Java

关注者 87 被浏览 6569

Java 中字节流与字符流的区别?

5 个回答 默认排序

胖胖
ciaoshen.com

94 人赞同了该回答

先推荐两篇不错的文章：
字符编码笔记：ASCII，Unicode和UTF-8
深入分析 Java 中的中文编码问题

首先明确“**字节（Byte）**”和“**字符（Character）**”的大小：

- **1 byte = 8 bit**
- **1 char = 2 byte = 16 bit** (Java默认UTF-16编码)

虽然1 bit才是数据真正的最小单位，但1 bit 的信息量太少了。要表示一个有用的信息，需要好几个bit一起表示。所以除了硬件层面存在1个比特位的寄存器，大多数情况下，**字节是数据最小的基本单位**。我们熟知的基本型的大小都是8 bit（也就是1字节）的整数倍：

- boolean: 1 byte
- short: 2 byte
- int: 4 byte
- float: 4 byte
- long: 8 byte
- double: 8 byte

ASCII
原本对于西方世界来讲，可能根本用不到“字符”这个东西。1个字节就解决全部问题了。因为一个字节8 bit，最多为 $2^8 = 256$ 个符号编码。英语26个字母，再加几个常用符号，标点，256个码位足够了。这就熟悉的**ASCII码**。如下图，ASCII码一共收录了空格及94个“可印刷字符”。每个字母或标点占一个字节。简简单单一个表，就把所有编码都解决了。

相关问题

Java 等语言的 GC 为什么不实时释放内存？ 25 个回答

Java 程序员转 Android 开发有哪些经验可以分享？ 15 个回答

垃圾回收机制中，引用计数法是如何维护所有对象引用的？ 11 个回答

金融工程（包括程序化交易和超短线交易中）对于 Java 的要求大约是什么水平？如何学习？ 25 个回答

为什么 Java 只有值传递，但 C# 既有值传递，又有引用传递，这种语言设计有哪些好处？ 34 个回答

相关 Live 推荐


程序员如何写好一篇技术文章？

但西方世界不光只有英语一门语言。什么德语，法语，西班牙语都有自己的特殊字母。但这也没什么大不了的。每个国家都可以定义属于自己语言的特殊编码标准，而且大小照样不超过256。因为ASCII码中本身就有很多空码位没有使用。比如**ISO/IEC 8859-n**系就是国际标准化组织定义的一系列8位字符集。其中最常见的**ISO/IEC 8859-1**就是法语，芬兰语所用的西欧字符集。也是每个字母或符号用1个字节表示，下面这张表解决战斗。

中文编码

但26个字母到了中文，日语，韩语为代表的东亚象形文字面前就太小儿科了。汉字少说也有十万个。别说是8 bit，就算是16 bit（$2^{16} = 65536$）也不一定够。如果说一个汉字代表一个**“字符”**，从这个时候开始，“字符”的概念开始凌驾于“字节”之上了。收到的每一个字节不能简单地解码成一个字母了，而是需要好几个字节组合成一个“汉字”。我国的汉字编码现行标准是**GB 18030**，每个字可以由1个、2个或4个字节组成，编码空间有161万个字符。另一个中国常用编码集是Big5。

Unicode

在出现Unicode之前，几乎每一种文字都有一套自己的编码方式。同一段**“字节流”**，在美帝可能是"hello world"，到我们天朝就变成“锟斤拷”，“烫烫烫”了。所以**“Unicode”**可谓大势所趋。它的理念非常简单：**全世界每个不同语言的不同字符都统一编码，全球通行。**最初，每个字符占用2个字节，总共$2^{16} = 65536$个字符空间。比如，下面是“中国”两个字的Unicode代码。从第四版开始加入的“扩展字符集”开始使用4个字节（32 bit）编码。目前Unicode收录的字符规模大概在12万左右。

> **中** ⇒ **4e2d 000d**
> **国** ⇒ **000a 56fd**

UTF-16

编码里最容易搞混的一件事就是：**Unicode只是一套符号的编码。但计算机具体怎么读取这套编码，又是另外一件事。**

比如既然Unicode常规字符集占用2个字节，系统可以每次老老实实读取两个字节。然后用一个特殊符号告诉系统某个字符属于附加字符集，需要再往后读2个字节。比如说Java系统默认的UTF-16就是就是这样编码解码的：

考虑下面这句话：（图源：深入分析 Java 中的中文编码问题）

> **I am 君山**

Unicode字符集中：

> **君=541b。**拆开存在两个字节里：“**54**”和“**1b**”。
> **山=5c71。**拆开存在两个字节里：“**5c**”和“**71**”。

是不是很朴素？

UTF-8
但上面UTF-16的缺点也很明显：**就是所有英语字符“I am”也被迫用2个字节来编码。**比如，

> **I=49。**在前面补零变成：“**00**”和“**49**”。

考虑到英语是使用最广泛的语言，用2个字节为1字节信息编码，浪费了内存空间。最好是让英语保持ASCII的编码，用1个字节，汉字等其他字符才用2个或更长的字节表示。这里就涉及到一个技术问题：**怎么让系统知道一个字符是用1个还是2个还是3个字节编码的呢？**这就是UTF-8做的事。

