SpringMvc2: 섹션3 메시지 정리 자료

💡 목 차

메시지란?
국제화란?
스프링 MessageSource 설정하기
    스프링
    스프링 부트
    메시지 파일 만들기
스프링 메시지 소스 사용
    학습 테스트
실제 웹 애플리케이션에 메시지 & 국제화 적용하기
LocaleResolver 직접 구현해보기
    스프링의 국제화 메시지 선택
    LocaleResolver 구현 방법

메시지란?

- 스프링에서 메시지(message)란 국제화(i18n) 및 외부화된 메시지를 관리하기 위한 기능
- 웹 애플리케이션 개발에서, 사용자에게 표시되는 텍스트 메시지는 html 등 코드 안에 하드 코딩 되기 보다는 별도의 리소스 파일 (`messages.properties` 등)로 관리되어야 한다.

- 텍스트 메시지 별도 관리의 장점은 다음과 같다.
    - 메시지 변경 시 코드를 수정할 필요가 없다.
    - 언어나 국가에 따라 국제화된 메시지를 제공할 수 있다.

- 텍스트 메시지 관리 예시
    - `messages.properties` (resources 파일 하부에 위치)

```
welcome.message=Welcome!
item.id=ID
item.name=Name
```

    - 타임 리프를 통한 메시지 표기

```
<h1 th:text="#{welcome.message}">Welcome!</h1>
<label th:text="#{item.id}">ID</label>
<label th:text="#{item.name}">Name</label>
```

        - `th:text` 와 `#{...}` 표기법을 사용하여 `messages.properties` 내부의 메시지를 참조할 수 있다.
        - 메시지 파일은 `MessageSource` 를 통해 참조할 수 있다.

국제화란?

- **국제화(Internationalization, i18n)**란 소프트웨어 제품을 다양한 언어와 문화권에서 사용할 수 있도록 설계하고 준비하는 과정.
    - (🙂 참고로 kubernetes 를 중간 글자 8자를 생략하여 k8s로 표현하기도 하는데,
      i18n도 마찬가지로 중간 18자를 생략하여 간략히 표기한 것입니다.)
    - 국가 마다 당연하게도 용어가 다르다.

        - ex) Product - 상품, Order - 주문 , Customer - 고객
        - 만약 뷰 파일에 하드코딩이 되어 있다면 국가별 서비스에서 적게는 화면 파일 수십 개, 많게는 그 수백, 수천 개 그 이상으로 하드코딩된 파일을 수정해야 한다.

          → 자바에서 지원하는 `java.util.Locale` 클래스와 스프링에서 지원하는 `org.springframework.context.MessageSource` 로 편리하게 국제화를 수행할 수 있다.
    - **참고)** 주요 국제화 작업
        1. 텍스트 문자열 외부화 및 다국어 번역
        2. 날짜/시간/숫자/화폐 형식 지역화
        3. 문자 인코딩 및 유니코드 지원
        4. 레이아웃 및 방향성(LTR/RTL) 지원
        5. 아이콘 및 이미지 지역화

- 메시지에서 메시지 파일(messages.properties)을 각 나라별로 별도로 관리하면 서비스를 국제화 할 수 있다.
- `messages_ko.properties`

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

- `messages_en.properties`

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

  Locale에 따라서,
    - 영어를 사용하는 사람이면 messages_en.properties 를 사용하고,
    - 한국어를 사용하는 사람이면 messages_ko.properties 를 사용하게 개발.

스프링 MessageSource 설정하기

스프링

- 스프링이 지원하는 메시지 관리 기능을 사용하려면 `MessageSource` 를 스프링 빈으로 등록하면 된다.
    - `MessageSource` 는 인터페이스이므로 다음 Config에서는 구현체인 `ResourceBundleMessageSource` 를 생성했다.
- 스프링에서 MessageSource에 대한 Config 설정하는 법

```
@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 같
    이 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

▼ **[자동 구성(Auto-Configuration)]** 단, 스프링 부트 3.2.3 기준으로는 messages.properties 파일이 resources 폴더 내부에 존재해야 MessageSource가 등록 된다. (해당 메시지 파일이 없다면 messageSource 빈이 등록되지 않는다.)

