Java 语言国际化的设计与实现*

冀振燕, 程 虎, 梅 嘉

(中国科学院 软件研究所,北京 100080)

E-mail: chenghu@126.com

http://www.ios.ac.cn

摘要: 指出本地化的缺点,深入分析了 Java 语言的内部编码,设计并实现了支持国际化的编译器,采用字符编码转换的方案,使多语种字符、字符串的操作达到了与英文字符、字符串完全一致的目标,并节省了类文件的存储空间,也使程序具备了有条件的字符集无关性,在一定程度上解决了"万码奔腾"的问题. 采用编译信息和与语境无关的程序代码分离的方案解决了提示信息的国际化问题,提出了采用小字库方案来解决含有本地字符的程序的真正跨平台问题. 通过在加载类时采用对类名进行字符编码转换的方案解决了 JDK(Java development kit)所未能解决的含有本地字符的类名的支持问题;针对 JDK 不同版本间的兼容性问题,提出采用根据字符编码特点来选用不同编码转换机制的解决方案. 测试结果证明,采用所设计的国际化方案可实现彻底的国际化支持.

关键词: Java 语言;国际化

中图法分类号: TP312 **文献标识码**: A

Java 语言的跨平台性使它成为计算机上的世界语,作为计算机的世界语,它的国际化是必要的. 另外,Internet 和其他分布网络的迅速发展也增加了创建国际化软件必要性和重要性.

1 本地化的缺点

以前通常采用本地化的方法来达到生成适合本地环境的软件的目标,本地化实现的思想是:修改程序中本地敏感的数据,使其支持特定的地方语言,符合特定的地方习惯表达[1].

虽然采用对西文软件本地化的方法可以达到生成适合本地环境的软件这一目标,但是,像这样直接对程序进行本地化存在如下几个问题:

(1) 程序员必须为每一个新的本地创建程序的一个新版本.

(2) 程序员必须维护同一个软件的几个版本. 备份每一个版本的软件,但该软件的大部分可能都是独立于语言的部分. 当程序员对软件进行修改时,无论所修改的是否与语言相关的部分,都要同时修改几个版本的软件.

(3) 每个新的本地版本都需要大量的工作去寻找和修改程序中所有与语言相关的部分.

(4) 如果采用直接修改西文软件的方法,还可能涉及到版权问题.

(5) 有时不能达到彻底的本地化. 例如,对于字符、字符串的本地化,就存在着一些问题:在许多程序语言中,像 C/C++,PASCAL 等,char 型数据定义为一个字节,即 8 位,而汉字通常是用两个字节表示的,违反了程序语言的规定,因而很难通过修改编译程序来实现对中文字符的支持. 至于汉字字符串,虽然程序语言中适用于

* 收稿日期: 1999-05-30; 修改日期: 1999-09-10

基金项目: 国家"九五"重点科技攻关资助项目(96-737-01-03)

作者简介: 冀振燕(1972—),女,河南人,博士,主要研究领域为语言编译,软件中文化,人工智能;程虎(1938—),男,江苏人,研究员,博士生导师,主要研究领域为人工智能,神经网络,语言编译等;梅嘉(1972—),男,四川人,硕士,主要研究领域为语言编译,Java 技术.

西文字符串的操作一般都适用于中文字符串,但是,由于汉字是两个字节的,所以,一个汉字通常被当做两个西文字符来处理,这样,在进行汉字字符串的操作时,像汉字的截取、插入、替换等,就有可能将一个汉字分为两截.

可见,为了生成适合不同本地的软件,采用本地化的方法存在着许多问题,为了避免这些问题,程序员在设计程序时就应该考虑将软件实现为一个国际化的软件,以适应不同的地方,满足不同国家用户的需求.

2 Java语言的内部编码

Java语言规范规定在内部使用Unicode2.0字符编码(JDK1.1以前的版本使用Unicode 1.1.5版本).除了注释、标识符、字符及字符串字面量的内容,Java程序中的所有输入元素都仅有ASCII字符或产生ASCII字符的Unicode转义序列构成,所以,字符在被读入时要被转换成Unicode字符[2].

