

Java 应用中的汉字乱码问题分析

刘长生, 谢 强, 丁秋林

(南京航空航天大学 计算机应用研究所, 江苏 南京 210016)

摘 要:根据 Java 应用中乱码出现的原因将问题分成了4类:由于编译不当导致的乱码、Web 应用中的乱码、数据库读写中的乱码和 I/O 读写中的乱码。在各个类别中,先给出出现乱码时的现象,然后对现象进行原因分析,再给出解决的办法。最后,根据做项目的实践经验,给出了一些解决汉字乱码问题的心得。

关键词:Java; 字符集; 中文乱码

中图分类号: TP319

文献标识码: A

文章编号: 1005-3751(2006)01-0158-04

Analysis of Chinese Character Encoding in Java Programming

LIU Chang-sheng, XIE Qiang, DING Qiu-lin

(Computer Application Institute, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

Abstract: First classify the problems into four catalogs according to different causes: incorrect compiling, communication Web application, R/W in database and R/W in I/O. After that, analyze the cause of phenomenon, and then give a solution in each catalog. Finally, come up with some conclusions which were summarized from practical project.

Key words: Java; character set; Chinese character encoding

0 前 言

现在大部分具有国际化特征的软件核心字符处理都是以 Unicode 为基础的,在软件运行时根据当时的 Locale/Lang/Codepage 设置确定相应的本地字符编码设置,并依此处理本地字符^[1]。在处理过程中需要实现 Unicode 和本地字符集的相互转换,甚或以 Unicode 为中间的两个不同本地字符集的相互转换。这种方式在网络环境下被进一步延伸,任何网络两端的字符信息也需要根据字符集的设置转换成可接受的内容^[2]。

Java 语言在内部采用 Unicode 表示字符,遵守 Unicode V2.0。Java 程序无论是从/往文件系统以字符流读/写文件,还是往 URL 连接写 HTML 信息,或从 URL 连接读取参数值,都会有字符编码的转换。这样做虽然增加了编程的复杂度,容易引起混淆,但却符合国际化的思想。

中文编码有 GB2312, GBK, BIG5, GB18030-2000 等, Jdk1.5 已经支持 GB18030-2000,如果不熟悉 Java 中的中文处理方式及相应规范,就很容易造成乱码问题。在很多种情况下都可能导致 Java 的中文乱码问题,可以把这些情况大致分成如下几类:由于编译不当导致的乱码、Web 应用中的乱码、数据库读写中的乱码和 I/O 读写中的乱码。下面文中对这几种情况做逐一探讨。

收稿日期:2005-04-11

作者简介:刘长生(1979—),男,江苏丰县人,硕士研究生,研究方向为企业信息化和 J2EE 应用;丁秋林,教授,博士生导师,研究方向为企业信息化与系统集成。

1 由于编译不当导致的乱码

1.1 现 象

下例是一个类的代码片断:

```
public void encodingTest(){
    String str="你";
    System.out.println(str);
}
```

上例在中文平台上缺省编译(-Encoding GB2312),生成 ZhClass,在英文平台上缺省编译(-Encoding ISO8859-1),生成 EnClass。ZhClass 在中文平台上执行可以正确显示汉字“你”,但是在英文平台上会出现乱码“?”。EnClass 在英文平台上执行可以正确显示汉字“你”,但是在中文平台上会出现乱码“??”。

1.2 原因分析

编译 Java 源程序要调用 Javac,如果没有指定 Encoding,则按照系统的默认 Encoding。中文平台上是 GB2312,英文平台上是 ISO8859-1。Java 的编译器实际上是调用 sun.tools.Javac.Main 这个类对源文件进行编译,这个类的 compile 函数中间有一个 encoding 的变量,-encoding 的参数其实直接传给 encoding 变量。编译器就是根据这个变量来读取 Java 文件的,然后用 UTF-8 形式编译成 class 文件^[3]。

如果用 GB2312 编译,在生成的 class 文件中会有 E4 BD A0 的字段,如果用 ISO8859-1 编译,则会在生成的 class 文件中找到 C1 84 C3 A3 的字段。这是因为 UTF-8 和 Unicode 有如下对应关系:

