3. 메시지, 국제화

#2.인강/5. 스프링 MVC 2/강의#

목차

프로젝트 설정

이전 프로젝트에 이어서 메시지, 국제화 기능을 학습해보자.
스프링 통합과 폼에서 개발한 상품 관리 프로젝트를 일부 수정해서 `message-start` 라는 프로젝트에 넣어두었다.
참고로 **메시지, 국제화 예제에 집중하기 위해서 복잡한 체크, 셀렉트 박스 관리 기능은 제거**했다.

프로젝트 설정 순서

1. `message-start` 의 폴더 이름을 `message` 로 변경하자.
2. **프로젝트 임포트**
   File ➡ Open ➡ 해당 프로젝트의 `build.gradle` 을 선택하자. 그 다음에 선택창이 뜨는데, Open as Project를 선택하자.
3. `ItemServiceApplication.main()` 을 실행해서 프로젝트가 정상 수행되는지 확인하자.

실행

- http://localhost:8080
- http://localhost:8080/message/items

메시지, 국제화 소개

메시지

악덕? 기획자가 화면에 보이는 문구가 마음에 들지 않는다고, **상품명**이라는 단어를 모두 **상품이름**으로 고쳐달라고 하면 어떻게 해야할까?
여러 화면에 보이는 상품명, 가격, 수량 등, `label`에 있는 단어를 변경하려면 다음 화면들을 다 찾아가면서 모두 변경해야 한다. 지금처럼 화면 수가 적으면 문제가 되지 않지만 화면이 수십개 이상이라면 수십개의 파일을 모두 고쳐야 한다.

- `addForm.html`, `editForm.html`, `item.html`, `items.html`

왜냐하면 해당 HTML 파일에 메시지가 하드코딩 되어 있기 때문이다.

이런 다양한 메시지를 한 곳에서 관리하도록 하는 기능을 메시지 기능이라 한다.

예를 들어서 `messages.properties`라는 메시지 관리용 파일을 만들고

```
item=상품
item.id=상품 ID
item.itemName=상품명
item.price=가격
item.quantity=수량
```

각 HTML들은 다음과 같이 해당 데이터를 key 값으로 불러서 사용하는 것이다.

addForm.html

```
<label for="itemName" th:text="#{item.itemName}"></label>
```

editForm.html

```
<label for="itemName" th:text="#{item.itemName}"></label>
```

국제화

메시지에서 한 발 더 나가보자.
메시지에서 설명한 메시지 파일(`messages.properties`)을 각 나라별로 별도로 관리하면 서비스를 국제화 할 수 있다.
예를 들어서 다음과 같이 2개의 파일을 만들어서 분류한다.

`messages_en.properties`

```
item=Item
item.id=Item ID
```

```
item.itemName=Item Name
item.price=price
item.quantity=quantity
```

`messages_ko.properties`

```
item=상품
item.id=상품 ID
item.itemName=상품명
item.price=가격
item.quantity=수량
```

영어를 사용하는 사람이면 `messages_en.properties` 를 사용하고,
한국어를 사용하는 사람이면 `messages_ko.properties` 를 사용하게 개발하면 된다.

이렇게 하면 사이트를 국제화 할 수 있다.

한국에서 접근한 것인지 영어에서 접근한 것인지는 인식하는 방법은 HTTP `accept-language` 해더 값을 사용하거나 사용자가 직접 언어를 선택하도록 하고, 쿠키 등을 사용해서 처리하면 된다.

메시지와 국제화 기능을 직접 구현할 수도 있겠지만, 스프링은 기본적인 메시지와 국제화 기능을 모두 제공한다. 그리고 타임리프도 스프링이 제공하는 메시지와 국제화 기능을 편리하게 통합해서 제공한다. 지금부터 스프링이 제공하는 메시지와 국제화 기능을 알아보자.

스프링 메시지 소스 설정

스프링은 기본적인 메시지 관리 기능을 제공한다.

메시지 관리 기능을 사용하려면 스프링이 제공하는 `MessageSource` 를 스프링 빈으로 등록하면 되는데, `MessageSource` 는 인터페이스이다. 따라서 구현체인 `ResourceBundleMessageSource` 를 스프링 빈으로 등록하면 된다.

