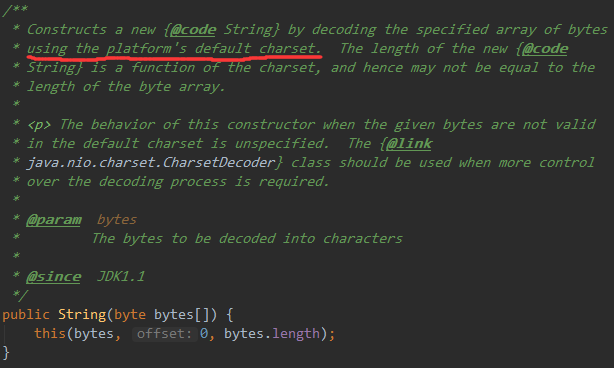
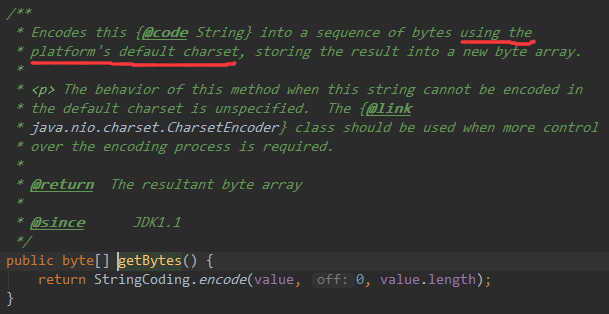
昨天，同事遇见一个挺奇怪的问题：同样的代码在同事A的电脑上可以跑通，在同事B的电脑上却跑不通，原因是两人程序的启动运行时的默认字符编码不同，一个是GBK另一个是UTF-8，恰好借此机会让我也巩固和梳理了一下字符串和字符编码的关系。

先来看两个JDK的API：

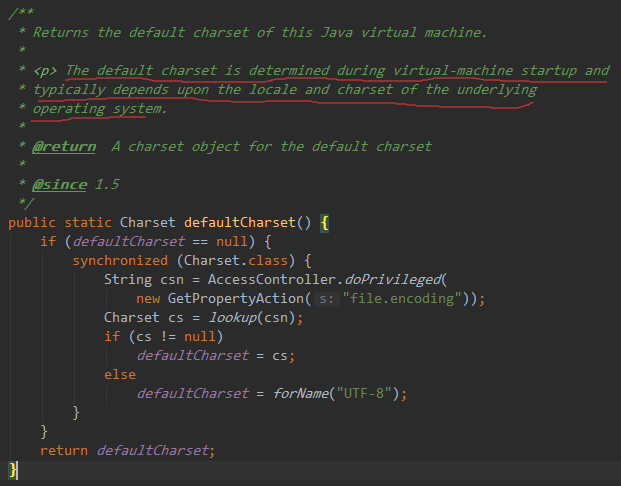




注意红色标注部分，使用平台默认的字符集，什么是平台的默认字符集呢？

通过搜索引擎得到的结论应该都是千篇一律的Windows系统默认的字符集是GBK，这个结果可能对也可能不对，正确的结论应该是这样的：**平台默认的字符集是指JVM启动时通过参数-Dfile.coding=指定的字符编码或者本地系统的字符集。**

如何验证我的结论是正确的呢，我们跟着上面截图的方法实现下去看看：



那么说日常我们使用IDE开发时，可以通过设置IDE来指定JVM启动时的字符编码或者IDE默认有指定字符集，这种情况下平台默认的字符编码可能不是系统的默认字符集，**因此对于字符编码的问题咱们不能武断的说使用的是平台默认字符集，比如Windows系统就是GBK，需要具体情况具体分析。最简单也通用的方法就是不论在开发过程中还是上线部署时都设置JVM启动的字符编码是”UTF-8”。**

下面对字符编码也做一个总结：

计算机系统只能理解处理01这样的二进制数据，为了将我们日常使用的字符转换成计算机能理解的数据就出现了码表，码表的作用就是将数字和字符做一个对应的表格。我们常用的码表有：

**ASCII码**

美国制定的一套字符编码，对英语字符与二进制位之间的关系，做了统一规定。ASCII码一共规定了128个字符的编码，比如空格"SPACE"是32（二进制00100000），大写的字母A是65（二进制01000001）。这128个符号（包括32个不能打印出来的控制符号），只占用了一个字节的后面7位，最前面的1位统一规定为0。

