Node.js

내장 모듈 - 김근형 강사

Assert

- Assert 모듈
  - 유닛 테스트를 위해서 Node.js에서 사용할 수 있는 테스트 모듈
  - 코드를 작성 시 우리가 원한 방향으로 동작하도록 자동으로 검사하는 테스트 함수들을 제공한다.
  - 테스트 주도 개발(Test Driven Development)이 대두되면서 주목하게된 모듈
  - 반드시 Assert 모듈은 require를 이용해 가져와야만 사용이 가능하다.

```
const assert = require("assert");
```

Assert

- Assert 제공함수

함수	내용
assert(value[, message])	주어진 value 값이 0 이거나 false인 경우 오류 발생. assert.ok() 와 동일
assert.equal(actual, expected[, message])	주어진 actual 값이 expected와 다를 경우 오류가 발생. 값의 타입은 무시한다.
assert.strictEqual(actual, expected[, message])	주어진 actual 값이 expected와 다를 경우 오류가 발생. 값의 타입을 검사한다.
assert.notEqual(actual, expected[, message])	주어진 actual 값이 expected와 같을 경우 오류가 발생. 값의 타입은 무시한다.
assert.notStrictEqual(actual, expected[, message])	주어진 actual 값이 expected와 같을 경우 오류가 발생. 값의 타입을 검사한다.
assert.deepEqual(actual, expected[, message])	주어진 actual 객체가 expected 객체와 동일하지 않을 경우 오류 발생. 값의 타입은 무시한다.
assert.deepStrictEqual(actual, expected[, message])	주어진 actual 객체가 expected 객체와 동일하지 않을 경우 오류 발생. 값의 타입도 검사한다.
assert.notDeepEqual(actual, expected[, message])	주어진 actual 객체가 expected 객체와 동일할 경우 오류 발생. 값의 타입은 무시한다.
assert.notDeepStrictEqual(actual, expected[, message])	주어진 actual 객체가 expected 객체와 동일할 경우 오류 발생. 값의 타입도 검사한다.

Assert

- Assert 제공함수

함수	내용
assert.throws(block, [error], [message])	주어진 block에서 에러가 발생하지 않았을 경우 오류 발생
assert.doesNotThrow(block, [message])	주어진 block에서 에러가 발생했을 경우 오류 발생
assert.fail([message])	무조건 오류를 발생시킬 경우 씀

Assert

- Assert 예제 - 1

```
const assert = require("assert");

let v1 = 10;
let v2 = 10;
let v3 = 20;
let v4 = '10';

// assert

assert(v1 == v2);
console.log('v1과 v2는 같습니다.');

// assert(v1 == v3); // error
// console.log('v1과 v3는 같습니다.');

assert(v1 == v4);
console.log('v1과 v4는 같습니다.');

// assert(v1 - v2); // error
// console.log("v1 - v2는 0이 아닙니다.");

assert(v1 - v3);
console.log("v1 - v4는 0이 아닙니다.");
```

```
// assert.equal

assert.equal(v1, v2);
console.log('v1과 v2는 같습니다.');

// assert.equal(v1, v3); // error
// console.log('v1과 v3는 같습니다.');

assert.equal(v1, v4);
console.log('v1과 v4는 같습니다.');


// assert.strictEqual

assert.strictEqual(v1, v2);
console.log('v1과 v2는 같습니다.');

// assert.strictEqual(v1, v4); // error
// console.log('v1과 v4는 같습니다.');
```

```
// assert.notEqual

// assert.notEqual(v1, v2); // error
// console.log('v1과 v2는 다릅니다.');

assert.notEqual(v1, v3);
console.log('v1과 v3는 다릅니다.');

// assert.notEqual(v1, v4); // error
// console.log('v1과 v4는 다릅니다.');


// assert.notStrictEqual

// assert.notStrictEqual(v1, v2); // error
// console.log('v1과 v2는 다릅니다.');

assert.notStrictEqual(v1, v3);
console.log('v1과 v3는 다릅니다.');

assert.notStrictEqual(v1, v4);
console.log('v1과 v4는 다릅니다.');
```

Assert

- Assert 예제 - 2

```
const assert = require("assert");

let obj1 = {
    a1 : 10,
    a2 : 20
}
let obj2 = {
    a1 : 10,
    a2 : 20
}
let obj3 = {
    a1 : 10,
    a2 : 30
}
let obj4 = {
    a1 : 10,
    a2 : '20'
}
```

```
// assert.deepEqual

assert.deepEqual(obj1, obj2);
console.log("obj1과 obj2는 같습니다.");

// assert.deepEqual(obj1, obj3); // error
// console.log("obj1과 obj3는 같습니다.");

assert.deepEqual(obj1, obj4);
console.log("obj1과 obj4는 같습니다.");


// assert.deepStrictEqual

assert.deepStrictEqual(obj1, obj2);
console.log("obj1과 obj2는 같습니다.");

// assert.deepStrictEqual(obj1, obj3); // error
// console.log("obj1과 obj3는 같습니다.");

// assert.deepStrictEqual(obj1, obj4); // error
// console.log("obj1과 obj4는 같습니다.");
```

```
// assert.notDeepEqual

// assert.notDeepEqual(obj1, obj2); // error
// console.log("obj1과 obj2는 다릅니다.");

assert.notDeepEqual(obj1, obj3);
console.log("obj1과 obj3는 다릅니다.");

// assert.notDeepEqual(obj1, obj4); // error
// console.log("obj1과 obj4는 다릅니다.");


// assert.notDeepStrictEqual

// assert.notDeepStrictEqual(obj1, obj2); // error
// console.log("obj1과 obj2는 다릅니다.");

assert.notDeepStrictEqual(obj1, obj3);
console.log("obj1과 obj3는 다릅니다.");

assert.notDeepStrictEqual(obj1, obj4);
console.log("obj1과 obj4는 다릅니다.");
```