직접 등록

```
```

```
@Bean
public MessageSource messageSource() {
    ResourceBundleMessageSource messageSource = new
ResourceBundleMessageSource();
    messageSource.setBasenames("messages", "errors");
    messageSource.setDefaultEncoding("utf-8");
    return messageSource;
}
```

- `basenames` : 설정 파일의 이름을 지정한다.
  - `messages` 로 지정하면 `messages.properties` 파일을 읽어서 사용한다.
  - 추가로 국제화 기능을 적용하려면 `messages_en.properties` , `messages_ko.properties` 와 같이 파일명 마지막에 언어 정보를 주면된다. 만약 찾을 수 있는 국제화 파일이 없으면 `messages.properties` (언어정보가 없는 파일명)를 기본으로 사용한다.
  - 파일의 위치는 `/resources/messages.properties` 에 두면 된다.
  - 여러 파일을 한번에 지정할 수 있다. 여기서는 `messages` , `errors` 둘을 지정했다.
- `defaultEncoding` : 인코딩 정보를 지정한다. `utf-8` 을 사용하면 된다.

스프링 부트

스프링 부트를 사용하면 스프링 부트가 `MessageSource` 를 자동으로 스프링 빈으로 등록한다.

스프링 부트 메시지 소스 설정

스프링 부트를 사용하면 다음과 같이 메시지 소스를 설정할 수 있다.

`application.properties`

```
spring.messages.basename=messages,config.i18n.messages
```

스프링 부트 메시지 소스 기본 값

`spring.messages.basename=messages`

`MessageSource` 를 스프링 빈으로 등록하지 않고, 스프링 부트와 관련된 별도의 설정을 하지 않으면 `messages` 라는 이름으로 기본 등록된다. 따라서 `messages_en.properties` , `messages_ko.properties` , `messages.properties` 파일만 등록하면 자동으로 인식된다.

메시지 파일 만들기

메시지 파일을 만들어보자. 국제화 테스트를 위해서 `messages_en` 파일도 추가하자.

- `messages.properties` :기본 값으로 사용(한글)
- `messages_en.properties` : 영어 국제화 사용

주의! 파일명은 massage가 아니라 messages다! 마지막 s에 주의하자

`/resources/messages.properties`

messages.properties

```
hello=안녕
hello.name=안녕 {0}
```

`/resources/messages_en.properties`

messages_en.properties

```
hello=hello
hello.name=hello {0}
```

스프링 메시지 소스 사용

MessageSource 인터페이스

```
public interface MessageSource {

  String getMessage(String code, @Nullable Object[] args, @Nullable String
defaultMessage, Locale locale);

  String getMessage(String code, @Nullable Object[] args, Locale locale) throws
NoSuchMessageException;
```

`MessageSource` 인터페이스를 보면 코드를 포함한 일부 파라미터로 메시지를 읽어오는 기능을 제공한다.

스프링이 제공하는 메시지 소스를 어떻게 사용하는지 테스트 코드를 통해서 학습해보자.

`test/java/hello/itemservice/message.MessageSourceTest.java`

```
package hello.itemservice.message;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.context.MessageSource;

import static org.assertj.core.api.Assertions.*;

@SpringBootTest
public class MessageSourceTest {

    @Autowired
    MessageSource ms;

    @Test
    void helloMessage() {
        String result = ms.getMessage("hello", null, null);
        assertThat(result).isEqualTo("안녕");
    }
}
```

- `ms.getMessage("hello", null, null)`
  - **code**: `hello`
  - **args**: `null`
  - **locale**: `null`

가장 단순한 테스트는 메시지 코드로 `hello`를 입력하고 나머지 값은 `null`을 입력했다.
`locale` 정보가 없으면 `basename`에서 설정한 기본 이름 메시지 파일을 조회한다. `basename`으로 `messages`를 지정 했으므로 `messages.properties`파일에서 데이터 조회한다.

MessageSourceTest 추가 - 메시지가 없는 경우, 기본 메시지

```
@Test
```

```
void notFoundMessageCode() {
    assertThatThrownBy(() -> ms.getMessage("no_code", null, null))
            .isInstanceOf(NoSuchMessageException.class);
}

@Test
void notFoundMessageCodeDefaultMessage() {
    String result = ms.getMessage("no_code", null, "기본 메시지", null);
    assertThat(result).isEqualTo("기본 메시지");
}
```

- 메시지가 없는 경우에는 `NoSuchMessageException` 이 발생한다.
- 메시지가 없어도 기본 메시지(`defaultMessage`)를 사용하면 기본 메시지가 반환된다.

MessageSourceTest 추가 - 매개변수 사용

```
@Test
void argumentMessage() {
    String result = ms.getMessage("hello.name", new Object[]{"Spring"}, null);
    assertThat(result).isEqualTo("안녕 Spring");
}
```

다음 메시지의 {0} 부분은 매개변수를 전달해서 치환할 수 있다.

