DEBUGGING NODE.JS

DEBUGGING METHODS OVERVIEW

- logging
- command line debugger
- node-inspector
- WebStorm

LOGGING

CHALLENGES DEBUGGING ASYNC CODE

Stack traces are truncated

- `domains` - usable today
- `AsyncListener` - landed in Node v0.11.9 (Unstable)
- userland
  - `longjohn` - adds overhead

USING THE BUILT IN DEBUGGER

NODE FLAGS

- `--debug` - open a port and listen for remote debuggers
- `--debug-brk` - same as `--debug` but breaks on the first line

JAVASCRIPT BUILTINS

- `debugger;` - acts as a breakpoint

EXECUTION CONTROL

- `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)

BREAKPOINT CONTROL

- `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 functions body
- `setBreakpoint('script.js', 1)`, `sb(...)` - Set breakpoint on first line of script.js
- `clearBreakpoint`, `cb(...)` - Clear breakpoint

USING THE INTERNAL DEBUGGER

01_internal.js

```
for (var i = 0; i<100; i++) {

  console.log('i', i);

  if (i === 10) {
    debugger;
  }
}
```

```
$ node debug 01_internal.js
< debugger listening on port 5858
connecting... ok
break in 01_internal.js:1
  1 for (var i = 0; i<100; i++) {
  2
  3   console.log('i', i);
debug> cont
< i 0
< i 1
< i 2
...
< i 10
break in 01_internal.js:6
  4
  5   if (i === 10) {
  6     debugger;
  7   }
  8 }
debug>
```

ADDING BREAKPOINTS

`setBreakpoint(line)` or `sb(line)`

```
debug> restart
...
debug> debug> setBreakpoint(3)
  1 for (var i = 0; i<100; i++) {
  2
* 3   console.log('i', i);
  4
  5   if (i === 10) {
  6     debugger;
debug> c
break in 01_internal.js:3
  1 for (var i = 0; i<100; i++) {
  2
* 3   console.log('i', i);
  4
  5   if (i === 10) {
debug>
```

WATCHING EXPRESSIONS

watch(expression) and unwatch(expression)

```
debug> watch('i');
debug> c
< i 0
break in 01_internal.js:3
Watchers:
  0: i = 1

  1 for (var i = 0; i<100; i++) {
  2
* 3   console.log('i', i);
  4
  5   if (i === 10) {
```

REMOVING BREAKPOINTS

clearBreakpoint(file, line) or cb(file, line)

```
debug> clearBreakpoint('src/01_internal.js', 3)
  1 for (var i = 0; i<100; i++) {
  2
  3   console.log('i', i);
  4
  5   if (i === 10) {
  6     debugger;
  7   }
  8 }
```

ATTACHING A REMOTE DEBUGGER

First we need to make node listen for remote debugging connections

```
$ node --debug-brk 01_internal.js
```

connect to the debugging session

```
$ node debug 127.0.0.1:5858
```

ENTER DEBUG MODE ON A RUNNING PROCESS

02_interval.js

```
var i = 0;
setTimeout(function tick() {
  i++;
  console.log(i);

  if (i === 10) {
    debugger;
  }

  if (i < 100) {
    setTimeout(tick , 1000);
  }
}, 1000);
```

```
$ node 02_interval.js
1
```

SIGNAL THE PROCESS

In another terminal, find the pid

```
ps | grep 02_interval
```

send a `SIGUSR1` signal to the process

```
$ kill -SIGUSR1 <pid>
```

In the first terminal you should see

```
Hit SIGUSR1 - starting debugger agent.
debugger listening on port 5858
```

NODE-INSPECTOR

- remote debugging interface for node.js
- runs in chrome
- based on Blink Developer Tools (WebKit Web Inspector).

USING NODE-INSPECTOR

Install

```
npm install -g node-inspector
```

Launch `node-inspector`

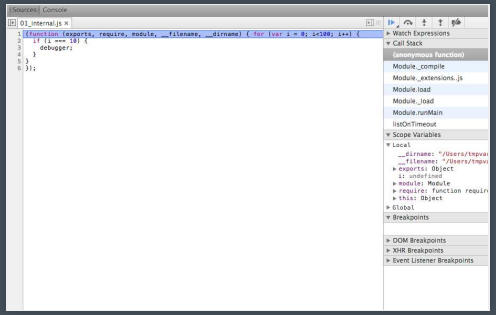
```
$ node-inspector
Node Inspector v0.6.2
   info  - socket.io started
Visit http://127.0.0.1:8080/debug?port=5858 to start debugging.
```

Launch the file to be debugged

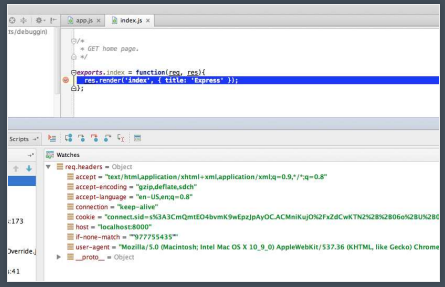
```
node --debug-brk 01_internal.js
```

OPEN IN CHROME

```
http://127.0.0.1:8080/debug?port=5858
```



WEBSTORM


app.js
index.js
ts/debuggin)
/*
* GET home page.
*/
exports.index = function(req, res){
res.render('index', { title: 'Express' });
};
Scripts
Watches
req.headers = Object
accept = "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
accept-encoding = "gzip,deflate,sdch"
accept-language = "en-US,en;q=0.8"
connection = "keep-alive"
cookie = "connect.sid=s%3A3CmQmtEO4bvmK9wEpzJpAyOC.ACMniKuJO%2FxZdCwKTN2%2B%2B6o%2BiU%2B0
host = "localhost:8000"
if-none-match = ""977755435""
user-agent = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
__proto__ = Object
:173
Override.j
:41

INSTALLING WEBSTORM

download the installer appropriate to your platform

```
http://www.jetbrains.com/webstorm/download/
```

Open and follow the instructions to install

DEBUGGING WITH WEBSTORM

- setup a project
- set a breakpoint by clicking left of the line number
- Click the little "bug" icon or press Control+d

OTHER GUI DEBUGGERS

- Brackets with Theseus
- Cloud9 IDE (c9.io)
- Eclipse with Google Chrome Developer Tools

SUMMARY

- Log everything
- Use the right tool for the job
- Get comfortable with the available tools