Assert

- Assert 예제 - 3

```
const assert = require("assert");

assert.throws(
    function() {
        throw new Error("Wrong value");
        // console.log("currect value"); // error
    },
    Error
);


assert.doesNotThrow(
    function() {
        console.log("Nothing to see here");
        // throw new Error("Something to see here"); // error
    },
    Error
);


// assert.fail("error!"); // error
```

Cluster

- Cluster
  - V8 엔진에서 제공하는 병렬처리에 사용되는 모듈
  - web worker와 동일한 기능을 한다.
  - 싱글 스레드 상에서 분산 처리를 가능하게 만들어주는 모듈
  - web worker 와 동일하지만 기능을 구현하는 함수는 web worker 와는 약간 상이하므로 node.js의 스펙과 web의 스펙을 다르게 구현해야 한다.
  - 여기서 master와 worker로 나뉘는 것을 볼 수 있는데 master는 메인으로 실행되는 프로세스를 의미하며 worker는 master에서 생성한 worker 객체들을 의미한다.

Cluster

- Cluster 주요 함수

함수	내용
disconnect()	모든 worker들을 종료한다.
exitedAfterDisconnect	worker가 연결을 끊고 종료할 경우 true를 리턴한다.
fork()	master 에서 새로운 worker를 생성한다.
id	worker 의 id를 리턴한다.
isConnected	master에 연결되어 있으면 true 아니면 false
isDead	worker 가 죽었으면 true 아니면 false
isMaster	현재 프로세스가 master 면 true 아니면 false
isWorker	현재 프로세스가 worker 면 true 아니면 false

Cluster

- Cluster 주요 함수

함수	내용
kill()	현재 worker 를 제거한다.
process	전역 자식 Process 를 리턴한다.
schedulingPolicy	schedulingPolicy 를 설정하거나 리턴한다.
send()	master 나 worker 로 메시지를 보낸다.
settings	cluster 의 세팅이 포함되어 있는 object 를 리턴한다.
setupMaster()	cluster 의 세팅을 바꾼다.
worker	현재의 worker object 를 리턴한다.
workers	master 에 포함된 모든 worker 들을 리턴한다.

Cluster

- Cluster 함수 예제

```
var cluster = require('cluster');

if (cluster.isWorker) {
  console.log('I am a worker');
} else {
  console.log('I am a master');
  cluster.fork();
  cluster.fork();
}
```

```
I am a master
I am a worker
I am a worker
```

Cluster

- Cluster 함수 예제

```
const cluster = require("cluster");

// SCHED_RR : Round Robin 방식으로 세팅
// SCHED_NONE : OS 설정에 맡김
cluster.schedulingPolicy = cluster.SCHED_RR;

// cluster.isMaster : 마스터 클러스터인지 확인
// cluster.isWorker : 워커 클러스터인지 확인
if(cluster.isMaster == true){

    // 3개의 작업을 병렬처리 함
    cluster.fork();
    cluster.fork();
    cluster.fork();

    cluster.on("online", function(worker){
        setInterval(() => {
            console.log(worker.process.pid, "동작");
        }, 1000);
    });
}
```

```
6192 동작
2336 동작
11512 동작
6192 동작
2336 동작
11512 동작
6192 동작
2336 동작
11512 동작
6192 동작
2336 동작
11512 동작
```

Crypto

- Crypto 모듈
  - node.js 에서 데이터 암호화 기능을 제공하는 모듈
  - 현재 존재하는 대부분의 암호화 알고리즘을 지원하고 있다.
  - Crypto 모듈은 다음과 같이 생성한다.

```
const crypto = require("crypto");
```

Crypto

- Crypto 모듈 제공 함수

함수	내용
constants	암호화 상수를 포함하는 object 반환
fips	FIPS 암호화 공급자가 사용 중인지 확인
createCipher()	특정 알고리즘 및 암호를 사용하여 암호 객체 생성
createCipheriv()	특정 알고리즘, 암호 및 초기화 벡터를 사용하여 암호 개체 생성
createDecipher()	특정 알고리즘 및 암호를 사용하여 해독 개체 생성
createDecipheriv()	특정 알고리즘, 암호 및 초기화 벡터를 사용하여 해독 개체 생성
createDiffieHellman()	DiffieHellman 키 교환 객체 생성
createECDH()	Elliptic Curve Diffie Hellmann 키 교환 객체 생성
createHash()	지정된 알고리즘을 사용하여 해시 개체 생성

