

Introduction to MEAN

Samkeun Kim <skim@hknu.ac.kr>

<http://cyber.hankyong.ac.kr>

The evolution of JavaScript

JavaScript:

- 웹을 위해 만들어진 **인터프리터 방식**의 프로그래밍 언어
- **Netscape Navigator** 웹 브라우저에 의해 처음으로 구현
- 웹 브라우저가 **클라이언트 측 로직을 실행**하기 위해 사용하는 프로그래밍 언어
- 더 빠른 브라우저들이 출시됨에 따라 점차 더 복잡한 **JavaScript** 애플리케이션 개발
- 라이브러리 & 툴 생성 / 개발 사이클 단축 => 훨씬 더 진보한 웹 애플리케이션 세대 탄생

⇒ *끊임없이 더 빠른 브라우저 요구*

Google 크롬 브라우저:

- JIT-compiling **V8 JavaScript** 엔진 장착 (2008년)
- **V8** 엔진: **JavaScript**를 훨씬 더 빠르게 실행할 수 있게
 - ✓ 웹 애플리케이션 개발 방법을 완전히 변형
- **V8** 엔진의 소스 코드 오픈
 - ✓ **JavaScript**가 브라우저 **바깥**에서 동작할 수 있게 만들어 줌

⇒ 이 혁명의 첫 번째 작품이 **Node.js**이다!

Node.js

- Ryan Dahl
- Non-blocking I/O 실험에 V8 엔진이 적합하다는 것을 깨달음
- Non-blocking 코드 단위(모듈)를 제작할 수 있는 작지만 강력한 Platform 창안 [2009년]
 - ✓ JavaScript의 Non-blocking 기능을 브라우저 바깥에서 이용할 수 있도록

Node의 모듈 시스템

- 제 3자의 모듈을 이용 => **플랫폼을 자유롭게 확장할 수 있게**
- 온라인 커뮤니티 반응:
 - ✓ Modern 웹 프레임워크에서부터 Robotics 서버 플랫폼에 이르기까지 다양한 툴들의 생성으로 이어짐

⇒ “JavaScript가 브라우저를 벗어나서 서버 측에서 동작하기 시작했다!”

Installing Node.js

Download

<https://nodejs.org/en/download/>

적절한 .msi파일(**Windows** 64 비트 버전) 다운로드

NOTE

Node.js와 io.js 프로젝트를 병합 한 후 버전 체계는 0.12에서 4.x까지는 연속되었다. 지금은 장기 지원(LTS) 정책을 사용한다. https://en.wikipedia.org/wiki/Long-term_support에서 해당 내용을 볼 수 있다.

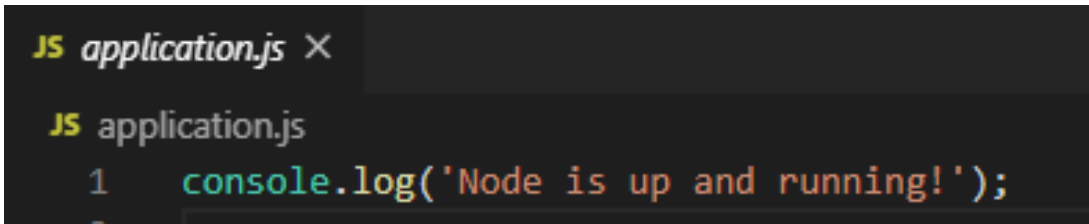
Installing Node.js on Windows

다운로드한 인스톨러 실행

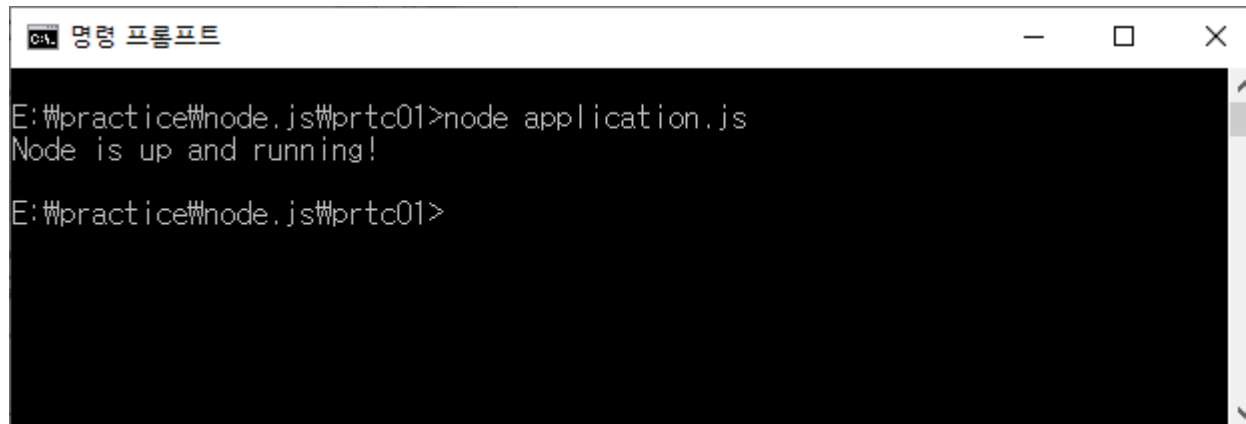


Running Node.js

Visual Studio Code에서 아래 코드를 포함하는 **application.js** 파일 생성하여 작업 폴더에 저장
(예: E:\practice\node.js\prtc01\)



```
JS application.js X
JS application.js
1 console.log('Node is up and running!');
```



```
C:\> 명령 프롬프트
E:\practice\node.js\prtc01>node application.js
Node is up and running!
E:\practice\node.js\prtc01>
```

Introduction to ECMAScript 2015

2009년 **ES5**가 나온 지 몇 년이 지난 후 **ES6** 스펙이 2015년 6월에 릴리즈되었다.

ES6는 JavaScript에 가장 커다란 변화를 제시했다.

즉 JavaScript 개발자들의 코딩하는 방식을 완전히 바꿔 놓은 특징들을 도입한 것이었다.

그래서 **ES2015**까지 만들어졌던 모든 특징들을 설명하기 보다는 앞으로 우리가 사용하게 될 기본 특징들을 살펴보겠다.

Modules

Node에서 모듈(module)은 **Language-level**에서 지원되는 특징이다.

⇒ 개발자들이 모듈 패턴으로 컴포넌트를 감쌀 수 있게 하고,
코드 내부에서 모듈을 export/import 할 수 있게 해 줌

⇒ ***export와 import***



The **CommonJS** module specification is the standard used in **Node.js** for working with modules.

