1. 概述

本文主要包括以下几个方面：编码基本知识，java，系统软件，url，工具软件等。

在下面的描述中，将以"中文"两个字为例，经查表可以知道其GB2312编码是"d6d0 cec4"，Unicode编码为"4e2d 6587"，UTF编码就是"e4b8ad e69687"。注意，这两个字没有iso8859-1编码，但可以用iso8859-1编码来"表示"。

2. 编码基本知识

最早的编码是iso8859-1，和ascii编码相似。但为了方便表示各种各样的语言，逐渐出现了很多标准编码，重要的有如下几个。

2.1. iso8859-1

属于单字节编码，最多能表示的字符范围是0-255，应用于英文系列。比如，字母'a'的编码为0x61=97。

很明显，iso8859-1编码表示的字符范围很窄，无法表示中文字符。但是，由于是单字节编码，和计算机最基础的表示单位一致，所以很多时候，仍旧使用iso8859-1编码来表示。而且在很多协议上，默认使用该编码。比如，虽然"中文"两个字不存在iso8859-1编码，以gb2312编码为例，应该是"d6d0 cec4"两个字符，使用iso8859-1编码的时候则将它拆开为4个字节来表示："d6 d0 ce c4"（事实上，在进行存储的时候，也是以字节为单位处理的）。而如果是UTF编码，则是6个字节"e4 b8 ad e6 96 87"。很明显，这种表示方法还需要以另一种编码为基础。

2.2. GB2312/GBK

这就是汉子的国标码，专门用来表示汉字，是双字节编码，而英文字母和iso8859-1一致（兼容iso8859-1编码）。其中gbk编码能够用来同时表示繁体字和简体字，而gb2312只能表示简体字，gbk是兼容gb2312编码的。

2.3. unicode

这是最统一的编码，可以用来表示所有语言的字符，而且是定长双字节（也有四字节的）编码，包括英文字母在内。所以可以说它是不兼容iso8859-1编码的，也不兼容任何编码。不过，相对于iso8859-1编码来说，uniocode编码只是在前面增加了一个0字节，比如字母'a'为"00 61"。

需要说明的是，定长编码便于计算机处理（注意GB2312/GBK不是定长编码），而unicode又可以用来表示所有字符，所以在很多软件内部是使用unicode编码来处理的，比如java。

2.4. UTF

考虑到unicode编码不兼容iso8859-1编码，而且容易占用更多的空间：因为对于英文字母，unicode也需要两个字节来表示。所以unicode不便于传输和存储。因此而产生了utf编码，utf编码兼容iso8859-1编码，同时也可以用来表示所有语言的字符，不过，utf编码是不定长编码，每一个字符的长度从1-6个字节不等。另外，utf编码自带简单的校验功能。一般来讲，英文字母都是用一个字节表示，而汉字使用三个字节。

注意，虽然说utf是为了使用更少的空间而使用的，但那只是相对于unicode编码来说，如果已经知道是汉字，则使用GB2312/GBK无疑是最节省的。不过另一方面，值得说明的是，虽然utf编码对汉字使用3个字节，但即使对于汉字网页，utf编码也会比unicode编码节省，因为网页中包含了很多的英文字符。

3. java对字符的处理

在java应用软件中，会有多处涉及到字符集编码，有些地方需要进行正确的设置，有些地方需要进行一定程度的处理。

3.1. getBytes(charset)

这是java字符串处理的一个标准函数，其作用是将字符串所表示的字符按照charset编码，并以字节方式表示。注意字符串在java内存中总是按unicode编码存储的。比如"中文"，正常情况下（即没有错误的时候）存储为"4e2d 6587"，如果charset为"gbk"，则被编码为"d6d0 cec4"，然后返回字节"d6 d0 ce c4"。如果charset为"utf8"则最后是"e4 b8 ad e6 96 87"。如果是"iso8859-1"，则由于无法编码，最后返回 "3f 3f"（两个问号）。

