Spring系列-7 国际化

原创 Ewen Seong 已于 2024-06-02 14:42:06 修改 阅读量3.4k 收藏 2 点赞数 2

分类专栏： Spring系列 文章标签： spring java

背景：

为了提高软件的通用性(应对不同的语言环境)、扩大软件的业务受众范围，软件被要求具备处理 国际化 的能力。Java和Spring为此分别提供了不同层
案：Java在java.util包中提供了支持国际化能力的API与工具类，Spring基于此进行封装并提供了容器级别的接口。
本文作为**Spring系列**文章的第四篇，内容包含JDK、Spring、Spring Boot国际化相关的API的使用和背后原理；其中，基于Spring框架介绍 国际化的
本文的重点内容，该部分会伴随着Spring源码进行。

1.国际化

Java定义了Locale类用于表示国际化类型，包含国家和地区两种信息：

```
static public final Locale SIMPLIFIED_CHINESE = createConstant("zh", "CN");
static public final Locale CHINA = SIMPLIFIED_CHINESE;
static public final Locale US = createConstant("en", "US");
...
```

其中，Locale.CHINA表示简体中文(对应zh_CN)；Locale.US表示美式英语(对应en_US).
根据语言和地区得到国际化对象后，可将用户信息翻译为该对象代表的国际化，案例如下：

(1) 国际化货币

```
@Test
public void testCurrencyInstance() {
    LOGGER.info("[CN] result is {}", NumberFormat.getCurrencyInstance(Locale.CHINA).format(100.00));
    LOGGER.info("[US] result is {}", NumberFormat.getCurrencyInstance(Locale.US).format(100.00));
    LOGGER.info("[default] result is {}", NumberFormat.getCurrencyInstance().format(100.00));
}
```

得到如下结果：

```
[main] INFO com.seong.test.JavaI18nTest - [CN] result is ￥100.00
[main] INFO com.seong.test.JavaI18nTest - [US] result is $100.00
[main] INFO com.seong.test.JavaI18nTest - [default] result is ￥100
```

(2) 国际化日期

```
@Test
public void testDateInstance() {
    Date date = new Date();
    LOGGER.info("[CN] result is {}", DateFormat.getDateInstance(DateFormat.DEFAULT, Locale.CHINA).format(date));
    LOGGER.info("[US] result is {}", DateFormat.getDateInstance(DateFormat.DEFAULT, Locale.US).format(date));
    LOGGER.info("[default] result is {}", DateFormat.getDateInstance(DateFormat.DEFAULT).format(date));
}
```

得到如下结果：

```
[main] INFO com.seong.test.JavaI18nTest - [CN] result is 2022-12-18
[main] INFO com.seong.test.JavaI18nTest - [US] result is Dec 18, 2022
[main] INFO com.seong.test.JavaI18nTest - [default] result is 2022-12
```

(3) 国际化时间

```java
@Test
public void testTimeInstance() {
    Date date = new Date();
    LOGGER.info("[CN] result is {}", DateFormat.getTimeInstance(DateFormat.DEFAULT, Locale.CHINA).format(date));
    LOGGER.info("[US] result is {}", DateFormat.getTimeInstance(DateFormat.DEFAULT, Locale.US).format(date));
    LOGGER.info("[default] result is {}", DateFormat.getTimeInstance(DateFormat.DEFAULT).format(date));
}
```

得到如下结果：

```
[main] INFO com.seong.test.JavaI18nTest - [CN] result is 15:28:37
[main] INFO com.seong.test.JavaI18nTest - [US] result is 3:28:37 PM
[main] INFO com.seong.test.JavaI18nTest - [default] result is 15:28
```

分析：

当不指定语言类型时，系统会使用默认的国际化类型，因此3个测试用例的[default]内容同[CN]。默认的国际化类型信息来自平台本身：

```java
@Test
public void testDefaultLocal() {
    String language = AccessController.doPrivileged(new GetPropertyAction("user.language", "en"));
    LOGGER.info("Default language is {}.", language);

    String country = AccessController.doPrivileged(new GetPropertyAction("user.country", ""));
    LOGGER.info("Default country is {}.", country);

    String region = AccessController.doPrivileged(new GetPropertyAction("user.region"));
    LOGGER.info("Default region is {}.", region);
}
```

得到如下结果：

```
[main] INFO com.seong.test.JavaI18nTest - Default language is
[main] INFO com.seong.test.JavaI18nTest - Default country is
[main] INFO com.seong.test.JavaI18nTest - Default region is nu
```

