Node.js 常用工具 Node.js GET/POST请求

Node.js 文件系统

Node.js 提供一组类似 UNIX（POSIX）标准的文件操作API。 Node 导入文件系统模块(fs)语法如下所示：

```
var fs = require("fs")
```

异步和同步

Node.js 文件系统（fs 模块）模块中的方法均有异步和同步版本，例如读取文件内容的函数有异步的 fs.readFile() 和同步的 fs.readFileSync()。

异步的方法函数最后一个参数为回调函数，回调函数的第一个参数包含了错误信息(error)。

建议大家使用异步方法，比起同步，异步方法性能更高，速度更快，而且没有阻塞。

实例

创建 input.txt 文件，内容如下：

```
菜鸟教程官网地址：www.runoob.com
文件读取实例
```

创建 file.js 文件, 代码如下：

```
var fs = require("fs");

// 异步读取
fs.readFile('input.txt', function (err, data) {
   if (err) {
       return console.error(err);
   }
   console.log("异步读取: " + data.toString());
});

// 同步读取
var data = fs.readFileSync('input.txt');
console.log("同步读取: " + data.toString());

console.log("程序执行完毕。");
```

以上代码执行结果如下：

```
$ node file.js
同步读取: 菜鸟教程官网地址：www.runoob.com
```

```
文件读取实例

程序执行完毕。
异步读取: 菜鸟教程官网地址:www.runoob.com
文件读取实例
```

接下来，让我们来具体了解下 Node.js 文件系统的方法。

打开文件

语法

以下为在异步模式下打开文件的语法格式：

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

参数

参数使用说明如下：

- **path** - 文件的路径。
- **flags** - 文件打开的行为。具体值详见下文。
- **mode** - 设置文件模式(权限)，文件创建默认权限为 0666(可读，可写)。
- **callback** - 回调函数，带有两个参数如：callback(err, fd)。

flags 参数可以是以下值：

Flag	描述
r	以读取模式打开文件。如果文件不存在抛出异常。
r+	以读写模式打开文件。如果文件不存在抛出异常。
rs	以同步的方式读取文件。
rs+	以同步的方式读取和写入文件。
w	以写入模式打开文件，如果文件不存在则创建。
wx	类似 'w'，但是如果文件路径存在，则文件写入失败。
w+	以读写模式打开文件，如果文件不存在则创建。
wx+	类似 'w+'， 但是如果文件路径存在，则文件读写失败。
a	以追加模式打开文件，如果文件不存在则创建。

ax	类似 'a'， 但是如果文件路径存在，则文件追加失败。
a+	以读取追加模式打开文件，如果文件不存在则创建。
ax+	类似 'a+'， 但是如果文件路径存在，则文件读取追加失败。

实例

接下来我们创建 file.js 文件，并打开 input.txt 文件进行读写，代码如下所示：

```
var fs = require("fs");

// 异步打开文件
console.log("准备打开文件! ");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
       return console.error(err);
   }
  console.log("文件打开成功! ");
});
```

以上代码执行结果如下：

```
$ node file.js
准备打开文件!
文件打开成功!
```

获取文件信息

语法

以下为通过异步模式获取文件信息的语法格式：

```
fs.stat(path, callback)
```

参数

参数使用说明如下：

- **path** - 文件路径。
- **callback** - 回调函数，带有两个参数如：(err, stats), **stats** 是 fs.Stats 对象。

fs.stat(path)执行后，会将stats类的实例返回给其回调函数。可以通过stats类中的提供方法判断文件的相关属性。例如判断是否为文件：

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

fs.stat('/Users/liuht/code/itbilu/demo/fs.js', function (err, stats) {
    console.log(stats.isFile());         //true
})
```

stats类中的方法有：

方法	描述
stats.isFile()	如果是文件返回 true，否则返回 false。
stats.isDirectory()	如果是目录返回 true，否则返回 false。
stats.isBlockDevice()	如果是块设备返回 true，否则返回 false。
stats.isCharacterDevice()	如果是字符设备返回 true，否则返回 false。
stats.isSymbolicLink()	如果是软链接返回 true，否则返回 false。
stats.isFIFO()	如果是FIFO，返回true，否则返回 false。FIFO是UNIX中的一种特殊类型的命令管道。
stats.isSocket()	如果是 Socket 返回 true，否则返回 false。

实例

接下来我们创建 file.js 文件，代码如下所示：

