Node系列-fs

> 本文对Node的内置模块fs进行介绍，包括模块的基本情况和简单使用。

1.0 模块介绍

fs文件模块(File System)是Node的内置核心模块之一，代码中可以通过 `var fs = require("fs")` 直接加载和使用，该模块提供了操作文件系统的诸多API，且所有方法均提供了**同步**和**异步**操作两个版本。

```
wendingding$ node
> fs
{ constants: { ··省略··· },
  Stats: [Function: Stats],
  F_OK: 0,
  R_OK: 4,
  W_OK: 2,
  X_OK: 1,
  access: [Function],
  accessSync: [Function],
  exists: [Function],
  existsSync: [Function],
  readFile: [Function],
  readFileSync: [Function],
  close: [Function],
  closeSync: [Function],
  open: [Function],
  openSync: [Function],
  read: [Function],
  readSync: [Function],
  write: [Function],
  writeSync: [Function],
  rename: [Function],
  renameSync: [Function],
  truncate: [Function],
  truncateSync: [Function],
  ftruncate: [Function],
  ftruncateSync: [Function],
  rmdir: [Function],
  rmdirSync: [Function],
  fdatasync: [Function],
  fdatasyncSync: [Function],
  fsync: [Function],
  fsyncSync: [Function],
  mkdir: [Function],
  mkdirSync: [Function],
  readdir: [Function],
  readdirSync: [Function],
  fstat: [Function],
  lstat: [Function],
```

```
  stat: [Function],
  fstatSync: [Function],
  lstatSync: [Function],
  statSync: [Function],
  readlink: [Function],
  readlinkSync: [Function],
  symlink: [Function],
  symlinkSync: [Function],
  link: [Function],
  linkSync: [Function],
  unlink: [Function],
  unlinkSync: [Function],
  fchmod: [Function],
  fchmodSync: [Function],
  lchmod: [Function],
  lchmodSync: [Function],
  chmod: [Function],
  chmodSync: [Function],
  lchown: [Function],
  lchownSync: [Function],
  fchown: [Function],
  fchownSync: [Function],
  chown: [Function],
  chownSync: [Function],
  _toUnixTimestamp: [Function: toUnixTimestamp],
  utimes: [Function],
  utimesSync: [Function],
  futimes: [Function],
  futimesSync: [Function],
  writeFile: [Function],
  writeFileSync: [Function],
  appendFile: [Function],
  appendFileSync: [Function],
  watch: [Function],
  watchFile: [Function],
  unwatchFile: [Function],
  realpathSync: [Function: realpathSync],
  realpath: [Function: realpath],
  mkdtemp: [Function],
  mkdtempSync: [Function],
  copyFile: [Function],
  copyFileSync: [Function],
  createReadStream: [Function],
  ReadStream:
   { [Function: ReadStream]
     super_:
      { [Function: Readable]
        ReadableState: [Function: ReadableState],
        super_: [Object],
        _fromList: [Function: fromList] } },
  FileReadStream:
   { [Function: ReadStream]
     super_:
      { [Function: Readable]
        ReadableState: [Function: ReadableState],
        super_: [Object],
        _fromList: [Function: fromList] } },
  createWriteStream: [Function],
```

```
  WriteStream:
   { [Function: WriteStream]
     super_: { [Function: Writable] WritableState: [Function: WritableState],
     super_: [Object] } },
  FileWriteStream:
   { [Function: WriteStream]
     super_: { [Function: Writable] WritableState: [Function: WritableState],
     super_: [Object] } } }
```

在该模块中，所有的文件操作都提供了 `同步` 和 `异步` 两种方法(譬如： `readFile` 和 `readFileSync` 方法)。这些方法在使用的时候，其使用方式和参数情况基本都是一致的。

如果是异步方法，那么其语法形式总是为 `function (err,callback)` ，最后一个参数代表操作完成的回调函数，第一个参数通常是操作错误的异常(如果操作成功，该参数的值为 `null` 或 `undefined`)，需注意异步方法不能保证执行顺序，如果对多个异步任务的执行顺序有要求，那么应该把后面的任务写在前面任务的回调函数中。

