Java 编码中文问题研究及解决方案

Research and Resolution on Chinese Problem Based on Java

冯金辉 朱淼良 (杭州 浙江大学 计算机科学与技术学院 310027)

摘要:本文首先研究了中文问题产生的根源,然后详细分析了各种编码方式及产生中文问题的原因。接着介绍了系统的各个层次可能产生问题的表现以及原因。最后提出了解决中文问题的原则以及具体的解决方案。

关键词:编码 解码 ISO8859 Unicode UTF-8

1 引言

现在 Java 编程语言已经广泛应用于互联网世界,早在 Sun 公司开发 Java 语言的时候,就已经考虑到对非英文字符的支持了。Sun 公司公布的 Java 运行环境(JRE)本身就分英文版和国际版,其中国际版支持非英文字符。然而,因为中文字符集不只一个,而且不同的操作系统对中文字符的支持也不尽相同,所以会有许多和汉字编码处理有关的问题在我们进行应用开发中困扰着我们。有很多关于这些问题的解答,但是很多都只是针对某种特定开发环境或者特定情况的解决方法,在实际的应用过程中,需要使用者去一一尝试,才能够得到正确的解决方案,这就大大降低了使用这些解决方法的效率。本文将从汉字编码常识出发,分析 Java 中文问题,并提出解决问题的原则,使读者能够明白具体情况下应该如何处理,然后提出一系列的具体解决方案,希望对大家解决这个问题有所帮助。

2 Java 的编码方式

Java 中的字符数据是 16 位无符号型数据,它表示 Unicode 集,而不仅仅是 ASCII 集。它解决了 www 上很多的程序设计问题,比如说低成本的国际化(International),然而 Java 所处理的信息,绝大多数都是英文,对它们来说 7 位的 ACSII 码已经足够了,而 Unicode 却需要双倍的空间,所以 Java 的这种兼顾各种语言的做法既浪费了存储资源又降低了效率。而 Java 编程中中文问题的出现,归根结底是由于在 Java 的各个不同的编码方式之间进行字符转换的时候出错。因此,了解 Java 使用的各种编码方式,有助于我们解决 Java 中文问题。在 Java 中常用的编码方式有 ISO8859-1、Unicode、GB2312、GBK、UTF-8。

(1). ISO8859-1

ASCII 编码方式是单字节内码,它使用了单字节 8 位中的后 7 位,而第一位都为 0。为了应付越来越多的字符需要,如拉丁字母等,于是在 ASCII 的基础上扩展了 ISO8859-1 编码。ISO8859-1 同样是单字节内码,当 ISO8859-1 编码的字节首位为 0 时,字符可以一一映射到 ASCII 编码字符集。而当 ISO8859-1 编码的字节首位为 1 时,则可以增加 128 个字符。

(2). Unicode

Unicode 最初是双字节内码。它的出现是为了让每一个字符有唯一的表示形式。Unicode 编码是 ISO8859-1 的扩展。当 Unicode 编码的高位字节为 0 (00000000)时,低位字节等同于 ISO8859-1 编码。这时候,去除 Unicode 编码的高位字节,其编码就转化为 ISO8859-1 编码。这种方式适用于基于 ISO8859-1 编码方式的西欧语系环境。

(3). GB2312

GB2312-80 是双字节内码,它是在国内计算机汉字信息技术发展初始阶段制定的,其中包含了大部分常用的一、二级汉字,和 9 区的符号。该字符集是几乎所有的中文系统和国际化的软件都支持的中文字符集,这也是最基本的中文字符集。其编码范围是高位 0xa1-0xfe,低位也是 0xa1-0xfe;汉字从 0xb0a1 开始,结束于 0xf7fe。

(4). GBK

GBK 也是双字节内码,是 GB2312-80 的扩展,是

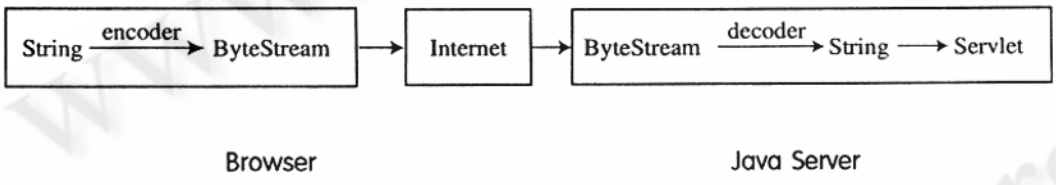
向上兼容的。它包含了 20902 个汉字，其编码范围是 0x8140 – 0xfefe，剔除高位 0x80 的字位。其所有字符都可以一对一映射到 Unicode 2.0，也就是说 JAVA 实际上提供了对 GBK 字符集的支持。这是现阶段 Windows 和其它一些中文操作系统的缺省字符集。