> 国际化涉及到时间、货币、数字和字符串等，本文后续内容围绕后者进行。

2.使用方式

2.1 JDK中的国际化

在资源路径下准备国际化文件

(1) 准备国际化文件：

国际化资源文件的命名需要遵循命名规范： `资源名_语言_国家/地区.properties` ，如：**messages_en_US.properties**表示美式英语，**messages_zh_CN**表示简体中文。

另外，语言和国家/地区可以缺省—□—用于表示默认的资源文件，如**message.properties**；当某个本地化类型在系统中找不到对应的资源文件时，就使项。

```
#resources/i18n文件夹下
#messages.properties文件
10000=您好，北京
10001=您好，南京
10002=您好，上海

#messages_en_US.properties文件
10000=Hello, Beijing
10001=Hello, Nanjing

#messages_zh_CN.properties文件

```

```
10000=你好，北京
10001=你好，南京
```

> 资源文件对文件内容有严格的要求：只能包含ASCII字符。所以必须将非ASCII字符的内容转换为Unicode代码的表示方式。

(2) 使用如下命令进行转换：

```
// native2ascii是JDK提供的命令
 native2ascii -encoding utf-8 ./i18n/messages.properties ./i18n/messages.properties
 native2ascii -encoding utf-8 ./i18n/messages_zh_CN.properties ./i18n/messages_zh_CN.properties
```

得到结果如下：

```
#resources/i18n文件夹下
#messages.properties文件
10000=\u60A8\u597D\uFF0C\u5317\u4EAC
10001=\u60A8\u597D\uFF0C\u5357\u4EAC
10002=\u60A8\u597D\uFF0C\u4E0A\u6D77

#messages_en_US.properties文件
10000=Hello, Beijing
10001=Hello, Nanjing

#messages_zh_CN.properties文件
10000=\u4F60\u597D\uFF0C\u5317\u4EAC
10001=\u4F60\u597D\uFF0C\u5357\u4EAC
```

(3) IDEA配置Transparent native-to-sacii conversion：
阅读与修改Unicode代码存在一定难度，通过对IDEA进行配置，开发人员可以友好地操作资源文件：

```
#resources/i18n文件夹下
#messages.properties文件
10000=您好，北京
10001=您好，南京
10002=您好，上海

#messages_en_US.properties文件
10000=Hello, Beijing
10001=Hello, Nanjing

#messages_zh_CN.properties文件
10000=你好，北京
10001=你好，南京
```

配置方式： `Preferences/Settings -> Editor -> File Encodings -> Transparent native-to-sacii conversion`

获取国际化资源

```
@Slf4j
@Slf4j
public class JavaI18nTest {
    @Test
    public void testJavaI18n() {
        ResourceBundle cnBundle = ResourceBundle.getBundle("i18n/messages", Locale.CHINA);
        LOGGER.info("[CHINA] 10000.msg is {}.", cnBundle.getString("10000"));
        LOGGER.info("[CHINA] 10001.msg is {}.", cnBundle.getString("10001"));
        LOGGER.info("[CHINA] 10002.msg is {}.", cnBundle.getString("10002"));
        ResourceBundle usBundle = ResourceBundle.getBundle("i18n/messages", Locale.US);
        LOGGER.info("[US] 10000.msg is {}.", usBundle.getString("10000"));
        LOGGER.info("[US] 10001.msg is {}.", usBundle.getString("10001"));
        LOGGER.info("[US] 10002.msg is {}.", usBundle.getString("10002"));
    }
}
```

运行后得到如下结果：

```
13:32:13.198 [main] INFO com.seong.test.JavaI18nTest - [CHINA] 10000.msg is 你好，北京.
13:32:13.201 [main] INFO com.seong.test.JavaI18nTest - [CHINA] 10001.msg is 你好，南京.
13:32:13.202 [main] INFO com.seong.test.JavaI18nTest - [CHINA] 10002.msg is 您好，上海.
13:32:13.204 [main] INFO com.seong.test.JavaI18nTest - [US] 10000.msg is Hello, Beiji
13:32:13.204 [main] INFO com.seong.test.JavaI18nTest - [US] 10001.msg is Hello, Nanji
13:32:13.204 [main] INFO com.seong.test.JavaI18nTest - [US] 10002.msg is 您好，上海.
```

从结果可知：红框标记的输出项的国际化信息来源于默认的资源文件messages.properties.

