Node.js - Buffers

Pure JavaScript is Unicode friendly, but it is not so for binary data. While dealing with TCP streams or the file system, it's necessary to handle octet streams. Node provides Buffer class which provides instances to store raw data similar to an array of integers but corresponds to a raw memory allocation outside the V8 heap.

Buffer class is a global class that can be accessed in an application without importing the buffer module.

Creating Buffers

Node Buffer can be constructed in a variety of ways.

Method 1

Following is the syntax to create an uninitiated Buffer of **10** octets −

```
var buf = new Buffer(10);
```

Method 2

Following is the syntax to create a Buffer from a given array −

```
var buf = new Buffer([10, 20, 30, 40, 50]);
```

Method 3

Following is the syntax to create a Buffer from a given string and optionally encoding type −

```
var buf = new Buffer("Simply Easy Learning", "utf-8");
```

Though "utf8" is the default encoding, you can use any of the following encodings "ascii", "utf8", "utf16le", "ucs2", "base64" or "hex".

Writing to Buffers

Syntax

Following is the syntax of the method to write into a Node Buffer −

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

Parameters

Here is the description of the parameters used −

- **string** − This is the string data to be written to buffer.
- **offset** − This is the index of the buffer to start writing at. Default value is 0.
- **length** − This is the number of bytes to write. Defaults to buffer.length.
- **encoding** − Encoding to use. 'utf8' is the default encoding.

Return Value

This method returns the number of octets written. If there is not enough space in the buffer to fit the entire string, it will write a part of the string.

Example

Live Demo

```
buf = new Buffer(256);
len = buf.write("Simply Easy Learning");

console.log("Octets written : "+  len);
```

When the above program is executed, it produces the following result −

```
Octets written : 20
```

Reading from Buffers

Syntax

Following is the syntax of the method to read data from a Node Buffer −

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

Parameters

Here is the description of the parameters used −

encoding − Encoding to use. 'utf8' is the default encoding.
start − Beginning index to start reading, defaults to 0.
end − End index to end reading, defaults is complete buffer.

Return Value

This method decodes and returns a string from buffer data encoded using the specified character set encoding.

Example

Live Demo

```
buf = new Buffer(26);
for (var i = 0 ; i < 26 ; i++) {
  buf[i] = i + 97;
}

console.log( buf.toString('ascii'));       // outputs: abcdefghijklmnopqrstuvwxyz
console.log( buf.toString('ascii',0,5));   // outputs: abcde
console.log( buf.toString('utf8',0,5));    // outputs: abcde
console.log( buf.toString(undefined,0,5)); // encoding defaults to 'utf8', outputs abcde
```

When the above program is executed, it produces the following result −

```
abcdefghijklmnopqrstuvwxyz
abcde
abcde
abcde
```

Convert Buffer to JSON

Syntax

Following is the syntax of the method to convert a Node Buffer into JSON object −

```
buf.toJSON()
```

Return Value

This method returns a JSON-representation of the Buffer instance.

Example

Live Demo

```
var buf = new Buffer('Simply Easy Learning');
var json = buf.toJSON(buf);

console.log(json);
```

When the above program is executed, it produces the following result −

```
{ type: 'Buffer',
   data:
   [
      83,
      105,
      109,
      112,
      108,
      121,
      32,
      69,
      97,
      115,
      121,
      32,
      76,
      101,
      97,
      114,
      110,
      105,
      110,
      103
   ]
}
```

Concatenate Buffers

Syntax

Following is the syntax of the method to concatenate Node buffers to a single Node Buffer −

```
Buffer.concat(list[, totalLength])
```

Parameters

Here is the description of the parameters used −

list − Array List of Buffer objects to be concatenated.
totalLength − This is the total length of the buffers when concatenated.

Return Value

This method returns a Buffer instance.