3.2. new String(charset)

这是java字符串处理的另一个标准函数，和上一个函数的作用相反，将字节数组按照charset编码进行组合识别，最后转换为unicode存储。参考上述getBytes的例子，"gbk" 和"utf8"都可以得出正确的结果"4e2d 6587"，但iso8859-1最后变成了"003f 003f"（两个问号）。

因为utf8可以用来表示/编码所有字符，所以new String( str.getBytes( "utf8" ), "utf8" ) === str，即完全可逆。

3.3. setCharacterEncoding()

该函数用来设置http请求或者相应的编码。

对于request，是指提交内容的编码，指定后可以通过getParameter()则直接获得正确的字符串，如果不指定，则默认使用iso8859-1编码，需要进一步处理。参见下述"表单输入"。值得注意的是在执行setCharacterEncoding()之前，不能执行任何getParameter()。java doc上说明：This method must be called prior to reading request parameters or reading input using getReader()。而且，该指定只对POST方法有效，对GET方法无效。分析原因，应该是在执行第一个getParameter()的时候，java将会按照编码分析所有的提交内容，而后续的getParameter()不再进行分析，所以setCharacterEncoding()无效。而对于GET方法提交表单是，提交的内容在URL中，一开始就已经按照编码分析所有的提交内容，setCharacterEncoding()自然就无效。

对于response，则是指定输出内容的编码，同时，该设置会传递给浏览器，告诉浏览器输出内容所采用的编码。

3.4. 处理过程

下面分析两个有代表性的例子，说明java对编码有关问题的处理方法。

3.4.1. 表单输入

User input  \*(gbk:d6d0 cec4)  browser  \*(gbk:d6d0 cec4)  web server  iso8859-1(00d6 00d 000ce 00c4)  class，需要在class中进行处理：getbytes("iso8859-1")为d6 d0 ce c4，new String("gbk")为d6d0 cec4，内存中以unicode编码则为4e2d 6587。

l 用户输入的编码方式和页面指定的编码有关，也和用户的操作系统有关，所以是不确定的，上例以gbk为例。

l 从browser到web server，可以在表单中指定提交内容时使用的字符集，否则会使用页面指定的编码。而如果在url中直接用?的方式输入参数，则其编码往往是操作系统本身的编码，因为这时和页面无关。上述仍旧以gbk编码为例。

l Web server接收到的是字节流，默认时（getParameter）会以iso8859-1编码处理之，结果是不正确的，所以需要进行处理。但如果预先设置了编码（通过request. setCharacterEncoding ()），则能够直接获取到正确的结果。

l 在页面中指定编码是个好习惯，否则可能失去控制，无法指定正确的编码。

3.4.2. 文件编译

假设文件是gbk编码保存的，而编译有两种编码选择：gbk或者iso8859-1，前者是中文windows的默认编码，后者是linux的默认编码，当然也可以在编译时指定编码。

Jsp  \*(gbk:d6d0 cec4)  java file  \*(gbk:d6d0 cec4)  compiler read  uincode(gbk: 4e2d 6587; iso8859-1: 00d6 00d 000ce 00c4)  compiler write  utf(gbk: e4b8ad e69687; iso8859-1: \*)  compiled file  unicode(gbk: 4e2d 6587; iso8859-1: 00d6 00d 000ce 00c4)  class。所以用gbk编码保存，而用iso8859-1编译的结果是不正确的。

class  unicode(4e2d 6587)  system.out / jsp.out  gbk(d6d0 cec4)  os console / browser。

l 文件可以以多种编码方式保存，中文windows下，默认为ansi/gbk。

l 编译器读取文件时，需要得到文件的编码，如果未指定，则使用系统默认编码。一般class文件，是以系统默认编码保存的，所以编译不会出问题，但对于jsp文件，如果在中文windows下编辑保存，而部署在英文linux下运行/编译，则会出现问题。所以需要在jsp文件中用pageEncoding指定编码。

