Spring Boot Application

스프링 부트 응용

사용자 인증과 예외 처리

한국기술교육대학교
온라인평생교육원

학습내용

- 사용자 인증 처리
- 예외 처리

학습목표

- 웹 애플리케이션을 사용하는 사용자를 인증하는 로그인 기능을 구현할 수 있다.
- 시스템에서 발생하는 예외를 적절히 처리할 수 있다.

사용자 인증 처리

1 로그인 화면 개발

〈1〉 인덱스 페이지 추가

인덱스 페이지

- 웹 애플리케이션에 접속한 사용자에게 처음으로 제공하는 페이지

1 로그인 화면 개발

〈1〉 인덱스 페이지 추가

- 스프링 부트는 src/main/resources 소스 폴더에 있는 static 폴더에 작성된 index.html 파일을 인덱스 페이지로 인지함

사용자 인증 처리

1 로그인 화면 개발

① 인덱스 페이지 추가

```
<!DOCTYPE html>
<html>
<head>
<title>메인 페이지</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<br>
<h3 align="center">게시판 프로그램입니다.</h3>
<hr>
<p align="center"><a href="board/getBoardList">글목록 바로가기</a></p>
<p align="center"><a href="board/login">로그인</a></p>
<br>
<hr>
</body>
</html>
```

1 로그인 화면 개발

② 컨트롤러 작성

- 사용자가 인덱스 페이지에서 [로그인] 링크를 클릭했을 때 로그인 화면으로 이동할 수 있도록 LoginController를 작성함

```
package com.mycompany.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.SessionAttributes;

@SessionAttributes("member")
@Controller
public class LoginController {

    @GetMapping("/board/login")
    public void loginView() {
    }

}
```

사용자 인증 처리

1 로그인 화면 개발

③ HTML 화면 작성

- templates/board 폴더에 login.html 파일을 작성함

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>회원 로그인</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body th:align="center">
<h1>로그인</h1>
<form th:action="login" method="post">
<table th:align="center" border="1" th:cellpadding="0" th:cellspacing="0">
<tr>
<td bgcolor="orange" th:text="아이디"></td>
<td><input name="id" type="text" size="10"></td>
</tr>
<tr>
<td bgcolor="orange" th:text="비밀번호"></td>
<td><input name="password" type="password" size="10"></td>
</tr>
<tr><td colspan="2" align="center"><input type="submit" value="로그인"></td></tr>
</table>
</form>
</body>
</html>
```

2 로그인 컴포넌트 개발

① 엔티티 클래스

- 사용자의 요청을 처리하기 위해서 가장 먼저 작성할 클래스가 회원 엔티티 클래스임

```
@Data
@Entity
public class Member {
    @Id
    private String id;
    private String password;
    private String name;
    private String role;
}
```

사용자 인증 처리

2 로그인 컴포넌트 개발

② 리포지터리 작성

- Member 엔티티를 이용하여 MEMBER 테이블에 대한 CRUD를 제공할 MemberRepository 인터페이스를 작성함

```
@Repository
public interface MemberRepository extends CrudRepository<Member, String> {
}
```

2 로그인 컴포넌트 개발

③ 비즈니스 클래스 작성

- MemberRepository를 이용하여 회원 정보를 조회하는 MemberService 클래스를 작성함

```
@Service("memberService")
public class MemberService {
    @Autowired
    private MemberRepository memberRepo;

    public Member getMember(Member member) {
        Optional<Member> findMember = memberRepo.findById(member.getId());
        if (findMember.isPresent())
            return findMember.get();
        else
            return null;
    }
}
```

사용자 인증 처리

2 로그인 컴포넌트 개발

⟨4⟩ 비즈니스 클래스 작성

- 실질적인 사용자 인증을 처리할 LoginController.login 메소드를 작성함

2 로그인 컴포넌트 개발

⟨4⟩ 비즈니스 클래스 작성

```
@Controller
public class LoginController {
    @Autowired
    private MemberService memberService;

