

KB금융그룹



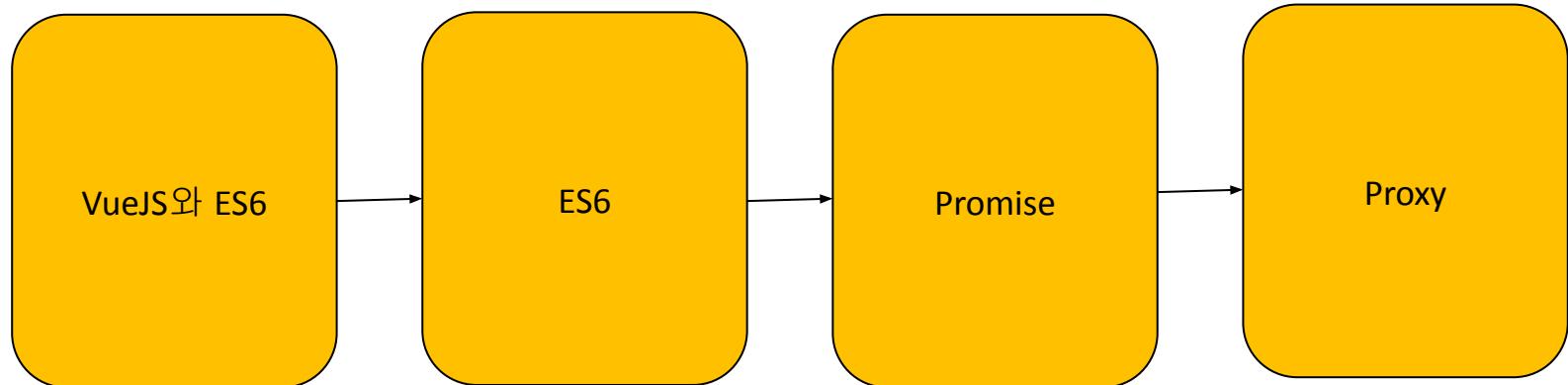
국민의 편의
금융파트너

2024년 상반기 K-디지털 트레이닝

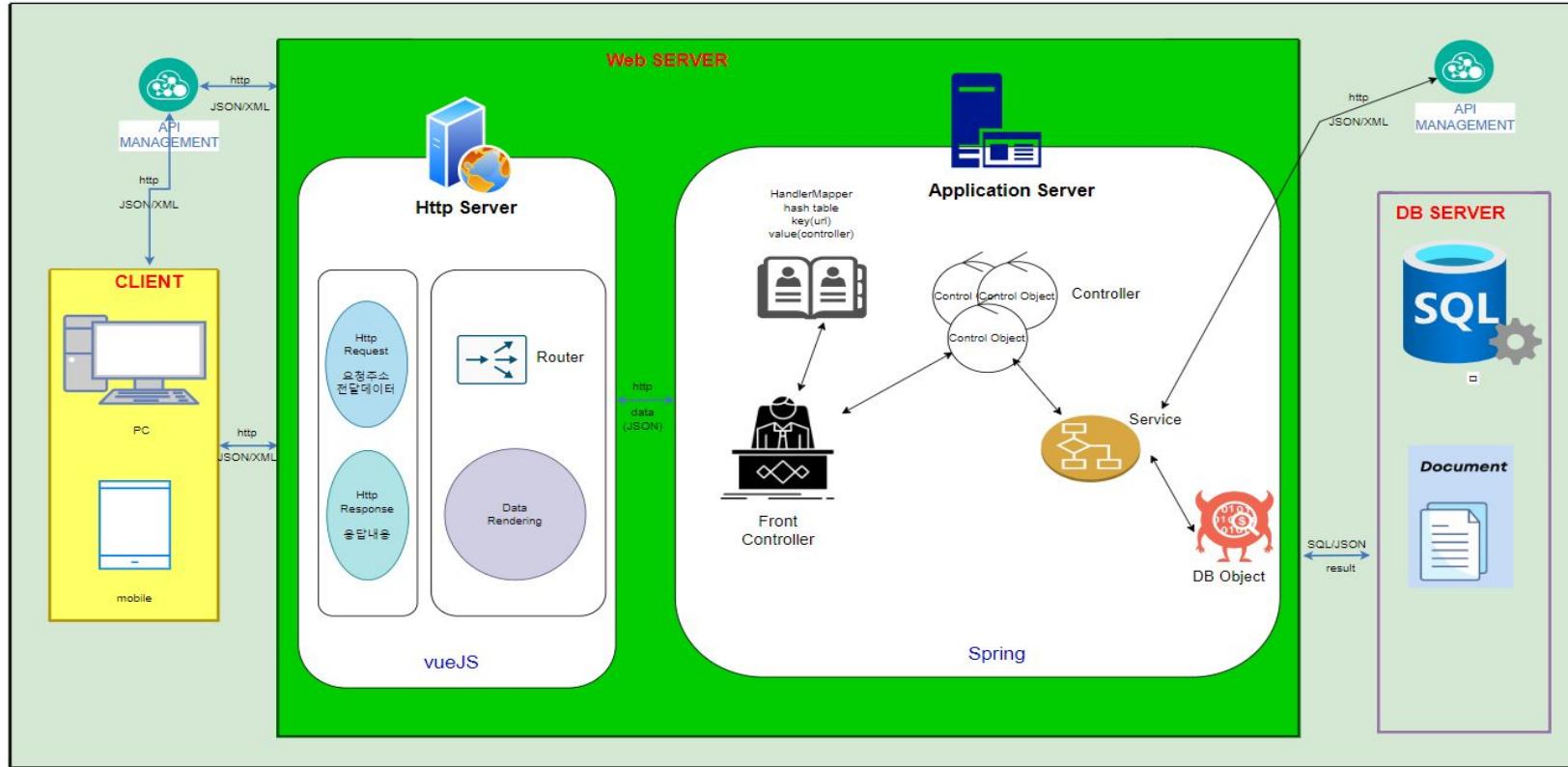
ES6

[KB] IT's Your Life



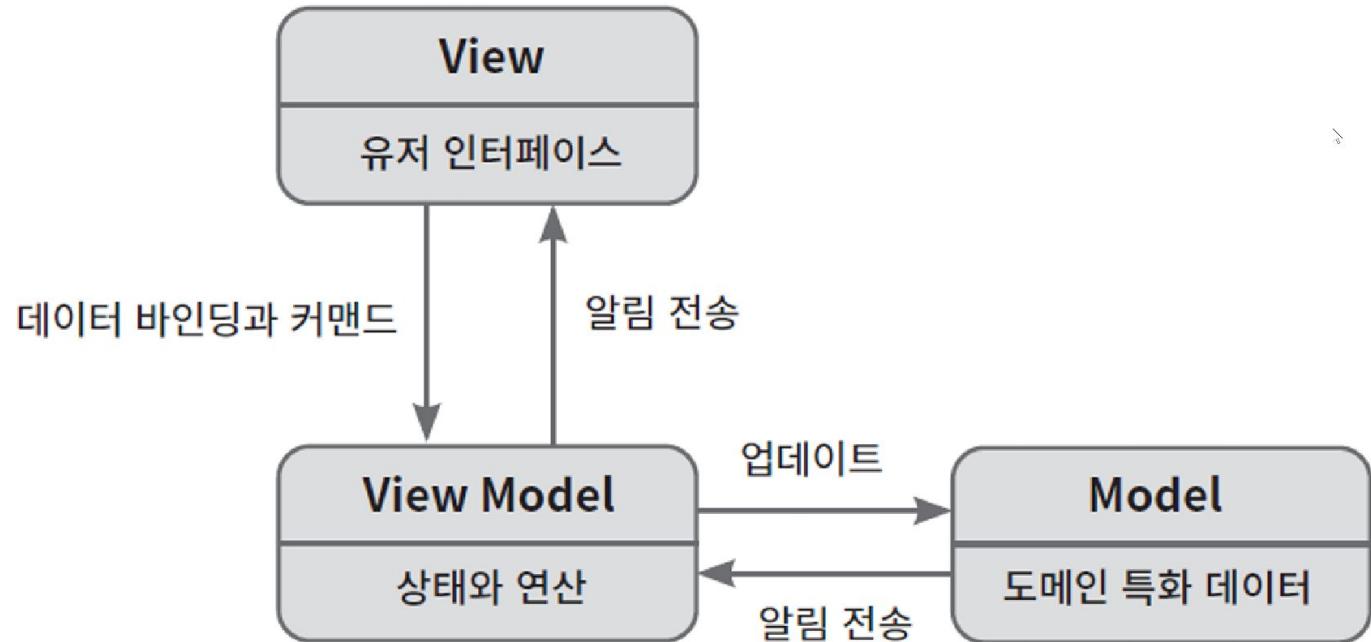


오늘의 목표



오늘의 목표

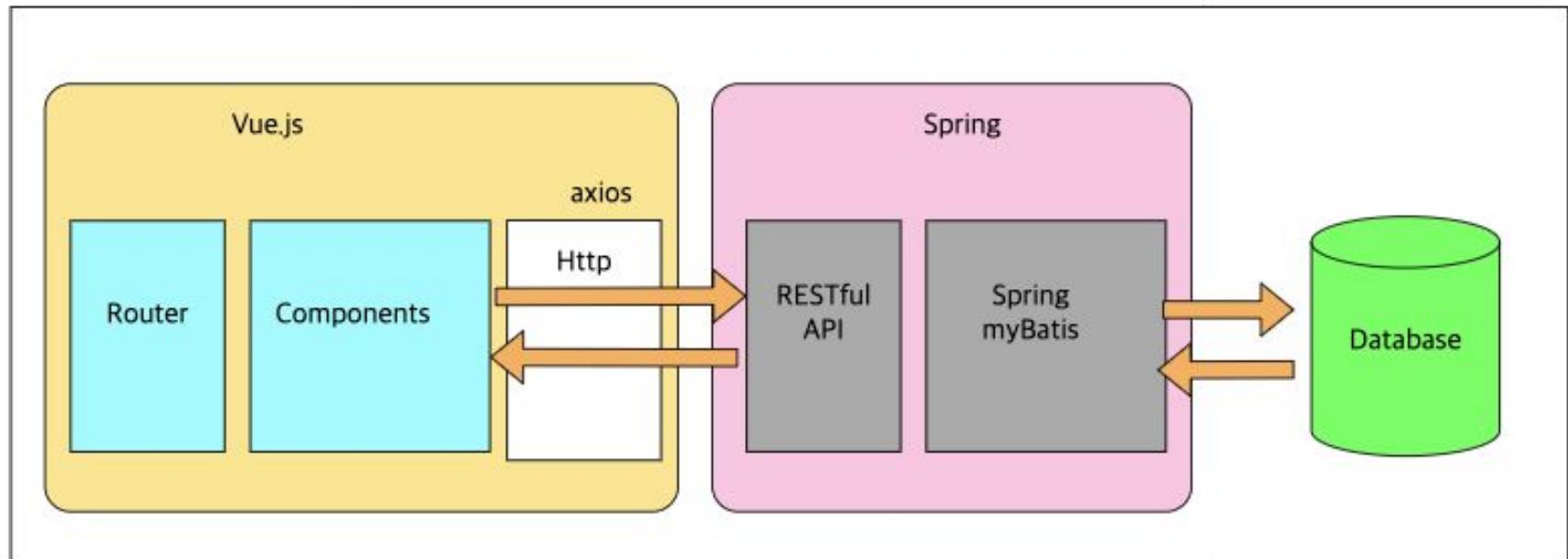
MVVM



vueJS/ES6

VueJS 개요

- 클라이언트-서버측 프레임워크



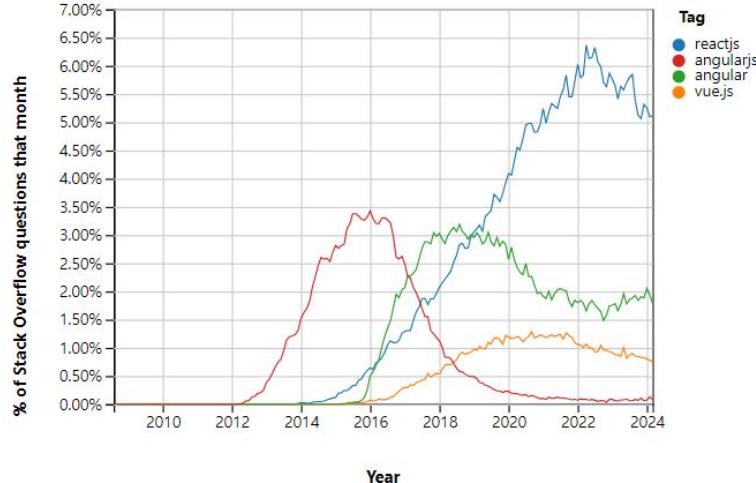
VueJS 개요

- [Vue vs React: Choosing a Front-End Framework for Project](#)
- <https://tech.kakao.com/>

	React	Vue
Released	Jordan Walke 2013	Evan You 2014
Coding Speed	normal	fast
Code Reusability	good	best
Usually Use	company	community
site	Uber, Netflix, 토스, 뱅크샐러드	kakao, naver, inflearn, programmers

VueJS 개요

- 상대적으로 쉽다.
 - 패키지가 많음.
 - 학습 곡선이 완만.
 - 템플릿 기반으로 재사용성, 생산성이 높음.
- 유연하고 가볍다.
 - 라이브러리 사이즈가 작음.
 - 초기 실행시간이 짧다.
 - 실행 메모리가 작다.



