Buffer的使用

王红元
coderwhy

实力IT教育

数据的二进制

- 计算机中所有的内容：文字、数字、图片、音频、视频最终都会使用二进制来表示。
- JavaScript可以直接去处理非常直观的数据：比如字符串，我们通常展示给用户的也是这些内容。
- 不对啊，JavaScript不是也可以处理图片吗？
  - 事实上在网页端，图片我们一直是交给浏览器来处理的；
  - JavaScript或者HTML，只是负责告诉浏览器一个图片的地址；
  - 浏览器负责获取这个图片，并且最终讲这个图片渲染出来；
- 但是对于服务器来说是不一样的：
  - 服务器要处理的本地文件类型相对较多;
  - 比如某一个保存文本的文件并不是使用 utf-8进行编码的，而是用 GBK，那么我们必须读取到他们的二进制数据，再通过GKB转换成对应的文字；
  - 比如我们需要读取的是一张图片数据（二进制），再通过某些手段对图片数据进行二次的处理（裁剪、格式转换、旋转、添加滤镜），Node中有一个Sharp的库，就是读取图片或者传入图片的Buffer对其再进行处理；
  - 比如在Node中通过TCP建立长连接，TCP传输的是字节流，我们需要将数据转成字节再进行传入，并且需要知道传输字节的大小（客服端需要根据大小来判断读取多少内容）；

Buffer和二进制

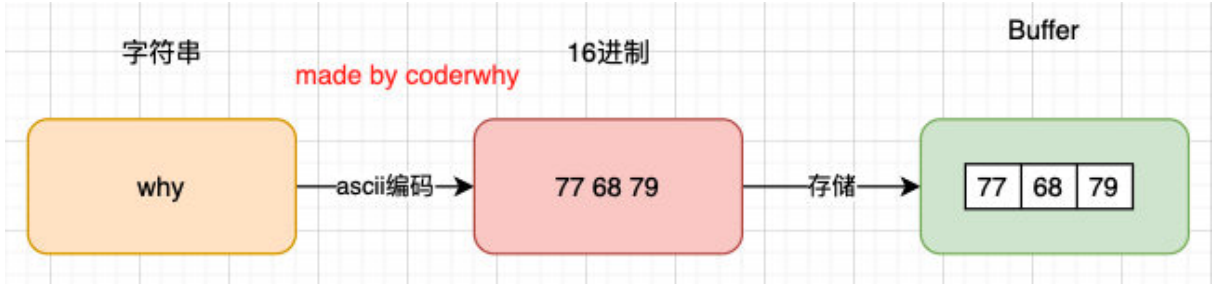
- 我们会发现，对于前端开发来说，通常很少会和二进制打交道，但是对于服务器端为了做很多的功能，我们必须直接去操作其二进制的数据；
- 所以Node为了可以方便开发者完成更多功能，提供给了我们一个类Buffer，并且它是全局的。
- 我们前面说过，Buffer中存储的是二进制数据，那么到底是如何存储呢？
  - 我们可以将Buffer看成是一个存储二进制的数组；
  - 这个数组中的每一项，可以保存8位二进制： 00000000
- 为什么是8位呢？
  - 在计算机中，很少的情况我们会直接操作一位二进制，因为一位二进制存储的数据是非常有限的；
  - 所以通常会将8位合在一起作为一个单元，这个单元称之为一个字节（byte）；
  - 也就是说 1byte = 8bit，1kb=1024byte，1M=1024kb;
  - 比如很多编程语言中的int类型是4个字节，long类型时8个字节；
  - 比如TCP传输的是字节流，在写入和读取时都需要说明字节的个数；
  - 比如RGB的值分别都是255，所以本质上在计算机中都是用一个字节存储的；

Buffer和字符串

- Buffer相当于是一个字节的数组，数组中的每一项对于一个字节的大小：
- 如果我们希望将一个字符串放入到Buffer中，是怎么样的过程呢？

```
const buffer01 = new Buffer("why");
console.log(buffer01);
```

- 它是怎么样的过程呢？




```
const buffer2 = Buffer.from("why");
console.log(buffer2);
```

如果是中文呢？

- 默认编码：utf-8

```
const buffer3 = Buffer.from("王红元");
console.log(buffer3); // <Buffer e7 8e 8b e7 ba a2 e5 85 83>

const str = buffer3.toString();
console.log(str); // 王红元
```

- 如果编码和解码不同：

```
const buffer3 = Buffer.from("王红元", 'utf16le');
console.log(buffer3);

const str = buffer3.toString('utf8');
console.log(str);
```

