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**

**实例**

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。");

**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。 |
| w | 以写入模式打开文件，如果文件不存在则创建。 |
| a | 以追加模式打开文件，如果文件不存在则创建。 |

**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！");

});

**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。

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**

var fs = require('fs');

fs.stat('/Users/liuht/code/itbilu/demo/fs.js', function (err, stats) {

console.log(stats.isFile());         //true

})

**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)

**参数**

参数使用说明如下：

* **file** - 文件名或通过fs.open() 方法返回文件描述符。
* **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！");

});

**二进制写入**

fs.writeFile(`images/${index}.jpg`, bytes, 'binary', function (err) {

    if (err) {

        console.log(`${index}.jpg 保存出错！`)

    }

    else {

        console.log(`${index}.jpg 保存成功！`)

    }

})

**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**

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！");

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());

}

});

});

**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)

**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**

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");

});

});

});

**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**

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！");

});

**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。");

});

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 );

});

});

**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 );

});

});

});