Crypto

- Crypto 모듈 제공 함수

함수	내용
createHmac()	지정된 알고리즘 및 키를 사용하여 Hmac 객체 생성
createSign()	지정된 알고리즘 및 키를 사용하여 Sign(서명) 개체 생성
createVerify()	지정된 알고리즘을 사용하여 Verify(확인) 개체 생성
getCiphers	지원되는 암호 알고리즘의 배열을 반환함
getCurves()	지원되는 타원 곡선 배열 반환
getDiffieHellman()	미리 정의된 Diffie Hellman 키 교환 개체 반환
getHashes()	지원되는 해시 알고리즘 배열을 반환함
pbkdf2()	암호 기반 키 파생 함수 2 구현 생성
pbkdf2Sync()	동기식 비밀번호 기반 키 파생 기능 2 구현

Crypto

- Crypto 모듈 제공 함수

함수	내용
privateDecrypt()	개인 키를 사용하여 데이터 암호 해독
timingSafeEqual()	두 버퍼를 비교하여 true면 동일, 그렇지 않으면 false를 반환
privateEncrypt()	개인 키를 이용하여 데이터 암호화
publicDecrypt()	공개키를 이용하여 데이터 복호화
publicEncrypt()	공개키를 이용하여 데이터 암호화
randomBytes()	랜덤 데이터 생성
setEngine()	일부 또는 모든 OpenSSL 기능에 대해 엔진 설정
update()	데이터를 암호화 하거나 복호화
final()	암호화된 데이터에 마지막 종료 블록을 추가

Crypto

- Crypto 예제

```
const crypto = require('crypto');

let ciphers = crypto.getCiphers();

// 지원하는 모든 암호화 방식 보여주기
for(let x of ciphers){
    console.log(x);
}
```

```
aes-128-cbc
aes-128-cbc-hmac-sha1
aes-128-cbc-hmac-sha256
aes-128-ccm
aes-128-cfb
aes-128-cfb1
aes-128-cfb8
aes-128-ctr
aes-128-ecb
aes-128-gcm
aes-128-ocb
aes-128-ofb
aes-128-xts
aes-192-cbc
aes-192-ccm
```

```
var crypto = require('crypto');

// 암호 객체 생성
var mykey = crypto.createCipher('aes-128-cbc', 'mypassword');
var mystr = mykey.update('abc', 'utf8', 'hex')
mystr += mykey.final('hex');

console.log(mystr);

// 복호 객체 생성
var mykey = crypto.createDecipher('aes-128-cbc', 'mypassword');
var mystr = mykey.update('34feb914c099df25794bf9ccb85bea72', 'hex', 'utf8')
mystr += mykey.final('utf8');

console.log(mystr);
```

```
34feb914c099df25794bf9ccb85bea72
abc
```

dns

- dns 모듈
  - 지정된 도메인의 dns 정보를 알아올 수 있는 모듈이다.
  - 지정된 도메인의 ip 주소 등의 정보를 파악할 수 있다.
  - dns는 다음과 같이 선언해서 사용이 가능하다.

```
var dns = require('dns');
```

dns

- dns 모듈 함수

함수	내용
getServers()	현재 서버에 속한 모든 IP 주소를 포함하는 배열을 반환함
lookup()	호스트 이름으로 정보를 찾는다. 콜백 함수는 IP 주소를 포함한 호스트 이름에 대한 정보를 포함한다.
lookupService()	IP주소와 포트로 정보를 검색한다. 콜백 함수는 호스트 이름과 같은 주소에 대한 정보를 포함한다.
resolve()	지정된 호스트 이름에 속하는 레코드 유형의 배열을 반환함
resolve4()	IPv4 주소를 조회한다. 콜백 함수는 IPv4 주소 배열을 포함한다.
resolve6()	IPv6 주소를 조회한다. 콜백 함수는 IPv6 주소 배열을 포함한다.
resolveCname()	지정된 호스트 이름에 대한 CNAME 레코드를 조회한다. 콜백 함수는 호스트 이름에 사용할 수 있는 도메인 배열을 포함한다.
resolveMx()	지정된 호스트 이름에 대한 메일 교환 레코드를 조회한다.

dns

- dns 모듈 함수

함수	내용
resolveNaptr()	지정된 호스트 이름에 대한 정규식 기반 레코드를 검색한다.
resolveNs()	지정된 호스트 이름에 대한 이름 서버 레코드를 조회한다.
resolveSoa()	지정된 호스트 이름에 대한 권한 시작 레코드를 조회한다.
resolveSrv()	지정된 호스트 이름에 대한 서비스 레코드를 조회한다.
resolvePtr()	지정된 호스트 이름에 대한 포인터 레코드를 조회한다.
resolveTxt()	지정된 호스트 이름에 대한 텍스트 쿼리 레코드를 검색한다.
reverse()	IP 주소를 호스트 이름 배열로 되돌리기
setServers()	서버의 IP 주소를 설정한다.

dns

- dns 예제

```
var dns = require('dns');

var w3 = dns.lookup('www.w3schools.com', function (err, addresses, family) {
  console.log(addresses); // 주소
  console.log(family); // 버전
});
```

```
192.229.179.87
4
```

fs

- fs(file system) 모듈
  - node.js에서 파일에 데이터를 쓰고 읽어올 수 있는 기능을 제공하는 모듈
  - fs 모듈에는 상당히 많은 기능들이 존재하는데 권한관련 기능들이 많이 존재한다.