```
package hello.itemservice;

import lombok.extern.slf4j.Slf4j;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;

import java.util.Arrays;

@Slf4j
@SpringBootApplication
public class ItemServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(ItemServiceApplication.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
        return args -> {
            System.out.println("Let's inspect the beans provided by Spring Boot:");

            String[] beanNames = ctx.getBeanDefinitionNames();
            Arrays.sort(beanNames);
            for (String beanName : beanNames) {
                if(beanName.startsWith("message"))  {
                    log.info(beanName);
                }
            }
        };
    }
}
```

```
2024-03-24T17:38:35.048+09:00  INFO 25272 --- [           main] h.itemservic
 .ItemServiceApplication     : messageConverters
2024-03-24T17:38:35.048+09:00  INFO 25272 --- [           main] h.itemservic
 .ItemServiceApplication     : messageItemController
2024-03-24T17:38:35.048+09:00  INFO 25272 --- [           main] h.itemservic
 .ItemServiceApplication     : messageSource
2024-03-24T17:38:35.048+09:00  INFO 25272 --- [           main] h.itemservic
 .ItemServiceApplication     : messageSourceProperties
```

- **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 인식**된다.

메시지 파일 만들기

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

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

- `messages_en.properties` : 영어 국제화 사용

  ```
  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.
- `java.util.Locale` (지역) 정보를 받는 파라미터가 있음을 알 수 있다.

학습 테스트

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

```
@SpringBootTest
public class MessageSourceTest {
    @Autowired
    MessageSource ms;

    @DisplayName("MessageSource의 getMessage를 통해 code값을 전달해 메시지를 가져올 수 있다.")
    @Test
    void helloMessage() {
        String result = ms.getMessage("hello", null, null);
        assertThat(result).isEqualTo("안녕");
    }
}
```

▼ 원래 성공이 나와야 하는데 실패가 출력되었다면? 이렇게 고친다. (한글 인코딩 문제)

```
expected: "안녕"
 but was: "??"
    at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance0(Native Me
    at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance(NativeCons
    at java.base/jdk.internal.reflect.DelegatingConstructorAccessorImpl.newInstance(Delega
    at java.base/java.lang.reflect.Constructor.newInstanceWithCaller(Constructor.java:499)
    at hello.itemservice.message.MessageSourceTest.helloMessage(MessageSourceTest.java:21)
```

- 한글 인코딩 문제인 것 같아서 문제를 찾아서 https://www.inflearn.com/questions/277955/한글-인코딩-관련-질문입니다

57

joonsang • 2021.08.07

안녕하세요 ~

똑같은 로그가 나와 공유 드립니다.

질문자님 말처럼, 아래 3개를 UTF-8 로 바꿉니다.

```
File >> Settings >> Editor >> File Encodings >> Global
File >> Settings >> Editor >> File Encodings >> Project Encoding
File >> Settings >> Editor >> File Encodings >> Properties File
```

변경 후, properties 파일을 확인해보면 한글이 "??" 으로 바뀌어 있을거에요.