```
var fs = require("fs");

console.log("准备打开文件! ");
fs.stat('input.txt', function (err, stats) {
   if (err) {
       return console.error(err);
   }
   console.log(stats);
   console.log("读取文件信息成功! ");

   // 检测文件类型
   console.log("是否为文件(isFile) ? " + stats.isFile());
   console.log("是否为目录(isDirectory) ? " + stats.isDirectory());
});
```

以上代码执行结果如下：

```
$ node file.js
准备打开文件!
```

```
{ dev: 16777220,
  mode: 33188,
  nlink: 1,
  uid: 501,
  gid: 20,
  rdev: 0,
  blksize: 4096,
  ino: 40333161,
  size: 61,
  blocks: 8,
  atime: Mon Sep 07 2015 17:43:55 GMT+0800 (CST),
  mtime: Mon Sep 07 2015 17:22:35 GMT+0800 (CST),
  ctime: Mon Sep 07 2015 17:22:35 GMT+0800 (CST) }
读取文件信息成功!
是否为文件(isFile) ? true
是否为目录(isDirectory) ? false
```

写入文件

语法

以下为异步模式下写入文件的语法格式：

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

writeFile 直接打开文件默认是 `w` 模式，所以如果文件存在，该方法写入的内容会覆盖旧的文件内容。

参数

参数使用说明如下：

- **file** - 文件名或文件描述符。
- **data** - 要写入文件的数据，可以是 String(字符串) 或 Buffer(缓冲) 对象。
- **options** - 该参数是一个对象，包含 {encoding, mode, flag}。默认编码为 utf8, 模式为 0666 ， flag 为 'w'
- **callback** - 回调函数，回调函数只包含错误信息参数(err)，在写入失败时返回。

实例

接下来我们创建 file.js 文件，代码如下所示：

```
var fs = require("fs");

console.log("准备写入文件");
fs.writeFile('input.txt', '我是通 过fs.writeFile 写入文件的内容',  function(err) {
   if (err) {
```

```
        return console.error(err);
    }
    console.log("数据写入成功! ");
    console.log("--------我是分割线-------------")
    console.log("读取写入的数据! ");
    fs.readFile('input.txt', function (err, data) {
       if (err) {
          return console.error(err);
       }
       console.log("异步读取文件数据: " + data.toString());
    });
});
```

以上代码执行结果如下：

```
$ node file.js
准备写入文件
数据写入成功!
--------我是分割线-------------
读取写入的数据!
异步读取文件数据: 我是通 过fs.writeFile 写入文件的内容
```

读取文件

语法

以下为异步模式下读取文件的语法格式：

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

该方法使用了文件描述符来读取文件。

参数

参数使用说明如下：

- **fd** - 通过 fs.open() 方法返回的文件描述符。
- **buffer** - 数据写入的缓冲区。
- **offset** - 缓冲区写入的写入偏移量。
- **length** - 要从文件中读取的字节数。
- **position** - 文件读取的起始位置，如果 position 的值为 null，则会从当前文件指针的位置读取。

- **callback** - 回调函数，有三个参数err, bytesRead, buffer，err 为错误信息， bytesRead 表示读取的字节数，buffer 为缓冲区对象。

实例

input.txt 文件内容为：

```
菜鸟教程官网地址：www.runoob.com
```

接下来我们创建 file.js 文件，代码如下所示：

```
var fs = require("fs");
var buf = new Buffer.alloc(1024);

console.log("准备打开已存在的文件！");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
       return console.error(err);
   }
   console.log("文件打开成功！");
   console.log("准备读取文件：");
   fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
      if (err){
         console.log(err);
      }
      console.log(bytes + "  字节被读取");

      // 仅输出读取的字节
      if(bytes > 0){
         console.log(buf.slice(0, bytes).toString());
      }
   });
});
```

以上代码执行结果如下：

```
$ node file.js
准备打开已存在的文件！
文件打开成功！
准备读取文件：
42  字节被读取
菜鸟教程官网地址：www.runoob.com
```

关闭文件

语法

以下为异步模式下关闭文件的语法格式：

```
fs.close(fd, callback)
```

该方法使用了文件描述符来读取文件。

参数

参数使用说明如下：

- **fd** - 通过 fs.open() 方法返回的文件描述符。
- **callback** - 回调函数，没有参数。

实例

input.txt 文件内容为：