fs

- fs(file system) 모듈 함수

함수	설명
access()	사용자가 이 파일 또는 디렉토리에 액세스할 수 있는지 확인
accessSync()	access()와 동일하지만 비동기식이 아닌 동기식으로 수행
appendFile()	파일에 데이터 추가
appendFileSync()	appendFile()와 동일하지만 비동기식이 아닌 동기식
chmod()	파일의 모드 변경
chmodSync()	chmod()와 동일하지만 비동기식이 아닌 동기식
chown()	파일의 소유자 변경
chownSync()	chown()와 동일하지만 비동기식이 아닌 동기식
close()	파일 닫기
closeSync()	close() 와 동일하지만 비동기식이 아닌 동기식
constants	파일 시스템의 상수 값을 포함하는 객체 반환
createReadStream()	읽기 가능한 개체 반환
createWriteStream()	쓰기 가능한 개체 반환
existsSync()	파일 또는 폴더가 있는지 확인
fchmod()	파일의 모드 변경
fchmodSync()	fchmod() 와 동일하지만 비동기식이 아닌 동기식
fchown()	파일의 소유자 변경
fchownSync()	fchown() 와 동일하지만 비동기식이 아닌 동기식
fdatasync()	컴퓨터에 저장한 파일과 동기화
fdatasyncSync()	fdatasync() 와 동일하지만 비동기식이 아닌 동기식
fstat()	파일 상태 변환

fs

- fs(file system) 모듈 함수

함수	설명
fstatSync()	fstat() 와 동일하지만 비동기식이 아닌 동기식
fsync()	컴퓨터에 저장된 파일과 동기화
fsyncSync()	fsync() 와 동일하지만 비동기식이 아닌 동기식
ftruncated()	파일 잘라내기
ftruncatedSync()	ftruncated() 와 동일하지만 비동기식이 아닌 동기식
futimes()	파일의 timestemp 변경
futimesSync()	futimes() 와 동일하지만 비동기식이 아닌 동기식
lchmod()	Mac OS X 파일 모드 변경
lchmodSync()	lchmod() 와 동일하지만 비동기식이 아닌 동기식
lchown()	Mac OS X 파일 소유자 변경
lchownSync()	lchown() 와 동일하지만 비동기식이 아닌 동기식
link()	링크 파일을 만든다.
linksync()	link() 와 동일하지만 비동기식이 아닌 동기식
lstat()	파일 상태 반환
lstatSync()	lstat() 와 동일하지만 비동기식이 아닌 동기식
mkdir()	새 디렉터리 만들기
mkdirSync()	mkdir() 와 동일하지만 비동기식이 아닌 동기식
mkdtemp()	새 임시 디렉토리 만들기
mkdtempSync()	mktemp() 와 동일하지만 비동기식이 아닌 동기식
open()	파일 열기
openSync()	open() 와 동일하지만 비동기식이 아닌 동기식
read()	파일 내용 읽기

fs

- fs(file system) 모듈 함수

함수	설명
readdir()	디렉토리의 내용 읽기
readdirSync()	readdir() 와 동일하지만 비동기식이 아닌 동기식
readFile()	파일 내용 읽기
readFileSync()	readFile() 와 동일하지만 비동기식이 아닌 동기식
readlink()	링크 값 읽기
readlinkSync()	readlink() 와 동일하지만 비동기식이 아닌 동기식
realpath()	절대 경로 이름 반환
realpathSync()	realpath() 와 동일하지만 비동기식이 아닌 동기식
rename()	파일 이름 바꾸기
renameSync()	rename() 와 동일하지만 비동기식이 아닌 동기식
rmdir()	디렉토리 제거
rmdirSync()	rmdir() 와 동일하지만 비동기식이 아닌 동기식
stat()	파일의 상태 반환
statSync()	stat() 와 동일하지만 비동기식이 아닌 동기식
symlink()	파일의 심볼 이름 만들기
symlinkSync()	symlink() 와 동일하지만 비동기식이 아닌 동기식
truncate()	파일 잘라내기
truncateSync()	truncate() 와 동일하지만 비동기식이 아닌 동기식
unlink()	링크 제거
unlinkSync()	unlink() 와 동일하지만 비동기식이 아닌 동기식
unwatchFile()	파일 이름에 대한 변경 내용 보기 중지
utimes()	파일의 타임스템프 변경

fs

- fs(file system)
  모듈 함수

함수	설명
utimesSync()	utimes() 와 동일하지만 비동기식이 아닌 동기식
watch()	파일 이름 또는 디렉터리 이름 변경 감시
watchFile()	파일 이름 변경 감시
write()	파일에 버퍼 쓰기
write()	파일에 데이터 쓰기
writeFile()	파일에 데이터 쓰기
writeFileSync()	writeFile() 와 동일하지만 비동기식이 아닌 동기식
writeSync()	write() 와 동일하지만 비동기식이 아닌 동기식 파일에 버퍼 쓰기
writeSync()	write() 와 동일하지만 비동기식이 아닌 동기식 파일에 버퍼 쓰기