l Java编译的时候会转换成统一的unicode编码处理，最后保存的时候再转换为utf编码。

l 当系统输出字符的时候，会按指定编码输出，对于中文windows下，System.out将使用gbk编码，而对于response（浏览器），则使用jsp文件头指定的contentType，或者可以直接为response指定编码。同时，会告诉browser网页的编码。如果未指定，则会使用iso8859-1编码。对于中文，应该为browser指定输出字符串的编码。

l browser显示网页的时候，首先使用response中指定的编码（jsp文件头指定的contentType最终也反映在response上），如果未指定，则会使用网页中meta项指定中的contentType。

3.5. 几处设置

对于web应用程序，和编码有关的设置或者函数如下。

3.5.1. jsp编译

指定文件的存储编码，很明显，该设置应该置于文件的开头。例如：<%@page pageEncoding="GBK"%>。另外，对于一般class文件，可以在编译的时候指定编码。

3.5.2. jsp输出

指定文件输出到browser是使用的编码，该设置也应该置于文件的开头。例如：<%@ page contentType="text/html; charset= GBK" %>。该设置和response.setCharacterEncoding("GBK")等效。

3.5.3. meta设置

指定网页使用的编码，该设置对静态网页尤其有作用。因为静态网页无法采用jsp的设置，而且也无法执行response.setCharacterEncoding()。例如：<META http-equiv="Content-Type" content="text/html; charset=GBK" />

如果同时采用了jsp输出和meta设置两种编码指定方式，则jsp指定的优先。因为jsp指定的直接体现在response中。

需要注意的是，apache有一个设置可以给无编码指定的网页指定编码，该指定等同于jsp的编码指定方式，所以会覆盖静态网页中的meta指定。所以有人建议关闭该设置。

3.5.4. form设置

当浏览器提交表单的时候，可以指定相应的编码。例如：<form accept-charset= "gb2312">。一般不必不使用该设置，浏览器会直接使用网页的编码。

4. 系统软件

下面讨论几个相关的系统软件。

4.1. mysql数据库

很明显，要支持多语言，应该将数据库的编码设置成utf或者unicode，而utf更适合与存储。但是，如果中文数据中包含的英文字母很少，其实unicode更为适合。

数据库的编码可以通过mysql的配置文件设置，例如default-character-set=utf8。还可以在数据库链接URL中设置，例如： useUnicode=true&characterEncoding=UTF-8。注意这两者应该保持一致，在新的sql版本里，在数据库链接URL里可以不进行设置，但也不能是错误的设置。

4.2. apache

appache和编码有关的配置在httpd.conf中，例如AddDefaultCharset UTF-8。如前所述，该功能会将所有静态页面的编码设置为UTF-8，最好关闭该功能。

另外，apache还有单独的模块来处理网页响应头，其中也可能对编码进行设置。

4.3. linux默认编码

这里所说的linux默认编码，是指运行时的环境变量。两个重要的环境变量是LC\_ALL和LANG，默认编码会影响到java URLEncode的行为，下面有描述。

建议都设置为"zh\_CN.UTF-8"。

4.4. 其它

为了支持中文文件名，linux在加载磁盘时应该指定字符集，例如：mount /dev/hda5 /mnt/hda5/ -t ntfs -o iocharset=gb2312。

另外，如前所述，使用GET方法提交的信息不支持request.setCharacterEncoding()，但可以通过tomcat的配置文件指定字符集，在tomcat的server.xml文件中，形如：<Connector ... URIEncoding="GBK"/>。这种方法将统一设置所有请求，而不能针对具体页面进行设置，也不一定和browser使用的编码相同，所以有时候并不是所期望的。

