

一文了解诸多的编码方式以及Java的编码解码

文章分类: JavaSE; 作者: Hackyle;

更新时间: Tue Jan 31 09:20:35 CST 2023

本文的主要内容

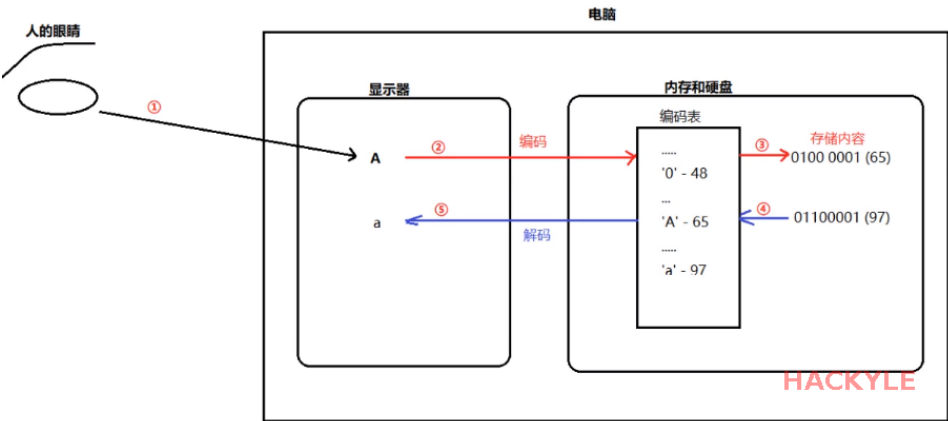
- 介绍常见编码格式 (ASCII、GBK、UTF-8等) 的基本原理
- Java中是如何处理多种编码格式的

注意: 本文对编码格式的阐述来源于互联网 (已忘记来源自哪里), 我做了一些排版调整和额外的补充说明。 仅仅作作为学习材料使用。

内容导览

- [编码格式](#)
 - [ASCII](#)
 - [ISO-8859-1](#)
 - [windows-1252](#)
 - [GB2312](#)
 - [GBK](#)
 - [GB18030](#)
 - [BIG5](#)
 - [Unicode](#)
 - [UTF-32](#)
 - [UTF-16](#)
 - [UTF-8](#)
 - [BOM](#)
- [编码解码](#)
- [JVM-char](#)
- [String](#)

编码格式



计算机如何存储与表示字符

把字符 'A' 通过编码表, 转换成二进制的这个过程就是编码

ASCII

背景:

- 很久以前, 有一群美国人, 他们决定用8个可以开合的晶体管来组合成不同的状态, 以表示世界上的万物。他们看到8个开关状态是好的, 于是他们把这称为"字节 (Byte)"。
- 再后来, 他们又做了一些可以处理这些字节的机器, 机器开动了, 可以用字节来组合出很多状态, 于是它们就这机器称为"计算机"。

开始计算机只在美国用。八位的字节一共可以组合出256(2的8次方)种不同的状态。给前128个编号 (0-127) 都赋予特殊的意义，于是ASCII码诞生了。

控制码：

- 他们把其中的编号从0开始到编号31，共计32种状态分别规定了特殊的用途，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。
- 二进制范围：0000 0000 - 0001 1111，对应的十六进制表示为：0x00 - 0x1F
- 例如：遇上00x10，终端就换行，遇上0x07，终端就向人们嘟嘟叫，例好遇上0x1b，打印机就打印反白的字，或者终端就用彩色显示字母。

字符码：

- 他们又把所有的空格、标点符号、数字、大小写字母分别用连续的字节状态表示，一直从第32号编到了第127号，这样计算机就可以用不同字节来存储英语的文字了。
- 大家看到这样，都感觉很好，于是大家都把这个方案叫做 ANSI 的"ASCII"编码 (**American Standard Code for Information Interchange, 美国信息互换标准代码**)，编码范围0~127。
- 当时世界上所有的计算机都用同样的ASCII方案来保存英文文字。

```

1 public class EncodeDecodeDemo {
2     public static void main(String[] args) {
3         //byte类型的取值范围也是[-128, 127]
4         //byte by = 0;
5
6         //ASCII码的范围时: [0,127]
7         for (byte i = 0; i < 127; i++) {
8             //注意: 前32个为不可打印字符
9             System.out.println("编号: "+i + "所代表的值: " +(char)i);
10        }
11    }
12 }
```

总结：

1. 一个英文字符一个字节 (Byte)，共有256个编号
2. ASCII只用了其中的一半，即编号0 - 127
3. 编号0 - 31的为控制编号，编号为31以上的为打印字符
4. 几个常见字母的ASCII码大小：“A”为65；“a”为97；“0”为48

