目录

你已经读了 25 %

1. 关于字符编码，你所需要知道的（ASCII,Unicode,Utf-8,GB2312… ）
1.1. 还是得从ASCII码説起
1.2. OEM字符集的衍生
1.3. 多字节字符集（MBCS）和中文字符集
1.4. ANSI标準、国家标準、ISO标準
1.5. Unicode的出现
1.5.1. Unicode字符集概述
1.5.2. 编码系统的变化
1.5.3. 常见的Unicode编码
1.5.3.1. UCS-2/UTF-16
1.5.3.2. UTF-8
1.5.3.3. GB18030
1.5.4. Unicode相关的常见问题
1.5.5. 乱码问题
1.5.6. 必要的术语解释
2. Unicode 和 UTF-8 有什么区别？

关于字符编码，你所需要知道的（ASCII,Unicode,Utf-8,GB2312… ）

发表于 2019-04-15 | 更新于 2020-01-04 | Java

字数总计: 8.8k | 閱读时长: 27 分鐘 | 閱读量: 620 | 评论数: 6

关于字符编码，你所需要知道的（ASCII,Unicode,Utf-8,GB2312… ）

> 转自 Kevin Yang

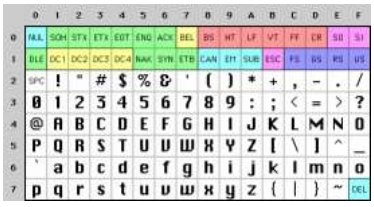
字符编码的问题看似很小，经常被技术人员忽视，但是很容易导致一些莫名其妙的问题。这里总结了一下字符编码的一些普及性的知识，希望对大家有所帮助。

还是得从ASCII码説起

説到字符编码，不得不説ASCII码的简史。计算机一开始发明的时候是用来解决数字计算的问题，后来人们发现，计算机还可以做更多的事，例如文本处理。但由于计算机只识"数"，因此人们必须告诉计算机哪个数字来代表哪个特定字符，例如65代表字母'A'，66代表字母'B'，以此类推。但是**计算机之间字符-数字的对应关係必须得一致，否则就会造成同一段数字在不同计算机上显示出来的字符不一样**。因此美国国家标準协会ANSI制定了一个标準，规定了常用字符的集合以及每个字符对应的编号，这就是ASCII字符集（Character Set），也称ASCII码。

当时的计算机普遍使用8比特字节作为最小的存储和处理单元，加之当时用到的字符也很少，26个大小写英文字母还有数字再加上其他常用符号，也不到100个，因此使用7个比特位就可以高效的存储和处理ASCII码，剩下最高位1比特被用作一些通讯系统的奇偶校验。

> 注意，字节代表系统能够处理的最小单位，不一定是8比特。只是现代计算机的事实标準就是用8比特来代表一个字节。在很多技术规格文献中，为了避免产生歧义，更倾向于使用8位组（Octet）而不是字节（Byte）这个术语来强调8个比特的二进制流。下文中为了便于理解，我会延用大家熟悉的"字节"这个概念。



ASCII字符集由95个可打印字符（0x20-0x7E）和33个控制字符（0x00-0x19，0x7F）组成。可打印字符用于显示在输出设备上，例如荧屏或者打印纸上，控制字符用于向计算机发出一些特殊指令，例如0x07会让计算机发

JerryC

目录
你已经读了 25 %
1. 关于字符编码，你所需要知道的（ASCII,Unicode,Utf-8,GB2312… ）
1.1. 还是得从ASCII码説起
1.2. OEM字符集的衍生
1.3. 多字节字符集（MBCS）和中文字符集
1.4. ANSI标準、国家标準、ISO标準
1.5. Unicode的出现
1.5.1. Unicode字符集概述
1.5.2. 编码系统的变化
1.5.3. 常见的Unicode编码
1.5.3.1. UCS-2/UTF-16
1.5.3.2. UTF-8
1.5.3.3. GB18030
1.5.4. Unicode相关的常见问题
1.5.5. 乱码问题
1.5.6. 必要的术语解释
2. Unicode 和 UTF-8 有什么区别？

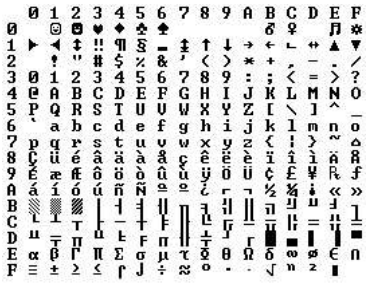
出哔的一声，0x00通常用于指示字符串的结束，0x0D和0x0A用于指示打印机的打印针头退到行首（回车）并移到下一行（换行）。

那时候的字符编解码系统非常简单，就是简单的查表过程。例如将字符序列编码为二进制流写入存储设备，只需要在ASCII字符集中依次找到字符对应的字节，然后直接将该字节写入存储设备即可。解码二进制流的过程也是类似。

OEM字符集的衍生

当计算机开始发展起来的时候，人们逐渐发现，ASCII字符集里那可怜的128个字符已经不能再满足他们的需求了。人们就在想，一个字节能够表示的数字（编号）有256个，而ASCII字符只用到了0x00~0x7F，也就是佔用了前128个，后面128个数字不用白不用，因此很多人打起了后面这128个数字的主意。可是问题在于，很多人同时有这样的想法，但是大家对于0x80-0xFF这后面的128个数字分别对应什么样的字符，却有各自的想法。这就导致了当时销往世界各地的机器上出现了大量各式各样的OEM字符集。

