每天撸个API – File System

> Stability: 3 - Stable
>
> File System 的每个API都有异步方式和同步方式，两种方式的区别在于异步方式具有一个回调函数，并且这个回调函数的第一个参数为 err，如果操作成功，这个参数为 `null` 或 `undefined`

rename：改名

fs.rename(oldPath, newPath, callback)

```
//Asynchronous rename
fs.rename('/path/demo.txt', '/path/demo1.txt', function (err) {
  if (err) {
    throw err;
  }
  console.log('renamed complete');
});
```

fs.renameSync(oldPath, newPath)

```
//Synchronous rename
fs.renameSync('path/demo.txt', '/path/demo1.txt');
```

ftruncate：截断文件(fd)

fs.ftruncate(fd, len, callback)

```
//Asynchronous ftruncate
fs.open('/path/demo1.txt', 'a', function (err, fd) {
  if (err) {
    throw err;
  }
  fs.ftruncate(fd, 50, function (err) {
    if (err) {
      throw err;
    }
    console.log(fd);
    fs.close(fd, function () {
      console.log('Done');
    });
  });
});
```

fs.ftruncateSync(fd, len)

```
//Synchronous ftruncate
var fd = fs.openSync('/path/demo1.txt', 'a');
fs.ftruncateSync(fd, 50);
console.log(fd);
fs.closeSync(fd);
console.log('Done');
```

---

truncate：截断文件(path)

---

fs.truncate(path, len, callback)

```
fs.open('/path/demo1.txt', 'r+', function (err, fd) {
  if (err) {
    throw err;
  }
  fs.truncate('/path/demo1.txt', 10, function (err) {
    if (err) {
      throw err;
    }
    console.log(fd);
```

```
    fs.close(fd, function () {
      console.log('Done');
    });
  });
});
```

fs.truncateSync(path, len)

```
//Synchronous truncate
var fd = fs.openSync('/path/demo1.txt', 'r+');
fs.truncateSync('/path/demo1.txt', 5);
console.log(fd);
fs.close(fd);
console.log('Done');
```

---

chown：更改所有权(path)

---

fs.chown(path, uid, gid, callback)

```
//Asynchronous chown
fs.chown('/path/demo1.txt', 1001, 1000, function (err) {
  if (err) {
    throw err;
  }
  console.log('chown complete');
});
```

fs.chownSync(path, uid, gid)

```
//Synchronous chown
fs.chownSync('/path/demo1.txt', 1000, 1000);
```

---

fchown：更改所有权(fd)

fs.fchown(fd, uid, gid, callback)

```
//Asynchronous fchown
fs.open('/path/demo1.txt', 'a', function (err, fd) {
  if (err) {
    throw err;
  }
  fs.fchown(fd, 1000, 1000, function (err) {
    if (err) {
      throw err;
    }
    console.log(fd);
    fs.close(fd, function () {
      console.log('Done');
    });
  });
});
```

fs.fchownSync(fd, uid, gid)

```
//Synchronous fchown
var fd = fs.openSync('/path/demo1.txt', 'a');
fs.fchownSync(fd, 1001, 1000);
console.log(fd);
fs.closeSync(fd);
console.log('Done');
```

lchown：更改所有权(符号链接)

fs.lchown(path, uid, gid, callback)

fs.lchownSync(path, uid, gid)

> linux上不支持，撸不动.....

