# 实验14：buffer模块

实验背景

利用已掌握的知识，完成实例。如图所示：

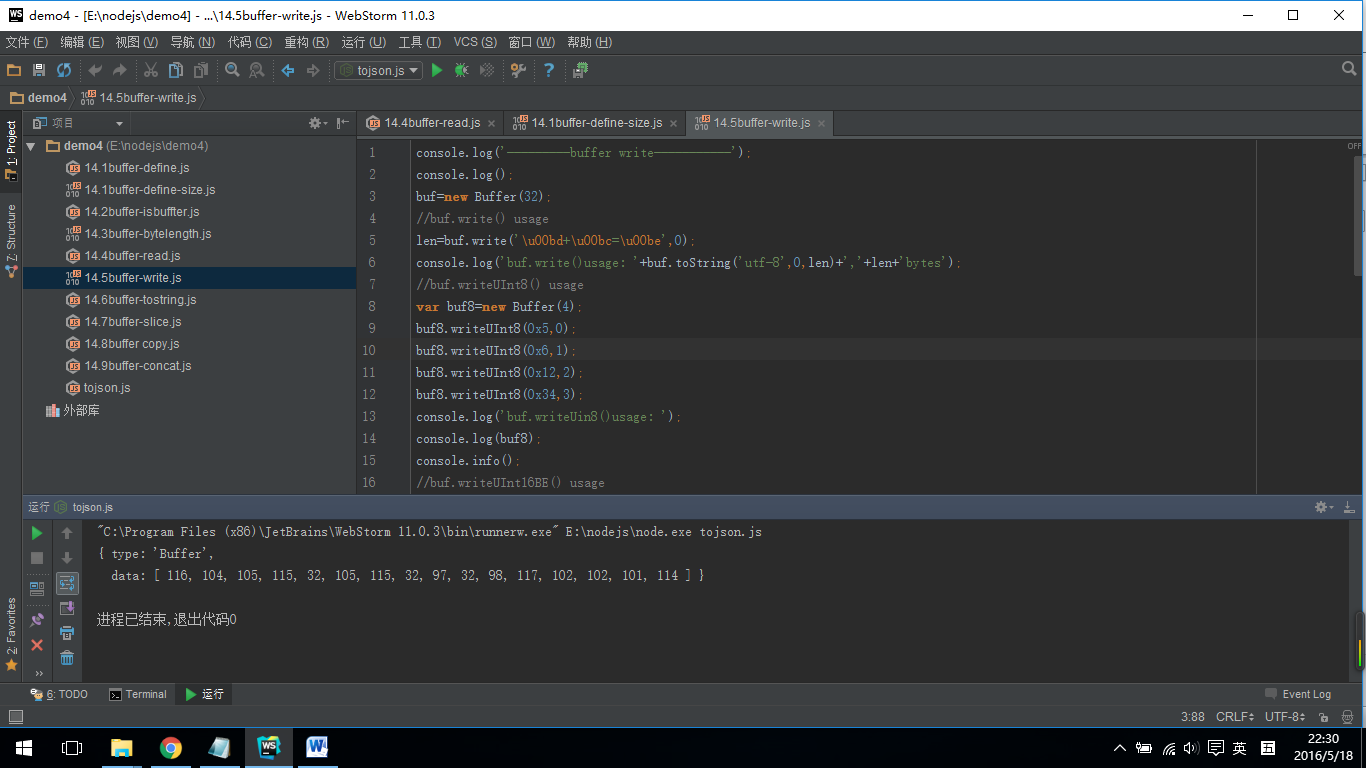


图1-1

实验目的

1、掌握创建 Buffer 类

2、掌握写入与读取缓冲区数据

3、掌握缓冲区其他操作

实验分析

本次实验要完成的有三个方面的知识。第一：完成实例《14.2buffer-defime-size》第二：完成实例《14.4buffer-read》,第三：完成实例《14.5buffer-write》。

实验步骤

1. 实例《14.2buffer-defime-size》

//定义并初始化buffer对象

var buffer=new Buffer('this is buffer','utf-8');

console.log('可变缓冲区buffer的长度是：'+buffer.length);

console.log('可变缓冲区buffer是：'+buffer);

