**字符编码简谈**

**总体概述**

**字符集**：字符集是一张码表，它规定了文字与数字的一一对应关系。与计算机的内部表示没有必然的联系。

**编码集**：将字符集存储到计算机里的方案。字符必须编码后才能被计算机处理。计算机使用的缺省编码方式就是计算机的内码。

ASCII字符集，它规定了0~127这128个数字与哪些字符的对应关系。

Unicode字符集为表达任意语言的任意字符而设计，它对世界上大部分的[文字系统](https://zh.wikipedia.org/wiki/%E6%96%87%E5%AD%97%E7%B3%BB%E7%B5%B1)进行了整理、编码，使得电脑可以用更为简单的方式来呈现和处理文字。

Gb2312与big5等方案既包括了字符集，又包括了编码集。

UTF-8以及UTF-16是一种针对[Unicode](https://zh.wikipedia.org/wiki/Unicode)的可变长度[字符编码](https://zh.wikipedia.org/wiki/%E5%AD%97%E5%85%83%E7%B7%A8%E7%A2%BC)，属于编码集。

**GB2312**

**来历**：GB 2312 或 GB 2312–80 是中华人民共和国国家标准简体中文字符集，全称《信息交换用汉字编码字符集·基本集》，又称GB0，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。

**产生原因**：早期的计算机使用7位的ASCII编码，为了处理汉字，程序员设计了用于简体中文的GB2312。

**字节数**：2字节

**兼容性**：向下兼容ASCII码，与big5、Unicode互不兼容。GB2312 是对 ASCII 的简体中文扩展。

**其他**：GB2312能表示7445个符号，包括6763个汉字，几乎覆盖所有高频率汉字。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。显然码点的个数已经超出了1个字节的表示范围，所以使用两个字节来编码一个汉字。

GB 2312的出现，基本满足了汉字的计算机处理需要，它所收录的汉字已经覆盖中国大陆99.75%的使用频率。但对于人名、古汉语等方面出现的罕用字和繁体字，GB 2312不能处理，因此后来GBK及GB 18030汉字字符集相继出现以解决这些问题。

**Big5**

**来历**： Big5是由台湾[财团法人信息产业策进会](https://zh.wikipedia.org/wiki/%E8%B3%87%E8%A8%8A%E5%B7%A5%E6%A5%AD%E7%AD%96%E9%80%B2%E6%9C%83)为[五大中文套装软件](https://zh.wikipedia.org/wiki/%E4%BA%94%E5%A4%A7%E4%B8%AD%E6%96%87%E5%A5%97%E8%A3%9D%E8%BB%9F%E9%AB%94)所设计的中文共通内码，在1983年12月完成公告，隔年3月，信息产业策进会与[台湾](https://zh.wikipedia.org/wiki/%E8%87%BA%E7%81%A3)13家厂商签定“16位个人电脑套装软件合作开发（BIG-5）项目，因为此中文内码是为台湾自行制作开发之“五大中文套装软件”所设计的，所以就称为Big5中文内码。五大中文套装软件虽然并没有如预期的取代国外的套装软件，但随着采用Big5码的[国乔中文系统](https://zh.wikipedia.org/w/index.php?title=%E5%9C%8B%E5%96%AC%E4%B8%AD%E6%96%87%E7%B3%BB%E7%B5%B1&action=edit&redlink=1)及[倚天中文系统](https://zh.wikipedia.org/wiki/%E5%80%9A%E5%A4%A9%E4%B8%AD%E6%96%87%E7%B3%BB%E7%B5%B1)先后在台湾市场获得成功，使得Big5码深远地影响繁体中文电脑[内码](https://zh.wikipedia.org/wiki/%E5%85%A7%E7%A2%BC)，直至今日。“五大码”的英文名称“Big5”后来被人按英文字序译回中文，以致现在有“五大码”和“大五码”两个中文名称。

