# Setup

- Unix 환경 권장 (Ubuntu, OSX, WSL 등)
- Node.js LTS 버전 설치 (https://nodejs.org/ko)
  - Ubuntu) apt install node.js
  - OSX) brew install node.js

- 도커 및 실습 문제 셋업
  - apt install docker.io docker-compose
  - cd calc_for_user (docker-compose.yml 경로)
  - docker-compose up -d
  - http://[본인IP]:20000/ 접속 확인

# Node.js: 모던 앱 개발 및 보안

2025. 4.

# $cat .profile

## 김지섭 Jisub Kim @jskimpwn

現) 금융보안원 레드팀 RED IRIS
前) 라온화이트햇 핵심연구팀

BoB 7기 취약점 분석 트랙 TOP 10
CTF w/ @zer0pts @SuperDiceCode @군필
Interested in Web2, Embedded System, blockchain

# Node.js 소개

# Node.js

- Javascript를 브라우저 밖에서 실행할 수 있도록 하는 런타임
  - 런타임: 특정 언어로 만든 프로그램을 실행할 수 있는 환경
- 비동기 이벤트 주도 JavaScript 런타임으로써 Node.js 는 확장성 있는 네트워크 애플리케이션을 만들 수 있도록 설계됨

# Node.js

- Javascript를 브라우저 밖에서 실행할 수 있도록 하는 런타임
  - 런타임: 특정 언어로 만든 프로그램을 실행할 수 있는 환경

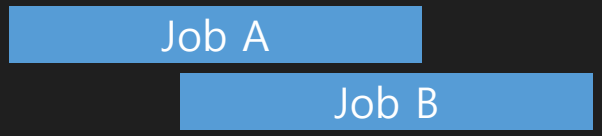
- 비동기 이벤트 주도 JavaScript 런타임으로써 Node.js 는 확장성 있는 네트워크 애플리케이션을 만들 수 있도록 설계됨

# Node.js 주요 특징

- 비동기 논블로킹(Non-blocking) IO
- 싱글스레드
- 이벤트 기반 (Event-driven)

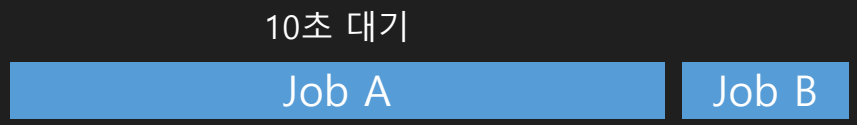
# 동기(Sync) vs 비동기(Async)


동기 (Sync)

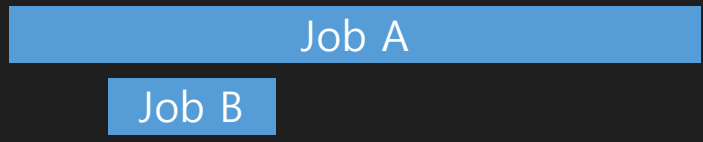



비동기 (Async)

# 동기(Sync) vs 비동기(Async)

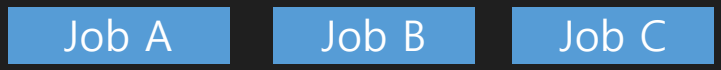



동기 (Sync)

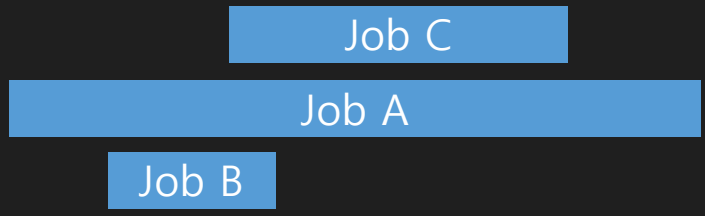



비동기 (Async)

# 동기(Sync) vs 비동기(Async)




동기 (Sync)

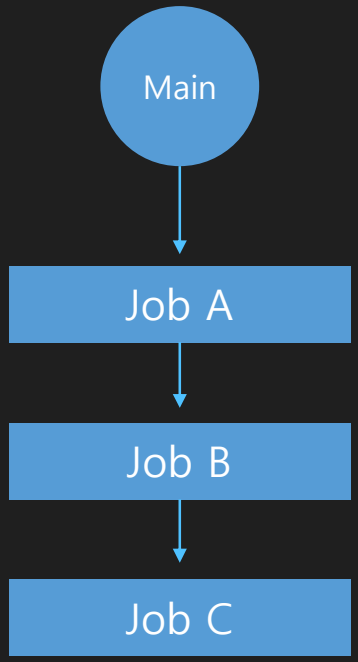



비동기 (Async)