ISO-8859-1

又称位Latin-1，它也是使用一个字节来表示一个字符，因为西欧文字也是字母拼接，只不过不是26个英文字母罢了。

0~127与ASCII码一样，128-255规定了不同的含义：

- 128-159：表示一些控制字符，这些字符也不常用；
- 160-255：表示了西欧字符；

```

1 import java.nio.charset.StandardCharsets;
2 import java.util.Arrays;
3
4 public class EncodeDecodeDemo {
5     public static void main(String[] args) {
6         byte[] bytes = "Ä".getBytes(StandardCharsets.ISO_8859_1);
7         System.out.println(Arrays.toString(bytes)); //打印出字节数组的内容: [-60
8         //System.out.println(new String(bytes, StandardCharsets.ISO_8859_1));
9
10        //获取该个西欧字符在ISO_8859_1编码表上的编号: 方案一
11        int num = bytes[0] & 0xFF; //将该个西欧字符的编号转换成int型的值
12        System.out.println(num);
13
14        //获取该个西欧字符在ISO_8859_1编码表上的编号: 方案二
15        //byte数据类型的取值范围: [-128, 127]
16        //-60意味着超过了byte的最大值127, 又从-128开始增加
```

TOP

```

17 //取值到byte的最大值127，意味着它的编号为128
18 //最终编号为：128 + |(-128+60)| = 196
19
20 //查询ISO_8859_1编码表，编号为196的字符，就是：Ä
21 }
22 }

```

ISO-8859-1																	
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF	
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI	
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US	
2x	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/	
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_	
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	
7x	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL	
8x	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3	
9x	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC	
Ax	NBSP	i	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	®	¯	°	
Bx	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿		
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	HACKYLE				ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	

总结：

1. 一个西欧字符使用一个字节 (Byte)
2. 使用ASCII没有使用的编号为128-255，来表示西欧字符

windows-1252

ISO-8859-1虽然号称是标准用于西欧国家，是他连欧元 (€) 这个符号都没有，因为欧元比较晚，而标准比较早。

实际使用中更为广泛的其实是windows-1252编码。这个编码与ISO8859-1基本上是一样的，区别在于数字128到159用于表示西欧的特殊字符

在很多应用程序中，即使文件声明它采用的是ISO8859-1编码，其实在解析的时候依然被当做windows-1252编码。

GB2312

等中国人们得到计算机时，已经没有可以利用的字节状态来表示汉字，况且有6000多个常用汉字需要保存呢。但是这难不倒智慧的中国人民，我们不客气地把那些127号之后的奇异符号们直接取消掉，**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。这种汉字方案叫做"GB2312"。**

编码规则：

- 前面的一个字节（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多个简体汉字了。
- 高字节的范围是1010_0001(十进制161)~1111_0111(十进制247)，低字节的范围是1010_0001(十进制161)~1111_1110 (十进制254)，共同可组合出以下中可能：

$$(247-161) * (254-161) = 7998$$

特殊字符：

- 在这些编码里，我们还数学符号、罗马希腊的字母、日文的假名们都编进去了；
- 连在 ASCII 里本来就有的数字、标点、字母都统统重新编了两个字节长的编码，这就是常说的**“全角”**字符，而原来在127号以下的那些就叫**“半角”**字符了。

如何识别：

- GB2312是对ASCII的中文扩展，也就是在兼容ASCII的同时，还包含7000多个汉字。
- **检测高字节位和低字节位，如果同时大于0x7F，则认为是GB2312，否则认为是ASCII编码。**
- 因为GB2312用的是大于127的编号，127的十六进制编码为0x7F，所以只要两个字节同时满足大于0x7F，就是GB2312。

```

1 public class EncodeDecodeDemo {
2     public static void main(String[] args) throws Exception {
3         //一个byte，8位，在Java中的取值范围：[-128, 127]
4         byte[] bytes = "中".getBytes("gb2312");
5
6         for (byte bb : bytes) {
7             //由于byte的取值范围最大只有127，而GB2312用的都是在127之后的编号
8             //所以需要将其转换成int型，更方便查看
9             int num = bb & 0xFF;
10            System.out.println(num);
11            System.out.println(Integer.toBinaryString(num));
12            System.out.println(Integer.toHexString(num));
13        }
14        //这个汉字"中"的编码如下
15        //二进制：1101,0110 1101,0000
16        //十六进制：0xD6 0xD0
17
18        byte[] bb = "A".getBytes("gb2312");
19        int tmp = bb[0] & 0xFF;
20        System.out.println(tmp); //输出：65，这就是ASCII吗对英文字母A的编码
21        System.out.println(Integer.toBinaryString(tmp)); //输出：1000001
22        System.out.println(Integer.toHexString(tmp)); //输出：0x41
23    }
24 }

