把字符转化为二进制01存储到计算机中的过程称之为编码(encode)

把存储在计算机中的二进制转化成字符的过程称之为解码(decode)

**字符集三个关键元素：**

字库表（character repertoire）

编码字符集(简称字符集)（coded character set， 简称charset）

字符编码（character encoding）

**字库表**：一个相当于所有可读或者可显示字符的数据库，字库表决定了整个字符集能够展现表 示的所有字符的范围

**编码字符集(简称字符集， 字符-编号集)**：即用一个编码值(码点)code point来表示一个字符（即该字符在子库表中的位置），码点可以理解为一个编号或者序号，unicoe字符集包含1114112 个码点，如果每个代码点都能够代表一个有效字符的话，就是包含1114112个字符。在Unicode字符集中，字母A对应的数值是十六进制下的0041，书写时前面加U+，所以Unicode里A的代码点是U+0041(or 65 in dicimal)。

字符和码点的映射表：字符<-----🡪码点 或者 字符<-----🡪编号

**字符编码(字符-编号-编码集)**：是编码字符集和实际存储数值之间的转换关系；

字符<---🡪码点/编号<---🡪存储编码

字符编码是定义在字符集上的映射规则

**为什么不直接把码点(字符编号)作为存储数据？**

既然字库表中的每一个字符都有一个自己的序号，直接把序号作为存储内容就好了。为什么还要多此一举通过字符编码把序号转换成另外一种存储格式呢？其实原因也比较容易理解：统一字库表的目的是为了能够涵盖世界上所有的字符，但实际使用过程中会发现真正用的上的字符相对整个字库表来说比例非常低。例如中文地区的程序几乎不会需要日语字符，而一些英语国家甚至简单的ASCII字库表就能满足基本需求。而如果把每个字符都用字库表中的序号来存储的话，每个字符就需要3个字节（这里以Unicode字库为例），这样对于原本用仅占一个字符的ASCII编码的英语地区国家显然是一个额外成本（存储体积是原来的三倍）。算的直接一些，同样一块硬盘，用ASCII可以存1500篇文章，而用3字节Unicode序号存储只能存500篇

**常见的编码字符集(简称字符集)：unicode, Ascii, gbk**

**ASCII**

**ASCII既表示字符集，又表示字符编码**

**ASCII直接将字符在编码字符集中的序号作为计算机底层存储数值，字符编号=底层存储编码**

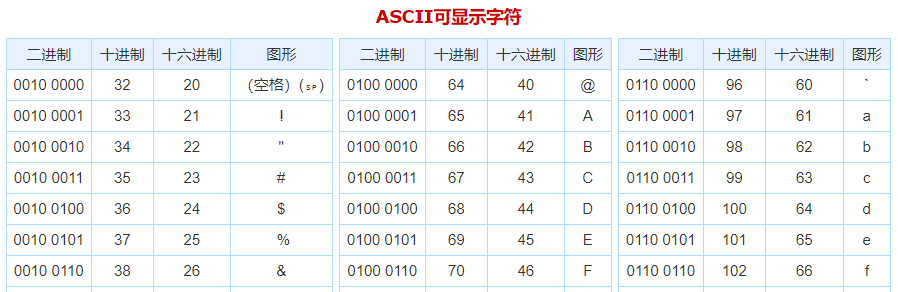
**比如在ASCII中A在表中排第65位，序号十进制是65，对应的二进制数值是0100 0001，所以 A在计算机中的存储内容是：0100 0001**

在计算机种中，1 字节对应 8 位二进制数，而每位二进制数有 0、1 两种状态，因此 1 字节可以组合出 256 种状态。如果这 256 中状态每一个都对应一个符号，就能通过 1 字节的数据表示 256 个字符。美国人于是就制定了一套编码（其实就是个字典），描述英语中的字符和这 8 位二进制数的对应关系，这被称为 ASCII 码。

标准ASCII 码用一个字节的前7位二进制来编码组合来表示128种可能的字符,包括大小写字母，数字0 到9、标点符号，以及在美式英语中使用的特殊控制字符。

十进制编号：0~127

二进制编号：00000000~011111111





**扩展ASCII**

**字符编号=底层存储编码**

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，**用字节中闲置的最高位编入新的符号**。

这样一来，这些欧洲国家使用的编码体系，**可以表示最多256个符号**。但是，这里又出现了新的问题**： 0 - 127 号字符上达成了一致，但对于 128 - 255 号字符上却有不同的编码方式**。由于不同的国家有不同的字母，于是造成同一个编码对于不同的国家来说，表示的字符可能也不同。比如，130在法语编码中代表了é，在希伯来语编码中却代表了字母Gimel (ג)；144 在阿拉伯人的 ASCII 码中是 **گ**，而在俄罗斯的 ASCII 码中是 **ђ**。但是不管怎样，所有这些编码方式中，0~127表示的符号是一样的，不一样的只是128--255的这一段。

**Unocode**

unicode就是一种字符集，可以把它想象成**字符-编号字典，**它记录了世界上所有的符号以及对应的编号，现在的规模可以容纳100多万个符号(unicoe字符集包含1114112 个码点，如果每个代码点都能够代表一个有效字符的话，就是包含1114112个字符)；它的想法很简单，就是为每个字符规定一个用来表示该字符的数字，仅此而已。