<비교>

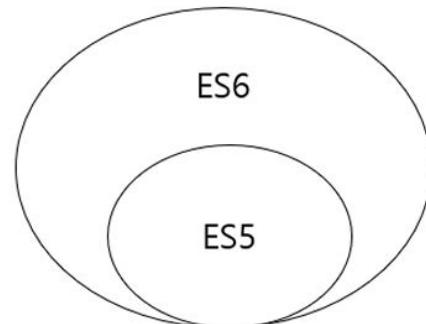
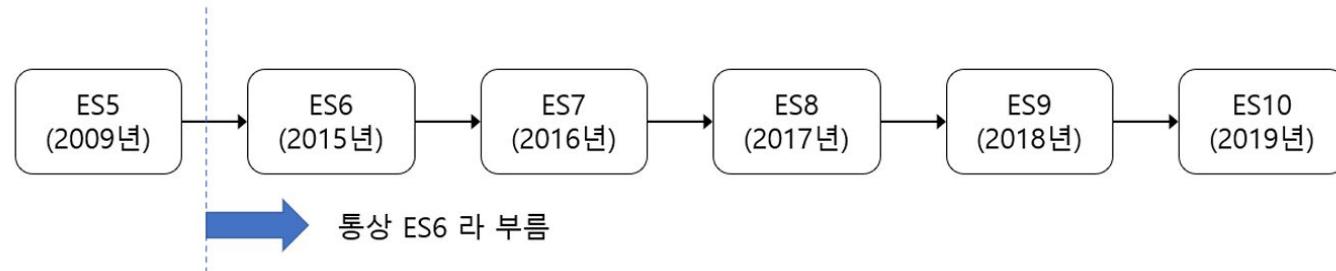
- Angular: TypeScript가 필수, 개발 필수 내용이 매우 많음.
- React: ES6필수, 숙련된 JS기술 필요, WebPack에 대해 별도로 학습



VueJS 개요

- **ES6**

- ECMAScript 6
- ECMA-262 기술 규격에 정의된 표준화된 스크립트 프로그래밍 언어



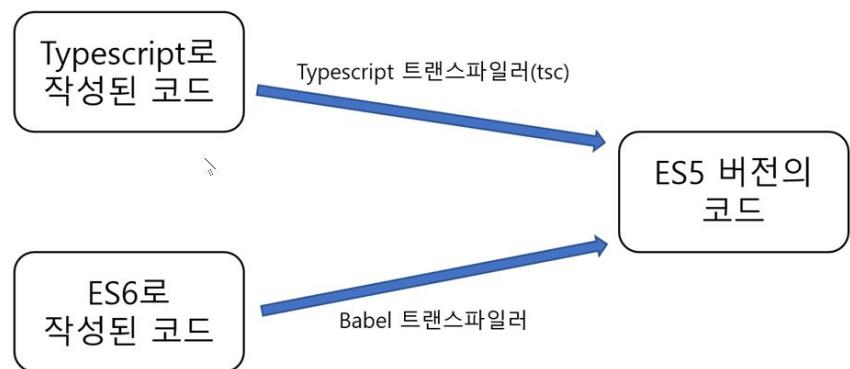
ES6는 이전 버전의 문법을 포함하여 지원
+
class, Arrow 함수 등이 추가

VueJS 특징

- **Transpile(트랜스파일)**

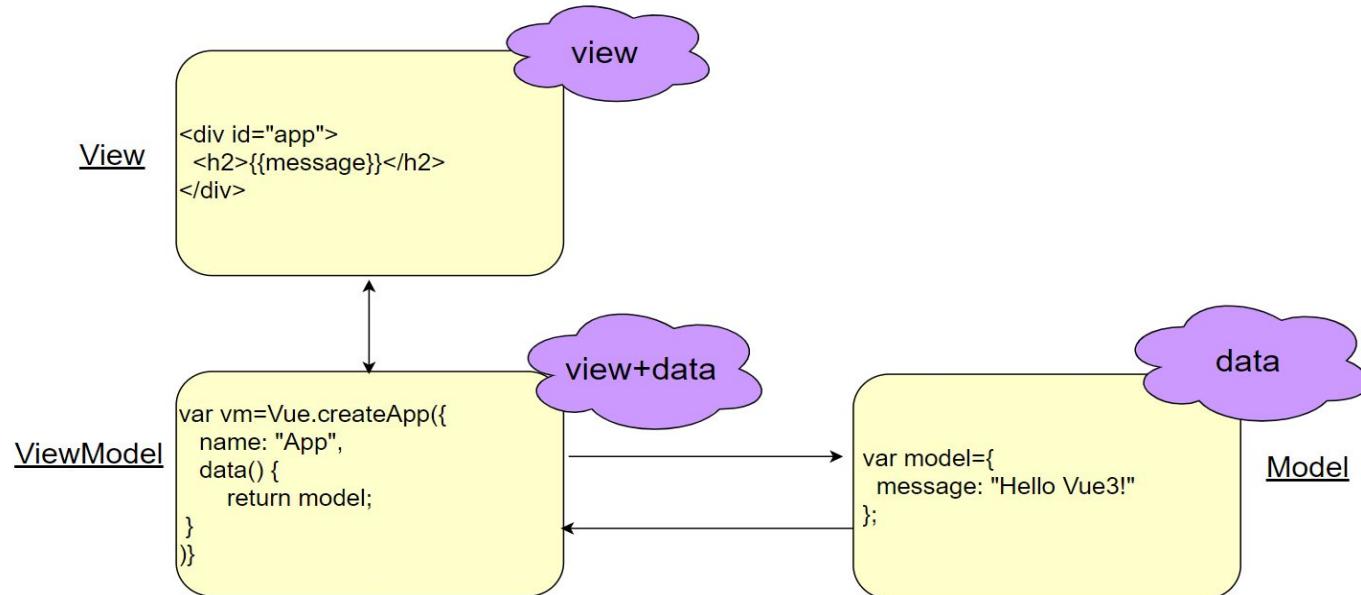
- Translate + Compile
- ES6나 TypeScript를 이전 버전 ES5로 변환
- 예) Babel, tsc

- vue3부터 ES6 필수가 됨.



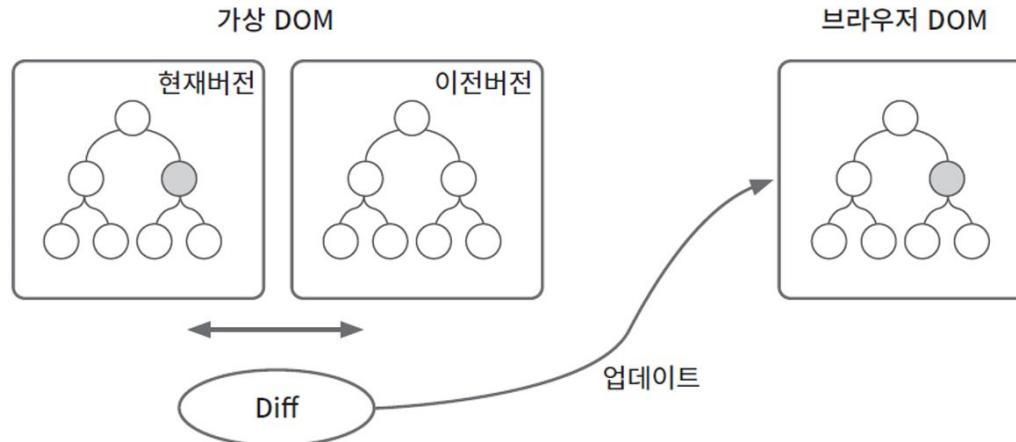
MVVM 패턴

- 어플리케이션을 Model-View-ViewModel로 분리해서 구현
- Model(data)가 변경되면 ViewModel(rendering, binding)에게 알리고 ViewModel이 View(view)를 갱신시킴.
- 엑셀시트의 표 → 그래프/차트로 연동