5. URL地址

URL地址中含有中文字符是很麻烦的，前面描述过使用GET方法提交表单的情况，使用GET方法时，参数就是包含在URL中。

5.1. URL编码

对于URL中的一些特殊字符，浏览器会自动进行编码。这些字符除了"/?&"等外，还包括unicode字符，比如汉子。这时的编码比较特殊。

IE有一个选项"总是使用UTF-8发送URL"，当该选项有效时，IE将会对特殊字符进行UTF-8编码，同时进行URL编码。如果改选项无效，则使用默认编码"GBK"，并且不进行URL编码。但是，对于URL后面的参数，则总是不进行编码，相当于UTF-8选项无效。比如"中文.html?a=中文"，当UTF-8选项有效时，将发送链接"%e4%b8%ad%e6%96%87.html?a=\x4e\x2d\x65\x87"；而UTF-8选项无效时，将发送链接"\x4e\x2d\x65\x87.html?a=\x4e\x2d\x65\x87"。注意后者前面的"中文"两个字只有4个字节，而前者却有18个字节，这主要时URL编码的原因。

当web server（tomcat）接收到该链接时，将会进行URL解码，即去掉"%"，同时按照ISO8859-1编码（上面已经描述，可以使用URLEncoding来设置成其它编码）识别。上述例子的结果分别是"\ue4\ub8\uad\ue6\u96\u87.html?a=\u4e\u2d\u65\u87"和"\u4e\u2d\u65\u87.html?a=\u4e\u2d\u65\u87"，注意前者前面的"中文"两个字恢复成了6个字符。这里用"\u"，表示是unicode。

所以，由于客户端设置的不同，相同的链接，在服务器上得到了不同结果。这个问题不少人都遇到，却没有很好的解决办法。所以有的网站会建议用户尝试关闭UTF-8选项。不过，下面会描述一个更好的处理办法。

5.2. rewrite

熟悉的人都知道，apache有一个功能强大的rewrite模块，这里不描述其功能。需要说明的是该模块会自动将URL解码（去除%），即完成上述web server（tomcat）的部分功能。有相关文档介绍说可以使用[NE]参数来关闭该功能，但我试验并未成功，可能是因为版本（我使用的是apache 2.0.54）问题。另外，当参数中含有"?& "等符号的时候，该功能将导致系统得不到正常结果。

rewrite本身似乎完全是采用字节处理的方式，而不考虑字符串的编码，所以不会带来编码问题。

5.3. URLEncode.encode()

这是Java本身提供对的URL编码函数，完成的工作和上述UTF-8选项有效时浏览器所做的工作相似。值得说明的是，java已经不赞成不指定编码来使用该方法（deprecated）。应该在使用的时候增加编码指定。

当不指定编码的时候，该方法使用系统默认编码，这会导致软件运行结果得不确定。比如对于"中文"，当系统默认编码为"gb2312"时，结果是"%4e%2d%65%87"，而默认编码为"UTF-8"，结果却是"%e4%b8%ad%e6%96%87"，后续程序将难以处理。另外，这儿说的系统默认编码是由运行tomcat时的环境变量LC\_ALL和LANG等决定的，曾经出现过tomcat重启后就出现乱码的问题，最后才郁闷的发现是因为修改修改了这两个环境变量。

建议统一指定为"UTF-8"编码，可能需要修改相应的程序。

5.4. 一个解决方案

上面说起过，因为浏览器设置的不同，对于同一个链接，web server收到的是不同内容，而软件系统有无法知道这中间的区别，所以这一协议目前还存在缺陷。

针对具体问题，不应该侥幸认为所有客户的IE设置都是UTF-8有效的，也不应该粗暴的建议用户修改IE设置，要知道，用户不可能去记住每一个web server的设置。所以，接下来的解决办法就只能是让自己的程序多一点智能：根据内容来分析编码是否UTF-8。

