Java网络传输中字符编码问题的研究

包竹苇,李 淼,张 建
BAO Zhu-wei,LI Miao,ZHANG Jian

中国科学院 合肥智能机械研究所,合肥 230031
Hefei Institute of Intelligent Machines,Chinese Academy of Sciences,Hefei 230031,China
E-mail:robertb9527@gmail.com

BAO Zhu-wei,LI Miao,ZHANG Jian.Research of character encoding in Java net transfers.Computer Engineering and Applications,2007,43(4):93-95.

Abstract: Aimimg at the problem of Chinese character encoding/decoding incorrectly in Java communications between client and server,this article analyzes the manifestation of various character list coding and Servlet default character set from the operating system inner part,the Java Virtual Machine and net transfers process.It puts forward resolving the character encoding problem in Java net transfers,and had used in DET6.0 NetRegister and NetReasoning system effectively.

Key words: Java virtual machine;net transfers;Chinese character encoding;DET

摘 要:针对Java客户端与服务器端通信,客户端浏览器显示等中文字符编码常常出现编解码错误、显示乱码的问题,从操作系统内部,Java虚拟机和网络传输过程进行研究,分析各种字符编码的表现形式,服务器端Servlet缺省字符集等,找出了问题的根源;提出在Java网络传输中中文字符编码问题的解决方法,并有效地应用于农业专家系统开发平台(DET6.0)的网络发布和知识推理中。

关键词:Java虚拟机;网络传输;中文编码;DET
文章编号:1002-8331(2007)04-0093-03 **文献标识码**:A **中图分类号**:TP311

1 引言

Java语言设计之初就考虑到了国际化的需求[1],Java平台内部的字符控制机制也采用了通用的国际标准字符集Unicode,对于世界上大多数语言都提供了较好的支撑环境,但目前ASCII、GB2312、GBK、UTF-8、ISO-8859-1等都还是网络传输中常用的编码方式,网络上的许多应用默认编码也是只考虑到了英文字符,未对中文字符提供有效的支持。在农业专家系统开发平台(DET6.0)网络注册和网络推理系统中,客户端与服务器端通信,客户端浏览器显示等中文字符编码也常常出现编解码错误、显示乱码的问题,因此本文研究在Java源代码=>Java字节码=>虚拟机=>操作系统=>网络传输中的编码转换环节,进行编/解码转换的处理,使之更好地适应中文环境。

2 字符编码的机制

英文字符最常用的编码方式是ASCII (American Standard Code for Information Interchange,美国信息交换标准代码),是基于罗马字母表的一套编码系统。它将一个字符用一个字节来表示,这样能够表达的字符数为28=16×16=256个,而英文字符再加上一些常用控制符等只用其中(\x00-\x7F)的部分就够了,所以ASCII码实际上所表达的字符数为128个。

利用大于128部分的空间的不同定义规则,就形成了对其他欧洲语言的扩展字符集ISO-8859系列,包括ISO-8859-1(西欧字符)、ISO-8859-2(中欧和东欧字符)、ISO-8859-7(希腊字符)等。它们8位编码的最高位为1,又分别扩展出了128个各自的字符。

但是对于许多亚洲语言,如中文、韩文和日文来说,用256个字符来表示所有的文字肯定是不够的,所以就用两个字节来表示一个字符。所采用的规则为如果第1个字符小于128(\x80)的仍和英文字符集编码方式保持兼容;如果第1个字符大于128(\x80)的,就当成是此字符的第1个字节,和后面紧跟的1个字节共同组成一个字符,这样可容纳的字符数就变成了128+128×256=32 896个。按照这种类似的方式有了简体中文的GB2312,繁体中文的BIG5和日文的SJIS字符集等。这些从ASCII扩展的编码方式,英文部分都是兼容的,但扩展部分的编码由于采用不同的方式制定,它们是不兼容的,虽然很多字在3种体系中写法一致(例如"中文"这两个字),但在相应字符集中的坐标不一致,所以GB2312编码的字符用BIG5看就全是乱码了。另外在浏览其他非英语国家的页面时(比如包含有德语的人名时)经常出现奇怪的汉字,就是由扩展位的编码冲突造成的。