在类文件中,字符是以Unicode或UTF-8格式[3]存储的:作为字节码指令操作数的字符是以Unicode字符的形式存储的,而构成字符串常量、标识符(文件名、类名、变量名、数组名等)的字符都是以UTF-8的格式存储在常数池中的,所以在生成类文件时,字符的Unicode码还要被转换成UTF-8格式.

Java程序的字符编码的转换如图1所示,虚线框表示在编译过程中所作的转换,实线框表示在类文件执行过程中所作的转换.虚线框与实现框的公共部分表示了类文件中字符的存储格式.实线箭头表示进行了转换;虚线箭头表示没有转换过程,指出了数据的流向.从图中可以看出,虚拟机中字符编码的转换实际上就是编译器中字符编码转换的逆过程:UTF-8格式首先转换成Unicode编码,字符的Unicode编码再转换为字符的本地字符集编码,输出.

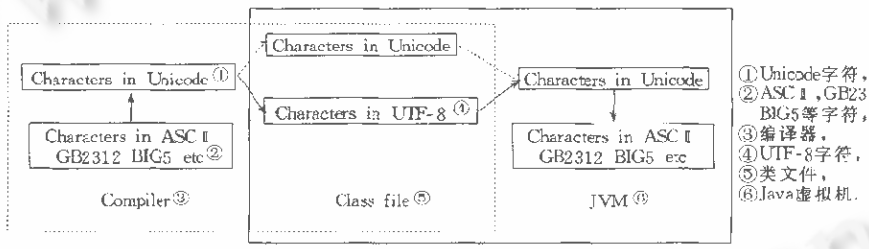

Fig.1 Transformations between Java code encodings
图1 Java程序字符编码的转换

3 国际化Java语言的实现[4,5]

编译器是决定程序语言是否支持本地化或国际化的关键,所以,在设计实现Java语言的编译系统时,为了支持多语种的注释、保留字、标识符、字符、字符串和提示信息,我们作了合理的设计,使Java编译系统实现为支持多语种的Java编译系统.

3.1 注释语句

国际化的注释是指程序注释语句中可以使用多国文字.注释语句的国际化支持是在词法分析模块中实现的,当词法分析模块对注释语句进行词法分析时,无论注释语句中是否含有多国文字字符,都直接将其滤除,不作任何处理.

3.2 保留字

保留字是由程序语言定义的具有固定意义的标识符,Java语言中的保留字包括关键字、布尔字面量"true"和"false"、空字面量"null"[2].

Java语言保留字的国际化,也是在词法分析模块部分进行处理的.首先构造一个保留字转换表,见表1,将汉字、日文等多语种保留字与英文保留字对应.在词法分析时,当扫描到一个标识符时,首先查询保留字转换表,若能在表中查到,则将其替换为相应的英文保留字,再查询编译器的符号表,返回该保留字的符号类型.

Table 1 Translation table of keywords

表 1 保留字转换表

Multilingual reserved words①	English reservec words②
如果	if
那么	then

①多国文字保留字,②英文保留字.

这样就使得含有多语种保留字的程序也能正确地编译、运行.无论是多国文字形式的保留字还是西文保留字都不能用做标识符.

对于保留字的国际化,一直存在着争议,大多数专家都认为保留字的国际化没有意义,会使程序员感到不习惯,而且还会影响程序速度.

3.3 字符、字符串

由于 Java 语言在内部使用 Unicode 字符集,而一般程序并不是由 Unicode 字符组成的,所以,在读入程序输入流时,编译器要将各种字符转换为 Unicode 字符[2].对于多字节字符通常有两种转换方案:一种方案是将一个多字节字符的编码当做多个单字节字符的编码处理,分别转换成 Unicode 编码.例如,像一个双字节的汉字字符编码就被当做两个单字节字符编码而转换为两个 Unicode 编码,这样做存在着一个问题:一个 N 字节的字符编码被转换为 N 个 Unicode 编码(即 N 个 16 位长),那么它就不能作为字符值,因为字符字面量是 16 位的.另一种方案是将字符编码(无论是否多字节字符)直接转换为相应的 Unicode 字符的编码,这样就解决了将多字节字符当做多个单字节字符处理所引起的问题.另外,还能节省存储空间.例如,按第 1 种方案,类文件常数池中的一个汉字字符占 4 个字节,而按第 2 种方案,则占 3 个字节.