（5）. GB18030 – 2000（GBK2K）

GBK2K 是变长内码，它在 GBK 的基础上进一步扩展了汉字，增加了藏、蒙等少数民族的字形。GBK2K 从根本上解决了字位不够，字形不足的问题。它也是未来的中文国家标准。

（6）. UTF – 8

UTF – 8 是变长内码，它提供了一种简便而向后兼容的方法。它的出现是为了提高效率。当操作的文本基本都是规则的 ASCII 编码时，使用双字节的 Unicode 编码显得效率很低。因此 Java 字节码内的字符串使用了 UTF – 8 这种中间格式。其与 Unicode 一一对应的编码格式如下：



7 位的 Unicode：0 X X X X X X X

11 位的 Unicode：1 1 0 X X X X X 1 0 X X X X X X

16 位的 Unicode：1 1 1 0 X X X X 1 0 X X X X X X 1 0 X X X X X

21 位的 Unicode：1 1 1 1 0 X X X 1 0 X X X X X X 1 0 X X X X 1 0 X X X X X

其中 X 代表的是 Unicode 的编码，按规则加上“1”和“0”之后则是 UTF – 8 的编码。

除了 7 位的编码，可以看出，UTF – 8 编码的第一个字节高位 1 的数目就代表了该编码的字节数。如 21 位的编码首字节高位有 4 个“1”，其编码也为 4 字节编码。

3　几种中文问题的情况

3.1　中文问题的表现形式

Java 中中文问题的出现，主要是由于中文字符串无法正确显示。由于环境的不同，其出现情况也不相同。主要可能有以下几种情况。

（1）. jsp 文件中中文字符无法正确显示，出现类似于拉丁字母的乱码。

（2）. jsp 文件中传输的中文参数无法正确显示，显示为“?”。

（3）. 将中文数据存入数据库之后，再读出显示为“?”。

以上这些情况随着环境的不同而变化，相同的程序在不同的环境下有些会出现问题，有些则一切正常。

3.2　产生中文问题的原因。

中文不能正确显示的原因，主要是因为字符在不同的编码之间转换的过程中出错。不管是在 JSP 还是在 Servlet 中，我们都是用 ServletRequest（或其子类）的方法 getParameter（String name）来访问请求参数的，这个方法返回的是 String，也就是说我们能得到的是已经对 Internet 传来的字节流解码所得的字符串。如果服务器能对这些字节流进行正确的解码，那将是件完美的事。其实说来也很简单，要做到这一点只需要服务器知道这些字节流在客户端是用什么编码进行编码也就行了。数据从客户端到服务器端的方式如图 1 所示。

我们希望其中的 encoder 与 decoder 相同，这样在编码解码的过程中就不会出错。可是服务器端并不能知道客户端采用何种方式编码，如果不加设置，它会采用缺省的 ISO8859 方式解码。如果 encoder 与 decoder 不同，这样就会导致数据出错。

编码（Encode）和解码（Decode）是两个相反的动作。编码是把字符按照某种映射标准（字符集），转换成字节，如我们对 Unicode 字符串“我是中国人”按照 GB2312 标准编码（byte bsg[ ] = “我是中国人”. getBytes（“GB2312”）；），就可以得到一个字节序列（bytes sequence），用十六进制的码值表示：0xCE0xD20xCA0xC70xD60xD00xB90xFA0xC80xCB，按照 UTF – 8 标准编码（byte bsu[ ] = “我是中国人”. getBytes（“UTF – 8”）；），就可以得到字节序列：

0xE60x880x910xE60x980xAF0xE40xB80xAD0xE50x9B0xBD0xE40xBA0xBA。而解码则是将字节序列按照

某种字符标准(解码,decoding),转换成字符串。如果将以上两个字节序列分别按对应得 GB2312 和 UTF－8 得格式解码,都能得到"我是中国人"得字符串。而如果将 0xCE0xD20xCA0xC70xD60xD00xB90xFA0xC80xCB 得字节序列按照 UTF－8 得格式解码,将得到错误的结果,即显示为乱码。