如果是同步方法，那么任何异常都会立即抛出，可以使用 `try/catch` 结构来处理异常。同步方法总是按顺序从上向下执行，是阻塞的。

```
//001 导入内置模块
var fs = require("fs");

//002 演示同步方法的执行
try {
    //方法说明：追加数据到文件中，如果文件不存在那么就创建
    fs.appendFileSync('./Hi.text', '文本数据');
    console.log('Success Info：创建文件成功!');
} catch(err)
{
    console.log('Error Info:',err);
}

//003 演示异步方法的使用
//方法说明：对文件执行重命名操作
fs.rename('./Hi.text', './Hello.text', function(error) {
    if(error) throw error;
    console.log("Success Info: 文件重命名成功！")

    //方法说明：查看重命名后的文件信息
    //依赖关系：需保证Hello.text文件已经存在
    fs.stat('./Hello.text', function(err, stats){
        if (err) throw err;
        console.log(`File Info: \n ${JSON.stringify(stats,null,4)}`);
      });
});
```

代码说明 上面给出了同步和异步方法使用的简单示例，代码中首先导入了fs模块，然后在当前目录中使用同步的方法(`appendFileSync`)来创建 `Hi.text` 文件并写入数据，随后调用异步方法(`rename`)来对文件执行重命名操作，继而打印(`stat`)该文件的信息。下面列出示例代码的执行细节。

BASH

```
wendingding$ node app.js
```

```
Success Info: 创建文件成功!
Success Info: 文件重命名成功!
File Info:
{
    "dev": 16777220,
    "mode": 33188,
    "nlink": 1,
    "uid": 501,
    "gid": 20,
    "rdev": 0,
    "blksize": 4194304,
    "ino": 8601332359,
    "size": 12,
    "blocks": 8,
    "atimeMs": 1543633694668.5117,
    "mtimeMs": 1543633694669.2383,
    "ctimeMs": 1543633694671.2622,
    "birthtimeMs": 1543633694668.5117,
    "atime": "2018-12-01T03:08:14.669Z",
    "mtime": "2018-12-01T03:08:14.669Z",
    "ctime": "2018-12-01T03:08:14.671Z",
    "birthtime": "2018-12-01T03:08:14.669Z"
}
wendingding$ cat Hello.text
文本数据
```

2.0 核心方法

fs文件模块(File System)中提供的方法很多，在具体介绍之前，为方便阅读我先简单列出这些方法并对它们进行功能性的区分。

```
001 读取和写入文件
open()        打开文件
close()       关闭文件
readFile()    读取文件的完整内容
writeFile()   把完整内容写入文件
read()        读取文件中指定部分的内容
write()       写入指定部分的内容到文件
appendFile()  追加数据到文件，如果文件不存在那么就创建

002 操作文件目录
mkdir()       创建文件夹(目录)
rmdir()       删除文件夹(目录)
readdir()     读取文件夹(目录)

003 文件或目录的信息
//获取信息
stat()        查看文件或目录信息
fstat()       查询文件的信息(open方法打开文件后可用)
lstat()       查看文件或目录信息(查看符号连接文件信息 only)
exists()      检查文件或目录是否存在
realpath()    获取文件或目录的绝对路径
access()      检查指定目录或文件的用户权限

//修改信息
```

```
utimes()      修改文件的访问时间和修改时间
futimes()     修改文件的访问时间和修改时间(open方法打开文件后可用)
chmod()       修改文件或目录的读写权限
fchmod()      修改文件或目录的读写权限(open方法打开文件后可用)
rename()      修改文件的名称(重命名)
truncate()    截断文件
ftruncate()   截断文件(open方法打开文件后可用)
watch()       监视文件
watchFile()   监视文件
```

① 读写文件

readFile 和 readFileSync

作用 异步(`同步`)读取文件的内容。

语法 `fs.readFile( path ,[ options ], callback ) | fs.readFileSync( path ,[ options ])`