在本编译器中就采用第 2 种转换方案.在作转换时,如果本地字符集的字符码值与 Unicode 字符集的码值有线性关系,则构造码值转换函数,否则构造转换表.在本编译器中就构造了 GBK 到 Unicode 的码值转换表,因为 GBK 的字符码值与 Unicode 字符码值之间没有线性关系,在读入 GBK 字符时,通过查表将其转换为相应的 Unicode 字符.对于每一个字符集,我们都可以构造其与 Unicode 字符集间的转换关系表或转换关系函数,在编译程序时,通过设置不同的编译参数来绑定不同的转换表或转换函数.

如图 1 所示,字符在类文件里是以 Unicode 或 UTF-8 的格式存储的,虚拟机在执行类文件时,对于常数池里以 UTF-8 格式存储的字符则首先转换成 Unicode 编码,为了能在本地平台上正确地显示,显然要根据本地平台的字符集作编译转换的逆转换,将 Unicode 字符转换为相应的本地字符,所以还应该构造 Unicode 字符集到本地字符集的转换表.特别值得提出的是:虚拟机在加载类时应该像处理别的标识符名一样来处理类名,即在加载类时要调用编码转换子程序,将类名中的本地字符的 UTF-8 格式转换为相应的本地字符编码,否则会找不到相应的类.

通过字符编码转换的处理,编译器就可以支持含有多语种字符、字符串的程序,而且对英文字符、字符串和对含有多国文字的字符、字符串的处理是一样的.例如,一个汉字字符的长度为 1 而不再被当做两个字符进行处理.另外,还有一个最大的好处是:如果程序的编译平台与程序的执行平台的字符集不同,但字符集都含有程序中所出现的本地字符,通过上述转换还可以达到有条件的字符集无关性.例如,如果编译平台的字符集是 GBK,执行平台的字符集是 BIG5,Java 程序中出现的汉字在两个字符集中都存在,该 Java 程序在编译后,GBK 的字符被转换成 Unicode 码或 UTF-8 格式,在执行时,Unicode 码或 UTF-8 格式就会被转换成 BIG5 字符,从而在运行平台上得以正确显示,这在某种程度上解决了汉字的"万码奔腾"问题.

3.4 标识符

标识符用来表示各种名字,如 Java 语言中的文件名、类名、方法名、数组名、变量名等.Java 语言中标识符的定义以字母或ASCII码下划线"_"、美元符号"$"为首字符,随后的字符可以是字母或数字的字符串[2].标识符是大小写敏感的,不能与保留字相同.作为国际化的标识符,显然,字母应该包括多语种字符,数字应该包括多语种数字.在 Unicode 字符集中,凡是名字中有"DIGIT"这个词的字符就是数字,凡是名字中有"LETTER"这个词的字符就是字母.具体范围见文献[6].

在进行标识符的词法分析过程中,当读入一个字符时,如果是第1个字符,要判断该字符是否属于Unicode字符集中的字母范围,若属于,则为有效字符,继续扫描;如果是第2个字符,除了要判断该字符是否属于Unicode字符集中的字母范围之外,还要判断其是否属于Unicode字符集中的数字范围,若属于,则为有效字符,继续扫描.其实,除了9个LATIN数字之外,其余的本地数字也可以作为标识符的第1个字符,但考虑到标识符定义的一致性,不将其作为有效的标识符的第1个字符.这样,本编译器就实现为支持多语种标识符的编译器.

3.5 编译信息的处理

编译信息是程序中与语言相关的部分,为了使本编译器的编译信息按照用户所喜欢的语言和形式显示出来,在实现本编译器时,将编译信息同与语境无关的程序代码分离开来,集中放在一个独立的表文件中,并且给每一条信息一个key.当编译器需要给出编译信息时,例如,发出编译错误信息,这个编译信息就通过信息的key去引用.首先,编译器查找编译信息表文件,寻找与那个key相对应的编译信息,然后取出这条信息,显示出来.

