

2장. 타임리프(Thymeleaf)



데이터 처리 및 화면 구현



Spring Boot

Thymeleaf 사용

● 타임리프(Thymeleaf)란?

- 서버사이드 Java 템플릿 엔진
- HTML 파일에 Java 코드를 간단히 연결해서 동적으로 화면을 구성할 수 있음
- JSP보다 장점이 많음
- HTML 그대로 브라우저에서 열어도 깨지지 않음 → 프론트엔드와 협업 용이
- 표현식 문법 단순 (th:text, th:each 등)
- Spring MVC와 자연스럽게 통합 (Model 객체 데이터 바인딩)
- 템플릿 엔진은 타임리프(Thymeleaf)를 비롯하여 Freemaker, Mustache, Groovy Templates이 있다.



Thymeleaf 사용

● 타임리프(www.thymeleaf.org) > Docs

Tutorials

These are both learning and reference materials for Thymeleaf.

Using Thymeleaf

All the Thymeleaf basics: from an introduction to Thymeleaf and must read for new and veteran Thymeleaf users.

- **Thymeleaf 3.1:** [Read online](#), [PDF](#), [EPUB](#), [MOBI](#)
Japanese translation: [Read online](#), [PDF](#), [EPUB](#), [MOBI](#)

3.1 A multi-language welcome

Our first task will be to create a home page for our grocery site.

The first version of this page will be extremely simple: just a title and a welcome message. The `src/main/resources/templates/home.html` file:

```
<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

  <head>
    <title>Good Thymes Virtual Grocery</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link rel="stylesheet" type="text/css" media="all"
          href="../../css/gtv.css" th:href="@{/css/gtv.css}" />
  </head>

  <body>

    <p th:text="#{home.welcome}">Welcome to our grocery store!</p>

  </body>

</html>
```



Thymeleaf 사용

- 타임리프 문법

- th:text="\${변수}" → 텍스트 출력
- th:each="item : \${list}" → 반복문
- th:if, th:unless → 조건문
- th:href="@{/경로}" → 링크

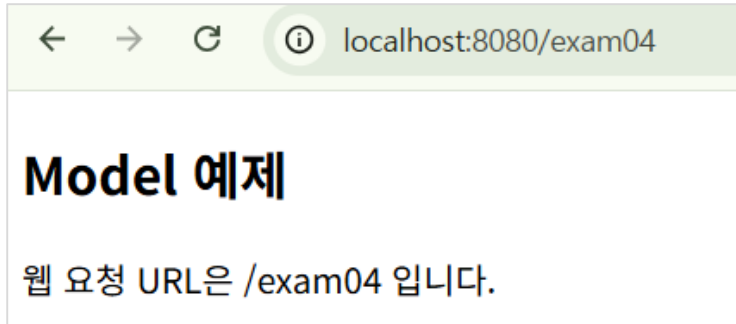
- 사용: html 문서에 태그로 명시해야 함

```
<html xmlns:th="http://www.thymeleaf.org">
```



요청 처리 메서드와 모델 유형

- Model 인터페이스로 메서드 작성



```
<body>
  <!-- <h2 th:text="${data1}"></h2> -->
  <h2>[[${data1}]]</h2>

  <p th:text="${data2}"></p>
</body>
```



요청 처리 메서드와 모델 유형

- Model 인터페이스로 메서드 작성

```
@Controller
public class Exam04Controller {

    @GetMapping("/exam04")
    public String requestMethod(Model model) {
        model.addAttribute("data1", "Model 예제");
        model.addAttribute("data2", "웹 요청 URL은 /exam04 입니다.");
        return "pages/view04"; //view04.html
    }
}
```



메인 페이지 만들기 – thymeleaf 사용

- templates > home.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>home</title>
<link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<body>
    <div id="content">
        <h2>Home</h2>

        <!-- 데이터(모델) 받기 -->
        <h3 th:text="${message}"></h3>

        <hr>
        <p>
            <a th:href="@{/time}">시간 페이지로 이동</a>
        </p>

    </div>
</body>
</html>
```



메인 페이지 만들기 – thymeleaf 사용

- templates > pages > time.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>시간 이란..</title>
<link rel="stylesheet" th:href="@{/css/style.css}">
<script th:src="@{/js/time.js}"></script>
</head>
<body>
    <div id="content">
        <h2>시간이란..</h2>
        <h3>내일 죽을 것처럼 오늘을 살고<br>
            영원히 살 것처럼 내일을 꿈꾸어라
        </h3>
        
        <p id="display">
    </div>
</body>
</html>
```



요청 파라미터(parameter)

- 요청 파라미터

요청 파라미터(request parameter)는 일반적인 웹 서버 애플리케이션에서 GET 방식의 쿼리문을 "이름=값(name=value)" 형태의 데이터로 전송한다.