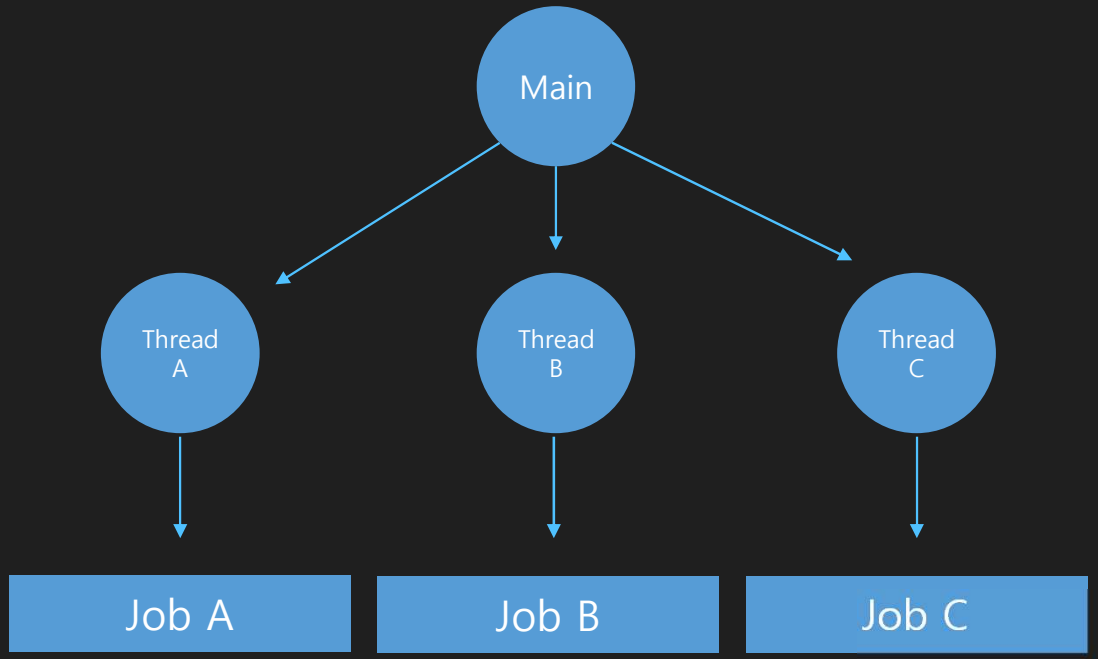
# 블로킹 vs 논블로킹

- 블로킹
  - 한 작업이 완료될 때까지 다른 작업을 시작하지 않고 기다림
    - 예) 은행에서 창구 직원이 한 고객의 업무를 모두 처리할 때까지 기다림
  - 호출된 함수에서 I/O작업등을 요청했을 경우 I/O작업이 처리되기 전까지 아무 일도 하지 못함
- 논블로킹
  - 작업을 시작하고 완료를 기다리지 않고 다음 작업을 진행
    - 예) 패스트푸드점에서 주문 후 진동벨을 받고 기다리는 동안 다른 일 처리
  - 호출된 함수에서 I/O작업등을 요청했을 경우 I/O작업의 처리여부와 관계없이 바로 다음 작업을 할 수 있음