```

总结

- 使用2个字节表示汉字
- 如果是英文字母，还是使用ASCII编码（1个字节）

GBK

编码规则：

- 后来还是不够用，于是干脆**不再要求低字节一定是127号之后的编码，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**，不管后面跟的是不是扩展字符集里的内容。
- 结果扩展之后的编码方案被称为 GBK 标准，GBK 包括了 GB2312 的所有内容，同时又增加了近20000个新的汉字（包括繁体字）和符号。

如何编码：

- GBK同样使用两个Bytes表示
- 其中**第一个字节**范围是1000_0001(十进制129)~1111_1110(十进制254)
- **第二个字节**范围是0111_0000(十进制64)~0111_1110(126)和1000_0000(十进制128)~1111_1110(十进制254)。
- 共同组合可以表示的汉字是：(254-64) * (126-64 + 254-128)

如何解析：

- 需要注意的是，第二个字节从64开始（64输出byte的正数范围之内，和ASCII码重合了），也就是说，第二个字节的最高位可能为0。
- 那如何知道它是汉字的一部分还是一个ASCII字符呢？
- 其中也很简单，因为汉字是用固定的连个字节表示的，在**解析二进制流的时候，如果第一个字节的最高位为1，那么就将下一个字节读进来解析为一个汉字**，而不用考虑它的最高位，解析完后，跳到第三个字节继续解析。

```

1 public class EncodeDecodeDemo {
2     public static void main(String[] args) throws Exception {
3         //一个byte，8位，在Java中的取值范围：[-128, 127]
4

```

TOP

```

5      byte[] bytes = "中".getBytes("GBK");
6
7      for (byte bb : bytes) {
8          //由于byte的取值范围最大只有127，而GBK用的都是在127之后的编号
9          //所以需要将其转换成int型，更方便查看
10         int num = bb & 0xFF;
11         System.out.println(num);
12         System.out.println(Integer.toBinaryString(num));
13         System.out.println(Integer.toHexString(num));
14     }
15     //这个汉字"中"的编码如下
16     //二进制: 1101,0110 1101,0000
17     //十六进制: 0xD6 0xD0
18     //因为GBK兼容GB2312，所以"中"这个汉字的两种编码结果一样
19
20     byte[] bb = "A".getBytes("GBK");
21     int tmp = bb[0] & 0xFF;
22     System.out.println(tmp); //输出: 65，这就是ASCII吗对英文字母A的编码
23     System.out.println(Integer.toBinaryString(tmp)); //输出: 1000001
24     System.out.println(Integer.toHexString(tmp)); //输出: 0x41
25
26     //GBK是支持繁体的，GB2312不支持繁体
27     byte[] bytes1 = "開".getBytes("gb2312");
28     byte[] bytes2 = "開".getBytes("GBK");
29     System.out.println(new String(bytes1, "gb2312")); //输出一个问号: ?，说明解码失败
30     System.out.println(new String(bytes2, "GBK")); //输出繁体: 開
31 }

```

总结

- GBK在GB2312的基础上，再增加了可编码的汉字，也就是说，GBK是兼容GB2312
- 使用2个字节，原来的ASCII字符仍然使用1个字节

GB18030

后来少数民族也要用电脑了，于是我们再扩展，又加了几千个新的少数民族的字，GBK 扩成了 GB18030。叫做 **"DBCS"** (Double Byte Charecter Set 双字节字符集)。

特性:

- 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
- 如果某个字节的值是大于127的，那么就认为一个双字节字符集里的字符（如中文）出现了。
- 反之，如果某个字节的值是小于127的，则认为它是一个单字节长的英文字符。

编码规则:

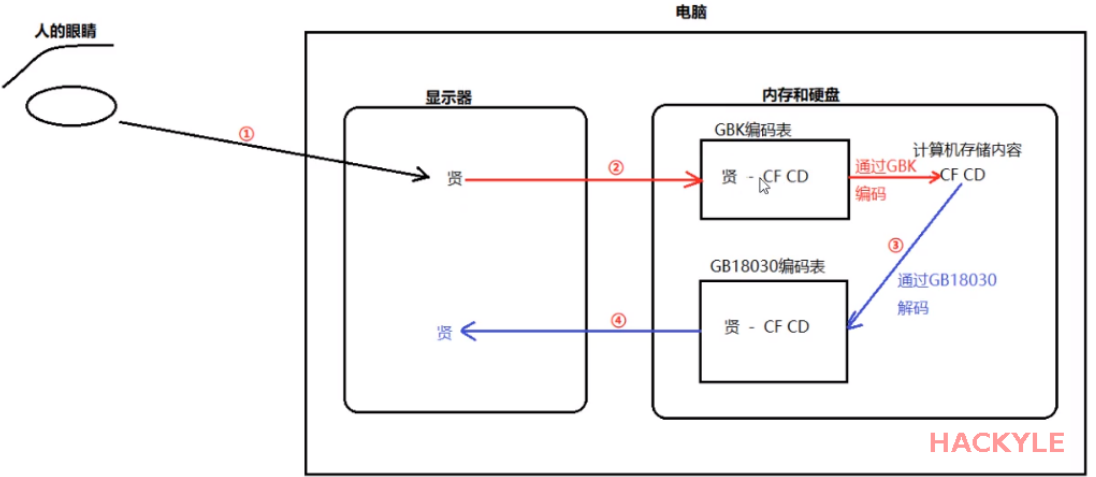
- BG18030使用变长编码，有的字符是两个字节，有的是四个字节。
- 在**两个字节**的编码中，字节表示范围与**GBK**一样。
- 在**四字字节**编码中，第一个字节的值从1000_0001(十进制129)~1111_1110(十进制254)，第二个字节的值从0011_0000(十进制48)到0011_1001(十进制57)，第三个字节的值从1000_0001 (十进制129)~1111_1110(十进制254)，第四个字节的值从0011_0000(十进制48)~0011_1001(十进制57);