////定义并初始化buffer16对象

var buffer16=new Buffer(16);

console.log('可变缓冲区buffer16的长度是：'+buffer16.length);

console.log('可变缓冲区buffer16是：'+buffer16);

buffer16.write('this is buffer',0,'utf-8');

console.log('可变缓冲区buffer16的长度是：'+buffer16.length);

console.log('可变缓冲区buffer16是：'+buffer16);

1. 实例《14.4buffer-read》

console.log('--------buffer read-----------');

console.log();

var str\_readuint8='';

var str\_readint8='';

var str\_readuint16le='';

var str\_readint16le='';

var str\_readuint16be='';

var str\_readint16be='';

var buf=new Buffer(4);

buf[0]=0x6;

buf[1]=0x8;

buf[2]=0x23;

buf[3]=0x57;

//str\_readuint8

str\_readuint8+='buf.readuint8(i)is: ';

for(var i=0;i<buf.length;i++){

str\_readuint8+=buf.readUInt8(i).toString(16)+' ';

}

console.log(str\_readuint8);

//str\_readint8

str\_readint8+='buf.readint8(i)is: ';

for(i=0;i<buf.length;i++){

str\_readint8+=buf.readInt8(i).toString(16)+' ';

}

console.log(str\_readint8);

//str\_readuint16le

str\_readuint16le+='buf.readuint16le(i)is: ';

for(i=0;i<buf.length-1;i++){

str\_readuint16le+=buf.readUInt16LE(i).toString(16)+' ';

}

console.log(str\_readuint16le);

//str\_readint16le

str\_readint16le+='buf.readint16le(i)is: ';

for(i=0;i<buf.length-1;i++){

str\_readint16le+=buf.readInt16LE(i).toString(16)+' ';

}

console.log(str\_readint16le);

//str\_readuint16be

str\_readuint16be+='buf.readuint16be(i)is: ';

for(i=0;i<buf.length-1;i++){

str\_readuint16be+=buf.readUInt16BE(i).toString(16)+' ';

}

console.log(str\_readuint16be);

//str\_readint16be

str\_readint16be+='buf.readint16be(i)is: ';

for(i=0;i<buf.length-1;i++){

str\_readint16be+=buf.readInt16BE(i).toString(16)+' ';

}

console.log(str\_readint16be);

console.info();

console.info('buf.readuint32le(i)is: '+buf.readUInt32LE(0).toString(16));

console.info('buf.readuint32BE(i)is: '+buf.readUInt32BE(0).toString(16));

console.info('buf.readint32le(i)is: '+buf.readInt32LE(0).toString(16));

console.info('buf.readint32BE(i)is: '+buf.readInt32BE(0).toString(16));

console.log('--------buffer read END-----------');

1. 实例《14.5buffer-write》

console.log('---------buffer write-----------');

console.log();

buf=new Buffer(32);

//buf.write() usage

len=buf.write('\u00bd+\u00bc=\u00be',0);

console.log('buf.write()usage: '+buf.toString('utf-8',0,len)+','+len+'bytes');

//buf.writeUInt8() usage

var buf8=new Buffer(4);

buf8.writeUInt8(0x5,0);

buf8.writeUInt8(0x6,1);

buf8.writeUInt8(0x12,2);

buf8.writeUInt8(0x34,3);

console.log('buf.writeUin8()usage: ');

console.log(buf8);

console.info();

//buf.writeUInt16BE() usage

var buf16BE=new Buffer(4);

buf16BE.writeUInt16BE(0x1234,0);

buf16BE.writeUInt16BE(0xabcd,2);

console.log('buf.writeUint16BE usage: ');

console.log(buf16BE);

//buf.writeUInt16LE() usage

var buf16LE=new Buffer(4);

buf16LE.writeUInt16LE(0x1234,0);

buf16LE.writeUInt16LE(0xabcd,2);

console.log('buf.writeUint16LE usage: ');

console.log(buf16LE);

console.info();

//buf.writeUInt32BE() usage

var buf32BE=new Buffer(4);

buf32BE.writeUInt32BE(0xfeedface,0);

console.log('buf.writeUint32BE usage: ');

console.log(buf32BE);