# 싱글스레드 / 멀티스레드

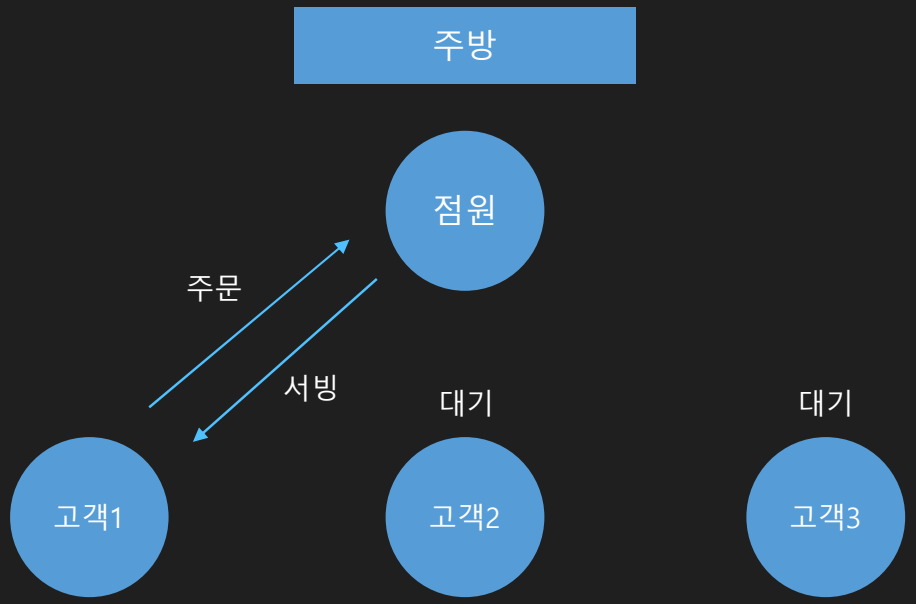



순차실행



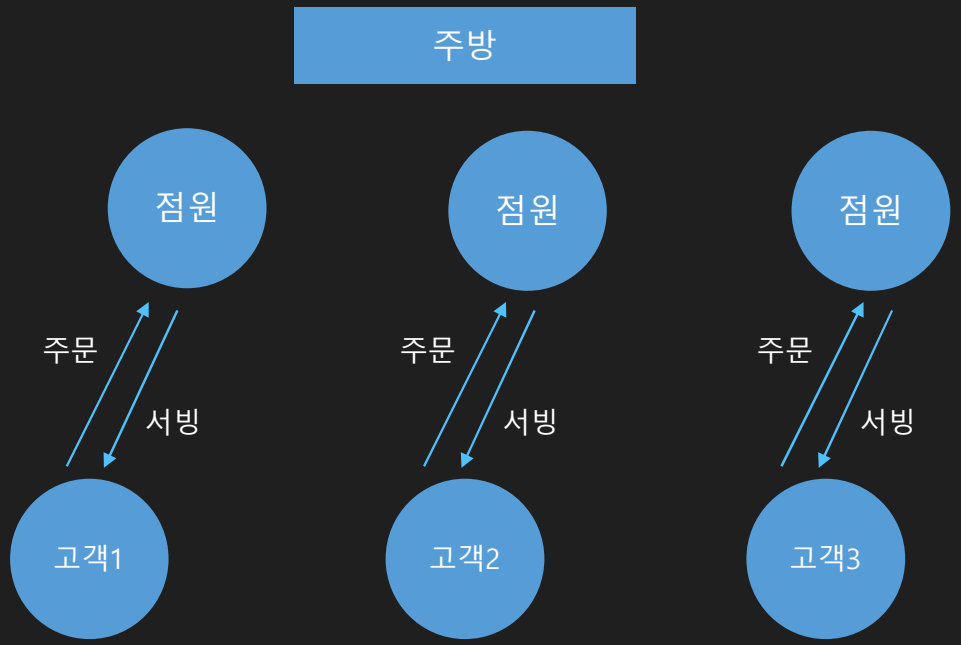

병렬실행

# Node.js: 싱글스레드



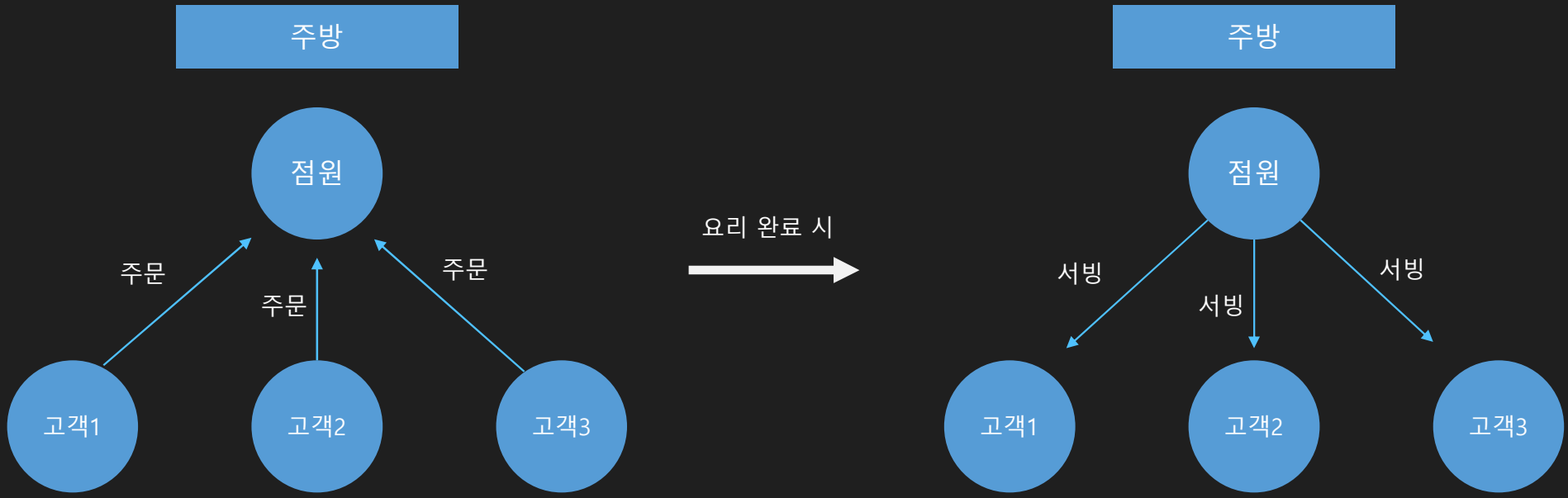

싱글스레드, 블로킹

# Node.js: 멀티스레드



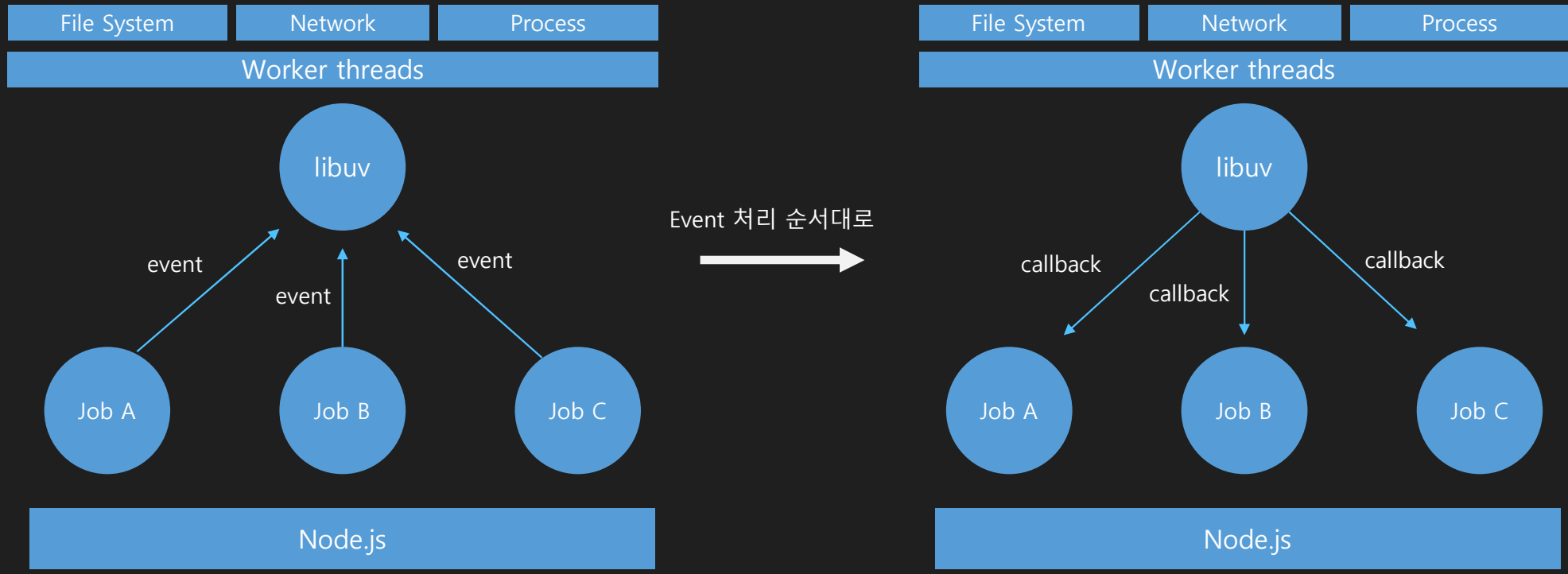

멀티스레드, 블로킹

# Node.js: 싱글스레드 + 논블로킹



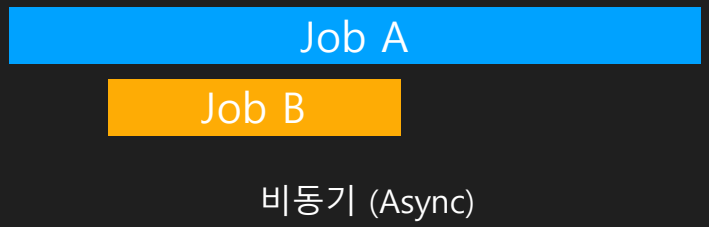

싱글스레드, 논블로킹

# Node.js: 싱글스레드 , 논블로킹 IO

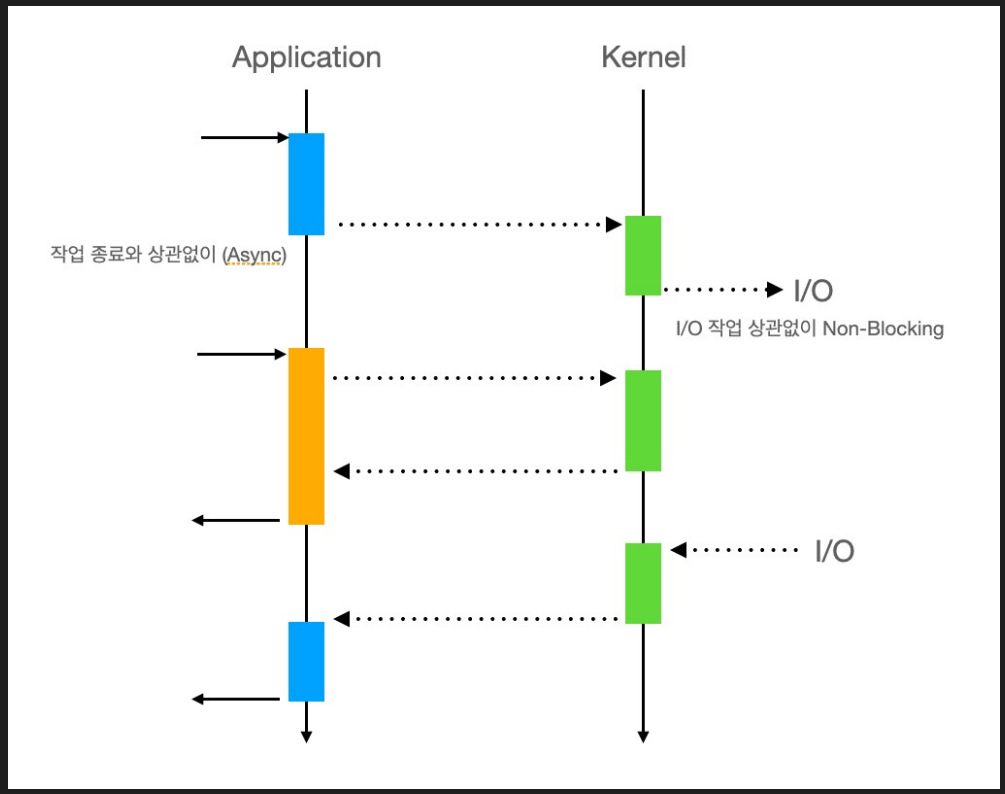



Node.js의 싱글스레드, 논블로킹

# Node.js: 싱글스레드 , 논블로킹 IO




비동기 (Async)




Node.js의 비동기 논블로킹 IO

# Node.js 주요 특징

- 비동기 논블로킹(Non-blocking) IO
- 싱글스레드
- 이벤트 기반 (Event-driven)

# Node.js의 장단점

| 장점 | 단점 |
| --- | --- |
| 멀티스레드 방식에 비해 컴퓨터 자원 적게 사용 | 싱글스레드라 CPU 코어 하나만 사용 |
| I/O 작업이 많은 서버로 적합 | CPU 작업이 많은 서버로 부적합 |
| 멀티스레드 방식보다 구현 쉬움 | 하나뿐인 스레드가 멈추면 종료됨 |
| 웹서버 내장 | 서버 규모가 커졌을 때 관리하기 어려움 |
| Javascript 사용 | 어중간한 성능 |
| JSON 과 높은 호환성 | |

# Express.js

- Node.js를 위한 빠르고 간결한 웹 프레임워크

```
1. Node.js 웹 애플리케이션을 작성할 폴더 생성 및 작업 디렉토리 설정
$ mkdir myapp
$ cd myapp

2. npm init 명령을 통해 애플리케이션에 대한 package.json 작성
$ npm init

3. express 모듈 설치
$ npm install express
```

# 실습1: Node.js 설치 및 웹앱 구현

// index.js

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

node index.js 명령어로 웹앱 실행 → localhost:3000 접속 테스트

# 실습2: Express 기본 라우팅

```