比较幸运的是UTF-8编码相当有规律，所以可以通过分析传输过来的链接内容，来判断是否是正确的UTF-8字符，如果是，则以UTF-8处理之，如果不是，则使用客户默认编码（比如"GBK"），下面是一个判断是否UTF-8的例子，如果你了解相应规律，就容易理解。

public static boolean isValidUtf8(byte[] b,int aMaxCount){

       int lLen=b.length,lCharCount=0;

       for(int i=0;i<lLen && lCharCount<aMaxCount;++lCharCount){

              byte lByte=b[i++];//to fast operation, ++ now, ready for the following for(;;)

              if(lByte>=0) continue;//>=0 is normal ascii

              if(lByte<(byte)0xc0 || lByte>(byte)0xfd) return false;

              int lCount=lByte>(byte)0xfc?5:lByte>(byte)0xf8?4

                     :lByte>(byte)0xf0?3:lByte>(byte)0xe0?2:1;

              if(i+lCount>lLen) return false;

              for(int j=0;j<lCount;++j,++i) if(b[i]>=(byte)0xc0) return false;

       }

       return true;

}

相应地，一个使用上述方法的例子如下：

public static String getUrlParam(String aStr,String aDefaultCharset)

throws UnsupportedEncodingException{

       if(aStr==null) return null;

       byte[] lBytes=aStr.getBytes("ISO-8859-1");

       return new String(lBytes,StringUtil.isValidUtf8(lBytes)?"utf8":aDefaultCharset);

}

不过，该方法也存在缺陷，如下两方面：

l 没有包括对用户默认编码的识别，这可以根据请求信息的语言来判断，但不一定正确，因为我们有时候也会输入一些韩文，或者其他文字。

l 可能会错误判断UTF-8字符，一个例子是"学习"两个字，其GBK编码是" \xd1\xa7\xcf\xb0"，如果使用上述isValidUtf8方法判断，将返回true。可以考虑使用更严格的判断方法，不过估计效果不大。

有一个例子可以证明google也遇到了上述问题，而且也采用了和上述相似的处理方法，比如，如果在地址栏中输入"[http://www.google.com/search?hl=zh-CN&newwindow=1&q=学习](http://www.google.com/search?hl=zh-CN&newwindow=1&q=%E5%AD%A6%E4%B9%A0)"，google将无法正确识别，而其他汉字一般能够正常识别。

最后，应该补充说明一下，如果不使用rewrite规则，或者通过表单提交数据，其实并不一定会遇到上述问题，因为这时可以在提交数据时指定希望的编码。另外，中文文件名确实会带来问题，应该谨慎使用。

6. 其它

下面描述一些和编码有关的其他问题。

6.1. SecureCRT

除了浏览器和控制台与编码有关外，一些客户端也很有关系。比如在使用SecureCRT连接linux时，应该让SecureCRT的显示编码（不同的session，可以有不同的编码设置）和linux的编码环境变量保持一致。否则看到的一些帮助信息，就可能是乱码。

另外，mysql有自己的编码设置，也应该保持和SecureCRT的显示编码一致。否则通过SecureCRT执行sql语句的时候，可能无法处理中文字符，查询结果也会出现乱码。

对于Utf-8文件，很多编辑器（比如记事本）会在文件开头增加三个不可见的标志字节，如果作为mysql的输入文件，则必须要去掉这三个字符。（用linux的vi保存可以去掉这三个字符）。一个有趣的现象是，在中文windows下，创建一个新txt文件，用记事本打开，输入"连通"两个字，保存，再打开，你会发现两个字没了，只留下一个小黑点。

6.2. 过滤器

如果需要统一设置编码，则通过filter进行设置是个不错的选择。在filter class中，可以统一为需要的请求或者回应设置编码。参加上述setCharacterEncoding()。这个类apache已经给出了可以直接使用的例子SetCharacterEncodingFilter。

6.3. POST和GET

很明显，以POST提交信息时，URL有更好的可读性，而且可以方便的使用setCharacterEncoding()来处理字符集问题。但GET方法形成的URL能够更容易表达网页的实际内容，也能够用于收藏。

从统一的角度考虑问题，建议采用GET方法，这要求在程序中获得参数是进行特殊处理，而无法使用setCharacterEncoding()的便利，如果不考虑rewrite，就不存在IE的UTF-8问题，可以考虑通过设置URIEncoding来方便获取URL中的参数。

