**Java中文字符编码问题详解 控制台输出**

Java的中文字符乱码问题一直很让人头疼。特别是在WEB应用中。网上的分析文章和解决方案都很多，但总是针对某些特定情况的。很多次遇到乱码问题后， 经过极为辛苦的调试和搜索资料后终于解决，满以为自己已经掌握了对付这些字符乱码怪兽的诀窍。可当过段时间，换了个应用或换了个环境，又会碰到那讨厌的火星文，并再次无所适从。于是下决心好好整理一下中文字符编码问题，以方便自己记忆，也为其他程序员兄弟们提供一份参考。  
首先要了解Java处理字符的原理。Java使用UNICODE来存储字符数据，处理字符时通常有叁个步骤：  
- 按指定的字符编码形式，从源输入流中读取字符数据  
- 以UNICODE编码形式将字符数据存储在内存中  
- 按指定的字符编码形式，将字符数据编码并写入目的输出流中。  
所以Java处理字符时总是经过了两次编码转换，一次是从指定编码转换为UNICODE编码，一次是从UNICODE编码转换为指定编码。如果在读入时用 错误的形式解码字符，则内存存储的是错误的UNICODE字符。而从最初文件中读出的字符数据，到最终在屏幕终端显示这些字符，期间经过了应用程序的多次 转换。如果中间某次字符处理，用错误的编码方式解码了从输入流读取的字符数据，或用错误的编码方式将字符写入输出流，则下一个字符数据的接收者就会编解码 出错，从而导致最终显示乱码。  
这一点，是我们分析字符编码问题以及解决问题的指导思想。  
好，现在我们开始一只只的解决这些乱码怪兽。  
**一、在Java文件中硬编码中文字符，在eclipse中运行，控制台输出了乱码。**  
例如，我们在Java文件中写入以下代码：  
**String text = “大家好”;**  
System.out.println（text）;  
如果我们是在eclipse里编译运行，可能看到的结果是类似这样的乱码：？？？？。那么，这是为什么呢？  
我们先来看看整个字符的转换过程。  
1. 在eclipse窗口中输入中文字符，并保存成UTF-8的Java文件。这里发生了多次字符编码转换。不过因为我们相信eclipse的正确性，所以我们不用分析其中的过程，只需要相信保存下的Java文件确实是UTF-8格式。  
2. 在eclipse中编译运行此Java文件。这里有必要详细分析一下编译和运行时的字符编码转换。  
- 编译：我们用Javac编译Java文件时，Javac不会智能到猜出你所要编译的文件是什么编码类型的，所以它需要指定读取文件所用的编码类型。默认 Javac使用平台缺省的字符编码类型来解析Java文件。平台缺省编码是操作系统决定的，我们使用的是中文操作系统，语言区域设置通常都是中国大陆，所 以平台缺省编码类型通常是GBK。这个编码类型我们可以在Java中使用System.getProperty（“file.encoding”）来查 看。所以Javac会默认使用GBK来解析Java文件。如果我们要改变Javac所用的编码类型，就要加上-encoding参数，如Javac -encoding utf-8 Test.Java。  
这里要另外提一下的是eclipse使用的是内置的编译器，并不能添加参数，如果要为Javac添加参数则建议使用ANT来编译。不过这并非出现乱码的塬因，因为eclipse可以为每个Java文件设置字符编码类型，而内置编译器会根据此设置来编译Java文件。  
- 运行：编译后字符数据会以UNICODE格式存入字节码文件中。然后eclipse会调用Java命令来运行此字节码文件。因为字节码中的字符总是 UNICODE格式，所以Java读取字节码文件并没有编码转换过程。虚拟机读取文件后，字符数据便以UNICODE格式存储在内存中了。  
3. 调用System.out.println来输出字符。这里又发生了字符编码转换。  
System.out.println使用了PrintStream类来输出字符数据至控制台。PrintStream会使用平台缺省的编码方式来输出字 符。我们的中文系统上缺省方式为GBK，所以内存中的UNICODE字符被转码成了GBK格式，并送到了操作系统的输出服务中。因为我们操作系统是中文系 统，所以往终端显示设备上打印字符时使用的也是GBK编码。如果到这一步，我们的字符其实不再是GBK编码的话，终端就会显示出乱码。  
那么，在eclipse运行带中文字符的Java文件，控制台显示了乱码，是在哪一步转换错误呢？我们一步步来分析。  
- 保存Java文件成UTF-8后，如果再次打开你没有看到乱码，说明这步是正确的。  
- 用eclipse本身来编译运行Java文件，应该没有问题。  
- System.out.println会把内存中正确的UNICODE字符编码成GBK，然后发到eclipse的控制台去。等等，我们看到在Run Configuration对话框的Common标签里，控制台的字符编码被设置成了UTF-8！问题就在这里。System.out.println已 经把字符编码成了GBK，而控制台仍然以UTF-8的格式读取字符，自然会出现乱码。  
将控制台的字符编码设置为GBK，乱码问题解决。  