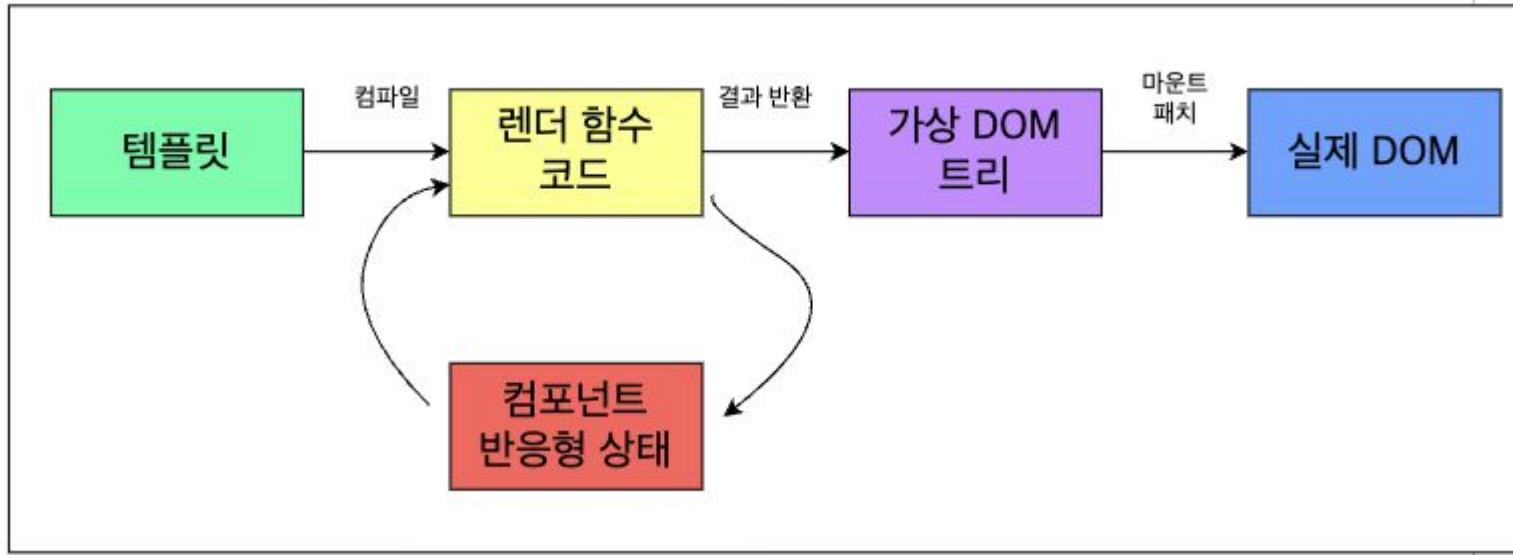


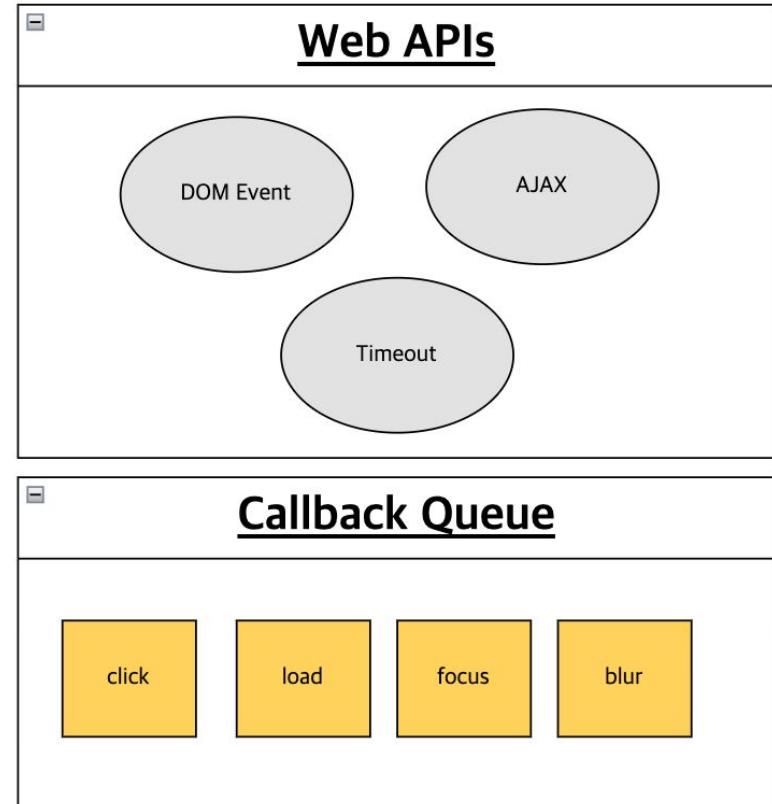
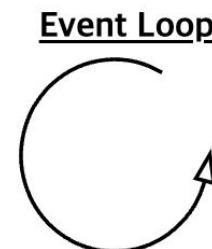
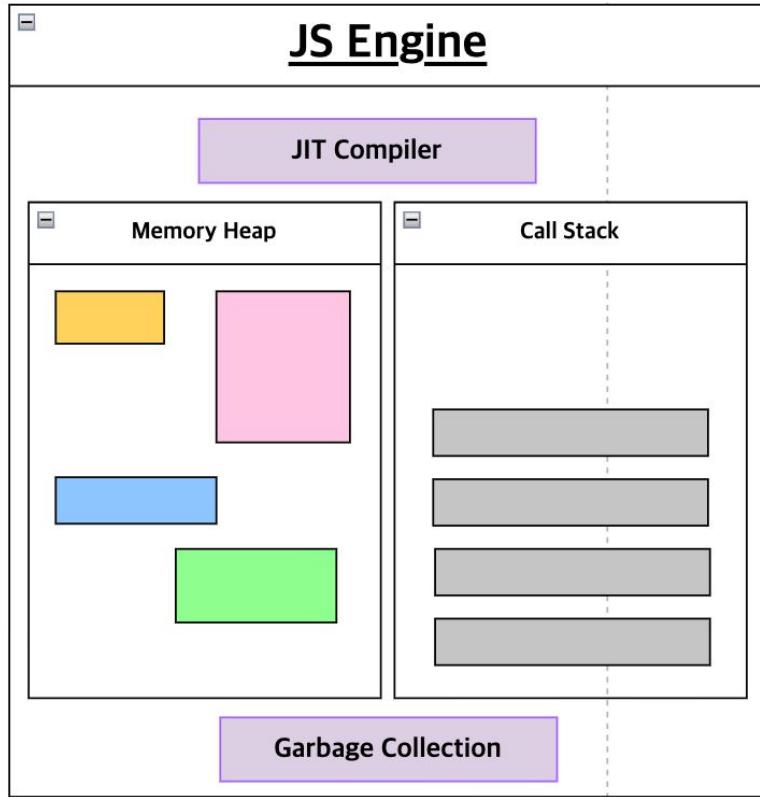
virtual DOM

- 가상 DOM : Document Object Model, 브라우저 메모리에서 관리되는 브라우저 DOM에 대한 추상 객체
- Vue 성능이 React 성능보다 좋음.
- 특정 화면의 일부분을 자주 갱신하는 경우에 성능이 좋음.

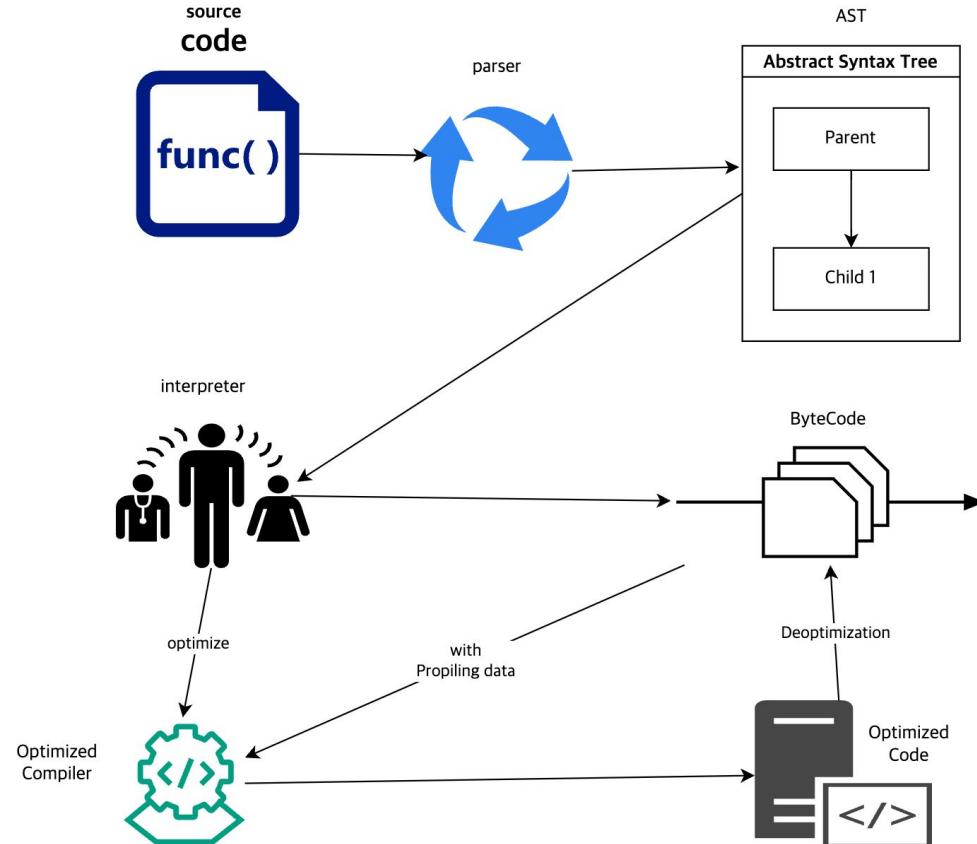


virtual DOM





JavaScript Engine



JavaScript Engine



- V8 — open source, developed by Google, written in C++
- Rhino — managed by the Mozilla Foundation, open source, developed entirely in Java
- SpiderMonkey — the first JavaScript engine, which back in the days powered Netscape Navigator, and today powers Firefox
- JavaScriptCore — open source, marketed as Nitro and developed by Apple for Safari
- KJS — KDE's engine originally developed by Harri Porten for the KDE project's Konqueror web browser
- Chakra (JScript9) — Internet Explorer
- Chakra (JavaScript) — Microsoft Edge
- Nashorn, open source as part of OpenJDK, written by Oracle Java Languages and Tool Group
- JerryScript — is a lightweight engine for the Internet of Things.

V8 used to have two compilers

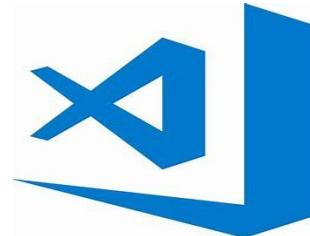
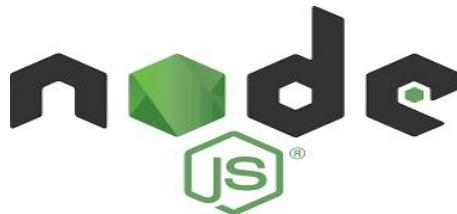
Before version 5.9 of V8 came out (released earlier this year), the engine used two compilers:

- full-codegen — a simple and very fast compiler that produced simple and relatively slow machine code.
- Crankshaft — a more complex (Just-In-Time) optimizing compiler that produced highly-optimized code.

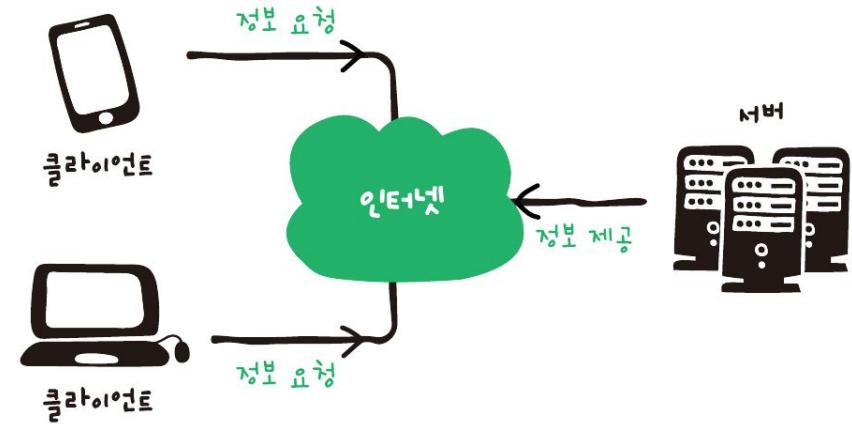
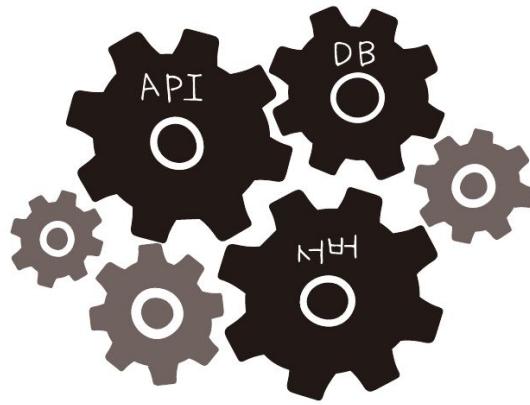
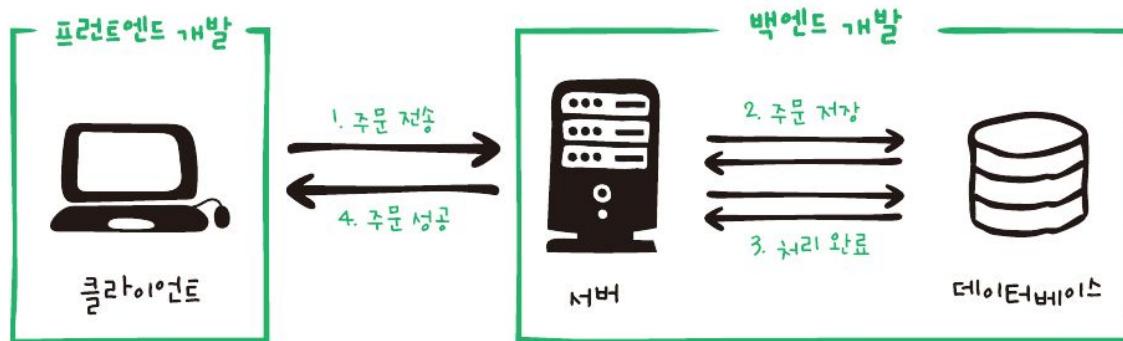
- The main thread does what you would expect: fetch your code, compile it and then execute it 메인 스레드: 코드를 가지고 오고, 컴파일하고, 실행
- There's also a separate thread for compiling, so that the main thread can keep executing while the former is optimizing the code 부가1 스레드: 코드 최적화
- A Profiler thread that will tell the runtime on which methods we spend a lot of time so that Crankshaft can optimize them 프로파일러(개요, 정리) 스레드: 코드 최적화를 알려줌
- A few threads to handle Garbage Collector sweeps 부가2 스레드: gc핸들링

