

参考链接：字符编码系列大纲“<https://segmentfault.com/a/1190000012470198>”

参考链接：https://blog.csdn.net/qq_36772866/article/details/88828478

参考链接：<https://mp.weixin.qq.com/s/Dp-zsxbAlHtNto1BmXyG5Q>

参考链接：<https://mp.weixin.qq.com/s/QjU9lSekpbaF7fugZbyzkg>

一、UNICODE编码

查看字符对应Unicode编码的网址：

[http://www.fileformat.info/info/unicode/char/search.htm?q=%E4%BD%A0&preview=entity\[1\]](http://www.fileformat.info/info/unicode/char/search.htm?q=%E4%BD%A0&preview=entity[1])

Unicode 是一个独立的字符集，它并不是和编码绑定的，你可以采用第一种方案，为每个字符分配固定长度的内存，也可以采用第二种方案，为每个字符分配尽量少的内存。

需要注意的是，Unicode 只是一个字符集，在制定的时候并没有考虑编码的问题，所以采用第二种方案时，就不能从字符集本身下手了，只能从字符编号下手，这样在存储和读取时都要进行适当的转换。

Unicode编码定义了这个世界上几乎所有字符（就是你眼睛看的字符比如ABC，汉字等）的数字表示，而且Unicode还兼容了很多老版本的编码规范，例如你熟悉的 ASCII码。

Unicode 可以使用的编码有三种，分别是：

- UTF-8：一种变长的编码方案，使用 1~6 个字节来存储；
- UTF-32：一种固定长度的编码方案，不管字符编号大小，始终使用 4 个字节来存储；
- UTF-16：介于 UTF-8 和 UTF-32 之间，使用 2 个或者 4 个字节来存储，长度既固定又可变。

UTF 是 Unicode Transformation Format 的缩写，意思是“Unicode转换格式”，后面的数字表明至少使用多少个比特位（Bit）来存储字符。

0.Unicode基本概念

码点：我们国家的每一个人都对应唯一的一个身份证号，而Unicode也为了每个字符发了一张身份证，这张“身份证”上有一串唯一的数字ID确定了这个字符。这串数字在整个计算机的世界具有唯一性，Unicode给这串数字ID起了个名字叫 [码点] 。

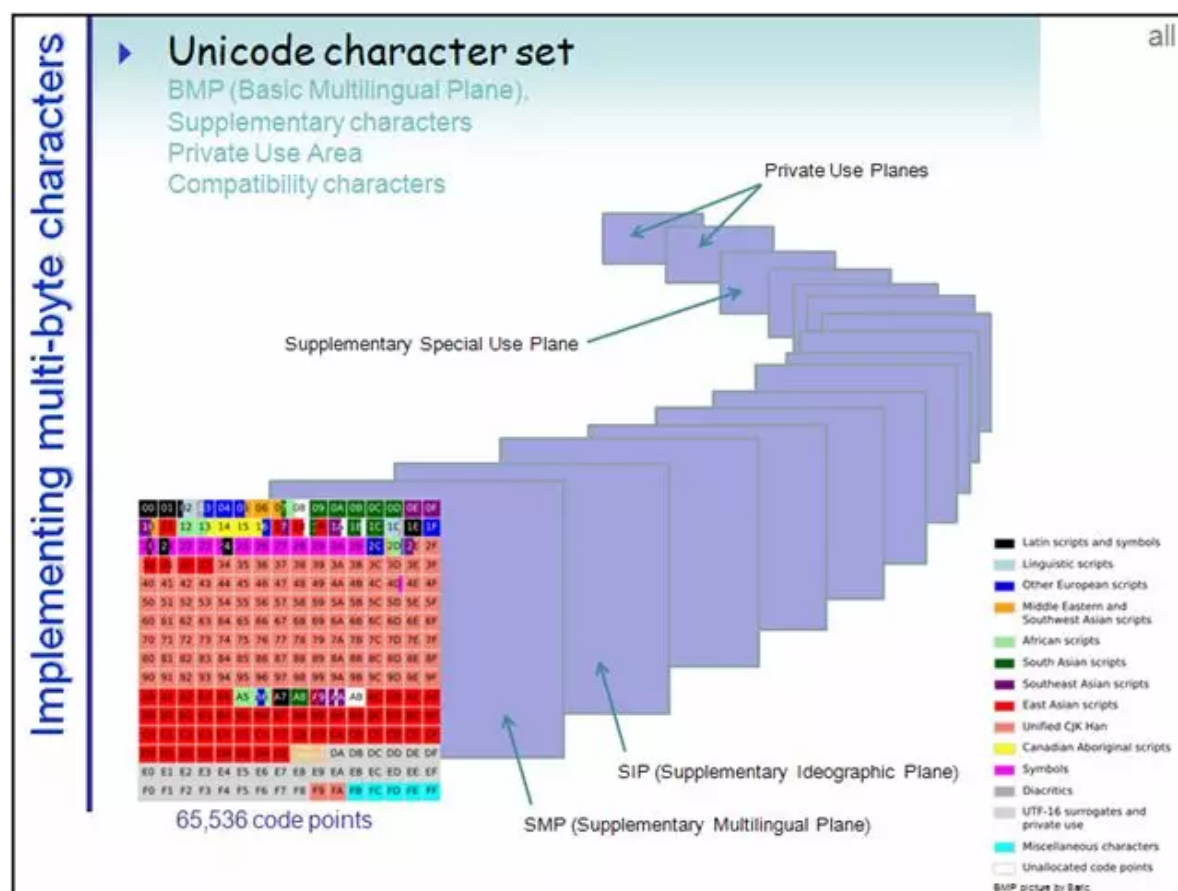
先来说一声码点是如何表示的：U+XXXXXX 是码点的表示形式，X 代表一个十六进制数字，可以有 4-6 位，不足 4 位前补 0 补足 4 位，超过则按是几位就是几位。字符A的ASCII码是众所周知是65吧，将65转换成16进制就是41 ($16 \times 4 + (16^0) \times 1 = 65$)，按照规则前面补 0，那么字符A的码点表示就是U+0041，依次类推B的码点表示就是U+0042...等等，汉字"你"的字符表示是“U+4F60”...

