以下列出了 Node.js Buffer 模块常用的方法（注意有些方法在旧版本是没有的）：

序号	方法 & 描述
1	**new Buffer(size)** 分配一个新的 size 大小单位为8位字节的 buffer。 注意, size 必须小于 kMaxLength，否则， RangeError。废弃的: 使用 Buffer.alloc() 代替（或 Buffer.allocUnsafe()）。
2	**new Buffer(buffer)** 拷贝参数 buffer 的数据到 Buffer 实例。废弃的: 使用 Buffer.from(buffer) 代替。
3	**new Buffer(str[, encoding])** 分配一个新的 buffer ，其中包含着传入的 str 字符串。 encoding 编码方式默认为 'utf8'。 废 Buffer.from(string[, encoding]) 代替。
4	**buf.length** 返回这个 buffer 的 bytes 数。注意这未必是 buffer 里面内容的大小。length 是 buffer 对象 数，它不会随着这个 buffer 对象内容的改变而改变。
5	**buf.write(string[, offset[, length]][, encoding])** 根据参数 offset 偏移量和指定的 encoding 编码方式，将参数 string 数据写入buffer。 offset 值是 0, encoding 编码方式默认是 utf8。 length 长度是将要写入的字符串的 bytes 大小。 返 类型，表示写入了多少 8 位字节流。如果 buffer 没有足够的空间来放整个 string，它将只会只 符串。 length 默认是 buffer.length - offset。 这个方法不会出现写入部分字符。

6	**buf.writeUIntLE(value, offset, byteLength[, noAssert])**
	将 value 写入到 buffer 里， 它由 offset 和 byteLength 决定，最高支持 48 位无符号整数，小
	如：
	`const buf = Buffer.allocUnsafe(6);`
	`buf.writeUIntLE(0x1234567890ab, 0, 6);`
	`// 输出: <Buffer ab 90 78 56 34 12>`
	`console.log(buf);`
	noAssert 值为 true 时，不再验证 value 和 offset 的有效性。 默认是 false。

7	**buf.writeUIntBE(value, offset, byteLength[, noAssert])**
	将 value 写入到 buffer 里， 它由 offset 和 byteLength 决定，最高支持 48 位无符号整数，大
	noAssert 值为 true 时，不再验证 value 和 offset 的有效性。 默认是 false。
	`const buf = Buffer.allocUnsafe(6);`
	`buf.writeUIntBE(0x1234567890ab, 0, 6);`
	`// 输出: <Buffer 12 34 56 78 90 ab>`
	`console.log(buf);`

8	**buf.writeIntLE(value, offset, byteLength[, noAssert])** 将value 写入到 buffer 里， 它由offset 和 byteLength 决定，最高支持48位有符号整数，小端 noAssert 值为 true 时，不再验证 value 和 offset 的有效性。 默认是 false。
9	**buf.writeIntBE(value, offset, byteLength[, noAssert])** 将value 写入到 buffer 里， 它由offset 和 byteLength 决定，最高支持48位有符号整数，大端 noAssert 值为 true 时，不再验证 value 和 offset 的有效性。 默认是 false。
10	**buf.readUIntLE(offset, byteLength[, noAssert])** 支持读取 48 位以下的无符号数字，小端对齐。noAssert 值为 true 时， offset 不再验证是否超 长度，默认为 false。
11	**buf.readUIntBE(offset, byteLength[, noAssert])**

	支持读取 48 位以下的无符号数字，大端对齐。noAssert 值为 true 时， offset 不再验证是否超 长度，默认为 false。
12	**buf.readIntLE(offset, byteLength[, noAssert])** 支持读取 48 位以下的有符号数字，小端对齐。noAssert 值为 true 时， offset 不再验证是否超 长度，默认为 false。
13	**buf.readIntBE(offset, byteLength[, noAssert])** 支持读取 48 位以下的有符号数字，大端对齐。noAssert 值为 true 时， offset 不再验证是否超 长度，默认为 false。
14	**buf.toString([encoding[, start[, end]]])** 根据 encoding 参数（默认是 'utf8'）返回一个解码过的 string 类型。还会根据传入的参数 sta 和 end (默认是 buffer.length)作为取值范围。
15	**buf.toJSON()** 将 Buffer 实例转换为 JSON 对象。
16	**buf[index]** 获取或设置指定的字节。返回值代表一个字节，所以返回值的合法范围是十六进制0x00到0xFF 至 255。
17	**buf.equals(otherBuffer)** 比较两个缓冲区是否相等，如果是返回 true，否则返回 false。
18	**buf.compare(otherBuffer)** 比较两个 Buffer 对象，返回一个数字，表示 buf 在 otherBuffer 之前，之后或相同。
19	**buf.copy(targetBuffer[, targetStart[, sourceStart[, sourceEnd]]])** buffer 拷贝，源和目标可以相同。 targetStart 目标开始偏移和 sourceStart 源开始偏移默认都 sourceEnd 源结束位置偏移默认是源的长度 buffer.length 。
20	**buf.slice([start[, end]])** 剪切 Buffer 对象，根据 start(默认是 0) 和 end (默认是 buffer.length) 偏移和裁剪了索引。 从 buffer 尾部开始计算的。
21	**buf.readUInt8(offset[, noAssert])** 根据指定的偏移量，读取一个无符号 8 位整数。若参数 noAssert 为 true 将不会验证 offset 偏 如果这样 offset 可能会超出buffer 的末尾。默认是 false。
22	**buf.readUInt16LE(offset[, noAssert])** 根据指定的偏移量，使用特殊的 endian 字节序格式读取一个无符号 16 位整数。若参数 noAss 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出 buffer 的末尾。默认是 false。
23	**buf.readUInt16BE(offset[, noAssert])** 根据指定的偏移量，使用特殊的 endian 字节序格式读取一个无符号 16 位整数，大端对齐。若 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出 buffer 的末尾。 false。
24	**buf.readUInt32LE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian 字节序格式读取一个无符号 32 位整数，小端对齐。 若 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer 的末尾。 false。
25	**buf.readUInt32BE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian 字节序格式读取一个无符号 32 位整数，大端对齐。 若 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer 的末尾。 false。
26	**buf.readInt8(offset[, noAssert])** 根据指定的偏移量，读取一个有符号 8 位整数。 若参数 noAssert 为 true 将不会验证 offset 偏 这意味着 offset 可能会超出 buffer 的末尾。默认是 false。