app.METHOD(PATH, HANDLER)
```

```
app.get('/', function (req, res) {
    res.send('Got a GET request');
  });

app.post('/', function (req, res) {
    res.send('Got a POST request');
});

app.post('/user', function (req, res) {
    res.send('Got a POST request at /user');
});
```

# 실습3: Express 정적(static) 파일

```
app.use(express.static('public’));
```

public/images/kitten.jpg
➔ http://localhost:3000/images/kitten.jpg

public/css/style.css
➔ http://localhost:3000/css/style.css

public/js/app.js
➔ http://localhost:3000/js/app.js

# 실습4: req.query, params, body

GET http://localhost:3000/?a=b

```
app.get('/', function (req, res) {
    console.log(req.query)
    res.send('Got a GET request');
});
```

GET http://localhost:3000/user/guest

```
app.get('/user/:id', function (req, res) {
    console.log(req.params)
    res.send('Got a GET request at /user');
});
```

POST http://localhost:3000/

{"a": "b"}

```
app.use(express.json())
app.post('/', function (req, res) {
    console.log(req.body)
    res.send('Got a POST request');
});
```

# Node.js Essentials

# Node.js 전역 객체

- Global Objects
  - global
  - process
  - console
  - require
  - __filename
  - __dirname
  - Exports
  - setInterval
  - setTimeout

# Node.js 전역 객체

- Global Objects
  - global