码点的取值范围：码点的取值范围目前是 U+0000 ~ U+10FFFF，理论大小为 $10FFFF+1=110000$ （为啥+1，因为从0开始嘛）。后一个 1代表是 65536（16的4次方），因为是 16 进制，所以前一个 1 是后一个 1 的 16 倍，所以总共有 $1 \times 16 + 1 = 17$ 个的 65536 的大小，粗略估算为 $17 \times 6万 = 102万$ ，所以这是一个百万级别的数。

为了更好分类管理如此庞大的码点数，把每 65536 个码点作为一个平面，总共 17 个平面。

而我们说的代理区就在平面里面，而平面又有很多讲究。为了帮你搞懂代理区，先来聊一聊这平面的事。

平面：由前面可知，码点的全部范围可以均分成 17 个 65536 大小的部分，这里的每一个部分就是一个平面（Plane）。编号从 0 开始，第一个平面称为 Plane 0。

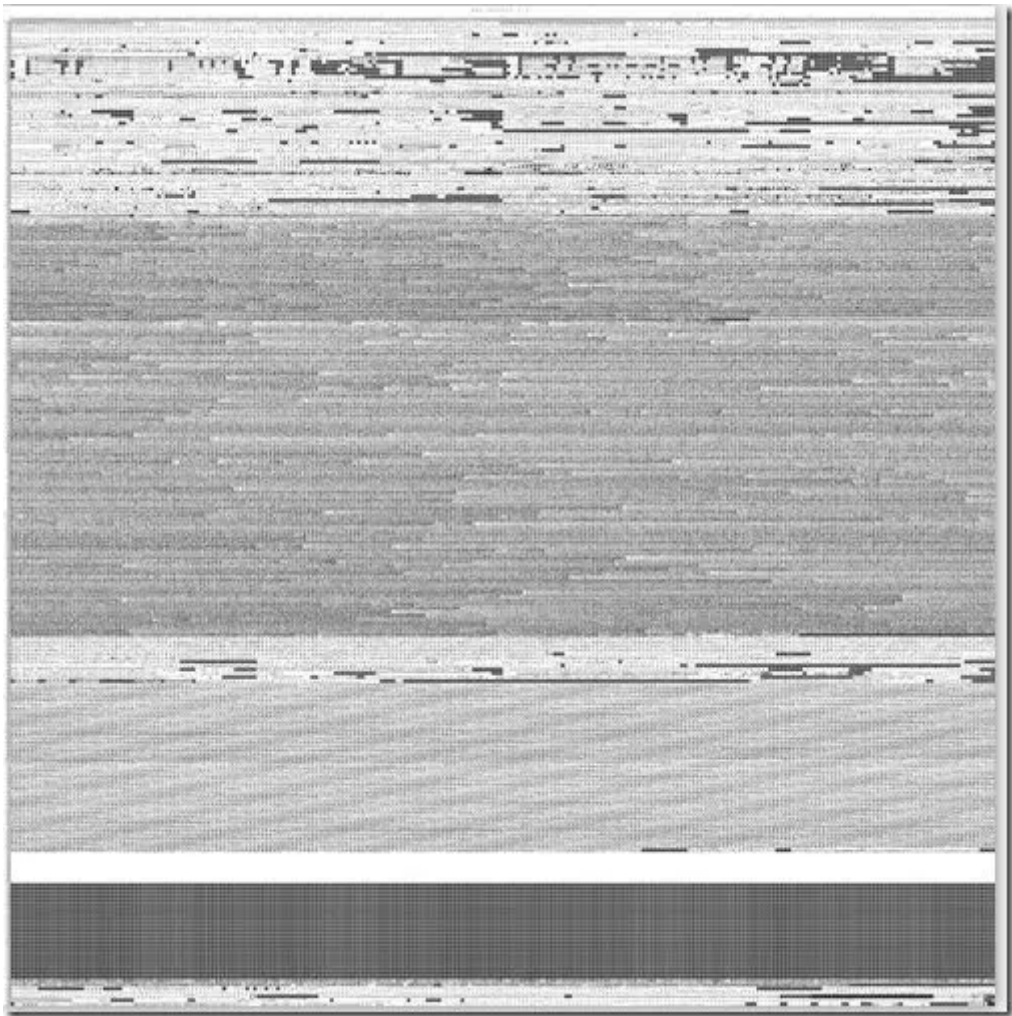


BMP：第一个平面即是 **BMP（Basic Multilingual Plane 基本多语言平面）**，也叫 Plane 0，它的码点范围是 U+0000 ~ U+FFFF。这也是我们最常用的平面，日常用到的字符绝大多数都落在这个平面内。就是上图中花花绿绿的平面。

UTF-16 只需要用两字节编码此平面内的字符。

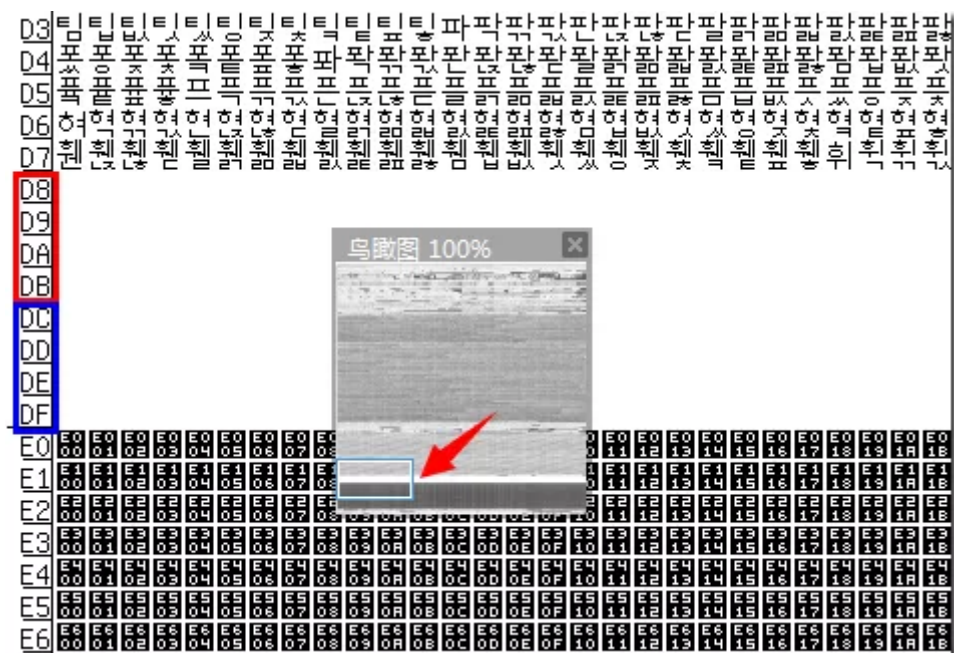
最常用的 BMP，它的码点空间也有 6 万多，如果把这些字符都放到一张图片上，会是什么情况呢？GNU Unifont 就制作了一张这样的图片。见<http://unifoundry.com/pub/unifont-7.0.03/unifont-7.0.03.bmp>

