


## 2. 타임리프 - 스프링 통합과 품

▲ 목차

# 1. 타임리프 스프링 통합

Tutorial: Thymeleaf + Spring

ViewResolvers are the objects in charge of obtaining View objects for a specific operation and locale. Typically, controllers ask ViewResolvers to forward to a view with a specific name (a String returned by the controller method), and then all the view

 <https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html>

03/25/2011	no	Plastic	Fertilizer used	1. Thymus u. citriodora	20
				1. Thymus herbabarona	5
				2. Thymus serpyllum	10

Add new Seed Starter

Seed Starter data

Date planted (MM-dd-yyyy)

Covered ☐

Type

타임리프에서는 위와같은 통합 메뉴얼을 제공한다.

## 추가되는 기능


- 스프링의 SpringEL 문법 통합
- `${@myBean.doSomething() }` 처럼 스프링 빈 호출 지원
- 편리한 폼 관리를 위한 추가 속성
  - `th:object`(기능 강화, 폼 커맨드 객체 선택)
  - `th:field`, `th:errors`, `th:errorclass`
- 폼 컴포넌트 기능
  - checkbox, radio button, List등을 편리하게 사용할 수 있는 기능 지원
- 스프링의 메시지 국제화 기능의 편리한 통합
- 스프링의 검증, 오류 처리 통합
- 스프링의 변환 서비스 통합(ConversionService)


## 스프링 부트에서는 대부분 자동화된다.

타임리프를 템플릿 엔진으로 스프링 빈에 등록하려면, 타임리프용 뷰 리졸버를 스프링 빈으로 등록해야 한다.

Tutorial: Thymeleaf + Spring

ViewResolvers are the objects in charge of obtaining View objects for a specific operation and locale. Typically, controllers ask ViewResolvers to forward to a view with a specific name (a String returned by the controller method), and then all the view

 <https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html#the-springstandard-dialect>



하지만 스프링 부트에서는 build.gradle에 타임리프 관련 의존성만 추가해주면 관련 설정을 모두 자동으로 스프링빈으로 등록해준다.

```
implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
```


build.gradle

그리고 대부분의 타임리프 관련 설정은 application.properties에 추가해 설정 및 변경이 가능하다.

⇒ 스프링 부트가 제공하는 타임리프 설정

Common Application Properties

Various properties can be specified inside your application.properties file, inside your application.yml file, or as command line switches. This appendix provides a list of common Spring Boot properties and references to the underlying classes that consume them. Property contributions can come from additional jar files on your classpath, so you should not consider this

 <https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#common-application-properties-templating>

## 2. 입력 폼 처리

타임리프의 속성을 이용해 입력 폼을 효율적으로 개선할 수 있다.

## 사용법

- `th:object` : 커맨드 객체를 지정한다.
- `*{...}` : 선택 변수 식으로 상위 태그에서 `th:object` 에서 선택한 객체에 접근한다.
- `th:field` : HTML 태그의 `id` , `name` , `value` 속성을 자동으로 처리해준다.
- 사용 예 : 상위 태그에 `itemName` 프로퍼티를 갖는 `object`를 `th:object` 로 선언해준 상태.
  - 렌더링 전

```
<input type="text" th:field="*{itemName}" />
```

- 렌더링 후

```
<input type="text" id="itemName" name="itemName" th:value="*{itemName}"/>
```

이 코드만 보서는 해당 속성의 강력함을 파악하기 쉽지 않으나 뒤에 검증(Validation)에서 이에 관련해 더 소개하도록 한다.

## 3. 요구사항 추가

폼에서 체크박스, 라디오 버튼, 셀렉트 박스를 타임리프를 이용하면 편리하게 사용할 수 있다.

### 3.1 단일 체크박스 1

#### HTML checkbox 주의점

우선 HTML checkbox의 특징으로 check가 되지 않는다면 클라이언트에서 서버로 값을 보내지 않는다는점을 기억해야 한다.

```
<input type="checkbox" id="open" name="open" class="form-check-input" />
```

즉 다음과 같은 태그가 있다고 할 때 내가 선택을 하지 않은 경우 서버에서는 open이라는 프로퍼티가 false가 아니라 아예 전송을 하지 않았기 때문에 `null` 상태이다.

(추가적으로 check를 하면 on이라는 값이 넘어가는데 스프링에선 이 문자를 true로 변환해주며 해당 역할은 스프링 타입 컨버터가 수행한다.)

이러한 특징은 수정의 경우 문제가 될 수 있다. 사용자가 의도적으로 체크를 안하고 저장을 할 경우 아무 값도 넘어가지 않기 때문에 서버 구현에 따라 값이 변경되지 않을 수 있다.(선택→선택해제)

#### 히든 필드를 이용한 해결

스프링 MVC는 히든 필드를 이용해 이런 문제를 해결한다.