    @GetMapping("/board/login")
    public void loginView() {
    }

    @PostMapping("/board/login")
    public String login(Member member, Model model) {
        System.out.println("인증 처리");
        Member findMember = memberService.getMember(member);
        if (findMember != null && findMember.getPassword().equals(member.getPassword())) {
            model.addAttribute("member", findMember);
            return "forward:getBoardList";
        } else {
            return "redirect:login";
        }
    }
}
```

사용자 인증 처리

3 로그인 상태 유지

1 인증 상태 유지

- 현재의 게시판은 로그인을 인증하지 않은 사용자도 게시판 기능을 사용할 수 있음
- 로그인 성공한 사용자만 게시판 기능을 사용할 수 있어야 함

3 로그인 상태 유지

1 인증 상태 유지

- 스프링에서 제공하는 Model에 데이터를 저장하면 자동으로 request 내장 객체에도 등록됨
- @SessionAttributes를 이용하면 Model에 등록한 데이터를 세션(HttpSession)에도 등록할 수 있음

사용자 인증 처리

3 로그인 상태 유지

〈2〉 @SessionAttributes 적용

- @SessionAttributes에 "member"를 지정하면 "member"라는 이름으로 Model에 등록된 데이터가 세션에도 등록됨

3 로그인 상태 유지

〈2〉 @SessionAttributes 적용

```
@SessionAttributes("member")
@Controller
public class LoginController {
    @Autowired
    private MemberService memberService;

    ~ 생략 ~

    @PostMapping("/board/login")
    public String login(Member member, Model model) {
        System.out.println("인증 처리");
        Member findMember = memberService.getMember(member);
        if (findMember != null && findMember.getPassword().equals(member.getPassword())) {
            model.addAttribute("member", findMember);
            return "forward:getBoardList";
        } else {
            return "redirect:login";
        }
    }
}
```

사용자 인증 처리

3 로그인 상태 유지

③ 세션 정보 이용

- 인증에 성공한 회원만 글목록을 볼 수 있도록 하려면 BoardController.getBoardList 메소드 세션을 기반으로 수정해야 함

```
@RequestMapping("/board/getBoardList")
public String getBoardList(Model model, HttpSession session) {
    if(session.getAttribute("member") == null) return "redirect:/";
    model.addAttribute("boardList", boardService.getBoardList());
    return "board/getBoardList";
}
```

3 로그인 상태 유지

④ 사용자 이름 출력

- 세션에 등록된 사용자의 정보는 다양하게 활용할 수 있음

사용자 인증 처리

3 로그인 상태 유지

④ 사용자 이름 출력

1 사용자 이름 출력

- 글목록 화면(getBoadList.html)에 사용자 이름을 출력하는 코드를 추가함

```
<h1>게시글 목록(Thymeleaf)</h1>
<h3><font color="red" th:text="${session['member'].name}"></font>
님 게시판 입장을 환영합니다.</h3>
```

3 로그인 상태 유지

④ 사용자 이름 출력

2 권한 제어

- 관리자(Admin)만 글삭제 기능을 사용할 수 있도록 글상세 화면(getBoard.html)을 수정함

```
<p th:align="center">
<a th:href="@{insertBoard}">글등록</a>   
<a th:if="${session['member'].role == 'ADMIN'}"
   th:href="@{deleteBoard(seq=${board.seq})}">글삭제</a>   
<a th:href="@{getBoardList}">글목록</a>
</p>
```

사용자 인증 처리

3 로그인 상태 유지

⑤ 로그아웃

- 사용자가 로그아웃을 요청하면 현재 브라우저와 매핑된 세션을 강제로 종료하고 인덱스 페이지로 이동하면 됨

3 로그인 상태 유지

⑤ 로그아웃

1 로그아웃 링크 추가

- 글목록 화면(getBoardList.html)에 로그아웃 링크를 추가함

```
<h1>게시글 목록(Thymeleaf)</h1>
<h3><font color="red" th:text="${session['member'].name}"></font>님 게시판
<a th:href="@{/board/logout}">LOG_OUT</a>
```

사용자 인증 처리

3 로그인 상태 유지

⑤ 로그아웃

2 로그아웃 처리

- LoginController 클래스에 logout 메소드를 추가하고 SessionStatus 객체를 이용하여 세션을 강제 종료함

