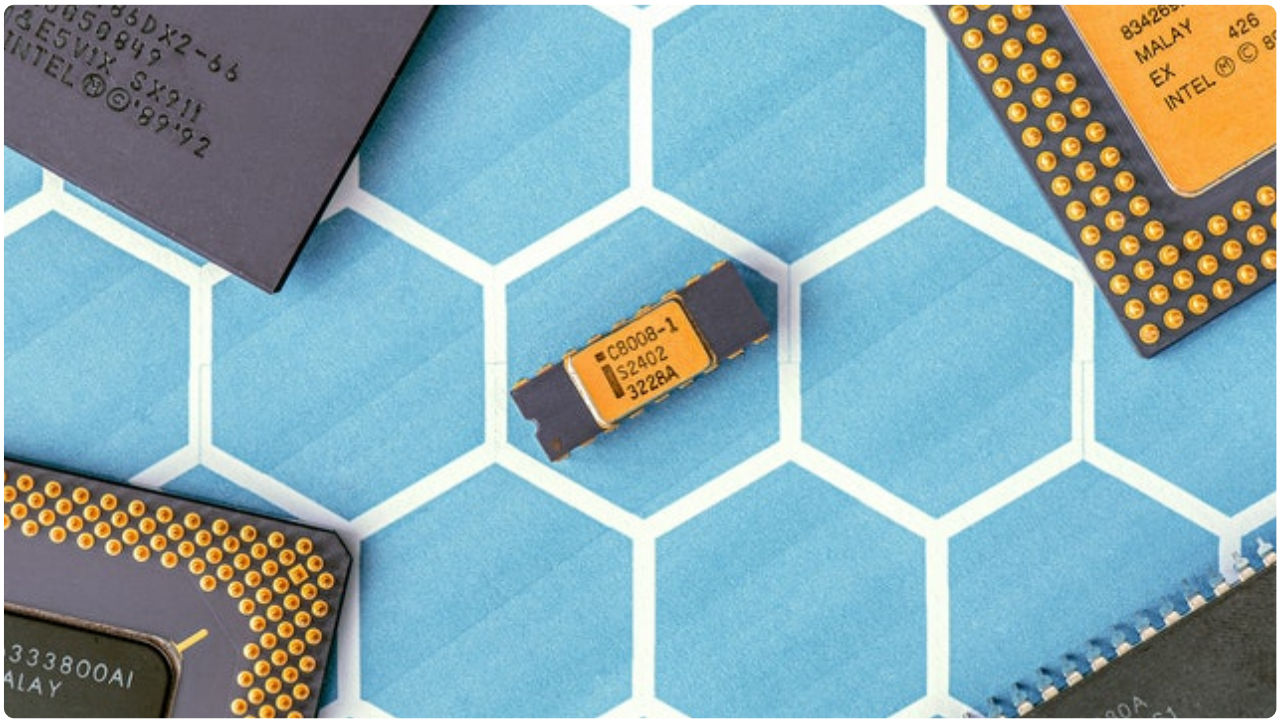
10 扩展之Spring MVC中如何实现国际化i18n

更新时间：2020-08-04 12:05:51



> 读书而不思考，等于吃饭而不消化。——波尔克

背景

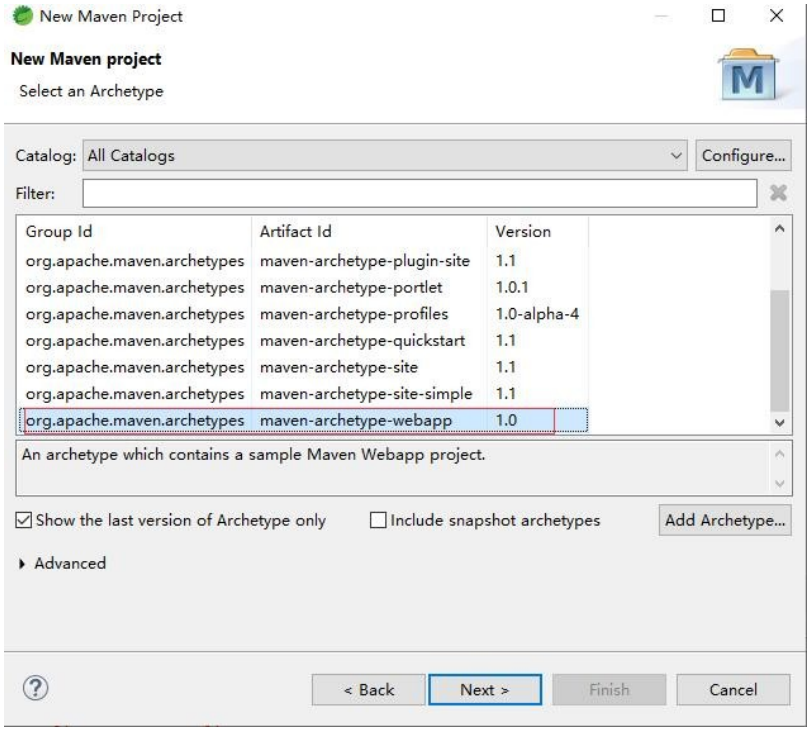
Spring MVC 的国际化是建立在Java国际化的基础之上的，其一样也是通过提供不同国家语言环境的消息资源，然后通过 Resource Bundle 加载指定 Locale 对应的资源文件，再取得该资源文件中指定 key 对应的消息。这整个过程与 Java 程序的国际化完全相同，只是 Spring MVC 框架对 Java 程序国际化进行了进一步的封装，从而简化了应用程序的国际化。

在 Spring MVC 中 i18n 是如何实现的？

Spring 提供了 ResourceBundleMessageSource 来封装 ResourceBundle 来完成国际化，Spring MVC 中需要实例化这个对象。

使用配置方式准备一个 spring-mvc 最简化项目

1 创建**maven**项目，选择**web**项目



2 添加依赖

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>
    <!-- Spring dependencies -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring.version}</version>
    </dependency>

            <!-- Servlet Dependency -->
    <dependency>
         <groupId>javax.servlet</groupId>
         <artifactId>javax.servlet-api</artifactId>
         <version>3.1.0</version>
         <scope>provided</scope>
    </dependency>
</dependencies>
```

3 修改或者添加配置文件

web.xml