解析:

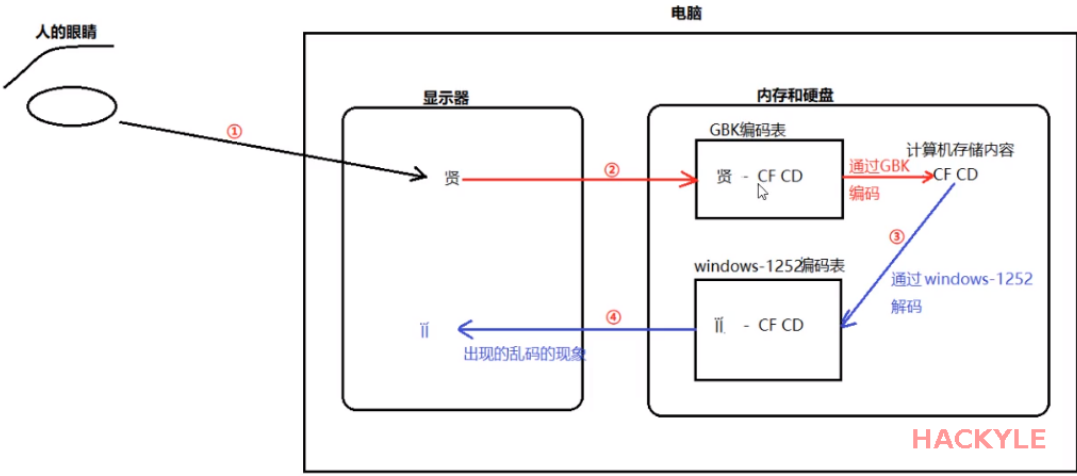
- 解析二进制时，如何知道是两个字节还是四个字节呢？
- 很简单，看**第二个字节的范围**，如果是**48到57就是四个字节**，因为两个字节编码中第二字节都比这个大。

GB18030兼容GBK、GB2312、ASCII，但是GB18030、GBK、GB2312与ISO8859-1是不兼容的。

TOP



兼容的编码和解码



不兼容的编码和解码

1. 编码格式
1. ASCII
2. ISO-8859-1
3. windows-1252
4. GB2312
5. GBK
6. GB18030
7. BIG5
8. Unicode
9. UTF-32
10. UTF-16
11. UTF-8
12. BOM
2. 编码解码
3. JVM-char
4. String

```
1 public class EncodeDecodeDemo {
2     public static void main(String[] args) throws Exception {
3         //一个byte，8位，在Java中的取值范围：[-128, 127]
4         byte[] bytes = "中".getBytes("GB18030");
5
6         for (byte bb : bytes) {
7             //由于byte的取值范围最大只有127，而GB18030用的都是在127之后的编号
8             //所以需要将其转换成int型，更方便查看
9             int num = bb & 0xFF;
10            System.out.println(num);
11            System.out.println(Integer.toBinaryString(num));
12            System.out.println(Integer.toHexString(num));
13        }
14        //这个汉字"中"的编码如下
15        //二进制：1101,0110 1101,0000
16        //十六进制：0xD6 0xD0
17        //因为GB18030兼容GBK、GB2312，所以"中"这个汉字的三种编码结果一样
18
19        byte[] bb = "A".getBytes("GB18030"); //GB18030对ASCII字符还是以1字节编码
20        int tmp = bb[0] & 0xFF;
21        System.out.println(tmp); //输出：65，这就是ASCII吗对英文字母A的编码
22        System.out.println(Integer.toBinaryString(tmp)); //输出：1000001
23        System.out.println(Integer.toHexString(tmp)); //输出：0x41
24
25        //GB18030、GBK都支持繁体的，所以都可以解析繁体字
```

