http://www.baeldung.com/spring-boot-internationalization

Guide to Internationalization in Spring Boot

Last modified: March 18, 2017

by [baeldung](http://www.baeldung.com/author/baeldung/)

* [**Spring**](http://www.baeldung.com/category/spring/)**+**

**The new Certification Class of *Learn Spring Security* is out:**

[**>> CHECK OUT THE COURSE**](http://www.baeldung.com/learn-spring-security-course#certification-class)

**1. Overview**

In this quick tutorial, we’re going to take a look at how we can **add internationalization to a Spring Boot application**.

**2. Maven Dependencies**

For development, we need the following dependency:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-thymeleaf</artifactId>      <version>1.5.2.RELEASE</version>  </dependency> |

The latest version of [spring-boot-starter-thymeleaf](https://search.maven.org/#search%7Cga%7C1%7Ca%3A%22spring-boot-starter-thymeleaf%22)can be downloaded from Maven Central.

**3. *LocaleResolver***

In order for our application to be able to determine which locale is currently being used, we need to add a *LocaleResolver* bean:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Bean  public LocaleResolver localeResolver() {      SessionLocaleResolver slr = new SessionLocaleResolver();      slr.setDefaultLocale(Locale.US);      return slr;  } |

The *LocaleResolver* interface has implementations that determine the current locale based on the session, cookies, the *Accept-Language* header, or a fixed value.

In our example, we have used the session based resolver *SessionLocaleResolver* and set a default locale with value *US*.

**4. *LocaleChangeInterceptor***

Next, we need to add an interceptor bean that will switch to a new locale based on the value of the *lang*parameter appended to a request:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Bean  public LocaleChangeInterceptor localeChangeInterceptor() {      LocaleChangeInterceptor lci = new LocaleChangeInterceptor();      lci.setParamName("lang");      return lci;  } |

**In order to take effect, this bean needs to be added to the application’s interceptor registry.**

To achieve this, our *@Configuration* class has to extend the *WebMvcConfigurerAdapter* class and override the *addInterceptors()* method:

|  |  |
| --- | --- |
| 1  2  3  4 | @Override  public void addInterceptors(InterceptorRegistry registry) {      registry.addInterceptor(localeChangeInterceptor());  } |

**5. Defining the Message Sources**

By default, a Spring Boot application will look for message files containing internationalization keys and values in the *src/main/resources* folder.

The file for the default locale will have the name *messages.properties*, and files for each locale will be named *messages\_XX.properties*, where *XX* is the locale code.

The keys for the values that will be localized have to be the same in every file, with values appropriate to the language they correspond to.

**If a key does not exist in a certain requested locale, then the application will fall back to the default locale value.**

Let’s define a default message file for the English language called *messages.properties*:

|  |  |
| --- | --- |
| 1  2  3  4 | greeting=Hello! Welcome to our website!  lang.change=Change the language  lang.eng=English  lang.fr=French |

Next, let’s create a file called *messages\_fr.properties* for the French language with the same keys:

|  |  |
| --- | --- |
| 1  2  3  4 | greeting=Bonjour! Bienvenue sur notre site!  lang.change=Changez la langue  lang.eng=Anglais  lang.fr=Francais |

**6. Controller and HTML Page**

Let’s create a controller mapping that will return a simple HTML page called*international.html* that we want to see in two different languages:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @Controller  public class PageController {        @GetMapping("/international")      public String getInternationalPage() {          return "international";      }  } |

Since we are using thymeleaf to display the HTML page, the locale-specific values will be accessed using the keys with the syntax *#{key}*:

|  |  |
| --- | --- |
| 1 | <h1 th:text="#{greeting}"></h1> |

If using JSP files, the syntax is:

|  |  |
| --- | --- |
| 1 | <h1><spring:message code="greeting" text="default"/></h1> |

If we want to access the page with the two different locales we have to add the parameter *lang* to the URL in the form:*/international?lang=fr*

If no *lang* parameter is present on the URL, the application will use the default locale, in our case *US*locale.

Let’s add a drop-down to our HTML page with the two locales whose names are also localized in our properties files:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <span th:text="#{lang.change}"></span>:  <select id="locales">      <option value=""></option>      <option value="en" th:text="#{lang.eng}"></option>      <option value="fr" th:text="#{lang.fr}"></option>  </select> |

Then we can add a jQuery script that will call the*/international* URL with the respective *lang* parameter depending on which drop-down option is selected:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <script src="<https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js>">  </script>  <script type="text/javascript">  $(document).ready(function() {      $("#locales").change(function () {          var selectedOption = $('#locales').val();          if (selectedOption != ''){              window.location.replace('international?lang=' + selectedOption);          }      });  });  </script> |

**7. Running the Application**

In order to initialize our application, we have to add the main class annotated with *@SpringBootApplication*:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @SpringBootApplication  public class InternationalizationApp {        public static void main(String[] args) {          SpringApplication.run(InternationalizationApp.class, args);      }  } |

Depending on the selected locale, we will view the page in either English or French when running the application.

Let’s see the English version:

[](http://www.baeldung.com/wp-content/uploads/2017/03/piceng.png)

And now let’s see the French version:

[](http://www.baeldung.com/wp-content/uploads/2017/03/picfr.png)

**8. Conclusion**

In this tutorial, we have shown how we can use the support for internationalization in a Spring Boot application.

The full source code for the example can be found [over on GitHub](https://github.com/eugenp/tutorials/tree/master/spring-boot).