下面这张表是IBM-PC机推出的其中一个OEM字符集，字符集的前128个字符和ASCII字符集的基本一致（为什么説基本一致呢，是因为前32个控制字符在某些情况下会被IBM-PC机当作可打印字符解释），后面128个字符空间加入了一些欧洲国家用到的重音字符，以及一些用于画线条画的字符。



事实上，大部分OEM字符集是兼容ASCII字符集的，也就是説，大家对于0x00~~0x7F这个範围的解释基本是相同的，而对于后半部分0x800~~xFF的解释却不一定相同。甚至有时候同样的字符在不同OEM字符集中对应的字节也是不同的。

不同的OEM字符集导致人们无法跨机器交流各种文档。例如职员甲发了一封简歷résumés给职员乙，结果职员乙看到的却是r?sum?s，因为é字符在职员甲机器上的OEM字符集中对应的字节是0x82，而在职员乙的机器上，由于使用的OEM字符集不同，对0x82字节解码后得到的字符却是?。

多字节字符集（MBCS）和中文字符集

上面我们提到的字符集都是基于单字节编码，也就是説，一个字节翻译成一个字符。这对于拉丁语系国家来説可能没有什么问题，因为他们通过扩展第8个比特，就可以得到256个字符了，足够用了。但是对于亚洲国家来説，256个字符是远远不够用的。因此这些国家的人为了用上电脑，又要保持和ASCII字符集的兼容，就发明了多字节编码方式，相应的字符集就称为多字节字符集。例如中国使用的就是双字节字符集编码（DBCS，Double Byte Character Set）。

对于单字节字符集来説，代码页中只需要有一张码錶即可，上面记录着256个数字代表的字符。程序只需要做简单的查表操作就可以完成编解码的过程。

> 代码页是字符集编码的具体实现，你可以把他理解为一张"字符-字节"映射表，通过查表实现"字符-字节"的翻译。下面会有更详细的描述。

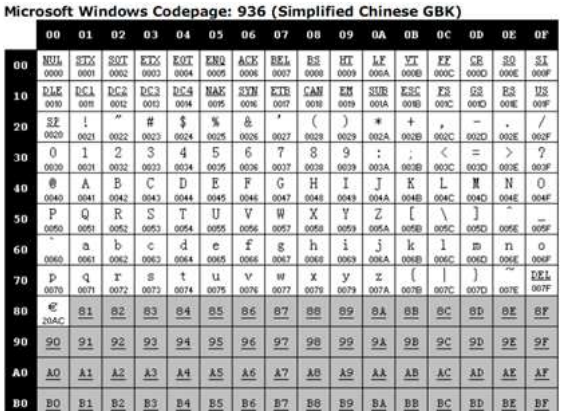
而对于多字节字符集，代码页中通常会有很多码錶。那么程序怎么知道该使用哪张码錶去解码二进制流呢？答案是，**根据第一个字节来选择不同的码錶进行解析**。

例如目前最常用的中文字符集GB2312，涵盖了所有简体字符以及一部分其他字符；GBK（K代表扩展的意思）则在GB2312的基础上加入了对繁体字符等其他非简体字符（GB18030字符集不是双字节字符集，我们在讲Unicode的时候会提到）。这两个字符集的字符都是使用1-2个字节来表示。Windows系统採用936代码页来实

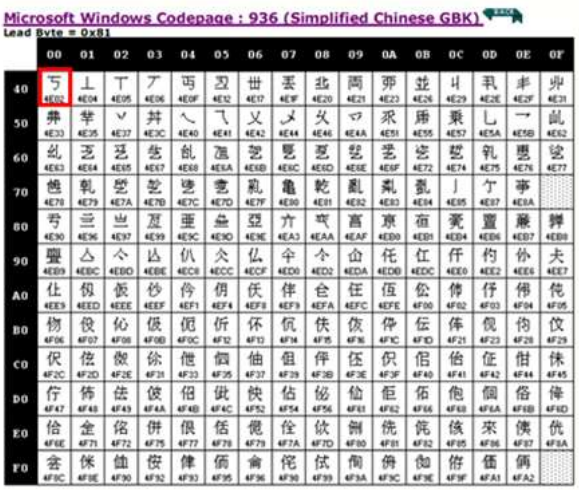
目录

你已经读了 25 %

1. 关于字符编码，你所需要知道的（ASCII,Unicode,Utf-8,GB2312… ）
1.1. 还是得从ASCII码説起
1.2. OEM字符集的衍生
1.3. 多字节字符集（MBCS）和中文字符集
1.4. ANSI标準、国家标準、ISO标準
1.5. Unicode的出现
1.5.1. Unicode字符集概述
1.5.2. 编码系统的变化
1.5.3. 常见的Unicode编码
1.5.3.1. UCS-2/UTF-16
1.5.3.2. UTF-8
1.5.3.3. GB18030
1.5.4. Unicode相关的常见问题
1.5.5. 乱码问题
1.5.6. 必要的术语解释
2. Unicode 和 UTF-8 有什么区别？



当字节的高位是1的时候，确切的説，当第一个字节位于0x81–0xFE之间时，根据第一个字节不同找到代码页中的相应的码錶，例如当第一个字节是0x81，那么对应936中的下面这张码錶：



（关于936代码页中完整的码錶信息，参见MSDN：http://msdn.microsoft.com/en-us/library/cc194913%28v=MSDN.10%29.aspx. ）