간단한 예를 살펴보자. 아래 코드를 포함하는 **lib.js** 파일이 있다고 하자:

```
exports.halfOf = function(x) {  
  ...  
  return x / 2;  
};
```

⇒ 아래처럼 **main.js** 파일에서 호출 가능:

```
const hello = require('./lib');  
...  
console.log(hello.halfOf(84));
```



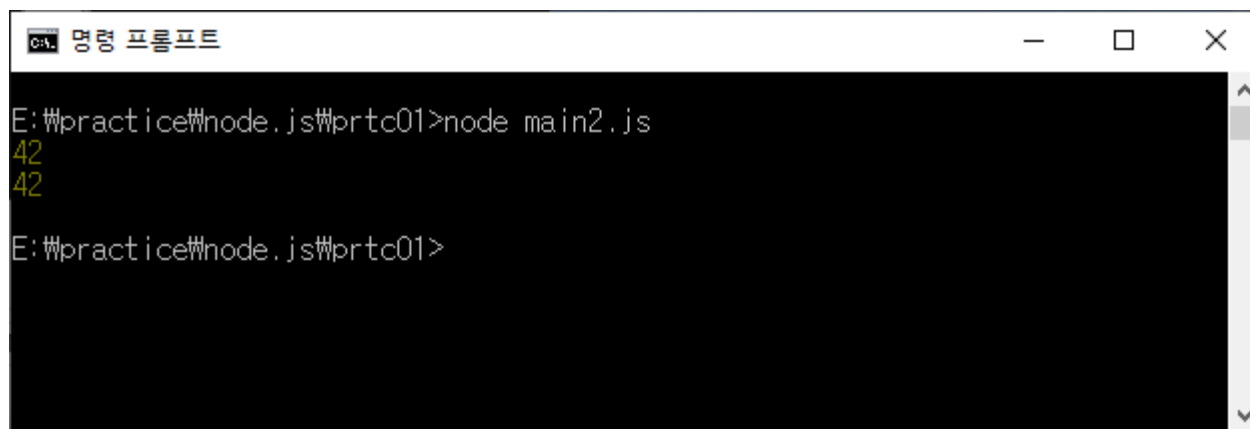
```
명령 프롬프트  
E:\practice\node.js\prtc01>node main.js  
42  
E:\practice\node.js\prtc01>
```

또 다른 예로, **lib2.js** 파일이 아래와 같다고 하자:

```
exports.halfOf = function(x) {  
  return x / 2;  
};  
  
exports.multiply = function(x, y) {  
  return x * y;  
};
```

⇒ **Main2.js** 파일에서 아래처럼 호출 가능:

```
const hello = require('./lib2');  
console.log(hello.halfOf(84));  
console.log(hello.multiply(21, 2));
```



A screenshot of a Windows Command Prompt window titled "명령 프롬프트". The window shows the execution of a Node.js script. The command prompt displays the path "E:\practice\node.js\prtc01" and the command "node main2.js". The output of the script is displayed as two lines of "42".

```
명령 프롬프트  
E:\practice\node.js\prtc01>node main2.js  
42  
42  
E:\practice\node.js\prtc01>
```

ES2015(ES6) 모듈은 디폴트 **export** 값도 지원한다.

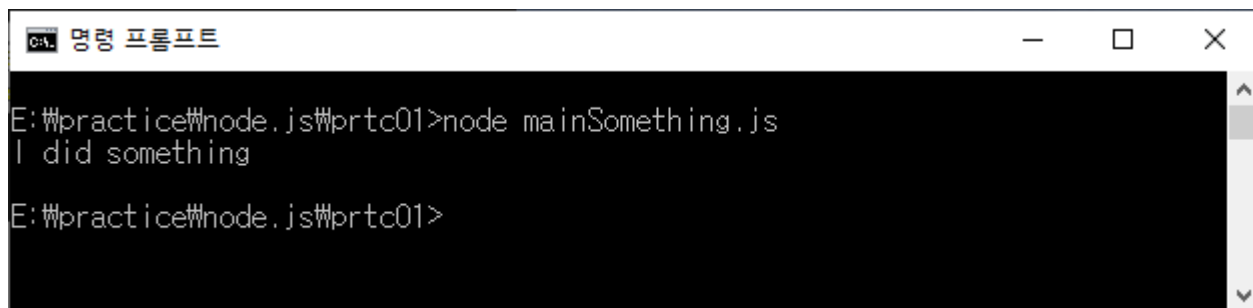
예를 들어, **doSomething.js**라는 파일에 아래 코드가 포함되어 있다고 가정해보자:

```
exports.default = function() {  
  console.log('I did something');  
}
```

⇒ **mainSomething.js** 파일에서 아래와 같이 사용할 수 있다:

```
const doSomething = require('./doSomething').default;  
doSomething();
```

디폴트 import는 모듈 이름을 사용하여 엔티티를 식별한다!



```
명령 프롬프트  
E:\practice\node.js\prtc01>node mainSomething.js  
I did something  
E:\practice\node.js\prtc01>
```

⇒ *모듈에 대해 이해했다면, 이제 **Class**로 옮겨가 보자.*

Classes

'Class versus Prototype'의 긴 논쟁

⇒ **Class**가 기본적으로 구조적인 측면에서 더 적합하다고 결론

Class

⇒ 인스턴스와 Static 멤버, 생성자, Super Call을 지원해 주는 **사용하기 쉬운 패턴**

예: *class-test.js*

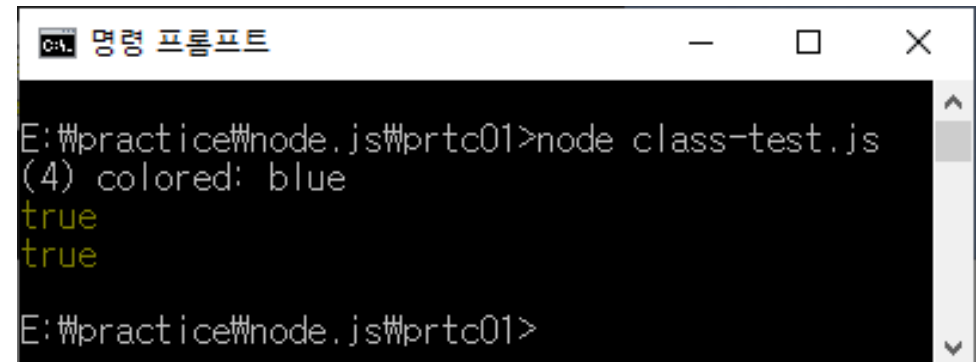
```
class Vehicle {
  constructor(wheels) {
    this.wheels = wheels;
  }
  toString() {
    return '(' + this.wheels + ')';
  }
}

class Car extends Vehicle {
  constructor(color) {
    super(4);
    this.color = color;
  }
  toString() {
    return super.toString() + ' colored: ' + this.color;
  }
}

let car = new Car('blue');
console.log(car.toString());

console.log( car instanceof Car);
console.log( car instanceof Vehicle);
```

Car 클래스가 Vehicle 클래스를 상속한다.



```
명령 프롬프트
E:\practice\node.js\prtc01>node class-test.js
(4) colored: blue
true
true
E:\practice\node.js\prtc01>
```

Arrow functions

Arrow(=>) syntax: **Function의 단축형**

Arrow 구문은 주로 두 가지 형태로 사용한다.

1. expression body:

```
const squares = numbers.map(n => n * n);
```

```
numbers.map(function(n) {  
  return n * n;  
});
```

2. statement body:

```
numbers.forEach(n => {  
  if (n % 2 === 0) evens.push(n);  
});
```

```
numbers.forEach(function(n) {  
  if (n % 2 === 0) evens.push(n);  
});
```

Let and Const

심볼 선언을 위해 사용하는 새로운 키워드들

let: var 키워드와 거의 동일. 전역 변수/로컬 변수 모두 동일하게 동작

단, **let**은 블록 내에서는 다르게 동작. 예를 들어, 아래 코드를 살펴보자:

```
1 function iterateVar() {  
2   for(var i = 0; i < 10; i++) {  
3     console.log(i);  
4   }  
5  
6   console.log(i)  
7 }  
8  
9 iterateVar();  
10
```

```
1 function iterateLet() {  
2   for (let i = 0; i < 10; i++) {  
3     console.log(i);  
4   }  
5  
6   console.log(i)  
7 }  
8  
9 iterateLet();  
10
```