개발환경구축

- 프로그램 설치 및 환경설정

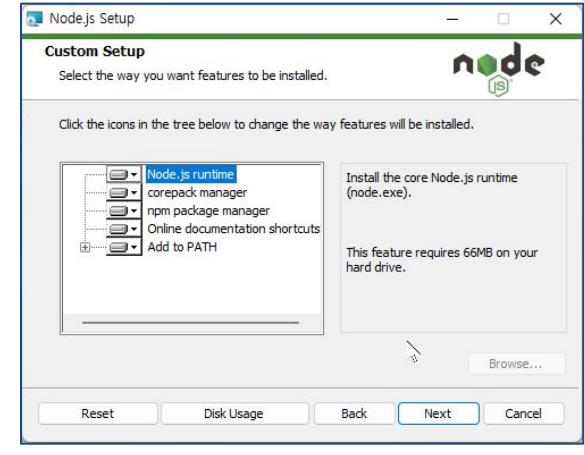
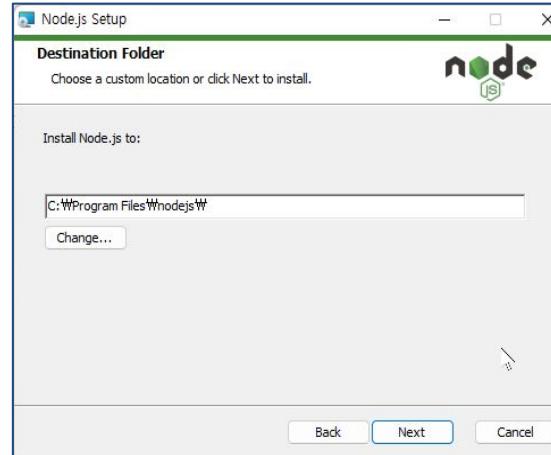
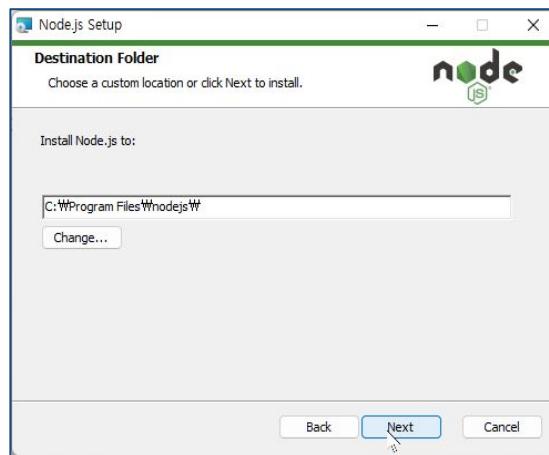
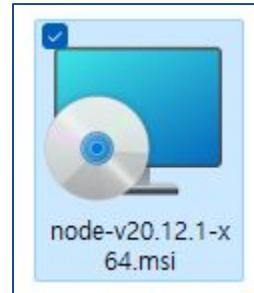
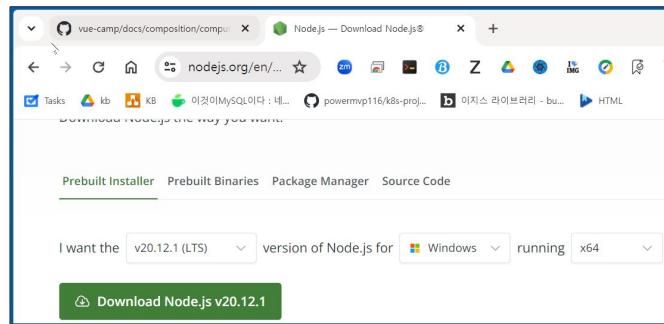


개발환경구축

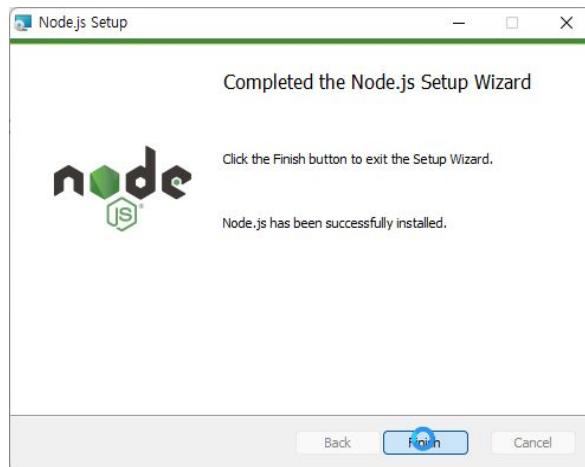
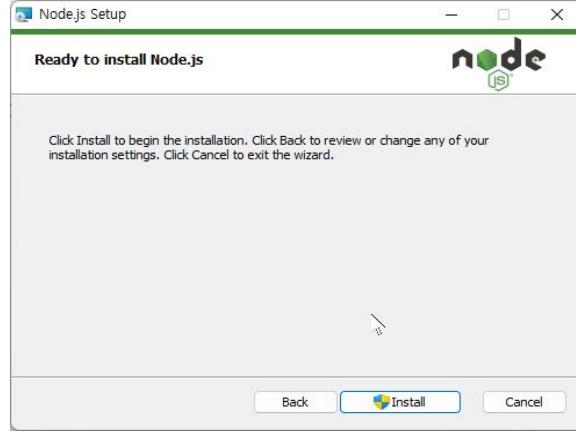
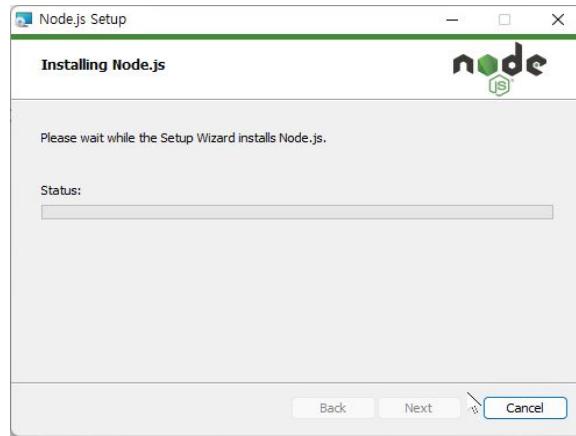
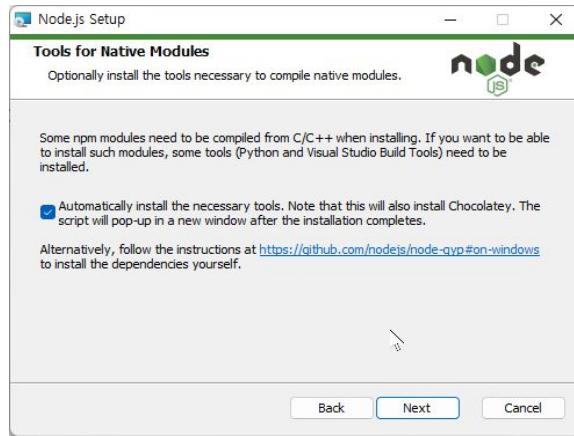


NodeJS 설치

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



NodeJS 설치



```
■ 선택 Install Additional Tools for Node.js
Tools for Node.js Native Modules Installation Script
-----
This script will install Python and the Visual Studio Build Tools, necessary to compile Node.js native modules. Note that Chocolatey and required Windows updates will also be installed.

This will require about 3 GB of free disk space, plus any space necessary to install Windows updates. This will take a while to run.

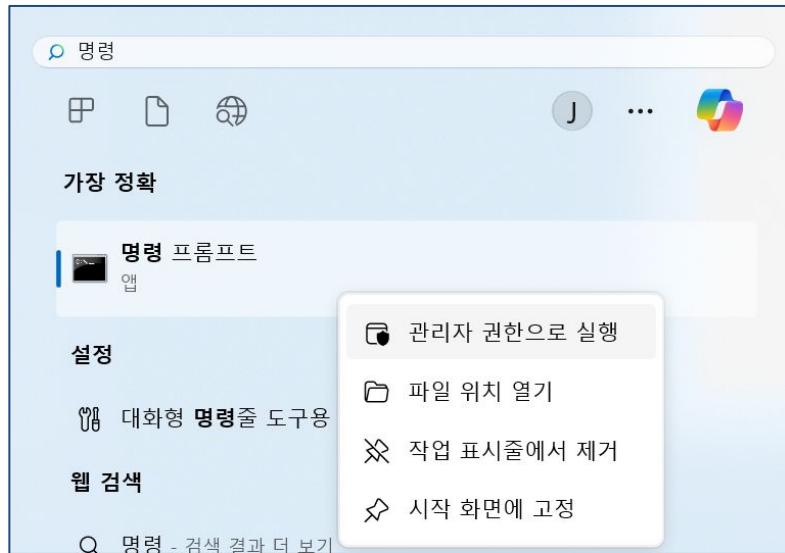
Please close all open programs for the duration of the installation. If the installation fails, please ensure Windows is fully updated, reboot your computer and try to run this again. This script can be found in the Start menu under Node.js.

You can close this window to stop now. Detailed instructions to install these tools manually are available at https://github.com/nodejs/node-gyp#on-windows

계속하려면 아무 키나 누르십시오 . . .
-----
```

```
■ 관리자: Windows PowerShell
-----[Output]
Download of vcredist140-86 ... 1EABC088D05A5746B0D05769A4938C859CA79\VC_redist.x86.exe
100% - Completed download of C:\Users\jung\Downloads\Chocolatey\VCredist140\14.38.39195\VC_redist.x86.exe (18.21 MB)
Download of VC_redist.x64.exe (19.21 MB) completed.
Hashes match.
Installing vcredist140-86 ...
vcredist140-86 has been installed.
Download of vcredist140-64 bit ...
Download of vcredist140-64 \VC_redist.x64.exe
100% - Completed download of C:\Users\jung\Downloads\Chocolatey\VCredist140\14.38.39195\VC_redist.x64.exe (24.24 MB)
Download of VC_redist.x64.exe (24.24 MB) completed.
Installing vcredist140-64 ...
vcredist140-64 has been installed.
vcredist140 may be able to be automatically uninstalled.
The upgrade of vcredist140 was successful!
The upgrade of vcredist140 to its current version is likely default.
Progress: Downloading vcredist2015.14.0.24215.20170201... 100%
vcredist2015 v14.0.24215.20170201 [Approved]
vcredist2015 package files upgrade completed. Performing other installation steps.
The upgrade of vcredist2015 was successful.
Software installed to 'C:\Program Files\Chocolatey\lib\vcredist2015'
Progress: Downloading python312-3.12.2... 52%
```

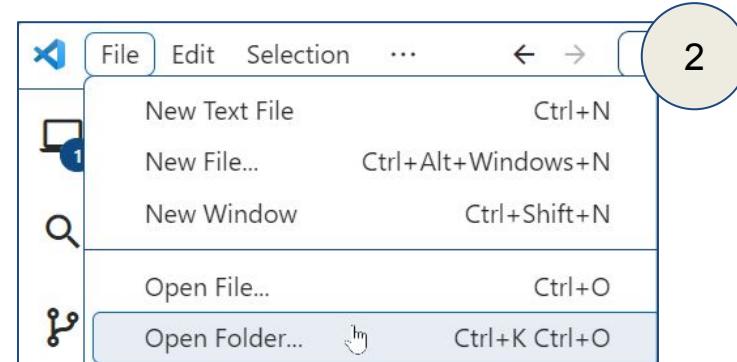
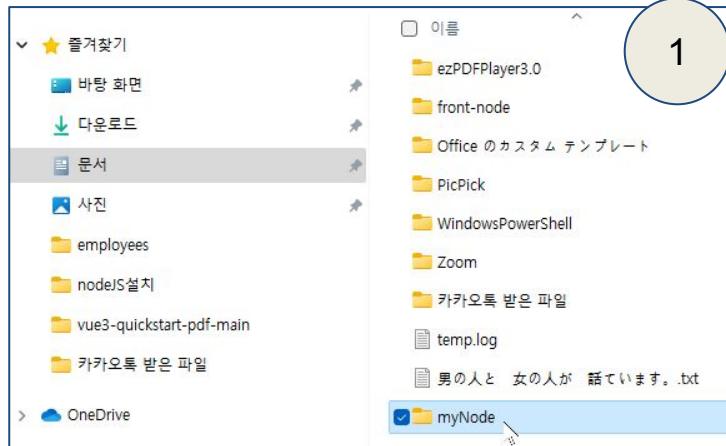
NodeJS 설치



The screenshot shows a Windows Command Prompt window titled '관리자: 명령 프롬프트 - node'. The command prompt is running in administrator mode. The output is as follows:

```
C:\Windows\system32>node
Welcome to Node.js v20.12.1.
Type ".help" for more information.
> console.log('welcome');
welcome
undefined
```

NodeJS 설치



A screenshot of the Visual Studio Code editor. The sidebar shows a folder named 'MYNODE' with a file 'hello.js' selected. The main area displays the following code:

```

1  function hello(name){
2      console.log(name + "님, 안녕하세요!!");
3  }
4
5  hello('홍길동');

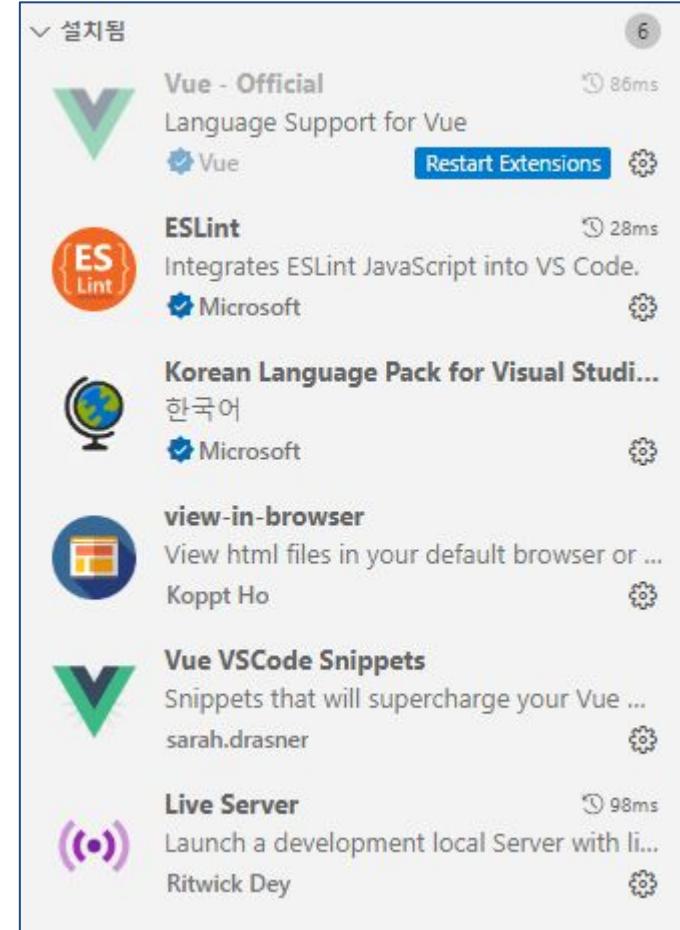
```

A circled '3' is on the right side of the editor area.

