

Java 中文处理研究

曹莉, 赵文静

(西安建筑科技大学 信息与控制工程学院, 陕西 西安 710055)

摘要:在简要介绍字符编码常识的基础上,深入分析了 Java 源程序在编译运行过程中的编码解码过程,从而找到了能直接在控制台上运行的 Java 类, JSP, Servlet 及 Java 程序和数据库之间中文乱码问题产生的根本原因,并分别给出了解决方法,从而较好地解决了 Java 程序在移植及输入输出过程中的乱码问题。

关键词:中文字符转码; Java; JSP; Servlet; 数据库

中图分类号:TP391.1;H127

文献标识码:A

文章编号:1673-629X(2006)05-0100-03

Study on Chinese Problem in Java

CAO Li, ZHAO Wen-jing

(Sch. of Info. and Control Eng., Xi'an Univ. of Architecture & Techn., Xi'an 710055, China)

Abstract:On the basis of introducing the general knowledge of character coding, this paper deeply analyzed the coding and decoding process in compilation and execution of the Java programs. It found the root cause from which the disordered code problem in Chinese come out, involving classes run on the console, JSP, Servlet and the accessing between Java program and database, then it gave some solutions for each instance. So the Java program can display Chinese characters functionally in different operation platform during input and output.

Key words:coding and decoding of Chinese characters; Java; JSP; Servlet; database

0 引言

目前,Java 编程语言在桌面应用程序、Internet 服务器等许多领域都得到了广泛的应用。但是由于 Java 核心字符处理是基于 UNICODE 编码的,而简体中文是采用 GBK 或者 GB2312,所以在进行应用开发时经常会出现中文乱码现象。鉴于此,这里对 Java 中文问题进行系统的研究。

1 编码知识简介

字符必须被编码后才能被计算机识别和处理。不同语言使用不同的字符编码格式,例如英文使用 ISO-8859-1 编码,简体中文采用 GBK 或者 GB2312,繁体中文为 BIG5 等。而为了符合国际化的需要又提出了 UNICODE 编码。

ISO-8859-1 为单字节字符集,它包含美国 ASCII 的低位(7-bit)。GB2312 为双字节字符集,由于 GB2312 支持的汉字较少,在 GB2312 的基础上进行扩充得到了 GBK^[1],其中收录了 21886 个符号。UNICODE 是由国际组织设计的,可以容纳全世界所有的语言文字。这些编码

格式之间的兼容关系为:ISO-8859-1, GB2312 和 GBK 英文部分是兼容的,扩展部分是不兼容的。而 UNICODE 与它们都不兼容^[2]。

2 Java 中文问题的分析与解决

文中的研究环境是:Windows XP SP2, Tomcat5.5.9, JDK1.5, MySQL Server 4.1。

常见的中文乱码问题往往出现在以下几种情况中:能直接在控制台上运行的 Java 类, JSP, Servlet 类, Java 程序和数据库之间。下面将分别对这几种情况进行讨论,分析中文问题产生的原因并给出解决方法。

2.1 能直接在控制台上运行的 Java 类的中文字符处理

对于能直接在控制台上运行的 Java 类,它的整个生命周期如图 1 所示。

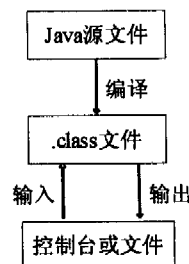


图 1 Java 类的生命周期

以简体中文 WinXP OS 为例,详细解释一下 Java 类在其整个生命周期中是如何被编码解码的,期望从中找到乱

收稿日期:2005-08-21

作者简介:曹莉(1981-),女,江苏连云港人,硕士研究生,研究方向为计算机软件开发技术;赵文静,教授,研究方向为计算机软件开发技术。

码的原因。首先编写一个含有中文字符串的 Java 源程序文件,文件保存时采用的是操作系统默认支持的 file.encoding 编码格式(在简体中文 WindowsXP 中 file.encoding 的值为 GBK)。接着用 JDK 的 javac.exe 编译该 Java 源程序,javac.exe 先将 Java 源程序从系统的 file.encoding 格式 GBK 转换为 UNICODE 格式,接着将其编译为以 UNICODE 编码的.class 文件。当运行这个.class 文件时,会产生一个 UNICODE 编码的输出(如果有输出的话),JRE 将之转换为 GBK 格式后输出到控制台或文件中。