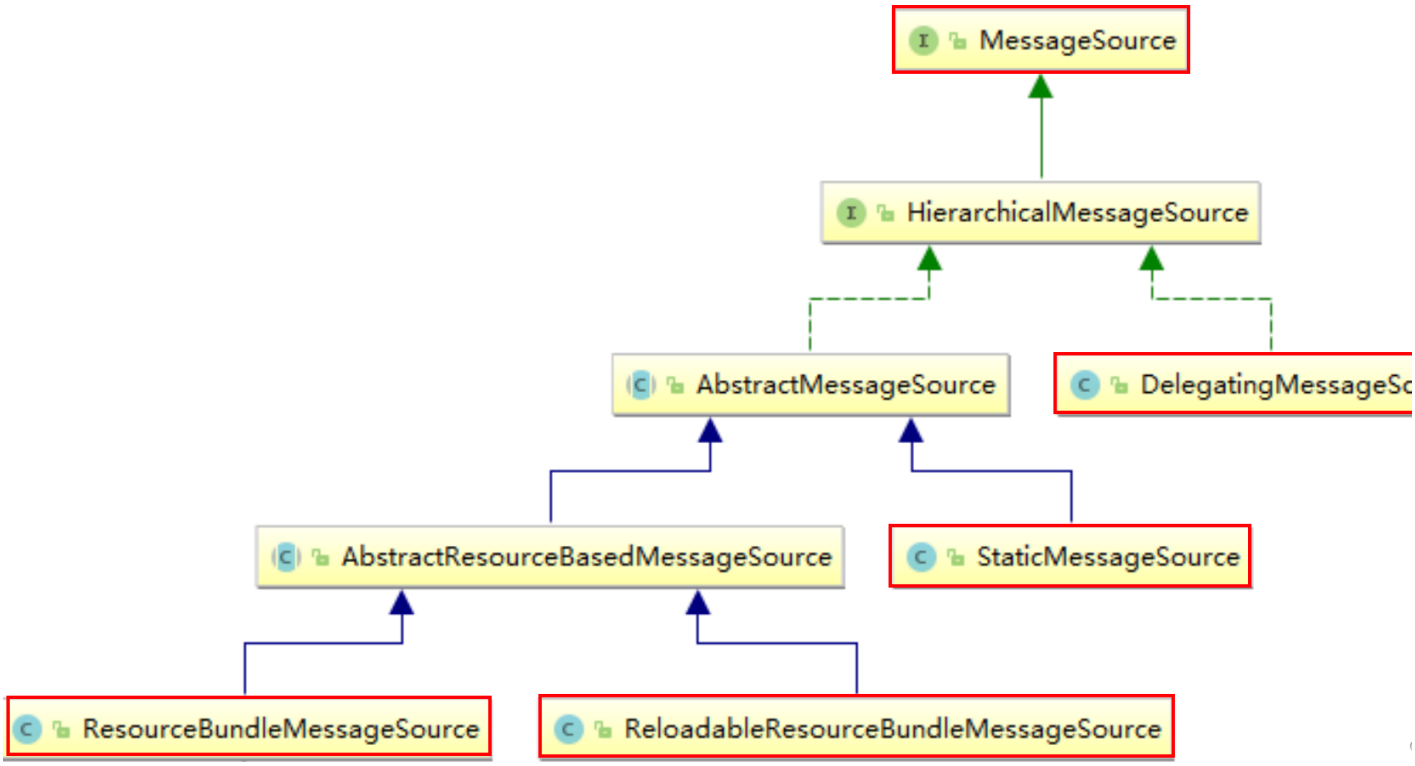
2.1.Spring项目

Spring对JDK的国际化API进行了封装和增强，提高了容器级别的接口**MessageSource**，该接口位于context模块，共提供三种获取国际化信息的接口，

```
1  package org.springframework.context;
2
3  import java.util.Locale;
4  import org.springframework.lang.Nullable;
5
6  public interface MessageSource {
7      // defaultMessage为默认值，获取失败时-返回该对象
8      @Nullable
9      String getMessage(String code, @Nullable Object[] args, @Nullable String defaultMessage, Locale locale);
10
11     // 常用
```

其中第二种较为常用，code表示待解析的字符串信息，args为占位符参数(来自JDK的API)，Locale为国际化对象，返回的字符串为本地化后的结果。

MessageSource接口的继承关系如下：



如上图所示，Spring共提供了四种实现类：DelegatingMessageSource, StaticMessageSource, ResourceBundleMessageSource, ReloadableResourceBundleMessageSource.