按照936代码页的码錶，当程序遇到连续字节流0x81 0x40的时候，就会解码为“丂”字符。

ANSI标準、国家标準、ISO标準

不同ASCII衍生字符集的出现，让文档交流变得非常困难，因此各种组织都陆续进行了标準化流程。例如美国ANSI组织制定了ANSI标準字符编码（注意，**我们现在通常説到ANSI编码，通常指的是平台的默认编码，例如英文操作系统中是ISO-8859-1，中文系统是GBK**），ISO组织制定的各种ISO标準字符编码，还有各国也会制定一些国家标準字符集，例如中国的GBK，GB2312和GB18030。

操作系统在发佈的时候，通常会往机器里预装这些标準的字符集还有平台专用的字符集，这样只要你的文档是使用标準字符集编写的，通用性就比较高了。例如你用GB2312字符集编写的文档，在中国大陆内的任何机器上都能正确显示。同时，我们也可以在一台机器上閱读多个国家不同语言的文档了，前提是本机必须安装该文档使用的字符集。

Unicode的出现

虽然通过使用不同字符集，我们可以在一台机器上查閱不同语言的文档，但是我们仍然无法解决一个问题：**在一份文档中显示所有字符**。为了解决这个问题，我们需要一个全人类达成共识的巨大的字符集，这就是Unicode字符集。

Unicode字符集概述

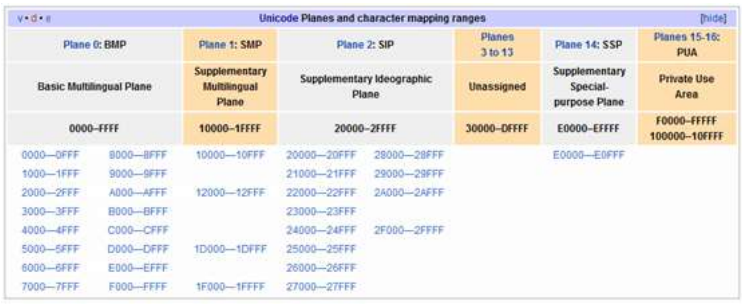
JerryC

目录

你已经读了 25 %

1. 关于字符编码，你所需要知道的（ASCII,Unicode,Utf-8,GB2312… ）
1.1. 还是得从ASCII码説起
1.2. OEM字符集的衍生
1.3. 多字节字符集（MBCS）和中文字符集
1.4. ANSI标準、国家标準、ISO标準
1.5. Unicode的出现
1.5.1. Unicode字符集概述
1.5.2. 编码系统的变化
1.5.3. 常见的Unicode编码
1.5.3.1. UCS-2/UTF-16
1.5.3.2. UTF-8
1.5.3.3. GB18030
1.5.4. Unicode相关的常见问题
1.5.5. 乱码问题
1.5.6. 必要的术语解释
2. Unicode 和 UTF-8 有什么区别？

2^{16}=65536个字符码空间。



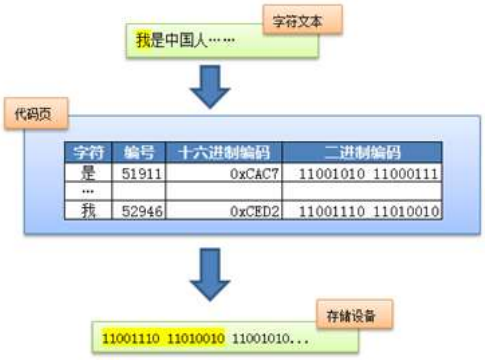
Unicode Planes and character mapping ranges [hide]

Plane 0: BMP		Plane 1: SMP	Plane 2: SIP		Planes 3 to 13	Plane 14: SSP	Planes 15-16: PUA
Basic Multilingual Plane		Supplementary Multilingual Plane	Supplementary Ideographic Plane		Unassigned	Supplementary Special-purpose Plane	Private Use Area
0000–FFFF		10000–1FFFF	20000–2FFFF		30000–DFFFF	E0000–EFFFF	F0000–FFFFF 100000–10FFFF
0000—0FFF	8000—8FFF	10000—10FFF	20000—20FFF	28000—28FFF		E0000—E0FFF	
1000—1FFF	9000—9FFF		21000—21FFF	29000—29FFF			
2000—2FFF	A000—AFFF	12000—12FFF	22000—22FFF	2A000—2AFFF			
3000—3FFF	B000—BFFF		23000—23FFF				
4000—4FFF	C000—CFFF		24000—24FFF	2F000—2FFFF			
5000—5FFF	D000—DFFF	1D000—1DFFF	25000—25FFF				
6000—6FFF	E000—EFFF		26000—26FFF				
7000—7FFF	F000—FFFF	1F000—1FFFF	27000—27FFF				

其中第0个层面BMP，基本涵盖了当今世界用到的所有字符。其他的层面要么是用来表示一些远古时期的文字，要么是留作扩展。我们平常用到的Unicode字符，一般都是位于BMP层面上的。目前Unicode字符集中尚有大量字符空间未使用。

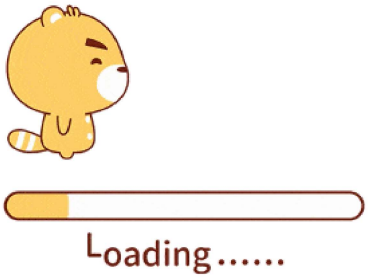
编码系统的变化

在Unicode出现之前，所有的字符集都是和具体编码方案绑定在一起的，都是直接将字符和最终字节流绑定死了，例如ASCII编码系统规定使用7比特来编码ASCII字符集；GB2312以及GBK字符集，限定了使用最多2个字节来编码所有字符，并且规定了字节序。这样的编码系统通常用简单的查表，也就是通过代码页就可以直接将字符映射为存储设备上的字节流了。例如下面这个例子：