下图是它的一个缩略版本：



SP: 后续的 16 个平面称为 **SP (Supplementary Planes)**。显然，这些码点已经是超过 U+FFFF 的了，所以已经超过了 16 位空间的理论上限，对于这些平面内的字符，UTF-16 采用了四字节编码。

代理区: 你可能还注意到前面的 BMP 缩略图中有一片空白，这白花花一片亮瞎了我们的猿眼的是啥呢？这就是所谓的代理区 (**Surrogate Area**) 了。



可以看到这段空白从 D8~DF。其中前面的红色部分 D800–DBFF 属于高代理区（High Surrogate Area），后面的蓝色部分 DC00–DFFF 属于低代理区（Low Surrogate Area），各自的大小均为 4×256=1024。

在UTF-16中会详细讲解如何使用四字节来编码BMP以外的字符的。

1.UTF-8编码

UTF-8 的编码规则很简单：如果只有一个字节，那么最高的比特位为 0；如果有多个字节，那么第一个字节从最高位开始，连续有几个比特位的值为 1，就使用几个字节编码，剩下的字节均以 10 开头。

Unicode和UTF8转换关系表如下：

| Unicode编码 | UTF-8字节流 |
|----------------------------|---|
| U+00000000 – U+0000007F | 0xxxxxxx |
| U+00000080 – U+000007FF | 110xxxxx 10xxxxxx |
| U+00000800 – U+0000FFFF | 1110xxxx 10xxxxxx 10xxxxxx |
| U+00010000 – U+001FFFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx |
| U+00200000 – U+03FFFFFF | 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx |
| U+04000000 – U+7FFFFFFF | 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx |

- 0xxxxxxx：单字节编码形式，这和 ASCII 编码完全一样，因此 UTF-8 是兼容 ASCII 的；
- 110xxxxx 10xxxxxx：双字节编码形式；
- 1110xxxx 10xxxxxx 10xxxxxx：三字节编码形式；
- 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx：四字节编码形式。

xxx 就用来存储 Unicode 中的字符编号。

下面是一些字符的编码实例（绿色部分表示本来的 Unicode 编号）：

| 字符 | N | æ | 齐 |
|------------------|----------|-------------------|----------------------------|
| Unicode 编号（二进制） | 01001110 | 11100110 | 00101110 11101100 |
| Unicode 编号（十六进制） | 4E | E6 | 2E EC |
| UTF-8 编码（二进制） | 01001110 | 11000011 10100110 | 11100010 10111011 10101100 |
| UTF-8 编码（十六进制） | 4E | C3 A6 | E2 BB AC |

对于常用的字符，它的 Unicode 编号范围是 0 ~ FFFF，用 1~3 个字节足以存储，只有及其罕见，或者只有少数地区使用的字符才需要 4~6个字节存储。

UTF-8和码点如何互相转换？

| Unicode编码(十六进制) | UTF-8 字节流(二进制) |
|-----------------|----------------------------------|
| 000000-00007F | 0xxxxxxx |
| 000080-0007FF | 110xxxxx 10xxxxxx |
| 000800-00FFFF | 1110xxxx 10xxxxxx 10xxxxxx |
| 010000-10FFFF | 11110xxx10xxxxxx10xxxxxx10xxxxxx |

对于Unicode的编码首先确定它的范围，找到它是对应的几字节。

对于0x00-0x7F之间的字符，UTF-8编码与[ASCII编码]完全相同。

“汉”字的Unicode编码是0x6C49。0x6C49在0x0800-0xFFFF之间，使用3字节模板：1110xxxx 10xxxxxx 10xxxxxx。将0x6C49写成二进制是：0110 1100 0100 1001，用这个比特流依次代替模板中的x，得到：**11100110 10110001 10001001**，即E6 B1 89。