첫 번째 함수는 루프 이후에 **i** 를 정상 출력, 두 번째 함수는 **let**으로 정의되었기 때문에 **오류** 발생!


```
C:\ 명령 프롬프트
E:\practice\node.js\prtc01>node iterateVar.js
0
1
2
3
4
5
6
7
8
9
10
E:\practice\node.js\prtc01>
```

```
C:\ 명령 프롬프트
E:\practice\node.js\prtc01>node iterateLet.js
E:\practice\node.js\prtc01>iterateLet.js:9
iterateLet();
^
ReferenceError: iterateLet is not defined
    at Object.<anonymous> (E:\practice\node.js\prtc01>iterateLet.js:9:1)
    at Module._compile (internal/modules/cjs/loader.js:776:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:787:10)
    at Module.load (internal/modules/cjs/loader.js:653:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:593:12)
    at Function.Module._load (internal/modules/cjs/loader.js:585:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:829:12)
    at startup (internal/bootstrap/node.js:283:19)
    at bootstrapNodeJSCore (internal/bootstrap/node.js:622:3)
E:\practice\node.js\prtc01>
```

Error!

const 키워드: 단일 할당문으로 제한

따라서 아래 코드는 오류를 발생시킨다:

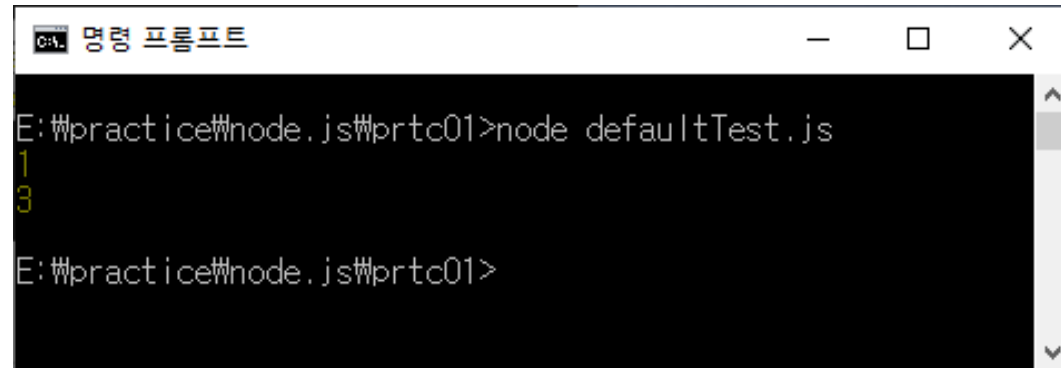
```
const me = 1  
me = 2
```

Default, Rest, and Spread

Default, Rest, Spread: 함수 파라미터와 관련된 3개의 새로운 특징 추가

Default: 함수 파라미터에 디폴트 값을 설정할 수 있게 한다.

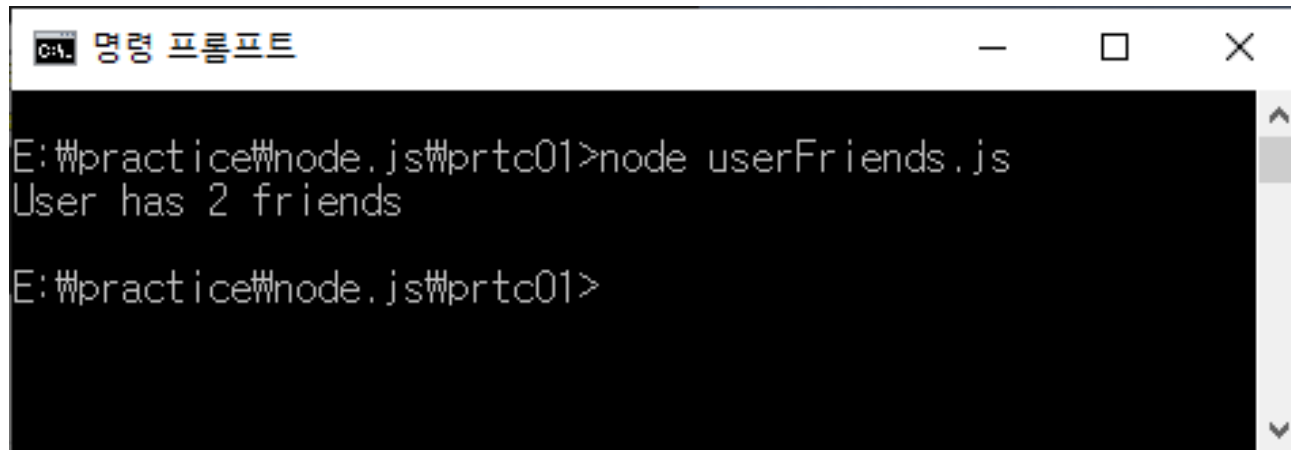
```
function add(x, y=0) {  
  return x + y;  
}  
  
var z1 = add(1); console.log(z1);  
var z2 = add(1,2); console.log(z2);
```



두 인자 중 하나의 인자 값이 전달되지 않거나 정의되지 않은 경우 y 값은 0으로 설정된다.

Rest 기능 - 배열을 따라오는 인자로서 전달할 수 있게 한다:

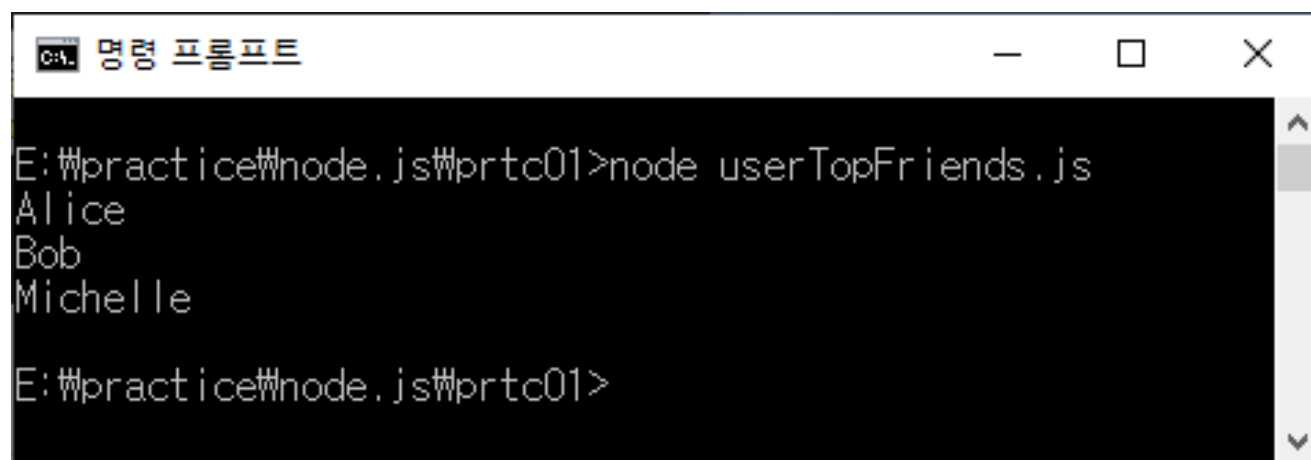
```
function userFriends(user, ...friends) {  
  console.log(user + ' has ' + friends.length + ' friends');  
}  
userFriends('User', 'Bob', 'Alice');
```



```
명령 프롬프트  
E:\practice\node.js\prtc01>node userFriends.js  
User has 2 friends  
E:\practice\node.js\prtc01>
```