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">

    <display-name>Spring MVC XML Configuration Example</display-name>

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <servlet>
        <servlet-name>my-dispatcher-servlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:dispatcher-servlet.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>my-dispatcher-servlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

dispatcher-servlet.xml

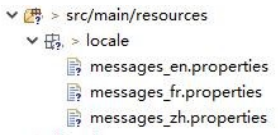
```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:component-scan base-package="com.programcreek.helloworld.controller" />
</beans>
```

4 控制器 Controller

spring mvc 国际化实例：

1.创建 **messages** 开头的不同 **locale** 的属性文件

在 Resources 目录下建立 locale 目录(名字任意取)，并创建 messages 开头的不同 locale 的属性文件



message-en.properties 文件内容：

```
app.name = resource bundle test invoked by {0}
```

message-zh.properties 文件内容：

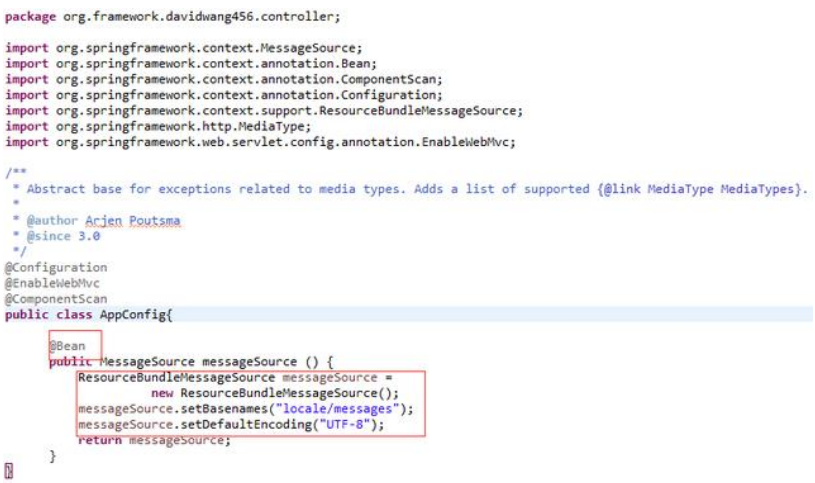
```
app.name = \u5feb\u4e50\u5b66\u4e60 {0}
```

message-fr.properties 文件内容：