A screenshot of the Visual Studio Code terminal. The tab bar at the top includes 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is underlined), and 'PO'. The terminal window shows the command 'PS C:\Users\jungw\Documents\myNode> node hello.js' followed by the output '홍길동님, 안녕하세요!!'. A circled '4' is on the right side of the terminal area.

VSCODE 환경설정

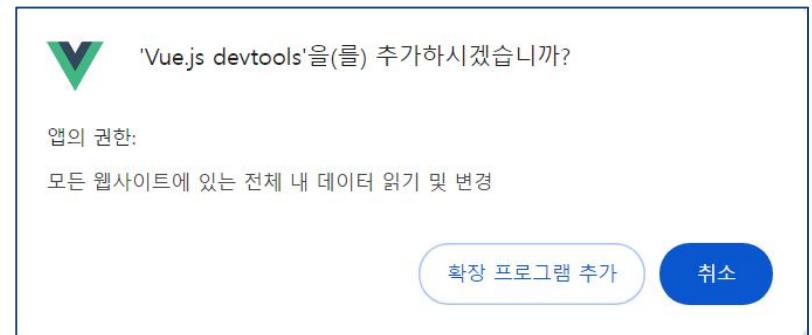
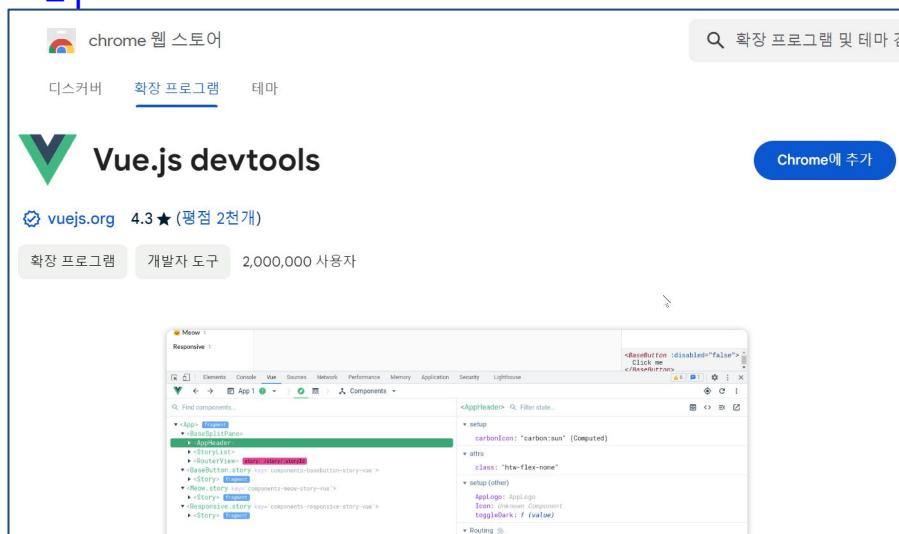
- **ESLint**
 - 자바 스크립트 코드 스타일, 문법 체크
- **Korean Language Pack**
 - 메뉴 한글로 변환
- **vue-in-browser**
 - 웹서버 없이 HTML페이지를 브라우징
- **Vue Language Pack(Volar)**
 - Vue.js 문법 체크, 코드 자동 완성
- **Vue-Official**
 - Volar deprecated
- **Vue VSCode Snippets**
 - 코드 블록 자동완성
- **Live Server**
 - 실시간 코드 변경 반영 웹서버



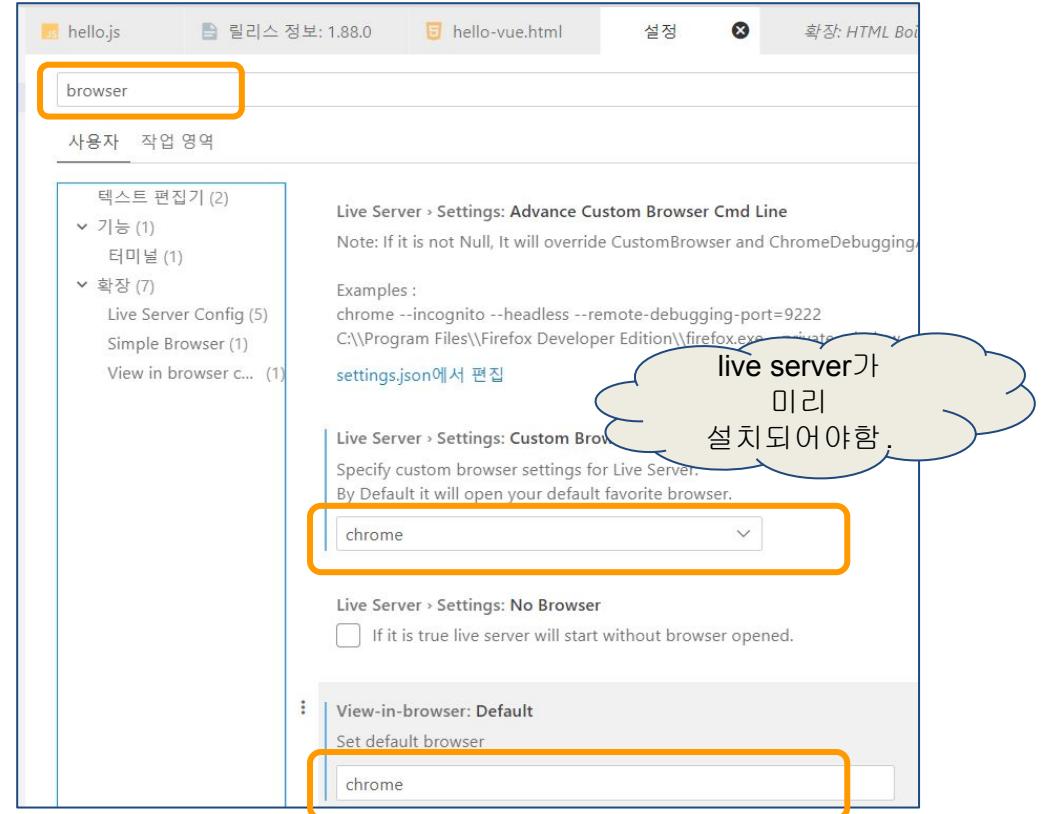
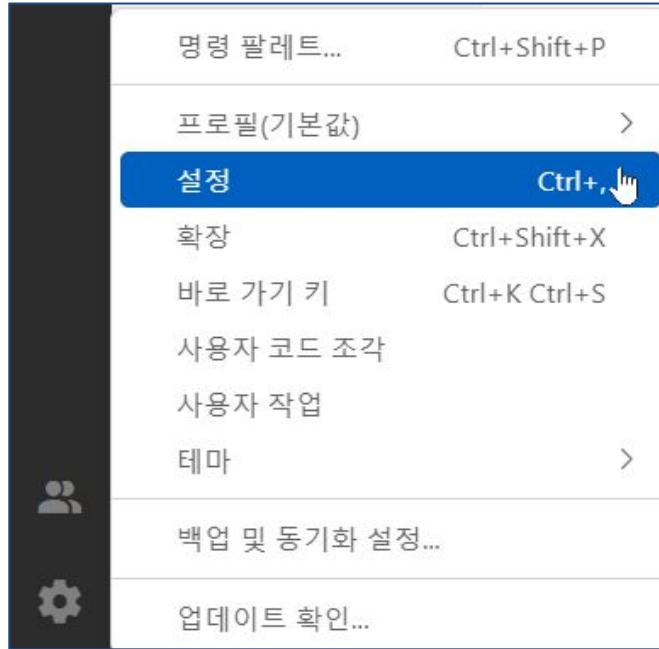
Vue.js devtools 설치

- **Vue.JS 디버깅**
- **브라우저에 확장 플러그인 형태로 설치**
 - **Vue 3버전용 Vue.js Devtools 6**

<https://chromewebstore.google.com/detail/vuejs-devtools/nhdogjmejiglipccpnnaanhbleajbpd?pli=1>



Vue.js devtools 설치



Vue.js devtools 설치

The screenshot illustrates the setup of the Vue.js devtools. On the left, a code editor shows the source code for a Vue.js application named 'Hello Vue3'. The browser window in the center displays the application's output: 'Hello Vue3'. The browser's toolbar includes a 'Vue' tab, which is highlighted with an orange box. A context menu is open over the browser window, with the 'Open with Live Server' option highlighted by an orange box. To the right, a detailed view of the Vue DevTools interface is shown, featuring various tabs like 'Vue', 'Find apps...', 'Find components', and 'data'. The 'data' tab shows the message 'Hello Vue3'. The bottom right corner of the devtools interface is also highlighted with an orange box.

127.0.0.1:5500/hello...

Vue

Hello Vue3

message: "Hello Vue3"

정의로 이동 F12
참조로 이동 Shift+F12
피킹 >
모든 참조 찾기 Shift+<Alt>+F12
기호 이름 바꾸기 F2
모든 항목 변경 Ctrl+F2
문서 서식 Shift+<Alt>+F
리팩터링... Ctrl+Shift+R
소스 작업...
잘라내기 Ctrl+X
복사 Ctrl+C
붙여넣기 Ctrl+V
Open with Live Server <Alt>+L <Alt>+O
Stop Live Server <Alt>+L <Alt>+C
View In Browser
View In Other Browsers
명령 팔레트... Ctrl+Shift+P

Global Components Timeline Plugin settings
Theme Auto Light Dark High contrast
Menu Step Scrolling Enable Debugging info Enable
More settings Devtools plugins Documentation Report a bug What's new

ES6

ES6 프로젝트 생성

```
PS C:\Users\jungw\Documents\myNode> mkdir es6test
```

디렉터리: C:\Users\jungw\Documents\myNode

Mode	LastWriteTime	Length	Name
----	-----	-----	-----
d-----	2024-04-09 오전 12:34		es6test

```
PS C:\Users\jungw\Documents\myNode> dir
```

디렉터리: C:\Users\jungw\Documents\myNode

Mode	LastWriteTime	Length	Name
----	-----	-----	-----
d-----	2024-04-09 오전 12:34		es6test
-a----	2024-04-09 오전 12:19	532	hello-vue.html
-a----	2024-04-08 오후 11:52	110	hello.js

```
PS C:\Users\jungw\Documents\myNode> cd ..\es6test\
```

```
PS C:\Users\jungw\Documents\myNode\es6test> 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 init` 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.
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 init` 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.

ES6 프로젝트 생성

```

Press ^C at any time to quit.
package name: (es6test)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\jungw\Documents\myNode\es6test\package.json:

{
  "name": "es6test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this OK? (yes) yes
PS C:\Users\jungw\Documents\myNode\es6test>

```

문제 출력 디버그 콘솔 터미널 포트

PS C:\Users\jungw\Documents\myNode> npm install --save-dev @babel/core @babel/cli @babel/preset-env core-js

[#.....] | idealTree:make-dir: sill placeDep ROOT semver@6.3.1 OK for: @babel/core@7.24.4 want: ^6[##
/ idealTree:make-dir: sill placeDep ROOT semver@6.3.1 OK for: @babel/core@7.24.4 want: ^6[####
idealTree:make-dir: sill placeDep ROOT commonjs@1.1.1 OK for: @babel/core@7.24.4 want: ^6[####

npm install --save-dev @babel/core @babel/cli
@babel/preset-env core-js

14 packages are looking for funding
run `npm fund` for details

새 터미널(Ctrl+Shift+)
[<Alt>] 터미널 분할(Ctrl+Shift+5)

ES6 프로젝트 생성

- npm으로 프로젝트 생성, 패키지 설치, 명령 실행 등 작업
- 프로젝트 생성하면 package.json과 node_modules 등이 생성

```

1  {
2    "name": "es6test",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    ▶ 디버그
7    "scripts": {
8      "test": "echo \"Error: no test specified\" && exit 1"
9    },
10   "author": "",
11   "license": "ISC"
12 }

```

npm 명령어	설명
npm init	프로젝트 초기화
npm install	package.json의 패키지 설치
npm install --save [패키지명]	패키지를 프로젝트 의존성으로 추가
npm install --save-dev [패키지명]	패키지를 프로젝트 개발 의존성 수준으로 추가
npm install --global [패키지명]	패키지를 전역 수준으로 추가
npm update --save	프로젝트 패키지 업데이트
npm run [스크립트명]	package.json의 스크립트 명령 실행
npm uninstall --save [패키지명]	패키지 삭제
npm cache clean	캐시 삭제

ES6 프로젝트 생성

MYNODE es6test > babel.config.json > [] presets

```

1  {
2      "presets": [ "@babel/env" ]
3  }

```

** 위치 주의!

babel.config.json을 생성, 입력 후 저장

MYNODE es6test > src > babel-test.js > ...