这种方式的缺点在于，字符和字节流之间耦合得太紧密了，从而限定了字符集的扩展能力。假设以后火星人入住地球了，要往现有字符集中加入火星文就变得很难甚至不可能了，而且很容易破坏现有的编码规则。

因此Unicode在设计上考虑到了这一点，将字符集和字符编码方案分离开。



也就是説，**虽然每个字符在Unicode字符集中都能找到唯一确定的编号（字符码，又称Unicode码），但是决定最终字节流的却是具体的字符编码**。例如同样是对Unicode字符“A”进行编码，UTF-8字符编码得到的字节流

JerryC

目录
你已经读了 25 %
1. 关于字符编码，你所需要知道的（ASCII,Unicode,Utf-8,GB2312… ）
1.1. 还是得从ASCII码説起
1.2. OEM字符集的衍生
1.3. 多字节字符集（MBCS）和中文字符集
1.4. ANSI标準、国家标準、ISO标準
1.5. Unicode的出现
1.5.1. Unicode字符集概述
1.5.2. 编码系统的变化
1.5.3. 常见的Unicode编码
1.5.3.1. UCS-2/UTF-16
1.5.3.2. UTF-8
1.5.3.3. GB18030
1.5.4. Unicode相关的常见问题
1.5.5. 乱码问题
1.5.6. 必要的术语解释
2. Unicode 和 UTF-8 有什么区别？

常见的Unicode编码

UCS-2/UTF-16

如果要我们来实现Unicode字符集中BMP字符的编码方案，我们会怎么实现？由于BMP层面上有2^{16}=65536个字符码，因此我们只需要两个字节就可以完全表示这所有的字符了。

举个例子，“中”的Unicode字符码是0x4E2D(01001110 00101101)，那么我们可以编码为01001110 00101101（大端）或者00101101 01001110（小端）。

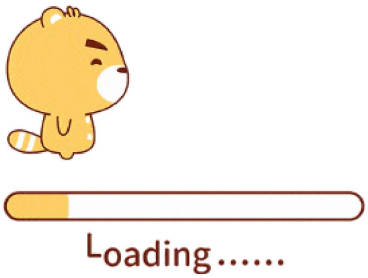
UCS-2和UTF-16对于BMP层面的字符均是使用2个字节来表示，并且编码得到的结果完全一致。不同之处在于，**UCS-2最初设计的时候只考虑到BMP字符，因此使用固定2个字节长度，也就是説，他无法表示Unicode其他层面上的字符，而UTF-16为了解除这个限制，支持Unicode全字符集的编解码，採用了变长编码，最少使用2个字节，如果要编码BMP以外的字符，则需要4个字节结对**，这里就不讨论那么远，有兴趣可以参考维基百科：UTF-16/UCS-2。

Windows从NT时代开始就採用了UTF-16编码，很多流行的编程平台，例如.Net，Java，Qt还有Mac下的Cocoa等都是使用UTF-16作为基础的字符编码。例如代码中的字符串，在内存中相应的字节流就是用UTF-16编码过的。

UTF-8

UTF-8应该是目前应用最广泛的一种Unicode编码方案。由于UCS-2/UTF-16对于ASCII字符使用两个字节进行编码，存储和处理效率相对低下，并且由于ASCII字符经过UTF-16编码后得到的两个字节，高字节始终是0x00，很多C语言的函数都将此字节视为字符串末尾从而导致无法正确解析文本。因此一开始推出的时候遭到很多西方国家的抵触，大大影响了Unicode的推行。后来聪明的人们发明了UTF-8编码，解决了这个问题。

UTF-8编码方案採用1-4个字节来编码字符，方法其实也非常简单。



（上图中的x代表Unicode码的低8位，y代表高8位）

对于ASCII字符的编码使用单字节，和ASCII编码一摸一样，这样所有原先使用ASCII编解码的文档就可以直接转到UTF-8编码了。对于其他字符，则使用2-4个字节来表示，其中，首字节前置1的数目代表正确解析所需要的字节数，剩余字节的高2位始终是10。例如首字节是1110yyyy，前置有3个1，説明正确解析总共需要3个字节，需要和后面2个以10开头的字节结合才能正确解析得到字符。

关于UTF-8的更多信息，参考维基百科：UTF-8。

GB18030

任何能够将Unicode字符映射为字节流的编码都属于Unicode编码。中国的GB18030编码，覆盖了Unicode所有的字符，因此也算是一种Unicode编码。只不过他的编码方式并不像UTF-8或者UTF-16一样，将Unicode字符的编号通过一定的规则进行转换，而只能通过查表的手段进行编码。

关于GB18030的更多信息，参考：GB18030。

JerryC

目录
你已经读了 25 %
1. 关于字符编码，你所需要知道的（ASCII,Unicode,Utf-8,GB2312… ）
1.1. 还是得从ASCII码説起
1.2. OEM字符集的衍生
1.3. 多字节字符集（MBCS）和中文字符集
1.4. ANSI标準、国家标準、ISO标準
1.5. Unicode的出现
1.5.1. Unicode字符集概述
1.5.2. 编码系统的变化
1.5.3. 常见的Unicode编码
1.5.3.1. UCS-2/UTF-16
1.5.3.2. UTF-8
1.5.3.3. GB18030
1.5.4. Unicode相关的常见问题
1.5.5. 乱码问题
1.5.6. 必要的术语解释
2. Unicode 和 UTF-8 有什么区别？

