编码简介 utf8、utf16 以及其它编码

文江博客
2023-01-19 18:36 内蒙古 | 前端开发工程师

UTF-8

UTF-8（8-bit Unicode Transformation Format）

是一种针对 Unicode 的可变长度字元编码，也是一种前缀码。它可以用来表示 Unicode 标准中的任何字元，且其编码中的第一个字节仍与ASCII兼容，这使得原来处理ASCII字元的软件无须或只须做少部分修改，即可继续使用。UTF-8 使用一至六个字节为每个字符编码（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

）。

编码规则

Unicode 字元的比特被分区为数个部分，并分配到UTF-8的字节串中较低的比特的位置。在 U+0080 的以下字元都使用内含其字元的单字节编码。这些编码正好对应7比特的ASCII字符。在其他情况，有可能需要多达4个字元组来表示一个字元。这些多字节的最高有效比特会设置成1，以防止与7比特的 ASCII 字符混淆，并保持标准的字节主导字符串运作顺利。

代码范围（十六进制）	标量值（scalar value，二进制）	UTF-8（二进制／十六进制）	注释
000000 - 00007F	00000000 00000000 0zzzzzzz	0zzzzzzz（00-7F）	ASCII字元范围，字节由零开始
000080 - 0007FF	00000000 00000yyy yyzzzzzz	110yyyyy（C0-DF) 10zzzzzz(80-BF）（五个y；六个z）	第一个字节由110开始，接着的字节由10开始
000800 - 00D7FF 00E000 - 00FFFF	00000000 xxxxyyyy yyzzzzzz	1110xxxx(E0-EF) 10yyyyyy 10zzzzzz（四个x；六个y；六个z）	第一个字节由1110开始，接着的字节由10开始

010000 - 10FFFF	000wwwxx xxxxyy yy yyzzzzzz	11110www(F0-F7) 10xxxxxx 10yyyyyy 10zzzzzz（三个w；六个x；六个y；六个z）	将由11110开始，接着的字节由10开始

以实例来解释 utf8 编码：

```
// 假设字符串 '10h我'var buf = new Buffer('10h我'); // buf: <Buffer 31 30
```

规范更新

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。根据规范，以下字节值将无法出现在合法 UTF-8 序列中：

编码（二进制）	编码（十六进制）	注释
1100000x	C0, C1	过长编码：双字节序列的头字节，但码点 <= 127
1111111x	FE, FF	无法达到：7或8字节序列的头字节
111110xx-1111110x	F8, F9, FA, FB, FC, FD	被RFC 3629规范：5或6字节序列的头字节
11110101-1111011x	F5, F6, F7	被RFC 3629规范：码点超过10FFFF的头字节

代码实现转换 UTF8 编码的 Bytes 为字符串

```
function convertBytesToUTF8(bytes, maxBytes) {  var index = 0;  maxBytes
```

代码中看出需要注意2点：

0xEF,0xBB,0xBF 是 **BOM（Byte order mark）**，UTF8 编码允许 BOM 存在，但不依赖也不推荐使用 BOM。不能正确识别 BOM 时，就会输出 ï»¿。
1-4 字节的不同处理完全遵从 RFC 3629 规范，剔除了不合法点字符。

code point: 码位

code unit：码元

UTF-16

UTF-16（16-bit Unicode Transformation Format）

是 Unicode 字符编码五层次模型的第三层：字符编码表（Character Encoding Form，也称为 storage format）的一种实现方式。即把 Unicode 字符集的抽象码位映射为16位长的整数（即码元）的序列，用于数据存储或传递。Unicode 字符的码位，需要

1个或者2个16位长的码元

来表示，因此这是一个变长表示。Unicode 的编码空间从 U+0000 到 U+10FFFF，共有 1,112,064 个码位（code point）可用来映射字符。Unicode 的编码空间可以划分为17个平面（plane），每个平面包含 216（65,536）个码位。17个平面的码位可表示为从U+xx0000到U+xxFFFF，其中xx表示十六进制值从00（16进制）到10（16进制），共计17个平面。第一个平面称为基本多语言平面（Basic Multilingual Plane, BMP），或称第零平面（Plane 0）。其他平面称为辅助平面（Supplementary Planes）。基本多语言平面内，从 U+D800 到U+DFFF之间的码位区块是永久保留不映射到Unicode字符。UTF-16就利用保留下来的 0xD800-0xDFFF 区段的码位来对辅助平面的字符的码位进行编码。

编码规则

U+0000 ~ U+D7FF 和 U+E000 ~ U+FFFF

这个范围即基本多语言平面（Basic Multilingual Plane, BMP），包含了最常用的字符，包含的码位范围是 U+0000 到 U+FFFF，只需要一个16位的码元即可表示。

U+10000 ~ U+10FFFF

其它平面（叫做辅助平面，Supplementary Planes）中的码位，在UTF-16中被编码为一对16位长的码元（即32bit,4Bytes），称作代理对（surrogate pair），具体方法是：

码位（0x10000~0x10FFFF）减去0x10000（BMP范围），剩下20比特长的数字，范围是0..0xFFFFF。
高位的10比特（数值范围为0..0x03FF）被加上0xD800得到第一个16位长的码元或称作高位代理（high surrogate），值的范围是0xD800..0xDBFF。
低位的10比特的值（值的范围也是0..0x3FF）被加上0xDC00得到第二个16位长的码元或称作低位代理（low surrogate），现在值的范围是0xDC00..0xDFFF。

