

Node.js 빌드 & 테스트

2024, OSSCA, Node.js - Code and Learn

Daeyeon Jeong

Index

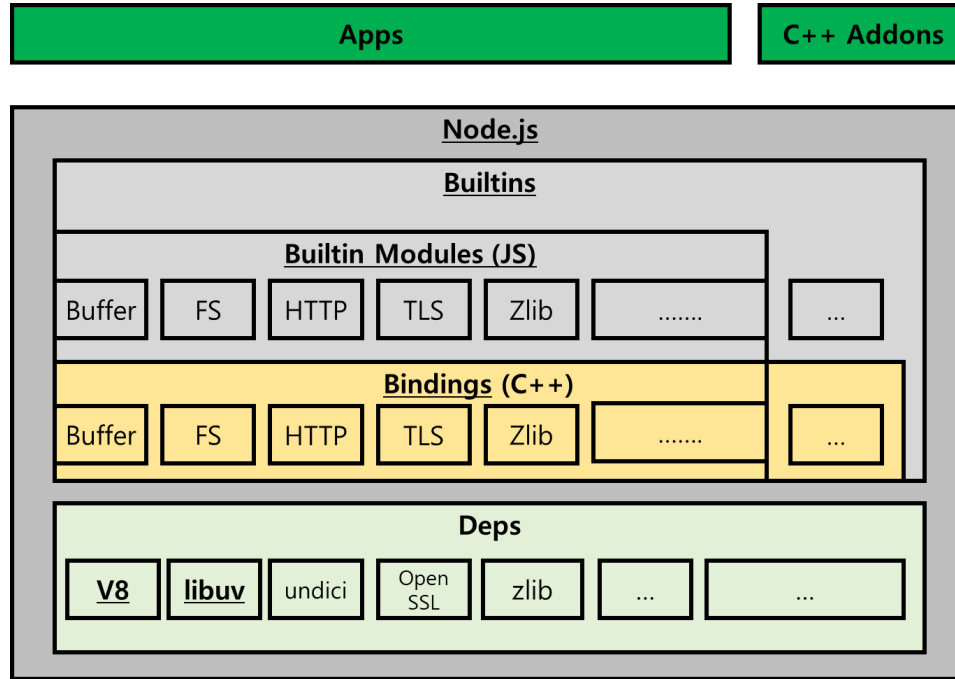
- Node.js 런타임
 - 전반적인 구조, 프로젝트 구성
 - 빌드 시스템 동작
 - 개발시 빌드 시간 단축 팁
 - Test Suite 구성
- 대상 : C++ 프로젝트 경험이 거의 없음

Node.js Terms

- `Built-in(s)` or `Built-in module(s)` : JavaScript code built-in to Node core.
- `(C++) bindings` : APIs used in JS builtins that are run in C++ code. (parts of builtins)
- `addons` : Dynamically-linked shared objects provide APIs via Node-API, NAN, ...

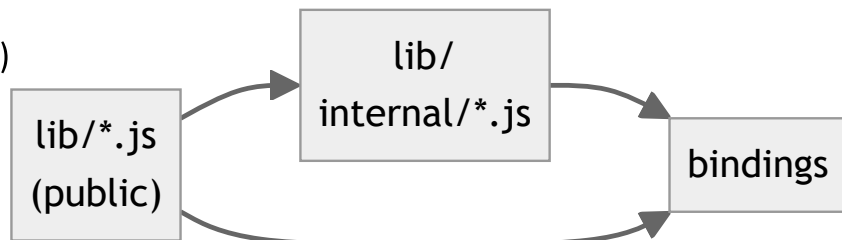
Architecture

Node.js runtime



Directory Structure

- lib - JavaScript Builtins
 - lib/**/*.js, internals, internal/bootstrap
- src
 - src/node_*.cc (bindings)
 - node_api.h (addons)
 - node_main.cc, env.cc, ...
- deps - v8, libuv, zlib, openssl, undici, sqlite ...
- tools - test.py, gyp, lint-md, dep_updaters, actions/commit-queue, ...
- test - parallel, sequential, cctest, addons, wpt, ...
- benchmark - compare.js, benchmark/**/*.js
- typings - internal modules code completions (tsconfig.json)
- doc - *.md, node.1