（这里补充一点：eclipse的控制台编码是继承了workspace的设置的，通常控制台编码里没有GBK的选项而且不能输入。我们可以先在 workspace的编码设置中输入GBK，然后在控制台的设置中就可以看到GBK的选项了，设置好后再把workspace的字符编码设置改回utf- 8就是。）  
**二、JSP文件中硬编码中文字符，在浏览器上显示乱码。**  
我们用eclipse编写一个JSP页面，使用tomcat浏览这个页面时，整个页面的中文字符都是乱码。这是什么塬因呢？  
JSP页面从编写到在浏览器上浏览，总共有四次字符编解码。  
1. 以某种字符编码保存JSP文件  
2. Tomcat以指定编码来读取JSP文件并编译

3. Tomcat向浏览器以指定编码来发送HTML内容  
4. 浏览器以指定编码解析HTML内容  
这里的四次字符编解码，有一次发生错误最终显示的就会是乱码。我们依次来分析各次的字符编码是如何设置的。  
- 保存JSP文件，这是在编辑器中设置的，比如eclipse中，设置文件字符类型为utf-8。  
- JSP文件开头的《%@ page language=“Java” contentType=“text/html; charset=utf-8” pageEncoding=“utf-8”%》，其中pageEncoding用来告诉tomcat此文件所用的字符编码。这个编码应该与eclipse 保存文件用的编码一致。Tomcat以此编码方式来读取JSP文件并编译。  
- page标签中的contentType用来设置tomcat往浏览器发送HTML内容所使用的编码。这个编码会在HTTP响应头中指定以通知浏览器。  
- 浏览器根据HTTP响应头中指定的字符编码来解析HTML内容。如：  
HTTP/1.1 200 OK  
Date： Mon， 01 Sep 2008 23:13:31 GMT  
Server： Apache/2.2.4 （Win32） mod\_jk/1.2.26  
Vary： Host，Accept-Encoding  
Set-Cookie： Java2000\_STYLE\_ID=1; Domain=www.Java2000.net; Expires=Thu， 03-Nov-2011 09:00:10 GMT; Path=/  
Content-Encoding： gzip  
Transfer-Encoding： chunked  
Content-Type： text/html;charset=UTF-8  
另外，HTML中有个标签<meta http-equiv=“Content-Type” content=“text/html; charset=UTF-8”>中也指定了charset。不过这个字符编码只有在当网页保存在本地作为静态网页时有效，因为没有HTTP头，所以浏览器 根据此标签来识别HTML内容的编码方式。  
现在在JSP文件中硬编码出现乱码的机会比较小了，因为大家都用了如eclipse的编辑器，基本上可以自动保证这几个编码设置的正确性。现在更多碰到的是在JSP文件中从其他数据源中读取中文字符所产生的乱码问题。  
**叁、在JSP文件中读取字符文件并在页面中显示，中文字符显示为乱码。**  
比如，我们在JSP文件中使用以下代码：  
《%  
BufferedReader reader = new BufferedReader（new FileReader（“D：\\test.txt”））;  
String content = reader.readLine（）;  
reader.close（）;  
%》  
《%=content%》  
test.txt里保存的是中文字符，但在浏览器上看到的乱码。这是个经常见到的问题。我们继续用之前的方法一步步来分析输入和输出流  
1. test.txt是以某种编码方式保存中文字符，比如UTF-8。  
2. BufferedReader直接读取test.txt的字节内容并以默认方式构造字符串。分析BufferedReader的代码，我们可以看到 BufferedReader调用了FileReader的read方法，而FileReader又调用了FileInputStream的native 的read方法。所谓native的方法，就是操作系统底层方法。那么我们操作系统是中文系统，所以FileInputStream默认用GBK方式读取 文件。因为我们保存test.txt用的是UTF-8，所以在这里读取文件内容使用GBK是错误的编码。  
3. 《%=content%》其实就是out.print（content），这里又用到了HTTP的输出流JspWriter，于是字符串content又被以JSP的page标签中指定的UTF-8方式编码成字节数组被发送到浏览器端。  
4. 浏览器以HTTP头中指定的方式解码字符，这时无论是用GBK还是UTF-8解码，显示的都是乱码。  
可见，我们字符编码转换在第二步时出错了，UTF-8的字符串被当做GBK读入了内存中。  
解决这个乱码问题有两种方法，一是把test.txt用GBK保存，则FileInputStream能正确读入中文字符；二是使用InputStreamReader来转换字符编码，如：  
