---

link: null
title: 珠峰架构师成长计划
description: 为了实现可读流，引用Readable接口并用它构造新对象
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=34 sentences=78, words=524

---

1. 自定义可读流

为了实现可读流，引用Readable接口并用它构造新对象

- 我们可以直接把供使用的数据push出去。
- 当push一个null对象就意味着我们想发出信号——这个流没有更多数据了。

```
var stream = require('stream');
var util = require('util');
util.inherits(Counter, stream.Readable);
function Counter(options) {
    stream.Readable.call(this, options);
    this._index = 0;
}
Counter.prototype._read = function() {
    if(this._index++<3){ this.push(this._index+''); }else{ this.push(null); } }; var counter="new" counter(); counter.on('data', function(data){ console.log("读到数据: " + data.tostring()); no maybe }); counter.on('end', console.log("读完了"); < code></3){>
```

2. 可写流

为了实现可写流，我们需要使用流模块中的Writable构造函数。 我们只需给Writable构造函数传递一些选项并创建一个对象。唯一需要的选项是write函数，该函数揭露数据块要往哪里写。

- chunk通常是一个buffer，除非我们配置不同的流。
- encoding是在特定情况下需要的参数，通常我们可以忽略它。
- callback是在完成处理数据块后需要调用的函数。这是写数据成功与否的标志。若要发出故障信号，请用错误对象调用回调函数

```
var stream = require('stream');
var util = require('util');
util.inherits(Writer, stream.Writable);
let stock = [];
function Writer(opt) {
    stream.Writable.call(this, opt);
}
Writer.prototype._write = function(chunk, encoding, callback) {
    setTimeout(()=>{
        stock.push(chunk.toString('utf8'));
        console.log("&#x589E;&#x52A0;: " + chunk);
        callback();
    },500)
};
var w = new Writer();
for (var i=1; i<=5; i++){ w.write("项目:" + i, 'utf8'); } w.end("结束写入",function(){ console.log(stock); }); < code></=5;>
```

3. 管道流

```
const stream = require('stream')

var index = 0;
const readable = stream.Readable({
    highWaterMark: 2,
    read: function () {
        process.nextTick(() => {
            console.log('push', ++index)
            this.push(index+'');
        })
    }
})

const writable = stream.Writable({
    highWaterMark: 2,
    write: function (chunk, encoding, next) {
        console.log('&#x5199;&#x5165;:', chunk.toString())
    }
})

readable.pipe(writable);
```

4. 实现双工流

有了双工流，我们可以在同一个对象上同时实现可读和可写，就好像同时继承这两个接口。 重要的是双工流的可读性和可写性操作完全独立于彼此。这仅仅是将两个特性组合成一个对象。

```
const {Duplex} = require('stream');
const inoutStream = new Duplex({
    write(chunk, encoding, callback) {
        console.log(chunk.toString());
        callback();
    },
    read(size) {
        this.push((++this.index)+'');
        if (this.index > 3) {
            this.push(null);
        }
    }
});

inoutStream.index = 0;
process.stdin.pipe(inoutStream).pipe(process.stdout);
```

5. 实现转换流

- 转换流的输出是从输入中计算出来的
- 对于转换流，我们不必实现read或write的方法，我们只需要实现一个transform方法，将两者结合起来。它有write方法的意思，我们也可以用它来push数据。

```
const {Transform} = require('stream');

const upperCase = new Transform({
    transform(chunk, encoding, callback) {
        this.push(chunk.toString().toUpperCase());
        callback();
    }
});

process.stdin.pipe(upperCase).pipe(process.stdout);
```

6. 对象流

默认情况下，流处理的数据是Buffer/String类型的值。有一个objectMode标志，我们可以设置它让流可以接受任何JavaScript对象。

```
const {Transform} = require('stream');
let fs = require('fs');
let rs = fs.createReadStream('./users.json');
rs.setEncoding('utf8');
let toJson = Transform({
    readableObjectMode: true,
    transform(chunk, encoding, callback) {
        this.push(JSON.parse(chunk));
        callback();
    }
});
let jsonOut = Transform({
    writableObjectMode: true,
    transform(chunk, encoding, callback) {
        console.log(chunk);
        callback();
    }
});
rs.pipe(toJson).pipe(jsonOut)
```

```
[
  {"name":"zfpx1","age":8},
  {"name":"zfpx2","age":9}
]
```

7. unshift

readable.unshift() 方法会把一块数据压回到Buffer内部。 这在如下特定情形下有用： 代码正在消费一个数据流，已经"乐观地"拉取了数据。 又需要"反悔-消费"一些数据，以便这些数据可以传给其他人用。

```
const {Transform} = require('stream');
const { StringDecoder } = require('string_decoder');
let decoder = new StringDecoder('utf8');
let fs = require('fs');
let rs = fs.createReadStream('./req.txt');

function parseHeader(stream, callback) {
    let header = '';
    rs.on('readable',onReadable);
    function onReadable() {

        let chunk;
        while(null != (chunk = rs.read())){
            const str = decoder.write(chunk);
            if(str.match(/\r\n\r\n/)){
                const split = str.split(/\r\n\r\n/);
                console.log(split);
                header+=split.shift();
                const remaining = split.join('\r\n\r\n');
                const buf = Buffer.from(remaining,'utf8');
                rs.removeListener('readable', onReadable);
                if(buf.length){
                    stream.unshift(buf);
                }
                callback(null,header,rs);
            }else{
                header += str;
            }
        }
    }
}
parseHeader(rs,function(err,header,stream){
    console.log(header);
    stream.setEncoding('utf8');
    stream.on('data',function (data) {
        console.log('data',data);
    });
});
```

```
Host: www.baidu.com
User-Agent: curl/7.53.0
Accept: */*

name=zfpx&age=9
```