Debugger

> *Stability: 2 - Stable*

Node.js includes a command-line debugging utility. The Node.js debugger client is not a full-featured debugger, but simple stepping and inspection are possible.

To use it, start Node.js with the `inspect` argument followed by the path to the script to debug.

```
$ node inspect myscript.js
< Debugger listening on ws://127.0.0.1:9229/621111f9-ffcb-4e82-b718-48a145fa5db8
< For help, see: https://nodejs.org/en/docs/inspector
<
< Debugger attached.
<
 ok
Break on start in myscript.js:2
  1 // myscript.js
> 2 global.x = 5;
  3 setTimeout(() => {
  4   debugger;
debug>
```

The debugger automatically breaks on the first executable line. To instead run until the first breakpoint (specified by a `debugger` statement), set the `NODE_INSPECT_RESUME_ON_START` environment variable to `1`.

```
$ cat myscript.js
// myscript.js
global.x = 5;
setTimeout(() => {
  debugger;
  console.log('world');
}, 1000);
console.log('hello');
$ NODE_INSPECT_RESUME_ON_START=1 node inspect myscript.js
< Debugger listening on ws://127.0.0.1:9229/f1ed133e-7876-495b-83ae-c32c6fc319c2
< For help, see: https://nodejs.org/en/docs/inspector
<
connecting to 127.0.0.1:9229 ... ok
< Debugger attached.
<
< hello
<
break in myscript.js:4
  2 global.x = 5;
  3 setTimeout(() => {
> 4   debugger;
  5   console.log('world');
  6 }, 1000);
debug> next
break in myscript.js:5
```

```
  3 setTimeout(() => {
  4   debugger;
> 5   console.log('world');
  6 }, 1000);
  7 console.log('hello');
debug> repl
Press Ctrl+C to leave debug repl
> x
5
> 2 + 2
4
debug> next
< world
<
break in myscript.js:6
  4   debugger;
  5   console.log('world');
> 6 }, 1000);
  7 console.log('hello');
  8
debug> .exit
$
```

The `repl` command allows code to be evaluated remotely. The `next` command steps to the next line. Type `help` to see what other commands are available.

Pressing `enter` without typing a command will repeat the previous debugger command.

Watchers

It is possible to watch expression and variable values while debugging. On every breakpoint, each expression from the watchers list will be evaluated in the current context and displayed immediately before the breakpoint's source code listing.

To begin watching an expression, type `watch('my_expression')`. The command `watchers` will print the active watchers. To remove a watcher, type `unwatch('my_expression')`.

Command reference

Stepping

- `cont`, `c`: Continue execution
- `next`, `n`: Step next
- `step`, `s`: Step in
- `out`, `o`: Step out
- `pause`: Pause running code (like pause button in Developer Tools)

Breakpoints

- `setBreakpoint()`, `sb()`: Set breakpoint on current line
- `setBreakpoint(line)`, `sb(line)`: Set breakpoint on specific line
- `setBreakpoint('fn()')`, `sb(...)`: Set breakpoint on a first statement in function's body

- `setBreakpoint('script.js', 1)`, `sb(...)`: Set breakpoint on first line of `script.js`
- `setBreakpoint('script.js', 1, 'num < 4')`, `sb(...)`: Set conditional breakpoint on first line of `script.js` that only breaks when `num < 4` evaluates to `true`
- `clearBreakpoint('script.js', 1)`, `cb(...)`: Clear breakpoint in `script.js` on line 1

It is also possible to set a breakpoint in a file (module) that is not loaded yet:

```
$ node inspect main.js
< Debugger listening on ws://127.0.0.1:9229/48a5b28a-550c-471b-b5e1-d13dd7165df9
< For help, see: https://nodejs.org/en/docs/inspector
<
< Debugger attached.
<
 ok
Break on start in main.js:1
> 1 const mod = require('./mod.js');
  2 mod.hello();
  3 mod.hello();
debug> setBreakpoint('mod.js', 22)
Warning: script 'mod.js' was not loaded yet.
debug> c
break in mod.js:22
 20 // USE OR OTHER DEALINGS IN THE SOFTWARE.
 21
>22 exports.hello = function() {
 23   return 'hello from module';
 24 };
debug>
```

It is also possible to set a conditional breakpoint that only breaks when a given expression evaluates to `true`:

```
$ node inspect main.js
< Debugger listening on ws://127.0.0.1:9229/ce24daa8-3816-44d4-b8ab-8273c8a66d35
< For help, see: https://nodejs.org/en/docs/inspector
< Debugger attached.
Break on start in main.js:7
  5 }
  6
> 7 addOne(10);
  8 addOne(-1);
  9
debug> setBreakpoint('main.js', 4, 'num < 0')
  1 'use strict';
  2
  3 function addOne(num) {
> 4   return num + 1;
  5 }
  6
  7 addOne(10);
  8 addOne(-1);
  9
```

```
debug> cont
break in main.js:4
  2
  3 function addOne(num) {
> 4   return num + 1;
  5 }
  6
debug> exec('num')
-1
debug>
```

Information

- `backtrace`, `bt`: Print backtrace of current execution frame
- `list(5)`: List scripts source code with 5 line context (5 lines before and after)
- `watch(expr)`: Add expression to watch list
- `unwatch(expr)`: Remove expression from watch list
- `watchers`: List all watchers and their values (automatically listed on each breakpoint)
- `repl`: Open debugger's repl for evaluation in debugging script's context
- `exec expr`, `p expr`: Execute an expression in debugging script's context and print its value

Execution control

- `run`: Run script (automatically runs on debugger's start)
- `restart`: Restart script
- `kill`: Kill script

Various

- `scripts`: List all loaded scripts
- `version`: Display V8's version

Advanced usage

V8 inspector integration for Node.js

V8 Inspector integration allows attaching Chrome DevTools to Node.js instances for debugging and profiling. It uses the Chrome DevTools Protocol.

V8 Inspector can be enabled by passing the `--inspect` flag when starting a Node.js application. It is also possible to supply a custom port with that flag, e.g. `--inspect=9222` will accept DevTools connections on port 9222.

To break on the first line of the application code, pass the `--inspect-brk` flag instead of `--inspect`.

```
$ node --inspect index.js
Debugger listening on ws://127.0.0.1:9229/dc9010dd-f8b8-4ac5-a510-c1a114ec7d29
For help, see: https://nodejs.org/en/docs/inspector
```

(In the example above, the UUID dc9010dd-f8b8-4ac5-a510-c1a114ec7d29 at the end of the URL is generated on the fly, it varies in different debugging sessions.)

If the Chrome browser is older than 66.0.3345.0, use `inspector.html` instead of `js_app.html` in the above URL.

Chrome DevTools doesn't support debugging worker threads yet. ndb can be used to debug them.