  - process
  - console
  - require
  - __filename
  - __dirname
  - exports
  - setInterval
  - setTimeout

```
> process
process {
  version: 'v18.17.1',
  versions: {
    node: '18.17.1',
    acorn: '8.8.2',
    ada: '2.5.0',
    ares: '1.19.1',
    brotli: '1.0.9',
    cldr: '43.0',
    icu: '73.1',
    llhttp: '6.0.11',
    modules: '108',
    napi: '9',
    nghttp2: '1.52.0',
    nghttp3: '0.7.0',
    ngtcp2: '0.8.1',
    openssl: '3.0.10+quic',
    simdutf: '3.2.12',
```

```
> require
[Function: require] {
  resolve: [Function: resolve] { paths: [Function: paths] },
  main: undefined,
  extensions: [Object: null prototype] {
    '.js': [Function (anonymous)],
    '.json': [Function (anonymous)],
    '.node': [Function (anonymous)]
  },
  cache: [Object: null prototype] {}
}
```

# Node.js 전역 객체: require

```
require('fs').readdirSync('/')
```

```
> require('fs').readdirSync('/')
[
  'Docker',    'bin',         'boot',
  'dev',       'etc',         'flag',
  'flag.txt',  'home',        'init',
  'lib',       'lib32',       'lib64',
  'libx32',    'lost+found',  'media',
  'mnt',       'opt',         'proc',
  'root',      'run',         'sbin',
  'snap',      'srv',         'sys',
  'tmp',       'usr',         'var'
]
>
```