在以上的过程中,出错主要发生在两个部分,即编码过程或者解码过程。我们采用 Unicode 做解释。编码过程:Unicode—>Byte,即将字符串按 Unicode 字符集转换成为 Byte 字节流。如果目标代码集不存在对应的代码,则得到的结果是 0x3f,即"?",而当字符串是 GB2312 格式的时候,符号区中的一些符号将被映射到一些公共的符号编码,最终显示为非"?"的一串杂乱字符。

解码过程是 Byte—>Unicode,即将 Byte 字节流按照 Unicode 字符集进行解码。如果 Byte 标识的字符在 Unicode 源代码集中不存在,则得到的结果是 0xfffd。在页面显示的时候,因为 0xfffd 不存在对应代码,最终也显示为"?"。

Java 中的中文问题其实就是编码与解码过程中错误的叠加,甚至是多次叠加之后产生的结果。因此,要彻底的解决中文问题,必须处理每一步的错误。

另外,Java 产品的最初应用都是基于西欧语系,因此,在双字节的 Unicode 内码转换为 ISO8859－1 编码时,可能会采取截取低位字节的方式。在英文环境中,这绝对没有问题,但是当文本中有中文存在时,也就产生了错误。从而也会产生中文问题。

4　解决中文问题的方法

搞清楚了中文问题产生的原因,也就相应的有了解决办法。既然中文问题是因为编码转换而产生的,那么我们只要搞清楚数据在各个部分是以什么样的编码方式存在,把不一致的转换过来就可以了。相对而言,Tomcat 作为一种免费的软件,它主要是基于英语环境,在某些细节方面处理的还不够完善,因此是产生问题最多的。而 resin 服务器则具有比较稳定的特性,对中文的支持也比较好,因此不大会有问题。在数据库系统方面,Oracle 和 DB2 等大型关系数据库在中文数据编码转换方面也处理的比较好,而 mysql 数据库由于并不支持 Unicode 的编码方式,因此在数据的写入与读出过程中也常常会产生中文问题。因此,本文一般讨论的中文问题解决方案,都是基于 Tomcat + mysql 的环境。而在其他的环境下,如果有中文问题,按照同样的规则则比较容易解决。

4.1　在 jsp 网页中定义输出字符集

大多数的浏览器都是采用 UTF－8 编码方式传输信息,当 jsp 页面中固定的中文字符无法正确显示时,可以将输出字符集设置为"GB2312"。采用如下方式:在页面的开头加入 <% @ page contentType = "text/html;charset = gb2312" % > 或者加入 meta 信息 < META HTTP－EQUIV = "Content－type" CONTENT = "text/html; charset = GB2312" >。这样在页面输出的时候,就可以采用 GB2312 的字符集,从而正确显示中文。

4.2　网页间中文参数的传输。

Java 在网络传输中使用的编码是"ISO8859－1",所以针对特定的输出字符集需要对中文参数进行转换。在这里,有一个原则就是要让中文参数的编码与输出字符集一致。即当按照 4.1 的方式定义了输出字符集为中文时,在使用 get 和 post 方法时需要进行转换。

```
public String getStr(String str){
  try{String temp_p = str;
    byte[ ]temp_t = temp_p. getBytes("ISO8859
－1");
      String temp = new String(temp_t,
"gb2312");
    return temp;
  }
  catch(Exception e){ }
  return "NULL";
}
public String setStr(String str){
  try{String temp_p = str;
    byte [ ] temp_t = temp_p. getBytes
("gb2312");
    String temp = new String(temp_t, "ISO8859
－1");
    return temp;
```

```
    }
    catch(Exception e){ }
    return "NULL";
  }
```

getStr 的作用是将接收的 ISO8859 -1 编码的参数转换为 gb2312 编码。SetStr 方法的作用与 getStr 相反,是将 gb2312 编码的参数转换为 ISO8859 -1 传输出去,。setStr 和 getStr 的使用主要是弥补编码自动转换的不足。这两个方法使用情况如下所示:

(1) 当 Form 采用 post 的方式发送参数时

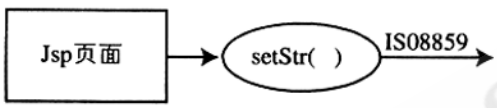


图 2　setStr 的使用

如果页面的输出字符集设置为 GB 时,参数传递时会自动将 GB 格式的编码转化为 ISO8859 -1 发送,否则,当使用缺省字符集 ISO8859 -1 时(如图 2),其中的中文参数则需要通过 setStr 方式手工转换为 ISO8859 -1 编码。