(1)对于 7 位的 Unicode(单字节,高位为 0),在其首位加 0 即可,即:

0 - - - - -

(2)对于 11 位的 Unicode(两个字节,高位字节高五位为 0),在其首位前加 110,在第六位和第七位之间添加 10,即:

1 1 0 - - - - 1 0 - - - - -

(3)对于 16 位的 Unicode(两个字节,高位字节高五位中有的不为 0),在其首位前加 111 0,在第六位和第七位之间添加 10,在十二和十三位之间添加 10,即:

1 1 1 0 - - - - 1 0 - - - - - 1 0 - - - - -

如果用 GB2312 进行编译,编译器首先以 GB2312 对读进来的字节进行解码,字符串“你”的 GB2312 码为 0xC4E3,然后转变成相应的 Unicode,字符串“你”的 Unicode 为 0x4F60。0x4F60 的二进制是 0 1 0 0 1 1 1 1 - - 0 1 1 0 0 0 0,根据规则用 UTF-8 补齐,变成:11100100 - - 10111101 - - 10100000(E4 - - BD - - A0),所以在生成的 class 文件中会有 E4,BD,A0 字段。其过程如图 1 所示。

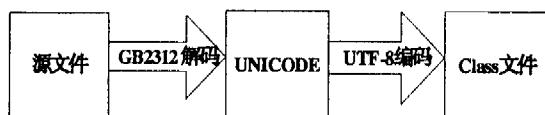


图 1 在中文平台编译流程

同样的道理,如果用 ISO8859-1 进行编译,编译器首先以 ISO8859-1 对读进来的字节进行解码,其中字符串“你”的原始编码为 0xC4E3,然后转变成相应的 Unicode,其中字符串“你”的 Unicode 为 0x00C4,0x00E3(ISO8859 系列字符集是单字节字符集,所以映射到 Unicode 要高字节补 0)。0x00C4 的二进制是 00000000 - - 11000100,因为每个字符都大于 7 位,因此用 11 位编码,根据规则用 UTF-8 补齐,变成 11000011 - - 10000100(C3 - - 84)。同理类推,0x00E3 会变成 11000011 - - 10100011(C3 - - A3),所以在生成的 class 文件中会存在 C3,84,C3,A3 字段,其过程如图 2 所示。

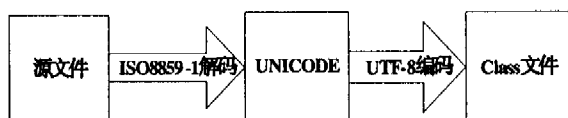


图 2 在英文平台编译流程

在中文平台上缺省编译后,其实 str 在运行态的 char[] 是 0x4F60,ZhClass 在中文平台上运行,系统的缺省编码是 GB2312,因此 CharToByteConverter(在 sun.io 的包中)会自动调用 GB2312 的 converter,把 str 转化成 0xC4E3,所以系统打印出“你”。但是如果 ZhClass 是在英文平台下运行,CharToByteConverter 的缺省值是 ISO8859-1,系统会自动调用 ISO8859-1 去转化 str,但是它无法解释,这时系统把它转变成 0x3F(?) 的 ISO8859-1 的编码),因此系统会输出“?”,出现了乱码。在英文平台上缺省编译后,其实 str 在运行态的 char[] 是 0x00C4

0x00E3,EnClass 在中文平台上运行,中文系统无法识别 0x00C4 0x00E3,因此会出现乱码“??”,但是,当 EnClass 在英文平台上运行,0x00C4 被转化成 0xC4,0x00E3 被转化成 0xE3,所以可以正确显示汉字“你”(其实每次显示“半”个汉字)。

1.3 解决办法

通过以上分析可以总结出对于此类问题的解决办法是:对于放在操作系统中的 Java 源程序,在编译时,要指定它内容的编码格式,具体来说用 - encoding 来指定。如果源程序中含有中文字符,而用 - encoding 指定为其它的编码字符,显然是要出错的。用 - encoding 指定源文件的编码方式为 GBK 或 GB2312,无论在什么系统上编译含有中文字符的 Java 源程序都不会有问题,它都会正确地将中文转化为 UTF-8 存储在类文件中^[3]。