从以上的转换过程可以看到,如果 Java 源程序是在简体中文 WinXP 中进行编译运行,不出现跨平台的情况,则只要系统中安装的是国际版的 JDK,一般情况下中文是可以正常显示的。但是经常会遇到这种情况:在简体中文 WinXP 上编写了一个 Java 源程序,然后经由网络传到另一个操作系统中运行。这时会发现所有中文字符串均不能正常显示。问题出现在 3 个环节:①源程序的编码格式与另一个操作系统中 JVM 获得的 file.encoding 格式无法匹配;②含有中文字符串的输入在编码时丢失信息;③含有中文字符串的输出直接用 file.encoding 编码,从而导致乱码。

对于第一种情况,在编译时需要通过 -encoding 明确地指出源程序的编码格式。一旦指定了 -encoding,编译器就会以所指定的编码格式为主。例如:javac -encoding gbk myProgram.java。编译器直接将 Java 源程序从 GBK 格式转换为 UNICODE 格式,而不再去理会当前所在操作系统的默认编码格式,很好地解决了移植过程中出现的乱码问题。

对于第二种和第三种情况,将之统称为 I/O 转码问题^[3]。对于这类问题,建议在程序中采用字符流来处理输入输出,也就是说 I/O 转码过程中的字节流按照指定的字符编码格式转换为字符流再进行输入输出。

```
public InputStreamReader(InputStream in, String charsetName)
    throws UnsupportedEncodingException;

public OutputStreamWriter(OutputStream out, String charsetName)
    throws UnsupportedEncodingException;
```

上面两个构造函数的第二个参数就是用来指定转码格式的。配合参数 charsetName,InputStreamReader 将输入从指定的 charsetName 编码格式转为 UNICODE 编码格式;而 OutputStreamWriter 将输出从 UNICODE 编码格式转为 charsetName 编码格式。比如,在英文 Windows 中编译运行 Java 源文件,如果需要输出含有简体中文的字符串,就必须指定输出的编码格式。方法如下:

```
FileOutputStream fos = new FileOutputStream(filename);
OutputStreamWriter writer = new OutputStreamWriter(fos,
    "GBK");
```

2.2 JSP 中的中文字符处理

运行于 Java 应用程序服务器的 JSP 主要为客户提供动态 Web 页。JSP 源文件首先被 Web 容器逐字转换为

Java 源文件,接着 JSP 编译器对其进行自动编译,产生临时的 Servlet 类文件。当有客户请求时,Web 容器就调用 JVM 来运行 class 文件,产生输出发送给浏览器。此过程如图 2 所示。

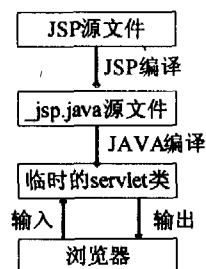


图 2 JSP 的生命周期

其中乱码问题一般出现在图中的 3 个地方:①JSP 编译过程中 JSP 源文件的转码;②JVM 对表单提交的参数的转码;③JVM 运行临时的 Servlet 类文件后产生的输出的转码。

下面逐个地解决这 3 个问题:

(1)JSP 编译过程中 JSP 源文件的转码。这个问题主要出现在跨平台情况下。如果没有在 JSP 源文件中设置它的编码格式,则 Web 容器在编译 JSP 源文件时自动获取当前所在操作系统支持的 file.encoding 格式。所以在简体中文 WinXP 中可直接用 file.encoding 格式将含有中文的 JSP 源文件转码为 UNICODE 格式,但是在其它操作系统中就会出现乱码和问号。