参数

- path　　(`string | Buffer | URL | integer类型`) 指定文件名或文件描述符。
- options　　(`Object | string类型`) 指定编码和处理文件系统的 `flag值` 。
- callBack　　(`function类型`)执行完的回调函数(`err, data`)，其中data是文件的内容。

```
001    文件系统的flag
'a'    打开文件用于追加，如不存在则创建。
'ax'   类似 'a'，但如果文件已存在则抛出异常。
'a+'   打开文件用于读取和追加，如不存在则创建。
'ax+'  类似 'a+'，但如果文件已存在则抛出异常。
'as'   以同步模式打开文件用于追加，如不存在则创建。
'as+'  以同步模式打开文件用于读取和追加，如不存在则创建。

'r'    打开文件用于读取。如不存在则抛出异常。
'r+'   打开文件用于读取和写入。如不存在则抛出异常。
'rs+'  以同步模式打开文件用于读取和写入。指示操作系统绕开本地文件系统缓存(不建议使用)。

'w'    打开文件用于写入，不存在则创建，存在则截断。
'wx'   类似 'w'，但如果文件已存在则抛出异常。
'w+'   打开文件用于读取和写入，不存在则创建，存在则截断。
'wx+'  类似 'w+'，但如果文件已存在则抛出异常。

002 参数使用注意
options参数可以接收对象对象，如{"encoding":"utf8",flag:'r'};
options参数也可以接收字符串，如"utf8",表示读取文件使用utf编码模式。
默认情况下encoding的值为null,flag的值为r表示读取文件。

003 fs.readFile() 会缓存整个文件。 为了最小化内存占用，尽可能优先使用 fs.createReadStream()。
```

writeFile 和 writeFileSync

作用 异步(`同步`)写入文件。

语法 `fs.writeFile(file,data,[ options ],callback) | fs.writeFileSync(file,data,[ options])`

参数

- file （ `string | Buffer | URL | integer类型` ）指定完整路径的文件名或文件描述符。
- data （ `string | Buffer | TypedArray | DataView类型` ）指定需要写入的内容。
- options （ `Object | string类型` ）配置对象，可选项有 `encoding、mode和flag` 。
- callBack （ `function类型`)执行完的回调函数(`err`),参数值为错误对象。

示例

```
//备注：readAndWrite.js文件内容
var fs = require("fs");
//001 读取文件内容(readFile)
fs.readFile("./Hello.text",function(error,data){
  if(error) throw error;
  console.log("001 第二个参数省略返回Buffer类型:\n",data);
})

fs.readFile("./Hello.text","utf8",function(error,data){
  if(error) throw error;
  console.log("002 第二个参数为字符编码:",data);
})

fs.readFile("./Hello.text",{"encoding":"utf8",flag:'r'},function(error,data){
  if(error) throw error;
  console.log("003 第二个参数为对象:",data);
})

//002 写入文件内容(writeFile)
var fullPath = "./Hi.text"
fs.writeFile(fullPath,"Beautiful",function(error){
  if(error) throw error;
  //读取指定文件的内容
  var textContent = fs.readFileSync(fullPath,"utf8");
  console.log(textContent);
})

//命令行执行和输出的结果为:
wendingding$ node readAndWrite.js
001 第二个参数省略返回Buffer类型:
 <Buffer e6 96 87 e6 9c ac 3a 4e 69 63 65 20 74 6f 20 6d 65 65 74 20 75 20>
Beautiful
002 第二个参数为字符编码: 文本:Nice to meet u
003 第二个参数为对象: 文本:Nice to meet u
```

说明

```
001 方法说明
在写数据到文件时候，如果文件不存在，则创建文件，如果文件已存在，则覆盖文件。
如果 data 是一个 buffer，则忽略 encoding。
options参数可以是对象，也可以是字符串，如果该参数是一个字符串，则用来指定字符编码。

002 对同一个文件多次使用 fs.writeFile() 且不等待回调，是不安全的。
```

open 和 openSync

作用 异步(`同步`)打开文件。
语法 `fs.open( path, flags , [ mode ], callback ) | fs.openSync( path, flags , [ mode ])`
参数

