在 Node.js 中，Buffer 类是随 Node 内核一起发布的核心库。Buffer 库为 Node.js 带来了一种存储原始数据的方法，可以让 Node.js 处理二进制数据，每当需要在 Node.js 中处理I/O操作中移动的数据时，就有可能使用 Buffer 库。原始数据存储在 Buffer 类的实例中。

**Buffer 与字符编码**

Buffer 实例一般用于表示编码字符的序列，比如 UTF-8 、 UCS2 、 Base64 、或十六进制编码的数据。 通过使用显式的字符编码，就可以在 Buffer 实例与普通的 JavaScript 字符串之间进行相互转换。

const buf = Buffer.from('runoob', 'ascii');

// 输出 72756e6f6f62

console.log(buf.toString('hex'));

// 输出 cnVub29i

console.log(buf.toString('base64'));

**Node.js 目前支持的字符编码包括：**

* **ascii** - 仅支持 7 位 ASCII 数据。如果设置去掉高位的话，这种编码是非常快的。
* **utf8** - 多字节编码的 Unicode 字符。许多网页和其他文档格式都使用 UTF-8 。
* **utf16le** - 2 或 4 个字节，小字节序编码的 Unicode 字符。支持代理对（U+10000 至 U+10FFFF）。
* **ucs2** - **utf16le** 的别名。
* **base64** - Base64 编码。
* **latin1** - 一种把 **Buffer** 编码成一字节编码的字符串的方式。
* **binary** - **latin1** 的别名。
* **hex** - 将每个字节编码为两个十六进制字符。

**创建 Buffer 类**

Buffer 提供了以下 API 来创建 Buffer 类：

* **Buffer.alloc(size[, fill[, encoding]])：** 返回一个指定大小的 Buffer 实例，如果没有设置 fill，则默认填满 0
* **Buffer.allocUnsafe(size)：** 返回一个指定大小的 Buffer 实例，但是它不会被初始化，所以它可能包含敏感的数据
* **Buffer.allocUnsafeSlow(size)**
* **Buffer.from(array)：** 返回一个被 array 的值初始化的新的 Buffer 实例（传入的 array 的元素只能是数字，不然就会自动被 0 覆盖）
* **Buffer.from(arrayBuffer[, byteOffset[, length]])：** 返回一个新建的与给定的 ArrayBuffer 共享同一内存的 Buffer。
* **Buffer.from(buffer)：** 复制传入的 Buffer 实例的数据，并返回一个新的 Buffer 实例
* **Buffer.from(string[, encoding])：** 返回一个被 string 的值初始化的新的 Buffer 实例

// 创建一个长度为 10、且用 0 填充的 Buffer。

const buf1 = Buffer.alloc(10);

// 创建一个长度为 10、且用 0x1 填充的 Buffer。

const buf2 = Buffer.alloc(10, 1);

// 创建一个长度为 10、且未初始化的 Buffer。

// 这个方法比调用 Buffer.alloc() 更快，

// 但返回的 Buffer 实例可能包含旧数据，

// 因此需要使用 fill() 或 write() 重写。

const buf3 = Buffer.allocUnsafe(10);

// 创建一个包含 [0x1, 0x2, 0x3] 的 Buffer。

const buf4 = Buffer.from([1, 2, 3]);

// 创建一个包含 UTF-8 字节 [0x74, 0xc3, 0xa9, 0x73, 0x74] 的 Buffer。

const buf5 = Buffer.from('tést');

// 创建一个包含 Latin-1 字节 [0x74, 0xe9, 0x73, 0x74] 的 Buffer。

const buf6 = Buffer.from('tést', 'latin1');

**写入缓冲区**

**语法**

写入 Node 缓冲区的语法如下所示：

buf.write(string[, offset[, length]][, encoding])

**参数**

参数描述如下：

* **string** - 写入缓冲区的字符串。
* **offset** - 缓冲区开始写入的索引值，默认为 0 。
* **length** - 写入的字节数，默认为 buffer.length
* **encoding** - 使用的编码。默认为 'utf8' 。

根据 encoding 的字符编码写入 string 到 buf 中的 offset 位置。 length 参数是写入的字节数。 如果 buf 没有足够的空间保存整个字符串，则只会写入 string 的一部分。 只部分解码的字符不会被写入。

**返回值**

返回实际写入的大小。如果 buffer 空间不足， 则只会写入部分字符串。

**实例**

buf = Buffer.alloc(256);

len = buf.write("www.runoob.com");

console.log("写入字节数 : "+ len);

执行以上代码，输出结果为：

$node main.js

写入字节数 : 14

**从缓冲区读取数据**

**语法**

读取 Node 缓冲区数据的语法如下所示：

buf.toString([encoding[, start[, end]]])

**参数**

参数描述如下：

* **encoding** - 使用的编码。默认为 'utf8' 。
* **start** - 指定开始读取的索引位置，默认为 0。
* **end** - 结束位置，默认为缓冲区的末尾。