**产生原因**：Big5码的产生，是因为当时[个人电脑](https://zh.wikipedia.org/wiki/%E5%80%8B%E4%BA%BA%E9%9B%BB%E8%85%A6)没有共通的内码，导致厂商推出的中文应用软件无法推广，并且与[IBM 5550](https://zh.wikipedia.org/wiki/IBM_5550)、[王安码](https://zh.wikipedia.org/w/index.php?title=%E7%8E%8B%E5%AE%89%E7%A2%BC&action=edit&redlink=1" \o "王安码（页面不存在）)等内码，彼此不能兼容；另一方面，台湾当时尚未推出中文编码标准。在这样的时空背景下，为了使台湾早日进入信息时代，所采行的一个项目。

**字节数**：2字节

**兼容性**：向下兼容ASCII码，与GB2312、Unicode互不兼容。Big5是对 ASCII 的繁体中文扩展。

**其他**：① Big5字符集共收录13,053个中文字，该字符集在中国台湾使用。耐人寻味的是该字符集重复地收录了两个相同的字：“兀”(0xA461及0xC94A)、“嗀”(0xDCD1及0xDDFC)。

②冲码问题：因为低比特字符中包含了编程语言、shell、script中，字符串或命令常会用到的特殊字符，例如0x5C“\”、0x7C“|”等。“\”在许多用途的字符串中是当作转义符号又称为转义字符，例如\n（换行）、\r（归位)、\t（tab）、\\（\本身符号）、\"（引号）等等。而“|”在UNIX操作系统中大多当作命令管线的使用，如"ls -la | more"等等。如果在字符串中有这些特殊的转义字符，会被程序或解释器解释为特殊用途。但是因为是中文的原因，故无法正确解释为上面所述的行为，因此程序可能会忽略此转义符号或是中断运行。若此，就违反了用户本来要当成中文字符一部分使用的本意。

低比特字符与ASCII重叠的字符如下︰

@ A-Z [ \ ] ^ \_ ` a-z { | } ~

在常用字如“功”（0xA55C）、“許”（0xB35C）、“蓋”（0xBB5C）、“育”（0xA87C）中时常出现，造成了许多软件无法正确处理以Big5编码的字符串或文件。这个问题被戏谑性地人名化，称为“許功蓋”或“許蓋功”（这三个字都有这种问题）。

一般的解决方法，是额外增加“\”的字符，因为“\\”会被解释为“\”，所以“成功\因素”这个字符串就能无误地被程序当作“成功因素”的字符串来处理。但是额外的困扰是，有些输出功能并不会把“\”当作特殊字符看待，所以有些程序或网页就会错误地常常出现在“许功盖”这些字后面多了“\”。

③Big5延伸：由于Big5码内的一万多个字，只是根据中华民国教育部颁布的《常用国字标准字体表》、《次常用国字标准字体表》等用字汇编而成，并没有考虑社会上流通的人名、地名用字、方言用字、化学及生物科学等用字，亦没有放入日语平假名及片假名字母。

所以在市面上支持Big5码的软件，有不少都自行在原本的编码外，添加一些符号及用字。如倚天Big5延伸，Code Page 950，中国海字集，Big5+，Big5-2003等。

**Unicode**

**来历**：

统一码联盟在1991年首次发布了The Unicode Standard。Unicode的开发结合了国际标准化组织所制定的ISO/IEC 10646，即通用字符集。Unicode与ISO/IEC 10646在编码的运作原理相同，但The Unicode Standard包含了更详尽的实现信息、涵盖了更细节的主题，诸如比特编码（bitwise encoding）、校对以及呈现等。

**产生原因**：Unicode是为了解决传统的字符编码方案的局限而产生的，例如ISO 8859-1所定义的字符虽然在不同的国家中广泛地使用，可是在不同国家间却经常出现不兼容的情况。很多传统的编码方式都有一个共同的问题，即容许电脑处理双语环境（通常使用拉丁字母以及其本地语言），但却无法同时支持多语言环境（指可同时处理多种语言混合的情况）。

