参考：<http://blog.csdn.net/revivedsun/article/details/51330396>

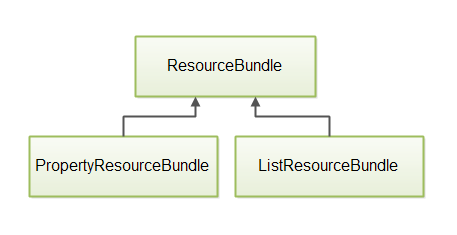
**Java中ResourceBundle类的使用**

**原文地址**：http://tutorials.jenkov.com/java-internationalization/resourcebundle.html

java.util.ResourceBundle类用来存储文本以及本地化相关数据。本文将介绍ResourceBundle抽象类及它的子类。

ResourceBundle的类层次结构

ResourceBundle类有两个子类，PropertyResourceBundle及ListResourceBundle。下面是这几个类的结构图。



PropertyResourceBundle将本地化的文本存储于Java property文件中。

创建ResourceBundle

可以按下面的方式创建ResourceBundle类的实例。

Locale locale = **new** Locale("en", "US");

ResourceBundle labels = ResourceBundle.getBundle("i18n.MyBundle", locale);

System.out.println(labels.getString("label1"));

首先需要创建Locale实例。然后将locale实例传给ResourceBundle.getBundle()方法，此外还需要传递要加载的resource bundle名称。最后可以通过ResourceBundle 的getString()，getObject()等方法访问本地化值。

实际上并没有创建ResourceBundle实例，而是创建了ResourceBundle子类的实例。实例的创建都是通过工厂方法来完成。首先ResourceBundle类将会查找ListResourceBundle，随后会查找PropertyResourceBundle。查找是根据ListResourceBundle类名来匹配resource boundle的名字来完成(getBundle()方法的第一个参数)。如果找不到，则根据PropertyResourceBundle resource bundle来匹配。

Property Files作为ResourceBundle

可以使用标准的属性文件来存储本地化文本。可以通过ResourceBundle 类加载这些属性。

Locale locale = new Locale("en", "US");

ResourceBundle labels = ResourceBundle.getBundle("i18n.MyBundle", locale);

System.out.println(labels.getString("label1"));

上面的例子中，你需要将MyBundle放在包i18n中。

resource bundle 的名称类似类名。如i18n.MyBundle意思是，包i18n中的属性文件MyBundle.properties。

下面是属性文件中的内容。

label1 = Label 1 is done!

label2 = Label 2 is through!

上述文件是由一组key,value对构成。key在=号左侧，value在右侧。value是你要本地化的内容。  
将不同的语言放置于不同的属性文件中

为了提供不同语言的字符串，需要为每种语言创建属性文件，文件名的格式为 xxxx\_语言编码。例如：

**[plain]** [view plain](http://blog.csdn.net/revivedsun/article/details/51330396) [copy](http://blog.csdn.net/revivedsun/article/details/51330396)

MyBundle.properties

MyBundle\_da.properties

MyBundle\_de.properties

MyBundle\_fr.properties

这些文件应该放在同一个包中。  
没有语言后缀的文件(例如：MyBundle.properties)是默认的属性文件。  
  
带有语言编码后缀的属性文件包含同样的key但是value是以不同的语言来标识。如danish  属性文件：

1. label1 = Label 1 er klar!
2. label2 = Label 2 er igennem!

将类作为ResourceBundle  
也可以使用一组类来获取资源，使用类，还可以使用除字符串值外的其它类型资源。  
与属性文件类似，可以创建带有 bundle base名称与语言后缀的一组类。

例如：  
i18n.MyClassBundle  
i18n.MyClassBundle\_da  
i18n.MyClassBundle\_en  
i18n.MyClassBundle\_de

下面是一个默认的bundle class文件实现的例子。

1. **import** java.util.ListResourceBundle;
3. **public** **class** MyClassBundle **extends** ListResourceBundle {
5. @Override
6. **protected** Object[][] getContents() {
7. **return** contents;
8. }
10. **private** Object[][] contents = {
11. { "price"   , **new** Double(10.00) },
12. { "currency", "EUR" },
13. };
14. }