- path (`string | Buffer | URL类型`) 指定的文件或路径。
- flags (`integer类型`) 指定对文件的操作，具体取值参考文件系统的 `flag` 值。
- mode (`integer类型`) 指定文件的模式，仅在创建时有效(默认可读可写)。
- callback (`Function类型`) 执行完毕的回调函数(`err` , `fd`)。

close 和 closeSync

作用 异步(`同步`)关闭文件。
语法 `fs.close( fd, callback ) | fs.closeSync( fd )`
参数

- fd (`integer类型`) 指定的文件描述符。
- callback (`Function类型`) 执行完毕的回调函数(`err` , `fd`)，参数值为错误对象。

read 和 readSync

write 和 writeSync

作用 异步(`同步`)的读取或写入文件。
语法

```
//同步 | 异步读取文件的内容。
fs.readSync(fd, buffer, offset, length, position)
fs.read(fd, buffer, offset, length, position, callback)

//同步 | 异步写入数据到文件中。
fs.writeSync(fd, buffer, offset, length, position)
fs.write(fd, buffer, offset, length, position, callback)
```

参数

- fd (`integer类型`) 通常为open方法回调中返回的文件描述符。
- buffer (`Buffer | TypedArray | DataView类型`) 指定将文件数据读取到哪个缓存区。
- offset (`integer类型`) 写入数据的开始位置(字节单位)。
- length (`integer类型`) 指定从文件读取的字节数。
- position (`integer类型`) 指定读取文件时的开始位置。
- callback (`Function类型`) 回调函数(`err` , `bytes(Read | Written)` , `buffer`)。

示例

```
//备注: openAndClose.js 文件的内容
//备注: 在当前目录中先准备好Hi.text文件, 内容为Nice !
var fs = require("fs");
var fullPath = "./Hi.text";
var bufferWrite = new Buffer(",请保持微笑 >.<");
var bufferRead = new Buffer(30);
```

```
console.log("000 打印文件的内容: ",fs.readFileSync(fullPath,"utf8"));

//001 打开文件
//语法: fs.open( path, flags , [ mode ], callback )
//参数: r+表示读取和写入文件，文件不存在则抛出异常
fs.open(fullPath,"r+",function(error,fd){
  if(error) throw error;
  console.log("001 打开文件成功! ")

  //002 写入文件
  //语法: fs.write(fd, buffer, offset, length, position, callback)
  fs.write(fd,bufferWrite,0,bufferWrite.length,5,function(err,writtenBytes,bufferData){
    if(err) throw err;
    console.log("002 写入的数据大小:",writtenBytes);
    console.log("002 写入的Buffer数据:",bufferData);
    console.log("002 写入的Buffer数据字符串:",bufferData.toString());

    //003 读取文件内容
    //fs.read(fd, buffer, offset, length, position, callback)
    fs.read(fd,bufferRead,0,bufferRead.length,0,function(err,readBytes,bufferData){
      if(err) throw err;
      console.log("003 读取的数据大小: ",readBytes);
      console.log("003 读取的Buffer数据:",bufferData);
      console.log("003 读取的Buffer数据字符串:",bufferData.toString());

      //004 关闭文件
      fs.close(fd,function(err){
        if(err) throw err;
        console.log("004 关闭文件成功! ");
      })
    });
  })
})

//命令行执行细节:
wendingding$ node openAndClose.js
000 打印文件的内容:  Nice !

001 打开文件成功!

002 写入的数据大小: 20
002 写入的Buffer数据:
 <Buffer 2c e8 af b7 e4 bf 9d e6 8c 81 e5 be ae e7 ac 91 20 3e 2e 3c>
002 写入的Buffer数据字符串: ,请保持微笑 >.<

003 读取的数据大小:  25
003 读取的Buffer数据:
<Buffer 4e 69 63 65 20 2c e8 af b7 e4 bf 9d e6 8c 81 e5 be ae e7 ac 91 20 3e 2e 3c ···00>
003 读取的Buffer数据字符串: Nice ,请保持微笑 >.<

004 关闭文件成功!
```

appendFile 和 appendFileSync

作用 异步(同步)追加数据到指定文件，如果文件不存在那么就先创建。

语法

`fs.appendFileSync(path, data, [options])`
`fs.appendFile(path, data, [options], callback)`