**字节数：**4字节

**其他**：

|  |  |  |  |
| --- | --- | --- | --- |
| **平面** | **始末字符值** | **中文名称** | **英文名称** |
| 0号平面 | U+0000 - U+FFFF | **基本多文种平面** | Basic Multilingual Plane，简称**BMP** |
| 1号平面 | U+10000 - U+1FFFF | **多文种补充平面** | Supplementary Multilingual Plane，简称**SMP** |
| 2号平面 | U+20000 - U+2FFFF | **表意文字补充平面** | Supplementary Ideographic Plane，简称**SIP** |
| 3号平面 | U+30000 - U+3FFFF | **表意文字第三平面**（未正式使用） | Tertiary Ideographic Plane，简称**TIP** |
| 4号平面至13号平面 | U+40000 - U+DFFFF | （尚未使用） |  |
| 14号平面 | U+E0000 - U+EFFFF | **特别用途补充平面** | Supplementary Special-purpose Plane，简称**SSP** |
| 15号平面 | U+F0000 - U+FFFFF | 保留作为**私人使用区（A区）**[[2]](https://zh.wikipedia.org/wiki/Unicode%E5%AD%97%E7%AC%A6%E5%B9%B3%E9%9D%A2%E6%98%A0%E5%B0%84" \l "cite_note-PUA-2) | Private Use Area-A，简称**PUA-A** |
| 16号平面 | U+100000 - U+10FFFF | 保留作为**私人使用区（B区）**[[2]](https://zh.wikipedia.org/wiki/Unicode%E5%AD%97%E7%AC%A6%E5%B9%B3%E9%9D%A2%E6%98%A0%E5%B0%84" \l "cite_note-PUA-2) | Private Use Area-B，简称**PUA-B** |

①UNICODE是一个世界级的字符集，全世界的每一个字符都有唯一的码点。码点具有这样的形式U+[XX]XXXX，其中，X是一个十六进制数字。UNICODE的范围目前是U+0000~U+10FFFF，超过100万。因为码点太多了，为了方便管理，把每65536个码点归为一组，称为一个平面（Plane），共有17个平面。

**基本多文种平面**（Basic Multilingual Plane, BMP），或称第0平面或0号平面（Plane 0），是Unicode中的一个编码区段。编码从U+0000至U+FFFF，日常用到的绝大多数字符都在这个平面。

**第一辅助平面**又称**多文种补充平面**（Supplementary Multilingual Plane，缩写SMP，或简称Plane 1），摆放拼音文字（主要为现时已不再使用的古老文字）、手写文字、音符、绘文字和其他图形符号。用于学者的专业论文中使用的古老或过时的语言书写符号，以及网络通信等使用的表情符号。范围在U+10000～U+1FFFD。

**第二辅助平面**又称为**表意文字补充平面**（Supplementary Ideographic Plane，缩写SIP，或简称Plane 2），整个范围在U+20000～U+2FFFD。整个平面配置的都是一些罕用的汉字或地区的方言用字，如粤语用字及越南语的字喃。

②The Unicode Standard也枚举了诸多的字符特性，包含了那些必须支持两种阅读方向的文字（由左至右或由右至左的文字阅读方向，例如阿拉伯文是由右至左）。Unicode与ISO/IEC 10646这两个标准在术语上的使用有些微的不同。

③Unicode字符集有多个编码方式，分别是UTF-8，UTF-16，UTF-32编码。

**UTF-8及UTF-16**

**来历**：1992年初，为创建良好的字节串编码系统以供多字节字符集使用，开始了一个正式的研究。ISO/IEC 10646的初稿中有一个非必须的附录，名为UTF。当中包含了一个供32比特的字符使用的字节串编码系统。这个编码方式的性能并不令人满意，但它提出了将0-127的范围保留给ASCII以兼容旧系统的概念。

1992年7月，X/Open委员会XoJIG开始寻求一个较佳的编码系统。Unix系统实验室（USL）的Dave Prosser为此提出了一个编码系统的建议。它具备可更快速实现的特性，并引入一项新的改进。其中，7比特的ASCII符号只代表原来的意思，所有多字节序列则会包含第8比特的符号，也就是所谓的最高有效比特。

1992年8月，这个建议由IBMX/Open的代表流传到一些感兴趣的团体。与此同时，贝尔实验室九号项目操作系统工作小组的肯·汤普逊对这编码系统作出重大的修改，让编码可以自我同步，使得不必从字符串的开首读取，也能找出字符间的分界。