```

1  let name = "john";
2  console.log(`Hello ${name}!!`);

```

src 폴더 생성 후, babel-test.js를 만들어 es6 문법 구현

MYNODE es6test > package.json > ...

```

1  {
2      "name": "es6test",
3      "version": "1.0.0",
4      "description": "",
5      "main": "index.js",
6      "scripts": {
7          "test": "echo \\\"Error: no test specified\\\" && exit 1",
8          "build": "babel src -d build"
9      },
10     "author": "...",
11     "license": "ISC"
12 }

```

내용추가

ES6 프로젝트 생성

ES6

```
PS C:\Users\jungw\Documents\myNode\es6test> npx babel src -d build
Successfully compiled 1 file with Babel (791ms).
```

```
MYNODE
  es6test > src > babel-test.js > ...
    1 let name = "john";
    2 console.log(`Hello ${name}!!`);
```

```
PS C:\Users\ellen\OneDrive\문서\myNode\es6test> npm run build
> es6test@1.0.0 build
> babel src -d build

Successfully compiled 1 file with Babel (525ms).
```

ES5

```
MYNODE
  es6test > build > babel-test.js > ...
    1 "use strict";
    2
    3 var name = "john";
    4 console.log("Hello ".concat(name, "!!"));
```

ES6 프로젝트 생성

babel 사용하는 경우

- npm init
- npm install --save-dev @babel/core @babel/cli @babel/preset-env core-js
- babel.config.json
- src/babel-test.js
- package.json에
 "build": "babel src -d build"
- npm run build 또는 npx babel src -d build
- node ./build/test.js

babel 사용하지 않는 경우
js 바로 실행

- 프로젝트 초기화
 npm init
- 소스코드 작성
 src/test.js
- 소스코드 실행
 node src/test.js

- **var**

- 호이스팅 (hoisting) 문제 발생
- 변수의 선언을 범위의 최상단으로 옮기는 행위
- 단계
 - 내부에 **var** 변수로 선언된 코드를 찾아서 메모리를 미리 할당
 - 호이스팅 후에 코드를 실행함.
 - 일반적인 변수 선언후 변수에 값을 할당하는 룰이 깨짐.
- 함수 단위로 호이스팅
 - 함수 단위의 **scope**은 지역변수로 인식.
 - 함수 이외의 블록 **scope**은 전역변수로 인식.
- 변수 재선언 가능
- 혼란 야기

어학사전

hoisting

영어사전 단어·속어 46

hoisting

명사 끌어 올리기, 들어올려 나르기.

```
console.log(A1);
var A1 = "hello";
```

ok

```
var A1 = 100;
console.log(A1);
var A1 = "hello";
console.log(A1);
```

ok

변수- var, let, const

ES6

```
let msg= "GLOBAL";
function outer(a) {
  let msg = "OUTER";
  console.log(msg);
  if (true) {
    let msg = "BLOCK"
    console.log(msg);
  }
}
```

babel

ES5

```
"use strict";
var msg = "GLOBAL";
function outer(a) {
  var msg = "OUTER";
  console.log(msg);
  if (true) {
    var _msg = "BLOCK";
    console.log(_msg);
  }
}
```

let을 사용하기를
강력 권고

지역변수로 인식하므로
msg변수를 생성

전역변수로 인식하므로
msg별개의 변수를 생성

변수- var, let, const

- **const** : 상수

- 변수에 대한 재할당 불가능
- 객체 자체에 대한 재할당 불가능

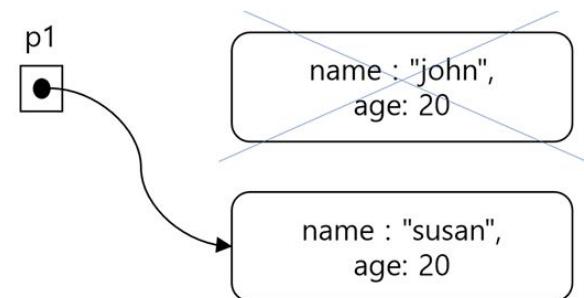
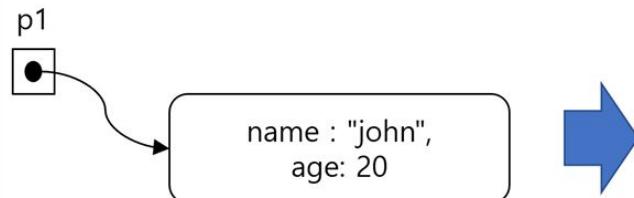
```
const p1 = { name : "john", age : 20 }
//오류 발생
p1 = { name:"susan", age: 20 };

console.log(p1);
```

불가능

```
const name = 'hong';
name = 'kim'; //오류, 재할당 불가
```

VM38:2 Uncaught TypeError: Assignment to constant variable.
at <anonymous>:2:4



**불가능-
주소를
변경 불가**

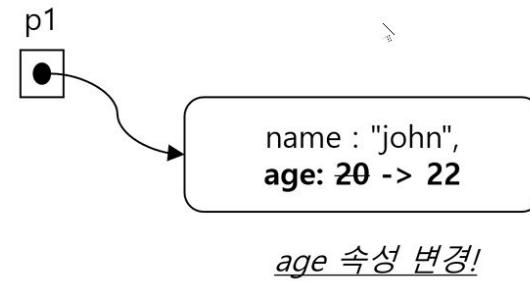
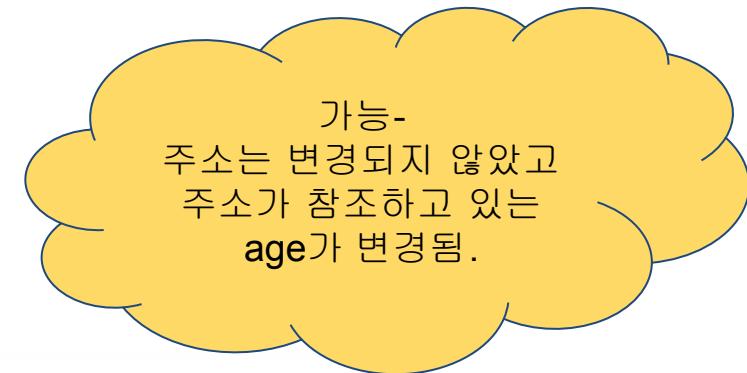
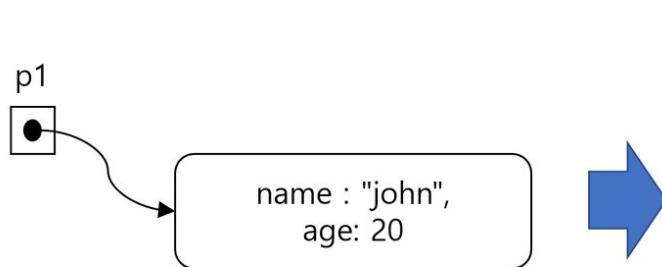
변수- var, let, const

- **const** : 상수

- 객체 자체에 대한 재할당 불가능
- 객체내부의 값 재할당 가능

```
const p1 = { name : "john", age : 20 }
p1.age = 22;

console.log(p1);
```



가변 파라메터

- 0개 이상의 값을 함수의 입력값으로 받을 때
- 가변적인 함수의 입력값이 있는 경우
- 입력 파라메터의 마지막에 설정하여 가변적인 여러 개의 값을 입력값으로 받을 수 있다.

```
function addContact(name, mobile,
    home="없음",
    address="없음",
    email="없음") {
    var str = `name=${name}, mobile=${mobile}, home=${home},
    address=${address}, email=${email}`;
    console.log(str);
}

addContact("홍길동", "010-222-3331")
addContact("이몽룡", "010-222-3331", "02-3422-9900", "서울시");
```

```
PS C:\Users\jungw\Documents\myNode\es6test> npx babel src -d build
Successfully compiled 2 files with Babel (672ms).
```

```
PS C:\Users\jungw\Documents\myNode\es6test> node .\build\test.js
name=홍길동, mobile=010-222-3331, home=없음,
address=없음, email=없음
name=이몽룡, mobile=010-222-3331, home=02-3422-9900,
address=서울시, email=없음
```

가변 파라메터

```
function foodReport(name, age, ...favoriteFoods) {  
    console.log(name + ", " + age);  
    console.log(favoriteFoods);  
}  
  
foodReport("이몽룡", 20, "짜장면", "냉면", "불고기");  
foodReport("홍길동", 16, "초밥");
```

마지막에 위치해야함.

```
PS C:\Users\jungw\Documents\myNode\es6test> npx babel src -d build  
Successfully compiled 143 files with Babel (5712ms).
```

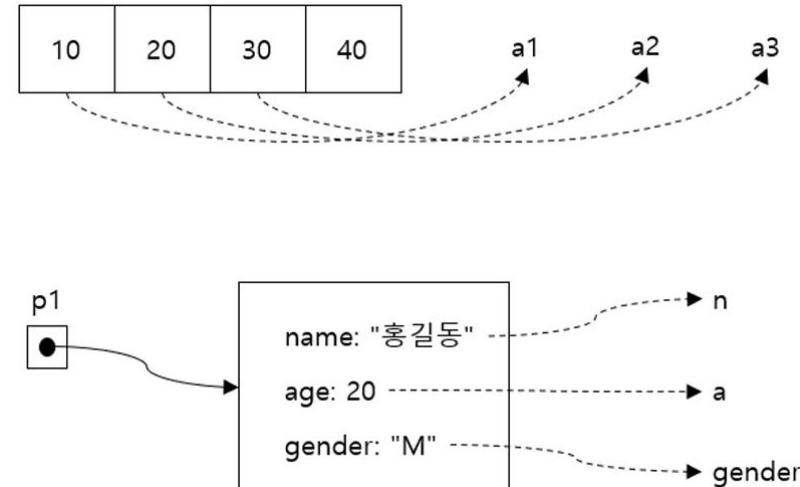
```
이몽룡, 20  
[ '짜장면', '냉면', '불고기' ]  
홍길동, 16  
[ '초밥' ]
```

구조분할 할당

- Destructuring Assignment
- 배열, 객체의 값들을 여러 변수에 나누어 할당할 수 있는 기능(자바의 String.split()과 유사)

```
let arr = [10, 20, 30, 40];
let [a1, a2, a3] = arr;
console.log(a1, a2, a3);

let p1 = { name:"홍길동", age:20, gender:"M" };
let { name: n, age:a, gender } = p1;
console.log(n, a, gender);
```



```
PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node 02-07.js
10 20 30
홍길동 20 M
```

구조분할 할당

```

function addContact1({name, phone, email="이메일없음", age=0}) {
    console.log(name, phone, email, age);
}

addContact1({ name : "이몽룡", phone : "010-3434-8989" })

function addContact2(contact) {
    if (!contact.email) contact.email = "이메일없음";
    if (!contact.age) contact.age = 0;
    let { name, phone, email, age } = contact;
    console.log(name, phone, email, age);
}

addContact2({ name : "이몽룡", phone : "010-3434-8989" })

function addContact3(name, phone, email="이메일없음", age=0) {
    console.log(name, phone, email, age);
}

addContact3("이몽룡", "010-3434-8989")

```

구조분할 할당을 하면
하나씩 꺼내어
여러줄에 걸쳐 변수에
옮길 필요가 없다.

구조분할 할당

Object

```

PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node 02-08.js
이몽룡 010-3434-8989 이메일없음 0
이몽룡 010-3434-8989 이메일없음 0
이몽룡 010-3434-8989 이메일없음 0

```

구조분할 할당

- Object을 구조분할 할당하는 경우는
key와 동일한 이름으로 하면 자동으로 분할하여 할당되지만,
key와 다른 경우 다른 변수명을 할당해주어야 함.

구조분할 할당은
{ }괄호 사용

```
//let { name, phone, email, age} = contact;  
let { name: n, phone, email, age} = contact;  
// console.log(name,phone,email,age);  
console.log(n,phone,email,age);
```

```
PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node 02-08.js  
이동룡 010-3434-8989 이메일없음 0  
이동룡 010-3434-8989 이메일없음 0  
이동룡 010-3434-8989 이메일없음 0
```

화살표 함수

- 간단하게 화살표 기호를 이용하여 함수를 정의 가능
- 트랜스파일 결과를 일반 함수 정의와 동일함.