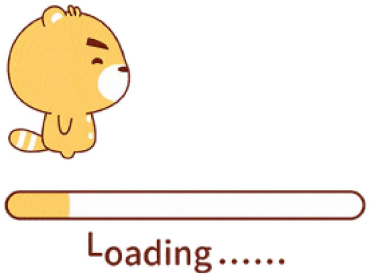
- Unicode是两个字节吗？
  Unicode只是定义了一个庞大的、全球通用的字符集，併为每个字符规定了唯一确定的编号，具体存储为什么样的字节流，取决于字符编码方案。推荐的Unicode编码是UTF-16和UTF-8。
- 带签名的UTF-8指的是什么意思？
  带签名指的是字节流以BOM标记开始。很多软件会"智能"的探测当前字节流使用的字符编码，这种探测过程出于效率考虑，通常会提取字节流前面若干个字节，看看是否符合某些常见字符编码的编码规则。由于UTF-8和ASCII编码对于纯英文的编码是一样的，无法区分开来，因此通过在字节流最前面添加BOM标记可以告诉软件，当前使用的是Unicode编码，判别成功率就十分準确了。但是需要注意，不是所有软件或者程序都能正确处理BOM标记，例如PHP就不会检测BOM标记，直接把它当普通字节流解析了。因此如果你的PHP文件是採用带BOM标记的UTF-8进行编码的，那么有可能会出现问题。
- Unicode编码和以前的字符集编码有什么区别？
  早期字符编码、字符集和代码页等概念都是表达同一个意思。例如GB2312字符集、GB2312编码，936代码页，实际上説的是同个东西。但是对于Unicode则不同，Unicode字符集只是定义了字符的集合和唯一编号，Unicode编码，则是对UTF-8、UCS-2/UTF-16等具体编码方案的统称而已，并不是具体的编码方案。所以当需要用到字符编码的时候，你可以写gb2312，codepage936，utf-8，utf-16，但请不要写unicode（看过别人在网页的meta标籤里头写charset=unicode，有感而发）。

乱码问题

乱码指的是程序显示出来的字符文本无法用任何语言去解读。一般情况下会包含大量?。乱码问题是所有计算机用户或多或少会遇到的问题。**造成乱码的原因就是因为使用了错误的字符编码去解码字节流，因此当我们在思考任何跟文本显示有关的问题时，请时刻保持清醒：当前使用的字符编码是什么**。只有这样，我们才能正确分析和处理乱码问题。

例如最常见的网页乱码问题。如果你是网站技术人员，遇到这样的问题，需要检查以下原因：

- 服务器返回的响应头Content-Type没有指明字符编码
- 网页内是否使用META HTTP-EQUIV标籤指定了字符编码
- 网页文件本身存储时使用的字符编码和网页声明的字符编码是否一致



注意，网页解析的过程如果使用的字符编码不正确，还可能会导致脚本或者样式表出错。具体细节可以参考我以前写过的文章：文档字符集导致的脚本错误和Asp.Net页面的编码问题。

不久前看到某技术论坛有人反馈，WinForm程序使用Clipboard类的GetData方法去访问剪切板中的HTML内容时会出现乱码的问题，我估计也是由于WinForm在获取HTML文本的时候没有用对正确的字符编码导致的。Windows剪贴板只支持UTF-8编码，也就是説你传入的文本都会被UTF-8编解码。这样一来，只要两个程序都是调用Windows剪切板API编程的话，那么复製粘贴的过程中不会出现乱码。除非一方在获取到剪贴板数据之后使用了错误的字符编码进行解码，才会得到乱码（我做了简单的WinForm剪切板编程实验，发现GetData使用的是系统默认编码，而不是UTF-8编码）。

关于乱码中出现?或者?，这里需要额外提一下，**当程序使用特定字符编码解析字节流的时候，一旦遇到无法解**

JerryC

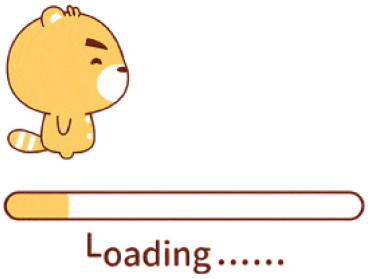
目录
你已经读了 25 %
1. 关于字符编码，你所需要知道的（ASCII,Unicode,Utf-8,GB2312… ）
1.1. 还是得从ASCII码説起
1.2. OEM字符集的衍生
1.3. 多字节字符集（MBCS）和中文字符集
1.4. ANSI标準、国家标準、ISO标準
1.5. Unicode的出现
1.5.1. Unicode字符集概述
1.5.2. 编码系统的变化
1.5.3. 常见的Unicode编码
1.5.3.1. UCS-2/UTF-16
1.5.3.2. UTF-8
1.5.3.3. GB18030
1.5.4. Unicode相关的常见问题
1.5.5. 乱码问题
1.5.6. 必要的术语解释
2. Unicode 和 UTF-8 有什么区别？

而你又无法得到原始字节流的时候，説明正确的信息已经彻底丢失了，尝试任何字符编码都无法从这样的字符文本中还原出正确的信息来。

必要的术语解释

字符集（Character Set），字面上的理解就是字符的集合，例如ASCII字符集，定义了128个字符；GB2312定义了7445个字符。而**计算机系统中提到的字符集準确来説，指的是已编号的字符的有序集合（不一定是连续）**。

