File

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)。 建议大家是用异步方法，比起同步，异步方法性能更高，速度更快，而且没有阻塞

文件的操作

fs.open

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

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。
```

fs.readFile

```
fs.readFile(file[, options], callback)
```

fs.writeFile

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

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：
path - 文件路径。
data - 要写入文件的数据，可以是 String(字符串) 或 Buffer(流) 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。
```

read

```
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。
```

write

```
fs.write(fd, data[, position[, encoding]], callback)
```

unlink

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

watchFile--每次文件访问的时候都会触发

```
fs.watchFile('message.text', (curr, prev) => {
  console.log(`the current mtime is: ${curr.mtime}`);
  console.log(`the previous mtime was: ${prev.mtime}`);
});

curr,prev 是fs.Stat的实例
```

watch

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

fs.watch('somedir', (eventType, filename) => {
      console.log(`event type is: ${eventType}`);
      if (filename) {
        console.log(`filename provided: ${filename}`);
      } else {
        console.log('filename not provided');
      }
  });

  eventType: 'rename'或者'change'
  filename: 触发事件的文件名称.
```

观测文件和文件夹的变动。

access

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

测试用户对文件和文件夹的权限。mode可以指定检查的权限。
fs.constants.F_OK - path is visible to the calling process. This is useful
for determining if a file exists, but says nothing about rwx permissions.
Default if no mode is specified.

fs.constants.R_OK - path can be read by the calling process.

fs.constants.W_OK - path can be written by the calling process.

fs.constants.X_OK - path can be executed by the calling process. This has no
effect on Windows (will behave like fs.constants.F_OK).

fs.access('/etc/passwd', fs.constants.R_OK | fs.constants.W_OK, (err) => {
  console.log(err ? 'no access!' : 'can read/write');
});
```

目录操作

readdir

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

mkdir

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

rmdir

rename

```
fs.rename(oldPath, newPath, callback)
```

fs.Stats

由fs.stat()返回

- isFile
- isDirectory

```
{
  dev: 2114,
  ino: 48064969,
  mode: 33188,
  nlink: 1,
  uid: 85,
  gid: 100,
  rdev: 0,
  size: 527,
  blksize: 4096,
  blocks: 8,
  atime: access Time,
  mtime: Modified Time,
  ctime: Change Time,
  birthtime: birthTime
}
```