**Buffer 与字符编码**

如下示例

// 读取字符串的二进制格式，指定其编码为 ascii

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

// 将字符串转为hex编码输出，输出 72756e6f6f62

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

// 将字符串转为base64编码，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

示例：

// 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**

1. 语法

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

1. 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'

1. 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

1. 实例

buf = Buffer.alloc(256);

len = buf.write("www.runoob.com");

console.log("写入字节数 : "+  len);

**从缓冲区读取数据**

1. 语法

读取 Node 缓冲区数据的语法如下所示：

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

1. 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。

1. 返回值

解码缓冲区数据并使用指定的编码返回字符串。

1. 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**

1. 语法

将 Node Buffer 转换为 JSON 对象的函数语法格式如下：

buf.toJSON()

1. 返回值

返回 JSON 对象。

1. 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);