常见为后两种: ResourceBundleMessageSource依赖于JDK提供的API，即依赖 `ResourceBundle.getBundle(basename, locale, classLoader, control)`
化；ReloadableResourceBundleMessageSource提供了定时更新国际化的能力而无需重启机器; 本文以ResourceBundleMessageSource为例介绍如何
中使用国际化(ReloadableResourceBundleMessageSource使用与之类似)。

以ResourceBundleMessageSource为例介绍：

配置国际化资源：
使用与2.1中相同的国际化资源.

配置Bean：

```
<bean id="myResource" class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="i18n/messages"/>
    <property name="defaultEncoding" value="UTF-8"/>
</bean>
```

当使用ReloadableResourceBundleMessageSource时，需要同时配置更新时间：

```
<bean id="myResource" class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
    <property name="basename" value="i18n/messages"/>
    <property name="defaultEncoding" value="UTF-8"/>
    <property name="cacheSeconds" value="5"/>
</bean>
```

> 注意：这里写法有问题(举例可忽略)，实际beanName需要设置为messageSource，原因在章节3原理部分介绍。

读取国际化资源：

```
@Slf4j
public class SpringDemoApplication {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("spring-context.xml");
        MessageSource messageSource = (MessageSource) context.getBean("myResource");

        LOGGER.info("[zh-CN] 10000.msg is {}.", messageSource.getMessage("10000", null, Locale.CHINA));
        LOGGER.info("[en-US] 10000.msg is {}.", messageSource.getMessage("10000", null, Locale.US));
    }
}
```

得到结果：

```
G org.springframework.beans.factory.support.DefaultListableBeanFactory - Creat
 com.seong.SpringDemoApplication - [zh-CN] 10000.msg is 你好，北京.
 com.seong.SpringDemoApplication - [en-US] 10000.msg is Hello, Beijing.
```

另外，对于Spring提供的ResourceBundleMessageSource类，项目可以选择将其作为Bean对象(如上)，也可以不作为Bean对象，如下所示：

```
@Slf4j
public class I18nTest {
    @Test
    public void testI18n() {
        MessageSource messageSource = buildMessageSource();
        LOGGER.info("[zh-CN] 10000.msg is {}.", messageSource.getMessage("10000", null, Locale.CHINA));
        LOGGER.info("[en-US] 10000.msg is {}.", messageSource.getMessage("10000", null, Locale.US));
    }

    private MessageSource buildMessageSource() {
        ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
```

得到运行结果：

```
[main] INFO com.seong.test.I18nTest - [zh-CN] 10000.msg is 你好，北京.
[main] INFO com.seong.test.I18nTest - [en-US] 10000.msg is Hello, Bei
```

2.2 SpringBoot项目

配置国际化资源：

使用与2.1中相同的国际化资源.

在application.yml中进行参数配置：

```yaml
spring:
  messages:
    basename: i18n/messages
    encoding: UTF-8
    fallback-to-system-locale: false
```

使用国际化资源：

```java
@Slf4j
@RunWith(SpringRunner.class)
@SpringBootTest
public class I18nTest {
    @Autowired
    private MessageSource messageSource;

    @Test
    public void testI18n() {
        LOGGER.info("[zh-CN] 10000.msg is {}.", messageSource.getMessage("10000",null, Locale.CHINA));
        LOGGER.info("[en-US] 10000.msg is {}.", messageSource.getMessage("10000",null, Locale.US));
    }
}
```

得到运行结果如下：

```
t.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
I18nTest                  : Started I18nTest in 3.213 seconds (JVM running for 4.371
I18nTest                  : [zh-CN] 10000.msg is 你好，北京.
I18nTest                  : [en-US] 10000.msg is Hello, Beijing.
```

3.原理

3.1 Spring项目国际化原理

在启动Spring容器时—执行refresh()的步骤中存在 `initMessageSource()` 方法：

```java
@Override
public void refresh() throws BeansException, IllegalStateException {
    //...
    registerBeanPostProcessors(beanFactory);

    // Initialize message source for this context.
    initMessageSource();

    //...

    // Instantiate all remaining (non-lazy-init) singletons.
    finishBeanFactoryInitialization(beanFactory);
    // ...
}
```