**非ASCII编码**

英语用128个符号编码就够了，但是用来表示其他语言，128个符号是不够的。比如，在法语中，字母上方有注音符号，它就无法用ASCII码表示。于是，一些欧洲国家就决定，利用字节中闲置的最高位编入新的符号。比如，法语中的é的编码为130（二进制10000010）。但是，这里又出现了新的问题。不同的国家有不同的字母，因此，哪怕它们都使用256个符号的编码方式，代表的字母却不一样。至于亚洲国家的文字，使用的符号就更多了，汉字就多达10万左右。一个字节只能表示256种符号，肯定是不够的，就必须使用多个字节表达一个符号。比如，简体中文常见的编码方式是GB2312，使用两个字节表示一个汉字，所以理论上最多可以表示256x256=65536个符号。

**GB2312**

中国国家标准简体中文字符集，由中国国家标准总局发布，1981 年 5 月 1 日实施。GB 2312 编码通行于中国大陆；新加坡等地也采用此编码。中国大陆几乎所有的中文系统和国际化的软件都支持 GB 2312。GB 2312 标准共收录 6763 个汉字，其中一级汉字 3755 个，二级汉字 3008 个；同时收录了包括拉丁字母、希腊字母、日文平假名及片假名字母、俄语西里尔字母在内的 682 个字符。GB 2312 对任意一个图形字符都采用两个字节表示，并对所收汉字进行了“分区”处理，每区含有 94 个汉字／符号，分别对应第一字节和第二字节。这种表示方式也称为区位码。

**GBK**

GBK 即汉字内码扩展规范，K 为汉语拼音 Kuo Zhan（扩展）中“扩”字的声母。GBK 向下与 GB 2312 完全兼容，向上支持 ISO 10646 国际标准，GBK 采用双字节表示。

**Unicode**

将世界上所有的符号都纳入其中。每一个符号都给予一个独一无二的编码，那么乱码问题就会消失。这就是Unicode，就像它的名字都表示的，这是一种所有符号的编码。

Unicode是一个很大的集合，现在的规模可以容纳100多万个符号。

但是Unicode只是一个符号集，它只规定了符号的二进制代码，却没有规定这个二进制代码应该如何存储。有两个严重的问题，第一个问题是，如何才能区别Unicode和ASCII？计算机怎么知道三个字节表示一个符号，而不是分别表示三个符号呢？第二个问题是，我们已经知道，英文字母只用一个字节表示就够了，如果Unicode统一规定，每个符号用三个或四个字节表示，那么每个英文字母前都必然有二到三个字节是0，这对于存储来说是极大的浪费，文本文件的大小会因此大出二三倍，这是无法接受的。

它们造成的结果是：1）出现了Unicode的多种存储方式，也就是说有许多种不同的二进制格式，可以用来表示Unicode。2）Unicode在很长一段时间内无法推广，直到互联网的出现。

**UTF-8**

互联网的普及，强烈要求出现一种统一的编码方式。UTF-8就是在互联网上使用最广的一种Unicode的实现方式。其他实现方式还包括UTF-16（字符用两个字节或四个字节表示）和UTF-32（字符用四个字节表示），不过在互联网上基本不用。**重复一遍，这里的关系是，UTF-8是Unicode的实现方式之一。**

UTF-8最大的一个特点，就是它是一种变长的编码方式。它可以使用1~4个字节表示一个符号，根据不同的符号而变化字节长度。

UTF-8的编码规则很简单，只有二条：

1）对于单字节的符号，字节的第一位设为0，后面7位为这个符号的unicode码。因此对于英语字母，UTF-8编码和ASCII码是相同的。

2）对于n字节的符号（n>1），第一个字节的前n位都设为1，第n+1位设为0，后面字节的前两位一律设为10。剩下的没有提及的二进制位，全部为这个符号的unicode码。

由此可见GB类的汉字编码与后文的Unicode和UTF-8是毫无关系的。因此会照成中文的乱码。

有了码表就可以实现**字符的输入输出，**我们来梳理几个我们常见的输入输出场景：

**文件**

我们在文件中输入的文字信息，最终都将按照01的格式转换成码表存储在计算机的储存媒体中比如硬盘，那么文本编辑文件是如何转换成01的格式呢？是根据文本存储时指定的编码格式来的，比如存储文件时选定编码格式为GBK则按照GBK的码表将你输入的文字信息转换成二进制格式存储，选定为UTF-8时就按UTF-8的码表存储，**程序在读取文件时将文件中的二进制内容输入到内存中，但是想要正确的显示成字符串则需要使用正确的码表来打开**，这就涉及到了本文最开头提到的String的构造方法，通过构造方法中传入正确的字符编码将字节数组转换成字符串。

