

(디지털 컨버전스) 스마트 콘텐츠와 웹 융합 응용 SW개발자 양성과정

훈련기간 : 2021.05.07 ~ 2021.12.08

@Component를 통해 @Repository,@Service,@controller를 스프링 빈에 등록해서 @Autowired로 연결한다고 보면된다.

@Autowired

```
private BoardService service;
```

스프링이 자동으로 객체를 찾을 수 있게 서포트함

thymeleaf에서는 아래와 같이 특정한 객체를 입력으로 받으면 HTML에서 th:object와 같은 키워드를 통해 정보를 획득할 수 있다.

MVC(Model View Controller) Pattern

Model: 다루는 데이터

View: 눈에 보이는 화면

Controller: URL 제어

Client가 요청하면 컨트롤러 진입 -> 컨트롤러 요청에 대한 작업 수행 -> 뷰쪽으로 데이터 전달 -> 사용자가 웹 사이트 접속

-> Controller는 사용자가 요청한 웹페이지를 서비스하기위해 모델을 호출한다. -> 모델은 데이터소스를 제어한 후 결과를 리턴한다.

-> 컨트롤러는 모델이 리턴한 결과를 view에 반영한다.

Spring MVC애플리케이션에서

@Controller 클래스 데이터로 모델 맵을 준비하고 렌더링 할 뷰를 선택하는 역할을 한다.

이 모델 맵은 뷰 기술의 완전한 추상화를 허용하며, Thymeleaf의 경우 Thymeleaf 컨텍스트 객체로 변환되어 템플릿에서 실행되는 표현식에 정의된 모든 변수를 사용할 수 있다.

Spring MVC는 뷰 모델 속성을 실행하는 동안 액세스 할 수 있는 데이터 조각을 호출 합니다. Thymeleaf 언어에서 동등한 용어는 컨텍스트 변수 이다.

Spring MVC의 뷰에 모델 속성을 추가하는 방법은 여러 가지가 있습니다. 다음은 몇 가지 일반적인 경우이다.

model.addAttribute

modelAndView 모델 속성이 포함 된 반환:

다음으로 주석이 달린 메소드를 통해 공통 속성을 노출한다. @ModelAttribute.

thymeleaf란?

웹서비스를 만들 때에는 서버의 데이터와 정적자원(html,css,image)을 조합해야한다.

보통 자바에서 웹 개발시 (Java Server Page)를 이용하여 진행한다.

JSP를 사용하면 스크립트릿과 html이 혼재된 상태가 되고 html태그의 반복적인 사용으로 인해 수정하기 어려운 상황이 된다.

이러한 상태를 해결할 수있는 것이 바로 템플릿 엔진이다.

템플릿 엔진이란 동적 콘텐츠를 생성하는 방법으로 html(Markup)과 데이터를 결합한 결과물을 만들어 주는 소프트웨어(또는 컴포넌트) 이다.

자바 템플릿 엔진 종류: Groovy, Mustache, Thymleaf, Freemaker, Jsp

타임리프는 이 템플릿 엔진 중하나이다. 스프링부트에있는 JSP가 아닌 타임리프를 사용할 것을 권장한다.

+

스크립트릿 (순수 자바코드 기술) 사용<%이안에 자바코드 들어감%>

SPA(Single Page Application)

최초 한번 전체 페이지를 다 불러오고 응답 데이터만 페이지 특정부분 렌더링

SSR(Server Side Rendering)

전통적인 웹 애플리케이션 방식. 요청시마다 서버에서 처리 후 새로고침으로 페이지에 대한 응답.(과거)

```
@Autowired
```

```
private BoardService service;
```

```
@GetMapping("/register")
```

```
public String getRegister (Board board, Model model) {
```

```
    log.info("getRegister()");
```

```
    return "/board/fourth/register";
```

```
}
```

thymeleaf에서는 아래와 같이 특정한 객체를 입력으로 받으면
HTML에서 th:object와 같은 키워드를 통해 정보를 획득할 수 있다.
즉 board 객체의 정보를 획득할 수 있음을 의미한다.

```
@PostMapping("/register")
```

```
public String postRegister (Board board, Model model) throws Exception {
```

```
    log.info("postRegister()");
```

```
    log.info("Board: " + board);
```

```
    service.register(board);
```

```
    model.addAttribute( attributeName: "msg",   attributeValue: "등록이 완료되었습니다!");
```

```
    return "/board/fourth/success";
```

```
}
```

model 즉 데이터
msg 라는 속성값에 "등록이 완료되었습니다!"을 맵핑함
key: msg, value: "등록이 완료되었습니다!" 라고 생각하면 편함(map같은)

데이터 이동부분

1.

```
@Autowired
private BoardService service;
```

```
@GetMapping("/register")
```

```
public String getRegister (Board board, Model model) {
    log.info("getRegister()");

    return "/board/fourth/register";
}
```

빈 board 객체 전달

2

register.html

```
<table>
<tr>
    <td>제목</td>
    <!-- th:field의 경우엔 Board 객체에 있는 title과 직접 맵핑시킴 -->
    <td><input type="text" name="title" th:field="*{title}"></td>
</tr>
<tr>
    <td>작성자</td>
    <td><input type="text" name="writer" th:field="*{writer}"></td>
</tr>
<tr>
    <td>본문</td>
    <!-- textarea는 글자를 여러개 입력할 수 있는 글 입력창이다 -->
    <td><textarea cols="50" rows="20" name="content" th:field="*{content}"></textarea></td>
</tr>
</table>
```