进入initMessageSource()内部：

```
// 简化后的代码
protected void initMessageSource() {
    ConfigurableListableBeanFactory beanFactory = getBeanFactory();
    if (beanFactory.containsLocalBean("messageSource")) {
        this.messageSource = beanFactory.getBean("messageSource", MessageSource.class);
        // Make MessageSource aware of parent MessageSource.
        if (this.parent != null && this.messageSource instanceof HierarchicalMessageSource) {
            HierarchicalMessageSource hms = (HierarchicalMessageSource) this.messageSource;
            if (hms.getParentMessageSource() == null) {
                // Only set parent context as parent MessageSource if no parent MessageSource
                // registered already
```

代码只有两个主分支，逻辑比较清晰：
如果BeanFactory中存在beanName为"messageSource"的BeanDefinition： 实例化该BeanDefinition并将Bean对象设置给AbstractApplicationConte
messageSource属性；同时如果该Spring容器存在父容器且该国际化对象为HierarchicalMessageSource类型(具备父子结构), 则一并设置国际化对象的
不存在"messageSource"： 实例化一个DelegatingMessageSource类型的对象，并注册到IOC容器中，同时赋值给AbstractApplicationContext的mess
性。如注释 `// Use empty MessageSource to be able to accept getMessage calls` ：DelegatingMessageSource无实际功能，是为了让调用不会抛空指
示：

```
// DelegatingMessageSource:
@Override
public String getMessage(String code, @Nullable Object[] args, Locale locale) throws NoSuchMessageException {
    if (this.parentMessageSource != null) {
        return this.parentMessageSource.getMessage(code, args, locale);
    }
    else {
        throw new NoSuchMessageException(code, locale);
    }
}
```

个人见解：
Spring容器在AbstractApplicationContext中引入了messageSource属性，并在启动容器的过程中将"messageSource"对应的Bean对象设置给messageS
这就明示着要将国际化对象作为一个容器级别的全局组件对外提供，因此Spring项目中如果设置了国际化对象，必须将beanName设置为"**messageSou**

在2.1章节中介绍了MessageSource的类继承关系图，仅展示了真正实现国际化功能的子类，为包含代理类如ApplicationContext系列。如下所示是AbstractApplicationContext中对MessageSource中接口的实现：

```
@Override
public String getMessage(String code, @Nullable Object[] args, @Nullable String defaultMessage, Locale locale) {
    return getMessageSource().getMessage(code, args, defaultMessage, locale);
}

@Override
public String getMessage(String code, @Nullable Object[] args, Locale locale) throws NoSuchMessageException {
    return getMessageSource().getMessage(code, args, locale);
}

@Override
```

即完全将国际化功能委托给了messageSource属性。
因此，在Spring项目中可以直接使用Spring容器对象或者messageSource对应的Bean对象来获取国际化信息。
在2.1中提到了ResourceBundleMessageSource依赖于JDK的API，这里稍作探究：

```
@Slf4j
@RunWith(SpringRunner.class)
@SpringBootTest
public class I18nTest {

```

```
    @Autowired
    private MessageSource messageSource;

    @Test
    public void testI18n() {
        LOGGER.info("[zh-CN] 10000.msg is {}.", messageSource.getMessage("10000",null, Locale.CHINA));
        LOGGER.info("[en-US] 10000.msg is {}.", messageSource.getMessage("10000",null, Locale.US));
    }
}
```

Step1: 跟进getMessage方法：

```
@Override
public final String getMessage(String code, @Nullable Object[] args, Locale locale) throws NoSuchMessageException {
    // 主干流程
    String msg = getMessageInternal(code, args, locale);
    if (msg != null) {
        return msg;
    }
    // 从支1：返回code本身作为国际化结果
    String fallback = getDefaultMessage(code);
    if (fallback != null) {
        return fallback;
    }
    // 从支2： 抛出异常
    throw new NoSuchMessageException(code, locale);
}
```

先调用 `getMessageInternal(code, args, locale)` 获取国际化信息，不为空则直接返回；
否则进入**从支1**，调用 `getDefaultMessage(code)` 逻辑，如果配置项useCodeAsDefaultMessage属性为true(默认为false)则将code本身作为国际化信息返
`getDefaultMessage(code)` 得到null；进入从支2—抛出异常。

Step2: 跟进getMessageInternal(code, args, locale)方法：

```
@Nullable
protected String getMessageInternal(@Nullable String code, @Nullable Object[] args, @Nullable Locale locale) {
    if (!isAlwaysUseMessageFormat() && ObjectUtils.isEmpty(args)) {
        String message = resolveCodeWithoutArguments(code, locale);
        if (message != null) {
            return message;
        }
    } else {
        argsToUse = resolveArguments(args, locale);
        MessageFormat messageFormat = resolveCode(code, locale);
        if (messageFormat != null) {
```