Java Copy

TOP

```
26 |         byte[] bytes1 = "開".getBytes("GB18030");
27 |         byte[] bytes2 = "開".getBytes("GBK");
28 |         System.out.println(new String(bytes1, "GB18030")); //输出繁体：開
29 |         System.out.println(new String(bytes2, "GBK")); //输出繁体：開
30 |     }
31 | }
```

总结

- 英文等原来是ASCII字符的，还是使用1字节编码
- 中文使用2字节或者4字节编码
- 支持繁体

BIG5

主要使用地区是台湾、香港使用中文繁体的地方。

编码规则：

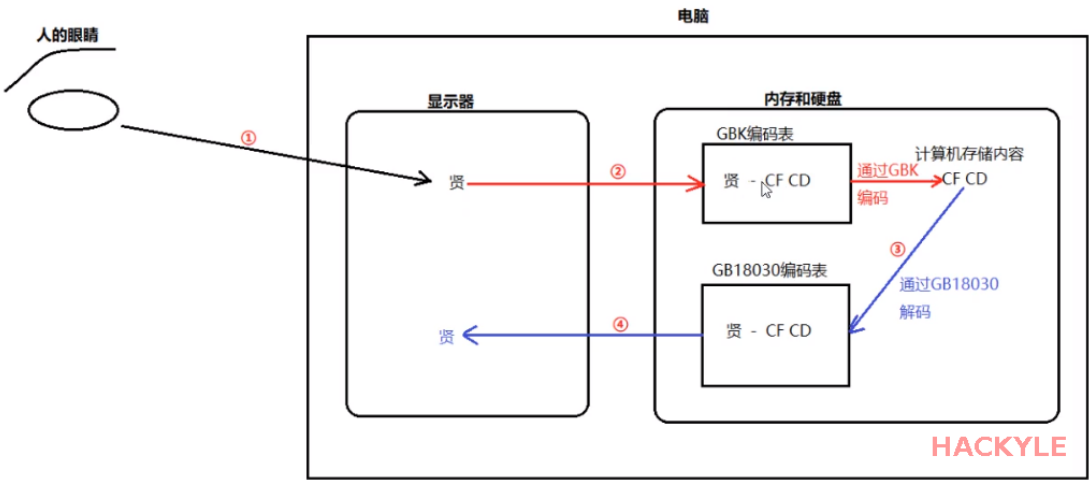
- 使用两个字节表示。
- 第一个字节的范围是1000_0001(十进制129)~1111_1110(十进制254);
- 第二个字节的范围是0100_0000(十进制64)~0111_1110(十进制126)和1010_0001(十进制161)~1111_1110(十进制254)。

由于编码规则不一致，中国大陆的GB18030、GBK、GB2312与港台的BIG5都不兼容！

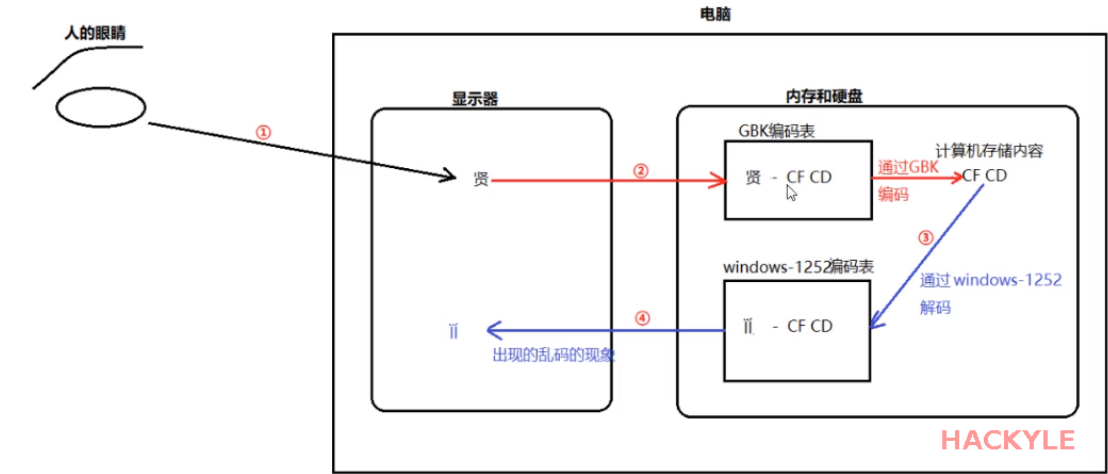
也就是说，如果港台人用他们的编码方式编辑的一个文件，在大陆境内打开是乱码！

Unicode

像中国大陆使用的编码：GBK、GB2312、ASCII、GB18030，与欧洲的ISO8859-1、港台地区的BIG5都是不兼容的。如果互相发送文件，那么打开来的文件都是乱码，如下图。