빌드 과정

1. Configuration

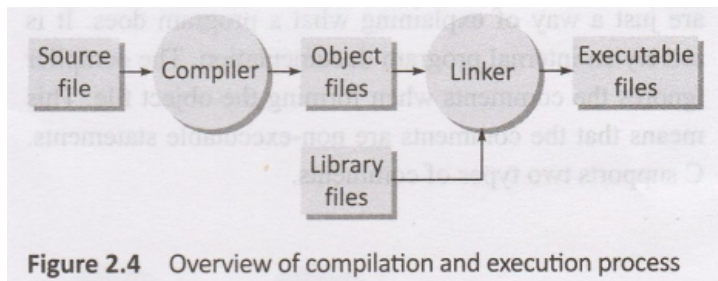
2. Build

```
$ ./configure
```

```
$ make -j4
```

빌드 과정

- 실행 파일, `executable`
- 구현된 소스 파일 각각을 machine code로 만든다. (전처리 - (컴파일 - 어셈블))
 - Compiler 이용
 - `obj (.o)` 파일과, `obj`를 집합한 `library (.a or .so)`를 생성
- `obj, library`를 통합해 `executable`를 만든다. (링크)
 - Linker 이용
 - `library`는 `static lib (.a)`, `shared lib (.so)` 타입이 있음.
 - `executable`는 `library`와 달리 실행 `entry`가 있다. (`main` 함수)



빌드 과정

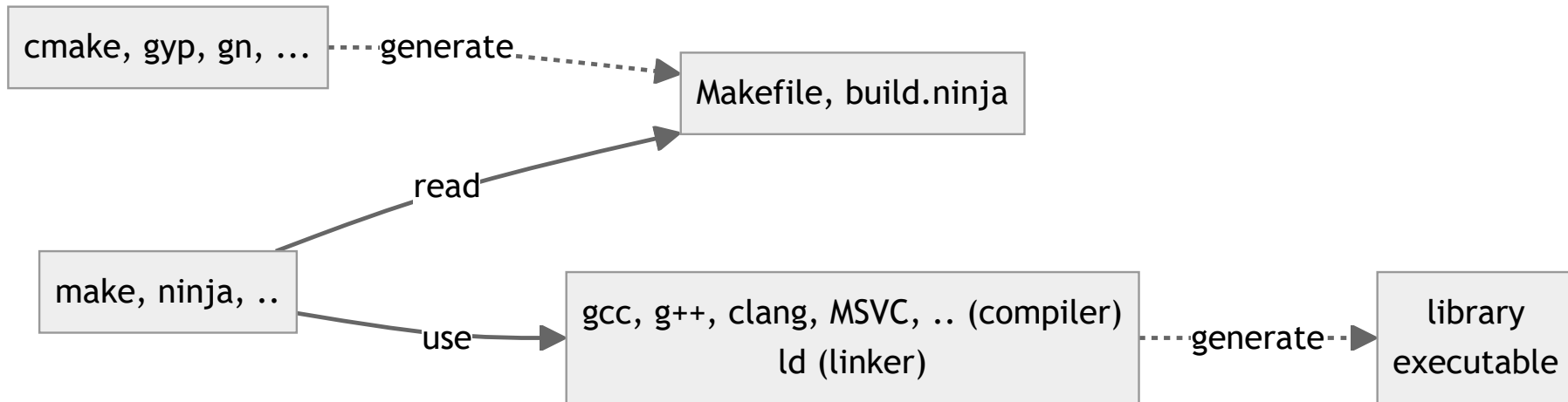
- Compiler, Linker를 다룰때 많은 파일, 설정이 있는 경우 하나씩 옵션으로 전달하기 어렵다.

```
$ gcc test.c -o test
```

```
$ gcc -o /home/daeyeon/code/nodeland/daeyeon.node/out/Release/obj.target/v8_initializers/gen/torque-generated/third_party/nodeland/daeyeon.node/out/Release/obj/gen/torque-generated/third_party/v8/builtins/array-sort-tq-csa.cc '-D_GLIBCXX_USE_CXX11_ABI=0' '-D__STDC_FORMAT_MACROS' '-DV8_TARGET_ARCH_X64' '-DV8_HAVE_TARGET_OS' '-DV8_TARGET_OS_LINUX' '-DV8_EMBEDDER_STRING="-node.16"' '-DENABLE_DISASSEMBLER' ... '-DV8_ENABLE_MAGLEV' '-DV8_ENABLE_TURBOFAN' '-DV8_ENABLE_WEBASSEMBLY' ... -Wno-int-in-bool-context -Wno-deprecated -Wno-stringop-overflow -W ....
```