整体流程比较清晰：
(1) 优先从国际化资源中获取，获取失败时再从**配置项commonMessages**对象中获取—😭—获取失败再从其父对象中获取；
(2) 从国际化资源中获取时，如果需要处理占位符则先处理占位符再处理国际化：

```
argsToUse = resolveArguments(args, locale);
MessageFormat messageFormat = resolveCode(code, locale);
if (messageFormat != null) {
    synchronized (messageFormat) {
        return messageFormat.format(argsToUse);
    }
}
```

进入resolveCode方法内部：

```
protected MessageFormat resolveCode(String code, Locale locale) {
    Set<String> basenames = getBasenameSet();
    for (String basename : basenames) {
    // ⚠获取ResourceBundle对象
        ResourceBundle bundle = getResourceBundle(basename, locale);
```

```
4
5
6           if (bundle != null) {
7               MessageFormat messageFormat = getMessageFormat(bundle, code, locale);
8               if (messageFormat != null) {
9                   return messageFormat;
10              }
11          }
12      }
13      return null;
14  }
```

(3) 不需要处理占位符：直接调用 `resolveCodeWithoutArguments(code, locale)` 获取；
进入resolveCodeWithoutArguments内部：

```
1   protected String resolveCodeWithoutArguments(String code, Locale locale) {
2       Set<String> basenames = getBasenameSet();
3       for (String basename : basenames) {
4           // ⚠获取ResourceBundle对象
5           ResourceBundle bundle = getResourceBundle(basename, locale);
6           if (bundle != null) {
7               String result = getStringOrNull(bundle, code);
8               if (result != null) {
9                   return result;
10              }
11          }
12      }
13      return null;
14  }
```

Step3: 跟进getResourceBundle(basename, locale);
step2中的(2)和(3)最终都会调用 `getResourceBundle(basename, locale)` 方法获取ResourceBundle对象，然后从ResourceBundle对象中获取国际化信息(ResourceBundle对象的属性中解析，逻辑较为简单)；
剩余的调用链依次为： `ResourceBundle bundle = doGetBundle(basename, locale);` -> `ResourceBundle.getBundle(basename, locale, classLoader, co`
者为JDK提供的API。
JDK中的国际化API实现方式不是本文关心的内容，因此不再继续往下进行；感兴趣的读者可以继续往下扒一下JDK是如何加载资源文件并构造Resourc
的流程(有问题可在留言区讨论)。

3.2 SpringBoot项目国际化原理

在Spring项目中手动配置了ResourceBundleMessageSource类型的Bean对象，而SpringBoot可以直接从IOC中获取而不需要配置；这是因为Spring Bo
配机制向IOC中默认注入了一个ResourceBundleMessageSource类型的Bean对象。

> 自动装配的逻辑不在本文介绍，将在介绍完Spring Bean/AOP 后补上，感兴趣读者可订阅Spring系列专题。
> 本章节认为读者对SpringBoot的自动装配已有基本了解。

spring.factories文件中引入MessageSourceAutoConfiguration类：

在[spring-boot-autoconfigure]模块的**spring.factories**文件的
`org.springframework.boot.autoconfigure.EnableAutoConfiguration` 属性中存在

`org.springframework.boot.autoconfigure.context.MessageSourceAutoConfiguration` 类型：

```
# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfigurati
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\
org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\
org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration,\
org.springframework.boot.autoconfigure.cassandra.CassandraAutoConfiguration,\
org.springframework.boot.autoconfigure.context.ConfigurationPropertiesAutoConfigurati
org.springframework.boot.autoconfigure.context.LifecycleAutoConfiguration,\
org.springframework.boot.autoconfigure.context.MessageSourceAutoConfiguration,\
```

MessageSourceAutoConfiguration注入条件：

MessageSourceAutoConfiguration类上存在@ConditionalOnMissingBean注解，如下所示：

```
@Configuration(proxyBeanMethods = false)
@ConditionalOnMissingBean(name = "messageSource", search = SearchStrategy.CURRENT)
public class MessageSourceAutoConfiguration {
//...
}
```

表示：当有beanName为messageSource时，该类的装配过程不会进行；因此用户可自定义MessageSource，如ReloadableResourceBundleMessageS

示例1: messageSource

▲ 点赞 3 ▼

```
import org.springframework.context.support.ReloadableResourceBundleMessageSourc
入依赖的package包/类
@Bean(name = "messageSource")
public ReloadableResourceBundleMessageSource messageSource() {
    ReloadableResourceBundleMessageSource messageBundle = new ReloadableResourc
MessageSource();
    messageBundle.setBasename("classpath:messages/messages");
    messageBundle.setDefaultEncoding("UTF-8");
    return messageBundle;
}
```

开发者ID:rovaniemi，项目名称:remember-me-back，代码行数:8，代码来源:CorsConfiguration.

