# Spring MVC2 05 [ 메시지, 국제화 ]

스프링 MVC 2편 - 백엔드 웹 개발 활용 기술

김영한

2022.10.20

## 메시지?

if ? 화면에 보이는 상품명이란 단어를 상품이름으로 다 고치고 싶어졌을때

상품명이라는 코드를 하나하나 고쳐야한다 → HTML이 하드코딩되어있끼 때문에

이러한 다양한 **label**를 한 곳에서 관리하도록 하는 기능을 메시지 기능이라 한다**.**

하드 코딩되어 있는 label이름등 등을 한번에 관리하는 것을 메시지 라고 한다.

## 국제화?

메시지 설정 파일을 나라별로 관리하게되면? 그 나라에 따른 label이 표출 될 수 있는 것.

사이트를 국제화 하는 것 → 메시지 파일을 여러 언어로 정의하는 것

접근하는 HTTP 헤더 나 사용자가 직접 언어를 선택하거나 쿠키 등을 사용해 처리

→ 메시지, 국제화 기능을 직접 구현도 가능하지만 스프링은 기본적으로 제공한다.

타임리프도 스프링 통합으로 편리하게 제공한다.

# 스프링 메시지 파일 설정

## 스프링 프레임워크시 빈등록

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

메시지

MessageSource 를 스프링 빈으로 등록하면 되는데, MessageSource 는 인터페이스이다.
구현체인 ResourceBundleMessageSource 를 스프링 빈으로 등록하면 된다.

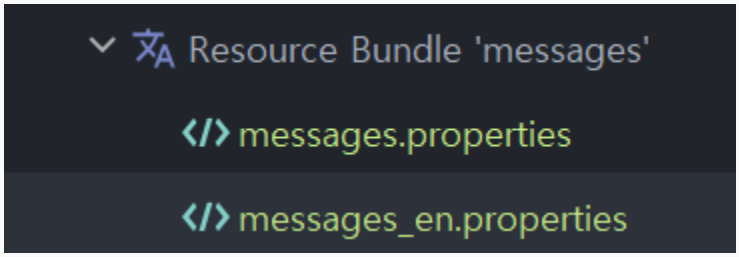
## 스프링 부트에서 등록

application.properties

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

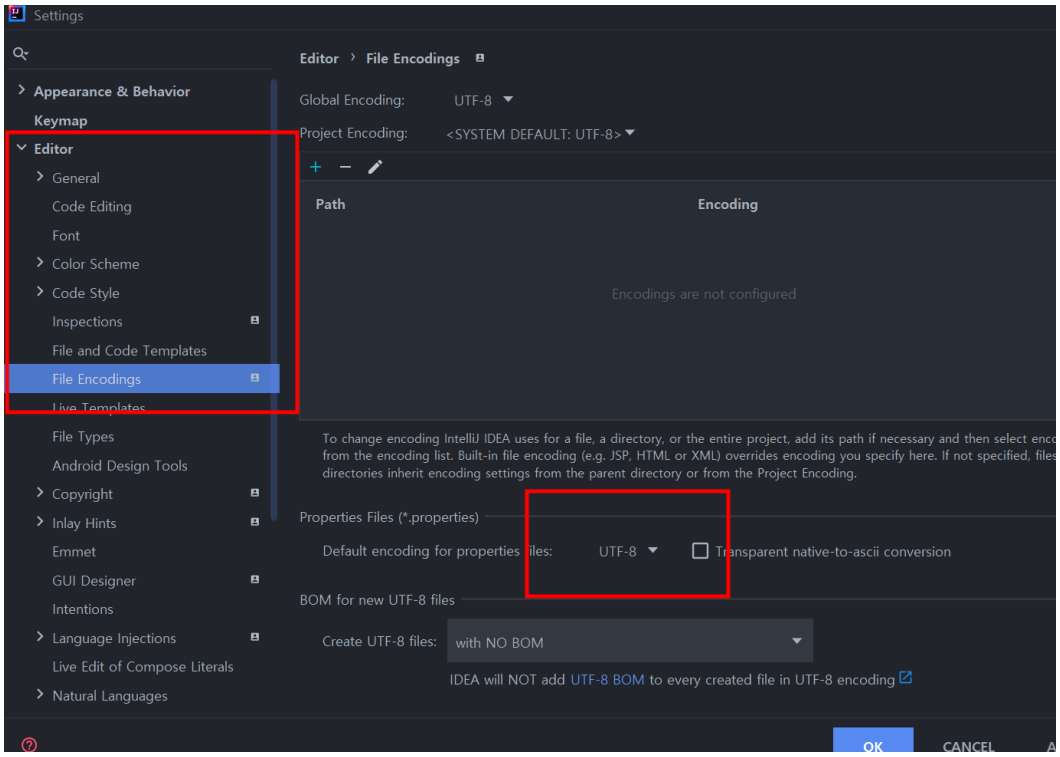
default ⇒ `spring.messages.basename=message; [하나일 때 생략 가능]`