- 그래서, `make` 나 `ninja` 등의 build tool을 사용한다.
- 그런데 `make` 를 이용해 configuration을 직접 작성하기도 어렵다.
 - 그래서 다른 툴 (meta-build system) 을 더 쓴다. (`cmake` , `gyp` , `gn` ...)

빌드 과정



Node.js 빌드 과정

Configuration

- `out/Makefile` , `out/**/*.mk` or `out/**/*.ninja` 등 빌드 구성 생성
- Options for 'embedders' to build customized Node.js
 - Fast bootup, memory footprint, ...
 - a) feature enabling, b) platform, c) debug/release, d) library linking, e) deps options ...
- GYP (Generate Your Projects)*, `tools/gyp` 를 사용
 - gyp 기반 빌드 구성 (Node Runtime 빌드 및 addon)
 - Chromium project. Python 사용. `make` 와 `ninja` 빌드용 configuration 파일을 생성 가능
 - No longer actively maintained. Discouraged for new projects.
 - GN (Generate Ninja) is a build system to replace GYP.
 - GN build is unofficially supported. (`doc/gn-build.md`)

Node.js 빌드 과정

- configure.py
 - 사용자 옵션 및 빌드환경을 기록한 `config.gypi` 를 생성하고 gyp를 실행
 - `run_gyp(gyp_args)` in `configure.py` 실행
 - `run_gyp: tools/gyp_node.py -Dbuild_type=Release ... -f make-linux` 실행
 - `tools/gyp_node.py` 에서 `config.gypi` 과 `node.gyp` 빌드 설정 파일들을 로드
 - `config.gypi, ... <-- node.gypi <-- node.gyp`
 - 그 결과 `out/Makefile`, `out/**/*.mk` or `out/**/*.build.ninja` 생성

```
$ ./configure --help
$ ./configure --verbose
$ ./configure --debug --debug-node

$ node -e 'console.log(process.config)'
```

Node.js 빌드 과정

Build

- Building dependencies (v8, libuv, zlib, openssl, icu, ...)
 - `out/**/obj/**/* .a`
- Building internal utils (node_js2c, node_mksnapshot, ...)
 - `out/**/node_js2c` , `out/**/node_mksnapshot`
- Converting JavaScript Builtins via `node_js2c`
 - ★ `out/**/gen/node_javascript.cc`
- Building `out/**/obj/libnode.a`
- Optimization
 - Creating `node_mksnapshot`
 - Creating a V8 snapshot^1 via `node_mksnapshot` -> `out/**/obj/gen/node_snapshot.cc`
- Making a Node.js executable (`node`) -> `libnode.a` + `node_snapshot.cc` , and ...