Spread 기능 - 배열을 호출 인자로 바꾼다:

```
function userTopFriends(firstFriend, secondFriend, thirdFriends) {  
  console.log(firstFriend);  
  console.log(secondFriend);  
  console.log(thirdFriends);  
}  
  
userTopFriends(...['Alice', 'Bob', 'Michelle']);
```



A screenshot of a Windows Command Prompt window titled "명령 프롬프트". The window shows the execution of a Node.js script. The command entered is "node userTopFriends.js". The output displayed is "Alice", "Bob", and "Michelle" on separate lines. The prompt "E:\practice\node.js\prtc01>" is visible at the bottom.

```
C:\> 명령 프롬프트  
E:\practice\node.js\prtc01>node userTopFriends.js  
Alice  
Bob  
Michelle  
E:\practice\node.js\prtc01>
```

Introducing MEAN

MEAN

MongoDB, Express, Angular, Node.js의 약어

기본 개념 => 오로지 JavaScript만 사용하여 애플리케이션의 모든 파트를 커버

장점

- **Single Language**가 애플리케이션 전반에 걸쳐서 사용됨
- 애플리케이션 모든 파트가 **MVC Architecture**의 사용을 지원
- 자료 구조의 직렬화/역직렬화가 더 이상 요구되지 않음
Data Marshaling이 JSON 객체를 사용하여 수행되기 때문에

몇 가지 중요한 질문들 ...

- 모든 컴포넌트들을 어떻게 연결하는가?
- **Node.js**는 거대한 모듈 생태계를 가지고 있는데 어떤 모듈을 사용해야 하나?
- JavaScript는 패러다임에 독립적인데, MVC 애플리케이션 구조를 어떻게 유지할 수 있나?
- JSON은 스키마가 없는 데이터 구조인데, 언제 어떻게 데이터를 모델링해야 하나?
- 사용자 인증을 어떻게 처리하는가? 실시간 상호 작용을 지원하기 위해 Node.js Non-blocking 아키텍처를 어떻게 사용해야 하는가?
- MEAN 애플리케이션 코드 기반을 어떻게 테스트할 수 있는가?
- DevOps 및 CI의 등장을 고려해 볼 때, MEAN 애플리케이션 개발 프로세스를 신속하게 처리하기 위해 어떤 종류의 JavaScript 개발 도구를 사용할 수 있나?

더 진행하기 전에 기본 프로그램들을 설치해보자.

Installing MongoDB

Download:

<http://mongodb.org/downloads>

MongoDB Download Center

Cloud

Server

Tools

Select the server you would like to run:

MongoDB Community Server

FEATURE RICH. DEVELOPER READY.

MongoDB Enterprise Server

ADVANCED FEATURES. PERFORMANCE GRADE.

Version

3.6.11 (previous release)