```
@GetMapping("/board/logout")
public String logout(SessionStatus status) {
    status.setComplete();
    return "redirect:/";
}
```

예외 처리

1 예외(Exception)와 예외 처리

〈1〉 예외 개념

- JAVA는 시스템에서 발생되는 문제를 시스템 에러(Error)와 예외(Exception)로 구분함

1 예외(Exception)와 예외 처리

〈1〉 예외 개념

- 에러는 전원 차단 같은 물리적인 문제이기 때문에 개발자가 제어할 수 없음
- 예외는 사용자의 부주의나 시스템의 결함으로 인해 발생되는 문제이므로 얼마든지 제어할 수 있음

예외 처리

1 예외(Exception)와 예외 처리

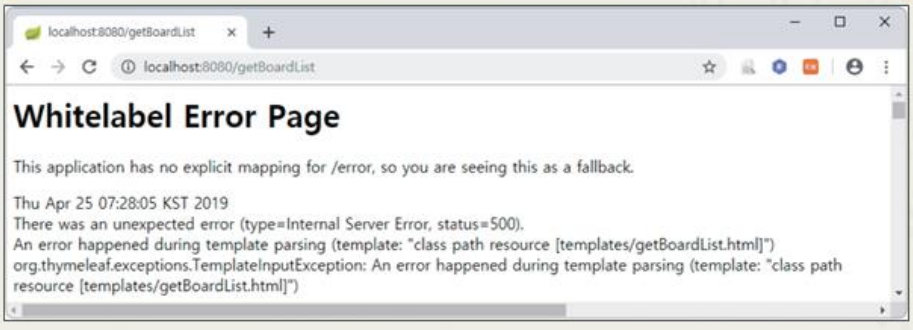
〈1〉 예외 개념

- 시스템에서 발생한 예외를 적절하게 처리하지 않으면 사용자는 시스템을 신뢰할 수 없음

1 예외(Exception)와 예외 처리

〈1〉 예외 개념

- 스프링 부트는 예외를 처리하지 않았을 때, 사용자에게 다음과 같은 에러 화면을 전송함



예외 처리

1 예외(Exception)와 예외 처리

〈2〉 사용자 정의 예외

- NullPointerException 같이 미리 정의된 예외도 있지만 사용자가 직접 예외를 정의할 수도 있음

1 예외(Exception)와 예외 처리

〈2〉 사용자 정의 예외

- 예외

Checked 예외	• 컴파일 과정에서 발생
Unchecked 예외	• 실행 시점에 발생

예외 처리

1 예외(Exception)와 예외 처리

〈2〉 사용자 정의 예외

- BoardException 클래스

- 게시판 애플리케이션에서 발생할 수 있는 모든 예외의 최상위 부모로 사용
- Unchecked 예외의 최상위 부모인 RuntimeException을 상속하여 구현하면 됨

1 예외(Exception)와 예외 처리

〈2〉 사용자 정의 예외

- BoardException 클래스

```
package com.mycompany.exception;

public class BoardException extends RuntimeException {
    private static final long serialVersionUID = 1L;

    public BoardException(String message) {
        super(message);
    }
}
```

예외 처리

1 예외(Exception)와 예외 처리

〈2〉 예외 처리

스프링에서 예외를 처리하는 방법	
@ControllerAdvice	모든 컨트롤러에서 발생하는 예외를 **일괄적**으로 처리
@ExceptionHandler	각 컨트롤러마다 발생하는 예외를 **개별적**으로 처리

1 예외(Exception)와 예외 처리

〈2〉 예외 처리

전역 예외 처리	• @ControllerAdvice를 사용하는 것
로컬 예외 처리	• @ExceptionHandler를 사용하는 것

예외 처리

2 예외 처리

① 사용자 정의 예외 작성

- 검색된 Board 엔티티가 없는 상황을 예외로 처리하기 위해 BoardNotFoundException 클래스를 작성함