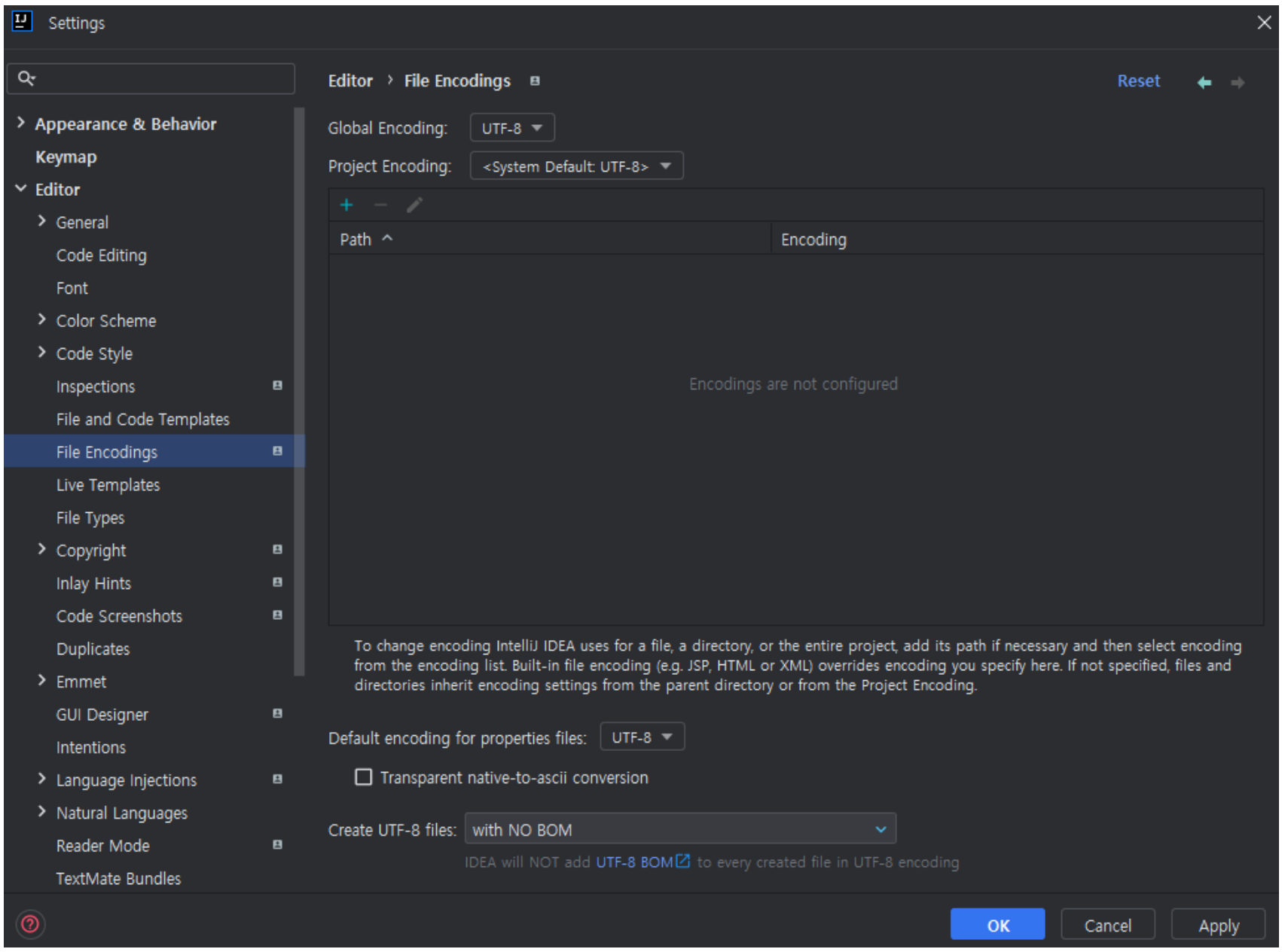
?? 을 다시 "안녕" 으로 수정하여 테스트 돌려보시면 될것 같네요.

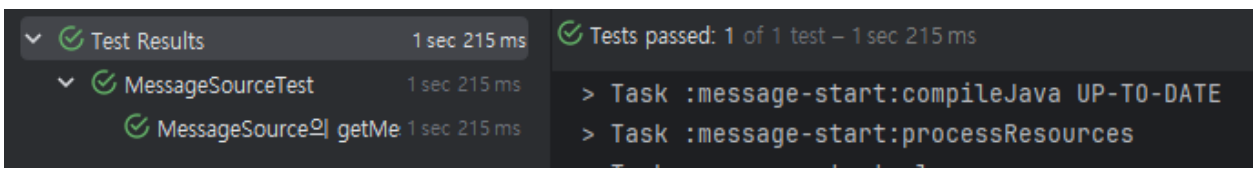
참고로 UTF-8 로 변경해도 변화가 없으실 경우

```
File >> Invaildate Caches >> Reset
```

한 번 해주시면 좋을듯 하네요.

그럼 열공하세요.





- `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

    > locale = null 인 경우 시스템 기본 locale 이 ko_KR 이므로 messages_ko.properties 조회 시도 → 조회 실패 →
    > `messages.properties` 조회

  - `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.

