## 编码浅谈

摘要：编码集，来历，描述，相互转换

## 引论

相信大家一定碰到过，打开某个网页，却显示一堆的乱码，如"бЇЯАзЪСЯ"、"�????????"？还记得 HTTP 中的 Accept-Charset、Accept-Encoding、Accept-Language、Content-Encoding、Content-Lang uage 等消息头字段？这些就是接下来我们要探讨的。

## 一、各种编码的来历及产生原因

话说计算机是由美国人弄出来的 ，他们觉得一个字节（可以表示256个编码）表示英语世界里所有的字母、数字和常用特殊符号已经绰绰有余了（其实ASCII只用了前127个编码）。后来欧洲人不干了，法国人说：我需要在小写字母加上变音符号（如：é），德国人说：我也要加几个字母（Ä ä、Ö ö、Ü ü、ß）。于是，欧洲人就将ASCII没用完的编码（128-255）为自己特有的符号编码（后来称之为“扩展字符集”）。等到我们中国人开始使用计算机的时候， 256个编码哪够？我泱泱大中华，汉字起码也得N多万吧，就连小学生都得要求掌握两三千字。国标局最后拍板：一个字节不够，那我们就用多个字节来为汉字编码吧，但是，国情那么穷，字节那么贵，三个字节伤不起，那就用俩字节吧，先给常用的几千汉字编个码，等以后国家强盛了人民富裕了，咱再扩展呗---于是GB2312就产生了。台湾同胞一看 ，全是简体字，还让不让我们写繁体字的活了，于是台湾同胞也自己弄了个繁体字编码---大五码（Big-5）。...

它基本道出了GB2312和Big5的由来。

UTF8是Unicode一种压缩形式，英文A在unicode中表示为0x0041，老外觉得这种存储方式太浪费，因为浪费了50%的空间，于是就把英文压缩成1个字节，成了utf8编码，但是汉字在utf8中占3个字节，显然用做中文不如ansi合算，这就是中国的网页用作ansi编码而老外的网页常用utf8的原因。

这就是UTF8的由来。

除了UCS-4，ISO 10646标准为“通用字符集”（UCS）定义了一种16位的编码形式（即UCS-2），其编码固定占用2个字节，它包含65536个编码空间（可以为全世界最常用的63K字符编码，为了兼容Unicode，0xD800-0xDFFF之间的码位未使用）。例：“汉”的UCS-2编码为6C49。

但俩个字节并不足以正真地“一统江湖”（a fixed-width 2-byte encoding could not encode enough characters to be truly universal），于是UTF-16诞生了，与UCS-2一样，它使用两个字节为全世界最常用的63K字符编码，不同的是，它使用4个字节对不常用的字符进行编码。UTF-16属于变长编码。

这就是UTF16的由来。

因为当时各个国家都像中国这样搞出一套自己的编码标准，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，也分别采用了不同的 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、输入的问题，但是那个台湾的愚昧封建人士写的算命程序就必须加装另一套支持 BIG5 编码的什么"倚天汉字系统"才可以用，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。他们采用的方法很简单：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"。

## 二、各种编码所占字节及其描述

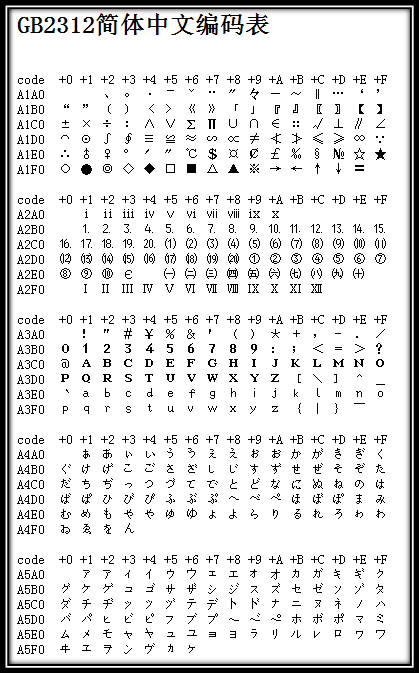
GB2312：一个小于127的字符的意义与原来相同，但两个大于127的字符连在一起时，就表示一个汉字，

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，这就是常说的"全角"字符，而原来在127号以下的那些就叫"半角"字符了。