```
package com.mycompany.exception;

public class BoardNotFoundException extends BoardException {
    private static final long serialVersionUID = 1L;

    public BoardNotFoundException(String message) {
        super(message);
    }
}
```

2 예외 처리

② 예외 처리기 작성

- @ControllerAdvice를 이용하여 발생된 예외에 대해서 적절한 예외 화면을 연결해주는 전역 예외 처리기를 등록함

예외 처리

2 예외 처리

〈2〉 예외 처리기 작성

```
package com.mycompany.exception;

import org.springframework.ui.Model;

@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(BoardException.class)
    public String handleCustomException(BoardException exception, Model model) {
        model.addAttribute("exception", exception);
        return "/errors/boardError";
    }

    public String handleException(Exception exception, Model model) {
        model.addAttribute("exception", exception);
        return "/errors/globalError";
    }
}
```

2 예외 처리

〈2〉 예외 처리기 작성

- @ExceptionHandler를 사용하면 특정 예외 타입에 대해 실행될 메소드를 적절하게 매핑할 수 있음

예외 처리

2 예외 처리

〈3〉 예외 페이지

- BoardNotFoundException이 발생했을 때 사용자에게 제공할 예외 화면을 작성해야 함

2 예외 처리

〈3〉 예외 페이지

1. src/main/resources 소스 폴더 templates 폴더에 새로운 errors 폴더를 생성함
2. boardError.html 파일을 작성함

예외 처리

2 예외 처리

〈3〉 예외 페이지

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="utf-8" />
</head>
<body>
<h1><font color="orange">BoardException 발생!</font></h1>
<a th:href="@{/}"> 메인 화면으로 </a><hr>
<table>
<tr>
<th bgcolor="orange" align="left">예외 메시지 : [[${exception.message}]]</th>
</tr>
<tr th:each="trace : ${exception.stackTrace}">
    <td th:text="${trace}">
</tr>
</table>
</body>
</html>
```