示例11: messageSource

▲ 点赞 2 ▼

```
import org.springframework.context.support.ReloadableResourceBundleMessageSourc
入依赖的package包/类
@Bean
ReloadableResourceBundleMessageSource messageSource() {
    ReloadableResourceBundleMessageSource bundleMessageSource = new ReloadableR
BundleMessageSource();
    bundleMessageSource.setBasename("classpath:i18n/messages");
    bundleMessageSource.setCacheSeconds(1800);
    bundleMessageSource.setDefaultEncoding("UTF-8");
    return bundleMessageSource;
}
```

开发者ID:WickedWitchWarsaw，项目名称:ThymeleafSpringDemo，代码行数:9，代码来源:I18NConfig.java

CSD

配置属性MessageSourceProperties:

在MessageSourceAutoConfiguration中定义了MessageSourceProperties的注入逻辑:

```
@Bean
@ConfigurationProperties(prefix = "spring.messages")
public MessageSourceProperties messageSourceProperties() {
    return new MessageSourceProperties();
}
```

`@ConfigurationProperties(prefix = "spring.messages")` 表示读取配置文件中**spring.messages**表示的信息并注入到MessageSourceProperties对象中

对应2.2章节中的配置:

在application.yml中进行参数配置:

Ewen Seong 已关注

```yaml
spring:
  messages:
    basename: i18n/messages
    encoding: UTF-8
    fallback-to-system-locale: false
```

上述属性对应MessageSourceProperties配置类中的属性：

```java
public class MessageSourceProperties {
    // 默认为resources资源路径下的"messages"
    private String basename = "messages";

    // 编码格式，一般设置为utf-8(默认是 UTF-8)
    private Charset encoding;

    // 指定资源、默认资源文件找不到时是否从当前系统语言对应配置文件中查找
    // 默认为true: 进行查找；false时-抛出异常
    private boolean fallbackToSystemLocale;
```

MessageSourceProperties本质上是一个临时的内存数据—🐭—为了读取作准备(实现配置文件 -> 内存)，实例化ResourceBundleMessageSource对象时置信息.

注入MessageSource：

在MessageSourceAutoConfiguration中定义了MessageSource的注入逻辑：

```java
@Bean
public MessageSource messageSource(MessageSourceProperties properties) {
    ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
    if (StringUtils.hasText(properties.getBasename())) {
        messageSource.setBasenames(StringUtils
                .commaDelimitedListToStringArray(StringUtils.trimAllWhitespace(properties.getBasename())));
    }
    if (properties.getEncoding() != null) {
        messageSource.setDefaultEncoding(properties.getEncoding().name());
    }
    messageSource.setFallbackToSystemLocale(properties.isFallbackToSystemLocale());
```

逻辑较为简单，直接从MessageSourceProperties中读取国际化配置信息构造ResourceBundleMessageSource对象，并注入到IOC容器中。

假设我们正在开发一个支持多国语言的Web应用程序，要求系统能够根据客户端的系统的语言类型返回对应的界面：英文的操作系统返回英文界面，而中文的操作系统则返回

Java国际化概念和使用介绍_java国际化是什么意思
Java 国际化的思想是将程序中的信息放在资源文件中,程序根据支持的国家及语言环境读取相应的资源文件。资源文件是 key-value 对,每个资源文件中的 key 是不变的,但 va

解释Java中的国际化和本地化是什么,你在项目中如何应用它?
在一个Java Web应用中,我们可以通过以下步骤实现国际化和本地化: 资源文件管理:将应用中的所有文本信息(如提示信息、按钮标签、菜单项等)提取到属性文件中,并为不同

如何在Java中实现国际化与本地化 **最新发布** 微赚淘客开发
国际化是指在开发过程中设计和编写代码，使其能够支持多种语言和地区，而无需对代码进行重大修改。具体包括文本资源的外置、支持多种字符编码等。本地化是指在应用