fs

- fs 예제 - 1

```
var fs = require("fs");

// fs.writeFile(__dirname+'/data/data1.txt', 'hello node.js', function(error){
//     console.log("비 동기식 저장 1");
// });

// fs.writeFile(__dirname+'/data/data2.txt', 'hello node.js', function(error){
//     console.log("비 동기식 저장 2")
// });

fs.appendFile(__dirname+'/data/data1.txt','안녕하세요',function(error){
    console.log('비 동기식 추가 1');
});

fs.appendFile(__dirname+'/data/data2.txt','반갑습니다',function(error){
    console.log('비 동기식 추가 2');
});

fs.readFile(__dirname+'/data/data1.txt',function(error, data){
    console.log("data1 : ", data.toString());
});

fs.readFile(__dirname+'/data/data2.txt',function(error, data){
    console.log("data2 : ", data.toString());
});
```

```
비 동기식 추가 1
비 동기식 추가 2
data1 :  hello node.js안녕하세요
data2 :  hello node.js반갑습니다
```

fs

- fs 에제 - 2

```
var fs = require("fs");

fs.writeFileSync(__dirname+'/data/data3.txt', 'hello node.js');
console.log("동기식 저장 1");

fs.writeFileSync(__dirname+'/data/data4.txt', 'hello node.js');
console.log("동기식 저장 2")

fs.appendFileSync(__dirname+'/data/data3.txt', '안녕하세요')
console.log('파일 내용 추가 1');

fs.appendFileSync(__dirname+'/data/data4.txt', '반갑습니다')
console.log('파일 내용 추가 2');

var data3 = fs.readFileSync(__dirname+'/data/data3.txt');
console.log('data3 : ', data3.toString());

var data4 = fs.readFileSync(__dirname+'/data/data4.txt');
console.log('data4 : ', data4.toString());
```

```
동기식 저장 1
동기식 저장 2
파일 내용 추가 1
파일 내용 추가 2
data3 :  hello node.js안녕하세요
data4 :  hello node.js반갑습니다
```

fs

- fs 예제 - 3

```
var fs = require("fs");
// 스트림을 활용한 읽고 쓰기
const writeStream = fs.createWriteStream(__dirname+'/data/data5.txt');
writeStream.on('finish', ()=>{
    console.log('파일 쓰기 완료');
});

writeStream.write('스트림을 이용해 데이터를 전달하게 되면\n');
writeStream.write('큰 메모리 용량 필요 없이 파일 전송이 가능해집니다.\n');
writeStream.end();
```

```
var fs = require("fs");
// 스트림을 활용한 읽기 highWaterMark는 버퍼의 크기를 지정해 주는 속성
const readStream = fs.createReadStream(__dirname+'/data/data5.txt',{highWaterMark : 16});
const data = [];

// 파일 읽기가 시작될 경우 발생하는 이벤트
readStream.on('data', (chunk) =>{
    data.push(chunk);
    console.log('data : ', chunk, chunk.length);
});

// 파일을 다 읽을경우 발생하는 이벤트
readStream.on('end', () =>{
    console.log('end : ', Buffer.concat(data).toString());
});

// 읽는 도중 에러 날 경우 발생하는 이벤트
readStream.on('error', (err) => {
    console.log('error : '+err);
});
```

```
data :  Buffer(16) [236, 138, 164, 237, 138, 184, 235, 166, 188, 236, 157, 132, 32, 236, 157, 180] 16
data :  Buffer(16) [236, 154, 169, 237, 149, 180, 32, 235, 141, 176, 236, 157, 180, 237, 132, 176] 16
data :  Buffer(16) [235, 165, 188, 32, 236, 160, 132, 235, 139, 172, 237, 149, 152, 234, 178, 140] 16
data :  Buffer(16) [32, 235, 144, 152, 235, 169, 180, 10, 237, 129, 176, 32, 235, 169, 148, 235] 16
data :  Buffer(16) [170, 168, 235, 166, 172, 32, 236, 154, 169, 235, 159, 137, 32, 237, 149, 132] 16
data :  Buffer(16) [236, 154, 148, 32, 236, 151, 134, 236, 157, 180, 32, 237, 140, 140, 236, 157] 16
end :  스트림을 이용해 데이터를 전달하게 되면
큰 메모리 용량 필요 없이 파일 전송이 가능해집니다.
```

fs

- fs 예제 - 4

```
var fs = require("fs");

// 파일 복사
const readStream = fs.createReadStream(__dirname+'/data/data5.txt');
const writeStream = fs.createWriteStream(__dirname+'/data/data6.txt');
readStream.pipe(writeStream);
```

process

- process 모듈
  - process 모듈은 require()를 사용하지 않고도 Node.js 애플리케이션에 접근할 수 있는 전역객체 이다.
  - 동작중인 프로세스에 접근할 수 있는 권한을 제공한다.
  - process 모듈은 동작하는 프로세스 및 시스템 아키텍처에 관한 풍부한 정보를 가지고 있다.