参数

- path　　(`string | Buffer | URL类型`) 指定的文件或路径。
- data　　(`string | Buffer类型`) 追加的具体数据。
- options　　(`Object | string类型`) 配置 `encoding、flag和mode` 等字段。
- callBack　　(`function类型`)执行完的回调函数(`err`)，参数值为错误对象。

示例

```
var fs = require("fs");
var filePath = "./Nice.text";

//001 同步追加数据到指定的文件(文件不存在)
try{
  //语法：fs.appendFileSync(path, data, [options])
  fs.appendFileSync(filePath,"妙：Nice!",{encoding:"utf8"});
  console.log('创建文件，并追加数据到文件');
}catch(err)
{
  console.log("Error Info:",err);
}

//002 异步方法追加数据
fs.appendFile(filePath," Best","utf8",function(err){
  if(err) throw err;
  console.log("Success Info: 追加数据成功！");

  //003 读取文件的内容验证
  var resultData = fs.readFileSync(filePath,"utf8");
  console.log("Success resultData:",resultData);
})

//命令行执行细节
wendingding$ node appendFile.js
创建文件，并追加数据到文件
Success Info: 追加数据成功!
Success resultData: 妙：Nice! Best
wendingding$ cat Nice.text
妙：Nice! Best
```

② 目录操作

mkdir 和 mkdirSync

作用 异步(`同步`)创建文件夹(目录)。
语法 `fs.mkdir( path , [ options ], callback ) | fs.mkdirSync( path , [ options ] )`
参数

- path (`string | Buffer | URL类型`) 指定的文件或路径。
- options (`Object | integer类型`) 配置选项有 `recursive` (是否创建父目录 10+)和 `mode` 。
- callBack (`function类型`)执行完的回调函数(`err`)， 参数值为错误对象。

readdir 和 readSync

作用 异步(`同步`)读取文件夹(目录)。
语法 `fs.readdir( path , [ options ] , callback ) | fs.readdirSync( path , [ options ] )`
参数

- path (`string | Buffer | URL类型`) 指定的文件或路径。
- options (`Object | integer类型`) 配置选项有 `encoding` 和 `withFileTypes` 。
- callBack (`function类型`)执行完的回调函数(`err` , `files`)， 表示错误对象和目录数组。

```HTML
Options配置项说明

[1]字符串作为参数，用于指定字符编码。
[2]对象类型作为参数
  encoding字段设置字符编码，默认为utf8 ，如果 encoding 设为 'buffer'，则返回的文件名是 Buffer。
  withFileTypes字段设置回调函数files数组的元素结构，设置为true则数组的元素是fs.Dirent，默认为false。
```

rmdir 和 rmdirSync

作用 异步(`同步`)移除文件夹(目录)。
语法 `fs.rmdir( path , callback ) | fs.rmdirSync( path )`
参数

- path (`string | Buffer | URL类型`) 指定的文件或路径。
- callback (`function类型`)执行完的回调函数(`err`)， 表示错误对象。

示例

```
//备注: file.js文件内容
var fs = require("fs");

//001 创建文件目录
fs.mkdir("./test",function(error){
  if(error) throw error;
  console.log("Success Info: ","创建目录成功")

  //002 读取目录
  fs.readdir("./nodefs",function(error,files){
    if(error) throw error;
    console.log("Success Info: ","读取目录成功")
    console.log("列出具体的内容:\n ",files);

    //003 删除目录
    // fs.rmdir("./test",function(){
    //   if(error) throw error;
    //   console.log("Success Info: ","删除test目录成功")
    // })
```

```
    //003 删除目录(遍历删除nodefs下面所有的子目录)
    files.forEach(element => {
      var fullPath = "./nodefs/" + element;
      console.log(fullPath);
      fs.rmdir(fullPath,function(){
        if(error) throw error;
        console.log("Success Info: ","删除"+fullPath+"目录成功")
      })
    });
  })
})

//命令行执行和输出的结果为:
wendingding$ node file.js
Success Info:  创建目录成功
Success Info:  读取目录成功
列出具体的内容:
  [ '.DS_Store', 'a', 'b' ]
./nodefs/.DS_Store
./nodefs/a
./nodefs/b
Success Info:  删除./nodefs/.DS_Store目录成功
Success Info:  删除./nodefs/a目录成功
Success Info:  删除./nodefs/b目录成功
```

