2021-11-25 06:00·[前端晚间课](https://www.toutiao.com/c/user/token/MS4wLjABAAAA_PvWQ7MCL76EBPAUKTbpWdZFlirOr6pABZJ2z8oKLEaZNKO3htUf-8al2LZXV1Wc/?source=tuwen_detail)

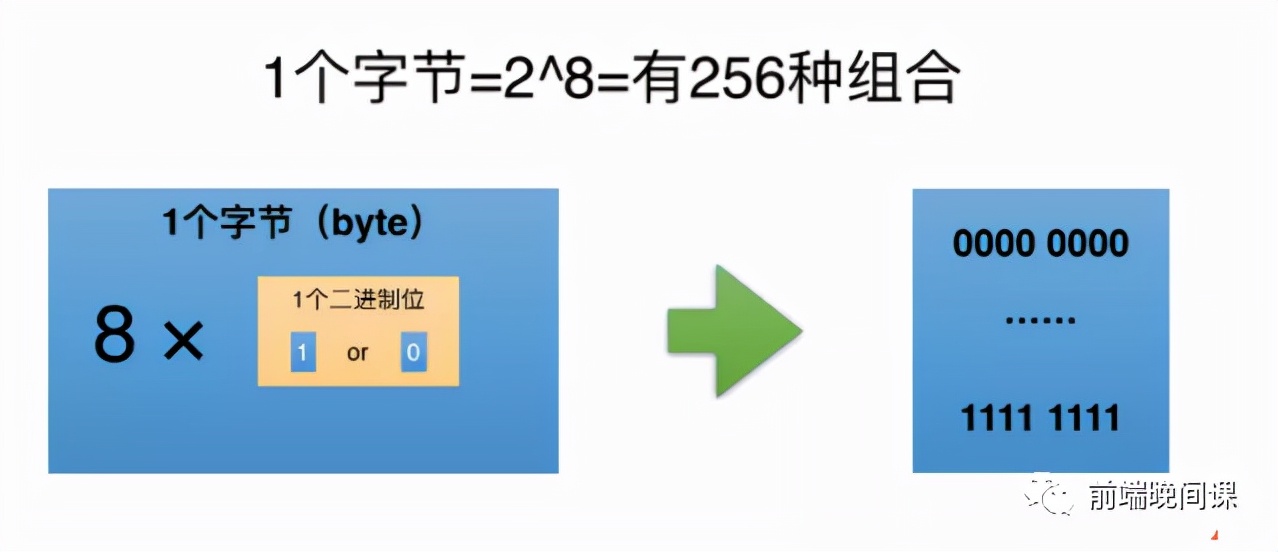
此文由前美团前端工程师@小鱼儿授权发布

**背景**

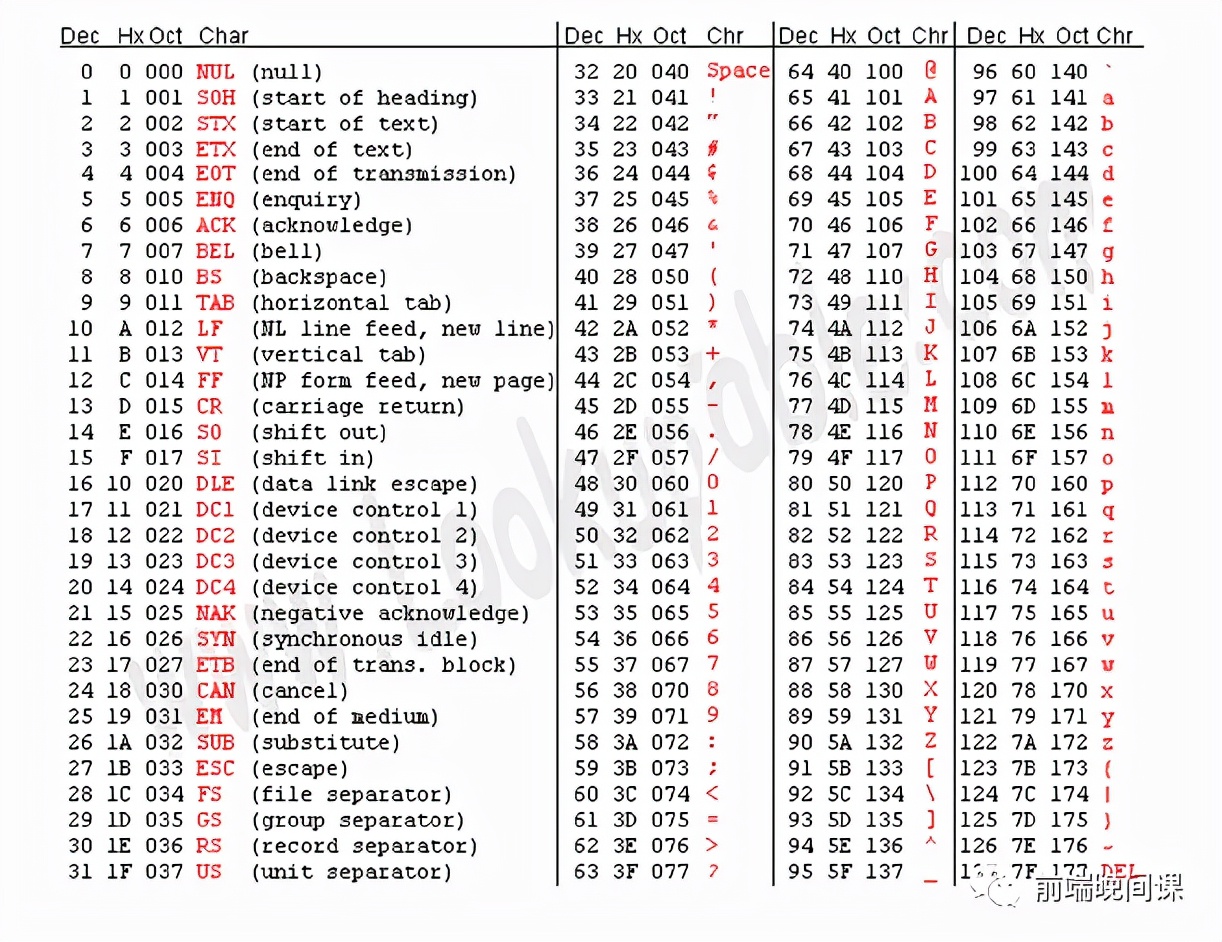
在日常开发中很少接触到字符的概念，大部分语言对字符的转换都已经封装的足够好，不需要开发人员过多考虑编码解码的问题。但是字符编码又经常在开发中遇见，所以这一篇文章就是解决，到底什么是字符集以及其编码。举个很简单的例子，调用字符串的length函数，其中的英文，汉字，emoji,长度分别是多少？length长度是如何计算的？iOS中的NSString，Javascript中的String在内存中的编码方式是什么？搞懂了字符编码，这些问题就迎刃而解了。

**编码、ASCII**

我们知道计算机只处理0和1，所有可见的文件，视频，音频等都是以二进制的形式存储和运算的。我们用八个二进制位表示一个字节，那么一个字节就可以代表256种“字符”。举个例子，字母“A”定义为65，用二进制表示是0100 0001。