和小伙伴们仔细梳理一下 Spring 国际化吧！从用法到源码！ 江南一点雨
国际化（Internationalization，简称 I18N）是指在 Java 应用程序中实现国际化的技术和方法。Java 提供了一套强大的国际化支持，使开发人员能够编写适应不同语言、地区

java入坑之国际化编程_java代码国际化
java入坑之国际化编程 一、字符编码 1.1概述 字符编码 --字符:0,a,我,①,,… --计算机只用0和1,1bit(0或者1) --ASCIL码(American Standard Code for Information Interchange)

Spring——i18n Java国际化和Spring国际化的区别_java 国际化i18n和...
资源加载方式不同:Java国际化通常使用 ResourceBundle 类来加载资源文件,而Spring国际化则使用 ResourceBundleMessageSource 类来加载资源文件。 框架集成:Spring

Java中的国际化与本地化处理 微赚淘客开发
国际化与本地化处理是Java开发中的重要环节，它不仅提升了软件的用户体验，也扩展了软件的市场覆盖范围。通过合理使用Java提供的国际化和本地化工具类，我们可以

Spring 国际化 i18n 和数据校验 Validation 快乐的小三菊
Spring i18n 国际化和 Validation 数据校验

java 国际化_JAVA国际化
Java国际化主要通过如下3个类完成 java.util.ResourceBundle:用于加载一个资源包 java.util.Locale:对应一个特定的国家/区域、语言环境。 java.text.MessageFormat:用于将

spring 国际化 yu
资源文件： message_en_US.properties login.page.name=loginpage !!! login.name=loginName login.pwd=password message_zh_CN.properties（需要进行转码） login.pa

Spring国际化实现 **热门推荐** 鹏霄万里
spring作为一个开源框架，对国际化自然提供了支持。说到国际化，有些人认为用处不大，那是因为仅仅从语言的角度考虑的。确实，如果仅简体中文就够了的话，没必要用

SpringMVC学习系列（8） 之 国际化 Mr_yem
在系列（7）中我们讲了数据的格式化显示，Spring在做格式化展示的时候已经做了国际化处理，那么如何将我们网站的其它内容（如菜单、标题等）做国际化处理呢？这就

Java Spring项目国际化(i18n)详细方法与实例
主要介绍了Java Spring项目国际化详细方法与实例,需要的朋友可以参考下

spring-framework-4.3.30.RELEASE-dist.zip
3. **MVC框架**：Spring Web MVC为构建基于HTTP的服务提供了强大的支持，包括模型-视图-控制器的设计模式、数据绑定、表单验证和国际化等功能。 4. **数据访问**：

spring-framework-4.3.6.RELEASE-dist.zip
Context模块提供了更丰富的上下文，支持国际化、事件传播、AOP代理等。SpEL则提供了一个强大的表达式语言，用于在运行时查询和操作对象属性。 2. **Data Access/I

spring-5.2.8.RELEASE-dist.zip
8. **国际化和本地化**：Spring MVC对国际化和本地化的支持更加友好，使得应用能轻松适应多语言环境。 9. **性能优化**：Spring团队对框架内部进行了大量的性能优化，

spring-web-5.2.4_spring-web_SSM框架_
6. **国际化和主题支持**：Spring Web允许应用提供多语言支持和不同主题的切换，通过LocaleResolver和ThemeResolver来实现。 7. **静态资源处理**：Spring Web可以配

JAVA (Springboot) i18n国际化语言配置 weixin_47054648
WebSocket是一种在单个TCP连接上进行全双工通信的协议。它允许服务端主动向客户端推送数据，同时也允许客户端向服务器发送数据。WebSocket通信协议于2011年被

spring-core-5.2.8.release.jar
spring-core-5.2.8.release.jar 是Spring框架的核心库文件，提供了一系列核心功能和工具类，用于支持Spring应用程序的开发和运行。 这个jar文件包含了Spring框架的核心模


Ewen Seong
码龄6年 暂无认证

84 6986 1万+ 17万+


Ewen Seong 已关注

Ewen Seong: lua官网地址: https://www.lua.org/

最新文章

Netty系列-1 NioEventLoopGroup和NioEventLoop介绍

LocalDateTime的序列化和反序列化

前端系列-9 Vue3生命周期和computed和watch

2024年 36篇 2023年 18篇

2022年 17篇 2021年 13篇

目录