2.UTF-32编码

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。

3.UTF-16

UFT-16 比较奇葩，它使用 2 个或者 4 个字节来存储。

对于 Unicode 编号范围在 0 ~ FFFF 之间的字符，UTF-16 使用两个字节存储，并且直接存储 Unicode 编号，不用进行编码转换，这跟 UTF-32 非常类似。

对于 Unicode 编号范围在 10000~10FFFF 之间的字符，UTF-16 使用四个字节存储，具体来说就是：将字符编号的所有比特位分成两部分，较高的一些比特位用一个值介于 D800~DBFF 之间的双字节存储，较低的一些比特位（剩下的比特位）用一个值介于 DC00~DFFF 之间的双字节存储。

如果你不理解什么意思，请看下面的表格：

| Unicode 编号范围 (十六进制) | 具体的 Unicode 编号 (二进制) | UTF-16 编码 | 编码后的 字节数 |
|------------------------|--------------------------|-------------------------------------|-------------|
| 0000 0000 ~ 0000 FFFF | xxxxxxxx xxxxxxxx | xxxxxxxx xxxxxxxx | 2 |
| 0001 0000---0010 FFFF | yyyy yyyy yyxx xxxx xxxx | 110110yy yyyyyyyy 110111xx xxxxxxxx | 4 |

位于 D800~0xDFFF 之间的 Unicode 编码是特别为四字节的 UTF-16 编码预留的，所以不应该在这个范围内指定任何字符。如果你真的去查看 Unicode 字符集，会发现这个区间内确实没有收录任何字符。

UTF-16 要求在制定 Unicode 字符集时必须考虑到编码问题，所以真正的 Unicode 字符集也不是随意编排字符的。

UTF-16是如何使用代理区编码的？

UTF-16 是一种变长的 2 或 4 字节编码模式。对于 BMP 内的字符使用 2 字节编码，其它的则使用 4 字节组成所谓的代理对来编码。

在前面的鸟瞰图中，我们看到了一片空白的区域，这就是所谓的代理区（**Surrogate Area**）了，代理区是 UTF-16 为了编码增补平面中的字符而保留的，总共有 2048 个位置，均分为高代理区（D800–DBFF）和低代理区（DC00–DFFF）两部分，各1024，这两个区组成一个二维的表格，共有 $1024 \times 1024 = 2^{10} \times 2^{10} = 2^4 \times 2^{16} = 16 \times 65536$ ，所以它恰好可以表示增补的 16 个平面中的所有字符。

| UTF-16 decoder | | | | |
|----------------|--------|--------|-----|--------|
| Lead \ Trail | DC00 | DC01 | ... | DFFF |
| D800 | 010000 | 010001 | ... | 0103FF |
| D801 | 010400 | 010401 | ... | 0107FF |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| DBFF | 10FC00 | 10FC01 | ... | 10FFFF |

什么是代理对？

一个高代理区（即上图中的Lead（头），行）的加一个低代理区（即上图中的Trail（尾），列）的编码组成一对即是一个代理对（**Surrogate Pair**），必须是这种先高后低的顺序，如果出现两个高，两个低，或者先低后高，都是非法的。

在图中可以看到一些转换的例子，如

(D8 00 DC 00) → U+10000，左上角，第一个增补字符

(DB FF DF FF) → U+10FFFF，右下角，最后一个增补字符

UTF-16为何要使用代理对？

UTF-16相当于牺牲了高代理区（D800–DBFF）和低代理区（DC00–DFFF）两部分空间，但是确新增了 $1024 \times 1024 = 16 \times 65536$ 的空间。依次来实现了扩容！

4.总结

只有 UTF-8 兼容 ASCII，UTF-32 和 UTF-16 都不兼容 ASCII，因为它们没有单字节编码。

如果你希望查看完整的 Unicode 字符集，以及各种编码方式，请猛击：<https://unicode-table.com/cn/> 以 GB2312 为例，该字符集收录的字符较少，所以使用 1~2 个字节编码。

- 对于 ASCII 字符，使用一个字节存储，并且该字的最高位是 0；
- 对于中国的字符，使用两个字节存储，并且规定每个字的最高位都是 1。

由于单字节和双字的最高位不一样，所以很容易区分一个字符到底用了几个字节。

5.UTF16转UTF8

首先需要知道 Unicode 编码范围 [U+00, U+10FFFF], 其中 [U+00, U+FFFF] 称为基础平面(BMP), 这其中的字符最为常用.