对于中文编码来说,GB2312字符集中包括6千多个常用汉字,但还是达不到包括全部汉字的要求,因此后来又制定了

基金项目:国家高技术研究发展计划(863)(the National High-Tech Research and Development Plan of China under Grant No.2003AA118040)。
作者简介:包竹苇(1979-),男,硕士研究生,研究方向:专家系统,模糊容错;李淼,研究员;张建,研究员。

GBK 以及 GB18030 标准,GBK 字符集是 GB2312 的扩展(K),GBK 里大约有 2 万 9 千多个字符,除了保持和 GB2312 兼容外,繁体中文字,甚至连日文的假名字符也能显示。而 GB18030[2] 则是一个更复杂的字符集,采用变长字节的编码方式,能够支持更多的字符。GBK 以及 GB18030 标准向下与 GB2312 编码兼容,向上支持 ISO10646 国际标准。

国际标准 ISO-10646 定义了通用字符集(Universal Character Set,UCS)。目前只分配了前 65 534 个码位(0x0000 到 0xFFFD),它是所有其他字符集标准的一个超集,它保证与其他字符集是双向兼容的,就是说,如果将任何文本字符串翻译到 UCS 格式,然后再翻译回原编码,不会丢失任何信息。UCS 包含了用于表达所有已知语言的字符,对于还没有加入的语言,由于正在研究怎样在计算机中最好地编码它们,因而最终它们都将被加入。UCS 还包括大量的图形的、印刷用的、数学用的和科学用的符号,包括所有由 TeX、Postscript、MS-DOS、MS-Windows、Macintosh、OCR 字体,以及许多其他字处理和出版系统提供的字符。UCS 里有些编码点分配给了组合字符。单个的组合字符不是一个完整的字符,它是一个类似于重音符或其他指示标记,加在前一个字符后面。这种机制在科学符号中特别有用,比如数学方程式和国际音标字母,可能会需要在一个基本字符后组合上一个或多个指示标记。但是不是所有的系统都需要支持象组合字符这样的 UCS 里所有的先进机制。因此 ISO10646 指定了 3 种实现级别,其中最高级别 3 支持所有的 UCS 字符,可以在任意一个字符上加上一个 tilde(颚化符号,西班牙语字母上面的~)或一个箭头(或两者都加)。

Unicode 协会公布的 Unicode 标准严密地包含了 ISO-10646 实现级别 3 的基本多语言面。在两个标准里所有的字符都在相同的位置并且有相同的名字,Unicode 与 ISO10646 标准码表相兼容。另外 Unicode 标准额外定义了许多与字符有关的语义符号学,Unicode 详细说明了绘制某些语言(比如阿拉伯语)表达形式的算法,处理双向文字(比如拉丁与希伯来文混合文字)的算法和排序与字符串比较所需的算法等。Unicode 已经成为字符编码最通用的国际标准,Java 平台内部的字符控制机制也是基于 Unicode 的[3]。

但是互联网上 70%以上的信息都是英文的,如果网络传输都用 Unicode,每个英文字符都要用两个字节来表示,对于空间和网络流量都是一种浪费。因此通常采用另一种编码方式 UTF-8(Unicode Transformation Form 8-bit form),它能有效地解决这一问题,将 UNICODE 的 2 字节字符用变长个(1~3 个字节)表示:对英文,仍然和 ASCII 一样用 1 个字节表示,这个字节的值小于 128(\x80);对其他语言的用一个值位于 128~256 之间的字节开始,再加后面紧跟的 2 个字节表示,共使用 3 个字节。这样在程序处理过程中所有字符用 16 位编码表示(双字节),但在存取转换成字节流时使用 UTF-8 格式转换,对于英文字符来说和原来用 ASCII 方式存取时相比大小仍然是一样的,而对中文来说和原来的 GB2312 编码方式相比,大小为:(3 字节/2 字节)=1.5 倍。

3 操作系统中的字符编码问题

对于 Java 平台[4]来说,在 Java 源代码->Java 字节码->虚拟机->操作系统->网络传输中都有编码转换环节,需要进行编/解码转换的处理,下面先从操作系统方面来考察。

JVM[5](Java Virtual Machine,Java 虚拟机)架设在操作系统之上,屏蔽了底层操作系统的不同,实现其跨平台性,它支持不同的 Locale,可以在 Java 程序中设置不同的 Locale 或是调节操作系统所使用的 Locale。下面通过程序段 List.java 列出 JVM 的默认属性,并在重新设置 Locale 后输出新的属性。