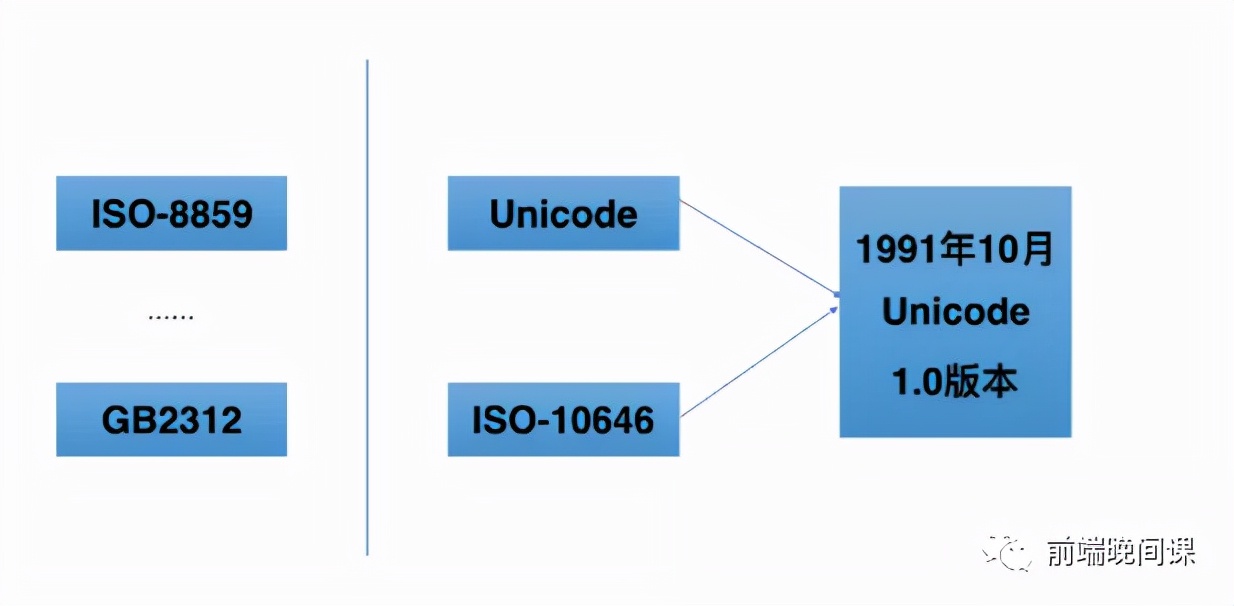


这种把A转换成0100 0001的形式就是一次映射的过程。再来看ASCII码一共有128个字符，那么就有128个映射。一个字节就足足的可以表示了。ASCII就是最早期的字符集。



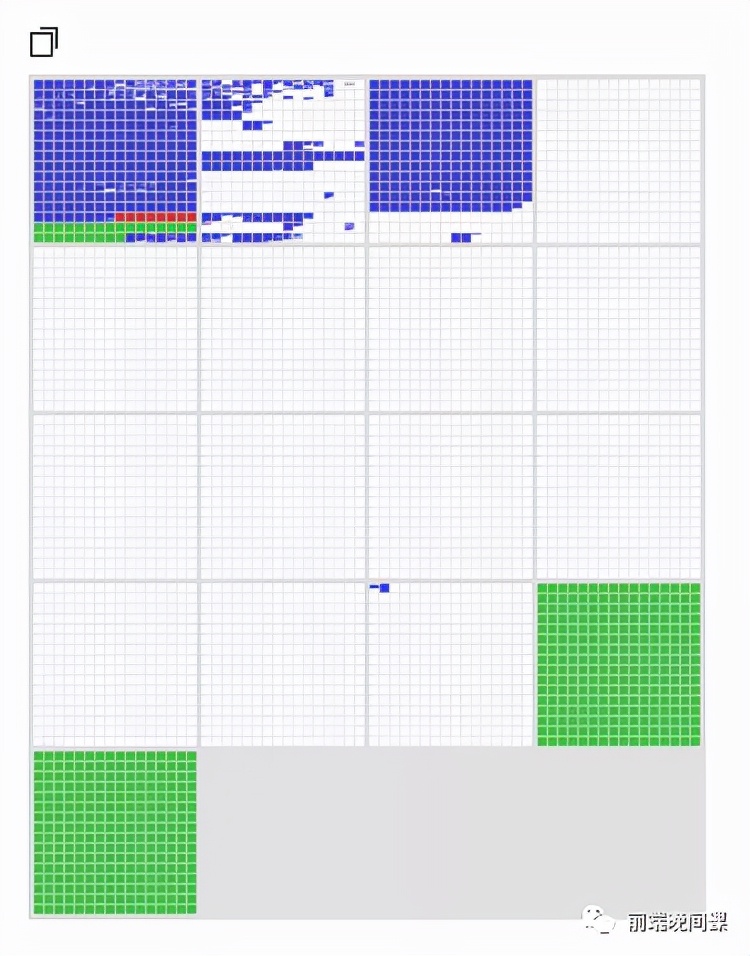
**字符集**

随着计算机的普及，需要做映射的字符越来越多，光常用汉字就几千个了，这时候ASCII码已经不够用了，涌现了很多字符集，ISO-8859，GB2312,GBK等，直到后来为了解决各个字符集各自为战的问题，分别产生了Unicode 组织和 ISO-10646工作小组，最后这两家组织也合并了，形成现如今的Unicode.

