**Buffer**模块: 缓存

Buffer.from(buffer): 复制传入实例的数据，并返回一个新的Buffer实例

Buffer.from(string): 返回一个被string的值初始化的新的buffer实例

Buffer.alloc(size, fill): 创建一个指定大小的Buffer实例，如果没有设置fill，则默认填满0

Buffer.allocUnsafe(10): 创建一个指定大小的buffer实例，如果没有fill，不会初始化实例（原 先存储的是什么就是什么）

buf.write(string[, offset[, length]][,encoding]): 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utf-8。返回值是写入的大小，超出部分忽略，而不会扩充

buf.toString(encoding[, start[, end]])): 返回值是指定编码的缓冲区数据

buf.toJSON(): 将对象变成json对象

Buffer.concat(listArr[, totalLength]): 合并缓存，totalLength表示合并后缓存的总长度

buf.compare(targetBuffer, targetstart, targetend, sourcestart, sourceend): 比较buffer1和 buffer2

buf.copy(targetbuf, targetstart, sourcestart, sourceend): 拷贝sourcebuffer到targetbuffer指定 位置，拷贝完后source改变不会影响target，也就是深拷贝

buf.slice(start, end): 剪切buf中的部分并返回，改剪切相当于浅拷贝，剪切部分改变剪切出 来的部分也会随之改变

**文件**模块：

var fs = require(‘fs’), 引入文件模块 //file system

fs.open(path[, mode], callback): 异步读取文件，异步读取文件最后一个参数为callback，callback第一个参数是 错误信息

fs.openSync(path, flag[, mode]): 同步读取文件，同步读取文件大多有返回值，flag表示打开文件方式（r:读取模式，w:写入模式， a:追加模式，当文件不存在是都会创建）

fs.readSync(fd, buffer, offset, length, 偏移量)：同步读取文件

fs.read(fd, buffer, offset, length, callback): 异步读取文件

fs.readFileSync(path, option): 同步读取文件，不需要打开文件，直接读取

fs.readFile(path, option, callback): 异步读取文件

fs.writeFile(path, buf|string, option, callback): 写入数据

fs.createReadStream(path[, option]): 创建读取流，返回一个ReadStream对象

fs.createWriteStream(): 创建文件写入流，返回一个writeStream对象

fs.statSync(path): 查看文件信息

fs.existsSync(path): 查看文件是否存在

fs.unlinkSync(path): 删除文件

fs.mkdir(fileName): 创建文件夹

fs.rmdir(filename): 删除文件夹

fs.realPathSync(file[, option]): 获取文件的绝对路劲

readStream.pipe(writeStream): 自动平衡读写操作，直接将文件读写到另一个文件

net:

Socket:

事件：

client.on(‘connect’, function () {})： 连接到服务器时触发

client.on(‘data, function (data) {}): 当有数据传输时触发

client.on(‘close’, function () {}): 监听关闭时间

client.on(‘end’,function () {}):

client.on(‘timeout’,function () {}):

client.on(‘error’,function () {}):

属性：

remoteAddress:

remotePost:

localAddress:

localPort:

方法：

client.setTimeout:

client.setEncoding:

client. write(string): 向Server发送字符串

client.end(): 关闭连接

Server: net.createServer() –> 创建服务

事件：

Server.on(‘listening’, callback) 服务启动触发的事件

Server.on(‘connection’,callback (socket) {}) 当有链接时触发的事件

Server.on(‘close’, callback(socket) {})

Server.on(‘err’, callback)

方法：

Server.listen(port, host)

Server.close()

Server.address()