```
const test1 = function(a,b) {  
    return a+b;  
}
```

```
const test2 = (a,b) =>{  
    return a+b;  
};
```

```
const test3 = (a,b) => a+b;
```

```
console.log(test1(3,4));  
console.log(test2(3,4));  
console.log(test3(3,4));
```

모두 동일한 함수 정의

```
PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node 02-09.js  
7  
7  
7
```

화살표 함수

- this 주의!

```
var obj = { result: 0 };
obj.add = function(x,y) {
    this.result = x+y;
}
obj.add(3,4)
console.log(obj)          // { result: 7 }
```

this → 전역변수
객체인 obj

```
PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node 02-10.js
{ result: 7, add: [Function (anonymous)] }
```

```
a = 100;
console.log(a);
```

가능
변수에 값을 바로
할당하는 경우
var로 인식

화살표 함수

```

var obj = { result: 0 };
obj.add = function(x,y) {
    this.result = x+y;
}
var add2 = obj.add;
//함수기능만 필요한 경우 따로 add2변수로 저장하여 사용 가능
console.log(add2 === obj.add)          //true, 동일한 함수
add2(3,4)
//add2는 함수만 가지고 있음.
//this.result라고 했을 때 this라는 것을 지정할 수 없음.
//전역변수 result를 만들고, result에 70이 할당됨.
//JS는 전역변수를 선언하지 않고 할당하면 그 시점에서 변수가 선언됨.
//헷갈림!
console.log(obj);           //{ result: 0 }
//원래의 객체인 obj는 변함이 없음.
console.log(result);        //7
//전역변수 result가 생성되고 값도 할당되어있음.

```

```

PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node 02-11.js
true
{ result: 0, add: [Function (anonymous)] }

```

화살표 함수

- **this**는 함수가 호출되는 시기에 바인딩됨.

- `apply()`, `call()`, `bind()` 함수

```
var add = function(x,y) {
  this.result = x+y;
}
```

```
var obj = {};
//1. apply() 사용
//add.apply(obj, [3,4])
//2. call() 사용
//add.call(obj,3,4)
//3. bind() 사용
add = add.bind(obj);
add(3,4)
```

```
console.log(obj); // { result : 7 }
```

`bind()`함수는
입력값이 `obj`이므로
기존 함수와 연결해보면
`obj +` 함수가 포함되어있는
새로운 함수가 리턴됨.

```
PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node 02-12.js
{ result: 7 }
```

화살표 함수

- 함수가 중첩되어 있는 경우 **this**의 상속이 다름.

- 일반함수 - **this**상속되지 않음.
- 화살표함수 - **this**상속됨.

```
var obj = { result:0 };
obj.add = function(x,y) {
  function inner() {
    this.result = x+y;
  }
  inner();
}
obj.add(3,4)

console.log(obj)      // { result: 0 }
console.log(result)   // 7
```

this는 전역 객체
result 전역변수 생성후
할당됨.

클로저에서
함수를 기능을
은닉하기 위해 함수를
겹쳐서 정의

obj에 값을 할당하고자
하는 경우
1) **apply()** 등의 함수로 변경
2) 화살표함수 사용.

화살표 함수

1. bind() 해결

```
var obj = { result:0 };
obj.add = function(x,y) {
  function inner() {
    this.result = x+y;
  }
  inner = inner.bind(this);
  inner();
}
obj.add(3,4)

console.log(obj)      // { result: 7 }
```

2. 화살표함수로 해결

```
var obj = { result:0 };
obj.add = function(x,y) {
  const inner = () => {
    this.result = x+y;
  }
  inner()
}
obj.add(3,4)

console.log(obj)      // { result: 7 }
```

this 객체를 가진 obj + inner()함수인
새로운 함수 리턴, outer함수에서 this는 obj

화살표함수는 outer함수의 this바인딩을 상속받음.

객체 리터럴

- 리터럴 : 소스 코드에서
특정한 자료형의 값을 직접 표현하는 방식
- 속명명과 변수명이 동일한 경우 한번만
씀.

```
var name = "홍길동";
var age = 20;
var email = "gdhong@test.com";

//var obj = { name: name, age: age, email: email };
var obj = { name, age, email };
console.log(obj);
```

1. Literal

[편집]

[한국어 위키백과 리터럴 문서](#)

[영어 위키백과 리터럴 문서](#)

프로그래밍 언어에서, 리터럴(literal)이란 소스 코드에서 특정한 자료형의 값을 직접 표현하는 방식을 말한다. 자세한 문법은 언어마다 다르지만, 이를테면 정수형 데이터일 경우 10진수라는 것을 표현하려면 앞에 아무 것도 안 붙이고 10 같은 식으로 쓰고, 16진수라는 것을 표현하려면 앞에 0x를 붙여 0xF3 같은 식으로 쓴다. 실수형 데이터일 경우 만약 double 자료형으로 썼다면 앞에 아무 것도 안 붙이고 3.14 같은 식으로 쓰고, float 자료형(주전되지 않음)으로 썼다면 뒤에 f를 붙여 10f 같은 식으로 쓰며, 10의 거듭제곱을 표현해야 할 경우 0.1E-5^[1] 같은 식으로 쓴다. 문자형(char형)의 경우 날자 앞뒤로 [작은따옴표\(\)](#)를 붙여 'a' 같은 식으로 쓰고, 문자열의 경우 [C언어](#)에서는 char형 날자의 배열로 선언하고, [Java](#)에서는 String 클래스로 선언한 다음 문자열 앞뒤로 [큰따옴표\("\)](#)를 붙여 "Hello world" 같은 식으로 쓴다.

- 객체내 메서드 표기법

```
let p1 = {  
    quantity : 2,  
    order : function() {  
        if (!this.amount) {  
            this.amount = this.quantity * this.price;  
        }  
        console.log("주문금액 : " + this.amount);  
    },  
    discount(rate) {  
        if (rate > 0 && rate < 0.8) {  
            this.amount = (1-rate) * this.price * this.quantity;  
        }  
        console.log((100*rate) + "% 할인된 금액으로 구매합니다.");  
    }  
}  
p1.discount(0.2);  
p1.order();
```

방법1- 속성명: 익명 함수 형태

방법2- 일반 함수 형태

PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node 02-17.js

20% 할인된 금액으로 구매합니다.

주문금액 : 320000

템플릿 리터럴

- `(백틱, backtick)으로 묶으면, \${}에 데이터를 넣어 원하는 문자열을 만들 수 있음.
- 맥에서는 option + ~
- `\${ }` : 수식, 변수, 리턴이 있는 함수() 가능

```
const d1 = new Date();
let name = "홍길동";
let r1 = `${name} 님에게 ${d1.toDateString()}에 연락했다.`;
console.log(r1);

let product = "갤럭시S7";
let price = 199000;
let str = `${product}의 가격은
${price}원 입니다.`;
console.log(str);
```

PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node 02-18.js
홍길동 님에게 Tue Apr 09 2024에 연락했다.
갤럭시S7의 가격은
199000원 입니다.

모듈(module)

- 특정한 목적을 가지고 만든 파일들의 모음(== 라이브러리와 유사)
- JS에서 **import**의 단위

node_modules

```
> .bin
> @ampproject
> @babel
> @jridgewell
> @nicolo-ribaudo
> ansi-styles
> anymatch
> babel-plugin-polyfill-corejs2
> babel-plugin-polyfill-corejs3
> babel-plugin-polyfill-regene...
> balanced-match
```

```
> binary-extensions
> brace-expansion
> braces
> browserslist
> caniuse-lite
> chalk
> chokidar
> color-convert
> color-name
> commander
> concat-map
> convert-source-map
> core-js
```

대표적인 모듈 시스템

- AMD : 가장 오래됨.
require.js 라이브러리를 통해 처음 개발
- CommonJS : Node.js 서버를 위해 개발
- UMD : AMD, CommonJS를 함께 사용하기 위해 개발

2015년 표준으로 등재

모듈(module)

- 일반적으로 스크립트 파일 하나는 모듈 하나
- 모듈 안에서(하나의 파일 안에서) 선언된 변수, 함수는 파일 내 scope를 가진다.
 - private, local 범위라 부름.
- 자바에서처럼 public으로 선언하고 싶으면 export로 선언해주면 된다.



모듈(module)

MYNODE

- es6test
- src
 - ch02
 - > build
 - modules
 - 02-19-module.js**
 - 02-01.js
 - 02-02.js

```
es6test > src > ch02 > modules > JS 02-19-module.js > ...
1 let base = 100;
2 const add = (x) => base+x;
3 const multiply = (x) => base*x;
4 const getBase = ()=>base;
5
6 export { add, multiply };
7 export default getBase;
8
```

export

MYNODE

- es6test
- src
 - ch02
 - 02-20-main.js**
 - 02-21.js

```
es6test > src > ch02 > JS 02-20-main.js
1 import getBase, { add } from './modules/02-19-module.js';
2
3 console.log(add(4));
4 console.log(getBase());
5
```

import

```
PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node .\02-20-main.js
104
100
```

Promise

Promise

- 비동기 처리를 하는 경우 **Callback Hell**에 자주 빠짐.
- 콜백함수(함수를 매개변수로 넣어 실행)들이 중첩되어 순서를 정의하기가 쉽지 않음.
 - 에러를 찾기 어려움.
 - 예외처리를 정의하기 어려움.
- ES6 Promise**를 이용해 작업의 순서를 명확히 정의할 수 있음.

```
getData(function(a){
  getMoreData(a, function(b){
    getMoreData(b, function(c){
      getMoreData(c, function(d){
        getMoreData(d, function(e){
          ...
        });
      });
    });
  });
});
```

```
const p = new Promise((resolve, reject) => {
  //비동기 작업 처리
  //resolve() 함수 호출되면 아래의 then()에 정의된 내용 처리됨.
  //reject() 함수 호출되거나 에러 발생시 catch()에 정의된 내용
  //처리됨.
});

p.then((result) => {
})
.catch((error) => {
})
```

Promise

- 가장 일반적인 Promise 패턴

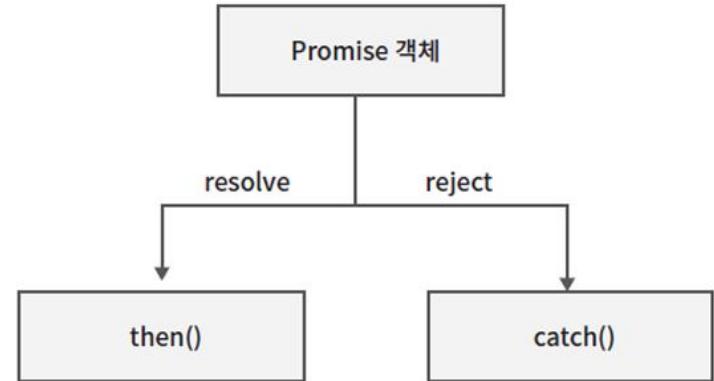
```

const p = new Promise((resolve, reject) => {
  setTimeout(()=> {
    var num = Math.random();          // 0~1사이의 난수 발생
    if (num >= 0.8) {
      reject("생성된 숫자가 0.8이상임 - " + num);
    }
    resolve(num);
  }, 2000)
}

p.then(result=> {
  console.log("처리 결과 : ", result)
})
.catch(error=>{
  console.log("오류 : ", error)
})

console.log("## Promise 객체 생성!")

```



```

PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node .\02-21.js
## Promise 객체 생성!
처리 결과 : 0.39033194833869134

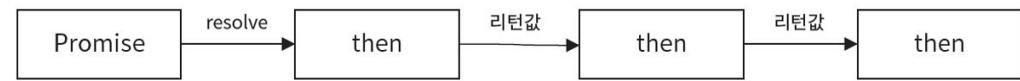
```

Promise

- 명확한 단계별 비동기 처리 정의 : **chain**
- then**함수의 리턴값이 다음 함수로 전달 가능

```
var p = new Promise((resolve, reject)=> {
  resolve("first!")
})

p.then((msg)=> {
  console.log(msg);
  throw new Error("## 에러!!");
  return "second";
})
.then((msg)=>{
  console.log(msg);
  return "third";
})
.then((msg)=>{
  console.log(msg);
})
.catch((error)=> {
  console.log("오류 발생 ==> " + error)
})
```



```
PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node .\02-22.js
first!
오류 발생 ==> Error: ## 에러 !!
```

Promise

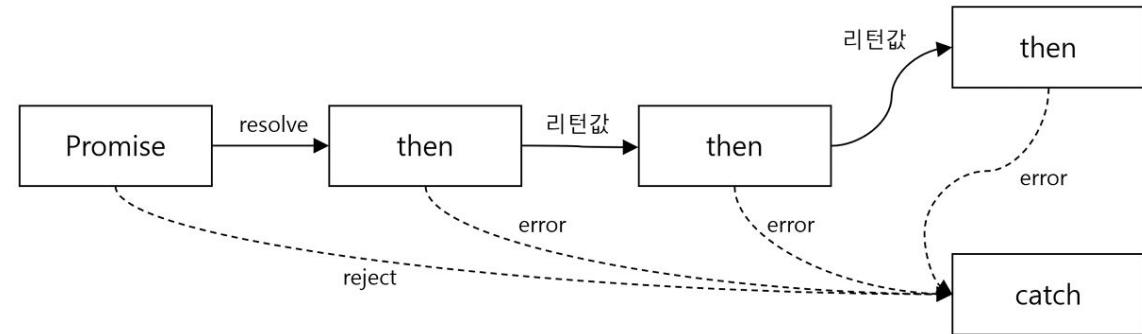
- then()내부에서 오류가 발생하면 가장 가까운 catch()가 호출됨.