兼容的编码和解码



不兼容的编码和解码

为了解决不同编码不相容的问题。ISO（国际标准化组织）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、所有字母和符号的编码！他们打算叫它"Universal Multiple-Octet Coded Character Set", 简称 UCS, 俗称 **"UNICODE"**。

编码思想：

- ISO 就直接规定Unicode**必须用两个字节**，也就是16位（两个字节）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。
- 由于"半角"英文符号只需要用到低8位，所以其高 8位永远是0，因此这种大气的方案在保存英文文本时会多浪费一倍的空间。

编码规则

- 用于字符编号的范围是：0X000000 ~ 0X10FFFF，包括110多万。
- 但大部分常用字符都早0X0000~0XFFFF之间，即65535个数字之内。每个字符都有一个Unicode编号，这个编码一般写成16进制。
- 大部分中文的编号范围在U+4E00~U+ 9FA5。

特性：

- 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"，请注意"字符"和"字节"两个术语的不同，“字节”是一个8位的物理存储单元，而“字符”则是一个文化相关的符号。
- 在UNICODE 中，一个字符就是两个字节。
- 但是，UNICODE 在制订时没有考虑与任何一种现有的编码方案保持兼容，这使得 GBK 与UNICODE 在汉字的内码编排上完全是不一样的，没有一种简单的算术方法可以把文本内容从UNICODE编码和另一种编码进行转换，这种转换必须通过**查表**来进行。

注意，Unicode只做了这样一件事：将所有字符分配了唯一的数字编码，它并没有规定这个编码怎么对应到二进制表示，所以，实现这种具体的字符到二进制的表示，有UTF-8、UTF-16、UTF-32等方案。

UTF-32

编码思想：统一使用四个字节表示所有字符（含ASCII）

编码规则：

- 若第一个字节是整数二进制中的最高位，最后一个字节是整数二进制中的最低位。这种字节序列被叫做“大端（Big Endian，BE）”。反之，则为小端（Little Endian，LE）；
- 大端对应的编码方式为UTF-32BE，小端对应的编码方式为UTF-32LE；

Unicode编码	UTF32-LE	UTF-BE
0x006C49	49 6C 00 00	00 00 6C 49
0x020C30	30 0C 02 00	00 02 0C 30

之所有大小端，是因为硬件的读取顺序不同

- 大端：数据的高字节保存在内存的低地址中，低字节保存到内存的高地址中，和我们的阅读习惯一致；
- 小端：反之，常用的X86结构采用小段模型。
- 大端方式符合人类的正常思维，小端方式利于计算机处理。

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，称为一个平面 (plane)。
- 目前一共有17个 (2^5) 平面 (65536*17=1,114,112也就是110多万)，也就是说，整个Unicode字符集的数量大小现在是2^21。
- 最前面的65536个字符位，称为基本平面 (简称BMP)，它的码点范围是从0到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+10FFFE

UTF-16的编码规则很简单

- 基本平面的字符占用2个字节，辅助平面的字符占用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，也就是)，要么是4个字节 (U+010000到U+10FFFF)。

那么问题来了，当我们遇到两个字节时，到底是把这两个字节当作一个字符还是与后面的两个字节起当作一个字符呢？

- 为了将两个字节的UTF-16编码与四个字节的UTF-16编码区分开来，Unicode编码的设计者将0xD800-0xDFFF保留下来，并称为代理区 (Surrogate)：
- 辅助平面的字符位共有2^20个，因此表示这些字符至少需要20个二进制位。UTF-16将这20个二进制位分成两半，前10位映射在U+D800到U+DBFF，称为高代理位 (H)，后10位映射在U+DC00到U+DFFF，称为低代理位 (L)。这意味着，一个辅助平面的字符，被拆成两个基本平面的字符表示
- 如果U≥0x10000，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：1101 10yy yyyy yyyy 1101 11xx xxxx xxxx

按照上述规则，Unicode编码0x10000-0x10FFFF的UTF-16编码有四个字节，前两个字节的高6位是110110，后两个字节的高6位是110111。

前两个字节的取值范围 (二进制) 是1101100000000000到1101101111111111，即0xD800-0xDBFF。

后两个字节取值范围 (二进制) 是1101110000000000到1101111111111111，即0xDC00-0xDFFF

因此，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解读

接下来，以汉字“吉”为例，说明UTF-16编码方式是如何工作的

- 汉字“吉”的Unicode码点为0x20BB7，该码点显然超出了基本平面的范围 (0x0000-0xFFFF)，因此需要使用四个字节表示。
- 首先用0x20BB7 - 0x10000计算出超出的部分，然后将其用20个二进制位表示 (不足前面补0)，结果为0001 0000 1011 1011 0111。
- 接着，将前10位映射到U+D800到U+DBFF之间，后10位映射到U+DC00到U+DFFF即可。U+D800对应的二进制数为1101 1000 0000 0000，直接填充后面的10个二进制位即可，得到1101 1000 0100 0010，转成16进制数则为0xD842。同理可得，低位为0xDFB7。
- 因此得出汉字“吉”的UTF-16编码为0xD842 0xDFB7