Node.js 빌드 과정

V8 snapshot

A V8 snapshot is a binary file that contains a snapshot of the V8 engine's state, including:

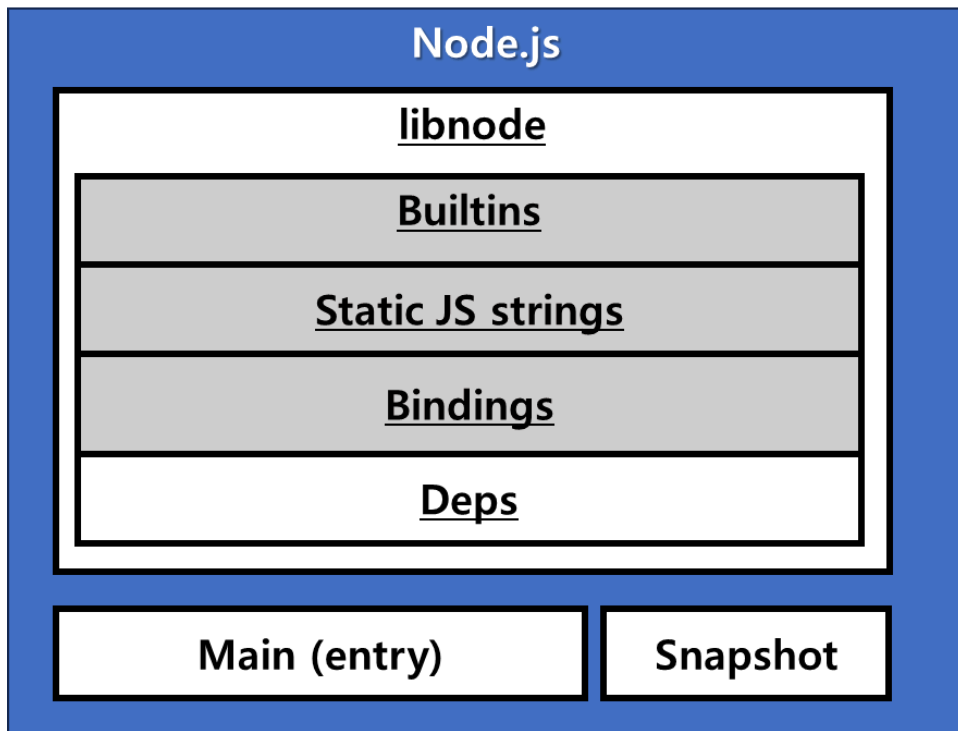
1. The V8 engine's configuration
2. The JavaScript code and its execution context
3. The heap objects and their references
4. The JavaScript stack and its frames

When a new Node.js process starts, it can load a V8 snapshot instead of initializing the V8 engine from scratch. This snapshot provides a pre-existing state that has already been initialized, which reduces the startup time significantly.

- `lib/internal/per_context/**`
- `lib/internal/bootstrap/**`
- `lib/internal/bootstrap/node.js`


Node.js 빌드 과정

Executable layout



Node.js 빠른 빌드

Fast build tips

- `make -j$(nproc)`
 - Parallel compilation to maximize computer resources
 - `ninja` (`doc/contributing/building-node-with-ninja.md`)
-  `ccache` (`BUILDING.md#speeding-up-frequent-rebuilds-when-developing`)
- `mold` (On GNU/Linux Only)

```
$ ccache -M 50GB
$ ccache -p
...
(~/.ccache/ccache.conf) max_size = 50.0G
$ ccache -s
Summary:
Hits:           600710 / 862387 (69.66 %)
Direct:         515268 / 710279 (72.54 %)
Preprocessed:   85442 / 241041 (35.45 %)

$ ccache -C
```

Node.js 빠른 빌드

```
# ★ When modifying only the JS layer in lib:
$ ./configure --node-builtin-modules-path "$(pwd)"

# Turn off creating snapshot
$ ./configure --without-node-snapshot

# Use shared libraries
$ ./configure --shared-openssl --shared-zlib

# Turn off features
$ ./configure --without-inspector --with-intl small-icu (or --without-intl)
```