OS

Windows 64-bit x64

Package

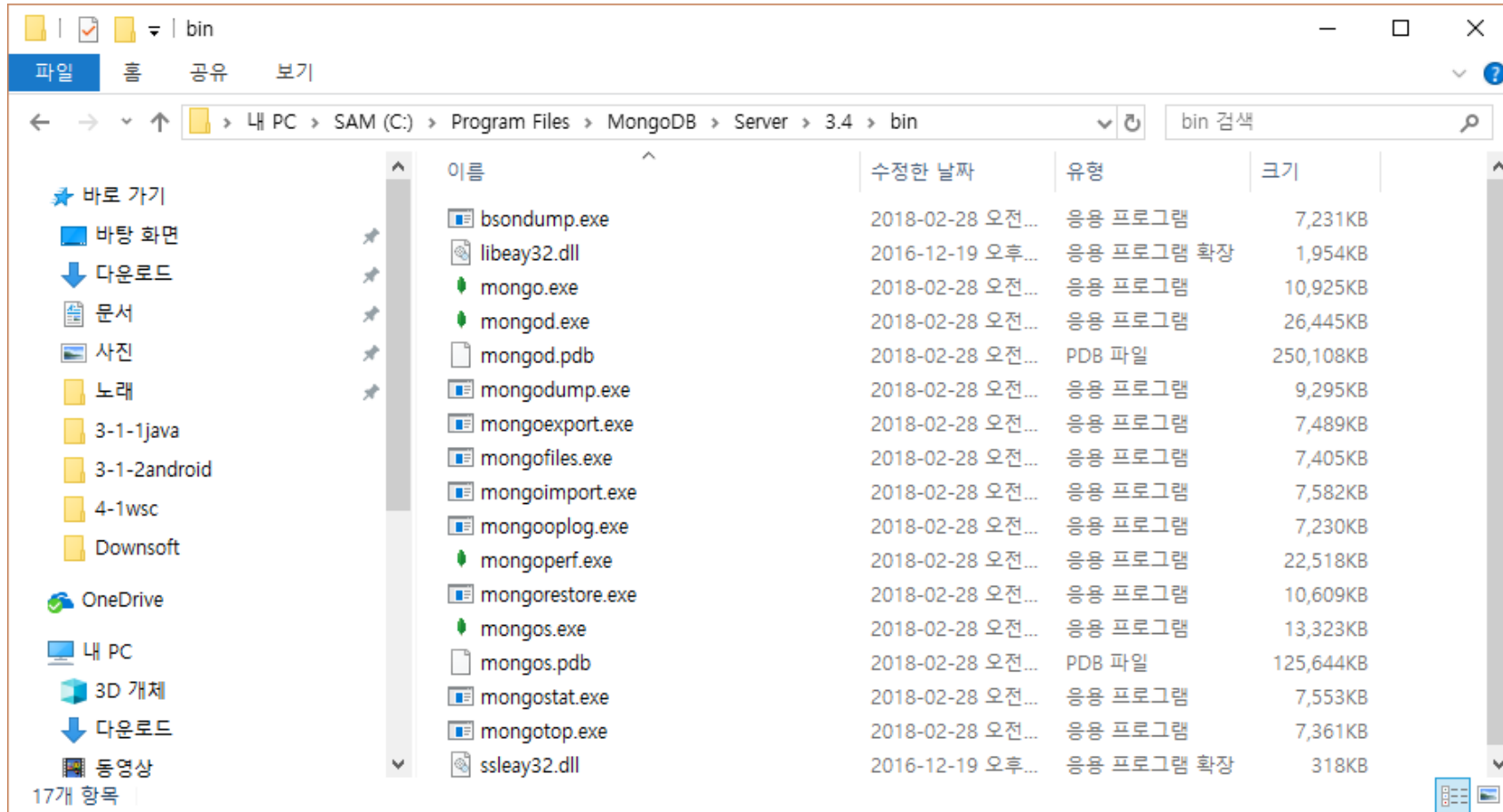
MSI

Download

- Release notes
- Changelog
- All version binaries
- Installation instructions

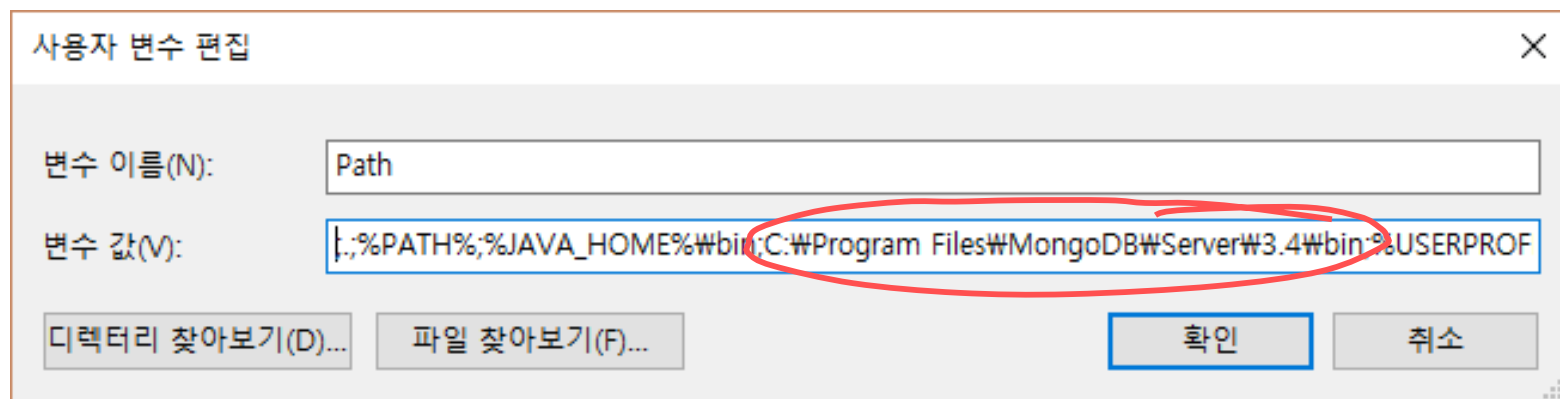
적절한 버전을 다운로드했으면 .msi 파일 실행

- ⇒ MongoDB: **C:\Program Files\MongoDB** 폴더에 설치됨
- ⇒ 실행 중에 디폴트 폴더를 사용하여 데이터 파일 저장(Windows에서 디폴트 폴더 위치: **C:\data\db**)



Running MongoDB manually

환경변수 **path** 설정



mongod 실행

- ⇒ MongoDB를 **독립형 서버**로 사용할 수 있는 메인 MongoDB 서버 프로세스 실행
- ⇒ DB 파일을 저장하는 폴더와 리스닝 할 포트(디폴트 포트: 27017) 필요

1) 디폴트 폴더(/data/db) 사용

명령 프롬프트에서 **C:**로 이동하여 아래 명령 실행:

```
> mkdir c:\data\db
```

```
> mongod
```

2) 대체 폴더(예: D:\mongoData\db) 사용

```
> mkdir d:\mongoData\db
```

```
> mongod --dbpath D:\mongoData\db
```

DB 데몬 프로그램 실행:

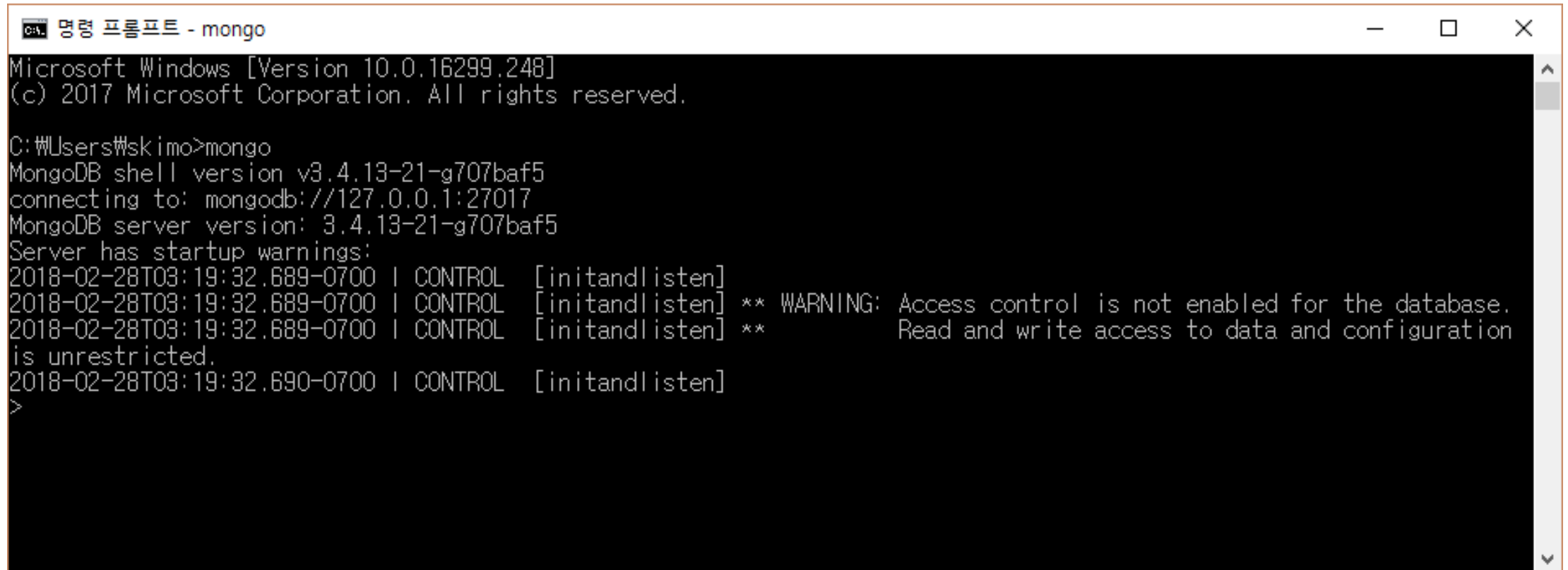
> **mongod --dbpath d:\mongoData\db**

```
명령 프롬프트 - mongod --dbpath d:\mongoData\db
2018-02-28T03:18:49.754-0700 | CONTROL | [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2018-02-28T03:18:49.755-0700 | CONTROL | [initandlisten] db version v3.4.13-21-g707baf5
2018-02-28T03:18:49.755-0700 | CONTROL | [initandlisten] git version: 707baf502bfc2abb33176c0dc5725672b5be1d97
2018-02-28T03:18:49.755-0700 | CONTROL | [initandlisten] OpenSSL version: OpenSSL 1.0.1u-fips 22 Sep 2016
2018-02-28T03:18:49.755-0700 | CONTROL | [initandlisten] allocator: tcmalloc
2018-02-28T03:18:49.755-0700 | CONTROL | [initandlisten] modules: none
2018-02-28T03:18:49.756-0700 | CONTROL | [initandlisten] build environment:
2018-02-28T03:18:49.756-0700 | CONTROL | [initandlisten]   distmod: 2008plus-ssl
2018-02-28T03:18:49.756-0700 | CONTROL | [initandlisten]   distarch: x86_64
2018-02-28T03:18:49.756-0700 | CONTROL | [initandlisten]   target_arch: x86_64
2018-02-28T03:18:49.756-0700 | CONTROL | [initandlisten] options: { storage: { dbPath: "d:\mongoData\db" } }
2018-02-28T03:18:49.831-0700 | STORAGE | [initandlisten] wiredtiger_open config: create,cache_size=3565M,session_max=2000
0,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal
,compressor=snappy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0),verb
ose=(recovery_progress),
2018-02-28T03:19:32.689-0700 | CONTROL | [initandlisten]
2018-02-28T03:19:32.689-0700 | CONTROL | [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-02-28T03:19:32.689-0700 | CONTROL | [initandlisten] **      Read and write access to data and configuration is u
nrestricted.
2018-02-28T03:19:32.690-0700 | CONTROL | [initandlisten]
2018-02-28T19:19:33.446+0900 | FTDC | [initandlisten] Initializing full-time diagnostic data capture with directory 'd
:\mongoData\db\diagnostic.data'
2018-02-28T19:19:34.144+0900 | INDEX | [initandlisten] build index on: admin.system.version properties: { v: 2, key: {
version: 1 }, name: "incompatible_with_version_32", ns: "admin.system.version" }
2018-02-28T19:19:34.144+0900 | INDEX | [initandlisten] building index using bulk method; build may temporarily
use up to 500 megabytes of RAM
2018-02-28T19:19:34.363+0900 | INDEX | [initandlisten] build index done. scanned 0 total records. 0 secs
2018-02-28T19:19:34.365+0900 | COMMAND | [initandlisten] setting featureCompatibilityVersion to 3.4
2018-02-28T19:19:34.368+0900 | NETWORK | [thread1] waiting for connections on port 27017
```