Unicode编码	UTF-16LE	UTF-16BE	UTF32-LE	UTF32-BE
0x006C49	49 6C	6C 49	49 6C 00 00	00 00 6C 49
0x020C30	30 DC 43 D8	D8 43 DC 30	30 0C 02 00	00 02 0C 30

- 大端：数据的高字节保存在内存的低地址中，低字节保存到内存的高地址中，和我们的阅读习惯一致；
- 小端：反之，常用的X86结构采用小段模型。
- 大端方式符合人类的正常思维，小端方式利于计算机处理。

UTF-8

编码思想：

- 国际标准组织(ISO)使用变长字节表示世界上的所有字符，每个字符使用的字节个数与其Unicode编号的大小有关。
- 编号小的使用的字节少，编号大的使用的字节就多，使用的字节个数从1到4个不等。

编码规则：

具体来说，各个Unicode编号范围对应的二进制格式如下表所示

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	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

HACKYLE

图中的x表示可以用的二进制位，而每个字节开头的1或0是固定的。

如何解码：

- 根据编码规则，如果一个字节开头的是0，则按照一个字节来解析；
- 如果一个字节开头的是110，则按照两个字节来解析，即该字节和其后的一个字节解析为一个字符；
- 如果一个字节开头的是1110，则该字节和其后的两个字节解析为一个字符；
- 如果一个字节开头的是11110，则该字节和其后的三个字节解析为一个字符；

一般特性：

- 指定英文字符使用1个字节，沿用原来的ASCII码
- 使用1~4个字节表示一个非英文符号，中文存储使用3个字节
- ASCII码中的内容用1个字节保存；欧洲的字符用2个字节保存；东亚的字符用3个字节保存；特殊符号（例如表情包）用4个字节
- Unicode是内存编码表示方案（规范），而utf-8是如何保存和传输Unicode的方案（实现）

BOM

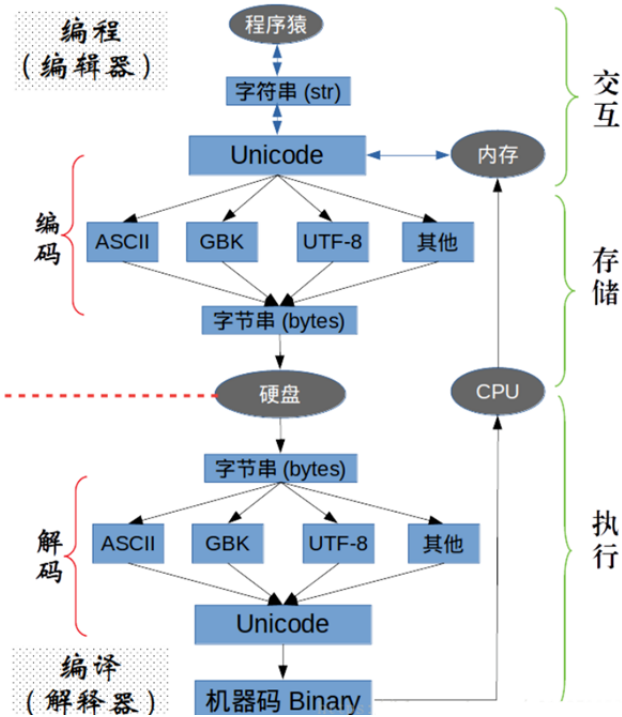
UTF编码	Byte Order Mark (BOM)
UTF-8 without BOM	无
UTF-8 with BOM	EF BB BF
UTF-16LE	FF FE
UTF-16BE	FE FF
UTF-32LE	FF FE 00 00
UTF-32BE	00 00 FE FF

HACKYLE

一个文件如果采用UTF-8 with BOM，即使这个文件没有任何数据内容，它内部也会有三个字节的数​​据，这个三个字节为：EFBBBF。

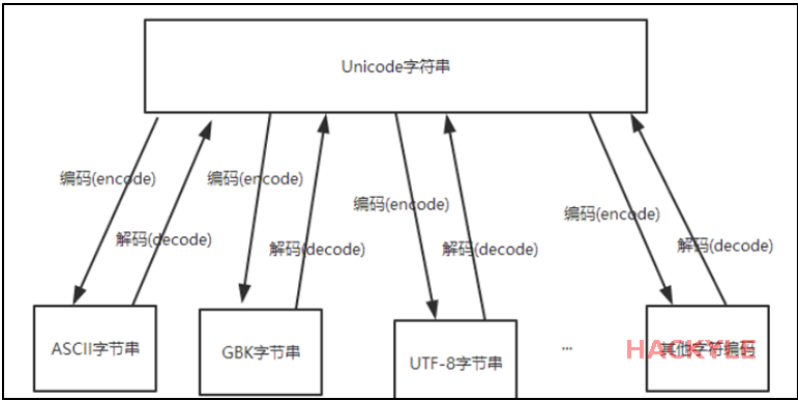
一个文件采用UTF-16LE编码，即时这个文件没有任何数据内容，它内部也会有两个字节的数​​据：FF FE