当然, 这 65536 个字符是远远不够的.

0x010000 - 0x10FFFF 为辅助平面, 共可存放 $16 * 65536$ 个字符, 划分为 16 个不同的平面。

Unicode和UTF8转换关系表如下:

| Unicode编码 | UTF-8字节流 |
|----------------------------|---|
| U+00000000 - U+0000007F | 0xxxxxxx |
| U+00000080 - U+000007FF | 110xxxxx 10xxxxxx |
| U+00000800 - U+0000FFFF | 1110xxxx 10xxxxxx 10xxxxxx |
| U+00010000 - U+001FFFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx |
| U+00200000 - U+03FFFFFF | 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx |
| U+04000000 - U+7FFFFFFF | 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx |

所以UTF16和UTF8之间的相互转换可以通过上表的转换表来实现, 判断Unicode码所在的区间就可以得到这个字符是由几个字节所组成, 之后通过移位来实现, 分为新的多个字节来存储。

步骤描述

- Step1: 获取该字符对应的Unicode码
- Step2: 判断该Unicode码所在的范围, 根据不同的范围, 来决定存储它的字节长度。
 - 如果介于U+00000000 - U+0000007F之间, 代表该字符采取一个字节存储, 那么直接通过这个新字节的unicode码, 即可转换为UTF-8码(这是这里的一种简称, 不同的编程语言有不同实现, 例如可以用两个字节来存储一个字符的信息, 解码时进行判断, 如果发现是UTF-8的多字节实现, 那么将多字节合并后再转为一个字符输出). 转换完毕
 - 如果介于U+00000080 - U+000007FF之间, 代表该字符采取两个字节存储, 那么将该Unicode码转为二进制, 取出高5位(这里不分大端序和小端序, 只以实际的码为准, 具体实现可以采取移位实现), 并加上头部110, 组成第一个字节; 再取出低6位(按顺序取), 加上头部10, 组成第二个字节。然后分别通过两个新的字节的unicode码, 可以转换为相应的UTF-8码. 转换完毕
 - 如果介于U+00000800 - U+0000FFFF之间, 代表该字符采取三个字节存储, 那么将该Unicode码转为二进制, 取出高4位, 并加上头部1110, 组成第一个字节; 再取出低6位(按顺序取), 加上头部10, 组成第二个字节; 再取出低6位(按

顺序取),加上头部10,组成第三个字节。然后分别通过三个新的字节的unicode码,可以转换为相应的UTF-8码.转换完毕

- 如果介于U+00010000 – U+001FFFFF之间,代表该字符采取四个字节存储(实际上,四个字节或以上存储的字符是很少的),那么将该Unicode码转为二进制,取出高3位,并加上头部11110,组成第一个字节;再取出低6位(按顺序取),加上头部10,组成第二个字节;再取出低6位(按顺序取),加上头部10,组成第三个字节;再取出低6位(按顺序取),加上头部10,组成第四个字节。然后分别通过四个新的字节的unicode码,可以转换为相应的UTF-8码.转换完毕
- 如果介于U+00200000 – U+03FFFFFF,代表该字符采取五个字节存储,那么将该Unicode码转为二进制,取出高2位,并加上头部11110,组成第一个字节;再取出低6位(按顺序取),加上头部10,组成第二个字节;再取出低6位(按顺序取),加上头部10,组成第三个字节;再取出低6位(按顺序取),加上头部10,组成第四个字节;再取出低6位(按顺序取),加上头部10,组成第五个字节。然后分别通过五个新的字节的unicode码,可以转换为相应的UTF-8码.转换完毕
- 如果介于U+04000000 – U+7FFFFFFF,代表该字符采取六个字节存储,那么将该Unicode码转为二进制,取出高1位,并加上头部11110,组成第一个字节;再取出低6位(按顺序取),加上头部10,组成第二个字节;再取出低6位(按顺序取),加上头部10,组成第三个字节;再取出低6位(按顺序取),加上头部10,组成第四个字节;再取出低6位(按顺序取),加上头部10,组成第五个字节;再取出低6位(按顺序取),加上头部10,组成第六个字节。然后分别通过六个新的字节的unicode码,可以转换为相应的UTF-8码.转换完毕