(2) 当 Form 采用 get 的方式获取中文参数时

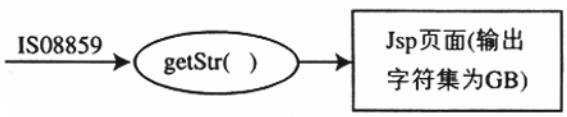


图 3　getStr 的使用

如果使用的是缺省字符集,则不需要进行字符转换。而当页面的输出字符集为 GB 时,则需要使用 getStr 的方法将 ISO8859 编码转换为 GB 编码。

由以上可以看出,setStr 和 getStr 方法的使用,都是为了保证数据流在网页间传递的过程中保持编码的一致性。最重要的就是做到在发送前将字符正确转换为 ISO8859 格式,而在页面中,则保持变量字符编码与输出字符集的一致。而这一原则也同样适用与页面与数据库之间的数据交互。

4.3　获取中文参数时定义解码方式。

在如上图 4.2 的方式中,中文参数数量较少的情况可以那样处理。然而,当 Form 格式中传递较多的参数时,需要对每一个参数都使用 getStr 的方式进行转换,这样就显得代码较乱而且降低效率,因此,可以在获取参数前,可以使用过滤器来设置请求实体得编码方式。如下:

```
request. setCharacterEncoding("gb2312");
```

并且设置响应实体得解码方式,如下:

```
response. setContentType ( " text/html; charset = gb2312" );
```

4.4　在编译 Servlet 和 JSP 时加入代码选项

在编译 Servlet 时使用 Java -encoding ISO8859 -1 *. java。在 JSP 的 ZONE 配置文件中,修改编译参数为:Compiler = builtin - javac - encoding ISO8859 -1。使用这种方法后,数据流都以 ISO8859 的编码方式存在,不需要再进行代码转换,因此可以正常显示中文。

4.5　有关数据库中文问题的处理

(1) 设置数据库 Encoding

一般的关系数据库都支持数据库 Encoding,在创建数据库时选用合适的编码方式,如 GB2312、GBK、UTF -8,这样数据库中的数据就以设置的编码方式存储。在数据的出口和入口都能自动进行数据转换,可以保证数据的正确性。而 ISO8859 的编码方式将会增加编码的复杂度,因此不建议使用。另外,选用数据库可以选用 Oracle、DB2 等大型数据库,mysql 数据库由于不支持 Unicode 编码方式,并且在数据出入口的自动换码方面有所欠缺,因此也不推荐使用。

(2) 通过转换函数实现 JDBC Driver 的字符转换

目前大多数 JDBC Driver 采用本地编码格式来传输中文字符,例如中文字符"0x4175"会被转成"0x41"和"0x75"进行传输。因此,向数据库中插入数据时,需要先将 Unicode 转成 Native code,当从数据库中查询数据时,则需要将 Native code 转换成 Unicode。但这种方式将使代码变得复杂并且难以移植,因此并非较好的方法。

以上五种基本的解决方案,其中的 1,3,4,5 都是比较规范性的方法,因此处理比较简单。并且也不具有较大的针对性,较为通用。然而也正因为此,不能有效的解决所有问题,方案 2,具有较大的针对性以及灵活性,可以自由的使用在需要的地方。也是最常用的一种方法。

(下转第 66 页)

(上接第 81 页)

5 总结

Java 编程语言应用于网络,这就要求 Java 对多国字符有很好的支持。Java 编程语言适应了计算的网络化的需求,为它能够在网络世界迅速成长奠定了坚实的基础。Java 的缔造者(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,需要我们付诸一些补偿性的措施。而在目前我们所考虑的解决方案更多的应该是具有普遍性的,而不是仅仅针对与某一些特定的运行环境。实际的处理过程中,遵循使字符串编码方式与当前页面解码方式一致的原则,针对应用环境处理双字节字符编码转换时的遗漏进行补救,从而可以解决各种不同环境下的中文问题。当然,这也导致了一个结果,当运行环境有所改变,那么程序又可能出现新的问题。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 ,它用四个字节来表示一个字符,也许等到那一天,程序员就不需要再为编码问题而烦恼。

参考文献

1 Justin Couch. Java 2 Enterprise Edition Bible. Wiley Publishing ,Inc. 2001.

2 段明辉, Java 编程技术中汉字问题的分析及解决, http://www - 900. ibm. com/developerWorks/cn/java/java_chinese/index. shtml , 2000 - 11 - 08.

3 http://www. unicode. org/

4 whodsow, J2EE Web 组件中中文及相关的问题, http://www. csdn. net/develop/article/21/21760. shtm.