字符码（Code Point）指的就是字符集中每个字符的数字编号。例如ASCII字符集用0-127这连续的128个数字分别表示128个字符；GBK字符集使用区位码的方式为每个字符编号，首先定义一个94X94的矩阵，行称为“区”，列称为“位”，然后将所有国标汉字放入矩阵当中，这样每个汉字就可以用唯一的“区位”码来标识了。例如“中”字被放到54区第48位，因此字符码就是5448。而Unicode中将字符集按照一定的类别划分到0~16这17个层面（Planes）中，每个层面中拥有2^{16}=65536个字符码，因此Unicode总共拥有的字符码，也即是Unicode的字符空间总共有17*65536=1114112。



编码的过程是将字符转换成字节流。

解码的过程是将字节流解析为字符。

字符编码（Character Encoding）是将字符集中的字符码映射为字节流的一种具体实现方案。例如ASCII字符编码规定使用单字节中低位的7个比特去编码所有的字符。例如‘A’的编号是65，用单字节表示就是0x41，因此写入存储设备的时候就是b’01000001’。GBK编码则是将区位码（GBK的字符码）中的区码和位码的分别加上0xA0（160）的偏移（之所以要加上这样的偏移，主要是为了和ASCII码兼容），例如刚刚提到的“中”字，区位码是5448，十六进制是0x3630，区码和位码分别加上0xA0的偏移之后就得到0xD6D0，这就是“中”字的GBK编码结果。

代码页（Code Page）一种字符编码具体形式。早期字符相对少，因此通常会使用类似表格的形式将字符直接映射为字节流，然后通过查表的方式来实现字符的编解码。现代操作系统沿用了这种方式。例如Windows使用936代码页、Mac系统使用EUC-CN代码页实现GBK字符集的编码，名字虽然不一样，但对于同一汉字的编码肯定是一样的。

大小端的説法源自《格列佛游记》。我们知道，鸡蛋通常一端大一端小，小人国的人们对于剥蛋壳时应从哪一端开始剥起有着不一样的看法。同样，计算机界对于传输多字节字（由多个字节来共同表示一个数据类型）时，是先传高位字节（大端）还是先传低位字节（小端）也有着不一样的看法，这就是计算机里头大小端模式的由来了。无论是写文件还是网络传输，实际上都是往流设备进行写操作的过程，而且这个写操作是从流的低地址向高地址开始写（这很符合人的习惯），对于多字节字来説，如果先写入高位字节，则称作大端模式。反之则称作小端模式。也就是説，大端模式下，字节序和流设备的地址顺序是相反的，而小端模式则是相同的。一般网络协议都採用大端模式进行传输，windows操作系统採用Utf-16小端模式。

参考链接：

1. The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)

JerryC

目录
你已经读了 25 %
1. 关于字符编码，你所需要知道的（ASCII,Unicode,Utf-8,GB2312… ）
1.1. 还是得从ASCII码説起
1.2. OEM字符集的衍生
1.3. 多字节字符集（MBCS）和中文字符集
1.4. ANSI标準、国家标準、ISO标準
1.5. Unicode的出现
1.5.1. Unicode字符集概述
1.5.2. 编码系统的变化
1.5.3. 常见的Unicode编码
1.5.3.1. UCS-2/UTF-16
1.5.3.2. UTF-8
1.5.3.3. GB18030
1.5.4. Unicode相关的常见问题
1.5.5. 乱码问题
1.5.6. 必要的术语解释
2. Unicode 和 UTF-8 有什么区别？

3. http://en.wikipedia.org/wiki/Universal_Character_Set
4. http://en.wikipedia.org/wiki/Code_page

Unicode 和 UTF-8 有什么区别？

> 转发自知乎 盛世唐朝 https://www.zhihu.com/question/23374078

很久很久以前，有一羣人，他们决定用8个可以开合的晶体管来组合成不同的状态，以表示世界上的万物。他们看到8个开关状态是好的，于是他们把这称为"**字节**"。再后来，他们又做了一些可以处理这些字节的机器，机器开动了，可以用字节来组合出很多状态，状态开始变来变去。他们看到这样是好的，于是它们就这机器称为"**计算机**"。

开始计算机只在美国用。八位的字节一共可以组合出256(2的8次方)种不同的状态。
他们把其中的编号从0开始的32种状态分别规定了特殊的用途，一但终端、打印机遇上约定好的这些字节被传过来时，就要做一些约定的动作：
遇上0×10, 终端就换行；
遇上0×07, 终端就向人们嘟嘟叫；
遇上0x1b, 打印机就打印反白的字，或者终端就用彩色显示字母。
他们看到这样很好，于是就把这些0×20以下的字节状态称为"控制码"。他们又把所有的空格、标点符号、数字、大小写字母分别用连续的字节状态表示，一直编到了第127号，这样计算机就可以用不同字节来存储英语的文字了。大家看到这样，都感觉很好，于是大家都把这个方案叫做 **ANSI** 的"Ascii"编码（American Standard Code for Information Interchange，美国信息互换标準代码）。当时世界上所有的计算机都用同样的**ASCII**方案来保存英文文字。