---

chmod：更改权限(path)

---

fs.chmod(path, mode, callback)

```
//Asynchronous chmod
fs.chmod('/path/demo1.txt', 0777, function (err) {
  if (err) {
    throw err;
  }
  console.log('chmod complete');
});
```

fs.chmodSync(path, mode)

```
//Synchronous chmod
fs.chmodSync('/path/demo1.txt', 0777);
```

> 这里的 mode 写成八进制 0777 或者对应的十进制 511 ,都可以达到Unix中chmod 777的效果。

---

fchmod：更改权限(fd)

---

fs.fchmod(fd, mode, callback)

```
//Asychronous fchmod
fs.open('/path/demo1.txt', 'a', function (err, fd) {
```

```
  if (err) {
    throw err;
  }
  fs.fchmod(fd, 0777, function (err) {
    if (err) {
      throw err;
    }
    console.log(fd);
    fs.close(fd, function () {
      console.log('Done');
    });
  });
});
```

fs.fchmodSync(fd, mode)

```
//Synchronous fchmod
var fd = fs.openSync('/path/demo1.txt', 'a');
fs.fchmodSync(fd, 0777);
console.log(fd);
fs.close(fd);
console.log('Done');
```

> 这里的 mode 写成八进制 0777 或者对应的十进制 511 ,都可以达到Unix中chmod 777的效果。

---

lchmod：更改权限(符号链接)

---

fs.lchmod(path, mode, callback)

fs.lchmodSync(path, mode)

> linux上不支持，撸不动……

---

stat：显示文件或文件系统状态(path)

fs.stat(path, callback)

```
//Asynchronous stat
fs.stat('/path/demo1.txt', function (err, stats) {
  if (err) {
    throw err;
  }
  console.log(stats);
});
```

回调中有两个参数 `(err, stats)` ，`stats`是一个`fs.stats`对象。

fs.statSync(path)

```
//Synchronous stat
var stats = fs.statSync('/home/yofine/example/demo1.txt');
console.log(stats);
```

log：

```
{ dev: 64770,
  mode: 33206,
  nlink: 1,
  uid: 1000,
  gid: 1000,
  rdev: 0,
  blksize: 4096,
  ino: 12845544,
  size: 5,
  blocks: 8,
  atime: Tue Dec 24 2013 00:46:47 GMT+0800 (CST),
  mtime: Tue Dec 24 2013 00:07:43 GMT+0800 (CST),
  ctime: Thu Dec 26 2013 13:04:53 GMT+0800 (CST) }
```

fstat：显示文件或文件系统状态(fd)

fs.fstat(fd, callback)

```
//Asynchronous stat
fs.open('/path/demo1.txt', 'a', function (err, fd) {
  if (err) {
    throw err;
  }
  fs.fstat(fd, function (err, stats) {
    if (err) {
      throw err;
    }
    console.log(stats);
    fs.close(fd, function () {
      console.log('Done');
    });
  });
});
```

fs.fstatSync(fd)

```
//Synchronous stat
var fd = fs.openSync('/path/demo1.txt', 'a');
var stats = fs.fstatSync(fd);
console.log(stats);
fs.closeSync(fd);
console.log('Done');
```

lstat：显示文件或文件系统状态(符号链接)

fs.lstat(path, callback)

```
//Asynchronous lstat
fs.lstat('/path/demo1.txt', function (err, stats) {
```

```
  if (err) {
    throw err;
  }
  console.log(stats);
});
```

fs.lstatSync(path)

```
//Synchronous lstat
var stats = fs.lstatSync('/path/demo1.txt');
console.log(stats);
```

fs.lstat 和 fs.stat 的区别：

> 当 `path` 为实际路径时，两者查看的文件或文件系统状态是一样的，但是当 `path` 为符号链接路径时，`fs.stat` 查看的是符号链接所指向实际路径的文件或文件系统状态，而 `fs.lstat` 查看的是符号链接路径的文件或文件系统状态。

---

link：硬链接

---

fs.link(srcpath, dstpath, callback)

```
//Asynchronous link
fs.link('/path/demo1.txt', '/path/demo1_h', function (err) {
  if (err) {
    throw err;
  }
  console.log('hardlink complete');
});
```

fs.linkSync(srcpath, dstpath)

```
//Synchronous link
fs.linkSync('/path/demo1.txt', '/path/demo1_h');
```