```

var p = new Promise((resolve, reject)=> {
  resolve("first!")
})

p.then((msg)=> {
  console.log(msg);
  throw new Error("## 에러!!")
  return "second";
})
.then((msg)=>{
  console.log(msg);
  return "third";
})
.then((msg)=>{
  console.log(msg);
})
.catch((error)=> {
  console.log("오류 발생 ==> " + error)
})

```



```

PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node .\02-22.js
first!
오류 발생 ==> Error: ## 에러 !!

```

전개 연산자

- **spread operator**라고 부름
- 객체나 배열을 복사할 때 사용
- 기존 객체나 배열은 그대로 두고 새로운 객체나 배열을 생성하여 변수에 할당

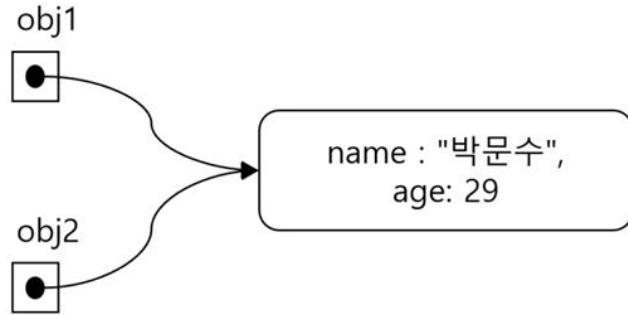
```
let obj1 = { name: "박문수", age: 29 };
let obj2 = obj1;           //shallow copy! obj1, obj2는 동일한 객체를 참조
let obj3 = { ...obj1 };   //객체 내부의 값은 복사하지만 obj3, obj1은 다른 객체
let obj4 = { ...obj1, email: "mspark@gmail.com" }; //새로운 속성 추가
```

```
obj2.age = 19;
console.log(obj1); // { name: "박문수", age: 19 }
console.log(obj2); // { name: "박문수", age: 19 } obj1과 동일!!
console.log(obj3); // { name: "박문수", age: 29 } age가 바뀌지 않음
console.log(obj1 == obj2); // true
console.log(obj1 == obj3); // false
```

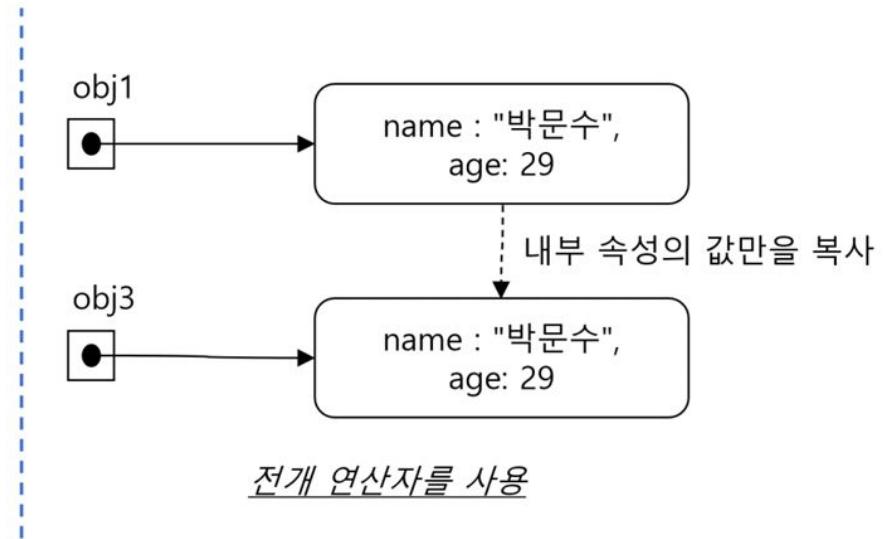
```
let arr1 = [ 100, 200, 300 ];
let arr2 = [ "hello", ...arr1, "world" ];
console.log(arr1);      // [ 100, 200, 300 ]
console.log(arr2);      // [ "hello", 100, 200, 300, "world" ]
```

```
PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node .\02-23.js
{ name: '박문수', age: 19 }
{ name: '박문수', age: 19 }
{ name: '박문수', age: 29 }
true
false
[ 100, 200, 300 ]
[ 'hello', 100, 200, 300, 'world' ]
```

전개 연산자



얕은 복사(Shallow Copy)



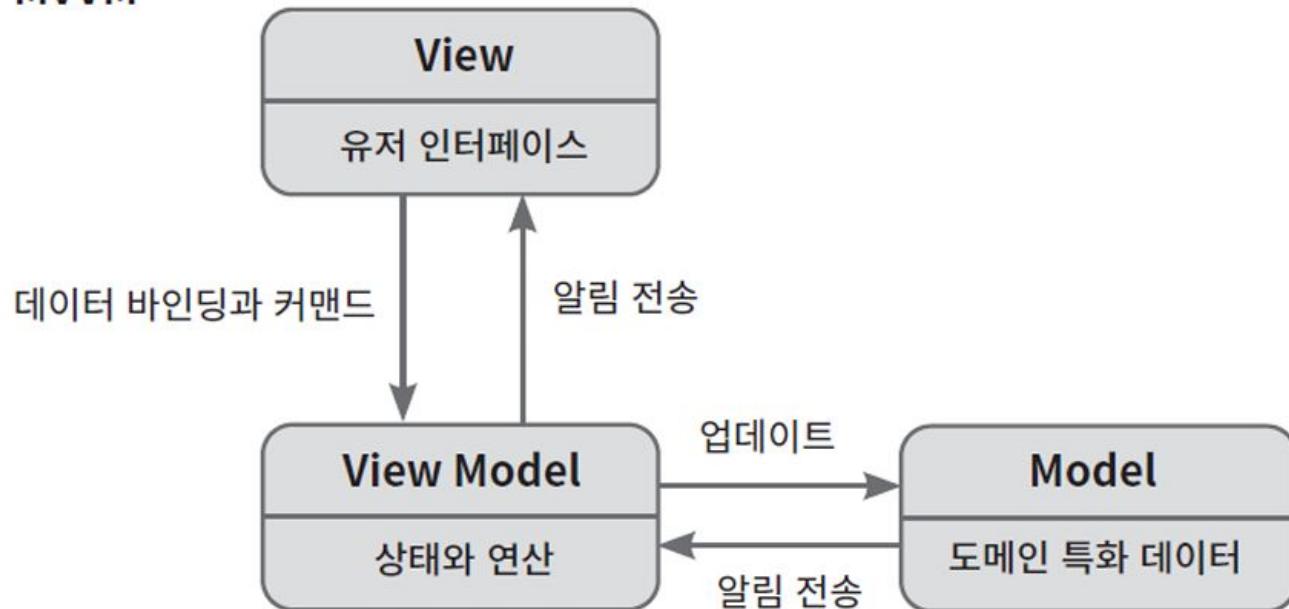
전개 연산자를 사용

Proxy

Proxy

- 객체의 속성에 값을 할당하거나 읽어올 때 특정한 작업을 정의할 수 있음.
- 일반적으로 직접적으로 구현하지는 않음.
- Vue3의 반응성에 Proxy가 내부적으로 사용됨.

MVVM



Proxy

```

let obj = { name : "홍길동", age :20 };
const proxy = new Proxy(obj, {
    get: function(target, key) {
        console.log("## get " + key)
        if (!target[key]) throw new Error(`존재하지 않는 속성(${key})입니다.`);
        return target[key];
    },
    set : function(target, key, value) {
        console.log("## set " + key)
        if (!target[key]) throw new Error(`존재하지 않는 속성(${key})입니다.`);
        target[key] = value;
        return true;
    }
})

console.log(proxy.name);           //읽기 작업 get 호출
proxy.name = "hahaha";           //쓰기 작업 set 호출
proxy.age = 30;                   //쓰기 작업 set 호출

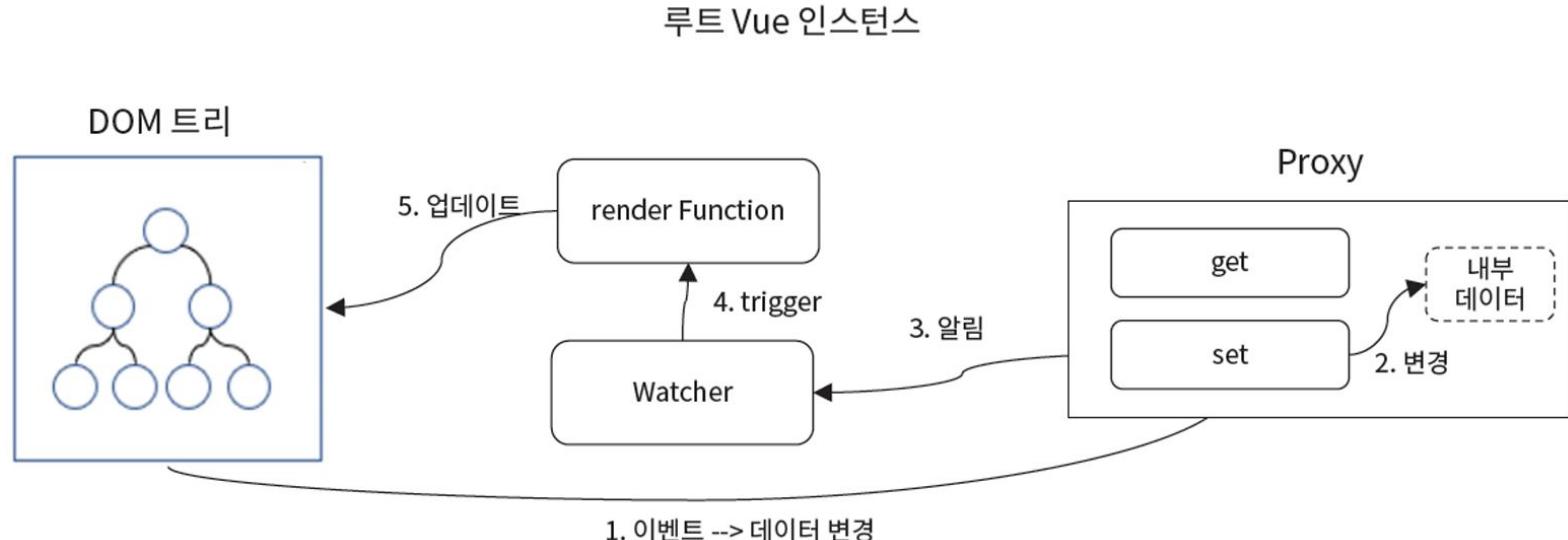
```

```

PS C:\Users\jungw\Documents\myNode\es6\test\src\ch02> node .\02-24.js
## get name
홍길동
## set name
## set age

```

Proxy



Proxy

```
var arr = [10,20,30];

const proxy = new Proxy(arr, {
  get: function(target, key, receiver) {
    console.log("## get " + key)
    if (!target[key]) throw new Error(`존재하지 않는 속성(${key})입니다`);
    return target[key];
  },
  set : function(target, key, value) {
    console.log("## set " + key)
    if (!target[key]) throw new Error(`존재하지 않는 속성(${key})입니다`);
    target[key] = value;
    return true;
  }
})

proxy[1] = 99;
//proxy[4] = 100; //오류발생
```

```
PS C:\Users\jungw\Documents\myNode\es6test\src\ch02> node .\02-25.js
## set 1
```

핵심 정리

- **VueJS 필요성**
- **ES6와 VueJS**
- **MVVM 패턴**
- **변수 선언 - var, let, const**
- **가변 파라메터**
- **구조변환 할당**
- **화살표 함수**
- **객체 리터럴**
- **템플릿 리터럴**
- **모듈**
- **Proxy**
- **Promise**