1993年1月25-29日的在圣地牙哥举行的USENIX会议首次正式介绍UTF-8。

**产生原因**：UTF-8（8-bit Unicode Transformation Format）是针对Unicode的可变长度字符编码，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仍与ASCII兼容，这使得原来处理ASCII字符的软件无须或只须做少部分修改，即可继续使用。

**字节数**：UTF-8是1~6字节（尽管如此，2003年11月UTF-8被RFC 3629重新规范，只能使用原来Unicode定义的区域，U+0000到U+10FFFF，也就是说最多四个字节），UTF-16是2或4字节

**兼容性**：utf-8向下兼容ASCII码，utf16不兼容ASCII码。

|  |  |  |  |
| --- | --- | --- | --- |
| 编码方式 | 码元类型 | 编码字节数 | 字节序 |
| UTF-8 | 单字节码元 | 单字节（ASCII字符）或多字节（非ASCII字符） | 单字节码元，无字节序问题 |
| UTF-16 | 双字节码元 | 双字节（BMP字符）或四字节（辅助平面字符） | 双字节码元，有字节序问题 |

**其他**：

|  |  |
| --- | --- |
| Unicode符号范围 | UTF-8编码方式 |
| 16进制 | 2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 |

1. UTF-8对所有常用的字符都可以用三个字节表示，而且它的另一种选择， UTF-16编码，对前述的第四种字符同样需要四个字节来编码，所以要决定UTF-8或UTF-16哪种编码比较有效率，还要视所使用的字符的分布范围而定。

对于utf-8而言，128个US-ASCII字符只需一个字节编码（Unicode范围由U+0000至U+007F）。带有附加符号的拉丁文、希腊文、西里尔字母、亚美尼亚语、希伯来文、阿拉伯文、叙利亚文及它拿字母则需要两个字节编码（Unicode范围由U+0080至U+07FF）。其他基本多文种平面（BMP）中的字符（这包含了大部分常用字，如大部分的汉字）使用三个字节编码（Unicode范围由U+0800至U+FFFF）。其他极少使用的Unicode 辅助平面的字符使用四至六字节编码（Unicode范围由U+10000至U+1FFFFF使用四字节，Unicode范围由U+200000至U+3FFFFFF使用五字节，Unicode范围由U+4000000至U+7FFFFFFF使用六字节）。

②UTF-16对于ASCII字符也必须使用两个字节(因为是16位码元)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，UTF-16一开始推出的时候就遭到很多西方国家的抵制，大大影响了Unicode的推行。于是后来又设计了UTF-8编码方式，才解决了这些问题。

1. **字节序标记BOM**：

a) 在将逻辑形式的码元序列(或可称之为逻辑编码)映射为物理形式的字节序列(或可称之为物理编码)时，因系统平台的差异，存在一个字节序(Byte-Order字节顺序)的问题。Unicode/UCS规范中推荐的标记字节顺序的方法是**BOM字节序标记**(Byte-Order Mark字节顺序标记)。

字节序标记BOM是Unicode码点值为FEFF(十进制为65279，二进制为1111 1110 1111 1111)的字符的别名。

最初，字符U+FEFF如果出现在字节流的开头，则用来标识该字节流的字节序——是高位在前还是低位在前；如果它出现在字节流的中间，则表达为该字符的原义——零宽度不中断空格(ZERO WIDTH NO-BREAK SPACE零宽度无断空白)。该字符名义上是个空格，实际上是零宽度的，即相当于是不可见也不可打印字符(平常使用较多的是ASCII空格字符，是非零宽度的，需要占用一个字符的宽度，为可见不可打印字符)。

从Unicode 3.2开始，U+FEFF只能出现在字节流的开头，且只能用于标识字节序，就如它的别名——字节序标记——所表示的意思一样；除此以外的用法已被舍弃。取而代之的是，使用U+2060来表示零宽度不中断空格。

b) 如果UTF-16编码的字节序列为大端序，则该字节序标记在字节流的开头呈现为0xFE 0xFF；若字节序列为小端序，则该字节序标记在字节流的开头呈现为0xFF 0xFE。如果UTF-32编码的字节序列为大端序，则该字节序标记在字节流的开头呈现为0x00 0x00 0xFE 0xFF；若字节序列为小端序，则该字节序标记在字节流的开头呈现为0xFF 0xFE 0x00 0x00。