6.UTF16转GBK

7.补充知识

(1) String.length是如何计算的

```
1 public class testT {
2     public static void main(String [] args){
3         String A = "hi你是乔戈里";
4         System.out.println(A.length());
5     }
6 }
```

上结果输出为7。

```
1 public class testStringLength {
2     public static void main(String [] args){
3         String B = "🎵"; // 这个就是那个音符字符，只不过由于当前的网页没支持这种编码，所以没显示。
4         System.out.println(B.length());
5     }
6 }
```

得到的结果为2。为何不是1呢？


```

1  /**
2      * Returns the length of this string.
3      * The length is equal to the number of <a
href="Character.html#unicode">Unicode
4      * code units</a> in the string.
5      *
6      * @return the length of the sequence of characters
represented by this
7      *         object.
8      */
9  public int length() {
10     return value.length;
11 }

```

得到了大体意思：返回字符串的长度，这一长度等于字符串中的 Unicode 代码单元的数目。

Java中 有内码和外码这一区分简单来说

- 内码：char或String在内存里使用的编码方式。
- 外码：除了内码都可以认为是“外码”。（包括class文件的编码）

而java内码：unicode (utf-16) 中使用的是utf-16. 所以上面的那句话再进一步解释就是：返回字符串的长度，这一长度等于字符串中的UTF-16的代码单元的数目。

代码单元指一种转换格式（UTF）中最小的一个分隔，称为一个代码单元（Code Unit），因此，一种转换格式只会包含整数个单元[2]。UTF-X 中的数字 X 就是各自代码单元的位数。

UTF-16 的 16 指的就是最小为 16 位一个单元，也即两字节为一个单元，UTF-16 可以包含一个单元和两个单元，对应即是两个字节和四个字节。我们操作 UTF-16 时就是以它的一个单元为基本单位的。

而上面我的例子中的那个字符的Unicode值就是“U+1D11E”，这个Unicode的值明显大于U+FFFF，所以对于这个字符UTF-16需要使用四个字节进行编码，也就是使用两个代码单元！

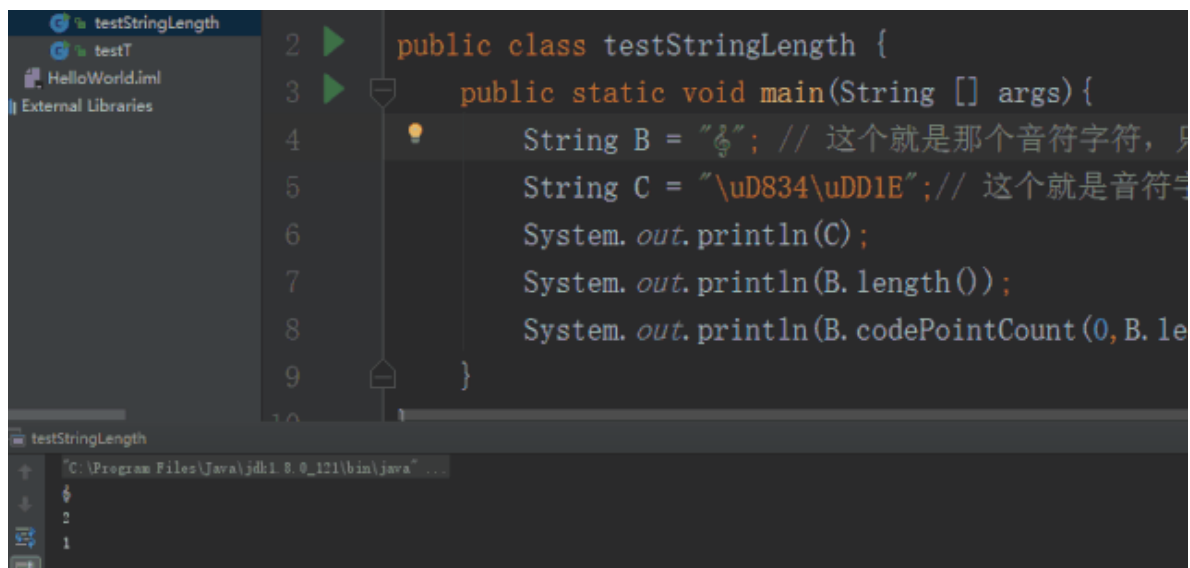
所以你也看到我的上面那个示例结果表示一个字符的String.length()长度是2！