실제 웹 애플리케이션에 메시지 & 국제화 적용하기

- **메시지를 등록하기**

    - 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
```

- messages_en.properties

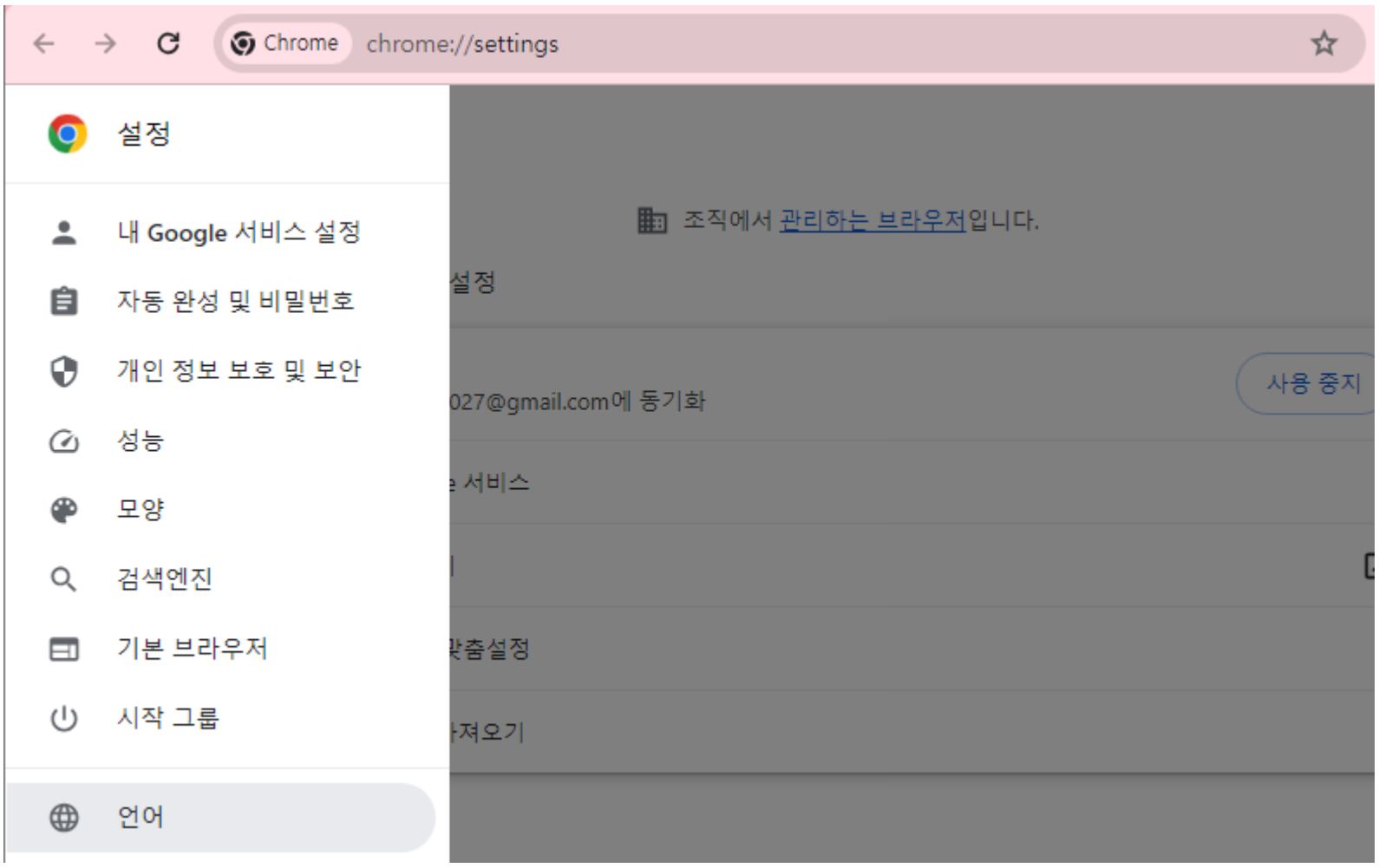
```
label.item=Item
label.item.id=Item ID
label.item.itemName=Item Name
label.item.price=price
label.item.quantity=quantity
page.items=Item List
page.item=Item Detail
page.addItem=Item Add
page.updateItem=Item Update
button.save=Save
button.cancel=Cancel
```

    - 템플릿 뷰에서는 `th:text="{(messages.properties에 등록된 코드명)}"` 문법으로 `MessageSource`의 `getMessage(…)` 메서드에 접근한다.

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

        - `page.addItem` 은 브라우저 설정에 따라 다른 언어로 출력된다.
            - `Accept-Language` 헤더가 US-en이 우선이면 스프링이 Locale에 따라 `messages_en.properties` 에 존재하는 `“Item Add”`로 치환되어 출력된다.

- **웹으로 확인하기**
    - 웹 브라우저의 언어 설정 값을 변경하면서 국제화 적용을 확인해보자.
    - 크롬 브라우저 설정 언어를 검색하고, 우선 순위를 변경하면 된다
        1. chrome://settings/
        2. 언어 → 기본 언어



- 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.)

LocaleResolver 직접 구현해보기

스프링의 국제화 메시지 선택

- **메시지 기능은 `Locale` 정보를 알아야 언어를 선택**할 수 있다.
- 결국 **스프링도 `Locale` 정보를 알아야 언어를 선택할 수 있는데, 스프링은 언어 선택시 기본으로 `Accept-Language` 헤더의 값을 사용**한다.

- **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

LocaleResolver 구현 방법

1. LocaleResolver 인터페이스를 구현하는 클래스를 만듭니다.

