link: null
title: 珠峰架构师成长计划
description: 最开始计算机只在美国用，八位的字节可以组合出256种不同状态。0-32种状态规定了特殊用途,一旦终端、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，如：
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=63 sentences=48, words=497

1. 字符发展历史

1.1 字节

- 计算机内部，所有信息最终都是一个二进制值
- 每一个二进制位（bit）有0和1两种状态，因此八个二进制位就可以组合出256种状态，这被称为一个字节(byte)

1.2 单位

- 8位 = 1字节
- 1024字节 = 1K
- 1024K = 1M
- 1024M = 1G
- 1024G = 1T

1.3 JavaScript中的进制

1.3.1 进制表示

```
let a = 0b10100;//&#x4E8C;&#x8FDB;&#x5236;
let b = 0o24;//&#x516B;&#x8FDB;&#x5236;
let c = 20;//&#x5341;&#x8FDB;&#x5236;
let d = 0x14;//&#x5341;&#x516D;&#x8FDB;&#x5236;
console.log(a == b);
console.log(b == c);
console.log(c == d);
```

1.3.2 进制转换

- 10进制转任意进制 10进制数.toString(目标进制)

```
console.log(c.toString(2));
```

- 任意进制转十进制 parseInt('任意进制字符串', 原始进制);

```
console.log(parseInt('10100', 2));
```

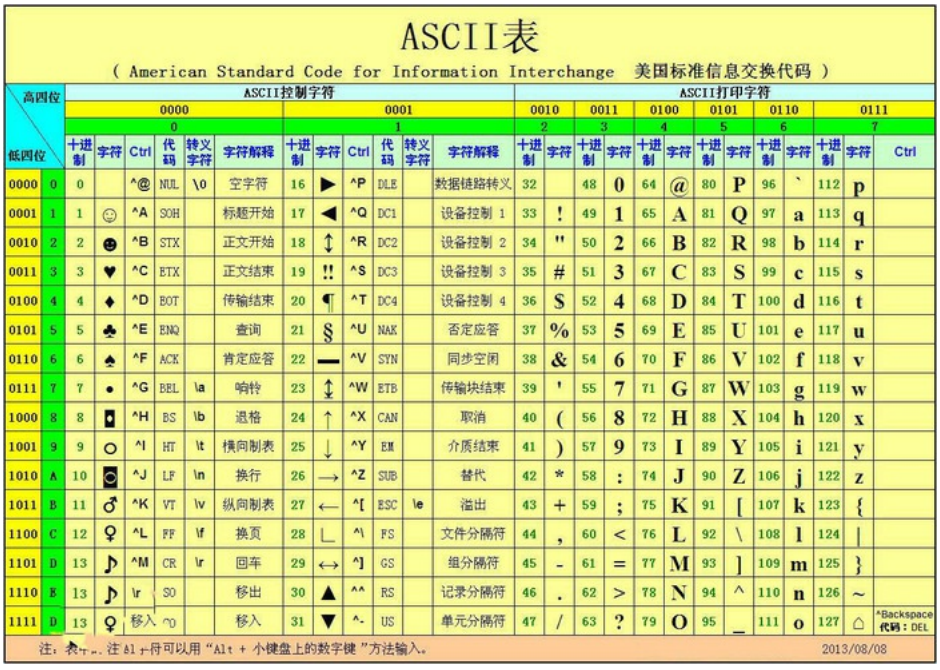
1.4 ASCII

最开始计算机只在美国用，八位的字节可以组合出256种不同状态。0-32种状态规定了特殊用途,一旦终端、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，如：

- 遇上0×10, 终端就换行；
- 遇上0×07, 终端就向人们嘟嘟叫；

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

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，最前面的一位统一规定为0



这个方案叫做 ASCII 编码

> American Standard Code for Information Interchange：美国信息互换标准代码

1.5 GB2312

后来西欧一些国家用的不是英文，它们的字母在ASCII里没有为了可以保存他们的文字，他们使用127号这后的空位来保存新的字母，一直编到了最后一位255。比如法语中的é的编码为130。当然了不同国家表示的符号也不一样，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)。

> 从128 到 255 这一页的字符集被称为扩展字符集。

中国为了表示汉字，把127号之后的符号取消了，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(247-161)*(254-161)=(7998)简体汉字了。
- 还把数学符号、日文假名和ASCII里原来就有的数字、标点和字母都重新编成两个字长的编码。这就是全角字符，127以下那些就叫半角字符。
- 把这种汉字方案叫做 GB2312。GB2312 是对 ASCII 的中文扩展

1.6 GBK

后来还是不够用，于是干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,又增加了近 20000 个新的汉字（包括繁体字）和符号。