**unicode码点范围：U+0x0000 至 U+0x10FFFF**

在表示一个Unicode的字符时，通常会用“U+”然后紧接着一组十六进制的数字来表示这一个字符的编号(当然也可以是二进制或者十进制编号)。**对于BMP平面的码点或者符号(十进制编号：0~65535，可以理解为unicode表中的前65536个字符)，U+四位十六进制数**，比如：

‘严’<🡪 U+4E25。对于其他平面，编号需要使用U+五位或六位十六进制数 。

至于这个**字符或者编号如何在计算机中的底层存储则没有任何规定**。比如，汉字‘严’的 Unicode 是十六进制数U+4E25，转换成二进制数足足有15位（100111000100101），也就是说，这个符号的表示至少需要2个字节。表示其他更大的符号，可能需要3个字节或者4个字节，甚至更多。

这里就有两个严重的问题：

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；

**Unicode字符集有三种编码方式**：UTF-8、UTF-16、UTF-32

**UTF-32**

UTF-32 是固定长度的编码，始终占用 4 个字节，足以容纳所有的 Unicode 字符，所以直接存储 Unicode 编号即可，不需要任何编码转换。浪费了空间，提高了效率。

**UTF-8**

**UTF-8** 就是在互联网上使用最广的一种 Unicode 的实现方式

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编码规则如下:**

1. 对于单个字节的字符，第一位设为 0，后面的 7 位对应这个字符的 Unicode 码点。因此，对于英文中的 0 - 127 号字符，与 ASCII 码完全相同。这意味着 ASCII 码那个年代的文档用 UTF-8 编码打开完全没有问题。

2. 对于需要使用 N 个字节来表示的字符（N > 1），第一个字节的前 N 位都设为 1，第 N + 1 位设为0，剩余的 N - 1 个字节的前两位都设位 10，剩下的二进制位则使用这个字符的 Unicode 码点来填充。

| **Unicode 十六进制码点范围** | **UTF-8 二进制** |
| --- | --- |
| 0000 0000 - 0000 007F(0~127) | 0xxxxxxx |
| 0000 0080 - 0000 07FF(128~2047) | 110xxxxx 10xxxxxx |
| 0000 0800 - 0000 FFFF(2048~65535) | 1110xxxx 10xxxxxx 10xxxxxx |
| 0001 0000 - 0010 FFFF(>65536) | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx |

**技巧：**

首先找到该Unicode编号所在的编号范围，进而可以找到与之对应的二进制格式，然后将该Unicode编号转化为二进制数（去掉高位的0），最后将该二进制数从右向左依次填入二进制格式的X中，如果还有X未填，则设为0 。

“马”的Unicode编号是：0x9A6C,整数编号是39532，对应第三个范围（2048 - 65535），其格式为：1110XXXX 10XXXXXX 10XXXXXX，39532 对应的二进制是 1001 1010 0110 1100，将二进制填入进入就为： 11101001 10101001 10101100 。

**UTF-16**

在了解 UTF-16 编码方式之前，先了解一下另外一个概念——"平面"。

在上面的介绍中，提到了 Unicode 是一本很厚的字典，她将全世界所有的字符定义在一个集合里。这么多的字符不是一次性定义的，而是分区定义。每个区可以存放 65536 个（2^16）字符/代码点，称为一个平面（plane）。目前，一共有 17 个平面：**编号为 #0-#16**。其中，Plane#0 叫做基本多语言平面（Basic Multilingual Plane，BMP），其余平面叫做补充平面（Supplementary Planes）

BMP码点范围是从 0 到65535（ 0~2^16-1），写成 16 进制就是从 U+0000 到 U+FFFF。所有最常见的字符都放在这个平面，这是 Unicode 最先定义和公布的一个平面。剩下的字符都放在辅助平面（简称 SMP ），码点范围从 U+010000 到 U+10FFFF。

UTF-16 编码介于 UTF-32 与 UTF-8 之间，同时结合了定长和变长两种编码方法的特点

**编码规则：**

Unicode字符集的码点(编号)取值范围为U+0000 至 U+10FFFF 或者 0X0000~0X10FFFF

**1 U<0x10000等价于 U <=0xFFFF等价于 U<65536**

**编码就是码点(编号)对应的二进制整数，**所以直接把代码点(编号)转换成 2 个字节就可以了，不足 16位的只需要把高字节全部填为0就可以。

2 **U≥0x10000 等价于U>=0xFFFF等价于U>=65536**

字符占用 4 个字节。

也就是说，UTF-16 的编码长度要么是 2 个字节（U+0000 到 U+FFFF），要么是 4 个字节（U+010000 到 U+10FFFF）

**解码**

解码的过程也十分简单：如果一个字节的第一位是 0 ，则说明这个字节对应一个字符；如果一个字节的第一位1，那么连续有多少个 1，就表示该字符占用多少个字节。

**Java字符编码**

Java虚拟机规范中明确说明了java的char类型使用的编码方案是UTF-16

Java中常见的char基本位于unicode表的BMP平面，码点(编号)范围十进制： 0 ~65535

十六进制：0x0000~0xFFFF

所以char字符的编码统一使用unicode码点(编号)的16位二进制(不足用0补齐)作为存储数值。

简单来说就是字符编号就是其存储编码.

char强制转化为int数值就是其对应的十进制编号。

char c = '一';  
System.out.println((int)c);  
System.out.println(Integer.toHexString(c));  
输出结果为：  
19968  
4e00  
以上2个数值，就是这个字符在UNICODE码表中的位置，也就是代码点（code point）

JDK里面的native2ascii.exe可以把字符转为对应的unicode编码

https://www.baeldung.com/java-char-encoding