一个汉字算两个英文字符！一个汉字算两个英文字符……



**Big5：**又称为**大五码**或**五大码**，是使用[繁体中文](http://zh.wikipedia.org/wiki/%E7%B9%81%E4%BD%93%E4%B8%AD%E6%96%87)（正体中文）社区中最常用的电脑[汉字字符集](http://zh.wikipedia.org/wiki/%E5%AD%97%E7%AC%A6%E9%9B%86)标准，共收录 13,060 个汉字。中文码分为[内码](http://zh.wikipedia.org/wiki/%E5%85%A7%E7%A2%BC)及交换码两类， Big5 属中文内码，知名的中文交换码有 CCCII、CNS11643。Big5 虽普及于[台湾](http://zh.wikipedia.org/wiki/%E5%8F%B0%E7%81%A3)、[香港](http://zh.wikipedia.org/wiki/%E4%B8%AD%E5%9C%8B%E9%A6%99%E6%B8%AF)与[澳门](http://zh.wikipedia.org/wiki/%E4%B8%AD%E5%9C%8B%E6%BE%B3%E9%96%80)等繁体中文通行区，但长期以来并非当地的国家标准，而只是[业界标准](http://zh.wikipedia.org/wiki/De_facto)。[倚天中文系统](http://zh.wikipedia.org/wiki/%E5%80%9A%E5%A4%A9%E4%B8%AD%E6%96%87%E7%B3%BB%E7%B5%B1)、[Windows](http://zh.wikipedia.org/wiki/Windows)等主要系统的字符集都是以 Big5 为基准，但厂商又各自增加不同的造字与造字区，派生成多种不同版本。2003 年， Big5 被收录到 CNS11643 中文标准交换码的附录当中，取得了较正式的地位。这个最新版本被称为 Big5-2003。

Big5 码是一套[双字节字符集](http://zh.wikipedia.org/wiki/%E5%8F%8C%E5%AD%97%E8%8A%82%E5%AD%97%E7%AC%A6%E9%9B%86)，使用了双八码存储方法，以两个字节来安放一个字。第一个字节称为"高位字节"，第二个字节称为"低位字节"。"高位字节"使用了 0x81-0xFE，"低位字节"使用了 0x40-0x7E，及 0xA1-0xFE。在 Big5 的分区中：



**UTF-8：**

UTF-8（8-bit Unicode Transformation Format）是一种针对[Unicode](http://zh.wikipedia.org/wiki/Unicode)的可变长度[字符编码（定长码）](http://zh.wikipedia.org/w/index.php?title=%E5%AE%9A%E9%95%BF%E7%A0%81&action=edit&redlink=1)，也是一种前缀码。它可以用来表示 Unicode 标准中的任何字符，且其编码中的第一个[字节](http://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82)仍与[ASCII]()兼容，这使得原来处理 ASCII 字符的[软件](http://zh.wikipedia.org/wiki/%E8%BB%9F%E9%AB%94)无须或只须做少部份修改，即可继续使用。因此，它逐渐成为[电子邮件](http://zh.wikipedia.org/wiki/%E9%9B%BB%E5%AD%90%E9%83%B5%E4%BB%B6)、[网页](http://zh.wikipedia.org/wiki/%E7%B6%B2%E9%A0%81)及其他[存储](http://zh.wikipedia.org/wiki/%E5%84%B2%E5%AD%98%E8%A3%9D%E7%BD%AE)或传送文字的应用中，优先采用的编码。[互联网工程工作小组](http://zh.wikipedia.org/wiki/%E7%B6%B2%E9%9A%9B%E7%B6%B2%E8%B7%AF%E5%B7%A5%E7%A8%8B%E5%B7%A5%E4%BD%9C%E5%B0%8F%E7%B5%84)（IETF）要求所有互联网协议都必须支持 UTF-8 编码。

UTF-8 使用一至四个[字节](http://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82)为每个字符编码：