③ 文件信息

access 和 accessSync

作用 异步(同步)检查指定文件或目录的用户权限。

语法 `fs.access( path, [ mode ], callback ) | fs.accessSync( path , [ mode ])`

参数

- path　　(`string | Buffer | URL类型`) 指定的文件或路径。
- callBack　　(`function类型`)执行检查完成的回调函数，如果失败则唯一的error参数有值。
- mode　　(`integer类型`) 默认值为 `fs.constants.F_OK` 要执行的可访问性检查。

HTML

```
001 文件的可访问性常量
F_OK 文件可见 == 0
R_OK 文件可读 == 4
W_OK 文件可写 == 2
X_OK 文件可执行(在 Windows上无效,效果同 fs.constants.F_OK) == 1

002 使用建议
不建议在调用 fs.open()、fs.readFile() 或 fs.writeFile() 之前使用 fs.access() 检查文件的可访问性。
因为其他进程可能在两个调用的间隙改变文件的状态。 应该直接打开、读取或写入文件，当文件无法访问时再处理错误。
```

stat 和 statSync

作用 异步(同步)查看文件的属性。

语法 `fs.stat( path ,[ options ], callback) | fs.statSync( path )`

参数

- path　　(`string | Buffer | URL类型`) 指定的文件或路径。
- options　　(`对象类型`)配置项bigint(布尔类型值)控制数值是否为 bigint 型。
- callback　　(`function类型`)回调函数(`err` , `stats`), `stats` 保存文件信息对象。

示例

```
//备注：test.js 文件内容
var fs = require("fs");

//001 stat方法说明：查看文件的属性
//不建议在调用 fs.open()、fs.readFile() 或 fs.writeFile() 之前使用 fs.stat() 检查文件是否存在。
//应该直接打开、读取或写入文件，当文件无效时再处理错误
fs.stat("./app.js", function(err,Info){
    console.log("Success Info:",Info);
    //查看是否为文件
    console.log(Info.isFile());
})

//002 statSync方法说明：查看目录(文件夹)属性
try{
  var fileInfo = fs.statSync("./nodefs");
  console.log("Success Info:",fileInfo);
}catch(err)
{
  console.log("Error Info: ",err);
}

//命令行执行和输出的结果为：
wendingding$ node test.js
Success Info: Stats {
  dev: 16777220,
  mode: 16877,
  nlink: 4,
  uid: 501,
  gid: 20,
  rdev: 0,
  blksize: 4194304,
  ino: 8601434025,
  size: 128,
  blocks: 0,
  atimeMs: 1543821747020.0398,
  mtimeMs: 1543821696833.2778,
  ctimeMs: 1543821701216.544,
  birthtimeMs: 1543821692999.7283,
  atime: 2018-12-03T07:22:27.020Z,
  mtime: 2018-12-03T07:21:36.833Z,
  ctime: 2018-12-03T07:21:41.217Z,
  birthtime: 2018-12-03T07:21:33.000Z }
  true
Success Info: Stats {
  dev: 16777220,

  mode: 33188,
  nlink: 1,
```

```
  uid: 501,
  gid: 20,
  rdev: 0,
  blksize: 4194304,
  ino: 8601330263,
  size: 839,
  blocks: 8,
  atimeMs: 1543821852077.4976,
  mtimeMs: 1543633822104.2488,
  ctimeMs: 1543633822104.2488,
  birthtimeMs: 1543632135491.0122,
  atime: 2018-12-03T07:24:12.077Z,
  mtime: 2018-12-01T03:10:22.104Z,
  ctime: 2018-12-01T03:10:22.104Z,
  birthtime: 2018-12-01T02:42:15.491Z }
bogon:fs wendingding$
```