为了解决这个问题,可以在 JSP 源文件开头写入:

```
<%@page pageEncoding="GBK"%>
```

它告诉 JSP 编译器应该将当前的 JSP 源文件从 GBK 格式转换为 UNICODE 格式,而不是从所在操作系统的 file.encoding 格式转换为 UNICODE 格式,从而可以避免因 JSP 源文件的移植而造成的乱码问题。

(2)JVM 对表单提交的参数的转码。在 JVM 运行 class 文件时,可能需要接收来自浏览器端提交的含有中文字符串的输入。Java 在网络传输中使用的编码是 ISO-8859-1,如果没有在 JSP 源文件中指定输入的编码格式,直接从 request.getParameter() 方式获得的字符串均是 ISO-8859-1 格式的。中文字符串按照 ISO-8859-1 格式编码后直接输出,在 Web 页上看到的必然是一堆乱码。所以在用 request.getParameter() 方法获取参数之前,需要在 JSP 源文件中加入下面这句:

```
<% request.setCharacterEncoding("gbk"); %>
```

这样就指定了输入的编码格式为 GBK,JVM 将接收到的输入从 ISO-8859-1 转换为 GBK 格式后再输出,很好地解决了乱码问题。但是要注意的是这个方法只适用于表单 post 方式提交的中文参数,对 get 方式不起作用。

还有一种解决方法是每次在用 request.getParameter() 方法获取中文参数之后,用 new String(parameter.getBytes("ISO-8859-1"), "GBK") 对其进行转码^[4],这样中文输入就可以正确地显示在 Web 页上了。

以上两种方法都是通过程序语句实现的,还可以用 filter 实现表单提交的参数的转码(笔者认为这是目前为止的最佳方法)。

① post 方式的解决方法:

首先在项目中建立目录 \ WEB-INF \ classes \ filters \, 再将 tomcat 安装目录下的类文件 \ webapps \ servlets - examples \ WEB-INF \ classes \ filters \ SetCharacterEncodingFilter.class 拷到新建的目录下,最后在项目的 web.xml 中加入以下几行就可以了:

```
<filter> <filter-name>Set Character Encoding</filter-name>
<filter-class>filters.SetCharacterEncodingFilter</filter-class>
<init-param> <param-name>encoding</param-name> <param-value>GBK</param-value> </init-param> </filter>
<filter-mapping> <filter-name>Set Character Encoding</filter-name> <url-pattern>/* </url-pattern> </filter-mapping>
```

② get 方式的解决方法:

配置 tomcat 安装目录下的 \ conf \ server.xml 文件,在 <Connector> 区块中加入属性:URIEncoding="GBK".

最后应重启 tomcat 才能使得设置生效。

经过以上的设置,不论含有中文的参数是通过表单 post/set 方式提交的,还是直接从 URL 传入的,在用 request.getParameter() 方法获取参数之后,无需做任何转码就可直接输出,而不会出现乱码问题。

(3) JVM 运行 Servlet 文件后产生的输出的转码。这是程序员最经常遇到的发生乱码问题的地方。如果不指定输出的编码格式,则在将输出从 UNICODE 格式转为 ISO-8859-1 格式发送到客户浏览器端后,浏览器直接按照默认的 ISO-8859-1 格式将中文输出,这样浏览器端看到的是乱码。为了解决这个问题,一定要在源文件开头写入:

```
<% @ page contentType = "text/html; charset = gbk" %>
```

这样浏览器在接到来自服务器端的以 ISO-8859-1 格式编码的数据后就知道应该用 GBK 格式来显示(而不是 ISO-8859-1 格式)^[5],从而将含有中文的数据正确地显示在客户的浏览器上。