//buf.writeUInt32LE() usage

var buf32LE=new Buffer(4);

buf32LE.writeUInt32LE(0xfeedface,0);

console.log('buf.writeUint32LE usage: ')

console.log(buf32LE);

console.info();

console.log('---------buffer write-----------');

扩展练习

1.思考并实现，buffer方法参考手册。

方法参考手册

以下列出了 Node.js Buffer 模块常用的方法（注意有些方法在旧版本是没有的）：

|  |  |
| --- | --- |
| **序号** | **方法 & 描述** |
| 1 | **new Buffer(size)**  分配一个新的 size 大小单位为8位字节的 buffer。 注意, size 必须小于 kMaxLength，否则，将会抛出异常 RangeError。 |
| 2 | **new Buffer(buffer)**  拷贝参数 buffer 的数据到 Buffer 实例。 |
| 3 | **new Buffer(str[, encoding])** 分配一个新的 buffer ，其中包含着传入的 str 字符串。 encoding 编码方式默认为 'utf8'。 |
| 4 | **buf.length** 返回这个 buffer 的 bytes 数。注意这未必是 buffer 里面内容的大小。length 是 buffer 对象所分配的内存数，它不会随着这个 buffer 对象内容的改变而改变。 |
| 5 | **buf.write(string[, offset[, length]][, encoding])** 根据参数 offset 偏移量和指定的 encoding 编码方式，将参数 string 数据写入buffer。 offset 偏移量默认值是 0, encoding 编码方式默认是 utf8。 length 长度是将要写入的字符串的 bytes 大小。 返回 number 类型，表示写入了多少 8 位字节流。如果 buffer 没有足够的空间来放整个 string，它将只会只写入部分字符串。 length 默认是 buffer.length - offset。 这个方法不会出现写入部分字符。 |
| 6 | **buf.writeUIntLE(value, offset, byteLength[, noAssert])** 将value 写入到 buffer 里， 它由offset 和 byteLength 决定，支持 48 位计算，例如：  var b = new Buffer(6);  b.writeUIntBE(0x1234567890ab, 0, 6);  // <Buffer 12 34 56 78 90 ab>  noAssert 值为 true 时，不再验证 value 和 offset 的有效性。 默认是 false。 |
| 7 | **buf.writeUIntBE(value, offset, byteLength[, noAssert])** 将value 写入到 buffer 里， 它由offset 和 byteLength 决定，支持 48 位计算。noAssert 值为 true 时，不再验证 value 和 offset 的有效性。 默认是 false。 |
| 8 | **buf.writeIntLE(value, offset, byteLength[, noAssert])** 将value 写入到 buffer 里， 它由offset 和 byteLength 决定，支持 48 位计算。noAssert 值为 true 时，不再验证 value 和 offset 的有效性。 默认是 false。 |
| 9 | **buf.writeIntBE(value, offset, byteLength[, noAssert])** 将value 写入到 buffer 里， 它由offset 和 byteLength 决定，支持 48 位计算。noAssert 值为 true 时，不再验证 value 和 offset 的有效性。 默认是 false。 |
| 10 | **buf.readUIntLE(offset, byteLength[, noAssert])** 支持读取 48 位以下的数字。noAssert 值为 true 时， offset 不再验证是否超过 buffer 的长度，默认为 false。 |
| 11 | **buf.readUIntBE(offset, byteLength[, noAssert])** 支持读取 48 位以下的数字。noAssert 值为 true 时， offset 不再验证是否超过 buffer 的长度，默认为 false。 |
| 12 | **buf.readIntLE(offset, byteLength[, noAssert])** 支持读取 48 位以下的数字。noAssert 值为 true 时， offset 不再验证是否超过 buffer 的长度，默认为 false。 |
| 13 | **buf.readIntBE(offset, byteLength[, noAssert])** 支持读取 48 位以下的数字。noAssert 值为 true 时， offset 不再验证是否超过 buffer 的长度，默认为 false。 |
| 14 | **buf.toString([encoding[, start[, end]]])** 根据 encoding 参数（默认是 'utf8'）返回一个解码过的 string 类型。还会根据传入的参数 start (默认是 0) 和 end (默认是 buffer.length)作为取值范围。 |
| 15 | **buf.toJSON()** 将 Buffer 实例转换为 JSON 对象。 |
| 16 | **buf[index]** 获取或设置指定的字节。返回值代表一个字节，所以返回值的合法范围是十六进制0x00到0xFF 或者十进制0至 255。 |
| 17 | **buf.equals(otherBuffer)** 比较两个缓冲区是否相等，如果是返回 true，否则返回 false。 |
| 18 | **buf.compare(otherBuffer)** 比较两个 Buffer 对象，返回一个数字，表示 buf 在 otherBuffer 之前，之后或相同。 |
| 19 | **buf.copy(targetBuffer[, targetStart[, sourceStart[, sourceEnd]]])** buffer 拷贝，源和目标可以相同。 targetStart 目标开始偏移和 sourceStart 源开始偏移默认都是 0。 sourceEnd 源结束位置偏移默认是源的长度 buffer.length 。 |