```
app.name=test de regroupement de ressources invoqu par {0}
```

2.配置 ResourceBundleMessageSource 的 bean 使之生效

可以通过 xml 或者 annotation 实现，在 spring mvc 中我们统一使用注解方式：



```
package org.framework.davidwang456.controller;

import org.springframework.context.MessageSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.support.ResourceBundleMessageSource;
import org.springframework.http.MediaType;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;

/**
 * Abstract base for exceptions related to media types. Adds a list of supported {@link MediaType MediaTypes}.
 *
 * @author Arjen Poutsma
 * @since 3.0
 */
@Configuration
@EnableWebMvc
@ComponentScan
public class AppConfig{

    @Bean
    public MessageSource messageSource () {
        ResourceBundleMessageSource messageSource =
                new ResourceBundleMessageSource();
        messageSource.setBasenames("locale/messages");
        messageSource.setDefaultEncoding("UTF-8");
        return messageSource;
    }
}
```

其中 basename 是指 locale 目录下 messages 开头的文件。

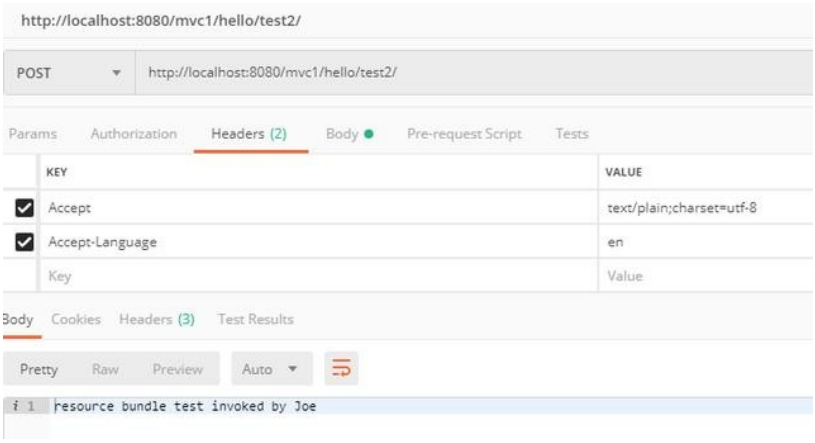
3. 在 Controller 控制器加上解析 Locale 的请求响应程序：

```
    @Autowired
    MessageSource messageSource;

    @RequestMapping("/test2")
    @ResponseBody
    public String test2 (Locale locale) {
        return messageSource.getMessage(
                "app.name", new Object[]{"Joe"}, locale);
    }
}
```

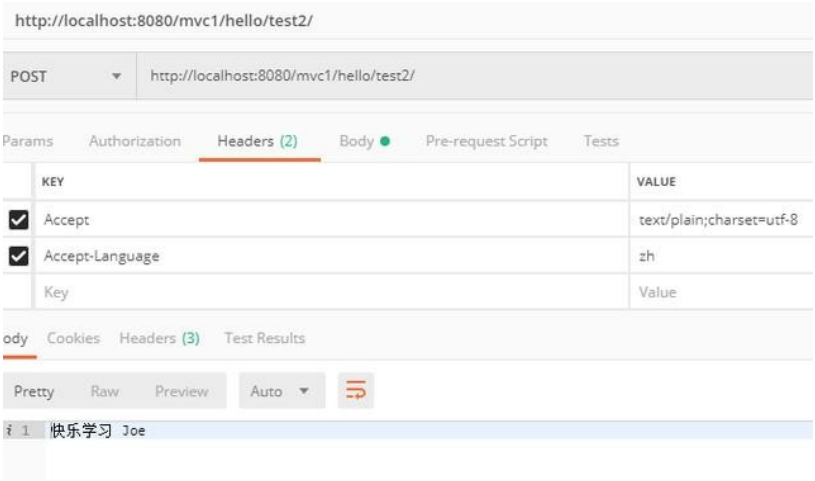
4. 测试结果

使用 postMan 发送请求，并查看响应结果。

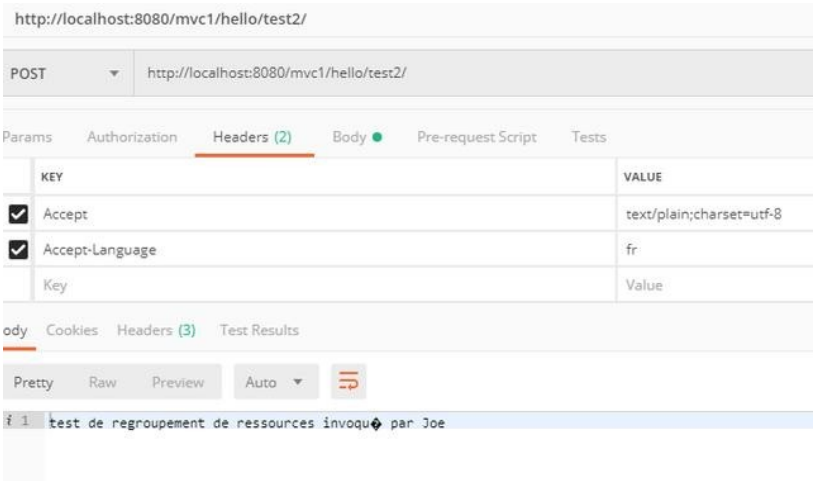
英文：使用Accept_Language 为 en，如下图所示：



中文：使用Accept_Language 为 zh，如下图所示：



法文：使用 Accept_Language 为 fr，如下图所示：



从上面的简单实例我们可以看到，通过传递不同的报文头，我们可以获取不同语言的响应，如果将这些响应不是直接返回而是传递给到视图，然后渲染视图后展示出来就是 Spring MVC 的国际化 i18n 了。为了深入了解 Spring MVC 的国际化 i18n 背后的原理，我们就要深入程序 Debug 了。

工作原理

Spring MVC 国际化 i18n 其实是分两步走的：在 ApplicationContext 启动时初始化 MessageSource，然后读取用户的 locale，最后根据 locale 读取属性 properties 文件读取键值。

1. 初始化 **Resource BundleMessageSource**

web 容器启动时调用初始化 AbstractApplicationContext#initMessageSource：