```
菜鸟教程官网地址：www.runoob.com
```

接下来我们创建 file.js 文件，代码如下所示：

```
var fs = require("fs");
var buf = new Buffer.alloc(1024);

console.log("准备打开文件！");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
       return console.error(err);
   }
   console.log("文件打开成功！");
   console.log("准备读取文件！");
   fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
      if (err){
         console.log(err);
      }

      // 仅输出读取的字节
      if(bytes > 0){
         console.log(buf.slice(0, bytes).toString());
      }

      // 关闭文件
      fs.close(fd, function(err){
         if (err){
            console.log(err);
```

```
         }
         console.log("文件关闭成功");
      });
   });
});
```

以上代码执行结果如下：

```
$ node file.js
准备打开文件!
文件打开成功!
准备读取文件!
菜鸟教程官网地址：www.runoob.com
文件关闭成功
```

截取文件

语法

以下为异步模式下截取文件的语法格式：

```
fs.ftruncate(fd, len, callback)
```

该方法使用了文件描述符来读取文件。

参数

参数使用说明如下：

- **fd** - 通过 fs.open() 方法返回的文件描述符。
- **len** - 文件内容截取的长度。
- **callback** - 回调函数，没有参数。

实例

input.txt 文件内容为：

```
site:www.runoob.com
```

接下来我们创建 file.js 文件，代码如下所示：

```
var fs = require("fs");
var buf = new Buffer.alloc(1024);
```

```
console.log("准备打开文件! ");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
       return console.error(err);
   }
   console.log("文件打开成功! ");
   console.log("截取10字节内的文件内容，超出部分将被去除。");

   // 截取文件
   fs.ftruncate(fd, 10, function(err){
      if (err){
         console.log(err);
      }
      console.log("文件截取成功。");
      console.log("读取相同的文件");
      fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
         if (err){
            console.log(err);
         }

         // 仅输出读取的字节
         if(bytes > 0){
            console.log(buf.slice(0, bytes).toString());
         }

         // 关闭文件
         fs.close(fd, function(err){
            if (err){
               console.log(err);
            }
            console.log("文件关闭成功! ");
         });
      });
   });
});
```

以上代码执行结果如下：

```
$ node file.js
准备打开文件!
文件打开成功!
截取10字节内的文件内容，超出部分将被去除。
文件截取成功。
读取相同的文件
site:www.r
文件关闭成功
```

删除文件

语法

以下为删除文件的语法格式：

```
fs.unlink(path, callback)
```

参数

参数使用说明如下：

- **path** - 文件路径。
- **callback** - 回调函数，没有参数。

实例

input.txt 文件内容为：

```
site:www.runoob.com
```

接下来我们创建 file.js 文件，代码如下所示：

```
var fs = require("fs");

console.log("准备删除文件！");
fs.unlink('input.txt', function(err) {
   if (err) {
       return console.error(err);
   }
   console.log("文件删除成功！");
});
```

以上代码执行结果如下：

```
$ node file.js
准备删除文件！
文件删除成功！
```

再去查看 input.txt 文件，发现已经不存在了。

创建目录

语法

以下为创建目录的语法格式：

```
fs.mkdir(path[, options], callback)
```

参数

参数使用说明如下：

- **path** - 文件路径。
- options 参数可以是：
  - **recursive** - 是否以递归的方式创建目录，默认为 false。
  - **mode** - 设置目录权限，默认为 0777。
- **callback** - 回调函数，没有参数。

实例

接下来我们创建 file.js 文件，代码如下所示：

```
var fs = require("fs");
// tmp 目录必须存在
console.log("创建目录 /tmp/test/");
fs.mkdir("/tmp/test/",function(err){
   if (err) {
       return console.error(err);
   }
   console.log("目录创建成功。");
});
```

以上代码执行结果如下：

```
$ node file.js
创建目录 /tmp/test/
目录创建成功。
```

可以添加 recursive: true 参数，不管创建的目录 /tmp 和 /tmp/a 是否存在：

```
fs.mkdir('/tmp/a/apple', { recursive: true }, (err) => {
  if (err) throw err;
});
```

读取目录

语法

以下为读取目录的语法格式：