**返回值**

解码缓冲区数据并使用指定的编码返回字符串。

**实例**

buf = Buffer.alloc(26);

for (var i = 0 ; i < 26 ; i++) {

buf[i] = i + 97;

}

console.log( buf.toString('ascii')); // 输出: abcdefghijklmnopqrstuvwxyz

console.log( buf.toString('ascii',0,5)); // 输出: abcde

console.log( buf.toString('utf8',0,5)); // 输出: abcde

console.log( buf.toString(undefined,0,5)); // 使用 'utf8' 编码, 并输出: abcde

执行以上代码，输出结果为：

$ node main.js

abcdefghijklmnopqrstuvwxyz

abcde

abcde

abcde

**将 Buffer 转换为 JSON 对象**

**语法**

将 Node Buffer 转换为 JSON 对象的函数语法格式如下：

buf.toJSON()

当字符串化一个 Buffer 实例时，[JSON.stringify()](https://www.runoob.com/js/javascript-json-stringify.html" \t "_blank) 会隐式地调用该 **toJSON()**。

**返回值**

返回 JSON 对象。

**实例**

const buf = Buffer.from([0x1, 0x2, 0x3, 0x4, 0x5]);

const json = JSON.stringify(buf);

// 输出: {"type":"Buffer","data":[1,2,3,4,5]}

console.log(json);

const copy = JSON.parse(json, (key, value) => {

return value && value.type === 'Buffer' ?

Buffer.from(value.data) :

value;

});

// 输出: <Buffer 01 02 03 04 05>

console.log(copy);

执行以上代码，输出结果为：

{"type":"Buffer","data":[1,2,3,4,5]}

<Buffer 01 02 03 04 05>

**缓冲区合并**

**语法**

Node 缓冲区合并的语法如下所示：

Buffer.concat(list[, totalLength])

**参数**

参数描述如下：

* **list** - 用于合并的 Buffer 对象数组列表。
* **totalLength** - 指定合并后Buffer对象的总长度。

**返回值**

返回一个多个成员合并的新 Buffer 对象。

**实例**

var buffer1 = Buffer.from(('菜鸟教程'));

var buffer2 = Buffer.from(('www.runoob.com'));

var buffer3 = Buffer.concat([buffer1,buffer2]);

console.log("buffer3 内容: " + buffer3.toString());

执行以上代码，输出结果为：

buffer3 内容: 菜鸟教程www.runoob.com

**缓冲区比较**

**语法**

Node Buffer 比较的函数语法如下所示, 该方法在 Node.js v0.12.2 版本引入：

buf.compare(otherBuffer);

**参数**

参数描述如下：

* **otherBuffer** - 与 **buf** 对象比较的另外一个 Buffer 对象。

**返回值**

返回一个数字，表示 **buf** 在 **otherBuffer** 之前，之后或相同。

**实例**

var buffer1 = Buffer.from('ABC');

var buffer2 = Buffer.from('ABCD');

var result = buffer1.compare(buffer2);

if(result < 0) {

console.log(buffer1 + " 在 " + buffer2 + "之前");

}else if(result == 0){

console.log(buffer1 + " 与 " + buffer2 + "相同");

}else {

console.log(buffer1 + " 在 " + buffer2 + "之后");

}

执行以上代码，输出结果为：

ABC在ABCD之前

**拷贝缓冲区**

**语法**

Node 缓冲区拷贝语法如下所示：

buf.copy(targetBuffer[, targetStart[, sourceStart[, sourceEnd]]])

**参数**

参数描述如下：

* **targetBuffer** - 要拷贝的 Buffer 对象。
* **targetStart** - 数字, 可选, 默认: 0
* **sourceStart** - 数字, 可选, 默认: 0
* **sourceEnd** - 数字, 可选, 默认: buffer.length

**返回值**

没有返回值。

**实例**

var buf1 = Buffer.from('abcdefghijkl');

var buf2 = Buffer.from('RUNOOB');

//将 buf2 插入到 buf1 指定位置上

buf2.copy(buf1, 2);

console.log(buf1.toString());

执行以上代码，输出结果为：

abRUNOOBijkl

**缓冲区裁剪**

Node 缓冲区裁剪语法如下所示：

buf.slice([start[, end]])

**参数**

参数描述如下：

* **start** - 数字, 可选, 默认: 0
* **end** - 数字, 可选, 默认: buffer.length

**返回值**

返回一个新的缓冲区，它和旧缓冲区指向同一块内存，但是从索引 start 到 end 的位置剪切。

**实例**

var buffer1 = Buffer.from('runoob');

// 剪切缓冲区

var buffer2 = buffer1.slice(0,2);

console.log("buffer2 content: " + buffer2.toString());

执行以上代码，输出结果为：

buffer2 content: ru

**缓冲区长度**

**语法**

Node 缓冲区长度计算语法如下所示：

buf.length;

**返回值**

返回 Buffer 对象所占据的内存长度。

**实例**

var buffer = Buffer.from('www.runoob.com');

// 缓冲区长度

console.log("buffer length: " + buffer.length);

执行以上代码，输出结果为：

buffer length: 14