# Node.js 전역 객체: require

```
require('fs').writeFile('a.txt',"testdata",err=>{err})
```

```
→  ~ ls -al a.txt
-rw-r--r-- 1 jskim jskim 8 Oct  7 21:56 a.txt
```

```
require('fs').readFileSync('a.txt',err=>{err})
```

```
> require('fs').readFileSync('a.txt',err=>{err})
<Buffer 74 65 73 74 64 61 74 61>
> require('fs').readFileSync('a.txt',err=>{err}).toString()
'testdata'
```

# Node.js 전역 객체: require

```
require('child_process').execSync('id').toString()
```

```
> require('child_process').execSync('id').toString()
'uid=1000(jskim) gid=1000(jskim) groups=1000(jskim),4(adm),
),46(plugdev),116(netdev),1001(docker)\n'
```

```
require('child_process').spawnSync('id').stdout.toString()
```

```
> require('child_process').spawnSync('id')
{
  status: 0,
  signal: null,
  output: [
    null,
    <Buffer 75 69 64 3d 31 30 30 30 28 6a 73 6b 69 6d 29 20 67 69 64 3d 31 30 30 30 28 6a 73 6b 69
d 31 30 30 30 28 6a 73 6b 69 6d 29 ... 115 more bytes>,
    <Buffer >
  ],
  pid: 6896,
  stdout: <Buffer 75 69 64 3d 31 30 30 30 28 6a 73 6b 69 6d 29 20 67 69 64 3d 31 30 30 30 28 6a 73
```

```
> require('child_process').spawnSync('id').stdout.toString()
'uid=1000(jskim) gid=1000(jskim) groups=1000(jskim),4(adm),20(dialout),
```

# Node.js 전역 객체: process

- process 객체: 현재 실행 중인 Node 프로세스 정보를 담고 있는 객체

| 속성 | 설명 |
| --- | --- |
| process.env | 호스트 환경변수 정보 |
| process.version | Node.js 버전 |
| process.execPath | Node.js 실행경로 |
| process.mainModule | main 모듈의 정보를 담고 있는 객체 |
| process.argv | 프로세스 실행 시 전달된 매개변수 |
| process.pid | Node.js process 의 pid 정보 |

# Node.js 전역 객체: process

- process.mainModule.require == process.mainModule[‘require’]
- == process.mainModule[‘r’+’equire’]

```
process.mainModule.require('child_process').execSync('id').toString()
```

```
→  ~ node index.js
uid=1000(jskim) gid=1000(jskim) groups=1000(jskim),4(adm),20(dial
dip),44(video),46(plugdev),116(netdev),1001(docker)
```

# Node.js Constructor/Prototype

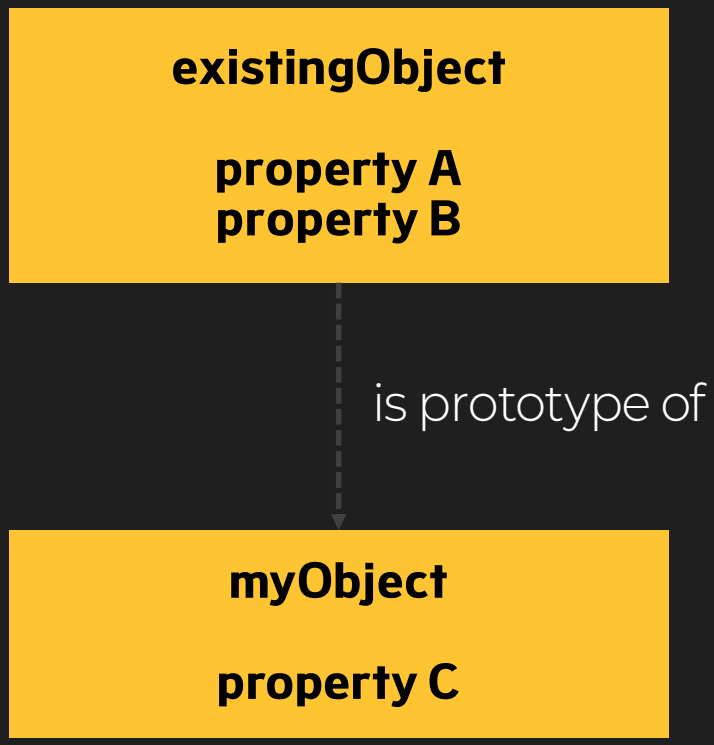
```
let myObject = {};
Object.getPrototypeOf(myObject);   // Object.prototype

let myString = "";
Object.getPrototypeOf(myString);   // String.prototype

let myArray = [];
Object.getPrototypeOf(myArray);    // Array.prototype

let myNumber = 1;
Object.getPrototypeOf(myNumber);   // Number.prototype
```