27	**buf.readInt16LE(offset[, noAssert])** 根据指定的偏移量，使用特殊的 endian 格式读取一个 有符号 16 位整数，小端对齐。 若参数 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出 buffer 的末尾。默认是 fals
28	**buf.readInt16BE(offset[, noAssert])** 根据指定的偏移量，使用特殊的 endian 格式读取一个 有符号 16 位整数，大端对齐。 若参数 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出 buffer 的末尾。默认是 fals
29	**buf.readInt32LE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian 字节序格式读取一个有符号 32 位整数，小端对齐。 若 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer 的末尾。 false。
30	**buf.readInt32BE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian 字节序格式读取一个有符号 32 位整数，大端对齐。 若 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer 的末尾。 false。
31	**buf.readFloatLE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian 字节序格式读取一个 32 位双浮点数，小端对齐。 若参为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer的末尾。默认是 fa
32	**buf.readFloatBE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian 字节序格式读取一个 32 位双浮点数，大端对齐。 若参为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer的末尾。默认是 fa
33	**buf.readDoubleLE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian字节序格式读取一个 64 位双精度数，小端对齐。 若参为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer 的末尾。默认是 fa
34	**buf.readDoubleBE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian字节序格式读取一个 64 位双精度数，大端对齐。 若参为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer 的末尾。默认是 fa
35	**buf.writeUInt8(value, offset[, noAssert])** 根据传入的 offset 偏移量将 value 写入 buffer。注意：value 必须是一个合法的无符号 8 位整 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 value 可能过大，或者 offset 可能 buffer 的末尾从而造成 value 被丢弃。 除非你对这个参数非常有把握，否则不要使用。默认是
36	**buf.writeUInt16LE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一号 16 位整数，小端对齐。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 value 可能过大，或者 offset 可能会超出buffer的末尾从而造成 value 被丢弃。 除非你对这个把握，否则尽量不要使用。默认是 false。
37	**buf.writeUInt16BE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一号 16 位整数，大端对齐。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 value 可能过大，或者 offset 可能会超出buffer的末尾从而造成 value 被丢弃。 除非你对这个把握，否则尽量不要使用。默认是 false。
38	**buf.writeUInt32LE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式(LITTLE-ENDIAN:小字节序)将 value 写入buf value 必须是一个合法的无符号 32 位整数，小端对齐。 若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着value 可能过大，或者offset可能会超出buffer的末尾从而造成 v 弃。 除非你对这个参数非常有把握，否则尽量不要使用。默认是 false。
39	**buf.writeUInt32BE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式(Big-Endian:大字节序)将 value 写入buffer。 必须是一个合法的有符号 32 位整数。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏

	这意味着 value 可能过大，或者offset可能会超出buffer的末尾从而造成 value 被丢弃。 除非你 非常有把握，否则尽量不要使用。默认是 false。
40	**buf.writeInt8(value, offset[, noAssert])**
41	**buf.writeInt16LE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一 signed 16 位整数。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味 能过大，或者 offset 可能会超出 buffer 的末尾从而造成 value 被丢弃。 除非你对这个参数非常 则尽量不要使用。默认是 false 。
42	**buf.writeInt16BE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一 signed 16 位整数。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味 能过大，或者 offset 可能会超出 buffer 的末尾从而造成 value 被丢弃。 除非你对这个参数非常 则尽量不要使用。默认是 false 。
43	**buf.writeInt32LE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一 signed 32 位整数。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味 能过大，或者 offset 可能会超出 buffer 的末尾从而造成 value 被丢弃。 除非你对这个参数非常 则尽量不要使用。默认是 false。
44	**buf.writeInt32BE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一 signed 32 位整数。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味 能过大，或者 offset 可能会超出 buffer 的末尾从而造成 value 被丢弃。 除非你对这个参数非常 则尽量不要使用。默认是 false。
45	**buf.writeFloatLE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer 。注意：当 value 不是一 点数类型的值时，结果将是不确定的。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏 这意味着 value可能过大，或者 offset 可能会超出 buffer 的末尾从而造成 value 被丢弃。 除非 数非常有把握，否则尽量不要使用。默认是 false。
46	**buf.writeFloatBE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer 。注意：当 value 不是一 点数类型的值时，结果将是不确定的。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏 这意味着 value可能过大，或者 offset 可能会超出 buffer 的末尾从而造成 value 被丢弃。 除非 数非常有把握，否则尽量不要使用。默认是 false。
47	**buf.writeDoubleLE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一 位double 类型的值。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意 可能过大，或者 offset 可能会超出 buffer 的末尾从而造成value被丢弃。 除非你对这个参数非 否则尽量不要使用。默认是 false。
48	**buf.writeDoubleBE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一 位double 类型的值。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意 可能过大，或者 offset 可能会超出 buffer 的末尾从而造成value被丢弃。 除非你对这个参数非 否则尽量不要使用。默认是 false。
49	**buf.fill(value[, offset][, end])** 使用指定的 value 来填充这个 buffer。如果没有指定 offset (默认是 0) 并且 end (默认是 buff ，将会填充整个buffer。