**网络**

Http协议是目前通用的网络传输协议，主要的请求方法是GET和POST，POST是将请求内容放在请求体中进行传输，这种请求方式只需确认请求发送方和请求接收方的字符编码格式相同即可避免乱码的问题，这里就不再赘诉，我们主要是统计一下不同的GET请求实现方式对中文编码的不同。

**URL编码**

因为网络标准[RFC 1738](http://www.ietf.org/rfc/rfc1738.txt" \t "http://www.ruanyifeng.com/blog/2010/02/_blank)做了硬性规定：只有字母和数字[0-9a-zA-Z]、一些特殊符号"$-\_.+!\*'(),"[不包括双引号]、以及某些保留字，才可以不经过编码直接用于URL。

这意味着，如果URL中有汉字，就必须编码后使用。但是麻烦的是，RFC 1738没有规定具体的编码方法，而是交给应用程序（浏览器）自己决定。这导致"URL编码"成为了一个混乱的领域。

**情况1：网址路径中包含汉字**

在IE和Firefox中测试，结论是网址路径的编码，用的是utf-8编码。

**情况2：请求参数中包含汉字**

注意，此时属于参数中包含字符串，不属于网址路径，不要与情况1混淆。现在的浏览器基本都会采用UTF-8统一编码

**情况3：Get方法生成的URL包含汉字**

前面说的是直接输入网址的情况，假如直接用Get或Post方法发出HTTP请求。这时的编码方法由网页的编码决定，也就是由HTML源码中字符集的设定决定。<meta http-equiv="Content-Type" content="text/html;charset=xxxx">

**情况4：Ajax调用的URL包含汉字**

前面三种情况都是由浏览器发出HTTP请求，最后一种情况则是由Javascript生成HTTP请求，也就是Ajax调用。在这种情况下，IE和Firefox的处理方式完全不一样。在Ajax调用中，IE总是采用GB2312编码（操作系统的默认编码），而Firefox总是采用utf-8编码。对于不同的操作系统不同的浏览字符编码也不同这个问题让人感到头痛，在开发过程中，我们可以使用Javascript先对URL编码，然后再向服务器提交，不要给浏览器插手的机会。因为Javascript的输出总是一致的，所以就保证了服务器得到的数据是格式统一的。

Javascript函数：encodeURI()是Javascript中用来对URL编码的函数。

它着眼于对整个URL进行编码，因此除了常见的符号以外，对其他一些在网址中有特殊含义的符号"; / ? : @ & = + $ , #"，也不进行编码。编码后，它输出符号的utf-8形式，并且在每个字节前加上%。

**Tomcat的字符编码**

请求数据通过网络传输首先到达HTTP服务器，HTTP服务器解析传输过来的字节码封装成Request对象，假如字节码使用的不正确这又会出现乱码的问题。

tomcat内部对于http request，有两种字符编码的配置：

****get方式的字符编码****

第一种情况：get和post的编码保持一致，post方式的编码是什么，get方式的编码就是什么。

 server.xml中进行如下配置的话，get方式的字符编码和post方式的字符编码保持一致。

<Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443" useBodyEncodingForURI="true"/>

第二种情况：不指定useBodyEncodingForURI或者useBodyEncodingForURI="false"。

这时get和post的字符编码各自设置，互相没有关系。配置方法如下：

通过server.xml文件的URIEncoding进行设置，如果没有配置URIEncoding，那么用缺省的ISO-8859-1。

****post方式的字符编码****

在工程的web.xml中添加Spring的过滤器

<filter>   
 <filter-name>encodingFilter</filter-name>   
 <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>   
 <init-param>   
 <param-name>encoding</param-name>   
 <param-value>UTF-8</param-value>   
 </init-param>   
 <init-param>   
 <param-name>forceEncoding</param-name>   
 <param-value>true</param-value>   
 </init-param>   
</filter>

**如果没有进行配置**，那么从http header中取出content-type,然后从content-type的值中取出charset的值，用charset的值作为post的字符编码。**如果从http header中没有取到content-type或者charset**，那么，就使用缺省的ISO-8859-1。