2 Web 应用中的乱码问题

2.1 现象

从一个 JSP 页面输入的中文被处理后返回的结果却是一些问号或乱码。

2.2 原因分析

图 3 说明了从 JSP 源文件到客户端的流程。

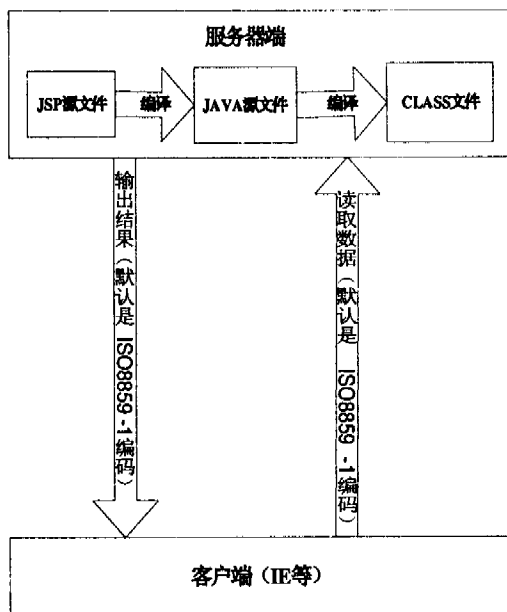


图 3 Web 服务器处理示意图

在这一过程中有字符编码转换的地方包括:

1)JSP 编译:Java 应用服务器将根据 JVM 的 file.encoding 值读取 JSP 源文件,编译生成 Java 源文件,再根据 file.encoding 值写回文件系统。

2)Java 源文件被编译为 class 文件:这个过程与一般的源程序编译成 class 文件相同。从这里开始 Servlet 和 JSP 的运行就类似了,只不过 Servlet 的编译不是自动进行的。对于 JSP 程序,对产生的 Java 中间文件的编译是自动进行的(在程序中直接调用 sun.tools.Javac.Main 类)。

3)客户端读取服务器段的输出和服务器端读客户端

的输入:Servlet 需要将 HTML 页面内容转换为 browser 可接受的 encoding(用 content - type 指定)内容发送出去。在目前的 Servlet 的规范中,如果不指定的话,通过 Web 提交时输入的 ServletRequest 和输出时的 ServletResponse 缺省都是以 ISO8859-1 进行编码/解码的(这里的编码/解码方式是和操作系统环境中的语言环境是无关的)。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,虽然这样在浏览器端如果设置是中文能够正确显示,但实际上读写的是“字节”。如下例:从一个中文客户端的浏览器表单中提交“世界你好”这 4 个中文字到服务器时,首先浏览器按照 GBK 方式编码成字节流 CA C0 BD E7 C4 E3 BA C3,然后 8 个字节按照 URLEncoding 的规范转成 %CA% C0% BD% E7% C4% E3%BA% C3,正是因为传入的是 GBK/GB2312 编码,而服务器却以 ISO8859-1 处理,因此才出现结果的乱码。

2.3 解决办法

在 JSP 编译时,如果当前系统语言支持 GBK,那么这时候不会出现 encoding 问题。如果是英文的系统,则要将 JVM 的 file. encoding 值置成 GBK。系统语言如果是 GB2312,则根据需要,确定要不要设置 file. encoding,将 file. encoding 设为 GBK 可以解决潜在的 GBK 字符乱码问题。

在客户端读取服务器端的输出和服务端读客户端的输入时,解决这种乱码的正确的方式是应该根据客户端浏览器设置 ServletRequest 和 ServletResponse,用相应语言的编码方式进行输入解码/输入编码,例如,当根据浏览器的头信息中的“Accept - Language”为 zh - cn(中文)时,设置请求的解码方式和输出的字符集编码方式使用 GBK:

```
String clientLan=request.getHeader("Accept - Language");
```

```
if ( clientLan.equals("zh - cn") ) {
```

```
    request.setCharacterEncoding("GBK");
```

```
    response.setContentType("text/html;charset = GBK");
```

```
}
```