UTF-8编码本身没有字节序的问题，但仍然有可能会用到BOM——有时被用来标示某文本是UTF-8编码格式的文本；再强调一遍：在UFT-8编码格式的文本中，如果添加了BOM，则只用它来标示该文本是由UTF-8编码方式编码的，而不用来说明字节序，因为UTF-8编码不存在字节序问题。

c) 许多Windows程序(包含记事本)会添加BOM到UTF-8编码格式的文件中。然而，在类Unix系统中，这种作法则不被建议采用。

因为它会影响到无法识别它的编程语言，如gcc会报告源码文件开头有无法识别的字符；而在PHP中，如果没有激活输出缓冲(output buffering)，它会使得页面内容开始被送往浏览器(即header头被提交)，这使PHP脚本无法指定header头(HTTP Header)。

对于已在IANA注册的字符编码(这里的字符编码实际为字符编码模式CES)UTF-16BE、UTF-16LE、UTF-32BE和UTF-32LE等来说，不可使用BOM。因为其名称本身已决定了其字节顺序。对于已注册的字符编码(这里的字符编码实际为字符编码方式CEF)UTF-16和UTF-32来说，则必须在文本开头使用BOM。

d) 作为逻辑意义上的UTF-16编码(码元序列)，由于历史的原因，在映射为物理意义上的字节序列时，分为UTF-16BE(Big Endian)、UTF-16LE(Little Endian)两种情况。比如，“ABC”这三个字符的UTF-16编码(码元序列)为：00 41 00 42 00 43；其对应的各种字节序列如下：

|  |  |
| --- | --- |
| UTF-16BE（without BOM） | 00 41 00 42 00 43 |
| UTF-16LE（without BOM） | 41 00 42 00 43 00 |
| UTF-16BE（with BOM） | FE FF 00 41 00 42 00 43 |
| UTF-16LE（with BOM） | FF FE 41 00 42 00 43 00 |

③UTF-8同其他的多字节码元编码方式相比具有以下优点：

a) UTF-8的编码空间足够大，未来Unicode新标准收录更多字符，UTF-8也能适应，因此不会再出现UTF-16那样的尴尬。

b) UTF-8是变长编码(准确地说是变长码元序列，而码元本身是固定长度为8位单字节的，也就是说，UTF-8采用的单字节码元)，比如一个字节足以容纳所有的ASCII字符，就用一个字节来存储，不必在高位补0以浪费更多的字节来存储，因此在英语作为国际语言的现实情况下，UTF-8因其ASCII字符的单字节编码这一特性可节省空间。

c) UTF-8完全直接兼容ASCII码，而非不完全间接兼容。

d) UTF-8的码元序列的第一个字节指明了后面所跟的字节的数目(即带有前缀码)，这对字节流的前向解析非常有效。

e) 也因为UTF-8编码带有前缀码，所以容错性好，即使在传输过程中发生局部的字节错误，比如即便丢失、增加、改变了某些字节，也不会导致所有后续字符全部错乱这样传递性、连锁性的错误问题(否则，若存在错误传递性、连锁性的话，一旦中间某些字节出错，则必须丢弃从出错点开始到结尾的所有编码字节，比如GB码、UTF-32码就是如此)，因此很容易重新同步，具有很强的鲁棒性(即健壮性)。

f) 由于UTF-8编码没有状态，从UTF-8字节流的任意位置开始可以有效地找到一个字符的起始位置，字符边界很容易界定、检测出来，所以具有很好的“自同步性”。

g) UTF-8已经成为互联网所采用的字符编码方式的事实标准。

h) UTF-8是字节顺序无关的(因为是单字节码元，而非像UTF-16、UTF-32这样的多字节码元)，它的字节顺序在所有系统中都是一样的，其码元序列与字节序列相同，因此它实际上并不需要字节顺序标记BOM(Byte-Orde Mark)，虽然Windows系统经常“多此一举”地加上BOM。