`hello.name=안녕 {0}` ➡ Spring 단어를 매개변수로 전달 ➡ `안녕 Spring`

국제화 파일 선택

locale 정보를 기반으로 국제화 파일을 선택한다.

- Locale이 `en_US` 의 경우 `messages_en_US` ➡ `messages_en` ➡ `messages` 순서로 찾는다.
- `Locale` 에 맞추어 구체적인 것이 있으면 구체적인 것을 찾고, 없으면 디폴트를 찾는다고 이해하면 된다.

MessageSourceTest 추가 - 국제화 파일 선택1

```
@Test
void defaultLang() {
    assertThat(ms.getMessage("hello", null, null)).isEqualTo("안녕");
    assertThat(ms.getMessage("hello", null, Locale.KOREA)).isEqualTo("안녕");
```

```
}
```

- `ms.getMessage("hello", null, null)` : locale 정보가 없으므로 `messages` 를 사용
- `ms.getMessage("hello", null, Locale.KOREA)` : locale 정보가 있지만, `message_ko` 가 없으므로 `messages` 를 사용

MessageSourceTest 추가 - 국제화 파일 선택2

```
@Test
void enLang() {
    assertThat(ms.getMessage("hello", null,
Locale.ENGLISH)).isEqualTo("hello");
}
```

- `ms.getMessage("hello", null, Locale.ENGLISH)` : locale 정보가 `Locale.ENGLISH` 이므로 `messages_en` 을 찾아서 사용

웹 애플리케이션에 메시지 적용하기

실제 웹 애플리케이션에 메시지를 적용해보자.

먼저 메시지를 추가 등록하자.

`messages.properties`

```
label.item=상품
label.item.id=상품 ID
label.item.itemName=상품명
label.item.price=가격
label.item.quantity=수량

page.items=상품 목록
page.item=상품 상세
page.addItem=상품 등록
page.updateItem=상품 수정
```

```
button.save=저장
button.cancel=취소
```

타임리프 메시지 적용

타임리프의 메시지 표현식 `#{...}`를 사용하면 스프링의 메시지를 편리하게 조회할 수 있다.
예를 들어서 방금 등록한 상품이라는 이름을 조회하려면 `#{label.item}`이라고 하면 된다.

렌더링 전

```
<div th:text="#{label.item}"></h2>
```

렌더링 후

```
<div>상품</h2>
```

타임리프 템플릿 파일에 메시지를 적용해보자.

적용 대상

`addForm.html`
`editForm.html`
`item.html`
`items.html`

addForm.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="utf-8">
    <link th:href="@{/css/bootstrap.min.css}"
          href="../css/bootstrap.min.css" rel="stylesheet">
```

```
    <style>
        .container {
            max-width: 560px;
        }
    </style>
</head>
<body>

<div class="container">

    <div class="py-5 text-center">
        <h2 th:text="#{page.addItem}">상품 등록</h2>
    </div>

    <h4 class="mb-3">상품 입력</h4>

    <form action="item.html" th:action th:object="${item}" method="post">
        <div>
            <label for="itemName" th:text="#{label.item.itemName}">상품명</
label>
            <input type="text" id="itemName" th:field="*{itemName}"
class="form-control" placeholder="이름을 입력하세요">
        </div>
        <div>
            <label for="price" th:text="#{label.item.price}">가격</label>
            <input type="text" id="price" th:field="*{price}" class="form-
control" placeholder="가격을 입력하세요">
        </div>
        <div>
            <label for="quantity" th:text="#{label.item.quantity}">수량</label>
            <input type="text" id="quantity" th:field="*{quantity}"
class="form-control" placeholder="수량을 입력하세요">
        </div>

        <hr class="my-4">

        <div class="row">
            <div class="col">
                <button class="w-100 btn btn-primary btn-lg" type="submit"
```

```
th:text="#{button.save}">저장</button>
            </div>
            <div class="col">
                <button class="w-100 btn btn-secondary btn-lg"
                        onclick="location.href='items.html'"
                        th:onclick="|location.href='@{/message/items}'|"
                        type="button" th:text="#{button.cancel}">취소</button>
            </div>
        </div>

