[spring 国际化](http://blog.csdn.net/gaogaoshan/article/details/21227079)

2014-03-14 10:47 2247人阅读 [评论](http://blog.csdn.net/gaogaoshan/article/details/21227079#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/gaogaoshan/article/details/21227079#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

spring（8） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

目录[(?)[+]](http://blog.csdn.net/gaogaoshan/article/details/21227079)

**java基础MessageFormat**

JDK的java.util包中提供了几个支持本地化的格式化操作工具类：NumberFormat、DateFormat、MessageFormat。

MessageFormat在NumberFormat和DateFormat的基础上提供了强大的占位符字符串的格式化功能，它支持时间、货币、数字以及对象属性的格式化操作。下面的实例演示了一些常见的格式化功能：

**[java]** [view plain](http://blog.csdn.net/gaogaoshan/article/details/21227079) [copy](http://blog.csdn.net/gaogaoshan/article/details/21227079)

[在CODE上查看代码片](https://code.csdn.net/snippets/235131)

1. //①信息格式化串
2. String pattern1 = "{0}，你好！你于{1}在工商银行存入{2} 元。";
3. String pattern2 = "At {1,time,short} On{1,date,long}，{0} paid {2,number, currency}.";
5. //②用于动态替换占位符的参数
6. Object[] params = {"John", **new** GregorianCalendar().getTime(),1.0E3};
8. //③使用默认本地化对象格式化信息
9. String msg1 = MessageFormat.format(pattern1,params);
11. //④使用指定的本地化对象格式化信息
12. MessageFormat mf = **new** MessageFormat(pattern2,Locale.US);
13. String msg2 = mf.format(params);
14. System.out.println(msg1);
15. System.out.println(msg2);

pattern1是简单形式的格式化信息串，通过{n}占位符指定动态参数的替换位置索引，{0}表示第一个参数，{1}表示第二个参数，以此类推。   
pattern2格式化信息串比较复杂一些，除参数位置索引外，还指定了参数的类型和样式。从pattern2中可以看出格式化信息串的语法是很灵活的，一个参数甚至可以出现在两个地方：如 {1,time,short}表示从第二个入参中获取时间部分的值，显示为短样式时间；而{1,date,long}表示从第二个入参中获取日期部分的值，显示为长样式时间。关于MessageFormat更详细的使用方法，请参见JDK的Javadoc。

在②处，定义了用于替换格式化占位符的动态参数，这里，我们使用到了JDK5.0自动装包的语法，否则必须采用封装类表示基本类型的参数值。   
在③处，通过MessageFormat的format()方法格式化信息串。它使用了系统默认的本地化对象，由于我们是中文平台，因此默认为Locale.CHINA。

在④处，我们显式指定MessageFormat的本地化对象。

运行上面的代码，输出以下信息：

**引用**

John，你好！你于07-1-8 下午9:58在工商银行存入1,000元。   
At 9:58 PM OnJanuary 8, 2007，John paid $1,000.00.

**国际化**

**资源文件定义**

如果应用系统中某些信息需要支持国际化功能，则必须为希望支持的不同本地化类型分别提供对应的资源文件，并以规范的方式进行命名。国际化资源文件的命名规范规定资源名称采用以下的方式进行命名：

**引用**

<资源名>\_<语言代码>\_<国家/地区代码>.properties

其中，语言代码和国家/地区代码都是可选的。<资源名>.properties命名的国际化资源文件是默认的资源文件，即某个本地化类型在系统中找不到对应的资源文件，就采用这个默认的资源文件。<资源名>\_<语言代码>.properties命名的国际化资源文件是某一语言默认的资源文件，即某个本地化类型在系统中找不到精确匹配的资源文件，将采用相应语言默认的资源文件。

举一个例子：假设资源名为resource，则语言为英文，国家为美国，则与其对应的本地化资源文件命名为resource\_en\_US.properties。信息在资源文件以属性名/值的方式表示：