字节序问题在进行信息交换时会带来不小的麻烦。如果字节序未协商好，将导致乱码；若协商结果为双方一个采用大端一个采用小端，则必然有一方要进行大小端转换，性能损失不可避免(字节序的大小端问题其实不像看起来那么简单，有时会涉及硬件、操作系统、上层应用软件多个层次，可能会导致多次转换)。

i) 字节FE(二进制为1111 1110)和FF(二进制为1111 1111)在UTF-8编码中永远不会出现(因为UTF-8编码方式中，每个字节只能以0、110、1110、11110或10开头，详见后文介绍)。因此可以用称之为零宽度不中断空格(ZERO WIDTH NO-BREAK SPACE)的字符(Unicode字符名称为U+FEFF)作为字节顺序标记BOM来标明UTF-16或UTF-32文本的字节序。

(Windows系统中BOM有时也用在UTF-8编码的文本文件的开头，虽然UTF-8编码不存在字节序问题，但Windows却用BOM来表明该文本文件的编码格式为UTF-8，看起来这有点“多此一举”，)

j) UTF-8编码可以通过屏蔽位和移位操作快速读写。

k) 字符串比较时strcmp()和wcscmp()的返回结果相同，因此使排序变得更加容易。

**Unicode与gb2312、BIG5、UTF-8之间的转换**

在Windows平台下，有一个最简单的转化方法，就是使用内置的记事本小程序Notepad.exe。打开文件后，点击"文件"菜单中的"另存为"命令，会跳出一个对话框，在最底部有一个"编码"的下拉条。里面有四个选项：ANSI，Unicode，Unicode big endian 和 UTF-8。

1）ANSI是默认的编码方式。对于英文文件是ASCII编码，对于简体中文文件是GB2312编码（只针对Windows简体中文版，如果是繁体中文版会采用Big5码）。 2）Unicode编码指的是UCS-2编码方式，即直接用两个字节存入字符的Unicode码。这个选项用的little endian格式。

3）Unicode big endian编码与上一个选项相对应。我在下一节会解释little endian和big endian的涵义。

4）UTF-8编码，也就是上一节谈到的编码方法。 选择完"编码方式"后，点击"保存"按钮，文件的编码方式就立刻转换好了。

**补充文档**

**内码：**

在计算机科学及相关领域当中，内码指的是“将资讯编码后，透过某种方式储存在特定记忆装置时，装置内部的编码形式”。在不同的系统中，会有不同的内码。

在以往的英文系统中，内码为ASCII。 在繁体中文系统中，目前常用的内码为大五码（big5）。在简体中文系统中，内码则为国标码（GB18030等）。

为了软件开发方便，如国际化与本地化，现在许多系统会使用Unicode做为内码，

**交换码**

交换码是指用于交换文件所使用的编码。

对于计算机而言，不同的系统有可能使用不同的内码。但如果不同系统间要交换文件，则会发生乱码现象。解决方法则为，在交换文件前，文件提供者先将由内码形式储存的文件转换成交换码形式再做交换。在接收文件后，文件接收者再由交换码转成内码。

为了方便起见，许多系统的内码则直接使用交换码，如ASCII广为各种系统所使用。

中文交换码:在台湾，中文资讯交换码（CCCII）为最早提出的交换码，但没有成为官方标准。目前的官方标准为中文标准交换码（CNS 11643），但并未得到民间支持。

对繁体中文而言，中文编码的事实标准为大五码，或国际码(Unicode)。鲜有系统或软件使用中文标准交换码做为内码，甚至在交换文件时也不会转成此交换码。

**统一码**

对统一码系统而言，UTF-8、UTF-16、UTF-32皆可说是交换码。

引用：

<http://www.cnblogs.com/benbenalin/tag/UTF-8/>

http://www.cnblogs.com/happyday56/p/4135845.html

<https://zh.wikipedia.org/wiki/Unicode#Unicode.E5.AD.97.E7.AC.A6.E5.B9.B3.E9.9D.A2.E6.98.A0.E5.B0.84>

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

<https://zh.wikipedia.org/wiki/UTF-8#.E6.AD.B7.E5.8F.B2>

<https://zh.wikipedia.org/wiki/UTF-16>

<https://zhuanlan.zhihu.com/p/26261762>