```
fs.readdir(path, callback)
```

参数

参数使用说明如下：

- **path** - 文件路径。
- **callback** - 回调函数，回调函数带有两个参数err, files，err 为错误信息，files 为 目录下的文件数组列表。

实例

接下来我们创建 file.js 文件，代码如下所示：

```
var fs = require("fs");

console.log("查看 /tmp 目录");
fs.readdir("/tmp/",function(err, files){
   if (err) {
       return console.error(err);
   }
   files.forEach( function (file){
       console.log( file );
   });
});
```

以上代码执行结果如下：

```
$ node file.js
查看 /tmp 目录
input.out
output.out
test
test.txt
```

删除目录

语法

以下为删除目录的语法格式：

```
fs.rmdir(path, callback)
```

参数

参数使用说明如下：

- **path** - 文件路径。
- **callback** - 回调函数，没有参数。

实例

接下来我们创建 file.js 文件，代码如下所示：

```
var fs = require("fs");
// 执行前创建一个空的 /tmp/test 目录
console.log("准备删除目录 /tmp/test");
fs.rmdir("/tmp/test",function(err){
   if (err) {
       return console.error(err);
   }
   console.log("读取 /tmp 目录");
   fs.readdir("/tmp/",function(err, files){
      if (err) {
          return console.error(err);
      }
      files.forEach( function (file){
          console.log( file );
      });
   });
});
```

以上代码执行结果如下：

