**The buffer class in Node.js**

[Node.js servers most often need to read and write to the filesystem](https://blog.logrocket.com/understanding-using-globs-node-js/) and, of course, files are stored in binaries. Node.js also deals with TCP streams, which secure connections to receivers before sending binary data in small chunks.

Streams of data being sent to the receiver need to be stored somewhere until the receiver is ready to take in more chunks of data for processing. This is where the Node.js buffer class comes into play. It handles and stores binary data outside of the [V8 engine](https://en.wikipedia.org/wiki/V8_(JavaScript_engine)).

Let’s review the different buffer methods and learn how to use them in our Node.js application.

**Node.js buffer methods**

One cool thing about the Node.js buffer module is that you don’t need to import it into your application before using its methods. Let’s review some important Node.js buffer methods that you should know.

**Buffer.alloc()**

The Buffer.alloc() method creates a new buffer of any size. When you use this method, you assign the size of the buffer in bytes. The expression below creates a buffer with a byte size of 6:

const buf = Buffer.alloc(6);

console.log(buf);

// This will print <Buffer 00 00 00 00 00 00>

**Buffer.write()**

The Buffer.write() method writes a string to the buffer, which can be useful when you need to stream strings in the form of buffers. You can write a string to a buffer using the below method:

const buf = Buffer.alloc(100); // Creating a new Buffer

const len = buf.write("Hello world!"); // Writing to the Buffer

// len is now 12

The Buffer.write() function returns the length of the string, which is stored in the buffer.

**Buffer.byteLength()**

You can check the length of a buffer object with the Buffer.byteLength() method. The code below demonstrates how to create a buffer, attach a size to it, and check the size of the buffer you just created:

var buf = Buffer.alloc(6);

//check the length of buffer created

var buffLen = Buffer.byteLength(buf);

//print buffer length

console.log(buffLen);

// This will print <6>

**Buffer.compare()**

The Buffer.compare() method enables you to compare two buffer objects to check whether they are equal. This method returns -1, 0, or 1, depending on the result of the comparison.

You can compare buffer objects with the Buffer.compare() method as seen below:

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

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

var a = Buffer.compare(buf1, buf2);

//This will return 0

console.log(a);

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

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

var a = Buffer.compare(buf1, buf2);

//This will return -1

console.log(a);

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

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

var a = Buffer.compare(buf1, buf2);

//This will return 1

console.log(a);

**Buffer.concat()**

Just like string concatenation, you can join two or more buffer objects into one object. You can also get the length of the new object:

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

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

var buffer3 = Buffer.from('z');

var arr = [buffer1, buffer2, buffer3];

/\*This will print buffer, !concat [ <Buffer 78>, <Buffer 79>, <Buffer 7a> ]\*/

console.log(arr);

//concatenate buffer with Buffer.concat method

var buf = Buffer.concat(arr);

//This will print <Buffer 78 79 7a> concat successful

console.log(buf);

**buf.entries()**

With buf.entries(), you can return a loop of indexes and bytes from the content of a buffer object, which is used to know the position and size of buffer contents:

var buf = Buffer.from('xyz');

for (a of buf.entries()) {

/\*This will print arrays of indexes and byte of buffer content \\[ 0, 120 \][ 1, 121 \][ 2, 122 ]\*/

console.log(a);

}

**Buffer.fill()**

The Buffer.fill() method enables you to create a buffer, allocate a size, and fill it with a specified value. The expression below shows how to use the Buffer.fill() method:

const b = Buffer.alloc(10).fill('a');

console.log(b.toString());

// This will print aaaaaaaaaa

**Buffer.from()**

The buffer.from() method enables you to create a new buffer from any object, like strings, buffer, arrays, and ArrayBuffer(). All you have to do is specify the object you want to create a buffer from.The syntax for using this method is Buffer.from(object[, offsetOrEncoding[,length]]).