如下图所示，这里UTF-8可变长编码用到了一个小技巧：**用几位冗余信息告诉系统，当前字符有没有结束，是不是还需要继续往下读下一个字节。**

可以看到如果一个字节是**以“0”开头**的，说明是一个ASCII字符，只占一个字节。如果是**“11”开头**的，说明这个字符占用多个字节。后续每个“10”打头的字节都是这个字符的一部分。

下图演示了字符串**“I am 君山”**用 UTF-8 编码的结果：

> **君 = 541b = 0101 0100 0001 1011 (Unicode)**

需要用3个字节编码，把0101010000011011切成3部分变成：

> 0101 010000 011011

分别套上UTF-8字符头：

> **1110** 0101 **10** 010000 **10** 011011 = e5 90 9b

所以如上图所示，**“君”**字用UTF-8编码就成了：**e5909b。**

总而言之，一切都是字节流，其实没有字符流这个东西。字符只是根据编码集对字节流翻译之后的产物。

Java I/O的编码系统
Java IO库有两个支系，

- 面向字节流的**InputStream**和**OutputStream**
- 面向字符的**Reader**和**Writer**

之前说了，字节流的InputStream和OutputStream是一切的基础。实际总线中流动的只有字节流。需要对字节流做特殊解码才能得到字符流。Java中负责从字节流向字符流解码的桥梁是：

根据上面的编码规则，Reader返回的是一个解码后的Unicode码元。包装在一个int整型里返回。

```
abstract int read();
```

也就是收到3个字节后，去掉UTF-8报头，拼装起来得到“君”字的Unicode码元。

> **1110** 0101 **10** 010000 **10** 011011 ⇒ 0101 0100 0001 1011

然后包装在4个字节的int整型里返回：

> 0101 0100 0001 1011 ⇒ 0000 0000 0000 0000 0101 0100 0001 1011

write()方法是一个相反的编码过程：

```
abstract void write(int c);
```

输入一个Unicode码元的int型，如果设定编码是UTF-8，内部会自动切割并添加报头。

以下是InputStreamReader和OutputStreamWriter的结构图，（图源：深入分析 Java 中的中文编码问题）

从图中可以猜到，实际负责编码和解码的是**StreamDecoder**类和**StreamEncoder**类。过程中必须指定使用的字符编码集**Charset**。所以InputStreamReader和OutputStreamWriter的构造器都带有Charset类型的参数。

> **InputStreamReader**(InputStream in, Charset cs)
> **OutputStreamWriter**(OutputStream out, Charset cs)

如果没有指定编码集，将使用系统默认编码集。而我们经常使用的FileInputReader和FileOutputWriter就是InputStreamReader和OutputStreamWriter的派生类。

内存String编码

另外一个要使用到Charset编码集的地方，是String的构造器和getBytes()方法。也可以通过参数控制具体使用的编码集。

```
String s = "这是一段中文字符串";
byte[] b = s.getBytes("UTF-8");
String n = new String(b,"UTF-8");
```

nio的字符编码

另外nio包里的ByteBuffer的asCharBuffer()方法也可以实现字节流和字符流之间的转换。

```
FileChannel fc=new FileInputStream(f).getChannel();
ByteBuffer bf=ByteBuffer.allocate(1024);
fc.read(bf);
```

但这里有个坑需要注意，asCharBuffer()方法，默认以**UTF-16BE**来解码byteBuffer里的字节。每个字符2字节。而String # getBytes()使用系统默认编码方式，大多数情况都不是UTF-16BE。所以经常CharBuffer里读取出来的会是乱码。

编辑于 2016-10-20

▲ 94 ▼ 10 条评论 分享 收藏 感谢 ··· 收起

陈肖恩
spam屠夫

2 人赞同了该回答

字节流就是普通的二进制流，读出来的是bit
字符流就是在字节流的基础按照字符编码处理，处理的是char

发布于 2016-01-07

▲ 2 ▼ 添加评论 分享 收藏 感谢 ···

士大夫
无产阶级

1 人赞同了该回答

完全不同，一个是二进制数据可以表达所以东西，一个就是一堆字符，只能是是字

编辑于 2016-10-19

▲ 1 ▼ 添加评论 分享 收藏 感谢 ···

用心阁
软件工程师

你搞清楚字符和字节的区别了吗？
一个字节是8位，只能有256个值，如果用来表示文字，可以表示ASCII码，包括控制字符，数字，符号，英文字母，西欧字母，制表符。

但是中文少说有几千汉字，所以一个字节表示不了，所以就用两个字节，编码方案有GB2312，GBK，Big5等。

后来又出现统一字符集，把各个常用语言都容纳进来，肯定1个字节也放不下。

Java使用Unicode，用char这个数据类型表示一个多字节的字符。

编辑于 2016-01-07

▲ 0 ▼ 添加评论 分享 收藏 感谢 ···

匿名用户

Java字符编码是Unicode的.不管是哪种unicode吧, 反正它都不是一个字符一个字节的.

所以字符流和字节流的差别在于基本单位不同.

发布于 2016-01-07

▲ 0 ▼ 添加评论 分享 收藏 感谢 ···

王森
java运算符 编辑话题经验

使用匿名身份回答

B I H " </> 1. • 📷 ▶ Σ

写回答...