Debugging Guide

Edit on GitHub

This guide will help you get started debugging your Node.js apps and scripts.

Enable Inspector

When started with the `--inspect` switch, a Node.js process listens for a debugging client. By default, it will listen at host and port 127.0.0.1:9229. Each process is also assigned a unique UUID.

Inspector clients must know and specify host address, port, and UUID to connect. A full URL will look something like `ws://127.0.0.1:9229/0f2c936f-b1cd-4ac9-aab3-f63b0f33d55e`.

Node.js will also start listening for debugging messages if it receives a `SIGUSR1` signal. (`SIGUSR1` is not available on Windows.) In Node.js 7 and earlier, this activates the legacy Debugger API. In Node.js 8 and later, it will activate the Inspector API.

Security Implications

Since the debugger has full access to the Node.js execution environment, a malicious actor able to connect to this port may be able to execute arbitrary code on behalf of the Node.js process. It is important to understand the security implications of exposing the debugger port on public and private networks.

Exposing the debug port publicly is unsafe

If the debugger is bound to a public IP address, or to 0.0.0.0, any clients that can reach your IP address will be able to connect to the debugger without any restriction and will be able to run arbitrary code.

By default `node --inspect` binds to 127.0.0.1. You explicitly need to provide a public IP address or 0.0.0.0, etc., if you intend to allow external connections to the debugger. Doing so may expose you to a potentially significant security threat. We suggest you ensure appropriate firewalls and access controls in place to prevent a security exposure.

See the section on 'Enabling remote debugging scenarios' on some advice on how to safely allow remote debugger clients to connect.

Local applications have full access to the inspector

Even if you bind the inspector port to 127.0.0.1 (the default), any applications running locally on your machine will have unrestricted access. This is by design to allow local debuggers to be able to attach conveniently.

Browsers, WebSockets and same-origin policy

Websites open in a web-browser can make WebSocket and HTTP requests under the browser security model.

An initial HTTP connection is necessary to obtain a unique debugger session id. The same-origin-policy prevents websites from being able to make this HTTP connection. For additional security against DNS

rebinding attacks, Node.js verifies that the 'Host' headers for the connection either specify an IP address or `localhost` or `localhost6` precisely.

These security policies disallow connecting to a remote debug server by specifying the hostname. You can work-around this restriction by specifying either the IP address or by using ssh tunnels as described below.

Inspector Clients

Several commercial and open source tools can connect to the Node.js Inspector. Basic info on these follows:

node-inspect

- CLI Debugger supported by the Node.js Foundation which uses the Inspector Protocol.
- A version is bundled with Node.js and can be used with `node inspect myscript.js`.
- The latest version can also be installed independently (e.g. `npm install -g node-inspect`) and used with `node-inspect myscript.js`.

Chrome DevTools 55+, Microsoft Edge

- **Option 1**: Open `chrome://inspect` in a Chromium-based browser or `edge://inspect` in

Edge. Click the Configure button and ensure your target host and port are listed.

- **Option 2**: Copy the `devtoolsFrontendUrl` from the output of `/json/list` (see above) or the --inspect hint text and paste into Chrome.

Visual Studio Code 1.10+

- In the Debug panel, click the settings icon to open `.vscode/launch.json`. Select "Node.js" for initial setup.

Visual Studio 2017

- Choose "Debug > Start Debugging" from the menu or hit F5.
- Detailed instructions.

JetBrains WebStorm 2017.1+ and other JetBrains IDEs

- Create a new Node.js debug configuration and hit Debug. `--inspect` will be used by default for Node.js 7+. To disable uncheck `js.debugger.node.use.inspect` in the IDE Registry.

chrome-remote-interface

- Library to ease connections to Inspector Protocol endpoints.

Gitpod

- Start a Node.js debug configuration from the

`Debug` view or hit `F5`. Detailed instructions

Eclipse IDE with Eclipse Wild Web Developer extension

- From a .js file, choose "Debug As... > Node program", or
- Create a Debug Configuration to attach debugger to running Node.js application (already started with `--inspect`).

Command-line options

The following table lists the impact of various runtime flags on debugging:

Flag	Meaning
--inspect	• Enable inspector agent • Listen on default address and port (127.0.0.1:9229)
--inspect=*[host:port]*	• Enable inspector agent • Bind to address or hostname *host* (default: 127.0.0.1) • Listen on port *port* (default: 9229)
--inspect-brk	• Enable inspector agent • Listen on default address and port (127.0.0.1:9229) • Break before user code starts
--inspect-brk=	• Enable inspector agent

[host:port]	• Bind to address or hostname *host* (default: 127.0.0.1) • Listen on port *port* (default: 9229) • Break before user code starts
`node inspect` *`script.js`*	• Spawn child process to run user's script under --inspect flag; and use main process to run CLI debugger.
`node inspect --port=xxxx` *`script.js`*	• Spawn child process to run user's script under --inspect flag; and use main process to run CLI debugger. • Listen on port *port* (default: 9229)

Enabling remote debugging scenarios

We recommend that you never have the debugger listen on a public IP address. If you need to allow remote debugging connections we recommend the use of ssh tunnels instead. We provide the following example for illustrative purposes only. Please understand the security risk of allowing remote access to a privileged service before proceeding.

Let's say you are running Node.js on a remote machine, remote.example.com, that you want to be able to debug. On that machine, you should start the node process with the inspector listening only to localhost (the default).

```
node --inspect server.js
```

Now, on your local machine from where you want to initiate a debug client connection, you can setup an ssh tunnel:

```
ssh -L 9221:localhost:9229 user@remote.example.com
```

This starts a ssh tunnel session where a connection to port 9221 on your local machine will be forwarded to port 9229 on remote.example.com. You can now attach a debugger such as Chrome DevTools or Visual Studio Code to localhost:9221, which should be able to debug as if the Node.js application was running locally.

Legacy Debugger

The legacy debugger has been deprecated as of Node.js 7.7.0. Please use `--inspect` and Inspector instead.

When started with the **--debug** or **--debug-brk** switches in version 7 and earlier, Node.js listens for debugging commands defined by the discontinued V8 Debugging Protocol on a TCP port, by default `5858`. Any debugger client which speaks this protocol can connect to and debug the running process; a couple popular ones are listed below.

The V8 Debugging Protocol is no longer maintained or documented.

Built-in Debugger

Start `node debug script_name.js` to start your script under the builtin command-line debugger. Your script

starts in another Node.js process started with the `--debug-brk` option, and the initial Node.js process runs the `_debugger.js` script and connects to your target.

node-inspector

Debug your Node.js app with Chrome DevTools by using an intermediary process which translates the Inspector Protocol used in Chromium to the V8 Debugger protocol used in Node.js.

Report Node.js issue | Report website issue | Get Help

Thank you **username** for being a Node.js contributor **0 contributions**