| 20 | **buf.slice([start[, end]])** 剪切 Buffer 对象，根据 start(默认是 0 ) 和 end (默认是 buffer.length ) 偏移和裁剪了索引。 负的索引是从 buffer 尾部开始计算的。 |
| 21 | **buf.readUInt8(offset[, noAssert])** 根据指定的偏移量，读取一个有符号 8 位整数。若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 如果这样 offset 可能会超出buffer 的末尾。默认是 false。 |
| 22 | **buf.readUInt16LE(offset[, noAssert])** 根据指定的偏移量，使用特殊的 endian 字节序格式读取一个有符号 16 位整数。若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出 buffer 的末尾。默认是 false。 |
| 23 | **buf.readUInt16BE(offset[, noAssert])** 根据指定的偏移量，使用特殊的 endian 字节序格式读取一个有符号 16 位整数。若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出 buffer 的末尾。默认是 false。 |
| 24 | **buf.readUInt32LE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian 字节序格式读取一个有符号 32 位整数。 若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer 的末尾。默认是 false。 |
| 25 | **buf.readUInt32BE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian 字节序格式读取一个有符号 32 位整数。 若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer 的末尾。默认是 false。 |
| 26 | **buf.readInt8(offset[, noAssert])** 根据指定的偏移量，读取一个 signed 8 位整数。 若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出 buffer 的末尾。默认是 false。 |
| 27 | **buf.readInt16LE(offset[, noAssert])** 根据指定的偏移量，使用特殊的 endian 格式读取一个 signed 16 位整数。 若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出 buffer 的末尾。默认是 false。 |
| 28 | **buf.readInt16BE(offset[, noAssert])** 根据指定的偏移量，使用特殊的 endian 格式读取一个 signed 16 位整数。 若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出 buffer 的末尾。默认是 false。 |
| 29 | **buf.readInt32LE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian 字节序格式读取一个 signed 32 位整数。 若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer 的末尾。默认是 false。 |
| 30 | **buf.readInt32BE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian 字节序格式读取一个 signed 32 位整数。 若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer 的末尾。默认是 false。 |
| 31 | **buf.readFloatLE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian 字节序格式读取一个 32 位浮点数。 若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer的末尾。默认是 false。 |
| 32 | **buf.readFloatBE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian 字节序格式读取一个 32 位浮点数。 若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer的末尾。默认是 false。 |
| 33 | **buf.readDoubleLE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian字节序格式读取一个 64 位double。 若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer 的末尾。默认是 false。 |
| 34 | **buf.readDoubleBE(offset[, noAssert])** 根据指定的偏移量，使用指定的 endian字节序格式读取一个 64 位double。 若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 offset 可能会超出buffer 的末尾。默认是 false。 |
| 35 | **buf.writeUInt8(value, offset[, noAssert])** 根据传入的 offset 偏移量将 value 写入 buffer。注意：value 必须是一个合法的有符号 8 位整数。 若参数 noAssert 为 true 将不会验证 offset 偏移量参数。 这意味着 value 可能过大，或者 offset 可能会超出 buffer 的末尾从而造成 value 被丢弃。 除非你对这个参数非常有把握，否则不要使用。默认是 false。 |
| 36 | **buf.writeUInt16LE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一个合法的有符号 16 位整数。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味着 value 可能过大，或者 offset 可能会超出buffer的末尾从而造成 value 被丢弃。 除非你对这个参数非常有把握，否则尽量不要使用。默认是 false。 |