process

- process 예제 - 1

```
console.log(process.env);           // 컴퓨터 환경과 관련된 정보를 가진 객체
console.log(process.version);       // node.js의 버전
console.log(process.versions);      // node.js와 연관된 프로그램들의 버전을 가진 객체
console.log(process.arch);          // 프로세서의 아키텍처(arm/ia32/x64)
console.log(process.platform);      // 플랫폼(win32/linux/sunos/freebsd/darwin)
console.log(process.memoryUsage()); // 메모리 사용 정보를 가진 객체
console.log(process.uptime());      // 현재 프로그램이 실행된 시간
```

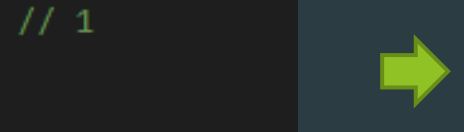
```
{ALLUSERSPROFILE: 'C:\ProgramData', AMD_ENTRYPOINT: 'vs/workbench/services/extensions/nod
e/extensionHostProcess', APPDATA: 'C:\Users\User\AppData\Roaming', APPLICATION_INSIGHTS_N
O_DIAGNOSTIC_CHANNEL: 'true', CommonProgramFiles: 'C:\Program Files\Common Files'}
v12.18.3
{node: '12.18.3', v8: '7.8.279.23-node.39', uv: '1.38.0', zlib: '1.2.11', brotli: '1.0.
7'}
x64
win32
0.3253298
```

process

- process 예제 - env
  - process.env는 서비스의 중요한 키를 저장하는 공간으로 사용된다.

```
process.env.TEST = 1;
console.log(process.env.TEST); // 1

delete process.env.TEST;
console.log(process.env.TEST); // undefined
```



```
1
undefined
```

process

- process 예제 - argv
  - 프로그램의 매개변수를 호출할 때 쓰는 파라미터

```
process.argv.forEach(function(val, index, array) {
    console.log(index + ': ' + val);
});
```

```
PS C:\Users\User\OneDrive\수업\수업\node.js\v2.0\source\ch04> node 13_process3.js one two three
0: C:\Program Files\nodejs\node.exe
1: C:\Users\User\OneDrive\수업\수업\node.js\v2.0\source\ch04\13_process3.js
2: one
3: two
4: three
```

process

- process 예제 - exit
  - 실행중인 node 프로세스를 종료할 때 쓰는 함수.
  - 독립적인 프로그램에서 수동으로 노드를 멈추게 하기 위해 사용한다.

```
let i = 1;
setInterval(() => {
    if(i===5){
        console.log('종료!');
        process.exit();
    }
    console.log(i);
    i += 1;
}, 1000);
```

```
1
2
3
4
종료!
```

OS

- os 모듈
  - os 모듈은 애플리케이션에서 많이 사용되는 모듈은 아니지만 운영체제와 시스템의 정보를 가져올 수 있는 모듈이다.
  - 따라서 실행 환경에 따라서 결과 값이 다르게 나올 수 있다.

```
var os = require('os');

console.log(os.tmpdir());        // 임시 저장 폴더의 위치
console.log(os.endianness());    // CPU의 endianness(BE 또는 LE)
console.log(os.hostname());      // 호스트 이름(컴퓨터 이름)
console.log(os.type());          // 운영체제 이름
console.log(os.platform());      // 운영체제 플랫폼
console.log(os.arch());          // 운영체제 아키텍처
console.log(os.release());       // 운영체제 버전
console.log(os.uptime());        // 운영체제가 실행된 시간
console.log(os.loadavg());       // 로드 에버리지 정보를 담은 배열
console.log(os.totalmem());      // 시스템의 총 메모리
console.log(os.freemem());       // 시스템의 가용 메모리
console.log(os.cpus());          // CPU의 정보를 담은 객체
console.log(os.networkInterfaces()); // 네트워크 인터페이스 정보를 담은 배열
console.log(os.EOL);             // 운영체제의 개행 문자(\n 이나 \r\n 같은 문자)
```

```
C:\Users\User\AppData\Local\Temp
LE
DESKTOP-RCIL9HI
Windows_NT
win32
x64
10.0.19041
22274
(3) [0, 0, 0]
17086562304
9762811904
```

path

- path 모듈
  - 폴더와 파일의 경로를 쉽게 조작하도록 도와주는 모듈
  - OS마다 경로구분자가 다르기 때문에 path모듈이 필요함
    - window : \로 구분
    - POSIX : /로 구분

path

- path 모듈 함수

함수	설명
path.sep	경로 구분자. windo는 \ , POSIX는 /
path.delimiter	환경 변수의 구분자. process.env.PATH를 입력하면 여러 개의 경로가 이 구분자로 되어 있음. Windows는 세미콜론(;) 이고 POSIX는 콜론(:) 임
path.dirname(경로)	파일이 위치한 폴더 경로를 보여준다.
path.extname(경로)	파일의 확장자를 보여준다.
path.basename(경로, 확장자)	파일의 이름(확장자 포함)을 보여준다. 파일의 이름만 표시하고 싶다면 basename의 두 번째 인자로 파일의 확장자를 넣어주면 된다.
path.parse(경로)	파일 경로를 root, dir, base, ext, name 으로 분리한다.
path.format(객체)	path.parse() 한 객체를 파일 경로로 합친다.
path.normalize(경로)	/나 \를 실수로 여러 번 사용했거나 혼용했을 때 정상적인 경로로 변환해준다.
path.isAbsolute(경로)	파일의 경로가 절대경로인지 상대경로인지 true나 false로 알려준다.
path.relative(기준경로, 비교경로)	경로를 두 개 넣으면 첫 번째 경로에서 두 번째 경로로 가는 방법을 알려준다.