방법은 간단하다. 기존 체크박스 태그 앞에 히든 필드를 하나 추가하고 `_open` 과 같이 언더스코어( `_` )를 붙혀 전송하면 체크를 해제했음을 인식할 수 있다. 히든필드는 항상 전송된다는 점을 노린 방식인데, 체크를 해제한 경우 `open` 은 전송되지 않고 `_open` 만 전송되는데 이 경우 스프링 MVC는 체크를 해제했다고 판단한다.

⇒ 체크 해제를 위한 히든 필드

```
<input type="hidden" name="_open" value="on">
```

⇒ 기존 checkbox 코드에 히든 필드 추가

```
<input type="hidden" name="_open" value="on"><!-- 히든 필드 추가 -->
<input type="checkbox" id="open" name="open" class="form-check-input" />
```

이렇게 작성을 하면 체크박스를 선택하지 않고 저장했을 때 서버에서는 `null` 이 아니라 명확하게 `false` 를 받을 수 있다. 이는 스프링 MVC에서 `_open`만 존재하는 것을 확인 후, `open`의 값이 체크되지 않았음을 인식한다.

### 3.2 단일 체크박스 2

3.1에서 언급했던 히든필드는 타임리프의 폼 기능을 사용해 자동으로 생성해줄 수 있다.

결론부터 말하자면 `th:field` 속성을 이용하면 `id`, `name`, `value`, `field`뿐 아니라 `hidden field`와 `checked`속성까지 자동으로 설정해준다.

⇒ `Object.open`의 값은 true이다.

⇒ 렌더링 전 체크박스 태그

```
<input type="text" th:field="*{open}" />
```

⇒ 렌더링 후 체크박스 태그

```
<input type="text" id="open" name="open" th:value="true" checked="checked"/>
<input type="hidden" name="_open" value="on"><!-- 히든 필드 추가 -->
```

### 3.3 멀티 체크박스

체크 박스를 멀티로 사용해서, 하나 이상을 나타나게도 할 수 있고 체크할수도 있게 할 수 있다.

**Tip : @ModelAttribute의 또다른 사용법**

보통 API의 Request로 사용하는 이 어노테이션은 별도로 어노테이션에 붙혀주면 해당 메서드의 반환값이 자동으로 Model에 담기게 된다.

```
@ModelAttribute("data")
public String data(){
    return "testData"
}
```

⇒ data라는 이름으로 해당 속성을 사용할 수 있고 testData가 값으로 사용된다.

사용법

- 렌더링 전 타임리프 코드

```
<div>
  <div>등록 지역</div>
  <div th:each="region : ${regions}" class="form-check form-check-inline">
    <input type="checkbox" th:field="*{regions}" th:value="${region.key}"
      class="form-check-input">
    <label th:for="${#ids.prev('regions')}}"
      th:text="${region.value}" class="form-check-label">서울</label>
  </div>
</div>
```

멀티 체크박스 사용 코드

- regions라는 반복가능한 요소를 `th:each` 로 반복하도록 선언한다.
- `th.field` 속성을 넣으면 `th.value`에 들어가는 `region.key` 값과 비교해 값이 포함되었으면 `checked`가 추가된다. 즉, 자동으로 `value`와 비교해서 `checked` 여부를 설정할 수 있다.
- `th:for="${#ids.prev('regions')}}"`

⇒ 멀티 체크박스는 같은 이름의 여러 체크박스가 생성된다. `name` 속성은 모두 같아도 되지만 `id`는 유일해야하기에 모두 달라야 한다. 타임리프에서는 편의목적으로 `ids`라는 프로퍼티를 제공하며 다음과 같은 메서드도 제공한다. `ids`는 반복하는 요소의 인덱스로 1,2,3처럼 숫자다.

- `ids.prev(...)` : 아이디 앞에 인수값을 문자열 결합해준다. (Ex: `ids.prev('data')→ data1`)
- `ids.next(...)` : 아이디 뒤에 인수값을 문자열 결합해준다. (Ex: `ids.prev('data')→ 1data`)

- 렌더링 후 타임리프 코드

```
<div>
  <div>등록 지역</div>
  <div class="form-check form-check-inline">
    <input type="checkbox" value="SEOUL" class="form-check-input"
      id="regions1" name="regions">
    <input type="hidden" name="_regions" value="on"/>
    <label for="regions1" class="form-check-label">서울</label>
  </div>
  <div class="form-check form-check-inline">
    <input type="checkbox" value="BUSAN" class="form-check-input"
      id="regions2" name="regions">
    <input type="hidden" name="_regions" value="on"/>
    <label for="regions2" class="form-check-label">부산</label>
  </div>
  <div class="form-check form-check-inline">
    <input type="checkbox" value="JEJU" class="form-check-input"
      id="regions3" name="regions">
    <input type="hidden" name="_regions" value="on"/>
    <label for="regions3" class="form-check-label">제주</label>
  </div>
</div>
```