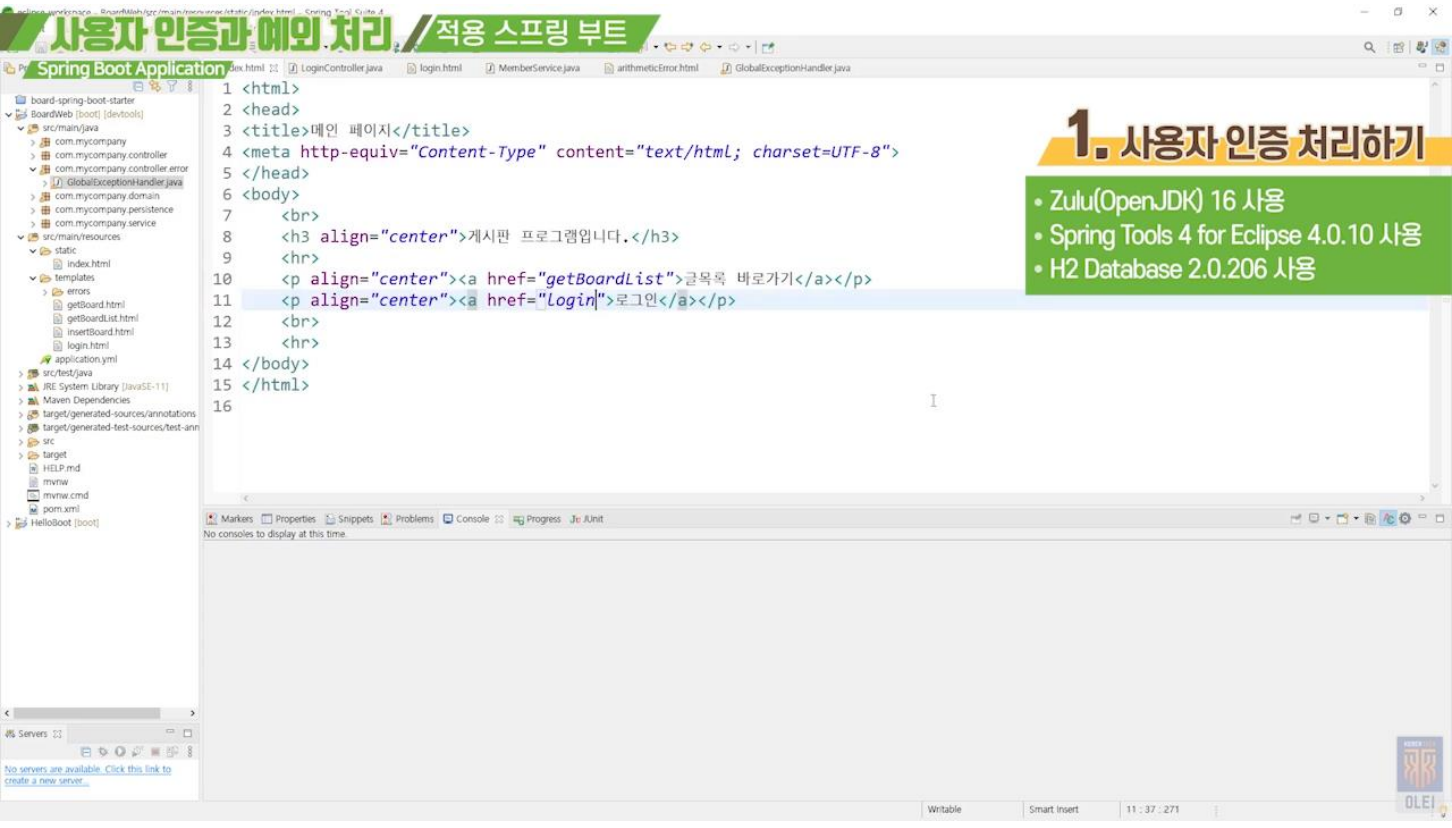
2 예외 처리

〈4〉 예외 재정의

- 컨트롤러마다 예외 처리를 로직을 구현하고 싶으면 해당 컨트롤러에 @ExceptionHandler가 설정된 메소드를 작성하면 됨

```
@ExceptionHandler(SQLException.class)
public String sqlError(SQLException exception, Model model) {
    model.addAttribute("exception", exception);
    return "/errors/sqlError";
}
```