Buffer的其他创建

- Class: Buffer
  - Static method: Buffer.alloc(size[, fill[, encoding]])
  - Static method: Buffer.allocUnsafe(size)
  - Static method: Buffer.allocUnsafeSlow(size)
  - Static method: Buffer.byteLength(string[, encoding])
  - Static method: Buffer.compare(buf1, buf2)
  - Static method: Buffer.concat(list[, totalLength])
  - Static method: Buffer.from(array)
  - Static method: Buffer.from(arrayBuffer[, byteOffset[, length]])
  - Static method: Buffer.from(buffer)
  - Static method: Buffer.from(object[, offsetOrEncoding[, length]])
  - Static method: Buffer.from(string[, encoding])
  - Static method: Buffer.isBuffer(obj)
  - Static method: Buffer.isEncoding(encoding)

Buffer.alloc

- 来看一下Buffer.alloc:
  - 我们会发现创建了一个8位长度的Buffer，里面所有的数据默认为00；

```
const buffer01 = Buffer.alloc(8);
console.log(buffer01); // <Buffer 00 00 00 00 00 00 00 00>
```

- 我们也可以对其进行操作

```
buffer01[0] = 'w'.charCodeAt();
buffer01[1] = 100;
buffer01[2] = 0x66;
console.log(buffer01);

console.log(buffer01[0]);
console.log(buffer01[0].toString(16));
```

Buffer和文件读取

- 文本文件的读取：

```
fs.readFile('./test.txt', (err, data) => {
  console.log(data); // <Buffer 48 65 6c 6c 6f 20 57 6f 72 6c 64>
  console.log(data.toString()); // Hello World
})
```

- 图片文件的读取

```
fs.readFile('./zznh.jpg', (err, data) => {
  console.log(data); // <Buffer ff d8 ff e0 ... 40418 more bytes>
});
```

```
sharp('./test.png').resize(1000, 1000).toBuffer()
    .then(data => {
        fs.writeFileSync('./test_copy.png', data);
    })
```

Buffer的创建过程

- 事实上我们创建Buffer时，并不会频繁的向操作系统申请内存，它会默认先申请一个8 * 1024个字节大小的内存，也就是8kb

```
Buffer.poolSize = 8 * 1024;
let poolSize, poolOffset, allocPool;

const encodingsMap = ObjectCreate(null);
for (let i = 0; i < encodings.length; ++i)
  encodingsMap[encodings[i]] = i;

function createPool() {
  poolSize = Buffer.poolSize;
  allocPool = createUnsafeBuffer(poolSize).buffer;
  markAsUntransferable(allocPool);
  poolOffset = 0;
}
createPool();
```

Buffer.from源码

- 假如我们调用Buffer.from申请Buffer：
  - 这里我们以从字符串创建为例
  - node/lib/buffer.js：290行

```
Buffer.from = function from(value, encodingOrOffset, length) {
  if (typeof value === 'string')
    return fromString(value, encodingOrOffset);

  if (typeof value === 'object' && value !== null) {…
  }

  throw new ERR_INVALID_ARG_TYPE(
    'first argument',
    ['string', 'Buffer', 'ArrayBuffer', 'Array', 'Array-like Object'],
    value
  );
};
```

fromString的源码

```
function fromString(string, encoding) {
  let ops;
  if (typeof encoding !== 'string' || encoding.length === 0) {
    if (string.length === 0)
      return new FastBuffer();
    ops = encodingOps.utf8;
    encoding = undefined;
  } else {
    ops = getEncodingOps(encoding);
    if (ops === undefined)
      throw new ERR_UNKNOWN_ENCODING(encoding);
    if (string.length === 0)
      return new FastBuffer();
  }
  return fromStringFast(string, ops);
}
```

fromStringFast

- 接着我们查看fromStringFast：
  - 这里做的事情是判断剩余的长度是否还足够填充这个字符串；
  - 如果不足够，那么就要通过 createPool 创建新的空间；
  - 如果够就直接使用，但是之后要进行 poolOffset的偏移变化；
  - node/lib/buffer.js：428行

```
function fromStringFast(string, ops) {
  const length = ops.byteLength(string);

  // 比8kb还要长
  // Buffer.poolSize >>> 1
  // 8kb 8192个字节,向右无符号移动一位 4kb
  // 如果length 大于等于4kb的,那么就直接通过createFromString
  if (length >= (Buffer.poolSize >>> 1))
    return createFromString(string, ops.encodingVal);

  // 如果剩余的空间不够了
  if (length > (poolSize - poolOffset))
    createPool();
```

```
// 在空间内分配内存
let b = new FastBuffer(allocPool, poolOffset, length);
const actual = ops.write(b, string, 0, length);
if (actual !== length) {
  // byteLength() may overestimate. That's a rare case, though.
  b = new FastBuffer(allocPool, poolOffset, actual);
}
poolOffset += actual;
alignPool();
return b;
```