Node.js Test

```
$ make test-only
```

```
# Makefile
.PHONY: test-only
test-only: all ## For a quick test, does not run linter or build docs.
    $(MAKE) build-addons
    $(MAKE) build-js-native-api-tests
    $(MAKE) build-node-api-tests
    $(MAKE) cctest
    $(MAKE) jstest
    $(MAKE) tooltest

...

.PHONY: jstest
jstest: build-addons build-js-native-api-tests build-node-api-tests ## Runs addon tests and JS tests
    $(PYTHON) tools/test.py $(PARALLEL_ARGS) --mode=$(BUILDTYPE_LOWER) \
        $(TEST_CI_ARGS) --skip-tests=$(CI_SKIP_TESTS) (JS_SUITES) $(NATIVE_SUITES)
```

```
$ python3.11 tools/test.py --mode=release --skip-tests= \
    default addons js-native-api node-api
```

Node.js Test

tools/test.py

```
# these suites represent special cases that should not be run as part of the
# default JavaScript test-run, e.g., internet/ requires a network connection,
# addons/ requires compilation.
```

```
IGNORED_SUITES = [
    'addons',
    'benchmark',
    'doctool',
    'embedding',
    'internet',
    'js-native-api',
    'node-api',
    'pummel',
    'tick-processor',
    'v8-updates'
]
```

```
def ArgsToTestPaths(test_root, args, suites):
    if len(args) == 0 or 'default' in args:
        def_suites = [s for s in suites if s not in IGNORED_SUITES]
    ...
```

```
# abort, async-hooks, es-module, known_issues, message, parallel, pseudo-tty,
# report, sequential, wasi, wpt, wasm-allocation
```

Node.js Test

- test 내 sub 폴더 이름 별로 테스트가 분류되어 있으며 파일도 naming prefix 규칙이 있음
 - `test/parallel` - 병렬로 실행 가능 테스트, 일반적인 각 모듈의 대한 테스트는 이곳에 작성됨
 - `test/sequential` - 순차적으로 실행되어야하는 테스트
 - `test/cctest` - gtest로 구성 Node.js embedder를 위한 테스트
 - `test/addons` - Node-API 테스트
 - `test/wpt` - Web 호환성 테스트, Web Platform Test 사용
 - ...
 - 파일명 예: `test-buffer-readuint.js`, `test-child-process-emfile.js`
- 개발시 수정 내용의 영향 범위를 고려해서 선택적으로 수행할 수 있음
 - `test/parallel`, `sequential`, `async-hooks` 는 대부분 모듈 테스트를 충족
 - `test/wpt` Web API와 관련된 경우 필요

Node.js Test

- 각 테스트 폴더에는 테스트 상태 파일 (`*.status`) 과 test suite 확인 파일 (`testcfg.py`) 파일이 있음
 - 예: `test/parallel/parallel.status` , `test/parallel/testcfg.py`
- 환경에 따라 SKIP 해야하거나 FLAKY (may work correctly most of the time, but occasionally fail) 표시

...

```
[$arch=arm || $arch=arm64]
```

```
# https://github.com/nodejs/node/pull/31178
```

```
test-crypto-dh-stateless: SKIP
```

```
[$system=macos]
```

```
# https://github.com/nodejs/node/issues/42741
```

```
test-http-server-headers-timeout-keepalive: PASS,FLAKY
```

```
test-http-server-request-timeout-keepalive: PASS,FLAKY
```

Node.js Test

```
$ tools/test.py -p color --report --timeout 60 --worker \  
test/parallel test/sequential test/async-hooks
```

```
# -p PROGRESS, --progress=PROGRESS (verbose, dots, actions, color, tap, mono, deopts)  
# --time Print timing information after running  
# --repeat 1
```

- `tools/test.py` 는 module 테스트이고 PR submit 전 local 에서 사전 체크해야 할 내용이 더 있음
 - Commit messages
 - Linting, Formating (CC, JS, MD) 등

Thank you

<https://github.com/daeyeon/code-and-learn>
daeyeon.dev@gmail.com