硬链接：

> 由于linux下的文件是通过索引节点（Inode）来识别文件，硬链接可以认为是一个指针，指向文件索引节点的指针，系统并不为它重新分配inode。每添加一个一个硬链接，文件的链接数就加1。硬链接相当于备份。

---

symlink：符号链接

---

fs.symlink(srcpath, dstpath, [type], callback)

```
//Asynchronous symlink
fs.symlink('/path/demo1.txt', '/path/demo1_s', function (err) {
  if (err) {
    throw err;
  }
  console.log('symlink complete');
});
```

fs.symlinkSync(srcpath, dstpath, [type])

```
//Synchronous symlink
fs.symlinkSync('/path/demo1.txt', '/path/demo1_s');
```

符号链接：

> 符号链接一类特殊的文件，相当于一个快捷方式，与硬链接不同，它是新建的一个文件，而且当 `ls -al` 时会标明是链接。

---

readlink：读取链接源地址

---

fs.readlink(path, callback)

```
//Asynchronous readlink
fs.readlink('/path/demo1_s', function (err, linkString) {
  if (err) {
    throw err;
  }
  console.log(linkString);
});
```

fs.readlinkSync(path)

```
//Synchronous readlink
var linkString = fs.readlinkSync('/path/demo1_s');
console.log(linkString);
```

> 以上两个的参数 `path` 要为 **符号链接。**

---

realpath：真实路径

---

fs.realpath(path, [cache], callback)

```
var cache = {'/example': '/home/yofine/example'};

//Asynchronous realpath
fs.realpath('/example/demo1_s', cache, function (err, resolvedPath) {
  if (err) {
    throw err;
  }
  console.log(resolvedPath);
});
```

log :

```
/home/yofine/example/demo1.txt
```

fs.realpathSync(path, [cache])

```
var cache = {'/example': '/home/yofine/example'};

//Synchronous realpath
var resolvedPath = fs.realpathSync('/example/demo1_s', cache);
console.log(resolvedPath);
```

log :

```
/home/yofine/example/demo1.txt
```

---

unlink：删除文件链接

---

fs.unlink(path, callback)

```
//Asynchronous unlink
fs.unlink('/path/demo2.txt', function (err) {
  if (err) {
    throw err;
  }
  console.log('unlink complete');
});
```

fs.unlinkSync(path)

```
//Synchronous unlink
fs.unlinkSync('/path/demo2.txt');
```

---

rmdir：删除空目录

fs.rmdir(path, callback)

```
//Asynchronous rmdir
fs.rmdir('/path/demo_dir', function (err) {
  if (err) {
    throw err;
  }
  console.log('rmdir complete');
});
```

fs.rmdirSync(path)

```
//Synchronous rmdir
fs.rmdirSync('/path/demo_dir');
```

mkdir：创建目录

fs.mkdir(path, [mode], callback)

```
//Asynchronous mkdir
fs.mkdir('/path/demo_dir', function (err) {
  if (err) {
    throw err;
  }
  console.log('mkdir complete');
});
```

fs.mkdirSync(path, [mode])

```
//Synchronous nkdir
fs.mkdir('/path/demo_dir');
```

`[mode]` 默认值为 `0777`

readdir：读取目录

fs.readdir(path, callback)

```
//Asynchronous readdir
fs.readdir('/path/example', function (err, files) {
  if (err) {
    throw err;
  }
  console.log(files);
});
```

fs.readdirSync(path)

```
//Synchronous readdir
var files = fs.readdirSync('/path/example');
console.log(files);
```

log :

```
[ 'demo1.txt', 'demo1_h', 'demo1_s', 'demo_dir' ]
```

open & close：打开/关闭文件

fs.open(path, flags, [mode], callback)

fs.close(fd, callback)

```
//Asynchronus open&close
fs.open('/path/demo1.txt', 'a', function (err, fd) {
  if (err) {
```

```
    throw err;
  }
  console.log(fd);
  fs.close(fd, function () {
    console.log('Async Done');
  });
});
```

fs.openSync(path, flags, [mode])

fs.closeSync(fd)

```
//Synchronous open&close
var fd = fs.openSync('/home/yofine/example/demo1.txt', 'a');
console.log(fd);
fs.closeSync(fd);
console.log('Done');
```