**引用**

greeting.common=How are you!   
greeting.morning = Good morning!   
greeting.afternoon = Good Afternoon!   
对应语言为中文，国家/地区为中国大陆的本地化资源文件则命名为resource\_zh\_ CN.properties，资源文件内容如下：   
greeting.common=\u60a8\u597d\uff01   
greeting.morning=\u65e9\u4e0a\u597d\uff01   
greeting.afternoon=\u4e0b\u5348\u597d\uff01

**ResourceBoundle**

如果应用程序中拥有大量的本地化资源文件，直接通过传统的File操作资源文件显然太过笨拙。Java为我们提供了用于加载本地化资源文件的方便类java.util.ResourceBoundle。   
来看下面的实例：   
  
代码清单5-16  ResourceBoundle

**Java代码  收藏代码**

1. ResourceBundle rb1 = ResourceBundle.getBundle("com/baobaotao/i18n/resource", Locale.US);
2. ResourceBundle rb2 = ResourceBundle.getBundle("com/baobaotao/i18n/resource", Locale.CHINA);
3. System.out.println("us:"+rb1.getString("greeting.common"));
4. System.out.println("cn:"+rb2.getString("greeting.common"));

rb1加载了对应美国英语本地化的resource\_en\_US.properties资源文件；而rb2加载了对应中国大陆中文的resource\_zh\_CN.properties资源文件。运行上面的代码，将输出以下信息：

**引用**

us:How are you!   
cn:您好！

加载资源文件时，如果不指定本地化对象，将使用本地系统默认的本地化对象。所以，在中文系统中，ResourceBundle.getBundle("com/baobaotao/i18n/resource")语句也将返回和代码清单5-14中rb2相同的本地化资源。

**在资源文件中使用格式化串MessageFormat**

在上面的资源文件中，属性值都是一般的字符串，它们不能结合运行时的动态参数构造出灵活的信息，而这种需求是很常见的。要解决这个问题很简单，只须使用带占位符的格式化串作为资源文件的属性值并结合使用MessageFormat就可以满足要求了。

**引用**

greeting.common=How are you!{0},today is {1}   
greeting.morning = Good morning!{0},now is {1 time short}   
greeting.afternoon = Good Afternoon!{0} now is {1 date long}

将该资源文件保存在fmt\_resource\_en\_US.properties中，按照同样的方式编写对应的中文本地化资源文件fmt\_resource\_zh\_CN.properties。

下面，我们联合使用ResourceBoundle和MessageFormat得到美国英文的本地化问候语：   
  
代码清单5-17  资源文件格式化串处理

**Java代码  收藏代码**

1. //①加载本地化资源
2. ResourceBundle rb1 =
3. ResourceBundle.getBundle("com/baobaotao/i18n/fmt\_ resource",Locale.US);
4. ResourceBundle rb2 =
5. ResourceBundle.getBundle("com/baobaotao/i18n/fmt\_ resource",Locale.CHINA);
6. Object[] params = {"John", **new** GregorianCalendar().getTime()};

9. String str1 = **new** MessageFormat(rb1.getString("greeting.common"),Locale. US).format(params); ②
10. String str2 =**new** MessageFormat(rb2.getString("greeting.morning"),Locale. CHINA).format(params);
11. String str3 =**new** MessageFormat(rb2.getString("greeting.afternoon"),Locale. CHINA).format(params);
12. System.out.println(str1);
13. System.out.println(str2);
14. System.out.println(str3);