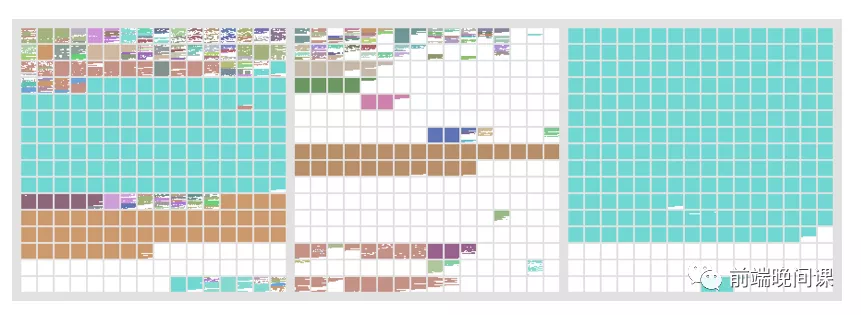


**Unicode**

Unicode是一种计算行业标准，用于对世界上大多数书写系统中表示的文本进行一致的编码，表示和处理。该标准由Unicode联盟维护，截至2019年5月，最新版本Unicode 12.1包含137994个字符的库，涵盖150个现代和历史脚本以及多个符号集和表情符号。Unicode标准的字符库与ISO / IEC 10646同步，并且两者的代码相同。Unicode也是一种字符集。通常会用U+十六进制表示，可存储0000 ~ 10FFFF 共 1114112 个值，2^16(65536)个号码组成一个平面，一共有17个平面，其中第一个0号平面占了绝大部分常用的字符。看下图可以比较直观的了解，其中每个最小的格子是一个字节即代表256个编码点，每个大格子有65536个编码点，蓝色区域是已被使用的区域，绿色是自用区，红色区域是代理区。



再将前三个格子放大，蓝绿色部分是汉字，棕色部分是朝鲜语，由这两张图可以最直观的了解Unicode存储空间。Unicode的实现方式有多种，其中最常用的是UTF-8和UTF-16,下面逐一介绍两个实现原理。

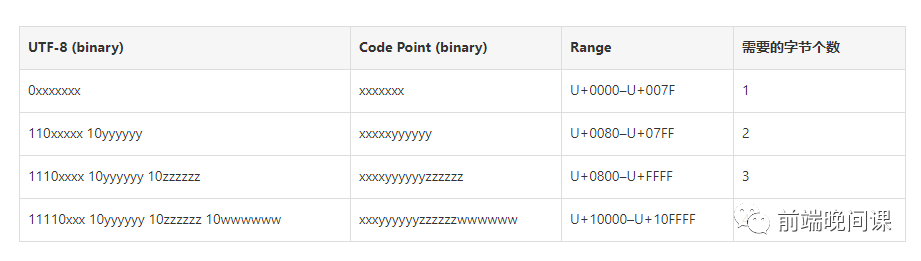


**UTF-8**

UTF-8是目前使用最广的Unicode编码方式，它是一种可变长的编码方式，从1个字节到4个字节不等。下图是它的编码规则：

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。



根据上表，解读 UTF-8 编码非常简单。如果一个字节的第一位是0，则这个字节单独就是一个字符；如果第一位是1，则连续有多少个1，就表示当前字符占用多少个字节。

下面，还是以汉字“严”为例，演示如何实现 UTF-8 编码。

“严”的 Unicode 是4E25（100111000100101），根据上表，可以发现4E25处在第三行的范围内（0000 0800 - 0000 FFFF），因此“严”的 UTF-8 编码需要三个字节，即格式是1110xxxx 10xxxxxx 10xxxxxx。然后，从“严”的最后一个二进制位开始，依次从后向前填入格式中的x，多出的位补0。这样就得到了，“严”的 UTF-8 编码是11100100 10111000 10100101，转换成十六进制就是E4B8A5。

**优点：**

1.兼容ASCII.