6.4. 简繁体编码转换

GBK同时包含简体和繁体编码，也就是说同一个字，由于编码不同，在GBK编码下属于两个字。有时候，为了正确取得完整的结果，应该将繁体和简体进行统一。可以考虑将UTF、GBK中的所有繁体字，转换为相应的简体字，BIG5编码的数据，也应该转化成相应的简体字。当然，仍旧以UTF编码存储。

例如，对于"语言 語言"，用UTF表示为"\xE8\xAF\xAD\xE8\xA8\x80 \xE8\xAA\x9E\xE8\xA8\x80"，进行简繁体编码转换后应该是两个相同的 "\xE8\xAF\xAD\xE8\xA8\x80>"。

**[Java与Unicode](http://hunnuxiaobo.iteye.com/blog/1022661)**

**博客分类：**

* [Java基础](http://hunnuxiaobo.iteye.com/category/77898)

[Java](http://www.iteye.com/blogs/tag/Java)[JVM](http://www.iteye.com/blogs/tag/JVM)[虚拟机](http://www.iteye.com/blogs/tag/èææº)[IDE](http://www.iteye.com/blogs/tag/IDE)[JDK](http://www.iteye.com/blogs/tag/JDK)

**Java与Unicode：**Java的class文件采用utf8的编码方式，JVM运行时采用utf16。Java的字符串是unicode编码的。总之，Java采用了unicode字符集，使之易于国际化。

Java支持哪些字符集：   
即Java能识别哪些字符集并对它进行正确地处理？查看Charset 类，最新的JDK支持160种字符集。可以通过static方法availableCharsets拿到所有Java支持的字符集。

Java代码



1. assertEquals(160, Charset.availableCharsets().size());
2. Set<String> charsetNames = Charset.availableCharsets().keySet();
3. assertTrue(charsetNames.contains("utf-8"));
4. assertTrue(charsetNames.contains("utf-16"));
5. assertTrue(charsetNames.contains("gb2312"));
6. assertTrue(Charset.isSupported("utf-8"));

需要在哪些时候注意编码问题？

1.从外部资源读取数据：  
这跟外部资源采取的编码方式有关，我们需要使用外部资源采用的字符集来读取外部数据：

Java代码



1. InputStream is = **new** FileInputStream("res/input2.data");
2. InputStreamReader streamReader = **new** InputStreamReader(is, "GB18030");

这里可以看到，我们采用了GB18030编码读取外部数据，通过查看streamReader的encoding可以印证：

Java代码



1. assertEquals("GB18030", streamReader.getEncoding());

正是由于上面我们为外部资源指定了正确的编码，当它转成char数组时才能正确地进行解码（GB18030 -> unicode）：

Java代码



1. **char**[] chars = **new** **char**[is.available()];
2. streamReader.read(chars, 0, is.available());

但我们经常写的代码就像下面这样：

Java代码



1. InputStream is = **new** FileInputStream("res/input2.data");
2. InputStreamReader streamReader = **new** InputStreamReader(is);

这时候InputStreamReader采用什么编码方式读取外部资源呢？Unicode？不是，这时候采用的编码方式是JVM的默认字符集，这个默认字符集在虚拟机启动时决定，通常根据语言环境和底层操作系统的 charset 来确定。可以通过以下方式得到JVM的默认字符集：

Java代码



1. Charset.defaultCharset();

为什么要这样？因为我们从外部资源读取数据，而外部资源的编码方式通常跟操作系统所使用的字符集一样，所以采用这种默认方式是可以理解的。  
好吧，那么我通过我的IDE Ideas创建了一个文件，并以JVM默认的编码方式从这个文件读取数据，但读出来的数据竟然是乱码。为何？呵呵，其实是因为通过Ideas创建的文件是以utf-8编码的。要得到一个JVM默认编码的文件，通过手工创建一个txt文件试试吧。