那如何统计字符串有几个字符该如何统计呢？

```

1  public class testStringLength {
2      public static void main(String [] args){
3          String B = "🎸"; // 这个就是那个音符字符，只不过由于当前的网页没
支持这种编码，所以没显示。
4          String C = "\uD834\uDD1E"; // 这个就是音符字符的UTF-16编码
5          System.out.println(C);
6          System.out.println(B.length());
7          System.out.println(B.codePointCount(0,B.length()));
8          // 想获取这个Java文件自己进行演示的，可以在我的公众号【程序员乔戈
里】后台回复 6666 获取
9      }
10 }

```



```
public class testStringLength {
    public static void main(String [] args) {
        String B = "🎵"; // 这个就是那个音符字符，只
        String C = "\uD834\uDD1E"; // 这个就是音符字
        System.out.println(C);
        System.out.println(B.length());
        System.out.println(B.codePointCount(0, B.le
    }
}
```

可以看到通过codePointCount()函数得知这个音乐字符是一个字符！

几个问题：

- 1.codePointCount是什么意思呢？
- 2.之前不是说音符字符是“U+1D11E”，为什么UTF-16是"\uD834\uDD1E"，这俩之间如何转换？
- 3.前面说了UTF-16的代码单元，UTF-32和UTF-8的代码单元是多少呢？

第一个问题：

codePointCount其实就是代码点数的意思，也就是一个字符就对应一个代码点数。

比如刚才音符字符（没办法打出来），它的代码点是U+1D11E，但它的代理单元是U+D834和U+DD1E，如果令字符串str = "\u1D11E"，机器识别的不是音符字符，而是一个代码点"/u1D11"和字符"E"，所以会得到它的代码点数是2，代码单元数也是2。

但如果令字符串str = "\uD834\uDD1E"，那么机器会识别它是2个代码单元代理，但是是1个代码点（那个音符字符），故而，length的结果是代码单元数量2，而codePointCount()的结果是代码点数量1。

第二个问题：

码点到 UTF-16 如何转换？

乔哥：继续上个例子。转换分成两部分：

1. BMP 中直接对应，无须做任何转换，也就是如果 $U < 0x10000$ ，U的UTF-16编码就是U对应的16位无符号整数；
2. 增补平面 SP 中，则需要做相应的计算。也就是如果 $U \geq 0x10000$ 的情况

我们先计算 $U' = U - 0x10000$ ，然后将U'写成二进制形式： $yyyy\ yyyy\ yyxx\ xxxx\ xxxx$ ，U的UTF-16编码（二进制）就是： $110110yyyyyyyyyy\ 110111xxxxxxxxxx$ 。

Unicode编码0x20C30，减去0x10000后，得到0x10C30，写成二进制是： $0001\ 0000\ 1100\ 0011\ 0000$ 。用前10位依次替代模板中的y，用后10位依次替代模板中的x，就得到： $1101100001000011\ 1101110000110000$ ，转换为16进制即0xD843 0xDC30。

上图是对应的转换规则：

- 首先 $U+1D11E-U+10000 = U+0D11E$
- 接着将 $U+0D11E$ 转换为二进制：0000 1101 0001 0001 1110，前10位是0000 1101 00 后10位是01 0001 1110
- 接着套用模板：110110yyyyyyyyyyy 110111xxxxxxxxxxx
- $U+0D11E$ 的二进制依次从左到右填入进模板：110110 **0000 1101 00** 110111 **01 0001 1110**
- 然后将得到的二进制转换为16进制：d834dd1e，也就是你看到的utf-16编码了

第三个问题：

同理，UTF-32 以 32 位一个单元，它只包含这一种单元就够了，它的一单元自然也就是四字节了。

UTF-8 的 8 指的就是最小为 8 位一个单元，也即一字节为一个单元，UTF-8 可以包含一个单元，二个单元，三个单元及四个单元，对应即是一，二，三及四字节。