적용 스프링 부트

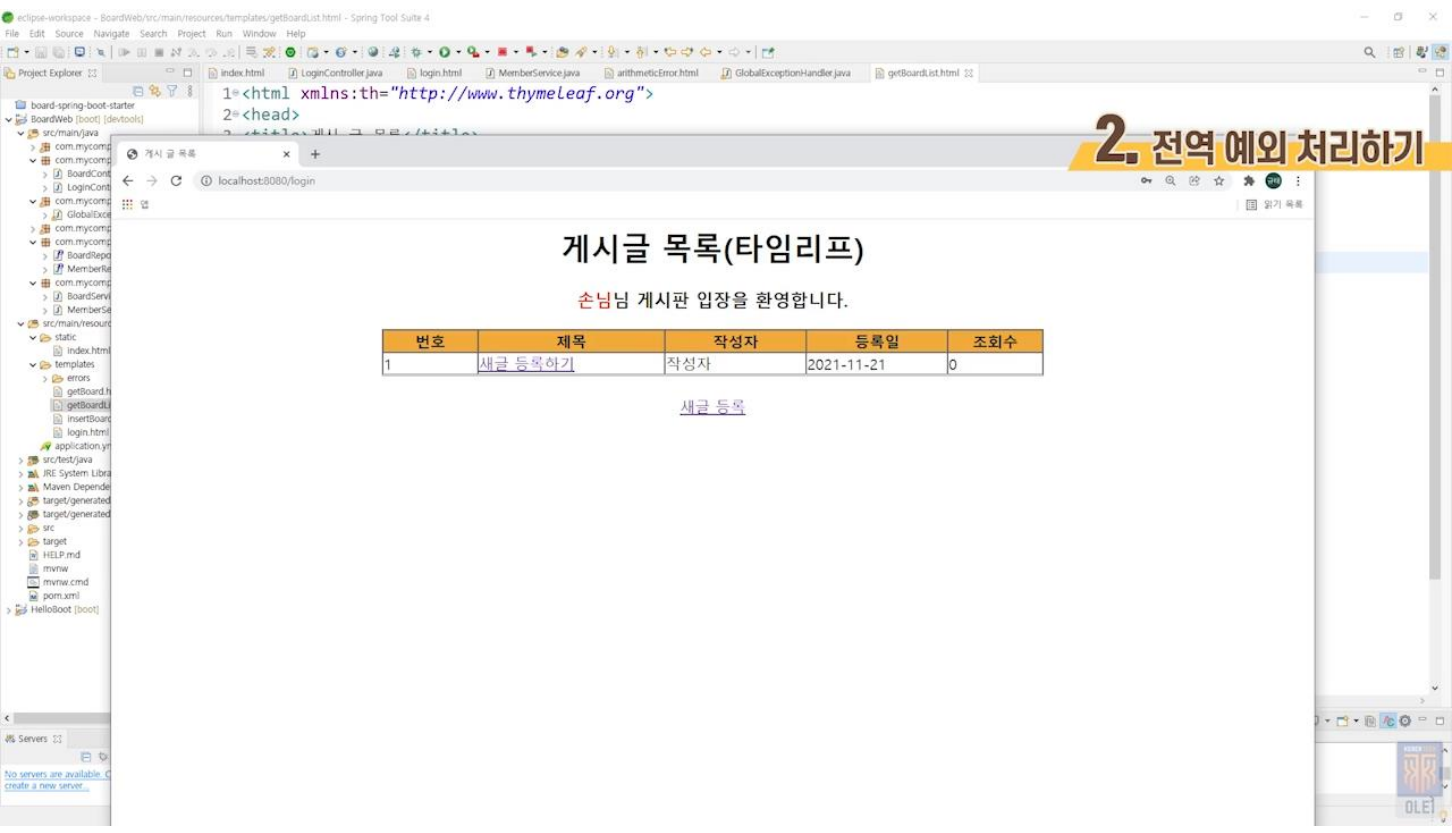


실습단계
로그인 인증, 예외 화면 처리
index 페이지 작성
static 폴더에 HTML 태그만을 이용하는 정적인 문서로 작성
기본 welcome 파일로 등록됨
브라우저에서 실행 확인
Tip! 화면을 잠깐 멈추고 소스를 타이핑하며 진행할 것
글목록 바로가기, 로그인 링크 제공
로그인 화면을 띄우기 위해 BoardController.java와 같은 위치에 LoginController.java 작성
insertBoard.html 파일 참조
Service class 작성
MemberRepository는 기존에 작업이 되어 있어서 MemberService 클래스만 작성
LoginController.java 수정
의존성 주입

적용 스프링 부트

실습단계
로그인 요청이 post 방식으로 들어올 때 실행될 메소드 등록
MemberService를 이용해서 User 객체를 SELECT
결과가 존재하고, 비밀번호가 일치하면 로그인 성공 인지
Controller 위에 @SessionAttributes 어노테이션 추가
글목록, 상세화면 등 다양한 화면에서 세션에 등록된 회원정보 이용 가능
MEMBER 테이블에 데이터 insert
로그인 인증 테스트
등록한 아이디와 비밀번호 입력 테스트
세션에 저장된 사용자 이름 출력
getBoardList.htm을 열어서 내용 추가

적용 스프링 부트



실습단계
예외를 발생시키기 위해 LoginController.java를 열어서 loginView( ) 메소드 수정
정수를 0으로 나누는 행위는 ArithmeticException을 발생시킴
예외 발생 시 일반 사용자는 화면을 이해하지 못하며, 개발자는 시스템의 신뢰도가 떨어지므로 예외처리 해야 함
예외 화면 작성
templates → errors → arithmeticErrors.html
예외가 발생한 곳 확인
GlobalExceptionHandler.java 클래스 생성
com.mycompany.controller.error 패키지 내에 생성
중요! GlobalExceptionHandler 위에 반드시 @ControllerAdvice 어노테이션 설정
어떤 타입의 예외가 발생 시 무슨 메소드가 실행될 지 매핑
나머지 예외 발생 시 handleException( ) 메소드가 동작해 기본 에러 페이지가 실행될 수 있도록 함
arithmeticError.html를 복사하여 globalError.html로 만드는 것이 좋음
Reloading