`스프링 부트에서는 기본으로 적용되고 메시지 파일을 만들어 사용하면 된다.`




`messages.properties 생성.`

주의점! 인텔리제이 **properties** 파일 인코딩을 변경하고 해야 한글 문제가 생기지 않는다.




미리만들어 둔 properties 파일이 있다면 인코딩을 변경하게 되었을시 ??? 등으로 인식 못하고 있을 수 있음. 정상적으로 수정 후 동작하면 문제없다.

```
hello=안녕
hello.name=안녕 {0}
```

`설정 후 동작 확인`

⇒ 결국 MessageSource 인터페이스가 빈 등록이 되어있는 것.

테스트로 확인하게 되면

## 테스트1. 메시지 확인

```
@SpringBootTest
public class MessageSourceTest {
    @Autowired
    MessageSource ms;
    @Test
    void helloMessage(){
```

```
        String result = ms.getMessage("hello", null, null);
                                        //code, argos, local 정보
        Assertions.assertThat(result).isEqualTo("안녕");
    }
}
```

테스트가 정상동작하는 것을 알 수 있다.

## 테스트**2.** 없는 메시지 확인

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

없는 이름을 출력시 Exception 확인

## 테스트 **3. default name** 설정

```
@Test
void notFoundMessageCodeDefault(){
    Assertions.assertThat(ms.getMessage("no_code", null,"기본 메시지", null))
            .isEqualTo("기본 메시지");
}
```

## 테스트 **4. argu** 설정

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

hello.name=안녕 {0} ⇒ Object 배열로 치환됨.

## 테스트 **5. locale** 설정

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

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

---

## 타임리프 메시지 적용

#{} 으로 사용한다

#{hello} ⇒ 안녕 으로 치환해줌

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

page.addItem=상품 등록 일때 페이지 소스에 적용되어 나타난다.

```
    <h2>상품 등록</h2>
```

파라미터로 설정시

```
hello.name=안녕 {0}
<p th:text="#{hello.name(${item.itemName})}"></p>
```

파라미터는 이렇게 사용가능하다.

```
<tr th:each="item : ${items}">
    <td th:text="#{hello.name(${item.itemName},${item.id})}"></td>
</tr>
```

배열 형식으로 지정해두고 사용

```
hello.name=안녕 {1}  ⇒ 일 경우 안녕 item.id (값)
```

```
hello.name=안녕 {0} ⇒ 일 경우 안녕 item.itemName (값)
```

설정한 인덱스의 값이 나오게 된다.

```
<td th:text="#{hello.name(${item.itemName},${item.id},'hello word')}"></td>
```

```
hello.name=안녕 {2} ⇒ 일 경우 안녕 hello word 출력
```

## 국제화 적용시키기

```
messages_en.properties
```

파일 설정해서 만든 후에.

타임리프의 메시지 선택만 할 수 있게 하면된다.

-웹 설정에서 언어만 영어로 바꾸면 언어에 맞는 메시지 파일이 설정된다.

[Accept-Language 가 변경된다 ]

**Accept-Language?**

클라이언트가 서버에 기대하는 언어 정보를 담아서 요청하는 HTTP 요청헤더.

**Locale** 정보를 알아야 사용하는데 스프링은 기본적으로 **Accept-Language** 헤더의 값을 사용하는데 변경해서도 사용가능 하다**.**

→ 쿠키나 사용자가 선택하는 방법등**.**

LocaleResolver 인터페이스로 적용한다.

스프링은 기본적으로 AcceptHeaderLocaleResolver를 사용하는것.

즉, 구현체를 변경해서 지정된 설정을 불러오는 것을 만들 수 있다.