**深入理解IO流中字符编码问题**

编码只发生在JVM和底层操作系统（以及网络传输）之间进行数据传输时，如果程序中没有IO操作，那么所有的String和Char都以unicode编码。当从磁盘读取文件或者往磁盘写入文件时使用的编码要一致，也就是编码和解码使用的字符集要一样才不会出现乱码！

**首先了解一下常用的编码：**

**ASCII** ： ASCII码是7位编码，ASCII字符集包括英文字母、阿拉伯数字和标点符号等字符。专门给英语国家设计的编码。

**GB2312：** 中文字符集 ， 只收录了6763个常用汉字， 字符集中除常用简体汉字字符外还包括希腊字母、日文等字符，未收录繁体中文汉字和一些生僻字。

**GBK：** GBK编码是GB2312编码的超集 ， 共收录了21003个汉字 ， 向下完全兼容GB2312。

ISO [***8859 -1***](http://baike.baidu.com/view/2771038.htm)： 又称Latin-1或“西欧 语言” 。它以ASCII为基础，在空置的0xA0-0xFF的范围内，加入96个字母及符号，藉以供使用变音符号的拉丁字母语言使用 。

Unicode ：又称万国码，顾名思义，unicode中收录了世界各国语言，用以解决传统编码的局限性。它为每种语言中的每个字符设定了统一并且唯一的二进制编码，以满足跨语言、跨平台进行文本转换、处理的要求。

UTF-8：（8-bit Unicode Transformation Format）是一种针对Unicode的可变长度字符编码。

**对于不同编码之间有很重要的两点需要知道：**

1、不同编码表示一个字符所占用字节是不相同的，其中ASCII码占一个字节，GB2312、GBK、unicode都用2字节表示一个字符。而UTF-8是可变长的，英文字母用1字节，汉字用3字节表示。

2、同一个字符在不同编码表中的位置也是不同的，比如汉字‘中’在GBK中是D6D0，而在unicode中是4E2D。也就导致了当汉字的解码方式和编码方式不同时会产生乱码。

**Unicode编码**

每个Java程序员都应该记住，Java使用的是Unicode编码。所有的字符在JVM中（内存中）只有一个存在形式就是Unicode。所以一个char占用2字节。

**编码只发生在JVM和底层操作系统（以及网络传输）之间进行数据传输时，如果程序中没有IO操作，那么所有的String和Char都以unicode编码。当从磁盘读取文件或者往磁盘写入文件时使用的编码要一致，也就是编码和解码使用的字符集要一样才不会出现乱码！**

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

File file=new File("d:\\utf.txt");

//用utf-8编码写入文件

BufferedWriter out=new BufferedWriter(

new OutputStreamWriter(new FileOutputStream("d:\\utf.txt"),"utf-8"));

out.write("汉字");

out.close();

//用gbk编码进行解码，会出现乱码

System.out.println("文件大小："+file.length()+"字节");

BufferedReader in=new BufferedReader(

new InputStreamReader(new FileInputStream("d:\\utf.txt"),"gbk"));

String line=in.readLine();

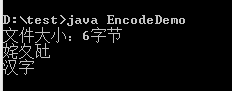
System.out.println(line);

//再次用utf-8编码，得到原文

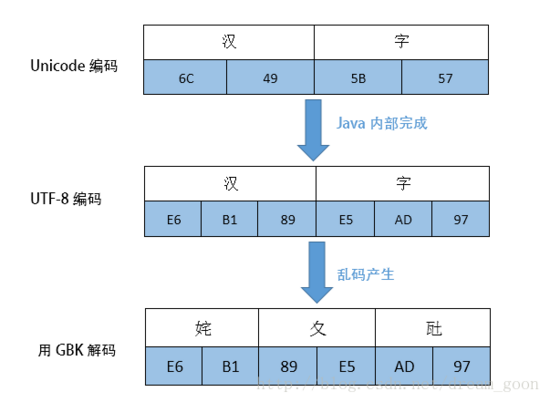
byte[] bs=line.getBytes("gbk");

System.out.println(new String(bs,"utf-8"));

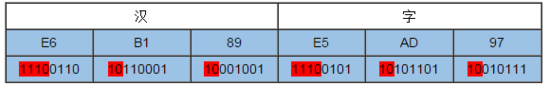
}



我们来看一下程序中涉及到的编码转换过程。



首先，Java会以默认的Unicode编码来对”汉字“编码，占用两个字节(6C49 5B57)存储在内存中，当要写入文件中时，就按程序指定的UTF-8编码对“汉字”重新编码，UTF-8一个汉字占用3个字节，得到E6 B1 89 E5 AD 97（11100110 10110001 10001001 11100101 10101101 10010111）存储在utf.txt中。因为UTF-8是可变长度的，那么用UTF-8解码文件时，怎么知道的几个字节表示一个字符呢？ 原来UTF-8编码时每个字节的高位部分都有表示头，如果一个字节高位是1110，则表示3个字节表示一个字符，应该再往后找2个以10开始的字节。 字节高位是10表示前面还有字节。



以上面这个例子说明：“汉字”共占6个字节，第一个字节是11100110，高位是1110表示后边还有2个字节，再往后读2个字节，3个字节解码成一个汉字——”汉“。同理解码”字“。 再例如有个字节是11010101，高位是110说明是2个字节表示一个字符，后面还有一个以10开始的字节。如果高位是0开始，表示是单字节字符，abc字母都占用一个字节。

程序中用GBK编码来解码UTF-8编码的”汉字“就会出现乱码，因为GBK编码无论是字母还是汉字都占用2个字节，它在解码时就会按顺序2个字符解码为一个字符，刚好E6B1在GBK编码表中是汉字“姹”，89E5是“夊”，AD97是“瓧”，最后解码成了“姹夊瓧”这个神奇的东西，如果对应位置GBK表中没有字符可能就显示“？”（ISO-8859-1就会这么干），所以啊，用什么编码就得用什么解码。

**那乱码还能恢复原来的内容吗？那是肯定滴，因为只是解码时出现了问题，文件并没有改变，还是这些01的二进制。通过乱码“ 姹夊瓧 ”重新得到其字节文件，然后再次用UTF-8去解码就可以了。**

byte[] bs=line.getBytes("gbk"); //得到GBK编码中的字节

System.out.println(new String(bs,"utf-8"));//用UTF-8去正确解码

如果在读写文件时没有在程序中显示指定编码方式，则使用操作系统默认的编码格式。注意是OS默认的编码格式，不是JVM默认的unicode，中文系统Windows默认是GBK，英文系统默认UTF-8。

Java的IO体系中面向字符的IO类只有Reader和Writer，但是最常用的FileReader和FileWriter类不支持自定义编码类型，只能使用系统默认编码。这样一来，读写文件的编码就一定一致了，也就减少了乱码的可能性。个人理解，这么做可能是强制帮助用户完成编码一致，降低乱码率。**如果要自定义编码，要用其父类InputStreamRreader和OutputStreamWriter。**这些类的具体区别和用法，会在以后的文章中讲到。

在实际开发中，只要设置编码一致，就很少出现乱码问题。只要你深刻的理解了Java编码的原理，就算出现乱码也能很快的去解决问题了。