MongoDB shell 시작하기

또 다른 명령 프롬프트 창을 띄워서

> **mongo**



```
명령 프롬프트 - mongo
Microsoft Windows [Version 10.0.16299.248]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\skimo>mongo
MongoDB shell version v3.4.13-21-g707baf5
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.13-21-g707baf5
Server has startup warnings:
2018-02-28T03:19:32.689-0700 | CONTROL | [initandlisten]
2018-02-28T03:19:32.689-0700 | CONTROL | [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-02-28T03:19:32.689-0700 | CONTROL | [initandlisten] ** Read and write access to data and configuration
is unrestricted.
2018-02-28T03:19:32.690-0700 | CONTROL | [initandlisten]
>
```

DB 테스트:

```
> db.articles.insert({ title: "Hello World"})
```

⇒ 새로운 articles 컬렉션을 만들고, **title** 속성이 포함된 JSON 객체 insert

articles 객체 검색:

```
> db.articles.find() OR
```

```
> db.articles.find().pretty()
```

콘솔 출력:

```
{ _id: ObjectId("52d02240e4b01d67d71ad577"), title: "Hello World" }
```

```
명령 프롬프트 - mongo
runClientFunctionWithRetries@src/mongo/shell/session.js:346:31
runCommand@src/mongo/shell/session.js:412:25
DB.prototype._runCommandImpl@src/mongo/shell/db.js:145:16
DB.prototype.runCommand@src/mongo/shell/db.js:161:20
DBCollection.prototype._dbCommand@src/mongo/shell/collection.js:173:1
Bulk/executeBatch@src/mongo/shell/bulk_api.js:903:22
Bulk/this.execute@src/mongo/shell/bulk_api.js:1154:21
DBCollection.prototype.insert@src/mongo/shell/collection.js:317:22
@(shell):1:1
2018-02-04T10:39:48.053+0900 I NETWORK [thread1] trying reconnect to 127.0.0.1:27
017 (127.0.0.1) failed
2018-02-04T10:39:48.056+0900 I NETWORK [thread1] reconnect 127.0.0.1:27017 (127.0
.0.1) ok
> db.articles.find()
> db.articles.insert({title: "Hello MongoDB"})
WriteResult({ "nInserted" : 1 })
> db.articles.find()
{ "_id" : ObjectId("5a7664d653da231fd56398d8"), "title" : "Hello MongoDB" }
> db.articles.find().pretty()
{ "_id" : ObjectId("5a7664d653da231fd56398d8"), "title" : "Hello MongoDB" }
>
```


Introducing npm

Node.js = 플랫폼

- ⇒ Feature들과 API들이 최소로 유지
- ⇒ 더 복잡한 기능 구현을 위해 Node.js의 **모듈 시스템**을 이용

Node.js 모듈을 설치하고, 갱신하고, 삭제하기 위한 가장 좋은 방법 => **npm** 사용

Npm은 주로 아래처럼 사용

- Third-party 모듈을 browsing, downloading, installing 하기 위한 패키지들의 레지스트리
- 로컬 및 글로벌 패키지를 관리하기 위한 CLI 도구

*관례상, **npm**은 Node.js 설치 과정 중에 설치된다.*

Using npm

Express 웹 프레임워크 모듈 설치 => npm의 동작 이해

Npm = 패키지 관리자

Public 모듈을 위한 중앙집중식 레지스트리를 유지

사용 가능한 공용 패키지를 찾아 보려면 공식 웹 사이트(<https://www.npmjs.com/>) 참조

레지스트리에 있는 대부분의 패키지는 오픈 소스 / Node.js 커뮤니티 개발자들이 기여

Package 개발자: 오픈 소스 모듈 개발 => 중앙 레지스트리에 Publish 할 수 있음

=> 다른 개발자들이 다운로드하여 자신의 프로젝트에서 사용

NOTE Node.js에 대한 더 자세한 내용은 <https://docs.npmjs.com> 참조

Npm 설치 프로세스

npm 설치 모드: Local / Global

디폴트 Local 모드가 더 자주 사용

⇒ 애플리케이션 폴더 안의 **node_modules** 폴더에 third-party 패키지들을 설치

Global 모드 - Node.js 에서 전역적으로 사용하고 싶은 패키지를 설치하는데 사용

- ⇒ 대부분의 경우 패키지 Author는 패키지를 전역적으로 설치해야 할 경우 그 사실을 명시해 주기 때문에, 명시되어 있지 않으면 무조건 Local 모드를 사용하면 됨
- ⇒ Windows에서 Global 모드로 설치할 경우 C:\Users\%USERNAME%\AppData\Roaming\npm\node_modules 폴더에 패키지 설치
- ⇒ 실행중인 모든 Node.js 애플리케이션에서 사용할 수 있다

Installing a package using npm

npm install 명령을 사용하여 패키지 설치:

```
> npm install <Package Unique Name>
```

Global로 모듈 설치(-g 플래그 사용):

```
> npm install -g <Package Unique Name>
```

Express를 로컬로 설치:

```
> npm install express
```

⇒ 로컬 node_modules 폴더에 최신 stable 버전의 Express 패키지 설치

특정 버전의 패키지를 설치하려면:

```
> npm install <Package Unique Name>@<Package Version>
```

예를 들어, Express 패키지의 두 번째 메인 버전을 설치하려면 다음 명령을 실행:

```
> npm install express@2.x
```

어떤 패키지가 종속 패키지를 가지면, npm은 패키지 폴더 내의 node_modules 폴더에 필수 패키지를 설치하여 이러한 종속성을 자동으로 해결한다.

앞의 예에서 Express 종속 패키지는 *node_modules/express/node_modules* 아래에 설치된다.

Removing a package using npm

설치된 패키지를 제거하려면:

```
> npm uninstall <Package Unique Name>
```

npm은 패키지를 찾아 로컬 node_modules 폴더에서 제거하려고 한다.

글로벌 패키지를 제거하려면:

```
> npm uninstall -g <Package Unique Name>
```

Updating a package using npm

패키지를 최신 버전으로 갱신하려면:

```
> npm update <Package Unique Name>
```

npm은 아직 존재하지 않더라도 이 패키지의 최신 버전을 다운로드하여 설치한다.

글로벌 패키지를 업데이트하려면:

```
> npm update -g <Package Unique Name>
```

Managing dependencies using the package.json file

앞의 방법은 하나의 패키지만 설치하는 경우에 Good!