在上面代码中异步方法回调函数中的 `Info` 和同步方法的返回值 `fileInfo` 都是 `fs.Stats` 类型对象。

`fs.Stats` 对象核心成员

mode	当前文件的权限标识。
nlink	当前文件的硬链接数量。
size	当前文件的大小(字节数)。
atime	当前文件的访问时间。
mtime	当前文件的修改时间。
ctime	最后改变文件状态的时间。
birthtime	创建文件的时间。
isFile ()	是否是一个文件。
isDirectory ()	是否是一个目录。
isSymbolicLink ()	是否是符号链接文件。

fstat 和 fstatSync

作用 异步(`同步`)查看文件的属性。
语法 `fs.fstat(fd ,[ options ], callback) | fs.fstatSync(fd ,[ options ])`
参数

- fd (`integer类型`) 使用open方法打开后返回的文件描述符。
- options (`对象类型`)配置项bigint(布尔类型值)控制数值是否为长整型。
- callback (`function类型`)回调函数(`err` , `stats`), `stats` 保存文件信息对象。

realpath 和 realpathSync

作用 异步(`同步`)计算文件路径，解析 .、.. 与符号链接。
语法 `fs.realpath(path ,[ options ], callback) | fs.realpathSync(path ,[ options ])`
参数

- path （ `string | Buffer | URL类型` ）指定的文件或路径。
- options （ `对象类型`)配置项encoding设置字符编码(默认为 `utf8`)。
- callback （ `function类型`)回调函数(`err` , `resolvedPath`)， `resolvedPath` 保存完整路径。

示例

```
//备注：fileInfo.js文件的内容
var fs = require("fs");
var fullPath = "./Hi.text";
//001 打开文件
fs.open(fullPath,"r",function(err,fd){
  if(err) throw err;
  console.log("Success Info: 打开文件成功！");
  //002 查看文件信息
  //语法：fs.fstat(fd ,[ options ], callback)
  fs.fstat(fd,function(err,fileInfo){
      if(err) throw err;
      console.log("Success fileInfo:",fileInfo);

      //003 打印文件的完整路径
      fs.realpath(fullPath,function(err,resultPath){
          if(err) throw err;
          console.log("Successs Info => ",resultPath);

          fs.close(fd,function(err){
            if(err) throw err;
            console.log("Success Info: 关闭文件成功！");
          })
      })
  })
})

//命令行执行细节
wendingding$ node fileInfo.js
Success Info: 打开文件成功!
Success fileInfo: Stats {
  dev: 16777220,
  mode: 33188,
  nlink: 1,
  uid: 501,
  gid: 20,
  rdev: 0,
  blksize: 4194304,
  ino: 8601453682,
  size: 25,
  blocks: 8,
  atimeMs: 1543894023563.2808,
  mtimeMs: 1543894022418.2258,
  ctimeMs: 1543894022418.2258,
  birthtimeMs: 1543633825190.9485,
  atime: 2018-12-04T03:27:03.563Z,
  mtime: 2018-12-04T03:27:02.418Z,
  ctime: 2018-12-04T03:27:02.418Z,
  birthtime: 2018-12-01T03:10:25.191Z }
Successs Info =>  /Users/文顶顶/Desktop/fs/Hi.text
Success Info: 关闭文件成功
```

rename 和 renameSync

作用 异步(`同步`)对文件进行重命名操作。
语法 `fs.rename( oldPath, newPath, callback ) | fs.renameSync( oldPath, newPath )`
参数

- oldPath (`string | Buffer | URL类型`)原来的文件全路径。
- newPath (`string | Buffer | URL类型`)目标文件全路径名称。
- callback (`function类型`)回调函数(`err`)，若文件已存在则覆盖。

示例

```
//备注：rename.js文件的内容
var fs = require("fs");
var oldPath = "./Hi.text";
var newPath = "./newHi.text";

//001 执行重命名操作
fs.rename(oldPath,newPath,function(err){
  if(err) throw err;
  console.log("Success Info:文件重命名成功！");

  //002 读取文件的内容
  fs.readFile(newPath,"utf8",function(err,fileData){
    if(err) throw err;
    console.log("Success fileData:",fileData);
  })
})

//命令行执行细节
wendingding$ node rename.js
Success Info:文件重命名成功!
Success fileData: Nice ,请保持微笑 >.<
```

“路还长，请别失望”