    </form>

</div> <!-- /container -->
</body>
</html>
```

페이지 이름에 적용

- `<h2>상품 등록 폼</h2>`
  - `<h2 th:text="#{page.addItem}">상품 등록</h2>`

레이블에 적용

- `<label for="itemName">상품명</label>`
  - `<label for="itemName" th:text="#{label.item.itemName}">상품명</label>`
  - `<label for="price" th:text="#{label.item.price}">가격</label>`
  - `<label for="quantity" th:text="#{label.item.quantity}">수량</label>`

버튼에 적용

- `<button type="submit">상품 등록</button>`
  - `<button type="submit" th:text="#{button.save}">저장</button>`
  - `<button type="button" th:text="#{button.cancel}">취소</button>`

editForm.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="utf-8">
```

```
    <link th:href="@{/css/bootstrap.min.css}"
          href="../css/bootstrap.min.css" rel="stylesheet">
    <style>
        .container {
            max-width: 560px;
        }
    </style>
</head>
<body>

<div class="container">

    <div class="py-5 text-center">
        <h2 th:text="#{page.updateItem}">상품 수정</h2>
    </div>

    <form action="item.html" th:action th:object="${item}" method="post">
        <div>
            <label for="id" th:text="#{label.item.id}">상품 ID</label>
            <input type="text" id="id" th:field="*{id}" class="form-control"
readonly>
        </div>
        <div>
            <label for="itemName" th:text="#{label.item.itemName}">상품명</
label>
            <input type="text" id="itemName" th:field="*{itemName}"
class="form-control">
        </div>
        <div>
            <label for="price" th:text="#{label.item.price}">가격</label>
            <input type="text" id="price" th:field="*{price}" class="form-
control">
        </div>
        <div>
            <label for="quantity" th:text="#{label.item.quantity}">수량</label>
            <input type="text" id="quantity" th:field="*{quantity}"
class="form-control">
        </div>
```

```
        <hr class="my-4">

        <div class="row">
            <div class="col">
                <button class="w-100 btn btn-primary btn-lg" type="submit"
th:text="#{button.save}">저장</button>
            </div>
            <div class="col">
                <button class="w-100 btn btn-secondary btn-lg"
                        onclick="location.href='item.html'"
                        th:onclick="|location.href='@{/message/items/{itemId}
(itemId=${item.id})}'|"
                        type="button" th:text="#{button.cancel}">취소</button>
            </div>
        </div>

    </form>

</div> <!-- /container -->
</body>
</html>
```

item.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="utf-8">
    <link th:href="@{/css/bootstrap.min.css}"
            href="../css/bootstrap.min.css" rel="stylesheet">
    <style>
        .container {
            max-width: 560px;
        }
    </style>
</head>
<body>
```

```
<div class="container">

    <div class="py-5 text-center">
        <h2 th:text="#{page.item}">상품 상세</h2>
    </div>

    <!-- 추가 -->
    <h2 th:if="${param.status}" th:text="'저장 완료'"></h2>

    <div>
        <label for="itemId" th:text="#{label.item.id}">상품 ID</label>
        <input type="text" id="itemId" name="itemId" class="form-control"
value="1" th:value="${item.id}" readonly>
    </div>
    <div>
        <label for="itemName" th:text="#{label.item.itemName}">상품명</label>
        <input type="text" id="itemName" name="itemName" class="form-control"
value="상품A" th:value="${item.itemName}" readonly>
    </div>
    <div>
        <label for="price" th:text="#{label.item.price}">가격</label>
        <input type="text" id="price" name="price" class="form-control"
value="10000" th:value="${item.price}" readonly>
    </div>
    <div>
        <label for="quantity" th:text="#{label.item.quantity}">수량</label>
        <input type="text" id="quantity" name="quantity" class="form-control"
value="10" th:value="${item.quantity}" readonly>
    </div>

    <hr class="my-4">

    <div class="row">
        <div class="col">
            <button class="w-100 btn btn-primary btn-lg"
                    onclick="location.href='editForm.html'"
                    th:onclick="|location.href='@{/message/items/{itemId}/
edit(itemId=${item.id})}'|"
```

```
                    type="button" th:text="#{page.updateItem}">상품 수정</button>
        </div>
        <div class="col">
            <button class="w-100 btn btn-secondary btn-lg"
                    onclick="location.href='items.html'"
                    th:onclick="|location.href='@{/message/items}'|"
                    type="button" th:text="#{page.items}">목록으로</button>
        </div>
    </div>