1.7 GB18030 / DBCS

又加了几千个新的少数民族的字， GBK扩成了 GB18030 通称他们叫做 DBCS

> Double Byte Character Set：双字节字符集。

在 DBCS 系列标准里，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

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

1.8 Unicode

ISO 的国际组织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 的编码！ Unicode 当然是一个很大的集合，现在的规模可以容纳100多万个符号。

- International Organization for Standardization：国际标准化组织。
- Universal Multiple-Octet Coded Character Set，简称 UCS，俗称 Unicode

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 保持其原编码不变，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。

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

- 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。

1.9 UTF-8

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 标准出现了，

> Universal Character Set（UCS）Transfer Format：UTF编码

- UTF-8 就是在互联网上使用最广的一种 Unicode 的实现方式
- UTF-8就是每次以8个位为单位传输数据
- 而UTF-16就是每次 16 个位
- UTF-8 最大的一个特点，就是它是一种变长的编码方式
- Unicode 一个中文字符占 2 个字节，而 UTF-8 一个中文字符占 3 个字节
- UTF-8 是 Unicode 的实现方式之一

1.10 编码规则

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。

```
Unicode&#x7B26;&#x53F7;&#x8303;&#x56F4;      |        UTF-8&#x7F16;&#x7801;&#x65B9;&#x5F0F;
(&#x5341;&#x516D;&#x8FDB;&#x5236;)          |             &#xFF08;&#x4E8C;&#x8FDB;&#x5236;&#xFF09;
---------------------+-------------------------------------------
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
```

- Unicode编码 (http://www.chi2ko.com/tool/CJK.htm)

```
function transfer(num) {
  let ary = ['1110', '10', '10'];
  let binary = num.toString(2);
  ary[2] = ary[2]+binary.slice(binary.length-6);
  ary[1] = ary[1]+binary.slice(binary.length-12,binary.length-6);
  ary[0] = ary[0]+binary.slice(0,binary.length-12).padStart(4,'0');
  let result =  ary.join('');
  return parseInt(result,2).toString(16);
}
//&#x4E07;
let result = transfer(0x4E07);//E4B887
```

1.11 联通不如移动

```
C1 1100 0001
AA 1010 1010
CD 1100 1101
A8 1010 1000

00000000011010 10->006A(106)->j
00000011011010 00->0368(872)->?
```

- GB2312 (http://tools.jb51.net/table/gb2312)
- unicode编码表1 (http://blog.csdn.net/hherima/article/details/9045765)
- unicode编码表2 (http://blog.csdn.net/hherima/article/details/9045861)

1.12 文本编码

使用NodeJS编写前端工具时，操作得最多的是文本文件，因此也就涉及到了文件编码的处理问题。我们常用的文本编码有UTF8和GBK两种，并且UTF8文件还可能带有BOM。在读取不同编码的文本文件时，需要将文件内容转换为JS使用的UTF8编码字符串后才能正常处理。

1.12.1 BOM的移除

BOM用于标记一个文本文件使用Unicode编码，其本身是一个Unicode字符（"\uFEFF"），位于文本文件头部。在不同的Unicode编码下，BOM字符对应的二进制字节如下：

```
Bytes      Encoding
----------------------------
FE FF       UTF16BE
FF FE       UTF16LE
EF BB BF    UTF8
```

因此，我们可以根据文本文件头几个字节等于啥来判断文件是否包含BOM，以及使用哪种Unicode编码。但是，BOM字符虽然起到了标记文件编码的作用，其本身却不属于文件内容的一部分，如果读取文本文件时不去掉BOM，在某些使用场景下就会有问题。例如我们把几个JS文件合并成一个文件后，如果文件中间含有BOM字符，就会导致浏览器JS语法错误。因此，使用NodeJS读取文本文件时，一般需要去掉BOM

```
function readText(pathname) {
    var bin = fs.readFileSync(pathname);
    if (bin[0] === 0xEF && bin[1] === 0xBB && bin[2] === 0xBF) {
        bin = bin.slice(3);
    }
    return bin.toString('utf-8');
}
```

1.12.2 GBK转UTF8

NodeJS支持在读取文本文件时，或者在Buffer转换为字符串时指定文本编码，但遗憾的是，GBK编码不在NodeJS自身支持范围内。因此，一般我们借助iconv-lite这个三方包来转换编码。使用NPM下载该包后，我们可以按下边方式编写一个读取GBK文本文件的函数。

```
var iconv = require('iconv-lite');
function readGBKText(pathname) {
    var bin = fs.readFileSync(pathname);
    return iconv.decode(bin, 'gbk');
}
```

1.12 扩展阅读

- utf-8 (http://www.ruanyifeng.com/blog/2007/10/ascii_unicode_and_utf-8.html)
- 字符编码的故事 (https://tianziyao.github.io/2017/07/03/%E5%AD%97%E7%AC%A6%E7%BC%96%E7%A0%81%E7%9A%84%E6%95%85%E4%BA%8B/)