예) 웹 요청 URL이 <http://.../search?q=전기차> 파라미터 이름은 q, 값은 전기차

- @RequestParam으로 요청 파라미터 처리

기본값은 웹 요청 URL로 전송되는 요청 파라미터 이름과 똑같이 설정한다.

이 요청 파라미터의 값은 매개변수의 타입에 따라 적절하게 변환된다.



요청 파라미터(parameter)

- @RequestParam으로 요청 파라미터 처리

웹 요청 URL이 /ex01?id=space&pw=12345



```
<body>  
    <h2 th:text="${data1}"></h2>  
    <p th:utext="${data2}"></p>  
</body>
```

pages/viewPaeg01.html



요청 파라미터(parameter)

- @RequestParam으로 요청 파라미터 처리

```
@Controller
public class Ex01Controller {

    @GetMapping("/ex01")
    public String method1(@RequestParam("id") String userId,
        @RequestParam("pw") String userPw,
        Model model) {
        model.addAttribute("data1", "@RequestParam 예제");
        model.addAttribute("data2", "id:" + userId + "<br>pw:" + userPw);
        return "pages/viewPage01";
    }
}
```



요청 파라미터(parameter)

- 경로 변수와 @PathVariable

경로 변수는 웹 요청 URL에 포함된 파라미터 값을 전달받을 때 사용하는 변수이다. 매핑 경로를 설정하는 @RequestMapping에 중괄호({ })를 사용해 요청 조건 값을 전달받는다.

예) URL이 <http://localhost:8080/boards/1> 이라면 경로 변수는 boards이고 값은 1이다.

- @PathVariable로 경로 변수 처리

```
@GetMapping("/{경로_변수}")  
    public String method1(@PathVariable("경로_변수") 매개변수) {  
        ...  
    }
```



요청 파라미터(parameter)

- 경로 변수와 @PathVariable



```
@Controller
public class Ex02Controller {

    @GetMapping("/boards/{id}")
    public String method1(@PathVariable("id") Long id,
        Model model) {
        model.addAttribute("data1", "@PathVariable 예제");
        model.addAttribute("data2", "글번호: " + id);
        return "pages/viewPage02";
    }
}
```



요청 파라미터(parameter)

- **@ModelAttribute로 요청 파라미터 처리**

@ModelAttribute는 form의 name 속성과 동일한 이름의 필드에 자동 매핑합니다.

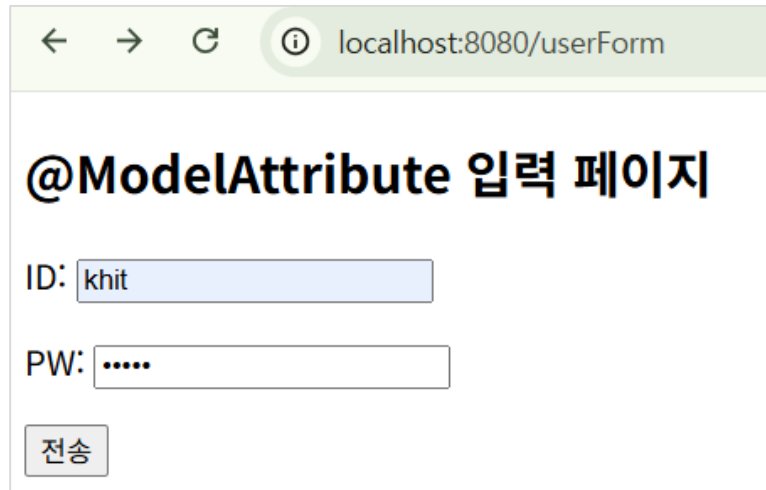
@ModelAttribute User user

폼에서 전송된 name="id", name="pw" 값을 User 객체의 setId(), setPw()로 자동 바인딩



요청 파라미터(parameter)

- @ModelAttribute로 요청 파라미터 처리



← → ↻ ⓘ localhost:8080/userForm

@ModelAttribute 입력 페이지

ID:

PW:



@ModelAttribute 예제 - 결과 페이지

id: khit
pw: 12345



요청 파라미터(parameter)

- @ModelAttribute로 요청 파라미터 처리

```
import lombok.Data;

@Data
public class User {
    private String id;
    private String pw;
}
```



요청 파라미터(parameter)

- @ModelAttribute로 요청 파라미터 처리

```
@Controller
public class Ex03Controller {

    // 입력 폼 페이지
    @GetMapping("/userForm")
    public String formPage() {
        return "pages/userForm";
    }

    // 결과 처리 (POST)
    @PostMapping("/login")
    public String resultPage(@ModelAttribute User user, Model model) {
        System.out.println("User: " + user);
        model.addAttribute("data1", "@ModelAttribute 예제 - 결과 페이지");
        model.addAttribute("data2",
            "id: " + user.getId() + "<br>pw: " + user.getPw());

        return "pages/viewPage03";
    }
}
```