另外,在 Servlet 的源代码中尽量不要有中文:因为在 MVC(Model View Controller)模式中,Servlet 主要是控制器(C)的角色,因此,应该通过 ResourceBundle 机制由 Servlet 控制转向到相应的显示器(JSP 或者 XSLT)中,所以应该将与本地界面语言相关的界面显示的部分从 Servlet 和后台的模块中完全剥离出来,放到相应的 ResourceBundle 文件中或者 XSLT 文件中。这样源程序里完全是英文,编译时完全不需要考虑字符集的问题。如果 Servlet 实在需要包含中文,则需要设置应用服务器的 Javac 编译选项,加上 - encoding 选项成系统缺省的字符集,如果把用中文编写的字符按照英文方式解码编译,然后再按照英文方式输出,虽然结果表面正确,实际上都成了面向“字节”编程^[4]。如果 Web 应用仅限于中文用户,可在 Web 服务器的 web.xml 中进行配置来解决中文乱码

问题,比如可在 weblogic 的 web.xml 中加入如下配置:

```
<context-param>
```

```
<param-name> weblogic. httpd. inputCharset. /* </param-name>
```

```
<param-value>GBK</param-value>
```

```
</context-param>
```

或者在 weblogic.xml 里加上如下配置:

```
<charset-params>
```

```
<input-charset>
```

```
<resource-path> /* </resource-path>
```

```
<Java-charset-name>GBK</Java-charset-name>
```

```
</input-charset>
```

```
</charset-params>
```

3 数据库应用中的乱码问题

数据库中的乱码问题和 Web 应用中的乱码问题类似,都是因为传入的是 GBK/GB2312 编码,而对方却以 ISO8859-1 处理。对于这类问题的解决办法是:

1)在 Java 程序这边指定输出编码是 GBK/GB2312。

2)在数据库方面设置指定传入/输出的编码是 GBK/GB2312。如:在 mysql 中,可以在配置文件 my. ini 中加入以下语句实现。

在[mysqld]区增加:

```
default-character-set=GBK
```

并增加:

```
[client]
```

```
default-character-set=GBK
```

在 SQL Server 中,可以将数据库默认的语言设置为 Simplified Chinese 来达到目的。

4 I/O 读写中的乱码问题

对于这种情况,建议在程序编写时,如果需要从用户端接收用户的可能含有中文的输入或含有中文的输出,程序中应该采用字符流来处理输入和输出,如下例:

```
import java.io. * ;
```

```
public class Read {
```

```
    public static void main(String[] args) throws IOException {
```

```
        String str= "\n 测试,这是内部中文字符串" + "\n this is english character";
```

```
        String strin= "";
```

```
        InputStreamReader isr = new InputStreamReader(System.in, "GB2312");
```

```
        OutputStreamWriter osw = new OutputStreamWriter(System.out, "GB2312");
```

```
        BufferedReader stdin = new BufferedReader(isr); //设置输入接口按中文编码
```

```
        BufferedWriter stdout = new BufferedWriter(osw); //设置输出接口按中文编码
```

```
        stdout.write("请输入中文:");
```

```
        stdout.flush();
```

```

    strin=stdin.readLine();
    stdout.write("您输入的中文是:"+strin);
    stdout.write(str);
    stdout.flush();
}

```

同时,在编译程序时,用以下方式来进行:

javac -encoding GB2312 Read.java

结果如图 4 所示。

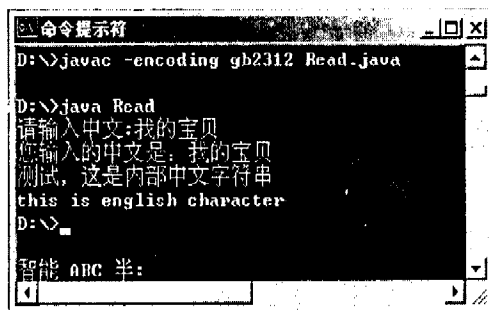


图 4 I/O 读写测试结果

5 结束语

由以上讨论可了解到:当 Java 编译器读源文件时,默认按 file.encoding 属性进行。但是生成的 class 文件,无论在何种平台下都用 UTF-8 进行存储。在 Java 读 class 文件时,一律用 UTF-8 进行。在 Java 字节流到字符流(或者反之)都是用含有隐含的解码处理的(缺省是按照系统