后来，就像建造巴比伦塔一样，世界各地都开始使用计算机，但是很多国家用的不是英文，他们的字母里有许多是ASCII里没有的，为了可以在计算机保存他们的文字，他们决定採用 127号之后的空位来表示这些新的字母、符号，还加入了很多画表格时需要用下到的横线、竖线、交叉等形状，一直把序号编到了最后一个状态255。从128 到255这一页的字符集被称"**扩展字符集**"。从此之后，贪婪的人类再没有新的状态可以用了，美帝国主义可能没有想到还有第三世界国家的人们也希望可以用到计算机吧！

等中国人们得到计算机时，已经没有可以利用的字节状态来表示汉字，况且有6000多个常用汉字需要保存呢。但是这难不倒智慧的中国人民，我们不客气地把那些127号之后的奇异符号们直接取消掉，规定：一个小于127的字符的意义与原来相同，但两个大于127的字符连在一起时，就表示一个汉字，前面的一个字节（他称之为高字节）从0xA1用到0xF7，后面一个字节（低字节）从0xA1到0xFE，这样我们就可以组合出大约7000多个简体汉字了。在这些编码里，我们还把数学符号、罗马希腊的字母、日文的假名们都编进去了，连在 ASCII 里本来就有的数字、标点、字母都统统重新编了两个字节长的编码，这就是常説的"全角"字符，而原来在127号以下的那些就叫"半角"字符了。中国人民看到这样很不错，于是就把这种汉字方案叫做 "**GB2312**"。GB2312 是对 ASCII 的中文扩展。

但是中国的汉字太多了，我们很快就就发现有许多人的人名没有办法在这里打出来，特别是某些很会麻烦别人的国家领导人。于是我们不得不继续把GB2312 没有用到的码位找出来老实不客气地用上。后来还是不够用，于是乾脆不再要求低字节一定是127号之后的内码，只要第一个字节是大于127就固定表示这是一个汉字的开始，不管后面跟的是不是扩展字符集里的内容。结果扩展之后的编码方案被称为**GBK** 标準，GBK包括了GB2312 的所有内容，同时又增加了近20000个新的汉字（包括繁体字）和符号。 后来少数民族也要用电脑了，于是我们再扩展，又加了几千个新的少数民族的字，GBK扩成了 **GB18030**。从此之后，中华民族的文化就可以在计算机时代中传承了。 中国的程序员们看到这一系列汉字编码的标準是好的，于是通称他们叫做"**DBCS**"（Double Byte Charecter Set 双字节字符集）。在DBCS系列标準里，最大的特点是两字节长的汉字字符和一字节长的英文字符并存于同一套编码方案里，因此他们写的程序为了支持中文处理，必须要注意字串里的每一个字节的值，如果这个值是大于127的，那么就认为一个双字节字符集里的字符出现了。那时候凡是受过加持，会编程的计算机僧侣们都要每天念下面这个咒语数百遍："一个汉字算两个英文字符！一个汉字算两个英文字符……"

因为当时各个国家都像中国这样搞出一套自己的编码标準，结果互相之间谁也不懂谁的编码，谁也不支持别人的编码，连大陆和台湾这样只相隔了150海里，使用着同一种语言的兄弟地区，也分别採用了不同的 DBCS 编码方案——当时的中国人想让电脑显示汉字，就必须装上一个"汉字系统"，专门用来处理汉字的显示、输入的

JerryC

目录
你已经读了 25 %
1. 关于字符编码，你所需要知道的（ASCII,Unicode,Utf-8,GB2312… ）
1.1. 还是得从ASCII码説起
1.2. OEM字符集的衍生
1.3. 多字节字符集（MBCS）和中文字符集
1.4. ANSI标準、国家标準、ISO标準
1.5. Unicode的出现
1.5.1. Unicode字符集概述
1.5.2. 编码系统的变化
1.5.3. 常见的Unicode编码
1.5.3.1. UCS-2/UTF-16
1.5.3.2. UTF-8
1.5.3.3. GB18030
1.5.4. Unicode相关的常见问题
1.5.5. 乱码问题
1.5.6. 必要的术语解释
2. Unicode 和 UTF-8 有什么区别？

可以用，装错了字符系统，显示就会乱了套！这怎么办？而且世界民族之林中还有那些一时用不上电脑的穷苦人民，他们的文字又怎么办？ 真是计算机的巴比伦塔命题啊！

正在这时，大天使加百列及时出现了——一个叫 ISO（国际标谁化组织）的国际组织决定着手解决这个问题。他们採用的方法很简单：废了所有的地区性编码方案，重新搞一个包括了地球上所有文化、所有字母和符号的编码！他们打算叫它"Universal Multiple-Octet Coded Character Set"，简称 **UCS**, 俗称 "**unicode**"。

unicode开始制订时，计算机的存储器容量极大地发展了，空间再也不成为问题了。于是 ISO 就直接规定必须用两个字节，也就是16位来统一表示所有的字符，对于ASCII里的那些"半角"字符，unicode包持其原编码不变，只是将其长度由原来的8位扩展为16位，而其他文化和语言的字符则全部重新统一编码。由于"半角"英文符号只需要用到低8位，所以其高8位永远是0，因此这种大气的方案在保存英文文本时会多浪费一倍的空间。