path

- path 예제

```
var path = require('path');

console.log(path.delimiter); // ;

var filename = path.basename('/Users/Refsnes/demo_path.js');
console.log(filename); // demo_path.js

var filename = path.basename('/Users/Refsnes/demo_path.js', '.js');
console.log(filename); // demo_path

var dirname = path.dirname('/Users/Refsnes/demo_path.js', '.js');
console.log(dirname);  // /Users/Refsnes

var ext = path.extname('/Users/Refsnes/demo_path.js');
console.log(ext);  // .js

var obj = { dir: 'C:\\Users\\Refsnes', base: 'demo_path.js' }
//Format the path object into a path string:
var p = path.format(obj);
console.log(p); // C:\Users\Refsnes\demo_path.js

console.log(path.isAbsolute('/test/demo_path.js')); //true
console.log(path.isAbsolute('test/demo_path.js')); //false
console.log(path.isAbsolute('C:\\test\\demo_path.js')); //true

var x = path.join('Users', 'Refsnes', 'demo_path.js');
console.log(x);  // Users\Refsnes\demo_path.js

var x = path.normalize('Users/Refsnes/../Jackson');
console.log(x);  // Users\Jackson
```

```
;
demo_path.js
demo_path
/Users/Refsnes
.js
C:\Users\Refsnes\demo_path.js
true
false
true
Users\Refsnes\demo_path.js
Users\Jackson
```

url

- url 모듈
  - 인터넷 주소를 쉽게 조작하도록 도와주는 모듈
  - url 모듈에서 제공하는 함수는 이전에 node.js에서 사용하던 방식과 WHATWG에서 제시한 표준으로 나뉠 수 있다.
  - url 모듈에서 제공하는 함수는 다음과 같다

함수	설명
url.parse(주소)	주소를 분해한다. 분해한 주소는 urlObject 형태로 리턴한다
url.format(객체)	분해되었던 url 객체를 원래 상태로 조립한다.
url.resolve(from, to)	url의 좌표를 바꾸거나 확인할 때 쓴다.

url

- url 모듈
  - parse를 사용하게 되면 url 객체에 대한 정보를 얻어올 수 있는데 얻어올 수 있는 정보는 다음과 같다.

함수	설명
urlObject.hash	# 뒤에 설정된 데이터를 긁어온다.
urlObject.host	호스트 이름과 포트번호를 반환한다.
urlObject.hostname	호스트 이름만 반환한다.
urlObject.pathname	호스트 이름을 제외한 경로를 반환한다.
urlObject.search	설정된 파라미터 이름과 값을 문자열 형태로 반환한다.
urlObject.href	전체 경로를 반환한다
urlObject.port	포트를 반환한다.
urlObject.protocol	프로토콜을 반환한다.
urlObject.query	설정된 파라미터 이름과 값을 객체 형태로 반환한다.

url

- url 모듈 예제 - 1

```
var url = require('url');
var adr = 'http://localhost:8080/abc/default.htm?year=2017&month=february#bob';
var q = url.parse(adr, true);

console.log(q.host); // localhost:8080
console.log(q.hostname)  // localhost
console.log(q.hash); // bob
console.log(q.pathname); // /abc/default.htm
console.log(q.search); // ?year=2017&month=february
console.log(q.href); // http://localhost:8080/abc/default.htm?year=2017&month=february#bob
console.log(q.port); // 8080
console.log(q.protocol); // http:
console.log(q.query); // { year: 2017, month: 'february' }


var qdata = q.query; // { year: 2017, month: 'february' }
console.log(qdata.month); // february
```

url

- url 모듈 예제 - 2

```
var url = require('url');

var x = url.format({
    protocol: 'https',
    hostname: 'example.com',
    pathname: '/some/path',
    query: {
        page: 1,
        format: 'json'
    }
});

// https://example.com/some/path?page=1&format=json
console.log(x.toString());
```

url

- url 객체
  - 객체 url을 선언하는 방식이 WHATWG에서 제공됨으로써 기존 방식에서 일부 기능이 변경되었다.
  - 객체에서 제공하는 함수는 위에서 parse를 통해 나온 내용과 크게 다르지 않다.
  - 단 차이점은 query 함수가 빠지고 대신 새로운 객체인 URLSearchParams 라는 객체가 들어오면서 기존 한 개에서 담당하던 기능을 두개로 분산시켰다.
  - 또한 일부 데이터를 바꿀 수 있도록 수정 가능하게 되면서 기존 url을 그대로 재활용 할 수 있게 되었다.