---

utimes：修改时间戳(path)

---

fs.utimes(path, atime, mtime, callback)

```
//Asynchronous utimes
fs.utimes('/path/demo1.txt', 1388648321, 1388648321, function (err) {
  if (err) {
    throw err;
  }
  console.log('utime complete');
});
```

fs.utimesSync(path, atime, mtime)

```
//Synchronous utimes
fs.utimesSync('/path/demo1.txt', 1388648321, 1388648321);
```

---

futimes：更改时间戳(fd)

fs.futimes(fd, atime, mtime, callback)

```
//Asynchronous futimes
fs.open('/path/demo1.txt', 'a', function (err, fd) {
  if (err) {
    throw err;
  }
  fs.futimes(fd, 1388648322, 1388648322, function (err) {
    if (err) {
      throw err;
    }
    console.log('futimes complete');
    fs.close(fd, function () {
      console.log('Done');
    });
  });
});
```

fs.futimesSync(fd, atime, mtime)

```
//Synchronous futimes
var fd = fs.openSync('/path/demo1.txt', 'a');
fs.futimesSync(fd, 1388648322, 1388648322);
fs.closeSync(fd);
console.log('Done');
```

fsync：同步

fs.fsync(fd, callback)

```
//Asynchronous fsync
```

```
fs.open('/path/demo2', 'a', function(err, fd) {
  if (err) throw err;
  fs.fsync(fd, function(err) {
    if (err) throw err;
    fs.close(fd, function(err) {
      if (err) throw err;
      console.log('Complete!')
    });
  });
});
```

fs.fsyncSync(fd)

```
//Synchronous fsync
var fd = fs.openSync('/path/demo2', 'a');
fs.fsyncSync(fd);
fs.closeSync(fd);
```

`fs.fsync` is just an asynchronous node wrapper for unix's fsync

---

write : 写入

```
var buffer = new Buffer('yofine');
```

fs.write(fd, buffer, offset, length, position, callback)

```
//Asynchronous write
fs.open('/path/demo1.txt', 'a', function(err, fd) {
  if (err) throw err;
  fs.write(fd, buffer, 0, buffer.length, null, function(err, written, buffer)
{
    if (err) throw err;
    console.log( written + 'bytes were written from buffer');
    fs.close(fd, function(err) {
      if (err) throw err;
      console.log('Complete');
```

```
    });
  });
});
```

fs.writeSync(fd, buffer, offset, length, position)

```
//Synchronous write
var fd = fs.openSync('/path/demo1.txt', 'a');
var written = fs.writeSync(fd, buffer, 0, buffer.length, null);
console.log(written + 'bytes were written from buffer');
fs.closeSync(fd);
```

---

read：读取

```
var buffer = new Buffer(100);
```

---

fs.read(fd, buffer, offset, length, position, callback)

```
//Asynchronous read
fs.open('/path/demo1.txt', 'r', function(err, fd) {
  if (err) throw err;
  fs.read(fd, buffer, 0, buffer.length, null, function(err, bytesRead, buffer)
{
    if (err) throw err;
    console.log('bytesRead : ' + bytesRead);
    fs.close(fd, function(err) {
      console.log('Complete!');
    });
  });
});
```

fs.readSync(fd, buffer, offset, length, position)

```
//Synchronous read
```

```
var fd = fs.openSync('/path/demo1.txt', 'r');
var bytesRead =  fs.readSync(fd, buffer, 0, buffer.length, null);
console.log('bytesRead : ' + bytesRead);
fs.close(fd);
```