Example

Live Demo

```
var buffer1 = new Buffer('TutorialsPoint ');
var buffer2 = new Buffer('Simply Easy Learning');
var buffer3 = Buffer.concat([buffer1,buffer2]);

console.log("buffer3 content: " + buffer3.toString());
```

When the above program is executed, it produces the following result −

```
buffer3 content: TutorialsPoint Simply Easy Learning
```

Compare Buffers

Syntax

Following is the syntax of the method to compare two Node buffers −

```
buf.compare(otherBuffer);
```

Parameters

Here is the description of the parameters used −

- **otherBuffer** − This is the other buffer which will be compared with **buf**

Return Value

Returns a number indicating whether it comes before or after or is the same as the otherBuffer in sort order.

Example

Live Demo

```
var buffer1 = new Buffer('ABC');
var buffer2 = new Buffer('ABCD');
var result = buffer1.compare(buffer2);

if(result < 0) {
   console.log(buffer1 +" comes before " + buffer2);
} else if(result === 0) {
   console.log(buffer1 +" is same as " + buffer2);
} else {
   console.log(buffer1 +" comes after " + buffer2);
}
```

When the above program is executed, it produces the following result −

```
ABC comes before ABCD
```

Copy Buffer

Syntax

Following is the syntax of the method to copy a node buffer −

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

Parameters

Here is the description of the parameters used −

targetBuffer − Buffer object where buffer will be copied.
targetStart − Number, Optional, Default: 0
sourceStart − Number, Optional, Default: 0
sourceEnd − Number, Optional, Default: buffer.length

Return Value

No return value. Copies data from a region of this buffer to a region in the target buffer even if the target memory region overlaps with the source. If undefined, the targetStart and sourceStart parameters default to 0, while sourceEnd defaults to buffer.length.

Example

Live Demo

```
var buffer1 = new Buffer('ABC');

//copy a buffer
var buffer2 = new Buffer(3);
buffer1.copy(buffer2);
console.log("buffer2 content: " + buffer2.toString());
```

When the above program is executed, it produces the following result −

```
buffer2 content: ABC
```

Slice Buffer

Syntax

Following is the syntax of the method to get a sub-buffer of a node buffer −

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

Parameters

Here is the description of the parameters used −

start − Number, Optional, Default: 0
end − Number, Optional, Default: buffer.length

Return Value

Returns a new buffer which references the same memory as the old one, but offset and cropped by the start (defaults to 0) and end (defaults to buffer.length) indexes. Negative indexes start from the end of the buffer.

Example

Live Demo

```
var buffer1 = new Buffer('TutorialsPoint');

//slicing a buffer
var buffer2 = buffer1.slice(0,9);
console.log("buffer2 content: " + buffer2.toString());
```

When the above program is executed, it produces the following result −

```
buffer2 content: Tutorials
```

Buffer Length

Syntax

Following is the syntax of the method to get a size of a node buffer in bytes −

```
buf.length;
```

Return Value

Returns the size of a buffer in bytes.

Example

Live Demo

```
var buffer = new Buffer('TutorialsPoint');

//length of the buffer
console.log("buffer length: " + buffer.length);
```

When the above program is executed, it produces following result −

```
buffer length: 14
```

Methods Reference

Following is a reference of Buffers module available in Node.js. For more detail, you can refer to the official documentation.

Class Methods

Sr.No.	Method & Description
1	**Buffer.isEncoding(encoding)** Returns true if the encoding is a valid encoding argument, false otherwise.
2	**Buffer.isBuffer(obj)** Tests if obj is a Buffer.
3	**Buffer.byteLength(string[, encoding])** Gives the actual byte length of a string. encoding defaults to 'utf8'. It is not the same as String.prototype.length, since String.prototype.length returns the number of characters in a string.
4	**Buffer.concat(list[, totalLength])** Returns a buffer which is the result of concatenating all the buffers in the list together.
5	**Buffer.compare(buf1, buf2)** The same as buf1.compare(buf2). Useful for sorting an array of buffers.

Revision #1
Created 16 December 2022 10:25:29 by Admin
Updated 16 December 2022 10:26:16 by Admin