# Node.js Constructor/Prototype

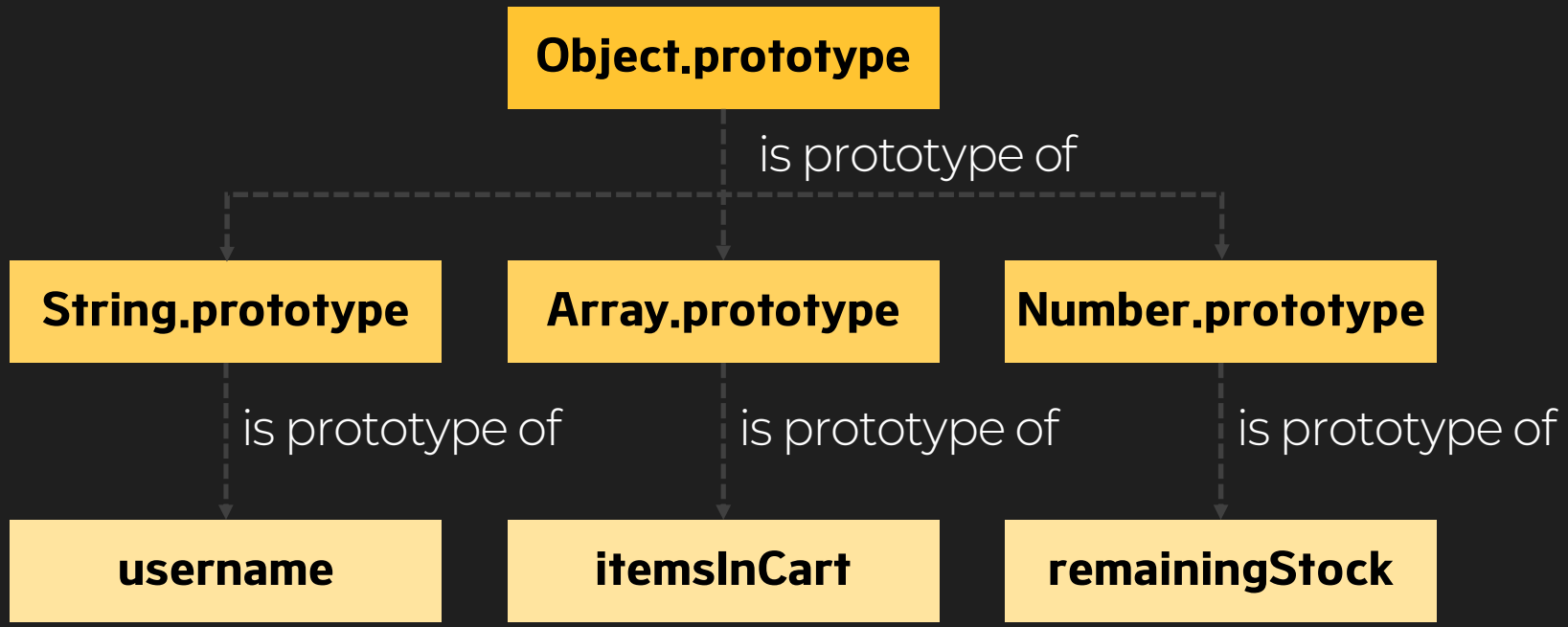


# Node.js Constructor/Prototype

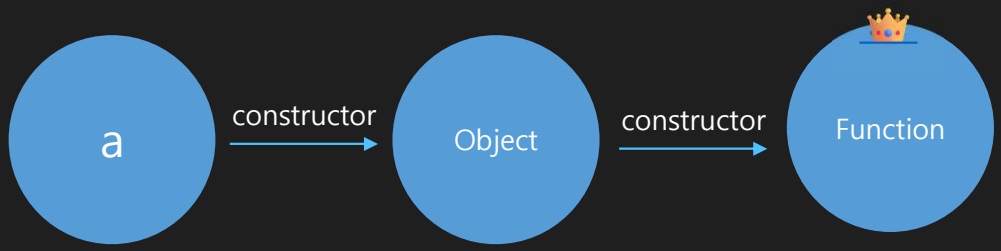
```
Welcome to Node.js v20.17.0.
Type ".help" for more information.
> let myObject = {}
undefined
> myObject.
myObject.__proto__              myObject.constructor           myObject.hasOwnProperty
myObject.isPrototypeOf          myObject.propertyIsEnumerable  myObject.toLocaleString
myObject.toString               myObject.valueOf

> myObject.
> Object.prototype.
Object.prototype.__proto__              Object.prototype.constructor
Object.prototype.hasOwnProperty         Object.prototype.isPrototypeOf
Object.prototype.propertyIsEnumerable   Object.prototype.toLocaleString
Object.prototype.toString               Object.prototype.valueOf
```

# Node.js Prototype Chain



# Node.js Constructor/Prototype




```
> let a = Object()
undefined
> a
{}
> a.constructor
[Function: Object]
> a.constructor.constructor
[Function: Function]
```

```
> a.constructor.constructor('return 3')
[Function: anonymous]
> a.constructor.constructor('return 3')()
3
```