```
/**
 * Initialize the MessageSource.
 * Use parent's if none defined in this context.
 */
protected void initMessageSource() {
    ConfigurableListableBeanFactory beanFactory = getBeanFactory();
    if (beanFactory.containsLocalBean(MESSAGE_SOURCE_BEAN_NAME)) {
        this.messageSource = beanFactory.getBean(MESSAGE_SOURCE_BEAN_NAME, 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.
                hms.setParentMessageSource(getInternalParentMessageSource());
            }
        }
        if (logger.isTraceEnabled()) {
            logger.trace("Using MessageSource [" + this.messageSource + "]");
        }
    }
    else {
        // Use empty MessageSource to be able to accept getMessage calls.
        DelegatingMessageSource dms = new DelegatingMessageSource();
        dms.setParentMessageSource(getInternalParentMessageSource());
        this.messageSource = dms;
        beanFactory.registerSingleton(MESSAGE_SOURCE_BEAN_NAME, this.messageSource);
        if (logger.isTraceEnabled()) {
            logger.trace("No '" + MESSAGE_SOURCE_BEAN_NAME + "' bean, using [" + this.messageSource + "]");
        }
    }
}
```

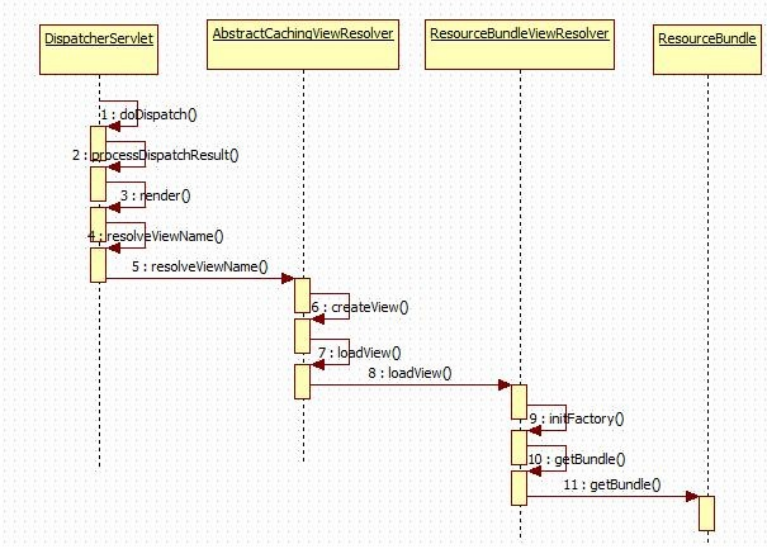
2. 根据 **locale** 解析键值

最终调用 ResourceBundleMessageSource#resolveCode 来解读 locale 信息