</div> <!-- /container -->
</body>
</html>
```

items.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="utf-8">
    <link th:href="@{/css/bootstrap.min.css}"
          href="../css/bootstrap.min.css" rel="stylesheet">
</head>
<body>

<div class="container" style="max-width: 600px">
    <div class="py-5 text-center">
        <h2 th:text="#{page.items}">상품 목록</h2>
    </div>

    <div class="row">
        <div class="col">
            <button class="btn btn-primary float-end"
                    onclick="location.href='addForm.html'"
                    th:onclick="|location.href='@{/message/items/add}'|"
                    type="button" th:text="#{page.addItem}">상품 등록</button>
        </div>
```

```
    </div>

    <hr class="my-4">
    <div>
        <table class="table">
            <thead>
            <tr>
                <th th:text="#{label.item.id}">ID</th>
                <th th:text="#{label.item.itemName}">상품명</th>
                <th th:text="#{label.item.price}">가격</th>
                <th th:text="#{label.item.quantity}">수량</th>
            </tr>
            </thead>
            <tbody>
            <tr th:each="item : ${items}">
                <td><a href="item.html" th:href="@{/message/items/{itemId}
(itemId=${item.id})}" th:text="${item.id}">회원id</a></td>
                <td><a href="item.html" th:href="@{|/message/items/$
{item.id}|}" th:text="${item.itemName}">상품명</a></td>
                <td th:text="${item.price}">10000</td>
                <td th:text="${item.quantity}">10</td>
            </tr>
            </tbody>
        </table>
    </div>

</div> <!-- /container -->

</body>
</html>
```

다음 부분을 잘 확인해서 메시지를 사용하도록 적용하자.

```
<th>ID</th>
<th>상품명</th>
<th>가격</th>
<th>수량</th>
```

```
<th th:text="#{label.item.id}">ID</th>
<th th:text="#{label.item.itemName}">상품명</th>
<th th:text="#{label.item.price}">가격</th>
<th th:text="#{label.item.quantity}">수량</th>
```

실행

잘 동작하는지 확인하기 위해 `messages.properties` 파일의 내용을 가격 ➡ 금액과 같이 변경해서 확인해보자. 정상 동작하면 다시 돌려두자.

참고로 파라미터는 다음과 같이 사용할 수 있다.

`hello.name=안녕 {0}`

`<p th:text="#{hello.name(${item.itemName})}"></p>`

정리

지금까지 메시지를 효율적으로 관리하는 방법을 알아보았다. 이제 여기에 더해서 국제화를 웹 애플리케이션에 어떻게 적용하는지 알아보자.

웹 애플리케이션에 국제화 적용하기

이번에는 웹 애플리케이션에 국제화를 적용해보자. 먼저 영어 메시지를 추가하자.

`messages_en.properties`

```
label.item=Item
label.item.id=Item ID
label.item.itemName=Item Name
label.item.price=price
label.item.quantity=quantity

page.items=Item List
page.item=Item Detail
```

```
page.addItem=Item Add
page.updateItem=Item Update

button.save=Save
button.cancel=Cancel
```

사실 이것으로 국제화 작업은 거의 끝났다. 앞에서 템플릿 파일에는 모두 `#{...}`를 통해서 메시지를 사용하도록 적용해두었기 때문이다.

웹으로 확인하기

웹 브라우저의 언어 설정 값을 변경하면서 국제화 적용을 확인해보자.
크롬 브라우저 ➡ 설정 ➡ 언어를 검색하고, 우선 순위를 변경하면 된다.
우선순위를 영어로 변경하고 테스트해보자.
웹 브라우저의 언어 설정 값을 변경하면 요청시 `Accept-Language`의 값이 변경된다.

`Accept-Language`는 클라이언트가 서버에 기대하는 언어 정보를 담아서 요청하는 HTTP 요청 헤더이다.
(더 자세한 내용은 모든 개발자를 위한 HTTP 웹 기본지식 강의를 참고하자.)

스프링의 국제화 메시지 선택

앞서 `MessageSource` 테스트에서 보았듯이 메시지 기능은 `Locale` 정보를 알아야 언어를 선택할 수 있다.

결국 스프링도 `Locale` 정보를 알아야 언어를 선택할 수 있는데, 스프링은 언어 선택시 기본으로 `Accept-Language` 헤더의 값을 사용한다.

LocaleResolver

스프링은 `Locale` 선택 방식을 변경할 수 있도록 `LocaleResolver` 라는 인터페이스를 제공하는데,
스프링 부트는 기본으로 `Accept-Language`를 활용하는 `AcceptHeaderLocaleResolver`를 사용한다.

LocaleResolver 인터페이스

```
public interface LocaleResolver {

  Locale resolveLocale(HttpServletRequest request);
  void setLocale(HttpServletRequest request, @Nullable HttpServletResponse
response, @Nullable Locale locale);
```

```
}
```

LocaleResolver 변경

만약 `Locale` 선택 방식을 변경하려면 `LocaleResolver`의 구현체를 변경해서 쿠키나 세션 기반의 `Locale` 선택 기능을 사용할 수 있다. 예를 들어서 고객이 직접 `Locale`을 선택하도록 하는 것이다.
관련해서 `LocaleResolver`를 검색하면 수 많은 예제가 나오니 필요한 분들은 참고하자.

정리