- ⇒ 여러 개의 패키지를 설치해야 할 경우 불편
- ⇒ 종속 패키지(dependencies)를 관리하기 위한 더 좋은 방법 필요

package.json

- ⇒ 애플리케이션의 루트 폴더에 **package.json**이라는 환경설정 파일 사용
- ⇒ **package.json** 파일에서 애플리케이션의 다양한 metadata 프로퍼티 정의
- ⇒ 기본적으로 애플리케이션 프로퍼티를 설명하는데 필요한 여러 속성들을 포함하는 JSON 파일:

```
{
  "name" : "MEAN",
  "version" : "0.0.1",
  "dependencies" : {
    "express" : "latest",
    "grunt" : "latest"
  }
}
```


Creating a package.json file

package.json 파일을 수동으로 만들 수 있지만, 더 쉬운 방법은 **npm init** 명령을 사용하는 것

```
> npm init
```

⇒ ***npm**은 애플리케이션에 관한 몇 가지 질문을 하고 새로운 **package.json** 파일을 자동으로 생성해 준다!*

```
CA. 명령 프롬프트
E:\practice\node.js\prtc01>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (prtc01)
version: (1.0.0)
description:
git repository:
keywords:
author:
license: (ISC)
About to write to E:\practice\node.js\prtc01\package.json:
{
  "name": "prtc01",
  "version": "1.0.0",
  "main": "main.js",
  "scripts": {
    "test": "echo \"/>

```

```
CA. 명령 프롬프트
E:\practice\node.js\prtc01>dir
E 드라이브의 볼륨: SAM1902
볼륨 일련 번호: 2071-8E4E

E:\practice\node.js\prtc01 디렉터리

2019-11-24 오후 01:39 <DIR> .
2019-11-24 오후 01:39 <DIR> ..
2019-03-03 오후 05:46 41 application.js
2019-03-06 오전 06:30 491 class-test.js
2019-03-03 오후 04:55 122 defaultTest.js
2019-03-06 오전 05:57 72 doSomething.js
2019-03-03 오후 04:36 134 iterateLet.js
2019-03-03 오후 04:36 134 iterateVar.js
2019-03-03 오후 06:44 55 lib.js
2019-03-03 오후 08:35 52 lib.mjs
2019-03-03 오후 07:44 117 lib2.js
2019-03-03 오후 06:44 65 main.js
2019-03-03 오후 08:37 65 main.mjs
2019-03-03 오후 07:45 105 main2.js
2019-03-06 오전 06:00 71 mainSomething.js
2019-03-06 오전 06:24 97 mainValidator.js
2019-11-24 오후 01:51 201 package.json
2019-03-03 오후 04:59 147 userFriends.js
2019-03-03 오후 05:04 224 userTopFriends.js
2019-03-06 오전 06:23 75 validator.js
2019-03-08 오전 10:26 8,704 실습과제01_007_홍길동.hwp
2019-03-08 오전 10:27 6,661 실습과제01_007_홍길동.zip
                20개 파일 17,633 바이트
                2개 디렉터리 5,053,939,712 바이트 남음

E:\practice\node.js\prtc01>
```

package.json

```
{ } package.json X
{ } package.json > ...
1  {
2    "name": "prtc01",
3    "version": "1.0.0",
4    "main": "main.js",
5    "scripts": {
6      "test": "echo \"Error: no test specified\" && exit 1"
7    },
8    "author": "",
9    "license": "ISC",
10   "description": ""
11  }
```

새 패키지를 설치하고 패키지 정보를 package.json
파일의 종속 패키지로 저장하는 추가 기능

명령 프롬프트

```
E:\practice\node.js\prtc01>npm install express --save
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN prtc01@1.0.0 No description
npm WARN prtc01@1.0.0 No repository field
+ express@4.17.1
added 50 packages from 37 contributors and audited 51 packages in 1.5s
found 0 vulnerabilities
```

node_modules 폴더가 생성되면서 core
모듈 및 express 모듈이 생성된다!

```
{ package.json > ...
1  {
2    "name": "prtc01",
3    "version": "1.0.0",
4    "main": "main.js",
5    "scripts": {
6      "test": "echo \"Error: no test specified\" && exit 1"
7    },
8    "author": "",
9    "license": "ISC",
10   "description": "",
11   "dependencies": {
12     "express": "^4.17.1"
13   }
14 }
```

Installing the package.json dependencies

애플리케이션의 루트 폴더에서 **package.json** 파일을 만든 후

```
> npm install
```

애플리케이션 종속 패키지들을 설치

npm은 **package.json** 파일을 자동으로 감지하여, 모든 애플리케이션 종속 패키지를 설치하고 로컬 node_modules 폴더 아래에 배치한다.

종속 패키지를 설치하는 대체 방법으로는 아래 npm update 명령을 사용하는 것이다:

```
> npm update
```

누락된 패키지가 설치되고, 모든 기존 종속 패키지는 지정된 버전으로 업데이트된다.

Updating the package.json file

```
> npm install express --save
```

새 패키지를 설치하고 패키지 정보를 package.json 파일의 종속 패키지로 저장하는 추가 기능

예: 최신 버전의 Express를 설치하고 종속 패키지로 저장

- ✓ 최신 버전의 Express를 설치하고, Express 패키지를 package.json 파일에 대한 종속성으로 추가

NOTE npm의 광범위한 설정 옵션에 대한 자세한 내용은 <https://docs.npmjs.com/files/package.json>의 공식 설명서를 참조하도록 한다.

실습과제 22-1

본문에 나오는 아래의 Node applications 실행:

```
lib.js / main.js
```

```
lib2.js / main2.js
```

```
doSomething.js / mainSomething.js
```

```
class-test.js
```

```
iterateVar.js
```

```
iterateLet.js
```

```
defaultTest.js
```

```
userFriends.js
```

```
userTopFriends.js
```

```
application.js
```

```
npm init => package.json 생성
```

Node.js로 만드는 웹 애플리케이션

Adoption



- 3+ million active users of Node.js, a 100% YOY increase
- 98% of the Fortune 500 companies regularly use Node.js
- 25% of developers at growth-stage companies in enterprise software are using Node.js
- 25% of developers at FinTech startups are using Node.js
- 33% of developers at Healthcare startups are using Node.js with the primary purpose of enabling rapid innovation
- 48% of developers are using Node.js at IoT companies
- 80% of developers at education startups are using the technology
- 2 Million Platform Downloads per Month
- 2 Billion package installs per month from 2 Million unique IP addresses across more than 200 countries

글로벌 기업의 Node.js 구축 사례

▶ Groupon

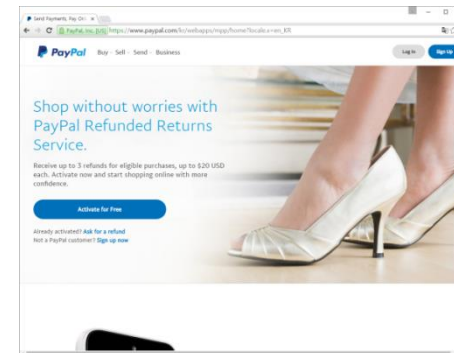
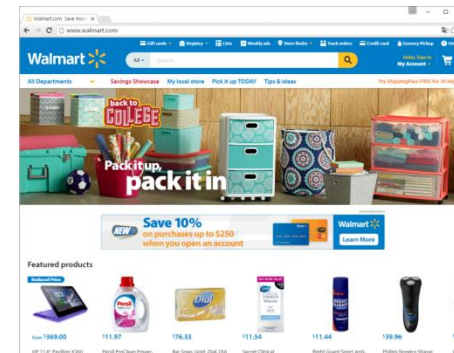
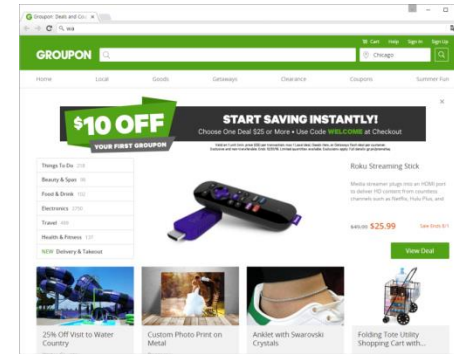
- **Rails**로 개발된 서버를 **Node**로 전환
- 더 적은 수의 서버로 읽기 속도는 50% 향상