요청 파라미터(parameter)

- @ModelAttribute로 요청 파라미터 처리

```
<h2>@ModelAttribute 입력 페이지</h2>
```

pages/userForm.html

```
<form th:action="@{/login}" method="post">
  <p>
    ID: <input type="text" name="id" required>
  </p>
  <p>
    PW: <input type="password" name="pw" required>
  </p>
  <p>
    <button type="submit">전송</button>
  </p>
</form>
```

```
<body>
  <h2 th:text="${data1}"></h2>
  <p th:utext="${data2}"></p>
</body>
```

pages/viewPage03.html



회원 관리 프로그램

- 출력 화면

회원 등록

이름:

이메일:

[목록으로](#)

➔

회원 목록

[회원 등록](#)

ID	이름	이메일
1	김선화	today@space.com
2	이정후	player@korea.kr



회원 관리 프로그램

- 회원 등록 – add.html

```
<h1>회원 등록</h1>
<form th:action="@{/members/add}" method="post">
    이름: <input type="text" name = "name"><br>
    이메일: <input type="text" name = "email"><br>
    <button type="submit">등록</button>
</form>
<p><a th:href="@{/members}">목록으로</a></p>
```



회원 관리 프로그램

- 회원 목록 – members.html

```
<h1>회원 목록</h1>
<p><a th:href="@{/members/add}">회원 등록</a></p>
<table>
  <tr>
    <th>ID</th>
    <th>이름</th>
    <th>이메일</th>
  </tr>
  <tr th:each="member : ${members}">
    <td th:text="${member.id}"></td>
    <td th:text="${member.name}"></td>
    <td th:text="${member.email}"></td>
  </tr>
</table>
```



회원 관리 프로그램

- MemberDTO – 자료형 클래스

```
public class Member {  
    private Long id;  
    private String name;  
    private String email;  
  
    public Member() {}  
  
    public Member(Long id, String name, String email) {  
        this.id = id;  
        this.name = name;  
        this.email = email;  
    }  
  
    public Long getId() { return id; }  
    public void setId(Long id) { this.id = id; }  
  
    public String getName() { return name; }  
    public void setName(String name) { this.name = name; }  
  
    public String getEmail() { return email; }  
    public void setEmail(String email) { this.email = email; }  
}
```



회원 관리 프로그램

- 롬복(Lombok) 라이브러리

롬복(Lombok)은 Getter, Setter와 등과 같은 코드를 자동으로 작성해주어 단순 반복되는 코드 작성을 줄일 수 있도록 도와주는 라이브러리이다.

pom.xml에 의존성 등록이 필요함

```
<dependency>  
  <groupId>org.projectlombok</groupId>  
  <artifactId>lombok</artifactId>  
  <optional>true</optional>  
</dependency>
```



회원 관리 프로그램

- MemberDTO – Lombok 사용

```
package com.springboot.model;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@NoArgsConstructor    //기본 생성자
@AllArgsConstructor    //매개변수를 모두 가진 생성자
@Setter             //setter
@Getter             //getter
public class MemberDTO {
    private Long id;        //순번
    private String name;    //이름
    private String email;   //이메일
}
```



회원 관리 프로그램

● MemberDTO – Lombok 사용

```
MemberDTO.java
MemberDTO
    email
    id
    name
    MemberDTO()
    MemberDTO(Long, String, String)
    getEmail() : String
    getId() : Long
    getName() : String
    setEmail(String) : void
    setId(Long) : void
    setName(String) : void
```



회원 관리 프로그램

- MemberRepository – 자료 저장(DAO)

```
@Repository
public class MemberRepository {

    private List<MemberDTO> list = new ArrayList<>();
    private long sequence = 0L;

    //회원 등록
    public MemberDTO save(MemberDTO dto) {
        dto.setId(++sequence);
        list.add(dto);
        return dto;
    }

    //회원 목록
    public List<MemberDTO> findAll(){
        return list;
    }
}
```



회원 관리 프로그램

- MemberController – 컨트롤러

```
@RequestMapping("/members")  
//@AllArgsConstructor //생성자 주입 - 객체 생성  
@Controller  
public class MemberController {  
  
    @Autowired  
    private MemberRepository repository;  
  
    //회원 등록 화면  
    @GetMapping("/add")  
    public String addForm() {  
        return "member/add"; //add.html  
    }  
}
```



회원 관리 프로그램

- MemberController – 컨트롤러

```
//회원 등록 처리
@PostMapping("/add")
public String add(@ModelAttribute MemberDTO dto) {
    //System.out.println("MemberDTO: " + dto);
    repository.save(dto);
    return "redirect:/members";
}

//회원 목록 보기
@GetMapping
public String list(Model model) {
    List<MemberDTO> members = repository.findAll();
    model.addAttribute("members", members);
    return "member/members";
}
}
```