# Node.js VM (SBX)

# Node.js vm

- 샌드박스 환경에서 신뢰할 수 없는 코드(Untrusted Code) 실행
- 메인 컨텍스트와 호스트에 대한 접근을 제한

# Node.js vm

`vm.runInNewContext(code[, contextObject[, options]])`

전달된 `contextObject`를 컨텍스트화하고 (undefined 면 새 contextObject를 생성)
코드를 컴파일하여 생성된 컨텍스트 내에서 실행한 결과를 반환
내부에서 실행되는 코드는 로컬 컨텍스트에 대한 액세스 권한이 없음

```
const vm = require('node:vm');

const contextObject = {
  animal: 'cat',
  count: 2,
};

vm.runInNewContext('count += 1; name = "kitty"', contextObject);
console.log(contextObject);
// Prints: { animal: 'cat', count: 3, name: 'kitty' }
```

# Node.js vm

```
let animal = 'cat'
let count = 2

eval(`count+=10; name="kitty"`)
```

✔

```
> count
12
> name
'kitty'
```

```
vm.runInNewContext(`count+=10; name="kitty"`)
```

❌

```
Uncaught ReferenceError: count is not defined
    at evalmachine.<anonymous>:1:1
    at Script.runInContext (node:vm:135:12)
    at Script.runInNewContext (node:vm:140:17)
    at Object.runInNewContext (node:vm:292:38)
    at REPL15:1:4
    at Script.runInThisContext (node:vm:123:12)
    at REPLServer.defaultEval (node:repl:569:29)
    at bound (node:domain:433:15)
    at REPLServer.runBound [as eval] (node:domain
    at REPLServer.onLine (node:repl:899:10)
```

# Node.js vm

```
console.log(process)
```

```
➜ nodejs node index.js
process {
  version: 'v18.17.1',
  versions: {
    node: '18.17.1',
    acorn: '8.8.2',
    ada: '2.5.0',
    ares: '1.19.1',
    brotli: '1.0.9',
    cldr: '43.0',
    icu: '73.1',
    llhttp: '6.0.11',
    modules: '108',
    napi: '9',
```

```
console.log( vm.runInNewContext('process') )
```

```
ReferenceError: process is not defined
    at evalmachine.<anonymous>:1:1
    at Script.runInContext (node:vm:135:12)
    at Script.runInNewContext (node:vm:140:17)
    at Object.runInNewContext (node:vm:292:38)
    at Object.<anonymous> (/home/jskim/nodejs/index.js:9:17)
    at Module._compile (node:internal/modules/cjs/loader:1256:1
    at Module._extensions..js (node:internal/modules/cjs/loader
    at Module.load (node:internal/modules/cjs/loader:1119:32)
    at Module._load (node:internal/modules/cjs/loader:960:12)
    at Function.executeUserEntryPoint [as runMain] (node:intern
_main:81:12)
```

# Install docker.io

- apt install docker.io docker-compose

- cd calc_public (docker-compose.yml 경로)
- docker-compose up -d

- http://[본인IP]:20000/

warmup:20000

calc0:30000

calc1:30001

calc2:30002

calc3:30003

calc4:30004

# 실습0: warmup (20000)

- flag1+flag2 = flag
- Get source → /source

- HINT: 정적 파일 라우팅과 req.query

# 실습1: calc 0

- Eval your code
- 출력 결과에 flag 가 들어가면 필터링?

- HINT

```
'flag'.replace('flag','_')
```

```
'flag'.substr(1)
```

# 실습2: calc1

- You can load node.js modules through `require`
- HINT

```
require('fs').readdirSync('.')
```

# 실습3: calc2

- You should run /readflag
- HINT

```
> ['apple'] == ['a'+'pple']
true
```

# 실습4: calc3

- You should run /readflag

- HINT: constructor, Function Object

```
> a.constructor.constructor('return 3')
[Function: anonymous]
> a.constructor.constructor('return 3')()
3
```

# 실습5: calc4

- You should run /readflag

- HINT
- process.binding
  == internalBinding

node / lib / internal / bootstrap / realm.js

Code Blame 458 lines (411 loc) · 14.3 KB

```
137     'test',
138     'test/reporters',
139   ]);
140   // Modules that will only be enabled at run time.
141   const experimentalModuleList = new SafeSet();
142
143   // Set up process.binding() and process._linkedBinding().
144   {
145     const bindingObj = { __proto__: null };
146
147     process.binding = function binding(module) {
148       module = String(module);
149       // Deprecated specific process.binding() modules, but not all, allow
150       // selective fallback to internalBinding for the deprecated ones.
151       if (processBindingAllowList.has(module)) {
152         if (runtimeDeprecatedList.has(module)) {
153           runtimeDeprecatedList.delete(module);
154           process.emitWarning(
155             `Access to process.binding('${module}') is deprecated.`,
156             'DeprecationWarning',
157             'DEP0111');
158         }
159         if (legacyWrapperList.has(module)) {
160           return requireBuiltin('internal/legacy/processbinding')[module]();
161         }
162         return internalBinding(module);
163       }
164       // eslint-disable-next-line no-restricted-syntax
165       throw new Error(`No such module: ${module}`);
166     };
```

# QnA

X @JisubK
FB @김지섭
IG @wltjqzla