由于高位代理比低位代理的值要小，所以为了避免混淆使用，Unicode标准现在称高位代理为前导代理（lead surrogates），低位代理为后尾代理（trail surrogates）。上述算法可理解为：辅助平面中的码位从U+10000到U+10FFFF，共计FFFFF个，即2^20=1,048,576个，需要20位来表示。如果用两个16位长的整数组成的序列来表示，第一个整数（称为前导代理）要容纳上述20位的前10位，第二个整数（称为后尾代理）容纳上述20位的后10位。还要能根据16位整数的值直接判明属于前导整数代理的值的范围（2^10=1024），还是后尾整数代理的值的范围（也是2^10=1024）。因此，需要在基本多语言平面中保留不对应于Unicode字符的2048个码位，就足以容纳前导代理与后尾代理所需要的编码空间。这对于基本多语言平面总计65536个码位来说，仅占3.125%.由于前导代理、后尾代理、BMP中的有效字符的码位，三者互不重叠，搜索是简单的：一个字符编码的一部分不可能与另一个字符编码的不同部分相重叠。这意味着UTF-16是自同步（self-synchronizing）:可以通过仅检查一个码元就可以判定给定字符的下一个字符的起始码元。UTF-8也有类似优点，但许多早期的编码模式就不是这样，必须从头开始分析文本才能确定不同字符的码元的边界。

U+D800 ~ U+DFFF

Unicode标准规定U+D800..U+DFFF的值不对应于任何字符。但是在使用UCS-2的时代，U+D800..U+DFFF内的值被占用，用于某些字符的映射。

UTF-16与UCS-2的关系

UTF-16来源于UCS-2 。在1980年代末，人们希望为世界上所有字符（Universal Character Set——

UCS

）开发一个统一的编码。最初的目标是扩展8位的ascii码为16位比特（216 = 65,536）。可惜，很快16位被证明不足够包括所有字符，共同开发UCS的IEEE和Unicode Consortium产生分歧：IEEE引入固定32位比特来编码字符（UCS-4），Unicode Consortium推出UTF-16。UTF-16可看成是UCS-2的父集。在没有辅助平面字符（surrogate code points）前，UTF-16与UCS-2所指的是同一的意思。但当引入辅助平面字符后，就称为UTF-16了。现在若有软件声称自己支持UCS-2编码，那其实是暗指它不能支持在UTF-16中超过2字节的字集。对于小于0x10000的UCS码，UTF-16编码就等于UCS码。

ASCII 与 ISO-8859-1

ASCII码是最基础的编码，共定义了128个字符（0-127）。这些字符分为控制字符和可显示字符（26个基本拉丁字母、阿拉伯数目字和英式标点符号）。ASCII使用了8位2进制，但最高位始终为0，并没有有效利用。而最高位置1，在空置的0xA0-0xFF的范围内，加入96个字母及符号，用以供使用附加符号的拉丁字母语言使用——这就是

ISO-8859-1

编码。

ISO-8859-1

编码兼容ASCII编码，但因它没有法语使用的 œ、Œ、Ÿ 三个字母及芬兰语使用的 Š、š、Ž、ž，故于1998年被ISO/IEC 8859-15所取代。另外，结合UTF8定义，可以看到，UTF8并没有兼容

ISO-8859-1

。

感觉上面讲的过于概念化，有时隔一阵来看，自己都觉得太拗口复杂了。

从 code point 这个核心点来讲一讲字符编码技术

对

code point

大家应该有印象：

String.fromCodePoint

就是从有序的 code point 序列得到对应的字符串。

```
String.fromCodePoint(num1[, ...[, numN]]) // ---> charactersString.fromCo
```

同时，有个大家更熟悉的类似 API

`String.fromCharCode`

，从 Unicode values 序列得到对应的字符串。

```
String.fromCharCode(0x2F804, 0x30)// "⍰0"
```

从不同的输出结果，大家应该都有一定的推测和预感了。

弄懂 `String.fromCharCode` 和 `String.fromCodePoint` 的区别

从 MDN 看，

`fromCharCode`

的定义并不好理解， Unicode values 没有准确的概念好理解。于是去翻spec，拿到了一个更精准易懂的定义：

Returns a String value containing as many characters as the number of arguments. Each argument specifies one character of the resulting String, with the first argument specifying the first character, and so on, from left to right. An argument is converted to a character by applying the operation `ToUint16` and regarding the resulting 16-bit integer as the **code unit** value of a character. If no arguments are supplied, the result is the empty String.

定义里引入了 code unit （码元）的概念。简单点说，在 Unicdoe 里，code unit 就是特定（位数）序列的bits：

UCS-4，code unit 是 4 字节的 bits；
UTF8，code unit 从 1/2/3/4 字节的 bits；
UCS-2/UTF16，code unit 就是 2 字节的 bits。UTF16 可能由 1 或 2 个 code unit 表示某个字符。

对 JavaScript 而言，code unit 就是 2 字节的 bits。所以：

```
// 对 0x2F804 执行 ToUint16 就是 0xF804// 所以，String.fromCharCode(0x2F8
```

所以我们可以得出结论：

0x0 -- 0xFFFF，String.fromCharCode 和 String.fromCodePoint 返回相同结果；
0x10000 -- 0x10FFFF（unicode 最大值），只有 String.fromCodePoint 可以得到正确字符。

扩展阅读

https://mathiasbynens.be/notes/javascript-encoding

所以 code point 是什么?

在字符编码技术（character encoding terminology）领域，code point 就是组成 code space 的任意数值（numerical values）。

对 ASCII 而言，就是 128 个 code points，范围从 0 - 127；
对 Unicode 而言，就是 1,114,112 个 code points，范围从 0x0 - 0x10FFFF。

那么这么多字符编码（character encoding）是什么?

本质上讲，字符编码就是

建立所有 code points 到 binary bytes 的映射规则

。

有一串 0/1 序列，我们可以转换为相应的 code point;
有 code point，我们知道怎么转换成 0/1 序列来存储传输。

发表评论