由于依赖于语言的编译信息被分离出来,并且集中于一个独立的表文件中,所以,对于不同的语言和习惯应构造不同的信息表文件来存放用这种语言和习惯来表达的编译信息.当用编译器编译程序时,通过设置不同的语境标志,将与这个语境标志相关的信息表文件与编译器绑定.这样,编译信息就可以按照用户喜欢的语言和习惯来表达了.

4 兼容性问题

在测试中我们发现,用JDK1.0编译含有本地字符的Java程序所生成的类文件用高版本的JDK的虚拟机执行会出现乱码;反之亦然,也即存在着类文件的兼容性问题.为了解决兼容性问题,我们首先分析一下JDK1.0和JDK1.1的字符编码转换的特点,以汉字"中"为例,见表2.

Table 2 Java code encodings' transformation during compiling

表2 编译过程中的字符编码转换

	GBK	Unicode	UTF-8
JDK1.0	11010110,11010000	00000000,11010110	11000011,10010110
		00000000,11010000	11000011,10010000
JDK1.1	11010110,11010000	01001110,00101101	11100100,10111000,10101101

从表2可以看出,在JDK1.0中,一个两字节的汉字字符被当做两个单字节字符处理,被转换为两个Unicode字符编码,在生成类文件时,被转换成两个双字节的UTF-8格式.可见,在JDK1.0中,除了127个ASCII字符是以一个字节的UTF-8格式(0xxxxxxx)表示外,汉字是以两个双字节的UTF-8格式(110xxxxx,10xxxxxx)表示,不会出现3个字节的UTF-8格式.而在JDK1.1中,一个两字节的汉字字符的GBK编码首先被转换为相应的汉字字符的Unicode编码,然后表示成3字节的UTF-8格式(1110xxxx,10xxxxxx,10xxxxxx).所以在JVM中作编码转换时,如果读入UTF-8格式以"110"开头,则按ASCII字符的转换方式处理,即将UTF-8格式转换成的Unicode编码作为字符的编码,只是汉字字符的编码是由两个连续的Unicode编码构成的;如果读入的UTF-8格式以"1110"开头,则UTF-8格式转换生成的Unicode编码还要转换成字符的本地编码,也即对采用不同转换机制的编译器编译生成的类文件,通过判断采用不同的逆转换,从而解决了兼容性问题.

5 含有本地字符的Java程序的跨平台

Java语言最大的特点是跨平台性.Java应用程序的字节码可以在具有Java虚拟机的不同操作系统、不同硬件平台上执行[1],但是,如果程序中含有本地字符,例如,在显示的字符串中含有汉字,就会影响程序的跨平台性.因为不同的操作系统采用的字符集可能不同,有的系统上甚至没有汉字字符集,这样,程序运行时就无法显示汉字字符.一般情况下,程序中的汉字都不多,重新安装汉字平台不但浪费空间,而且意义也不大.那么如何解决这个问题呢?我们可以通过构造小字库来解决,即将程序中要出现的本地字符写入一个小字库文件中,而程序中出现的本地字符都从小字库中读取,在程序移植时,连同小字库一起移植,这样,无论在什么环境下(像西文环

境),程序中的本地字符仍然可以正确显示,而且又避免了安装合适的本地字符平台的麻烦,同时也节省了大量的空间(完整的字库占用的空间很大),提高了本地字符的显示速度.

6 测试结果

例 1:

```
class 中文{
    /*声明一个字符串域*/
    static String 字符串="中文字符串";
    static void 方法(){
        char 字符='好';
        System.out.println(字符串+字符);}
    public static void main (String argv[]) {
        for (int 计数=0;计数<60;计数++){
        方法();    }   }  }
```

例 1 的 Java 源程序(中文.java)含有中文字符、字符串、注释、标识符(类名、方法名、域名、局部变量名),采用该程序对我们所实现的 Java 编译器和 Java 虚拟机进行测试.我们所实现的编译器成功地编译了该程序,生成了类文件:中文.class.生成的该类文件在我们所实现的虚拟机上也运行成功.