url

- url 예제 - 1

```
var url = require('url');
var q = new URL('http://localhost:8080/abc/default.htm?year=2017&month=february#bob');


console.log(q.host); // localhost:8080
console.log(q.hostname)  // localhost
console.log(q.hash); // bob
console.log(q.pathname); // /abc/default.htm
console.log(q.search); // ?year=2017&month=february
console.log(q.href); // http://localhost:8080/abc/default.htm?year=2017&month=february#bob
console.log(q.port); // 8080
console.log(q.protocol); // http:
```

url

- url 예제 - 2

```
var url = require('url');
var q = new URL('http://localhost:8080/abc/default.htm?year=2017&month=february#bob');

q.hostname = 'example';
q.hash = 'baz';
q.pathname = '/abc/efg/default.html'
q.search = '?year=2017&month=february&abc=xyz'
q.port = 8081;
q.protocol = 'https';
console.log(q.href); // https://example:8081/abc/efg/default.html?year=2017&month=february&abc=xyz#baz
```

url

- URLSearchParams
  - URLSearchParams 는 쿼리 부분의 읽기 및 쓰기에 대한 엑세스를 제공한다.
  - URLSearchParams 는 URL에서 searchParams라는 함수를 통해 얻을 수 있다.
  - URLSearchParams 에서 제공하는 함수는 다음과 같다.

함수	설명
getAll(key)	키에 해당하는 모든 값들을 가져온다.
get(key)	키에 해당하는 첫번째 값을 가져온다.
has(key)	해당 키가 존재하는지 확인한다.
keys()	searchParams의 모든 키를 반복키 객체로 가져온다.
values()	searchParams의 모든 값을 반복키 객체로 가져온다.
append(key, value)	해당 키와 값을 추가한다. 같은 키가 존재할 경우 해당 키의 값을 유지하고 하나를 더 추가한다.
set(key, value)	해당 키와 값을 추가한다. 같은 키가 존재할 경우 해당 키의 값을 제거하고 새로운 값을 대체한다.
delete(key)	해당 키에 해당되는 키와 값을 제거한다.
toStrinng()	조작한 searchParam 객체를 다시 문자열로 만든다. 이 문자열을 search에 대입하면 주소 객체에 반영된다.

url

- URLSearchParams 예제

```
var url = require('url');
var q = new URL('http://localhost:8080/abc/default.htm?year=2017&month=february&month=may');

var myParam = q.searchParams;

console.log(myParam.getAll('month')); // ['february', 'may']
console.log(myParam.get('month')); // february
console.log(myParam.has('year')); // true

for(var key of myParam.keys()) {
    console.log(key);
}
// year
// month
// month
for(var value of myParam.values()) {
    console.log(value);
}
// 2017
// february
// may

setTimeout(function(){
    myParam.append('filter','es3');
    myParam.append('filter','es5');
    console.log(myParam.getAll('filter')); // ['es3', 'es5']
}, 1000);

setTimeout(function(){
    myParam.set('filter','es6');
    console.log(myParam.getAll('filter')); // ['es6']
}, 2000);

setTimeout(function(){
    myParam.delete('filter');
    console.log(myParam.getAll('filter')); // []
}, 3000);

console.log(myParam.toString()); // year=2017&month=february&month=may
```

Events

- events 모듈
  - 자체적으로 이벤트를 만들 수 있게 해주는 모듈
  - 지금까지는 각 모듈에 정해진 이벤트만을 사용해야 했지만 사용자 정의에 맞추어 이벤트를 만들 수 있다.
  - Events 모듈에서 제공하는 함수는 다음과 같다.

함수	설명
on(eventname, callback)	이벤트 이름과 이벤트 발생 시의 콜백 함수를 연결해준다. 이렇게 연결하는 동작을 이벤트 리스닝이라고 한다.
addListener(eventname, callback)	on과 동일한 기능
emit(eventname)	이벤트를 호출하는 메서드. 이벤트 이름을 인자로 넣어주면 미리 등록해뒀던 이벤트 콜백이 실행된다.
once(eventname, callback)	한 번만 실행되는 이벤트를 리스닝 시킨다. emit을 통해 호출되면 두번 이상 호출되지 않는다.
removeAllListeners(eventname)	이벤트에 연결된 모든 이벤트 리스너를 제거한다.
removeListener(eventname, listener)	이벤트에 연결된 리스너를 하나씩 제거한다.
off(eventname, callback)	removeListener와 동일
listenerCount(eventname)	현재 이벤트에 리스너가 몇 개 연결되어 있는지 확인한다.

Events

- events 모듈 예제 - 1

```
const EventEmitter = require('events');

const myEvent = new EventEmitter();
myEvent.addListener('event1', () => {
  console.log('이벤트 1');
});
myEvent.on('event2', () => {
  console.log('이벤트 2');
});
myEvent.on('event2', () => {
  console.log('이벤트 2 추가');
});
myEvent.once('event3', () => {
  console.log('이벤트 3');
}); // 한 번만 실행됨

myEvent.emit('event1'); // 이벤트 호출
myEvent.emit('event2'); // 이벤트 호출

myEvent.emit('event3');
myEvent.emit('event3'); // 실행 안 됨

myEvent.on('event4', () => {
  console.log('이벤트 4');
});
myEvent.removeAllListeners('event4');
myEvent.emit('event4'); // 실행 안 됨

const listener = () => {
  console.log('이벤트 5');
};
myEvent.on('event5', listener);
myEvent.removeListener('event5', listener);
myEvent.emit('event5'); // 실행 안 됨

console.log(myEvent.listenerCount('event2'));
```

```
이벤트 1
이벤트 2
이벤트 2 추가
이벤트 3
2
```