```
$ node file.js
准备删除目录 /tmp/test
读取 /tmp 目录
.....
```

文件模块方法参考手册

以下为 Node.js 文件模块相同的方法列表：

序号	方法 & 描述
1	**fs.rename(oldPath, newPath, callback)** 异步 rename().回调函数没有参数，但可能抛出异常。
2	**fs.ftruncate(fd, len, callback)** 异步 ftruncate().回调函数没有参数，但可能抛出异常。

3	**fs.ftruncateSync(fd, len)** 同步 ftruncate()
4	**fs.truncate(path, len, callback)** 异步 truncate().回调函数没有参数，但可能抛出异常。
5	**fs.truncateSync(path, len)** 同步 truncate()
6	**fs.chown(path, uid, gid, callback)** 异步 chown().回调函数没有参数，但可能抛出异常。
7	**fs.chownSync(path, uid, gid)** 同步 chown()
8	**fs.fchown(fd, uid, gid, callback)** 异步 fchown().回调函数没有参数，但可能抛出异常。
9	**fs.fchownSync(fd, uid, gid)** 同步 fchown()
10	**fs.lchown(path, uid, gid, callback)** 异步 lchown().回调函数没有参数，但可能抛出异常。
11	**fs.lchownSync(path, uid, gid)** 同步 lchown()
12	**fs.chmod(path, mode, callback)** 异步 chmod().回调函数没有参数，但可能抛出异常。
13	**fs.chmodSync(path, mode)** 同步 chmod().
14	**fs.fchmod(fd, mode, callback)** 异步 fchmod().回调函数没有参数，但可能抛出异常。
15	**fs.fchmodSync(fd, mode)** 同步 fchmod().
16	**fs.lchmod(path, mode, callback)** 异步 lchmod().回调函数没有参数，但可能抛出异常。Only available on Mac OS X.
17	**fs.lchmodSync(path, mode)**

	同步 lchmod().
18	**fs.stat(path, callback)** 异步 stat(). 回调函数有两个参数 err, stats，stats 是 fs.Stats 对象。
19	**fs.lstat(path, callback)** 异步 lstat(). 回调函数有两个参数 err, stats，stats 是 fs.Stats 对象。
20	**fs.fstat(fd, callback)** 异步 fstat(). 回调函数有两个参数 err, stats，stats 是 fs.Stats 对象。
21	**fs.statSync(path)** 同步 stat(). 返回 fs.Stats 的实例。
22	**fs.lstatSync(path)** 同步 lstat(). 返回 fs.Stats 的实例。
23	**fs.fstatSync(fd)** 同步 fstat(). 返回 fs.Stats 的实例。
24	**fs.link(srcpath, dstpath, callback)** 异步 link().回调函数没有参数，但可能抛出异常。
25	**fs.linkSync(srcpath, dstpath)** 同步 link().
26	**fs.symlink(srcpath, dstpath[, type], callback)** 异步 symlink().回调函数没有参数，但可能抛出异常。 type 参数可以设置为 'dir', 'file', 或 'junction' (默认为 'file') 。
27	**fs.symlinkSync(srcpath, dstpath[, type])** 同步 symlink().
28	**fs.readlink(path, callback)** 异步 readlink(). 回调函数有两个参数 err, linkString。
29	**fs.realpath(path[, cache], callback)** 异步 realpath(). 回调函数有两个参数 err, resolvedPath。
30	**fs.realpathSync(path[, cache])** 同步 realpath()。返回绝对路径。
31	**fs.unlink(path, callback)** 异步 unlink().回调函数没有参数，但可能抛出异常。

32	**fs.unlinkSync(path)** 同步 unlink().
33	**fs.rmdir(path, callback)** 异步 rmdir().回调函数没有参数，但可能抛出异常。
34	**fs.rmdirSync(path)** 同步 rmdir().
35	**fs.mkdir(path[, mode], callback)** S异步 mkdir(2).回调函数没有参数，但可能抛出异常。 访问权限默认为 0777。
36	**fs.mkdirSync(path[, mode])** 同步 mkdir().
37	**fs.readdir(path, callback)** 异步 readdir(3). 读取目录的内容。
38	**fs.readdirSync(path)** 同步 readdir().返回文件数组列表。
39	**fs.close(fd, callback)** 异步 close().回调函数没有参数，但可能抛出异常。
40	**fs.closeSync(fd)** 同步 close().
41	**fs.open(path, flags[, mode], callback)** 异步打开文件。
42	**fs.openSync(path, flags[, mode])** 同步 version of fs.open().
43	**fs.utimes(path, atime, mtime, callback)**
44	**fs.utimesSync(path, atime, mtime)** 修改文件时间戳，文件通过指定的文件路径。
45	**fs.futimes(fd, atime, mtime, callback)**
46	**fs.futimesSync(fd, atime, mtime)**

	修改文件时间戳，通过文件描述符指定。
47	**fs.fsync(fd, callback)** 异步 fsync.回调函数没有参数，但可能抛出异常。
48	**fs.fsyncSync(fd)** 同步 fsync.
49	**fs.write(fd, buffer, offset, length[, position], callback)** 将缓冲区内容写入到通过文件描述符指定的文件。
50	**fs.write(fd, data[, position[, encoding]], callback)** 通过文件描述符 fd 写入文件内容。
51	**fs.writeSync(fd, buffer, offset, length[, position])** 同步版的 fs.write()。
52	**fs.writeSync(fd, data[, position[, encoding]])** 同步版的 fs.write().
53	**fs.read(fd, buffer, offset, length, position, callback)** 通过文件描述符 fd 读取文件内容。
54	**fs.readSync(fd, buffer, offset, length, position)** 同步版的 fs.read.
55	**fs.readFile(filename[, options], callback)** 异步读取文件内容。
56	**fs.readFileSync(filename[, options])**
57	**fs.writeFile(filename, data[, options], callback)** 异步写入文件内容。
58	**fs.writeFileSync(filename, data[, options])** 同步版的 fs.writeFile。
59	**fs.appendFile(filename, data[, options], callback)** 异步追加文件内容。
60	**fs.appendFileSync(filename, data[, options])** The 同步 version of fs.appendFile.
61	**fs.watchFile(filename[, options], listener)**

	查看文件的修改。
62	**fs.unwatchFile(filename[, listener])** 停止查看 filename 的修改。
63	**fs.watch(filename[, options][, listener])** 查看 filename 的修改，filename 可以是文件或目录。返回 fs.FSWatcher 对象。
64	**fs.exists(path, callback)** 检测给定的路径是否存在。
65	**fs.existsSync(path)** 同步版的 fs.exists.
66	**fs.access(path[, mode], callback)** 测试指定路径用户权限。
67	**fs.accessSync(path[, mode])** 同步版的 fs.access。
68	**fs.createReadStream(path[, options])** 返回ReadStream 对象。
69	**fs.createWriteStream(path[, options])** 返回 WriteStream 对象。
70	**fs.symlink(srcpath, dstpath[, type], callback)** 异步 symlink().回调函数没有参数，但可能抛出异常。

更多内容，请查看官网文件模块描述：File System。

← Node.js 常用工具 Node.js GET/POST请求 →