当然,本编译系统也支持由其他语种的字符构成的注释、字符、字符串、标识符.本编译系统还实现了多语种的提示信息显示方式.

另外,采用小字库的方式使程序在西文环境下也可以正确显示汉字.我们所实现的虚拟机也很好地解决了含有本地字符的类文件的兼容性问题,无论是用什么版本的编译器编译生成的类文件,都能在我们所实现的虚拟机上正确运行.

我们采用例 1 对所实现的编译系统 COMPILER(Java 编译器和 Java 虚拟机)与 SUN 公司的编译系统进行了测试,测试结果见表 3(表中的标识符是不包括类名、接口名的标识符).

Table 3 The result

表 3 测试结果

	JDK1.0		JDK1.1		JDK1.2		COMPILER①	
	javac	JVM②	javac	JVM	javac	JVM	Compiler	JVM
Chinese commentary③	√	√	√	√	√	√	√	√
Chinese character④	×	×	√	√	√	√	√	√
Chinese string⑤	√	√	√	√	√	√	√	√
Chinese identifier⑥	×	×	√	√	√	√	√	√
Chinese class/interface name⑦	×	×	√	×	√	×	√	√
Prompted message⑧	E	E	E	E	E	E	M	M

"√"——Surport⑨,"×"——Not surport⑩,"E"——English display⑪,"M"——Multi-Language infomation display⑫

①编译器,②Java 虚拟机,③中文注释,④中文字符,⑤中文字符串,⑥中文标识符,
⑦中文类名/接口名,⑧提示信息,⑨支持,⑩不支持,⑪英文显示,⑫多语种信息显示.

从表 3 可以看出,采用我们的国际化方案能实现彻底的国际化支持.

References:

[1] Ji, Zhen-yan, Cheng, Hu. Chinese Java. In: Huang, Chang-ning ed. Proceedings of the 1998 International Conference on Chinese Information Processing (ICCIP'98). Beijing: Tsinghua University Press, 1998. 524~529 (in Chinese).

[2] Gosling, J., Joy, B., Steele, G. The Java Language Specification. Reading, MA: Addision-Wesley Publishing Comany, 1996.

[3] Lindholm, T., Yellin, F. The Java Virtual Machine Specification. Reading, MA: Addision-Wesley Publishing Comany, 1996.

[4] Appel, A. W. Modern Compiler Implementation in Java: Basic Techniques. Cambridge, CA: The Press Syndicate of the

University of Cambridge, 1997.

[5] Louden, K.C. Compiler Construction: Principles and Practice. California: PWS Publishing Company, 1996.

[6] The Unicode Standard. http://www.unicode.org.

附中文参考文献:

[1] 冀振燕,程虎. Java 语言的汉化. 见:黄昌宁编. 1998 年中文信息处理国际会议论文集(ICCIP'98). 北京:清华大学出版社, 1998. 524～529.

Design and Implementation of Java's Internationalization

JI Zhen-yan, CHENG Hu, MEI Jia

(*Institute of Software, The Chinese Academy of Sciences, Beijing* 100080, *China*)

E-mail: chenghu@126.com

http://www.ios.ac.cn

Received May 30, 1999; accepted September 10, 1999

Abstract: The disadvantages of localization are pointed out in this paper, Java's intel encoding is also analyzed profoundly, and the compiler supporting internationalization is designed and implemented. With the character encoding transformation scheme, international characters and strings have the same operations as the English characters and strings, the storage spaces of class files are saved, and the applications have conditional encoding-independence and the problem of many Chinese character sets is solved to some extent. Separating compiler messages from codes independent of locale context supports the implementation of international information. Small font libraries are adopted to implement the real platform-independence of applications including local characters, the problem JKD (Java development kit) has not solved, and the characters' encoding transformation is performed during loading a class file. To solve the compatibility problem between different JDK versions, the different encoding transformation schemes are chosen according to the character's encoding characteristic. The test results prove that the internationalization scheme designed in this paper can solve the internationalization problems thoroughly.

Key words: Java programming language; internationalization