```
/**
 * Resolves the given message code as key in the registered resource bundles,
 * using a cached MessageFormat instance per message code.
 */
@Override
@Nullable
protected MessageFormat resolveCode(String code, Locale locale) {
    Set<String> basenames = getBasenameSet();
    for (String basename : basenames) {
        ResourceBundle bundle = getResourceBundle(basename, locale);
        if (bundle != null) {
            MessageFormat messageFormat = getMessageFormat(bundle, code, locale);
            if (messageFormat != null) {
                return messageFormat;
            }
        }
    }
    return null;
}
```

拓展

在 spring mvc 的视图显示不同 locale 的数据，可以从 DispatcherServlet 追溯。



QA

QA1： 为什么我看到有的实例中的属性文件配置，有的使用诸如 __en.properties，有的使用 _.en_US.properties?

其实在 java.util.Locale 的源码中，我们可以找到答案。

locale 既定义了跟语言有关的常量：

```
public final class Locale implements Cloneable, Serializable {

    static private final  Cache LOCALECACHE = new Cache();

    /** Useful constant for language.
     */
    static public final Locale ENGLISH = createConstant("en", "");

    /** Useful constant for language.
     */
    static public final Locale FRENCH = createConstant("fr", "");

    /** Useful constant for language.
     */
    static public final Locale GERMAN = createConstant("de", "");

    /** Useful constant for language.
     */
    static public final Locale ITALIAN = createConstant("it", "");

    /** Useful constant for language.
     */
    static public final Locale JAPANESE = createConstant("ja", "");

    /** Useful constant for language.
     */
    static public final Locale KOREAN = createConstant("ko", "");

    /** Useful constant for language.
     */
    static public final Locale CHINESE = createConstant("zh", "");
```

也定义了和区域相关的常量

```
/** Useful constant for language.
 */
static public final Locale CHINESE = createConstant("zh", "");

/** Useful constant for language.
 */
static public final Locale SIMPLIFIED_CHINESE = createConstant("zh", "CN");

/** Useful constant for language.
 */
static public final Locale TRADITIONAL_CHINESE = createConstant("zh", "TW");

/** Useful constant for country.
 */
static public final Locale FRANCE = createConstant("fr", "FR");

/** Useful constant for country.
 */
static public final Locale GERMANY = createConstant("de", "DE");

/** Useful constant for country.
 */
static public final Locale ITALY = createConstant("it", "IT");

/** Useful constant for country.
 */
static public final Locale JAPAN = createConstant("ja", "JP");

/** Useful constant for country.
 */
static public final Locale KOREA = createConstant("ko", "KR");

/** Useful constant for country.
 */
static public final Locale CHINA = SIMPLIFIED_CHINESE;

/** Useful constant for country.
 */
static public final Locale PRC = SIMPLIFIED_CHINESE;

/** Useful constant for country.
 */
static public final Locale TAIWAN = TRADITIONAL_CHINESE;

/** Useful constant for country.
 */
static public final Locale UK = createConstant("en", "GB");

/** Useful constant for country.
 */
static public final Locale US = createConstant("en", "US");

/** Useful constant for country.
 */
static public final Locale CANADA = createConstant("en", "CA");

/** Useful constant for country.
 */
static public final Locale CANADA_FRENCH = createConstant("fr", "CA");
```

根据 RFC 4647 规则，查找是根据最优匹配原则，先语言后地区。故如果一个语言仅仅在一个地区有使用的话或者多个地区使用的语言习惯一致的话，只要标识语言就可以了；

如果一个语言在多个地区使用，且使用的语言习惯有迥异，如汉字中的简体字和繁体字，那么就需要标明语言和地区了。

总结：

本篇为了阐明 Spring MVC 国际化的原理，使用了一个最简单的实例，在 Spring MVC 具体应用上，不仅仅可以通过报文头 Header 改变 Locale 来实现国际化 i18n （默认方式是AcceptHeaderLocaleResolver，通过请求头来解析Locale），还可以通过Session（SessionLocaleResolver）、cookie（CookieLocaleResolver）来改变 locale 来实现国际化。

}

09 Spring IoC容器ApplicationContext如何实现国际化

11 spring IoC容器中事件event消息的发送和接收内部工作原理揭秘