InputStreamReader sr = new InputStreamReader（new FileInputStream（“D：\\test.txt”），“utf-8”）;  
BufferedReader reader = new BufferedReader（sr）;  
这样，Java就会用utf-8的方式来从文件中读取字符数据。  
另外，我们可以通过在Java命令后带上Dfile.encoding参数来指定虚拟机读取文件使用的默认字符编码，例如Java -Dfile.encoding=utf-8 Test，这样，我们在Java代码里用System.getProperty（“file.encoding”）取到的值为utf-8。  
**四、JSP读取request.getParameter里的中文参数后，在页面显示为乱码。**  
在Java的WEB应用中，对request对象里的parameters的中文处理一直是常见也最难搞的一只大怪兽。经常是刚搞定了这边，那边又出了乱 码。而导致这种复杂性的，主要是此过程中字符编解码次数非常多，而且无论是浏览器还是WEB服务器特别是TOMCAT总是不能给我们一个比较满意的支持。  
首先我们来分析用GET方式上传参数的乱码情况。  
例如我们在浏览器地址栏输入以下URL：http://localhost:8080/test/test.jsp？param=大家好  
我们的JSP代码如此处理param这个参数：  
《% String text = request.getParameter（“param”）; %》  
《%=text%》  
而就这么简单的两句代码，我们很有可能在页面上看到这样的乱码：？ó？？？？  
网上对处理request.getParamter中的乱码有很多文章和方法，也都是正确的，只是方法太多让人一直不明白到底是为什么。这里给大家分析一下到底是怎么一回事。  
首先，我们来看看与request对象有哪些相关的编码设置：  
1. JSP文件的字符编码  
2. 请求这个带参数URL的源页面的字符编码  
3. IE的高级设置中的选项“总以utf-8方式发送URL地址”  
4. TOMCAT的server.xml中配置URIEncoding  
5. 函数request.setCharacterEncoding（）  
6. JS的encodeURIComponent函数与Java的URLDecoder类  
这么多条相关编码设置，也难怪大家被搞得头晕了。这里给大家根据各种情况给大家一一分析一下。见下表：



以上表格里的现象，除了指名在IE7上，其他全是在IE6上测试的结果。

由这个表我们可以看到，IE的“总以utf-8方式发送URL地址”设置并不影响对parameter的解析，而从页面请求URL和从地址栏输入URL居然也有不同的表现。  
根据这个表列出的现象，大家只要用smartSniff抓几个网络包，并稍稍调查一下TOMCAT的源代码，就可以得出以下结论：  
1. IE设置中的“总以utf-8方式发送URL地址”只对URL的PATH部分起作用，对查询字符串是不起作用的。也就是说，如果勾选了这个选项，那么类似 http://localhost:8080/test/大家好.jsp？param=大家好这种URL，前一个“大家好”将被转化成utf-8形式，而 后一个并没有变化。这里所说的utf-8形式，其实应该叫utf-8+escape形式，即%B4%F3%BC%D2%BA%C3这种形式。  
那么，查询字符串中的中文字符，到底是用什么编码传送到服务器的呢？答案是系统默认编码，即GBK。也就是说，在我们中文操作系统上，传送给WEB服务器的查询字符串，总是以GBK来编码的。  
2. 在页面中通过链接或location重定向或open新窗口的方式来请求一个URL，这个URL里面的中文字符是用什么编码的？答：是用该页面的编码类 型。也就是说，如果我们从某个源JSP页面上的链接来访问http://localhost:8080/test/test.jsp？param=大家好 这个URL，如果源JSP页面的编码是UTF-8，则大家好这几个字的编码就是UTF-8。  
而在地址栏上直接输入URL地址，或者从系统剪贴板粘贴到地址栏上，这个输入并非从页面中发起的，而是由操作系统发起的，所以这个编码只可能是系统的默认 编码，与任何页面无关。我们还发现，在不同的浏览器上，用链接方式打开的页面，如果在地址栏上再敲个回车，显示的结果也会不同。IE上敲回车后显示不变 化，而傲游上可能就会有乱码或乱码消失的变化。说明IE上敲回车，实际发送的是之前记忆下来的内存中的URL，而傲游上发送的从当前地址栏重新获取的 URL。  
3. TOMCAT的URIEncoding如果不加以设置，则默认使用ISO-8859-1来解码URL，设置后便用设置了的编码方式来解码。这个解码同时包 括PATH部分和查询字符串部分。可见，这个参数是对用GET方式传递的中文参数最关键的设置。不过，这个参数只对GET方式传递的参数有效，对POST 的无效。分析TOMCAT的源代码我们可以看到，在请求一个页面时，TOMCAT会尝试构造一个Request对象，在这个对象里，会从 Server.xml里读取URIEncoding的值，并赋值给Parameters类的queryStringEncoding变量，而这个变量将在 解析request.getParameter中的GET参数时用来指导字符解码。  