---

readFile：读取文件

---

fs.readFile(filename, [options], callback)

```
//Asynchronous readFile
fs.readFile('/path/demo1.txt', function(err, data) {
  if (err) throw err;
  console.log(data);
});
```

fs.readFileSync(filename, [options])

```
//Synchronous readFile
var data = fs.readFileSync('/path/demo1.txt');
console.log(data);
```

---

writeFile：写入文件

> replacing the file if it already exists. data can be a string or a buffer.

fs.writeFile(filename, data, [options], callback)

```
//Asynchronous writeFile
fs.writeFile('/path/demo1.txt', 'hello yofine', function(err) {
```

```
  if (err) throw err;
  console.log('saved');
});
```

fs.writeFileSync(filename, data, [options])

```
//Synchronous writeFile
fs.writeFileSync('/path/demo1.txt', 'hello yofine');
```

---

appendFile：附加写入文件

> Asynchronously append data to a file, creating the file if it not yet exists. data can be a string or a buffer.

fs.appendFile(filename, data, [options], callback)

```
//Asynchronous appendFile
fs.appendFile('/path/demo1.txt', 'yofine', function(err) {
  if (err) throw err;
  console.log('Complete');
});
```

fs.appendFileSync(filename, data, [options])

```
//Synchronous appendFile
fs.appendFileSync('/path/demo1.txt', 'yofine');
console.log('Complete');
```

---

watchFile：监视文件

fs.watchFile(filename, [options], listener)

```
fs.watchFile('/path/demo1.txt', function(curr, prev) {
  console.log('the current mtime is: ' + curr.mtime);
  console.log('the previous mtime was: ' + prev.mtime);
})
```

unwatchFile：终止监视文件

fs.unwatchFile(filename, [listener])

```
fs.unwatchFile('/path/demo1.txt')
```

watch：监视

> Watch for changes on filename, where filename is either a file or a directory. The returned object is a fs.FSWatcher.

fs.watch(filename, [options], [listener])

```
fs.watch('/path/demo1.txt', function(event, filename) {
  console.log(event);
  console.log(filename);
});
```

exists：检查是否存在

fs.exists(path, callback)

```
//Asynchronous exists
fs.exists('/path/demo1.txt', function(exists) {
  console.log(exists ? 'exists' : 'not exists');
})
```

fs.existsSync(path)

```
//Synchronous exists
var exists = fs.existsSync('/path/demo1.txt');
console.log(exists ? 'exists' : 'not exists');
```

createReadStream：创建可读流

fs.createReadStream(path, [options])

> options is an object with the following defaults:

```
{ flags: 'r',
  encoding: null,
  fd: null,
  mode: 0666,
  autoClose: true
}
```

```
fs.createReadStream('/path/demo1.txt', options);
```

```
var http = require('http');
var fs = requ{ flags: 'r',
  encoding: null,
  fd: null,
  mode: 0666,
  autoClose: true
}ire('fs');

http.createServer(function(req, res) {

  var filename = __dirname+req.url;

  var readStream = fs.createReadStream(filename);

  readStream.on('open', function () {
    readStream.pipe(res);
  });

  readStream.on('error', function(err) {
    res.end(err);
  });
}).listen(8080);
```

---

createWriteStream：创建可读流

---

fs.createWriteStream(path, [options])

> options is an object with the following defaults:

```
{ flags: 'w',
  encoding: null,
  mode: 0666 }
```

```
fs.createWriteStream('/path/demo1.txt', options)
```

```
var http = require('http');
var fs = require('fs');
```

```
http.createServer(function(req, res) {
  var writeStream = fs.createWriteStream('./output');

  req.pipe(writeStream);

  req.on('end', function () {
    res.writeHead(200, {"content-type":"text/html"});
    res.end('<form method="POST"><input name="test" /><input type="submit"></f
orm>');
  });

  writeStream.on('error', function (err) {
    console.log(err);
  });
}).listen(8080);
```