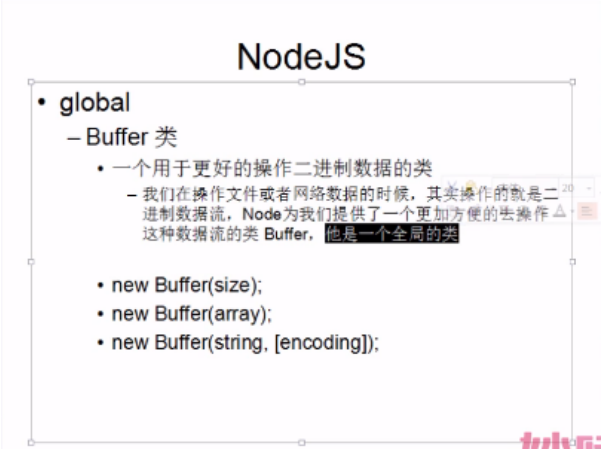


创建buffer类，类似于数组

Buffer 类的实例类似于整数数组，但 Buffer 的大小是固定的、且在 V8 堆外分配物理内存。 Buffer 的大小在被创建时确定，且无法调整。

为了使 Buffer 实例的创建更可靠、更不容易出错，各种 new Buffer() 构造函数已被 **废弃**，并由 Buffer.from()、Buffer.alloc()、和 Buffer.allocUnsafe() 方法替代。

开发者们应当把所有正在使用的 new Buffer() 构造函数迁移到这些新的 API 上。

- Buffer.from(array) 返回一个新建的包含所提供的字节数组的副本的 Buffer。
- 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) 返回一个新建的包含所提供的 Buffer 的内容的副本的 Buffer。
- Buffer.from(string[, encoding]) 返回一个新建的包含所提供的字符串的副本的 Buffer。
- Buffer.alloc(size[, fill[, encoding]]) 返回一个指定大小的被填满的 Buffer 实例。 这个方法会明显地比 Buffer.allocUnsafe(size) 慢，但可确保新创建的 Buffer 实例绝不会包含旧的和潜在的敏感数据。
- Buffer.allocUnsafe(size) 与 Buffer.allocUnsafeSlow(size) 返回一个新建的指定 size 的 Buffer，但它的内容**必须**被初始化，可以使用 buf.fill(0) 或完全写满。

如果 size 小于或等于 Buffer.poolSize 的一半，则 Buffer.allocUnsafe() 返回的 Buffer 实例**可能**会被分配进一个共享的内部内存池。

```
// 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('test');

// 创建一个包含 Latin-1 字节 [0x74, 0xe9, 0x73, 0x74] 的 Buffer。
const buf6 = Buffer.from('test', 'latin1');
```

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。

例子：

```
const buf = Buffer.from('hello world', 'ascii');

// 输出 68656c6c6f20776f726c64
console.log(buf.toString('hex'));

// 输出 aGVsbG8gd29ybGQ=
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。

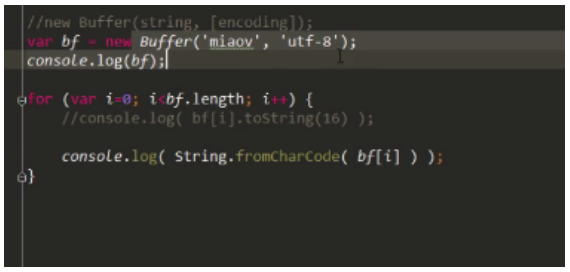
'base64' - Base64 编码。当从字符串创建 Buffer 时，按照 RFC4648 第 5 章的规定，这种编码也将正确地接受“URL 与文件名安全字母表”。

'latin1' - 一种把 Buffer 编码成一字节编码的字符串的方式（由 IANA 定义在 RFC1345 第 63 页，用作 Latin-1 补充块与 C0/C1 控制码）。

'binary' - 'latin1' 的别名。

'hex' - 将每个字节编码为两个十六进制字符。

注意：现代浏览器遵循 WHATWG 编码标准 将 'latin1' 和 ISO-8859-1 别名为 win-1252。 这意味着当进行例如 http.get() 这样的操作时，如果返回的字符编码是 WHATWG 规范列表中的，则有可能服务器真的返回 win-1252 编码的数据，此时使用 'latin1' 字符编码可能会错误地解码数据。



```
//new Buffer(string, [encoding]);
var bf = new Buffer('miaov', 'utf-8');
console.log(bf);

for (var i=0; i<bf.length; i++) {
    //console.log( bf[i].toString(16) );

    console.log( String.fromCharCode( bf[i] ) );
}
```

```
const bf5 = Buffer.from('空间看看');
```

```
console.log(String.fromCharCode(bf5)+"转化了")
```