128 个 US-ASCII 字符只需一个字节编码（ Unicode 范围由 U+0000 至 U+007F）。 带有[附加符号](http://zh.wikipedia.org/wiki/%E9%99%84%E5%8A%A0%E7%AC%A6%E5%8F%B7)的[拉丁文](http://zh.wikipedia.org/wiki/%E6%8B%89%E4%B8%81%E6%96%87)、[希腊文](http://zh.wikipedia.org/wiki/%E5%B8%8C%E8%87%98%E6%96%87)、[西里尔字母](http://zh.wikipedia.org/wiki/%E8%A5%BF%E9%87%8C%E7%88%BE%E5%AD%97%E6%AF%8D)、[亚美尼亚语](http://zh.wikipedia.org/wiki/%E4%BA%9E%E7%BE%8E%E5%B0%BC%E4%BA%9E%E8%AA%9E)、[希伯来文](http://zh.wikipedia.org/wiki/%E5%B8%8C%E4%BC%AF%E4%BE%86%E6%96%87)、[阿拉伯文](http://zh.wikipedia.org/wiki/%E9%98%BF%E6%8B%89%E4%BC%AF%E6%96%87)、[叙利亚文](http://zh.wikipedia.org/wiki/%E5%8F%99%E5%88%A9%E4%BA%9A%E6%96%87)及[它拿字母](http://zh.wikipedia.org/wiki/%E5%AE%83%E6%8B%BF%E5%AD%97%E6%AF%8D)则需要二个字节编码（Unicode 范围由 U+0080 至 U+07FF）。 其他[基本多文种平面](http://zh.wikipedia.org/wiki/%E5%9F%BA%E6%9C%AC%E5%A4%9A%E6%96%87%E7%A8%AE%E5%B9%B3%E9%9D%A2)（BMP）中的字符（这包含了大部分常用字）使用三个字节编码。 其他极少使用的 Unicode[辅助平面](http://zh.wikipedia.org/wiki/%E8%BC%94%E5%8A%A9%E5%B9%B3%E9%9D%A2)的字符使用四字节编码。 在处理经常会用到的 ASCII 字符方面非常有效。在处理扩展的拉丁字符集方面也不比 UTF-16 差。对于中文字符来说，比 UTF-32 要好。同时，由位操作的天性使然，使用 UTF-8 不再存在字节顺序的问题了，一份以 utf-8 编码的文档在不同的计算机之间是一样的比特流（在这一条上你得相信我，因为我不打算给你展示它的数学原理）。

总体来说，在 Unicode 字符串中不可能由码点数量决定显示它所需要的长度，或者显示字符串之后在文本缓冲区中光标应该放置的位置；组合字符、变宽字体、不可打印字符和从右至左的文字都是其归因。所以尽管在 UTF-8 字符串中字符数量与码点数量的关系比 UTF-32 更为复杂，在实际中很少会遇到有不同的情形。

**优点：**UTF-8 是 ASCII 的一个[超集](http://zh.wikipedia.org/wiki/%E8%B6%85%E9%9B%86)。因为一个纯 ASCII 字符串也是一个合法的 UTF-8 字符串，所以现存的 ASCII 文本不需要转换。为传统的扩展 ASCII 字符集设计的软件通常可以不经修改或很少修改就能与 UTF-8 一起使用。 使用标准的面向字节的排序例程对 UTF-8 排序将产生与基于 Unicode 代码点排序相同的结果。（尽管这只有有限的有用性，因为在任何特定语言或文化下都不太可能有仍可接受的文字排列顺序。） UTF-8 和 UTF-16 都是[可扩展标记语言](http://zh.wikipedia.org/wiki/%E5%8F%AF%E6%89%A9%E5%B1%95%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)文档的标准编码。所有其它编码都必须通过显式或文本声明来指定。 任何[面向字节](http://zh.wikipedia.org/w/index.php?title=%E9%9D%A2%E5%90%91%E5%AD%97%E8%8A%82&action=edit&redlink=1)的[字符串搜索算法](http://zh.wikipedia.org/wiki/%E5%AD%97%E7%AC%A6%E4%B8%B2%E6%90%9C%E7%B4%A2%E7%AE%97%E6%B3%95)都可以用于 UTF-8 的数据（只要输入仅由完整的 UTF-8 字符组成）。但是，对于包含字符记数的正则表达式或其它结构必须小心。

UTF-8 字符串可以由一个简单的算法可靠地识别出来。就是，一个字符串在任何其它编码中表现为合法的 UTF-8 的可能性很低，并随字符串长度增长而减小。举例说，字符值 C0,C1,F5 至 FF 从来没有出现。为了更好的可靠性，可以使用正则表达式来统计非法过长和替代值（可以查看 W3 [FAQ: Multilingual Forms](http://www.w3.org/International/questions/qa-forms-utf-8)上的验证 UTF-8 字符串的正则表达式）。