运行以上的代码，将输出以下信息：

**引用**

**How are you!John,today is 1/9/07 4:11 PM   
早上好！John，现在是下午4:11   
下午好！John，现在是2007年1月9日**

**spring 国际化MessageSource**

MessageSource的类结构

MessageSource分别被HierarchicalMessageSource和ApplicationContext接口扩展

HierarchicalMessageSource接口最重要的两个实现类是ResourceBundleMessageSource和ReloadableResourceBundleMessageSource。它们基于Java的ResourceBundle基础类实现，允许仅通过资源名加载国际化资源。ReloadableResourceBundleMessageSource提供了定时刷新功能，允许在不重启系统的情况下，更新资源的信息。StaticMessageSource主要用于程序测试，它允许通过编程的方式提供国际化信息。而DelegatingMessageSource是为方便操作父MessageSource而提供的代理类。

**ResourceBundleMessageSource**

代码清单5-18  通过ResourceBundleMessageSource配置资源

**Xml代码  收藏代码**

1. **<bean** id="myResource"
2. class="org.springframework.context.support.ResourceBundleMessageSource"**>**
3. <!--①通过基名指定资源，相对于类根路径-->
4. **<property** name="basenames"**>**
5. **<list>**
6. **<value>**com/baobaotao/i18n/fmt\_resource**</value>**
7. **</list>**
8. **</property>**
9. **</bean>**

代码清单5-19  访问国际化消息：ResourceBundleMessageSource

**Java代码  收藏代码**

1. String[] configs = {"com/baobaotao/i18n/beans.xml"};
2. ApplicationContext ctx = **new** ClassPathXmlApplicationContext(configs);
4. //①获取MessageSource的Bean
5. MessageSource ms = (MessageSource)ctx.getBean("myResource");
6. Object[] params = {"John", **new** GregorianCalendar(). getTime()};
8. //②获取格式化的国际化信息
9. String str1 = ms.getMessage("greeting.common",params,Locale.US);
10. String str2 = ms.getMessage("greeting.morning",params,Locale.CHINA);
11. String str3 = ms.getMessage("greeting.afternoon",params,Locale.CHINA);
12. System.out.println(str1);
13. System.out.println(str2);
14. System.out.println(str3);

比较代码清单5-19中的代码，我们发现最主要的区别在于我们无须再分别加载不同语言、不同国家/地区的本地化资源文件，仅仅通过资源名就可以加载整套的国际化资源文件。此外，我们无须显式使用MessageFormat操作国际化信息，仅通过MessageSource# getMessage()方法就可以完成操作了。这段代码的运行结果与代码清单5 17的运行结果完全一样。

**ReloadableResourceBundleMessageSource**

前面，我们提到该实现类比之于ResourceBundleMessageSource的唯一区别在于它可以定时刷新资源文件，以便在应用程序不重启的情况下感知资源文件的变化。很多生产系统都需要长时间持续运行，系统重启会给运行带来很大的负面影响。这时，通过该实现类就可以解决国际化信息更新的问题。请看下面的配置：   
  
代码清单5-20  通过ReloadableResourceBundleMessageSource配置资源

**Xml代码  收藏代码**

1. **<bean** id="myResource"
2. lass="org.springframework.context.support. ReloadableResourceBundleMessageSource"**>**
3. **<property** name="basenames"**>**
4. **<list>**
5. **<value>**com/baobaotao/i18n/fmt\_resource**</value>**
6. **</list>**
7. **</property>**
8. <!--① 刷新资源文件的周期，以秒为单位-->
9. **<property** name="cacheSeconds" value="5"**/>**
10. **</bean>**

在上面的配置中，我们通过cacheSeconds属性让ReloadableResourceBundleMessageSource每5秒钟刷新一次资源文件（在真实的应用中，刷新周期不能太短，否则频繁的刷新将带来性能上的负面影响，一般不建议小于30分钟）。cacheSeconds默认值为-1表示永不刷新，此时，该实现类的功能就蜕化为ResourceBundleMessageSource的功能。   
代码清单5-21  刷新资源：ReloadableResourceBundleMessageSource