2.没有字节序问题。（后面会讲到字节序）

3.对于英文编码较短，占用空间小。

4.可变长，空间足够大

5.容错性好，中间丢失字节，后面的字节还是可以根据编码规则解码，不影响后面的字符生成。

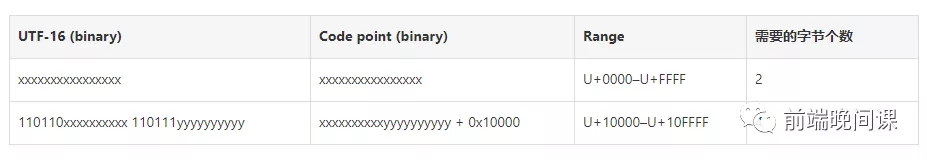
**缺点：**

1.对于中日韩的语言，一个字符需要三个字节表示，占用空间大。

2.计算长度效率低，由于是变长的，所以在计算字符串长度的时候执行效率比较低。

**UTF-16**

UTF-16也是经常用到的编码方式，同样它也是可变长的编码方式，下图是编码规则，字符长度2个字节或者4个字节表示一个字符。



随即就有了一个问题，当我们遇到两个字节，怎么看出它本身是一个字符，还是需要跟其他两个字节放在一起解读形成一个字符？

U+D800到U+DFFF是一个空段，这些码点不对应任何字符，编号大于U+0FFFF的字符，一半在U+D800到U+DBFF之间，一半在U+DC00到U+DFFF之间，当我们遇到两个字节，发现它的码点在U+D800到U+DBFF之间，就可以断定，紧跟在后面的两个字节的码点，应该在U+DC00到U+DFFF之间，这四个字节必须放在一起形成一个字节。刚好可以涵盖辅助面的字符。

**优点：**

1.由于是固定2个字节和4个字节，所以在计算字符串长度、执行索引操作时速度很快。

**缺点：**

1.UTF-16 能表示的字符数有 6 万多，但是实际上目前 Unicode 5.0 收录的字符已经达到 99024 个字符，早已超过 UTF-16 的存储范围。

2.UTF-16 存在字节序问题（大端和小端）使用时需要提前协定好。

3.容错性差，因为存在字节序的问题，如果中间某一个字节丢失时可能会导致后面的解码错误。

**UTF-32**

4 个字节表示一个代码值，固定长度,多出来的部分前面补0,这种编码方式占空间较多，使用场景很少。

**字节序**

字节序是指数据在存储器中的存放顺序，分为大端和小端两种。之所以会存在字节序的问题是因为寄存器的长度要大于一个字节，不同的操作系统读取字节的顺序不一样，

大端模式，是指数据的高字节在前，保存在内存的低地址中，与人类的读写法一致，数据的低字节在后，保存在内存的高地址中,文件前缀FE FF。（Mac OS是大端模式）

小端模式，是指数据的高字节在后，保存在内存的高地址中，而数据的低字节在前，保存在内存的低地址中,文件前缀FF FE。（x86和一般的OS（如windows，FreeBSD,Linux）使用的是小端模式）。



所以UTF-16存在一个字节序的问题，需要在文件前面声明，而UTF-8不存在这个问题，原因在于UTF-8的最小编码单位是1个字节，不会存在两个字节谁在高位谁在地位的问题。

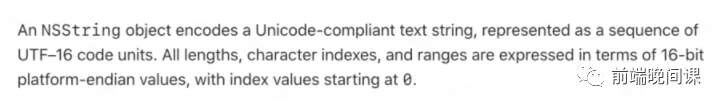
还是以汉字“严”为例，Unicode 码是4E25，需要用两个字节存储，一个字节是4E，另一个字节是25。存储的时候，4E在前，25在后，这就是 Big endian 方式（4E 25）。25在前，4E在后，这是 Little endian 方式（25 4E）。

**用途**

UTF-8，广泛用于数据存储及传输：例如html文档中的<meta charset="UTF-8">，以及python文件当出现中文的时候会在顶部加上“# coding: UTF-8”等**。**

UTF-16，而一些流行语言比如Java、Javascript、Python、Objective-C等字符串内部字符串都用UTF-16编码.在计算字符串长度搜索是的效率较好。