这时候，从旧社会里走过来的程序员开始发现一个奇怪的现象：他们的 strlen 函数靠不住了，一个汉字不再是相当于两个字符了，而是一个！是的，从unicode开始，无论是半角的英文字母，还是全角的汉字，它们都是统一的"**一个字符**"！同时，也都是统一的"**两个字节**"，请注意"**字符**"和"**字节**"两个术语的不同，"**字节**"是一个8位的物理存贮单元，而"**字符**"则是一个文化相关的符号。在unicode中，一个字符就是两个字节。一个汉字算两个英文字符的时代已经快过去了。

unicode同样也不完美，这里就有两个的问题，一个是，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，这对于存储空间来説是极大的浪费，文本文件的大小会因此大出二三倍，这是难以接受的。

unicode在很长一段时间内无法推广，直到互联网的出现，为解决unicode如何在网络上传输的问题，于是面向传输的众多 **UTF**（UCS Transfer Format）标準出现了，顾名思义，**UTF-8**就是每次8个位传输数据，而**UTF-16** 就是每次16个位。UTF-8就是在互联网上使用最广的一种unicode的实现方式，这是为传输而设计的编码，并使编码无国界，这样就可以显示全世界上所有文化的字符了。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，当字符在ASCII码的範围时，就用一个字节表示，保留了ASCII字符一个字节的编码做为它的一部分，注意的是unicode一箇中文字符佔2个字节，而UTF-8一箇中文字符佔3个字节）。从unicode到utf-8并不是直接的对应，而是要过一些算法和规则来转换。

Unicode符号範围(十六进制)	UTF-8编码方式（二进制）
0000 0000-0000 007F	0xxxxxxx
0000 0080-0000 07FF	110xxxxx 10xxxxxx
0000 0800-0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
0001 0000-0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

最后简单总结一下：

- 中国人民通过对 ASCII 编码的中文扩充改造，产生了 GB2312 编码，可以表示6000多个常用汉字。
- 汉字实在是太多了，包括繁体和各种字符，于是产生了 GBK 编码，它包括了 GB2312 中的编码，同时扩充了很多。
- 中国是个多民族国家，各个民族几乎都有自己独立的语言系统，为了表示那些字符，继续把 GBK 编码扩充为 GB18030 编码。
- 每个国家都像中国一样，把自己的语言编码，于是出现了各种各样的编码，如果你不安装相应的编码，就无法解释相应编码想表达的内容。
- 终于，有个叫 ISO 的组织看不下去了。他们一起创造了一种编码 UNICODE ，这种编码非常大，大到可以容纳世界上任何一个文字和标誌。所以只要电脑上有 UNICODE 这种编码系统，无论是全球哪种文字，只需要保存文件的时候，保存成 UNICODE 编码就可以被其他电脑正常解释。
- UNICODE 在网络传输中，出现了两个标準 UTF-8 和 UTF-16，分别每次传输 8个位和 16个位。于是就会有人产生疑问，UTF-8 既然能保存那么多文字、符号，为什么国内还有这么多使用 GBK 等编码的人？因为 UTF-8 等编码体积比较大，佔电脑空间比较多，如果面向的使用人羣绝大部分都是中国人，用 GBK 等编码也可以。

JerryC

目录
你已经读了 25 %
1. 关于字符编码，你所需要知道的（ASCII,Unicode,Utf-8,GB2312… ）
1.1. 还是得从ASCII码説起
1.2. OEM字符集的衍生
1.3. 多字节字符集（MBCS）和中文字符集
1.4. ANSI标準、国家标準、ISO标準
1.5. Unicode的出现
1.5.1. Unicode字符集概述
1.5.2. 编码系统的变化
1.5.3. 常见的Unicode编码
1.5.3.1. UCS-2/UTF-16
1.5.3.2. UTF-8
1.5.3.3. GB18030
1.5.4. Unicode相关的常见问题
1.5.5. 乱码问题
1.5.6. 必要的术语解释
2. Unicode 和 UTF-8 有什么区别？

Android Java

打赏

上一篇
Windows必装软件推荐

下一篇
好用的新浪图床工具推荐 - Weibo-Pi...

相关推荐

2019-04-10
Java知识点复习(二)

2019-03-29
Java知识点复习(一)

2018-07-19
Adapter

2018-07-19
9Patch 介绍

2018-07-15
第一行代码笔记-RecyclerView

2018-07-05
第一行代码笔记-ListView

评论

NickName E-Mail Website(http://)

記得留下你的昵稱和郵箱....可以快速收到回復

Emoji | Preview

Reply

6 Comments

Anonymous Chrome 69.0.3497.100 Windows 10.0
2019-10-26

Reply

JerryC

目录
你已经读了 25 %
1. 关于字符编码，你所需要知道的（ASCII,Unicode,Utf-8,GB2312... ）
1.1. 还是得从ASCII码説起
1.2. OEM字符集的衍生
1.3. 多字节字符集（MBCS）和中文字符集
1.4. ANSI标準、国家标準、ISO标準
1.5. Unicode的出现
1.5.1. Unicode字符集概述
1.5.2. 编码系统的变化
1.5.3. 常见的Unicode编码
1.5.3.1. UCS-2/UTF-16
1.5.3.2. UTF-8
1.5.3.3. GB18030
1.5.4. Unicode相关的常见问题
1.5.5. 乱码问题
1.5.6. 必要的术语解释
2. Unicode 和 UTF-8 有什么区别？

不错哦

Anonymous Chrome 69.0.3497.100 Mac OS 10.14.5
2019-09-01
Reply

繁体字，看的有点别扭，不过能看懂

repostone Chrome 63.0.3239.132 Windows 8.1
2019-05-17
Reply

真不知道。

Tengfei Safari 12.1.1 iOS 12.3
2019-05-13
Reply

赞

kacy Safari 11.1 Mac OS 10.13.4
2019-04-18
Reply

太长了😂😂