Node.js v6.10.0 文档

返回文档首页

目录

debugger (调试器)

查看英文版 / 参与翻译

稳定性: 2 - 稳定的

Node.js 包含一个进程外的调试工具，可以通过基于 TCP 的协议和内置调试客户端访问。 要使用它，可以带上 `debug` 参数启动 Node.js，并带上需要调试的脚本的路径；然后会显示一个提示，表明成功启动调试器：

```
$ node debug myscript.js
< debugger listening on port 5858
connecting... ok
break in /home/indutny/Code/git/indutny/myscript.js:1
  1 x = 5;
  2 setTimeout(() => {
  3   debugger;
debug>
```

Node.js 的调试器客户端还未支持全部特性，但可以做些简单的步骤和检测。

在脚本的源代码中插入 `debugger;` 语句，则会在代码的那个位置启用一个断点：

```
// myscript.js
x = 5;
setTimeout(() => {
  debugger;
  console.log('world');
}, 1000);
console.log('hello');
```

一旦运行调试器，则会在第 4 行出现一个断点：

```
$ node debug myscript.js
< debugger listening on port 5858
connecting... ok
break in /home/indutny/Code/git/indutny/myscript.js:1
  1 x = 5;
  2 setTimeout(() => {
  3   debugger;
debug> cont
< hello
break in /home/indutny/Code/git/indutny/myscript.js:3
  1 x = 5;
  2 setTimeout(() => {
  3   debugger;
  4   console.log('world');
  5 }, 1000);
debug> next
break in /home/indutny/Code/git/indutny/myscript.js:4
  2 setTimeout(() => {
  3   debugger;
  4   console.log('world');
  5 }, 1000);
  6 console.log('hello');
debug> repl
Press Ctrl + C to leave debug repl
> x
5
> 2+2
4
debug> next
< world
break in /home/indutny/Code/git/indutny/myscript.js:5
  3   debugger;
  4   console.log('world');
  5 }, 1000);
  6 console.log('hello');
  7
debug> quit
```

`repl` 命令用于运行代码。 `next` 命令用于步入下一行。 输入 `help` 可查看其他可用的命令。

按下 `enter` 键且不输入命令，可重复上一个调试命令。

监视器

查看英文版 / 参与翻译

可以在调试时监视表达式和变量的值。 在每个断点上，监视器列表中的每个表达式都会在当前上下文中被执行，并在断点的源代码列表之前立即显示。

输入 `watch('my_expression')` 开始监视一个表达式。 `watchers` 命令会打印已激活的监视器。 输入 `unwatch('my_expression')` 来移除一个监视器。

命令参考手册

步进

查看英文版 / 参与翻译

- `cont`, `c` - 继续执行
- `next`, `n` - 下一步
- `step`, `s` - 跳进函数
- `out`, `o` - 跳出函数
- `pause` - 暂停运行代码（类似开发者工具中的暂停按钮）

断点

查看英文版 / 参与翻译

- `setBreakpoint()`, `sb()` - 在当前行设置断点
- `setBreakpoint(line)`, `sb(line)` - 在指定行设置断点
- `setBreakpoint('fn()')`, `sb(...)` - 在函数体的第一条语句设置断点
- `setBreakpoint('script.js', 1)`, `sb(...)` - 在 script.js 的第 1 行设置断点
- `clearBreakpoint('script.js', 1)`, `cb(...)` - 清除 script.js 第 1 行的断点

也可以在一个还未被加载的文件（模块）中设置断点：

```
$ node debug test/fixtures/break-in-module/main.js
< debugger listening on port 5858
connecting to port 5858... ok
break in test/fixtures/break-in-module/main.js:1
  1 var mod = require('./mod.js');
  2 mod.hello();
  3 mod.hello();
debug> setBreakpoint('mod.js', 23)
Warning: script 'mod.js' was not loaded yet.
  1 var mod = require('./mod.js');
  2 mod.hello();
  3 mod.hello();
debug> c
break in test/fixtures/break-in-module/mod.js:23
 21
 22 exports.hello = () => {
 23   return 'hello from module';
 24 };
 25
debug>
```

信息

查看英文版 / 参与翻译

- `backtrace`, `bt` - 打印当前执行框架的回溯
- `list(5)` - 列出脚本源代码的 5 行上下文（前后各 5 行）
- `watch(expr)` - 添加表达式到监视列表
- `unwatch(expr)` - 从监视列表移除表达式
- `watchers` - 列出所有监视器和它们的值（每个断点会自动列出）
- `repl` - 打开调试器的 repl，用于在所调试的脚本的上下文中进行执行
- `exec expr` - 在所调试的脚本的上下文中执行一个表达式

执行控制

查看英文版 / 参与翻译

- `run` - 运行脚本（调试器开始时自动运行）
- `restart` - 重新启动脚本
- `kill` - 终止脚本

杂项

查看英文版 / 参与翻译

- `scripts` - 列出所有已加载的脚本
- `version` - 显示 V8 引擎的版本号

高级用法

查看英文版 / 参与翻译

启用和访问调试器的另一种方式，是启动 Node.js 时带上 `--debug` 命令行标志，或向一个已存在的 Node.js 进程发送 `SIGUSR1` 信号。

一旦一个进程以这种方式被设为调试模式，它就可以被 Node.js 调试器使用，通过连接到正在运行的进程的 `pid` 或通过正在监听的调试器的 URI 引用：

- `node debug -p <pid>` - 通过 `pid` 连接进程
- `node debug <URI>` - 通过 URI 连接进程，如 localhost:5858

Node.js 的 V8 检查器集合

查看英文版 / 参与翻译

注意：这是一个试验的特性。

V8 的检查器集成可以附加 Chrome 的开发者工具到 Node.js 实例以用于调试和性能分析。

当启动一个 Node.js 应用时，V8 检查器可以通过传入 `--inspect` 标志启用。 也可以通过该标志提供一个自定义的端口，如 `--inspect=9222` 会在 9222 端口接受开发者工具连接。

要想在应用代码的第一行断开，可以在除 `--inspect` 之外再提供 `--debug-brk` 标志。

```
$ node --inspect index.js
Debugger listening on port 9229.
Warning: This is an experimental feature and could change
To start debugging, open the following URL in Chrome:
    chrome-devtools://devtools/remote/serve_file/@60cd6e85
```