.example.boardv1._core.errors.ex.Exception403;
import com.example.boardv1._core.errors.ex.Exception404;
import com.example.boardv1._core.errors.ex.Exception500;

@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(exception = Exception400.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public String ex400(Exception400 e) {
        String html = String.format("""
                <script>
                    alert('%s');
                    history.back();
                </script>
                """, e.getMessage());
        return html;
    }

    @ExceptionHandler(exception = Exception401.class)
    @ResponseStatus(HttpStatus.UNAUTHORIZED)
    public String ex401(Exception401 e) {
        String html = String.format("""
                <script>
                    alert('%s');
                    location.href = '/login-form';
                </script>
                """, e.getMessage());
        return html;
    }

    @ExceptionHandler(exception = Exception403.class)
    @ResponseStatus(HttpStatus.FORBIDDEN)
    public String ex403(Exception403 e) {
        String html = String.format("""
                <script>
                    alert('%s');
                    history.back();
```

```
                </script>
                """, e.getMessage());
        return html;
    }

    @ExceptionHandler(exception = Exception404.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public String ex404(Exception404 e) {
        String html = String.format("""
                <script>
                    alert('%s');
                    history.back();
                </script>
                """, e.getMessage());
        return html;
    }

    @ExceptionHandler(exception = Exception500.class)
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    public String ex500(Exception500 e) {
        String html = String.format("""
                <script>
                    alert('%s');
                    history.back();
                </script>
                """, "관리자에게 문의하세요");
        System.out.println("에러 : " + e.getMessage());
        return html;
    }

    // 미처리 예외 안전망
    @ExceptionHandler(exception = RuntimeException.class)
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    public String exUnknown(RuntimeException e) {
        String html = String.format("""
                <script>
                    alert('%s');
                    history.back();
                </script>
                """, "관리자에게 문의하세요");
        System.out.println("미처리 예외 : " + e.getMessage());
        e.printStackTrace();
        return html;
    }
}
```

---

## 7. 개선 요약

| # | 문제 | 현상 | 개선 |
|---|---|---|---|

| # | 문제 | 현상 | 개선 |
|---|---|---|---|
| 1 | RuntimeException 핸들러 없음 | Whitelabel 에러 페이지 노출 | `RuntimeException` 핸들러 추가 (안전망) |
| 2 | 댓글등록 시 검증 부재 | FK 위반 에러 → 미처리 | 문제 1의 안전망으로 커버 |
| 3 | 게시글쓰기에 user 누락 | user_id가 null로 저장 | sessionUserId 파라미터 추가 |
| 4 | HTTP 상태 코드 없음 | 모든 에러가 200 OK 응답 | `@ResponseStatus` 추가 |

## 8. 예외 처리 설계 원칙 정리

```
1. Controller → 인증 체크 (Exception401)
2. Service    → 자원 존재 여부 (Exception404)
                → 권한 체크 (Exception403)
                → 비즈니스 규칙 위반 (Exception400)
3. GlobalExceptionHandler → 한 곳에서 모든 예외 처리
4. RuntimeException 핸들러 → 예상치 못한 에러의 안전망
```

| 계층 | 책임 | 사용 예외 |
|---|---|---|
| Controller | 인증 체크 | Exception401 |
| Service | 자원 조회 실패 | Exception404 |
| Service | 권한 검증 실패 | Exception403 |
| Service | 비즈니스 규칙 위반 (중복 등) | Exception400 |
| Service | 예상치 못한 서버 에러 | Exception500 |
| GlobalExceptionHandler | 모든 예외를 사용자 친화적 응답으로 변환 | 전부 |