3

```
@PostMapping("/register")
```

```
public String postRegister (Board board, Model model) throws Exception {
    log.info("postRegister()");
    log.info("Board: " + board);

    service.register(board);

    model.addAttribute( attributeName: "msg",    attributeValue: "등록이 완료되었습니다!");

    return "/board/fourth/success";
}
```

```
<form id="board" th:action="@{register}" th:object="${board}" method="post">
```

board 객체 작성후 전달

4

register.html

1. 빈 board 객체를 받는다.
2. 클라이언트 입력을 하고 제출한다.
3. post맵핑으로 전달 받은 객체를 log.info(board)로 보여주고 service를 통해 테이블에 최종 저장한다.
4. 클라이언트는 성공페이지와 등록 완료 메시지를 받는다.
(model.addAttribute로 model객체에 메소드를 이용해서 화면전송)

HTML script

document는 현재 웹 상에 떠 있는 페이지 자체를 의미한다고 보면 됨
form 태그에 id="board"에 해당하는 객체를 얻어옴
id="btnRegister"를 클릭했을때 URL register로 보낸다.
PostMapping
self.location = "lists"; list URL 이동

```
<script>
$(document).ready(function() {
    var formObj = $("#board")
    $("#btnRegister").on("click", function() {
        formObj.attr("action", "register");
        formObj.attr("method", "post");
        formObj.submit();
    });
    $("#btnList").on("click", function() {
        self.location = "lists";
    });
});
</script>
```

데이터 이동부분

Board 객체 구조

```
@Getter
@Setter
ToString
public class Board {
    private int boardNo;
    private String title;
    private String content;
    private String writer;
    private Date regDate;
}
```

서비스 인터페이스

두개 추상메서드

1. 등록
2. 리스트용

```
package com.example.demo.service;

import com.example.demo.entity.Board;

import java.util.List;

public interface BoardService {

    public void register(Board board) throws Exception;

    public List<Board> list() throws Exception;
}
```

서비스 구현

등록 메서드 : post로 전달받은 객체 저장
리스트 메서드 : 저장소 리스트 반환

```
package com.example.demo.service;

import com.example.demo.entity.Board;
import com.example.demo.repository.BoardRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

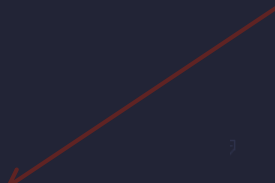
// Service는 여기서 register가 여러 방식으로 동작할 수 있음을 명시한다.
// 또한 Controller의 Autowired에 자동으로 연결되도록 서포트한다.
@Service
public class BoardServiceImpl implements BoardService {

    @Autowired
    private BoardRepository repository;

    @Override
    public void register(Board board) throws Exception {
        repository.create(board);
    }

    @Override
    public List<Board> list() throws Exception {
        return repository.list();
    }
}
```

저장소와 연결



저장소

```
@Repository
public class BoardRepository {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public void create(Board board) throws Exception {
        String query = "insert into board (title, content, writer) values (?, ?, ?)";

        jdbcTemplate.update(query, board.getTitle(), board.getContent(), board.getWriter());
    }
}
```

jdbcTemplate.update : 전달받은 값 테이블에 저장
(JDBC에 접근 방법)

```
public List<Board> list() throws Exception {
    // RowMapper를 통해 얻은 행을 하나씩 List에 집어넣으니
    // results엔 DB에서 얻어온 행 정보들이 들어있다.
    List<Board> results = jdbcTemplate.query(
        sql: "select board_no, title, content, writer, reg_date from board " +
            "where board_no > 0 order by board_no desc",
        // Row: 행
        // 여러개의 Column(열)들이 행 1개에 포함되어 있음
        // 여러 열들을 얻어서 행으로 맵핑하는 작업을 수행함
        new RowMapper<Board>() {
            @SneakyThrows
            @Override
            public Board mapRow(ResultSet rs, int rowNum) throws SQLException {
                Board board = new Board();

                // rs.getInt()는 DB에 있는 정수형 정보를 얻어옴
                board.setBoardNo(rs.getInt( columnLabel: "board_no"));
                // rs.getString()은 DB에 있는 문자열 정보를 얻어옴
                board.setTitle(rs.getString( columnLabel: "title"));
                board.setContent(rs.getString( columnLabel: "content"));
                board.setWriter(rs.getString( columnLabel: "writer"));
                // rs.getDate()는 DB에 있는 날짜 정보를 얻어옴
                // board.setRegDate(rs.getDate("reg_date"));
                SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss");
                board.setRegDate(sdf.parse( source: rs.getDate( columnLabel: "reg_date") + " " + rs.getTime( columnLabel: "reg_date")));
                return board;
            }
        }
    );
    return results;
}
```

수동으로 list를 불러오는 방법

where board_no > 0 order by board_no desc 내림차순으로 불러오기
나중에 jpa사용시 생략할 수 있음