정리

- 다수의 체크박스가 필요한 경우에는 `th:each`를 이용해 요소를 반복 생성할 수 있다.
- `th.field`와 `th.value`를 같이 작성하면 타임리프가 자동으로 두 값을 비교해 `checked` 설정을 해준다.
- 반복 생성되는 태그의 아이디를 모두 다르게 생성해줘야 하기에 `ids`라는 편의 프로퍼티를 제공한다.  
⇒ `ids`에는 `next`, `prev`라는 메서드가 제공되며 인수로 넘겨주는 값이 `id`의 앞/뒤로 결합되 아이디가 된다.

3.4 라디오 버튼

다중 선택이 안되는 라디오 버튼은 선택지 중 하나를 선택할 수 있다.

사용법 자체는 멀티 체크박스를 구현할때와 거의 유사하다.

사용법

- 표현할 라디오버튼이 하나가 아닌이상 `th:each` 속성을 이용해 반복한다.
- 작성법 자체는 멀티 체크박스와 동일하다.
- 예제 코드

```
<div th:object="${item}">
  ...
  <div th:each="type : ${반복요소}" class="form-check form-check-inline">
    <input type="radio" th:field="*{라디오요소}" th:value="${type.비교값}" class="form-check-input">
    <label th:for="${#ids.prev('data')}}" th:text="${type.description}"
      class="form-check-label">
  </div>
</div>
```

BOOK

```
        </label>
    </div>
</div>
```

⇒ 반복요소의 count만큼 내부 태그가 반복되어 생성된다.

⇒ th:field에 들어간 th:object에서 선언한 item의 요소는 th:value의 값과 비교해 **checked** 가 자동으로 표현된다.

⇒ 멀치 체크박스과 동일하게 th:each속성 내부에서 ids를 이용해 인덱스접근및 인수로 넘겨준 값과 문자열 결합으로 고유한 아이디를 만들 수 있다.

## 특징

- 라디오버튼은 아무것도 선택하지 않을 경우 아무 값도 넘어가지 않아 Null이 된다.
- 체크박스와는 다르게 별도의 히든필드가 자동으로 생성되진 않는다.
  - ⇒ 라디오버튼은 선택이 한 번 되면 항상 하나를 선택하도록 되어있어 히든 필드를 사용하지 않는다.

## Tip - 타임리프에서 ENUM(or ETC) 직접 접근하기

타임리프는 SpringEL문법으로 서버에 있는 자바 객체에 직접 접근할수도 있다. 대신 해당 객체의 FQCN(Fully Qualified Class Name)을 알아야 한다.

```
    ${T(FQCN)}
```

⇒ 사용 예 : `${T(catsbi.me.itemservice.domain.item.ItemEnum.ONE).name()}`

⇒ 결과 : **ONE**

⇒ 주의사항: 패키지 경로가 바뀔 경우 자바 컴파일러는 타임리프까지 오류를 잡아주지 못하기에 추천하지 않는 방법이다.

## 3.5 셀렉트 박스

라디오 버튼과 비슷하게 여러선택지 중 하나를 선택할 때 사용할 수 있다.

### 사용법

- 반복요소가 있는 셀렉트박스 역시 th:each로 요소를 반복하며 하나하나의 값을 다룬다.
- 수정페이지에서 기본으로 선택되어야하는 값이 있다면 th:field 요소를 이용해 가능하다.
- 예제 코드

```
<select th:field="${item.deliveryCode}" class="form-select">
    <option value="">==배송 방식 선택==</option>
    <option th:each="deliveryCode : ${deliveryCodes}"
            th:value="${deliveryCode.code}"
            th:text="${deliveryCode.displayName}">FAST
    </option>
</select>
```

⇒ select 태그에 선언된 th:field 속성과 option에서 선언된 th:value를 타임리프가 비교해 일치할 경우 자동으로 selected 된다.

- 렌더링 된 예제 코드는 다음과 같다.

(빠른배송이 선택된 상태라 가정한다.)

```
<select class="form-select" id="deliveryCode" name="deliveryCode">
    <option value="">==배송 방식 선택==</option>
    <option value="FAST" selected="selected">빠른 배송</option>
    <option value="NORMAL">일반 배송</option>
    <option value="SLOW">느린 배송</option>
</select>
```

⇒ th:field가 빠른배송이기에 option중 빠른 배송의 값과 일치하기에 해당 태그에 selected가 추가되었다.