下面是danish  resource bundle的实现。

1. public class MyClassBundle\_da extends ListResourceBundle {
3. @Override
4. protected Object[][] getContents() {
5. return contents;
6. }
8. private Object[][] contents = {
9. { "price"   , new Double(75.00) },
10. { "currency", "DKK" },
11. };
13. }

注意contents数组，是一个包含key,value的二维数组。price，currency是key，而key右侧为本地化的值。获取ListResourceBundle 实例的方式与获取PropertyResourceBundle实例的方式一样，下面是获取默认ResourceBundle 及丹麦语ResourceBundle 的例子。

**[java]** [view plain](http://blog.csdn.net/revivedsun/article/details/51330396) [copy](http://blog.csdn.net/revivedsun/article/details/51330396)

1. Locale locale = new Locale("de", "DE"); //no bundle for German -> default
2. ResourceBundle bundle = ResourceBundle.getBundle("i18n.MyClassBundle", locale);
4. System.out.println("price   : " + bundle.getObject("price"));
5. System.out.println("currency: " + bundle.getObject("currency"));

8. locale = new Locale("da", "DK");
9. bundle = ResourceBundle.getBundle("i18n.MyClassBundle", locale);
11. System.out.println("price   : " + bundle.getObject("price"));
12. System.out.println("currency: " + bundle.getObject("currency"));
13. 输出是：
14. price   : 10.0
15. currency: EUR
16. price   : 75.0
17. currency: DKK

从ResourceBundle中获取值  
获取ResourceBundle实例后可以通过下面的方法获得本地化值。  
getObject(String key);  
getString(String key);  
getStringArray(String key);  
还可以通过keySet()方法获取所有的key。Set<String> keys = bundle.keySet();  
其它ResourceBundle 方法可以通过查看文档获得。

参考：<http://blog.csdn.net/u012345283/article/details/42082253>

# java.util.ResourceBundle用法详解

## 初识国际化和ResourceBundle

这个类主要用来解决国际化和本地化问题。国际化和本地化可不是两个概念，两者都是一起出现的。可以说，国际化的目的就是为了实现本地化，详细的介绍可以看本文的最后。比如对于“取消”，中文中我们使用“取消”来表示，而英文中我们使用“cancel”。若我们的程序是面向国际的（这也是软件发展的一个趋势），那么使用的人群必然是多语言环境的，实现国际化就非常有必要。而ResourceBundle可以帮助我们轻松完成这个任务：当程序需要一个特定于语言环境的资源时（如 String），程序可以从适合当前用户语言环境的资源包（大多数情况下也就是.properties文件）中加载它。这样可以编写很大程度上独立于用户语言环境的程序代码，它将资源包中大部分（即便不是全部）特定于语言环境的信息隔离开来。

这使编写的程序可以：

* 轻松地本地化或翻译成不同的语言
* 一次处理多个语言环境
* 以后可以轻松进行修改，以便支持更多的语言环境

## 测试及验证

下面我们来模拟一个多语言的环境

定义四个资源文件：res\_en\_US.properties、res\_zh\_CN.properties、res\_zh.properties、res.properties

res\_en\_US.properties：cancelKey=cancel

res\_zh\_CN.properties：cancelKey=\u53D6\u6D88（取消）

res\_zh.properties：cancelKey=\u53D6\u6D88zh（取消zh）

res.properties：cancelKey=\u53D6\u6D88default（取消default）

命名规则按照：资源名+\_语言\_国别.properties，每个资源文件中定义了本地化的信息，那么系统如何取到对应的资源文件呢？

**[java]** [view plain](http://blog.csdn.net/u012345283/article/details/42082253) [copy](http://blog.csdn.net/u012345283/article/details/42082253)

1. ResourceBundle bundle = ResourceBundle.getBundle("res", **new** Locale("zh", "CN"));

其中new Locale("zh", "CN");这个对象就告诉了程序你的本地化信息，就拿这个来说吧：程序首先会去classpath下寻找res\_zh\_CN.properties

若不存在，那么会去找res\_zh.properties，若还是不存在，则会去寻找res.properties，要还是找不到的话，那么就该抛异常了：MissingResourceException