| 37 | **buf.writeUInt16BE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一个合法的有符号 16 位整数。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味着 value 可能过大，或者 offset 可能会超出buffer的末尾从而造成 value 被丢弃。 除非你对这个参数非常有把握，否则尽量不要使用。默认是 false。 |
| 38 | **buf.writeUInt32LE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入buffer。注意：value 必须是一个合法的有符号 32 位整数。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味着value 可能过大，或者offset可能会超出buffer的末尾从而造成 value 被丢弃。 除非你对这个参数非常有把握，否则尽量不要使用。默认是 false。 |
| 39 | **buf.writeUInt32BE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入buffer。注意：value 必须是一个合法的有符号 32 位整数。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味着value 可能过大，或者offset可能会超出buffer的末尾从而造成 value 被丢弃。 除非你对这个参数非常有把握，否则尽量不要使用。默认是 false。 |
| 40 | **buf.writeInt8(value, offset[, noAssert])**<="" td=""> |
| 41 | **buf.writeInt16LE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一个合法的 signed 16 位整数。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味着 value 可能过大，或者 offset 可能会超出 buffer 的末尾从而造成 value 被丢弃。 除非你对这个参数非常有把握，否则尽量不要使用。默认是 false 。 |
| 42 | **buf.writeInt16BE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一个合法的 signed 16 位整数。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味着 value 可能过大，或者 offset 可能会超出 buffer 的末尾从而造成 value 被丢弃。 除非你对这个参数非常有把握，否则尽量不要使用。默认是 false 。 |
| 43 | **buf.writeInt32LE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一个合法的 signed 32 位整数。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味着 value 可能过大，或者 offset 可能会超出 buffer 的末尾从而造成 value 被丢弃。 除非你对这个参数非常有把握，否则尽量不要使用。默认是 false。 |
| 44 | **buf.writeInt32BE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一个合法的 signed 32 位整数。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味着 value 可能过大，或者 offset 可能会超出 buffer 的末尾从而造成 value 被丢弃。 除非你对这个参数非常有把握，否则尽量不要使用。默认是 false。 |
| 45 | **buf.writeFloatLE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer 。注意：当 value 不是一个 32 位浮点数类型的值时，结果将是不确定的。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味着 value可能过大，或者 offset 可能会超出 buffer 的末尾从而造成 value 被丢弃。 除非你对这个参数非常有把握，否则尽量不要使用。默认是 false。 |
| 46 | **buf.writeFloatBE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer 。注意：当 value 不是一个 32 位浮点数类型的值时，结果将是不确定的。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味着 value可能过大，或者 offset 可能会超出 buffer 的末尾从而造成 value 被丢弃。 除非你对这个参数非常有把握，否则尽量不要使用。默认是 false。 |
| 47 | **buf.writeDoubleLE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一个有效的 64 位double 类型的值。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味着 value 可能过大，或者 offset 可能会超出 buffer 的末尾从而造成value被丢弃。 除非你对这个参数非常有把握，否则尽量不要使用。默认是 false。 |
| 48 | **buf.writeDoubleBE(value, offset[, noAssert])** 根据传入的 offset 偏移量和指定的 endian 格式将 value 写入 buffer。注意：value 必须是一个有效的 64 位double 类型的值。 若参数 noAssert 为 true 将不会验证 value 和 offset 偏移量参数。 这意味着 value 可能过大，或者 offset 可能会超出 buffer 的末尾从而造成value被丢弃。 除非你对这个参数非常有把握，否则尽量不要使用。默认是 false。 |
| 49 | **buf.fill(value[, offset][, end])** 使用指定的 value 来填充这个 buffer。如果没有指定 offset (默认是 0) 并且 end (默认是 buffer.length) ，将会填充整个buffer。 |

实验思考

1.思考并实现，独立设计一个小应用。