缺省编码方式),最早的字节流解码过程从 javac 的代码编译就开始了。Java 中的字符 character 存储单位是双字节的 Unicode^[1],而英文字符无论怎么转换都不会失真,因为在不同的字符集里这一部分是兼容的。归根到底,Java 中的中文乱码问题的产生是由于向系统里传入了错误的 encoding 参数。

总之,Java 的创始者(Java Soft)已经考虑到 Java 编程语言对多国字符的支持,只是现在的解决方案有很多缺陷在里面,需要额外付诸一些补偿性的措施^[5]。而世界标准化组织也在努力把人类所有的文字统一在一种编码之中,其中一种方案是 ISO10646,它用 4 个字节来表示一个字符。当然,在这种方案未被采用之前,还是希望 Java Soft 能够严格地测试它的产品,为用户带来更多的方便。

参考文献:

- [1] Sun White Paper. Internationalization[EB/OL]. <http://java.sun.com>,2001.
- [2] Eckel B. Thinking in Java(Second Edition)[M]. 北京:机械工业出版社,2002.
- [3] Sundy F. Java 中文问题详解[EB/OL]. <http://www.99net.net>,2003.
- [4] 车 东. Java 中文处理学习笔记[EB/OL]. <http://www.chedong.com>,2005-03-16.
- [5] 文 心. Jsp 乱码分析[EB/OL]. <http://www.zeroyit.com>,2004.

(上接第 136 页)

库表二执行 SQL. Clear 清空;使用 SQL. Add('Select * from Tk2picture where id = :id')语句和 Parameters. ParamByName('id').value = ADOQuery1. FieldByName('id').asInteger 语句定位;使用 open 语句打开图库表二;使用 TBlobField(ADOQuery2. FieldByName('tu')). SaveToFile(Ftu) 语句从图库表二中查询出相应图片并存储成图文件 Ftu。然后将图库表一的有关记录字段和图库表二检索出并存储为 Ftu 的图文件传递给列表视图 ListView1 组件所设项目。依此循环,直到查完所有符合该图片类别的记录,将查询结果放入列表视图 ListView1 组件框内供显示调用^[4]。

2.2 按双结构形式设计网络图库获得的效果

按照双结构形式设计方法建成的网络图库,经调试运行,使用者确实感到图库通过网络打开运行时速度比原来得快多,用户在终端屏幕前等待的时间大大缩短。在图片经过优化处理,入库图片达到 4258 幅后,在网速大致相同的环境下做了测试:图库通过网络打开时用户在屏幕前等待约 2 秒时间,完全达到了实用要求,使用者感到满意。

3 结 论

一般情况下采用单结构形式设计网络图库比较合适,

即将网络图库的各个字段和图作为一条记录结构,数据字段之间不产生任何冗余。而对于较大型的、要求保存图片质量比较高、存储字节比较多、调用方式多变、调用比较频繁的网络图库,采用双结构形式设计网络图库比较适合,即将图库记录中所有文字、数据、时间等类通过网络调用速度比较快的信息字段作为一种结构,而将图 ID 字段和通过网络调用速度比较慢的图信息字段作为另一种结构,两种结构之间通过图片 ID 建立对应联系。这样一来,通过编程和使用技巧,可以大大提高通过网络调用图片的速度。

参考文献:

- [1] 王 定,陈 波. Internet 简明教程[M]. 北京:清华大学出版社,2005.
- [2] 张春林,马成勇,刘 均. Delphi7 数据库系统设计与开发[M]. 北京:清华大学出版社,2003.
- [3] 求是科技. SQL Server 2000 数据库管理与开发技术大全[M]. 北京:人民邮电出版社,2004.
- [4] 张立科. Delphi7 组件编程参考手册[M]. 北京:人民邮电出版社,2003.
- [5] 刘 艺. Delphi 模式编程[M]. 北京:机械工业出版社,2004.