Debugging Node.js Core

2024, OSSCA, Node.js - Code and Learn

Daeyeon Jeong

Debugging

References

- https://nodejs.org/en/learn/getting-started/debugging
- https://nodejs.org/docs/latest/api/cli.html#--inspecthostport
- https://code.visualstudio.com/docs/nodejs/nodejs-debugging
- https://v8.dev/docs/inspector

Debugging JS

Debugging Procedures

- Step 1. Node.js를 V8 Inspector Server로서 실행
- Step 2. Chrome DevTools과 같은 Inspector Protocol을 따르는 Client를 이용해 상기 Server에 접속

Recommand configuration

- `--node-builtin-modules-path` 빌드 옵션 사용 추천
  - 빠른 수정, breakpoint 미동작이나 정확한 code mapping을 위함

Debugging JS

Inspector - Server

```
$ ./node --inspect-brk example.js

Debugger listening on ws://127.0.0.1:9229/0f2c936f-b1cd-4ac9-aab3-f63b0f33d55e
For help, see: https://nodejs.org/en/docs/inspector

# inspector client가 접속하면 아래 메세지가 출력됨.
# Debugger attached.
```

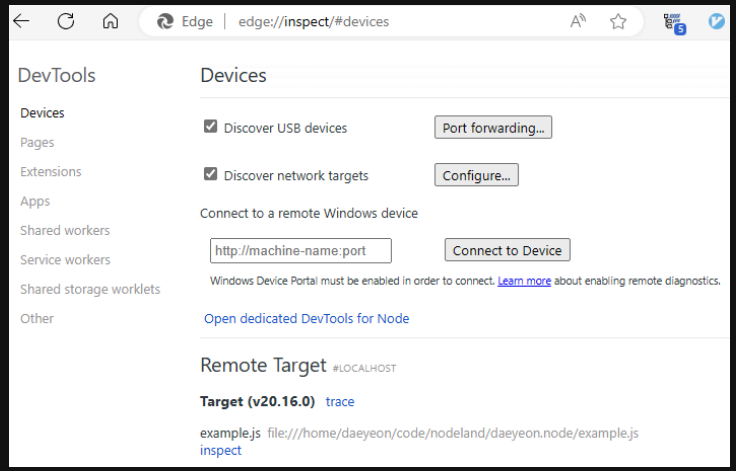
- `--inspect-brk` : Inspector client 접속 대기 상태가 됨. 연결되면 `example.js` 첫 라인에서 break
  - `--inspect-brk-node` : Bootstrap (`lib/internal/bootstrap/node.js`)부터 디버깅시 사용
- `--inspect-wait` : 첫 번째 라인이 아닌 원하는 위치(debugger;)에서 멈추고 싶을 때 사용

Debugging JS

Inspector - Client

Option 1. Chrome DevTools 이용

- Chrome DevTools 지원 브라우저 주소창에 `chrome://inspect` 입력
- Remote Target에서 연결할 Target 확인
- `Open dedicated DevTools for Node`를 선택하여 Target 연결, 또는 `inspect` 선택



Debugging JS

Inspector - Client

Option 2. VSCode (Attach) - 이미 실행 중인 Node.js 프로세스에 Inspector 연결

```
// .vscode/launch.json
{
  "configurations": [
    ...,
    {
      "type": "node",
      "request": "attach",
      "name": "Attach (Node)",
      "port": 9229,
      "skipFiles": [
        // "<node_internals>/**"
      ],
    }
  ]
}
```

- https://code.visualstudio.com/docs/nodejs/nodejs-debugging

Debugging JS

Inspector - Client

Option 2. VSCode (Launch) - 새 Node.js 프로세스를 실행하며 Inspector 연결

```
// .vscode/launch.json
{
  "configurations": [
    ...,
    {
      "type": "node",
      "request": "launch",
      "name": "Node (Inspector)",
      "skipFiles": [ // "<node_internals>/**" ],
      "cwd": "${workspaceFolder}",
      "runtimeExecutable": "${workspaceFolder}/out/Release/node",
      "program": "${file}",
      // "program": "${workspaceFolder}/example.js",
      ...
    },
  ]
}
```

Debugging JS

Tips

`'NODE_DEBUG'` 환경변수 : JS 모듈의 Debug Log 출력 (release 빌드도 사용 가능)

```
$ NODE_DEBUG=http,net node test/parallel/test-http-wget.js

NET 23811: setupListenHandle null 0 4 0 undefined
NET 23811: setupListenHandle: create a handle
NET 23811: bind to ::
NET 23811: createConnection [ { port: 35931 }, null, [Symbol(normalizedArgs)]: true ]
....
HTTP 23811: server socket close
HTTP/1.1 200 OK
Content-Type: text/plain
Date: Thu, 01 Aug 2024 11:52:50 GMT
Connection: close
```

```
// Searching for available categories: `debuglog` keyword in lib/**/*.js
// lib/net.js
let debug = require('internal/util/debuglog').debuglog('net', (fn) => {
  debug = fn;
});
```

Debugging JS

Tips

- `process._rawDebug()` : stream 추상화를 통하지 않고 stderr로 출력. (stream/console 개발시 유용)
- `util.inspect.defaultOptions.showHidden = true` : log에 non-enumerable, internal 속성 포함
- `--expose-gc` : garbage collection를 실행하는 `global.gc()` 사용 - 메모리 문제 디버깅 시 유용

Debugging C++

gdb or lldb

```
# build with debugging symbols for Node.js parts
$ ./configure --debug-node
```

Option: VSCode (Launch/Attach)

```
// .vscode/launch.json
  "configurations": [
    {
      "type": "cppdbg",
      "name": "Node (Debug)",
      "request": "launch",
      "program": "${workspaceFolder}/out/Release/node",
      "args": [ ... ],
      "cwd": "${workspaceFolder}",
      "environment": [ { "name": "NODE_DEBUG_NATIVE", "value": "..." } ],
      ...
    },
  ]
}
```

- https://code.visualstudio.com/docs/cpp/cpp-debug

Debugging C++

Tips

`'NODE_DEBUG_NATIVE'` 환경변수 : C++ 모듈의 Debug Log 출력 (release 빌드도 사용 가능)

```
$ NODE_DEBUG_NATIVE=worker node test/parallel/test-worker-onmessage.js
```

```
// per_process::Debug(...)
// Searching for available categories: src/debug_utils.h
enum class DebugCategory : unsigned int {
#define V(name) name,
  DEBUG_CATEGORY_NAMES(V)
#undef V
};
```

Debugging C++

Tips

- `compile_commands.json` 생성 : IntelliSense 및 정적 분석 등에 활용

```
# compile_commands.json: information for development environments to understand code context
$ ./configure -C

# bear --  make -j$(nproc)
# ninja -t out/Release/compdb > compile_commands.json
```

- `--allow-natives-syntax` : V8 intrinsics 사용. `%` prefix. (deps/v8/src/runtime/runtime.h)

```
# node --allow-natives-syntax -e "%DebugPrint(CustomEvent);"
DebugPrint: 0x30090ec4df29: [Function]
 - map: 0x23238004e699 <Map[64](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x30090ec4d521 <JSFunction Event (sfi = 0x2323800477d9)>
 - function prototype: 0x1fe9531539f9 <CustomEvent map = 0x23238004e841>
 - kind: DerivedConstructor
...
 - source code: (type, options = kEmptyObject) {
    if (arguments.length === 0)
      throw new ERR_MISSING_ARGS('type');
      ...
```

Thank you

https://github.com/daeyeon/code-and-learn

daeyeon.dev@gmail.com