2.字符串和字节数组的相互转换  
我们通常通过以下代码把字符串转换成字节数组：

Java代码



1. "string".getBytes();

但你是否注意过这个转换采用的编码呢？其实上面这句代码跟下面这句是等价的：

Java代码



1. "string".getBytes(Charset.defaultCharset());

也就是说它根据JVM的默认编码（而不是你可能以为的unicode）把字符串转换成一个字节数组。  
反之，如何从字节数组创建一个字符串呢？

Java代码



1. **new** String("string".getBytes());

同样，这个方法使用平台的默认字符集解码字节的指定数组（这里的解码指从一种字符集到unicode）。

字符串编码迷思：

Java代码



1. **new** String(input.getBytes("ISO-8859-1"), "GB18030")

上面这段代码代表什么？有人会说： “把input字符串从ISO-8859-1编码方式转换成GB18030编码方式”。如果这种说法正确，那么又如何解释我们刚提到的java字符串都采用unicode编码呢？  
这种说法不仅是欠妥的，而且是大错特错的，让我们一一来分析，其实事实是这样的：我们本应该用GB18030的编码来读取数据并解码成字符串，但结果却采用了ISO-8859-1的编码，导致生成一个错误的字符串。要恢复，就要先把字符串恢复成原始字节数组，然后通过正确的编码GB18030再次解码成字符串（即把以GB18030编码的数据转成unicode的字符串）。注意，字符串永远都是unicode编码的。  
但编码转换并不是负负得正那么简单，这里我们之所以可以正确地转换回来，是因为 ISO8859-1 是单字节编码，所以每个字节被按照原样 转换为 String ，也就是说，虽然这是一个错误的转换，但编码没有改变，所以我们仍然有机会把编码转换回来！

总结：   
所以，我们在处理java的编码问题时，要分清楚三个概念：Java采用的编码：unicode，JVM平台默认字符集和外部资源的编码。

程序示例：

Java代码



1. **public** **static** **void** main(String args[])
2. {
3. System.out.println("default charset : "+Charset.defaultCharset());
4. String str = "abc你好";//string with UTF-8 charset
6. **byte**[] bytes = str.getBytes(Charset.forName("UTF-8"));//convert to byte array with UTF-8 encode
7. **for** (**byte** b : bytes)
8. {
9. System.out.print(b + " ");
10. }
11. System.out.println();
12. **try**
13. {
14. String str1 = **new** String(bytes, "UTF-8");//to UTF-8 string
15. String str2 = **new** String(bytes, "ISO-8859-1");//to ISO-8859-1 string
16. String str3 = **new** String(bytes, "GBK");//to GBK string
18. System.out.println(str1);//abc你好
19. System.out.println(str2);//abc??????
20. System.out.println(str3);//abc浣犲ソ
22. System.out.println();
23. **byte**[] bytes2 = str2.getBytes(Charset.forName("ISO-8859-1"));
24. **for** (**byte** b : bytes2)
25. {
26. System.out.print(b + " ");
27. }
28. System.out.println();
29. String str22 = **new** String(bytes2, "UTF-8");
30. System.out.println(str22);//abc你好
32. System.out.println();
33. **byte**[] bytes3 = str3.getBytes(Charset.forName("GBK"));
34. **for** (**byte** b : bytes3)
35. {
36. System.out.print(b + " ");
37. }
38. System.out.println();
39. String str33 = **new** String(bytes3, "UTF-8");
40. System.out.println(str33);//abc你好
41. } **catch** (UnsupportedEncodingException e)
42. {
43. e.printStackTrace();
44. }
45. }

运行结果：

Html代码



1. default charset : GBK
2. 97 98 99 -28 -67 -96 -27 -91 -67
3. abc你好
4. abc??????
5. abc浣犲ソ
7. 97 98 99 -28 -67 -96 -27 -91 -67
8. abc你好
10. 97 98 99 -28 -67 -96 -27 -91 -67
11. abc你好