编码解码



编码：将字符转换为对应的二进制序列的过程叫做字符编码;

解码：将二进制序列转换为对应的字符的过程叫做字符解码.



Unicode是所有编码方式的转换的桥梁!

JVM-char

JVM使用Unicode对字符进行处理。

- char本质上是一个固定占用两个字节的无符号正整数，这个正整数对应与Unicode编码，用于标识那个Unicode编号对应的字符。
- 由于固定占用两个字节，char只能标识Unicode编号在65535以内的字符，不能超出。如果超出了，则需要使用string来表示。

char的三种赋值形式

```
1 public class MainTest {
2     public static void main(String[] args) {
3         //第一种：直接使用字符常量来赋值
4         char c1 = 'a';
5         char c2 = '中';
6         System.out.println(c1);
7         System.out.println(c2);
8
9         //第二种：直接使用Unicode编码来赋值
10        char c3 = 0X4e2d; //该字符的Unicode编码，十六进制形式
11        char c4 = 20013; //将该Unicode转换为十进制
12        System.out.println(c3);
13    }
14 }
```

TOP

```

13         System.out.println(c4);
14
15         //第三种：直接使用Unicode的码值形式来赋值
16         char c5 = '\u4e2d'; //其中\表示转义字符，u表示使用十六进制表示
17         System.out.println(c5);
18
19         //注意：char只能保存UTF-16中的基本码面（基本码面内的所有字符只占两个字节）
20         //如果是辅助码面（占四个字节）的话，则char保存不了，需要使用String
21     }
22 }

```

String

编码：字符串 --> 字节数组

- String类的getBytes() 方法进行编码，将字符串，转为对应的二进制，并且这个方法可以指定编码表。如果没有指定码表，该方法会使用操作系统默认码表。
- 注意：中国大陆的Windows系统上默认的编码一般为GBK。在Java程序中可以使用getProperty("file.encoding")方式得到当前的默认编码。

```

1 //编码实例
2 import java.io.UnsupportedEncodingException;
3 import java.util.Arrays;
4
5 public class MainTest {
6     public static void main(String[] args) throws UnsupportedEncodingException {
7         String value = System.getProperty("file.encoding"); //可以通过settings, Edit
8         System.out.println("系统默认的编码为 " + value);
9
10        String str = "中文";
11        byte[] bytes01 = str.getBytes(); //系统默认的编码方式，取决于上文中的"file.encoding"
12        System.out.println(Arrays.toString(bytes01));
13        //输出: [-28, -72, -83, -26, -106, -121], 表明一个中文字符，编码为3个字节
14
15        byte[] gbks = str.getBytes("GBK");
16        System.out.println(Arrays.toString(gbks));
17        //输出: [-42, -48, -50, -60], 表明一个中文字符，编码为2个字节
18
19        byte[] gb2312s = str.getBytes("gb2312");
20        System.out.println(Arrays.toString(gb2312s));
21        //输出: [-42, -48, -50, -60], 表明一个中文字符，编码为2个字节
22
23    }
24 }

```

解码的过程：字节数组 --> 字符串

- String类的构造函数完成。
- String(byte[] bytes) 使用系统默认码表
- String(byte[], charset) 指定码表
- 注意：我们使用什么字符集（码表）进行编码，就应该使用什么字符集进行解码，否则很有可能出现乱码（兼容字符集不会）。

```

1 //解码实例
2 import java.io.UnsupportedEncodingException;
3 import java.util.Arrays;
4
5
6 public class MainTest {
7     public static void main(String[] args) throws UnsupportedEncodingException {
8         //核心要义，用什么方式编码，就用什么方式解码，除非编码方式能兼容，如GBK兼容BG231

```

TOP

◀ ▶

版权声明： 非明确标注皆为原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上本文链接及此声明。
原文链接：<https://blog.hackyle.com/article/java/encode-decode>

NAME

EMAIL

LINK


File Edit View Format Tools Table Help

↶ ↷

B *I* U ~~S~~

☰

☰

A  *I*_x

{ }

Ω

😊

☰

☰

☰

☰

☰

☰

...

Input comment, please

p

0 words 