List.java

```
import java.util.*;
import java.text.*;
public class List {
public static void main(String[] args){
Locale list[] = DateFormat.getAvailableLocales();
System.out.println("系统全部可用 Locale:");
for(int i=0;i < list.length;i++){
System.out.println(list[i].toString()+":"+list[i].getDisplayName());
}
System.out.println("系统属性:");
System.getProperties().list(System.out);
}
}
```

结果:

表 1 编码测试结果

	file.encoding	user.language	user.region
GNU/Linux2.4.x (LANG=en_US LC_ALL=en_US)	ISO-8859-1	en	US
GNU/Linux2.4.x (LANG=zh_CN LC_ALL=zh_CN.GBK)	GBK	zh	CN
Windows(Locale:en_US English)	ISO-8859-1	en	US
Windows(Locale: zh_CN 中文)	GBK	zh	CN

可见在 GNU/Linux 2.4.x 以及 Windows 系统中,不管是中英文系统,JVM 的缺省编码方式由系统的"本地语言环境"设置确定,和操作系统的类型无关。

考察"Java 网络传输"这个字符串(9 个字符),在 file.encoding 设置为编码方式 ISO-8859-1 时,内部编码为:

```
string=Java ???? length=9
char[0]='J' byte=74 \u4A   BASIC_LATIN
char[1]='a' byte=97 \u61   BASIC_LATIN
char[2]='v' byte=118 \u76  BASIC_LATIN
char[3]='a' byte=97 \u61   BASIC_LATIN
char[4]=' ' byte=32 \u20   BASIC_LATIN
char[5]='?' byte=63 \u3F   BASIC_LATIN
char[6]='?' byte=63 \u3F   BASIC_LATIN
char[7]='?' byte=63 \u3F   BASIC_LATIN
char[8]='?' byte=63 \u3F   BASIC_LATIN
```

"网络传输"这四个中文字符全部表示成了'?',编码为 char(63),信息丢失。

设置为 GBK 时,内部编码为:

```
string=Java 网络传输 length=9
char[0]='J' byte=74 \u4A   BASIC_LATIN
char[1]='a' byte=97 \u61   BASIC_LATIN
char[2]='v' byte=118 \u76  BASIC_LATIN
char[3]='a' byte=97 \u61   BASIC_LATIN
char[4]=' ' byte=32 \u20   BASIC_LATIN
char[5]='网' byte=81 \u51  CJK
char[6]='络' byte=-36      CJK
char[7]='传' byte=32 \u20  CJK
```

char[8]='输' byte=-109 CJK

再依次使用各种编码方式进行实验，可以得知Java平台内部是按照字节流=>字符流=>字节流方式进行处理，在Java字节流到字符流和字符流到字节流都是含有隐含的解码处理的(缺省是按照系统缺省编码方式,可以指定解码方式)。最早的字节流解码过程从Javac的代码编译就开始了,进行编解码时,在不同的步骤按照需要指定字符集。

4 网络传输中字符编码问题

在Java网络传输中,字符仍然要分解为字节,在服务器和客户端之间传递的是字节流。从一个中文客户端的浏览器表单中提交"网络传输"这4个中文字到服务器,首先浏览器按照GBK方式将字符编码成字节流CD F8 C2 E7 B4 AB CA E4,然后这8个字节按照URLEncoding的规范转成:%CD%F8%C2%E7%B4%AB%CA%E4。

服务器端的Servlet缺省字符集为ISO-8859-1，如果不指定编码方式的话,通过WEB提交时的输入ServletRequest和输出时的ServletResponse缺省都是ISO-8859-1方式编解码的。因此,即使服务器操作系统的语言环境是中文,上面输入的请求仍然按英文解码成8个UNICODE字符,输出时仍按照英文再编码成8个字节,并按照ISO-8859-1方式编解码,显示结果就出现了乱码:

ServletRequest's Charset Encoding=null
ServletResponse's Charset Encoding=ISO-8859-1
char[0]=? \uCD LATIN_1_SUPPLEMENT
char[1]=? \uF8 LATIN_1_SUPPLEMENT
char[2]=? \uC2 LATIN_1_SUPPLEMENT
char[3]=? \uE7 LATIN_1_SUPPLEMENT
char[4]=? \uB4 LATIN_1_SUPPLEMENT
char[5]=? \uAB LATIN_1_SUPPLEMENT
char[6]=? \uCA LATIN_1_SUPPLEMENT
char[7]=? \uE4 LATIN_1_SUPPLEMENT

可以修改浏览器传输信息的头部分中的"Accept-Language",设置为"zh-cn",使请求的解码方式和输出的字符集编码方式使用GBK，令ServletRequest和ServletResponse用相应语言的编码方式进行输入解码/编码:

```
clientLanguage= req.getHeader("Accept-Language");
if(clientLanguage.equals("zh-cn")){
req.setCharacterEncoding("GBK");
res.setContentType("text/html;charset=GBK");
}
```

这种情况下输出为:

'网络传输' length=4
ServletRequest's Charset Encoding = GBK
ServletResponse's Charset Encoding = GBK
char[0]='网' byte=81 \u51 CJK
char[1]='络' byte=-36 CJK
char[2]='传' byte=32 \u20 CJK
char[3]='输' byte=-109 CJK

能够正确地显示传输后的中文字符。

而实际上造成这个问题的根源就是Servlet标准制定方将ServletRequest和ServletResponse的编码/解码方式设置成了固定的字符集ISO-8859-1,没有考虑到程序国际化的需要。一个按照国际化规范设计的Web应用中，应该能够基于浏览器的本地语言环境自动进行编/解码的适应，采用以下方法让Servlet进行自动判断浏览器类型:

```
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException {
//从HTTP请求的头信息中获取客户端的语言设置
String clientLanguage = req.getHeader("Accept-Language");
//简体中文
if(clientLanguage.equals("zh-cn")){
req.setCharacterEncoding("GBK");
res.setContentType("text/html;charset=GBK");
}
//繁体中文
else if(clientLanguage.equals("zh-tw")){
req.setCharacterEncoding("BIG5");
res.setContentType("text/html;charset=BIG5");
}
//日文
else if(clientLanguage.equals("jp")){
req.setCharacterEncoding("SJIS");
res.setContentType("text/html;charset=SJIS");
}
//缺省认为是英文
else {
req.setCharacterEncoding("ISO-8859-1");
res.setContentType("text/html;charset=ISO-8859-1");
}
//设置好request的解码方式和response的编码方式后，进行后续的操作。
}
```

经过中文输入-解码-中间Unicode处理-UTF-8存储-中间Unicode处理-编码-中文输出环节,最终达到中文字符经过网络传输步骤的正确显示。也可以直接找到Servlet的源代码的编码处理部分,HttpUtils中的static private String parseName,在返回前将sb(StringBuffer)转换为byte bs[],然后return new String(bs,"GB2312"),修改后自己指定编码:HashTable form=HttpUtils.parsePostData(…)。

5 结论

虽然目前网络应用中很多只是考虑到英文用户的需要,并没有对中文字符做出良好的支持,但是了解了字符编码的原理以及编/解码的处理过程，就可以按照需要对Java网络传输中的字符编/解码进行控制,以得到正确的结果。目前农业专家系统开发平台(DET6.0)[6]的网络发布和知识推理中,中文字符编码的问题,通过文中的方法得到了有效的解决。

(收稿日期:2006年4月)

参考文献:

[1] 冀振燕,程虎,梅嘉.Java语言国际化的设计与实现[J].软件学报,2000,11(11):1541-1546.
[2] Leon.中文化与GB18030[EB/OL].[2001-07].http://www0.ccidnet.com/tech/os/2001/07/31/58_2811.html.
[3] 段明辉.Java编程技术中汉字问题[EB/OL].[2000-11].http://www-128.ibm.com/developerworks/cn/java/java_chinese/index.html.
[4] Gosling J,Joy B,Steele G.The Java Language Specification.Reading[M].MA:Addision-Wesley Publishing Company,1996.
[5] Lindholm T,Yellin F.The Java Virtual Machine Specification Reading[M].MA:Addision-Wesley Publishing Company,1996.
[6] 863课题"开放式农业专家系统与信息处理平台"技术报告[R].中国科学院合肥智能机械研究所,2005.