[[](https://lp.logrocket.com/blg/learn-more)](https://lp.logrocket.com/blg/learn-more)

[Over 200k developers use LogRocket to create better digital experiences](https://lp.logrocket.com/blg/learn-more)

**[[https://blog.logrocket.com/wp-content/uploads/2022/08/rocket-button-icon.png](https://lp.logrocket.com/blg/learn-more)Learn more →](https://lp.logrocket.com/blg/learn-more)**

The offset or encoding parameter is optional; it’s used for specifying string encoding while converting to buffer. If you don’t specify the encoding parameter when creating buffers from a string, the buffer will be created with a default encoding of utf8.

You can also specify the number of bytes to expose with the length parameter, especially when creating buffer from ArrayBuffer. While creating a buffer from an array, the array bytes should fall between the range of 0 and 255. Otherwise, the array entries will be shortened to fit in.

The example below shows how to create a buffer from strings, arrays, and ArrayBuffer() using the buffer.from() method:

// Create a buffer from a string

var mybuff = Buffer.from("Nigeria");

//Print Buffer Created

console.log(mybuff);

// Create a buffer from a buffer

// Create buffer from string

var mybuff = Buffer.from("Nigeria");

// Create buffer from the first buffer created

var buff = Buffer.from(mybuff);

// Print out final buffer created.

console.log(buff);

// create a buffer from an array

const buf = Buffer.from([0x62, 0x75, 0x66, 0x66, 0x65, 0x72]);

// Create a buffer from an arraybuffer

const ab = new ArrayBuffer(10);

// Specify offset and length

const buf = Buffer.from(ab, 0, 2);

console.log(buff);

**buff.includes()**

If you want to determine whether a buffer object contains any values, you can use the buff.includes() method. With this method, you can search buffers to ascertain whether they contain expressions you wish to search for. The method returns a boolean true or false depending on whether a value is found:

const buf = Buffer.from('this is a buffer');

console.log(buf.includes('this'));

// This will print true

console.log(buf.includes(Buffer.from('a buffer example')));

// This will print false

**Buffer.isEncoding()**

To tell binaries apart, they must be encoded. You can use the Buffer.isEncoding() method to confirm whether a particular character encoding type is supported:

console.log(Buffer.isEncoding('hex'));

// This will print true

console.log(Buffer.isEncoding('utf-8'));

// This will print true

console.log(Buffer.isEncoding('utf/8'));

// This will print false

console.log(Buffer.isEncoding('hey'));

// This will print false

**buf.slice()**

Just like the slice method in JavaScript, buf.slice() is used to create a new array from selected elements in an array. When you slice an array, you create a new array with a list of elements you choose in the slice:

var a = Buffer.from('uvwxyz');

var b = a.slice(2,5);

console.log(b.toString());

//This will print wxy

**Buffer swap**

Buffer swap is used to swap the byte order of a buffer object. This method can also be used for fast [endianness](https://en.wikipedia.org/wiki/Endianness) conversion.

You can use buf.swap16(), buf.swap32(), and buf.swap64() to swap the byte order of a 16-bit, 32-bit, and 64-bit buffer object, respectively:

const buf1 = Buffer.from([0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8]);

console.log(buf1);

// This will print <Buffer 01 02 03 04 05 06 07 08>

//swap byte order to 16 bit

buf1.swap16();

console.log(buf1);

// This will print <Buffer 02 01 04 03 06 05 08 07>

//swap byte order to 32 bit

buf1.swap32();

console.log(buf1);

// This will print <Buffer 03 04 01 02 07 08 05 06>

//swap byte order to 64 bit

buf1.swap64();

console.log(buf1);

// This will print <Buffer 06 05 08 07 02 01 04 03>

**buf.json()**

The buf.json() method returns a JSON version of the buffer object:

const buf = Buffer.from([0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8]);

console.log(buf.toJSON());

// This will print {"type":"Buffer", data:[1,2,3,4,5,6,7,8]}