Spring MVC Internationalization (i18n) and Localization (L10n) Example

Internationalization (i18n) or localization (L10n) is used to change language of a web application for better interaction. In this article, I will try to configure Spring MVC web application to use the Internationalization concept.

1. mvc-dispatcher-servlet.xml file

```
1  <mvc:interceptors>
2      <!-- Changes the locale when a 'lang' request parameter is sent; e.g. /?lang=tr -->
3      <bean id="localeChangeInterceptor"
4  class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
5          <property name="paramName" value="lang"/>
6      </bean>
7  </mvc:interceptors>
8
9  <bean id="localeResolver"
10      class="org.springframework.web.servlet.i18n.CookieLocaleResolver">
11     <property name="defaultLocale" value="tr"/>
12     <property name="cookieName" value="localizationCookie"/>
13     <property name="cookieMaxAge" value="3600"/>
14 </bean>
15
16 <bean id="messageSource"
17      class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
18     <property name="basename" value="WEB-INF/messages"/>
19     <property name="defaultEncoding" value="UTF-8"/>
20     <property name="cacheSeconds" value="3600"/>
</bean>
```

CookieLocaleResolver is **LocaleResolver** implementation that uses a cookie sent back to the user in case of a custom setting, with a fallback to the specified default locale or the request's accept-header locale. This is particularly useful for stateless applications without user sessions. The cookie may optionally contain an associated time zone value as well; alternatively, you may specify a default time zone

In line 17, we added properties files into **WEB-INF** directory because Spring docs says: In contrast to **ResourceBundleMessageSource**, this class supports reloading of properties files through the "cacheSeconds" setting, and also through programmatically clearing the properties cache. Since application servers typically cache all files loaded from the classpath, it is necessary to store resources somewhere else (for example, in the "WEB-INF" directory of a web app). Otherwise changes of files in the classpath will not be reflected in the application.

2. Properties Files

We should define **.properties** file for each language which is named with same prefix, such as **messages** and ends with specific language abbrevition **en, tr or etc.**

messages_tr.properties file

```
1  com.codesenior.languageName=English
2  com.codesenior.languageUrl=/?lang=EN
3  com.codesenior.search=Arama
```

messages_en.properties file

```
com.codesenior.languageName=Turkish
com.codesenior.languageUrl=/?lang=TR
com.codesenior.search=Search
```

3. JSP File

```
<div id="PanelDilSecimi">
    <a class="englishBtn" href="<spring:message code="com.codesenior.languageUrl"/>">
        <spring:message code="com.codesenior.languageName"/>
    </a>
</div>
```

spring:message is used display the message from the corresponds properties file by checking the current user's locale.

4. Controller Class

```
@RequestMapping(value = "/", method = RequestMethod.GET)
@ResponseBody
public String index(Locale locale) {
     return locale.getLanguage();
}
```

After this configuration you can change localization by browsing http://www.example.com/?lang=EN for English and http://www.example.com/?lang=TR for Turkish locale