我们可以来写个测试程序验证一下：

**[java]** [view plain](http://blog.csdn.net/u012345283/article/details/42082253) [copy](http://blog.csdn.net/u012345283/article/details/42082253)

1. **package** bundle.test;
3. **import** java.util.Locale;
4. **import** java.util.ResourceBundle;
6. **public** **class** BundleTest {
8. **public** **static** **void** main(String args[]) {
9. ResourceBundle bundle = ResourceBundle.getBundle("res", **new** Locale("zh", "CN"));
10. String cancel = bundle.getString("cancelKey");
11. System.out.println(cancel);
13. bundle = ResourceBundle.getBundle("res", Locale.US);
14. cancel = bundle.getString("cancelKey");
15. System.out.println(cancel);
17. bundle = ResourceBundle.getBundle("res", Locale.getDefault());
18. cancel = bundle.getString("cancelKey");
19. System.out.println(cancel);
21. bundle = ResourceBundle.getBundle("res", Locale.GERMAN);
22. cancel = bundle.getString("cancelKey");
23. System.out.println(cancel);
24. }
25. }

输出：

取消

cancel

取消

取消

这里前三个都在我们的预期范围之内，但是最后一个GERMAN，应该会去使用res.properties这个资源包吧？怎么使用了res\_zh\_CH.properties？

原来ResourceBundle为我们提供了一个fallback（也就是一个备用方案），这个备用方案就是根据当前系统的语言环境来得到的本地化信息。

所以若是找不到GERMAN的，之后就会去找CHINA了，所以找到了res\_zh\_CH.properties这个资源包

这点我也是看了源代码才明白的，下面就贴上一些关键的源代码：

**[java]** [view plain](http://blog.csdn.net/u012345283/article/details/42082253) [copy](http://blog.csdn.net/u012345283/article/details/42082253)

1. ResourceBundle baseBundle = **null**;
2. **for** (Locale targetLocale = locale;
3. targetLocale != **null**;
4. targetLocale = control.getFallbackLocale(baseName, targetLocale)) {// 这里就是去拿备用方案的
5. // do something 我们暂时不关心
6. }

跟踪control.getFallbackLocale(baseName, targetLocale)看看备用方案到底是什么？

**[java]** [view plain](http://blog.csdn.net/u012345283/article/details/42082253) [copy](http://blog.csdn.net/u012345283/article/details/42082253)

1. **public** Locale getFallbackLocale(String baseName, Locale locale) {
2. **if** (baseName == **null**) {
3. **throw** **new** NullPointerException();
4. }
5. Locale defaultLocale = Locale.getDefault();
6. **return** locale.equals(defaultLocale) ? **null** : defaultLocale;
7. }

当显式定义的本地化信息并不是当前系统的本地化信息时，若未能通过显式定义的找到资源包，那么就去转而通过当前系统的本地化信息去找了~

最后放一点小知识吧~

## 国际化和本地化

国际化（Internationalization）是设计一个适用于多种语言和地区的应用程序的过程。适用于多种语言和地区的含义是当使用不同语言及处于不同的地区的用户在使用这个应用程序时，应用程序必须使用他们能看懂的语言和符合他们文化习惯来显示信息。国际化有时候被简称为i18n，因为有18个字母在国际化的英文单词的字母i和n之间。

一个国际化的程序通常具有以下特征：  
有一个附加的本地化数据（localized  data）及拥有在全世界各个地区执行的能力。

* 文本的元素，比如状态信息或GUI截面的lables，不是直接写（hardcoded）在程序中，而是被存储在本地化的数据中，并且能被程序正确的动态的使用。
* 支持新的语言时，不需要修改程序，不需要重新编译。
* 文化差异的数据，比如日期和货币，必须根据用户的语言和习惯显示不同的格式。
* 可以被迅速的本地化。

        本地化（Localization）是指通过增加本地描述的构件（locale-specific components ）和文字翻译工作来使应用程序适应于不同的语言和地区的过程。本地化有时候被简称为l10n，应为有10个字母在本地化的英文单词的字母l和n之间（即localization）。通常本地化最耗时的工作应该是文字翻译。本地化工作者们要根据地区的具体需求来为日期、数字和通货等数据建立新的格式。其他类型的数据，像声音，图象等，也需要根据具体需要来决定是否本地化。