**缺点：**因为每个字符使用不同数量的字节编码，所以寻找串中第 N 个字符是一个 O(N) 复杂度的操作 — 即，串越长，则需要更多的时间来定位特定的字符。同时，还需要位变换来把字符编码成字节，把字节解码成字符。

**UTF-16：**尽管有 Unicode 字符非常多，但是实际上大多数人不会用到超过前 65535 个以外的字符。因此，就有了另外一种 Unicode 编码方式，叫做 UTF-16(因为 16 位 = 2 字节)。UTF-16 将 0–65535 范围内的字符编码成 2 个字节，如果真的需要表达那些很少使用的"星芒层(astral plane)"内超过这 65535 范围的 Unicode 字符，则需要使用一些诡异的技巧来实现。UTF-16 编码最明显的优点是它在空间效率上比 UTF-32 高两倍，因为每个字符只需要 2 个字节来存储（除去 65535 范围以外的），而不是 UTF-32 中的 4 个字节。并且，如果我们假设某个字符串不包含任何星芒层中的字符，那么我们依然可以在常数时间内找到其中的第 N 个字符，直到它不成立为止这总是一个不错的推断。

其编码方法是：如果字符编码 U 小于 0x10000，也就是十进制的 0 到 65535 之内，则直接使用两字节表示； 如果字符编码 U 大于 0x10000，由于 UNICODE 编码范围最大为 0x10FFFF，从 0x10000 到 0x10FFFF 之间 共有 0xFFFFF 个编码，也就是需要 20 个 bit 就可以标示这些编码。用 U'表示从 0-0xFFFFF 之间的值，将其前 10 bit 作为高位和 16 bit 的数值 0xD800 进行 逻辑 or 操作，将后 10 bit 作为低位和 0xDC00 做 逻辑 or 操作，这样组成的 4 个 byte 就构成了 U 的编码。 对于 UTF-32 和 UTF-16 编码方式还有一些其他不明显的缺点。不同的计算机系统会以不同的顺序保存字节。这意味着字符 U+4E2D 在 UTF-16 编码方式下可能被保存为 4E 2D 或者 2D 4E，这取决于该系统使用的是大尾端 ( big-endian )还是小尾端 ( little-endian ) 。（对于 UTF-32 编码方式，则有更多种可能的字节排列。）只要文档没有离开你的计算机，它还是安全的——同一台电脑上的不同程序使用相同的字节顺序 ( byte order)。但是当我们需要在跨系统之间传输这个文档的时候，也许在万维网中，我们就需要一种方法来指示当前我们的字节是怎样存储的。不然的话，接收文档的计算机就无法知道这两个字节 4E 2D 表达的到底是 U+4E2D 还是 U+2D4E。

为了解决这个问题，多字节的 Unicode 编码方式定义了一个"字节顺序标记( Byte Order Mark)"，它是一个特殊的非打印字符，你可以把它包含在文档的开头来指示你所使用的字节顺序。对于 UTF-16，字节顺序标记是 U+FEFF。如果收到一个以字节 FF FE 开头的 UTF-16 编码的文档，你就能确定它的字节顺序是单向的( one way)的了；如果它以 FE FF 开头，则可以确定字节顺序反向了。

**UNICODE:**不得不单独说 Unicode，像天朝一样，当计算机传到世界各个国家时，为了适合当地语言和字符，设计和实现类似 GB232/GBK/GB18030/BIG5 的编码方案。这样各搞一套，在本地使用没有问题，一旦出现在网络中，由于不兼容，互相访问就出现了乱码现象。

为了解决这个问题，一个伟大的创想产生了——Unicode 。Unicode 编码系统为表达任意语言的任意字符而设计。它使用 4 字节的数字来表达每个字母、符号，或者表意文字( ideograph)。每个数字代表唯一的至少在某种语言中使用的符号。（并不是所有的数字都用上了，但是总数已经超过了 65535，所以 2 个字节的数字是不够用的。）被几种语言共用的字符通常使用相同的数字来编码，除非存在一个在理的语源学( etymological )理由使之不这样做。不考虑这种情况的话，每个字符对应一个数字，每个数字对应一个字符。即不存在二义性。不再需要记录"模式"了。U+0041 总是代表'A'，即使这种语言没有'A'这个字符。