**Objective-C中的NSString**



**Java中的String类**



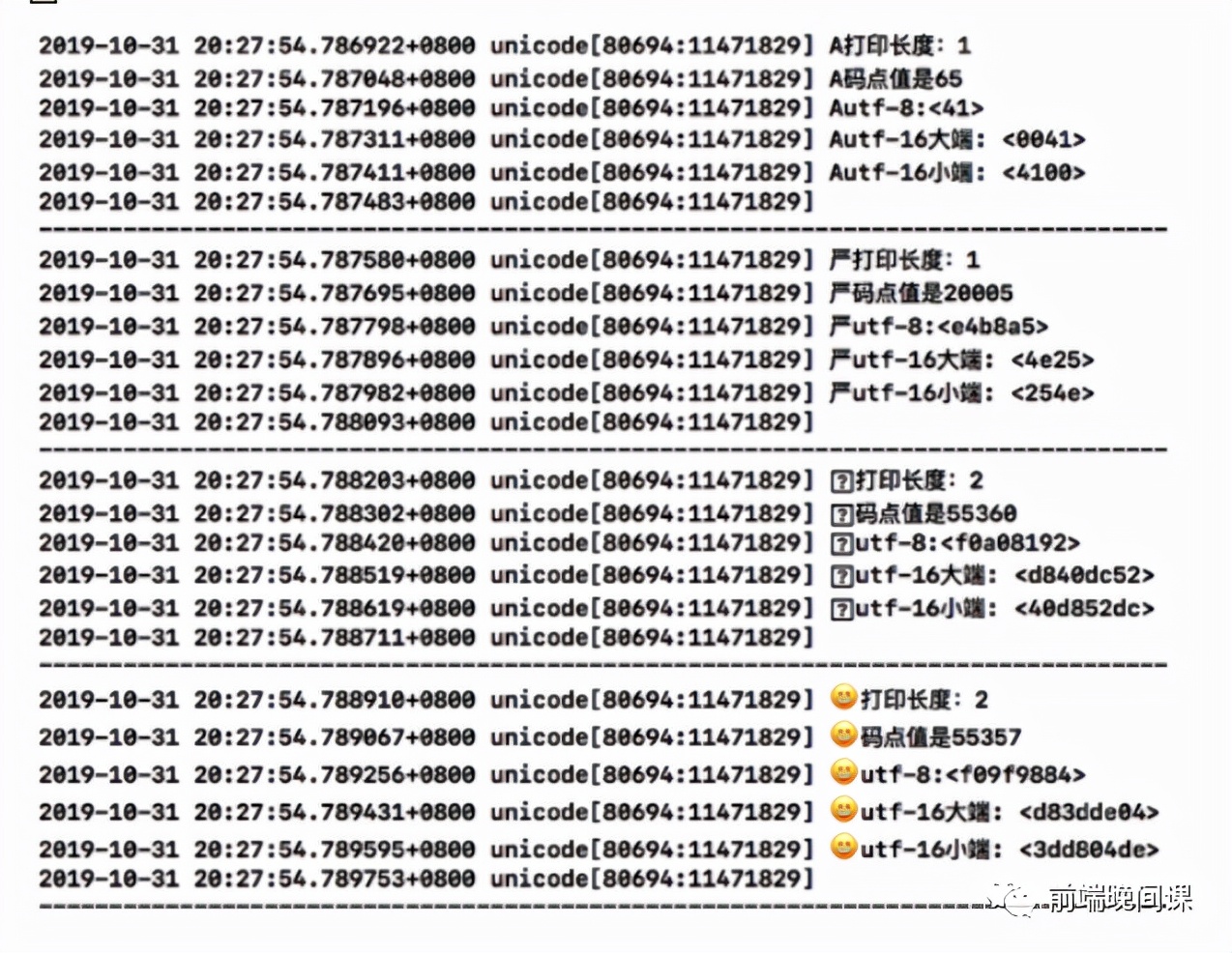
**实践**

**1.日常字母，汉字，表情分别用**UTF-8**和**UTF-16**表示分别用多少字节？**

字母UTF-8用一个字节，UTF-16用两个字节。

大部分的汉字UTF-8编码后由三个字节如下图的“严”是e4b8a5,而用UTF-16编码仅用2个字节即4e25。

大部分表情等特殊符号UTF-8编码后占4个字节，UTF-16编码后也占4个字节。



**2.字符计算长度length方法是怎么计算的?**

像NSString,java,javascript等语言，由于是UTF-16编码的，在计算长度的时候由总字节/2得来的。

**3.实际开发时，如果想计算字符实际长度该怎么计算？**

1）一种是通过判断码点所在位置进行判断,只要落在0xD800到0xDBFF的区间，就要连同后面2个字节一起读取.