▶ Walmart

- 블랙 프라이데이 기간동안 **Node** 서버로 온라인 트래픽의 55%를 처리
- 서버의 CPU 이용률은 1% 전후
- 단 한대의 서버도 다운되지 않음

▶ PayPal

- 자바로 개발된 서버를 **Node**로 전환
- 초당 처리건수 두배 증가
- 응답시간 35% 단축
- 코드량 33% 감소
- 2014년 **Node** 개발을 위해 **JavaScript** 개발자 400명 채용



▶ Yahoo

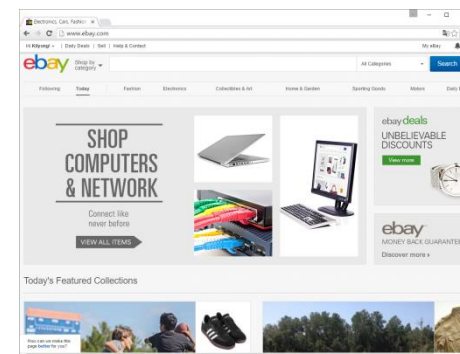
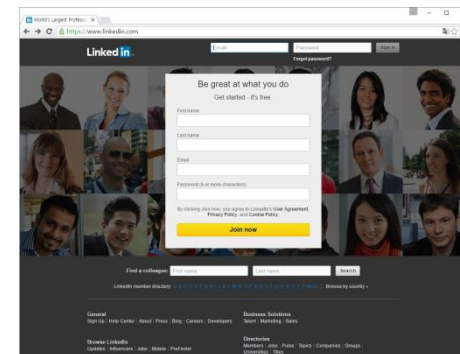
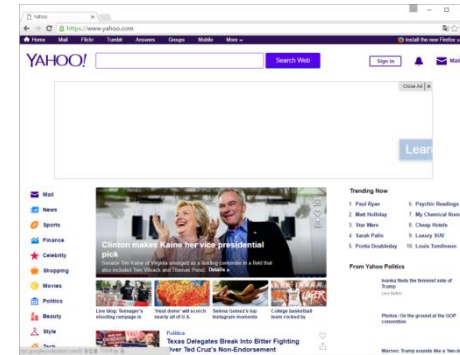
- 분당 168만 ~ 200만 건의 요청 처리에 **Node** 서버 사용
- 200여명의 개발자가 **Node** 개발
- 500개의 내부 모듈과 800개의 외부 모듈 개발

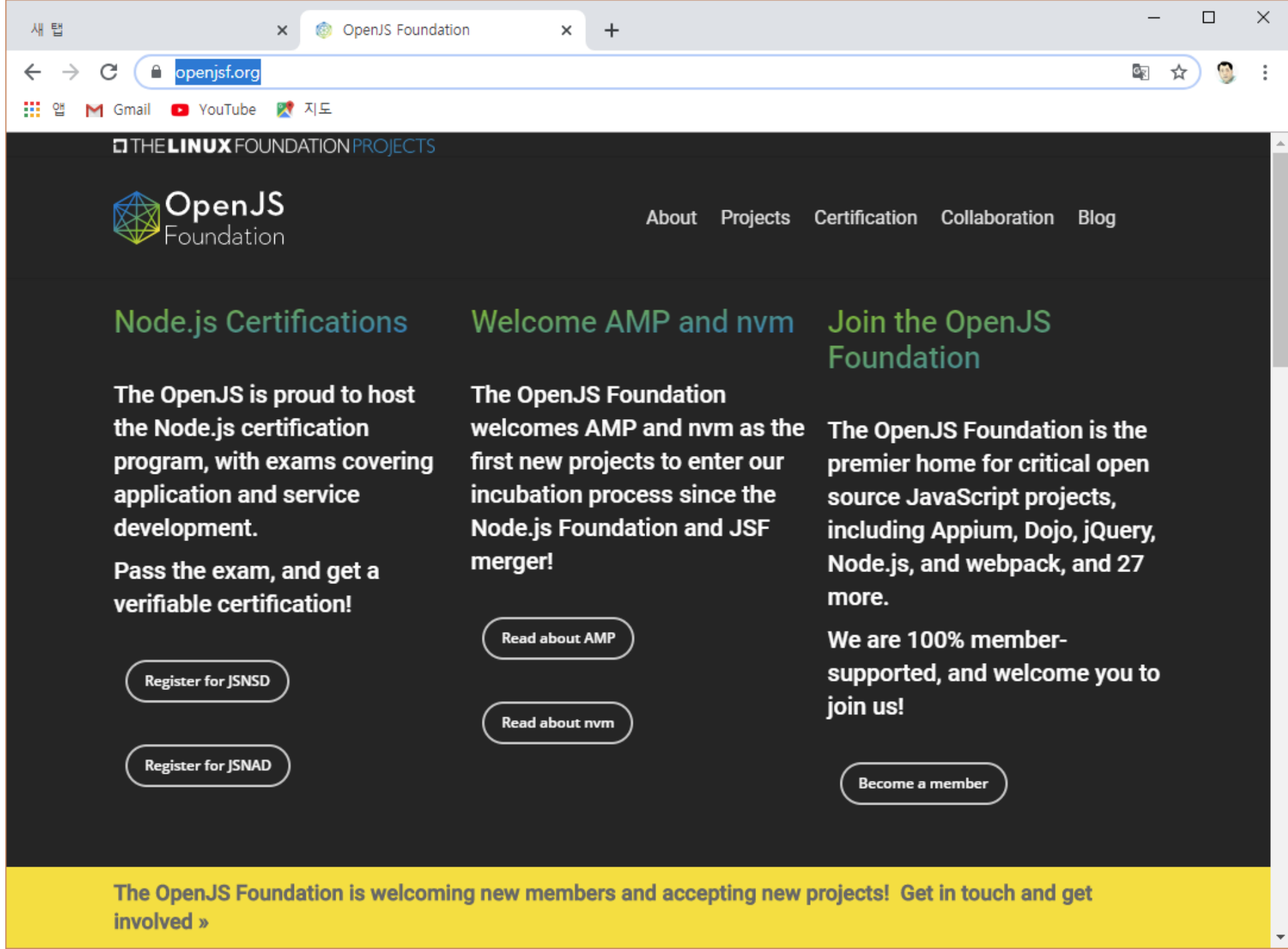
▶ LinkedIn

- **Rails**로 개발된 서버를 **Node**로 전환

▶ eBay

- 프론트 엔드는 **Node**로 개발
- 백 엔드 서비스는 **Scala, Java**로 개발









새 탭





OpenJS Certification Program -


+

← → ↺


openjsf.org/certification/

 앱  Gmail  YouTube  지도

 OpenJS Foundation


About Projects **Certification** Collaboration Blog



OpenJS Node.js Services Developer (JSNSD)

The OpenJS Node.js Services Developer certification is designed for anyone looking to demonstrate competence in creating RESTful Node.js servers and services (or Microservices) with a particular emphasis on security practices.

[REGISTER FOR THE JSNSD EXAM »](#)



OpenJS Node.js Application Developer (JSNAD)

The OpenJS Node.js Application Developer certification is designed for anyone looking to demonstrate competence with Node.js to create applications of any kind, with a focus on knowledge of Node.js core APIs.

[REGISTER FOR THE JSNAD EXAM »](#)

