流的概念

流是一组有序的，有起点和终点的字节数据传输手段 它不关心文件的整体内容，只关注是否从文件中读到了数据，以及读到数据之后的处理 流是一个抽象接口，被 Node 中的很多对象所实现。比如HTTP 服务器request和response对象都是流。

1.可读流createReadStream

实现了stream.Readable接口的对象,将对象数据读取为流数据,当监听data事件后,开始发射数据

```
fs.createReadStream = function(path, options) {
  return new ReadStream(path, options);
};
util.inherits(ReadStream, Readable);
```

1.1 创建可读流

```
var rs = fs.createReadStream(path,[options]);
```

1. path读取文件的路径
2. options
    - flags打开文件要做的操作,默认为'r'
    - encoding默认为null
    - start开始读取的索引位置
    - end结束读取的索引位置
    - highWaterMark读取缓存区默认的大小64kb

> 如果指定utf8编码highWaterMark要大于3个字节

1.1.1 监听data事件

流切换到流动模式,数据会被尽可能快的读出

```
rs.on('data', function (data) {
    console.log(data);
});
```

1.1.2 监听end事件

该事件会在读完数据后被触发

```
rs.on('end', function () {
    console.log('读取完成');
});
```

1.1.3 监听error事件

```
rs.on('error', function (err) {
    console.log(err);
});
```

1.1.4 设置编码

与指定{encoding:'utf8'}效果相同，设置编码

```
rs.setEncoding('utf8');
```

1.1.5暂停触发data恢复触发data

通过pause()方法和resume()方法

```
rs.on('data', function (data) {
    rs.pause();
    console.log(data);
});
setTimeout(function () {
    rs.resume();
},2000);
```

2.可写流createWriteStream

实现了stream.Writable接口的对象来将流数据写入到对象中

```
fs.createWriteStream = function(path, options) {
  return new WriteStream(path, options);
};

util.inherits(WriteStream, Writable);
```

2.1 创建可写流

```
var ws = fs.createWriteStream(path,[options]);
```

1. path写入的文件路径
2. options
   - flags打开文件要做的操作,默认为'w'
   - encoding默认为utf8
   - highWaterMark写入缓存区的默认大小16kb

2.1.1 write方法

```
ws.write(chunk,[encoding],[callback]);
```

1. chunk写入的数据buffer/string
2. encoding编码格式chunk为字符串时有用，可选
3. callback 写入成功后的回调

> 返回值为布尔值，系统缓存区满时为false,未满时为true

2.1.2 end方法

```
ws.end(chunk,[encoding],[callback]);
```

> 调用该方法关闭文件,迫使系统缓存区的数据立即写入文件中。不能再次写入

2.1.3 drain方法

```
var fs = require('fs');
var ws = fs.createWriteStream('./2.txt',{highWaterMark:5});
var i = 0;
function write(){
    var flag = true;
    while (flag&&i<10){
        flag = ws.write(''+i++);
    }
}
write();
ws.on('drain', function () {
    write();
});
```

3.pipe方法

3.1 pipe方法的原理

```
var fs = require('fs');
var ws = fs.createWriteStream('./2.txt');
var rs = fs.createReadStream('./1.txt');
```

```
rs.on('data', function (data) {
    var flag = ws.write(data);
    if(!flag)
    rs.pause();
});
ws.on('drain', function () {
    rs.resume();
});
rs.on('end', function () {
    ws.end();
});
```

3.2 pipe用法

```
readStream.pipe(writeStream);
var from = fs.createReadStream('./1.txt');
var to = fs.createWriteStream('./2.txt');
from.pipe(to);
```

> 将数据的滞留量限制到一个可接受的水平，以使得不同速度的来源和目标不会淹没可用内存。