在[计算机科学领域中](http://zh.wikipedia.org/wiki/%E9%9B%BB%E8%85%A6%E7%A7%91%E5%AD%B8)， Unicode （统一码、万国码、单一码、标准万国码）是业界的一种标准，它可以使电脑得以体现世界上数十种文字的系统。Unicode 是基于[通用字符集](http://zh.wikipedia.org/wiki/%E9%80%9A%E7%94%A8%E5%AD%97%E7%AC%A6%E9%9B%86)（ Universal Character Set）的标准来发展而来，并且同时也以书本的形式[1]对外发表。Unicode 还不断在扩增， 每个新版本插入更多新的字符。直至目前为止的第六版，Unicode 就已经包含了超过十万个[字符](http://zh.wikipedia.org/wiki/%E5%AD%97%E7%AC%A6)（在[2005](http://zh.wikipedia.org/wiki/2005%E5%B9%B4)年，Unicode 的第十万个字符被采纳且认可成为标准之一）、一组可用以作为视觉参考的代码图表、一套编码方法与一组标准[字符编码](http://zh.wikipedia.org/wiki/%E5%AD%97%E7%AC%A6%E7%BC%96%E7%A0%81)、一套包含了上标字、下标字等字符特性的枚举等。Unicode 组织（The Unicode Consortium）是由一个非营利性的机构所运作，并主导 Unicode 的后续发展，其目标在于：将既有的字符编码方案以 Unicode 编码方案来加以取代，特别是既有的方案在多语言环境下，皆仅有有限的空间以及不兼容的问题。

**（可以这样理解：Unicode 是字符集，UTF-32/ UTF-16/ UTF-8 是三种字符编码方案。）**

UCS & UNICODE

通用字符集（Universal Character Set，UCS）是由[ISO](http://zh.wikipedia.org/wiki/%E5%9C%8B%E9%9A%9B%E6%A8%99%E6%BA%96%E5%8C%96%E7%B5%84%E7%B9%94)制定的 ISO 10646（或称 ISO/IEC 10646）标准所定义的标准字符集。历史上存在两个独立的尝试创立单一字符集的组织，即国际标准化组织（ISO）和多语言软件制造商组成的[统一码联盟](http://zh.wikipedia.org/wiki/%E7%B5%B1%E4%B8%80%E7%A2%BC%E8%81%AF%E7%9B%9F)。前者开发的 ISO/IEC 10646 项目，后者开发的[统一码](http://zh.wikipedia.org/wiki/%E7%B5%B1%E4%B8%80%E7%A2%BC)项目。因此最初制定了不同的标准。

[1991 年](http://zh.wikipedia.org/wiki/1991%E5%B9%B4)前后，两个项目的参与者都认识到，世界不需要两个不兼容的字符集。于是，它们开始合并双方的工作成果，并为创立一个单一编码表而协同工作。从 Unicode 2.0 开始，Unicode 采用了与 ISO 10646-1 相同的字库和字码；ISO 也承诺，ISO 10646 将不会替超出 U+10FFFF 的 UCS-4 编码赋值，以使得两者保持一致。两个项目仍都存在，并独立地公布各自的标准。但统一码联盟和 ISO/IEC JTC1/SC2 都同意保持两者标准的码表兼容，并紧密地共同调整任何未来的扩展。在发布的时候，Unicode 一般都会采用有关字码最常见的字型，但 ISO 10646 一般都尽可能采用[Century 字型](http://zh.wikipedia.org/w/index.php?title=Century%E5%AD%97%E5%9E%8B&action=edit&redlink=1)。

## 三、编码集之间的相互转换

  为了要将Unicode转换为UTF-8，当然要知道他们的区别到底在什么地方。下面来看一下，在Unicode中的编码是怎样转换成UTF-8的，在UTF-8中，如果一个字符的字节小于0x80（128）则为ASCII字符，占一个字节，可以不用转换，因为UTF-8兼容ASCII编码。假如在Unicode中汉字“你”的编码为“u4F60”，把它转换为二进制为100111101100000，然后按照UTF-8的方法进行转换。可以将Unicode二进制从地位往高位取出二进制数字，每次取6位，如上述的二进制就可以分别取出为如下所示的格式，前面按格式填补，不足8位用0填补。

           unicode:  100111101100000                  4F60  
  
           utf-8:    11100100,10111101,10100000       E4BDA0

## 结语

## 参考文献

维基百科. GB2312. <http://zh.wikipedia.org/wiki/GB_2312>,

维基百科. Unicode. <http://zh.wikipedia.org/wiki/Unicode>,

https://zh.wikipedia.org/wiki/%E5%A4%A7%E4%BA%94%E7%A2%BC

[http://xdotnet.cnblogs.com](http://xdotnet.cnblogs.com/)