var index = -1;

var string = '12';

var length = string.length;

var output = [];

while (++index < length) {

var charCode = string.charCodeAt(index);

var character = string.charAt(index);

if (charCode >= 55296 && charCode <= 56319) {

output.push(character + string.charAt(++index));

} else {

output.push(character);

}

}

console.log(output) //["", "1", "2"]

consolo.log(0xD800 ===55296) // true

2.）ECMAScript 6版本 增强了对Unicode的支持，基本解决了这个问题。

let s = '12';

let output = [];

for(let s of string ){

output.push(s)

}

console.log(output) //["", "1", "2"]

Array.from(string).length

**4.javascript字符串和码点之间的转换方法？**

* String.fromCodePoint()：从Unicode码点返回对应字符

console.log(String.fromCodePoint(9731, 9733, 9842, 0x2F804));

// expected output: "☃★♲你"

* String.prototype.codePointAt()：从字符返回对应的码点

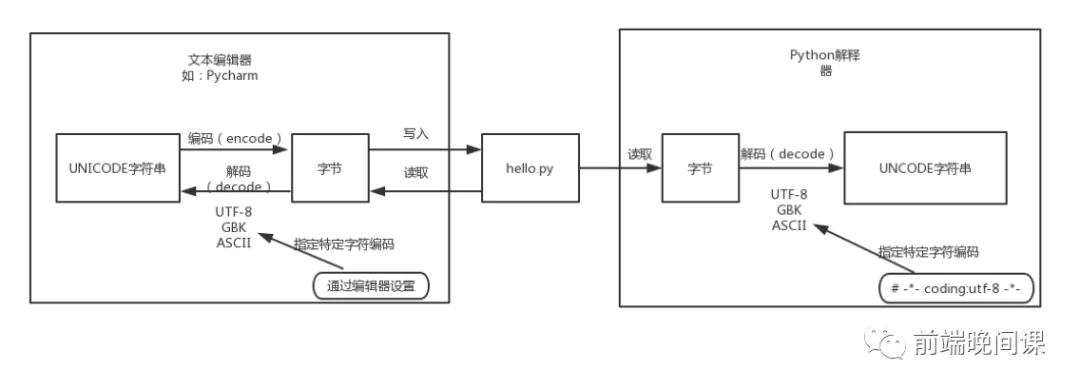
var icons = '☃★♲';

console.log(icons.codePointAt(1));

// expected output: "9733"

**5.文件编码是**UTF-8**,和字符串**UTF-16**编码之间有什么关系？**

这里以python为例，我们都知道，磁盘上的文件都是以二进制格式存放的，其中文本文件都是以某种特定编码的字节形式存放的。对于程序源代码文件的字符编码是由编辑器指定的，比如我们使用Pycharm来编写Python程序时会指定工程编码和文件编码为UTF-8，那么Python代码被保存到磁盘时就会被转换为UTF-8编码对应的字节（encode过程）后写入磁盘。当执行Python代码文件中的代码时，Python解释器在读取Python代码文件中的字节串之后，需要将其转换为Unicode字符串（decode过程）之后才执行后续操作。



**总结**

上面讲了Unicode字符集的起源，它的三种编码方式，UTF-8,UTF-16,UTF-32,编码空间，编码规则，优缺点以及用途，之后结合实际场景应用介绍了js中一些实践方法，不同语言会有自己的方式，还有就是文件编码和执行时字符编码。这些知识点在实际开发中不常见，但是深入了解其原理，对日常开发和解决问题会有帮助。

http://reedbeta.com/blog/programmers-intro-to-unicode/

https://zh.wikipedia.org/zh-cn/Unicode#%E6%A8%99%E6%BA%96

http://www.ruanyifeng.com/blog/2007/10/ascii\_unicode\_and\_utf-8.html

http://www.ruanyifeng.com/blog/2014/12/unicode.html

https://www.cnblogs.com/yyds/p/6171340.html