4. request.setCharacterEncoding函数只对POST的参数有效，对GET的参数无效。且这个函数必须是在第一次调用 request.getParameter之前使用。这是因为Parameters类有两个字符编码参数，一个是encoding，另一个是 queryStringEncoding，而setCharacterEncoding设置的是encoding，这个是在解析POST的参数是才用到 的。  
所以，这就导致了我们通常都要分开处理POST和GET的字符编码，用TOMCAT自带的filter只能处理POST的，另外要设置URIEncoding来设置GET的。这样很麻烦而且URIEncoding无法根据内容来动态区分编码，总还是一个问题。  
在调查TOMCAT的代码时发现了另一个在server.xml里的参数useBodyEncodingForURI，可以解决这个问题。这个参数设成 true后，TOMCAT就会用request.setCharacterEncoding所设置的字符编码来同样解析GET参数了。这样，那个 SetCharacterEncodingFilter就可以同时处理GET和POST参数了。  
知道了以上知识后，我们再来分析一下前面表格中列出的几个典型现象。  
第一条，请求源页面的编码为UTF-8，而TOMCAT的URIEncoding未指定，则TOMCAT用ISO8859-1方式来解码参数，所以从request中读出来后，内存中存储的为错误的UNICODE数据，导致之后到屏幕显示的所有转换全部出错。  
第九条，请求源页面编码为GBK，而TOMCAT的URIEncoding也为GBK，TOMCAT用GBK方式去解码塬本用GBK编码的字符，解码正确，内存中的UNICODE值正确，最终显示正确的中文。  
第十叁条，请求源页面编码为UTF-8，TOMCAT的URIEncoding也为UTF-8，而在IE6中最终显示的中文字符，如果是奇数个数，则最后一个会显示为乱码。这是为什么呢？  
我的猜测是，这是因为IE6将URL地址发送时，对查询字符串是直接对UTF-8格式的字符使用GBK来编码，而不是对UNICODE的字符来用GBK编 码，所以UTF-8的数据没有经过UNICODE而直接编码成了GBK。而到了TOMCAT这边，GBK的编码又被当成UTF-8做了解码。所以这个过程 中经过了UTF-8转换成GBK，然后又从GBK转换成UTF-8的过程，而这种转换，恰好就会出现奇数个中文字符串的最后一位为乱码的现象。而在IE7 中，估计把这种现象当做BUG已经被解决了，即在发送地址时会先转成UNICODE再编码成GBK。那么估计在IE7的浏览器+中文操作系统环境下，如果 我们把TOMCAT的URIEncoding设置成GBK，无论JSP编码成什么格式，都不会出现乱码。这个没测试，请大家自己验证。  
其他几条就不再做分析了，有兴趣的大家自己分析。  
**五、对URL做Encode和Decode**  
对于request参数的中文乱码问题，个人觉得最好的还是用URLEncode/URLDecode，因为如果你的WEB站点要支持国际化，最好就是保证从IE递送过来的参数永远是正确的UTF-8编码。  
在IE端，我们可以用JS脚本来对参数编码：encodeURIComponent（），编码后中文字符便变成了%B4%F3%BC%D2%BA%C3这 种形式。在Java端，可以用Java.net.URLDecoder.decode来解码。不过这里要注意一个问题，就是TOMCAT会自动先对URL 做一次decode，我们可以在TOMCAT的UDecoder类中看到这一点。不过TOMCAT并非使用了URLDecoder.decode，而是自 己编写了一个decode函数。网上有些文章上介绍过一种处理乱码的方法便是在JS中对参数做两次encodeURIComponent，在Java中做 一次decode，可以解决一些没有设置URIEncoding时发生的乱码问题。不过个人觉得如果弄懂了整个字符编码转换的过程，基本上是用不到这种方 法的。  
**六、从数据库中读取中文字符数据，在页面上显示为乱码。**  
对于数据库中读取中文字符出现乱码的问题，本人遇到的还比较少，所以暂时没有总结。如果大家有类似的经验，欢迎补充说明，我一定注明作者身份。  
好了，对各种字符乱码问题的分析就总结到这里，相信只要把握“以指定编码读取--转换为UNICODE--以指定编码输入”这基本步骤，初学者也可以很快 分析出字符乱码的根源所在。另外我建议不要随便使用new String（str.getBytes（enc1），enc2）这种方式来强行转码，也不要随便使用网上的字符转码函数，我觉得只会把问题隐藏更深更复 杂化。我们应该清晰地分析整个字符流的编解码过程，自然可以找出乱码的根源所在，从而保证整个字符流动中，在内存中的UNICODE始终是正确的。