Node.js v6.10.0 文档

返回文档首页

目录

全局变量

查看英文版 / 参与翻译

这些对象在所有模块中都是可用的。 有些对象实际上不在全局作用域内，而是在模块作用域内，这个在文档中会注明。

以下列出的对象是特定于 Node.js 的。 有些内置对象是 JavaScript 语言本身的一部分，它们也可以全局访问。

Buffer 类

查看英文版 / 参与翻译

新增于: v0.1.103

- `<Function>`

用于处理二进制数据。详见 buffer 章节。

__dirname

查看英文版 / 参与翻译

新增于: v0.1.27

- `<String>`

当前模块的目录名。 等同于 `__filename` 的 `path.dirname()`。

`__dirname` 实际上不是一个全局变量，而是每个模块内部的。

例子，从 `/Users/mjr` 运行 `node example.js`：

```
console.log(__dirname);
// 输出: /Users/mjr
console.log(path.dirname(__filename));
// 输出: /Users/mjr
```

__filename

查看英文版 / 参与翻译

新增于: v0.0.1

- `<String>`

当前模块的文件名。 这是当前模块文件的解析后的绝对路径。

对于主程序而言，无需与在命令行中使用的文件名相同。

查看 `__dirname` 了解当前模块的目录名。

`__filename` 实际上不是一个全局变量，而是每个模块内部的。

例子：

从 `/Users/mjr` 运行 `node example.js`：

```
console.log(__filename);
// 输出: /Users/mjr/example.js
console.log(__dirname);
// 输出: /Users/mjr
```

给出两个模块：`a` 和 `b`，其中 `b` 是 `a` 的一个依赖，目录结构如下：

- `/Users/mjr/app/a.js`
- `/Users/mjr/app/node_modules/b/b.js`

`b.js` 的 `__filename` 返回 `/Users/mjr/app/node_modules/b/b.js`，`a.js` 的 `__filename` 返回 `/Users/mjr/app/a.js`。

clearImmediate(immediateObject)

查看英文版 / 参与翻译

新增于: v0.9.1

`clearImmediate` 描述在定时器章节。

clearInterval(intervalObject)

查看英文版 / 参与翻译

新增于: v0.0.1

`clearInterval` 描述在定时器章节。

clearTimeout(timeoutObject)

查看英文版 / 参与翻译

新增于: v0.0.1

`clearTimeout` 描述在定时器章节。

console

查看英文版 / 参与翻译

新增于: v0.1.100

- `<Object>`

用于打印 `stdout` 和 `stderr`。 详见 `console` 章节。

exports

查看英文版 / 参与翻译

新增于: v0.1.12

`module.exports` 的一个简短的引用。 何时使用 `exports` 与何时使用 `module.exports` 详见模块系统文档。

`exports` 实际上不是一个全局变量，而是每个模块内部的。

详见模块系统文档。

global

查看英文版 / 参与翻译

新增于: v0.1.27

- `<Object>` 全局的命名空间对象。

在浏览器中，顶层作用域就是全局作用域。 这意味着在浏览器中，如果在全局作用域内使用 `var something` 会定义一个全局变量。 在 Node.js 中则不同。 顶层作用域不是全局作用域；在 Node.js 模块中使用 `var something` 会是模块内部的。

module

查看英文版 / 参与翻译

新增于: v0.1.16

- `<Object>`

当前模块的引用。 具体地说，`module.exports` 用于定义一个模块导出什么，且通过 `require()` 引入。

`module` 实际上不是一个全局变量，而是每个模块内部的。

详见模块系统文档。

process

查看英文版 / 参与翻译

新增于: v0.1.7

- `<Object>`

进程对象。详见 `process` 对象章节。

require()

查看英文版 / 参与翻译

新增于: v0.1.13

- `<Function>`

用于引入模块。详见模块章节。 `require` 实际上不是一个全部变量，而是每个模块内部的。

require.cache

查看英文版 / 参与翻译

新增于: v0.3.0

- `<Object>`

当模块被引入时，它们会缓存到这个对象。 通过从该对象删除键值，下次调用 `require` 时会重新加载模块。 注意，这不适用于原生插件，因为重新加载会导致错误。

require.extensions

查看英文版 / 参与翻译

新增于: v0.3.0　废弃于: v0.10.6

稳定性: 0 - 废弃的

- `<Object>`

Instruct `require` on how to handle certain file extensions.

Process files with the extension `.sjs` as `.js`:

```
require.extensions['.sjs'] = require.extensions['.js'];
```

Deprecated In the past, this list has been used to load non-JavaScript modules into Node.js by compiling them on-demand. However, in practice, there are much better ways to do this, such as loading modules via some other Node.js program, or compiling them to JavaScript ahead of time.

Since the Module system is locked, this feature will probably never go away. However, it may have subtle bugs and complexities that are best left untouched.

require.resolve()

查看英文版 / 参与翻译

新增于: v0.3.0

使用内部的 `require()` 机制来查找模块的位置，但不会加载模块，只返回解析后的文件名。

setImmediate(callback[, ...args])

查看英文版 / 参与翻译

新增于: v0.9.1

`setImmediate` 描述在定时器章节。

setInterval(callback, delay[, ...args])

查看英文版 / 参与翻译

新增于: v0.0.1

`setInterval` 描述在定时器章节。

setTimeout(callback, delay[, ...args])

查看英文版 / 参与翻译

新增于: v0.0.1

`setTimeout` 描述在定时器章节。