**Java代码  收藏代码**

1. String[] configs = {"com/baobaotao/i18n/beans.xml"};
2. ApplicationContext ctx = **new** ClassPathXmlApplicationContext(configs);
4. MessageSource ms = (MessageSource)ctx.getBean("myResource");
5. Object[] params = {"John", **new** GregorianCalendar().getTime()};
7. **for** (**int** i = 0; i < 2; i++) {
8. String str1 = ms.getMessage("greeting.common",params,Locale.US);
9. System.out.println(str1);
10. Thread.currentThread().sleep(20000); //①模拟程序应用，在此期间，我们更改资源文件
11. }

在①处，我们让程序睡眠20秒钟，在这期间，我们将fmt\_resource\_zh\_CN.properties资源文件的greeting.common键值调整为：

**引用**

---How are you!{0},today is {1}---

我们将看到两次输出的格式化信息分别对应更改前后的内容，也即本地化资源文件的调整被自动生效了：

**引用**

How are you!John,today is 1/9/07 4:55 PM   
---How are you!John,today is 1/9/07 4:55 PM---

**容器级的国际化信息资源**

在如图5-7所示的MessageSource类图结构中，我们发现ApplicationContext实现了MessageSource的接口。也就是说ApplicationContext的实现类本身也是一个MessageSource对象。   
  
将ApplicationContext和MessageSource整合起来，乍一看挺让人费解的，Spring这样设计的意图究竟是什么呢？原来Spring认为：在一般情况下，国际化信息资源应该是容器级。我们一般不会将MessageSource作为一个Bean注入到其他的Bean中，相反MessageSource作为容器的基础设施向容器中所有的Bean开放。只要我们考察一下国际化信息的实际消费场所就更能理解Spring这一设计的用意了。国际化信息一般在系统输出信息时使用，如Spring MVC的页面标签，控制器Controller等，不同的模块都可能通过这些组件访问国际化信息，因此Spring就将国际化消息作为容器的公共基础设施对所有组件开放。   
  
既然一般情况下我们不会直接通过引用MessageSource Bean使用国际信息，那如何声明容器级的国际化信息呢？我们其实在5.1.1节讲解Spring容器的内部工作机制时已经埋下了伏笔：在介绍容器启动过程时，我们通过代码清单5-1对Spring容器启动时的步骤进行剖析，④处的initMessageSource()方法所执行的工作就是初始化容器中的国际化信息资源：它根据反射机制从BeanDefinitionRegistry中找出名称为“messageSource”且类型为org.springframework.context.MessageSource的Bean，将这个Bean定义的信息资源加载为容器级的国际化信息资源。请看下面的配置：

代码清单5-22  容器级资源的配置

**Xml代码  收藏代码**

1. <!--①注册资源Bean,其Bean名称只能为messageSource -->
2. **<bean** id="messageSource"
3. class="org.springframework.context.support.ResourceBundleMessageSource"**>**
4. **<property** name="basenames"**>**
5. **<list>**
6. **<value>**com/baobaotao/i18n/fmt\_resource**</value>**
7. **</list>**
8. **</property>**
9. **</bean>**

下面，我们通过ApplicationContext直接访问国际化信息，如代码清单5 23所示：   
  
代码清单5-23  通过ApplicationContext访问国际化信息

**Java代码  收藏代码**

1. String[] configs = {"com/baobaotao/i18n/beans.xml"};
2. ApplicationContext ctx = **new** ClassPathXmlApplicationContext(configs);
3. //①直接通过容器访问国际化信息
4. Object[] params = {"John", **new** GregorianCalendar().getTime()};
6. String str1 = ctx.getMessage("greeting.common",params,Locale.US);
7. String str2 = ctx.getMessage("greeting.morning",params,Locale.CHINA);
8. System.out.println(str1);
9. System.out.println(str2);

运行以上代码，输出以下信息：

**引用**

How are you!John,today is 1/9/07 5:24 PM   
早上好！John，现在是下午5:24

假设MessageSource Bean名字没有命名为“messageSource”，以上代码将抛出NoSuchMessageException异常。