```
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.LocaleResolver;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.Locale;

@Configuration
public class CustomLocaleResolver implements LocaleResolver {

    @Override
    public Locale resolveLocale(HttpServletRequest request) {
        // 세션에서 Locale 정보를 가져옵니다.
        Locale locale = (Locale) request.getSession().getAttribute("sessionLocale");

        // 세션에 Locale 정보가 없다면 기본 Locale을 반환합니다.
        return (locale != null) ? locale : Locale.getDefault();
    }

    @Override
    public void setLocale(HttpServletRequest request, HttpServletResponse response, Locale lo
        // 세션에 Locale 정보를 저장합니다.
        request.getSession().setAttribute("sessionLocale", locale);
    }
}
```

2. 웹 애플리케이션에서 Locale 변경 요청을 처리할 컨트롤러 메서드를 작성합니다.

(MessageItemController 에 다음과 같이 작성)

```
//2. 웹 애플리케이션에서 Locale 변경 요청을 처리할 컨트롤러 메서드를 작성합니다.
@GetMapping("/changeLocale")
public String changeLocale(@RequestParam("lang") String language, HttpServletRequest request)
    // 요청 파라미터로 받은 언어 코드를 이용해 Locale을 생성합니다.
    Locale locale = new Locale(language);

    // LocaleResolver를 이용해 세션에 Locale 정보를 저장합니다.
    request.getSession().setAttribute("sessionLocale", locale);

    // 원하는 뷰 이름을 반환합니다.
    return "redirect:/message/items";
}
```

3. 애플리케이션에 CustomLocaleResolver를 등록합니다.

```
package hello.itemservice.localeresolver;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.LocaleResolver;

@Configuration
public class WebConfig {

    @Bean
    public LocaleResolver localeResolver() {
        return new CustomLocaleResolver();
    }
}
```

- 이제 `/changeLocale?lang=ko` 와 같은 요청을 보내면 세션에 한국어 로케일 정보가 저장되고, 애플리케이션에서 해당 로케일을 사용하게 됩니다.
- 쿠키 기반으로 구현하려면 `CookieLocaleResolver`를 사용하면 됩니다. 세부 구현 방법은 위 예제와 비슷하지만, 로케일 정보를 세션 대신 쿠키에 저장하게 됩니다.
- 필요한 경우 `LocaleChangeInterceptor`를 사용하여 요청 파라미터로 로케일 변경을 처리할 수도 있습니다.

4. **(Optional) items.html에 추가**

```
<!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>
    <form th:action="@{/message/items/changeLocale}" method="get">
        <label for="localeSelect">Select Language:</label>
        <select id="localeSelect" name="lang">
            <option value="en">English</option>
            <option value="ko">Korean</option>
            <!-- 추가적인 언어 옵션을 더 추가할 수 있습니다. -->
        </select>
        <button type="submit">Change Locale</button>
    </form>
```

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

- 폼의 `action` 속성은 앞서 설명한 `/changeLocale` 엔드포인트로 향하고 있습니다.
- 셀렉트박스에는 `en` (영어), `ko` (한국어), `ja` (일본어) 옵션이 포함되어 있습니다. 필요에 따라 추가적인 언어 옵션을 더 추가할 수 있습니다.
- `name` 속성은 `lang`으로 설정되어 있습니다. 이는 앞서 설명한 `/changeLocale` 엔드포인트에서 `@RequestParam("lang")` 파라미터와 매핑됩니다.
- 변경된 Locale에 따라 `th:text="#{welcome.message}"` 부분에 해당 언어의 메시지가 렌더링될 것입니다.

- English를 선택했을 경우 (예시)