在一个含有中文字符串的 JSP 源文件中,只要上述的 3 个地方都进行了正确的转码,就可以避免 Web 页的中文乱码问题。关于 JSP 源文件中由于操作数据库而产生的乱码问题,下文中将会给出详细的解释。

2.3 Servlet 中的中文问题

Servlet 为客户浏览器提供 Web 页的过程如图 3 所示。

和 JSP 相似,乱码一般出现在 3 个地方,下面逐个对其进行解决:

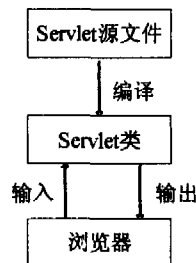


图 3 Servlet 类的生命周期

(1) Servlet 编译过程中 Servlet 源文件的转码。如果将编辑好的含有中文字符串的 Servlet 源文件移植到别的操作系统中,必须用 -encoding 参数指定 Servlet 源文件的编码格式。

(2) JVM 运行 Servlet 类时对所接收输入的转码。在接收客户浏览器端提交的含有中文字符串的参数时,与 JSP 类似,需要设置输入的编码格式为 GBK,这样才能正确地将中文参数显示出来。

(3) JVM 运行 Servlet 类后产生的输出的转码。在将输出发送给浏览器之前,必须要设定发送的内容是什么类型的,是以何种格式编码的。对于含有中文的响应,一定要在 Servlet 源文件中加上:

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

只有遵循上面的 3 步处理,客户浏览器端才能看到正常显示的中文。

2.4 Java 程序和数据库之间的中文问题

Java 编程中另一个经常出现乱码的地方是数据库中数据的读写。大部分数据库默认支持的编码格式为 ISO-8859-1,也就是说数据库是以 ISO-8859-1 格式存储数据的^[6]。当 Java 程序(包括 JSP,Servlet 等)向数据库插入含有中文的数据时,必须先将数据从 GBK 格式转为 ISO-8859-1 格式。同理,Java 程序直接从数据库中读出的数据都是 ISO-8859-1 格式的,所以需要将其转码为 GBK 格式再输出。具体做法如下所示:

```
入库: String str1 = "中国"; String str2 = new String(str1.getBytes("gbk"), "ISO-8859-1");
```

```
出库: String str3 = result.getString(1);
```

```
String str4 = new String(str3.getBytes("ISO-8859-1"), "GBK");
```

上述的方法是通过在数据库入口和出口处对含有中文的数据进行强制编码格式转换来解决乱码问题。如果需要处理的数据中含有大量的中文字段,那么对每个含有中文的字符串都必须进行强制转换,重复的体力劳动必定会让人觉得很麻烦。

其实可以将数据库默认支持的编码格式直接改为 GBK 或者 GB2312,这样就可以避免大量的重复劳动。以 MySQL 为例,这可分为两种情况:一种是将数据库的编码格式临时指定为 GBK;还有一种是修改数据库的配置文

(下转第 108 页)

储标量的 NAF 表示形式。而由于新的算法对标量编码是从最左到右进行,编码和主计算合并,因此不需要存储标量 u 和 v 的新的编码,这对于内存空间受限的设备来说尤其重要。

(3) 安全性分析。

攻击者虽然可以通过测试电量的消耗来区分点加和倍点运算,但由于新提出的算法通过采用固定模式 $0x, \dots, 0x$ 对标量进行处理,他始终得到相同的序列 $DDA \mid DDA \mid \dots \mid DDA$,其中 D 表示倍点运算, A 表示点加运算,因此他通过 SPA 攻击得不到秘密 u 和 v 。故新的算法是抗 SPA 攻击的。

3 结束语

在内存空间和计算时间负担增加不多的情况下,基于 Sharmir-NAF 提出了一个新的抗 SPA 的多点乘算法。虽然本算法是抗 SPA 的多点乘算法,是针对多点乘运算的,但通过同构等方法,本算法也同样适用于安全的点乘运算。

参考文献:

- [1] Kocher P, Jaffe J, Jun B. Introduction to Differential Power Analysis and Related Attacks[EB/OL]. URL: <http://www.cryptography.com/dpa/technical/index.html>, 1998.
- [2] Kocher P, Jaffe J, Jun B. Differential Power Analysis[A]. In Proceedings of CRYPTO'99, LNCS vol 1666[C]. [s. l.]: Springer-Verlag, 1999. 388-397.
- [3] Coron J S. Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems[A]. In Proceedings of CHES'99, LNCS vol 1717[C]. [s. l.]: Springer-Verlag, 1999. 292-302.
- [4] Montgomery P L. Speeding the Pollard and Elliptic Curve Methods for Factorizations[J]. Mathematics of Computation, 1987, 48: 243-264.
- [5] Okeya K, Takagi T, Vuillaume C. On the Exact Flexibility of the Flexible Countermeasure against Side Channel Attacks[A]. In The 9th australasian conference on information security and privacy, ACISP 2004, LNCS vol 3108[C]. [s. l.]: Springer-Verlag, 2004. 466-477.
- [6] Okeya K, Takagi T. The Width- w NAF Method Provides Small Memory and Fast Elliptic Scalar Multiplications Secure against Side Channel Attacks[J]. IEICE Transactions, 2004, E87-A: 75-84.
- [7] Lee M K. SPA-Resistant Simultaneous Scalar Multiplication[A]. IN Approaches or Methods of Security Engineering Workshop, LNCS vol 3481[C]. [s. l.]: Springer-Verlag, 2005. 314-321.

(上接第 102 页)

件。下面来具体看一下这两种方法。

(1) 将数据库的编码格式临时指定为 GBK。首先在建数据库时指定该数据库的编码格式为 GBK,在 MySQL 控制台中写:CREATE DATABASE test character set 'gbk';在 Java 程序中连接这个数据库时,指定 client 端使用的编码格式为 GBK,具体代码如下:

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost: 3306/test? user = " + " root&password = password&useUnicode = true&characterEncoding = gbk"); //连接数据库
```

(2) 修改 MySQL 配置文件。将 MySQL 安装目录中 my.ini 文件的[MySQL]和[client]两个区中的 default-character-set = latin1 都改为 default-character-set = gbk,然后重启 MySQL 服务。这样就将 MySQL 默认的编码格式改成了 GBK,可以直接存取中文数据而不会出现乱码问题。针对 Java 程序和数据库之间的中文乱码问题,笔者提出了三种解决方案,读者可以根据自己的实际情况选择最方便、效率最高的方法。

3 结束语

其实 Java 程序的中文乱码问题并没有想像的那么复

杂,经过对 Java 程序转换过程的分析,知道乱码现象存在的根本原因是含有中文字符的字符串(其编码格式为 GBK 或者 GB2312)被当作别的编码格式编码或者解码了。所以只要保证程序入口和出口的汉字信息不失真就可以解决 Java 中文乱码问题。

参考文献:

- [1] Wang Yu. Multibyte-character processing in J2EE[EB/OL]. <http://www.javaworld.com>, 2004-04-19.
- [2] Fengsundy. Java 中文问题详解,底层编码解剖[EB/OL]. <http://www.pconline.com.cn/481175.html>. 2004-10-29.
- [3] Johnlhr. Java 处理中文化问题详解[EB/OL]. <http://www.globebbs.com/bbs/showthread.php>. 2005-01-03.
- [4] 周竞涛,王明微. 关于 Java 中文问题的几条分析原则[EB/OL]. <http://www.javafan.net/article/20050330155356600.html>. 2002.
- [5] 段明辉. Java 编程技术中汉字问题的分析及解决[EB/OL]. http://www-128.ibm.com/developerworks/cn/java/java_chinese/index.html. 2000.
- [6] Monty M, Widenius, Axmark D, et al. MySQL Reference Manual[M]. [s. l.]: O'Reilly & Associates, Inc, 2002.