

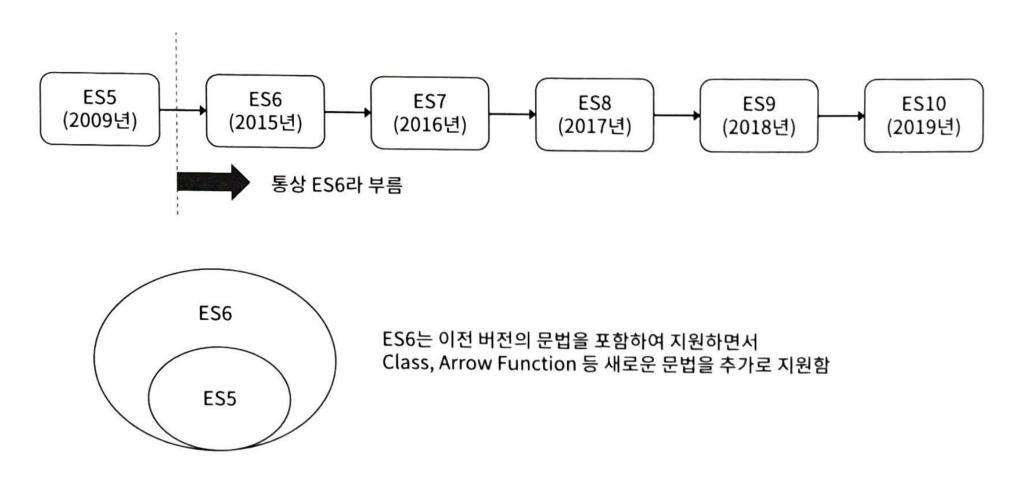
2025년 상반기 K-디지털 트레이닝

Vue.js를 위한 ES6

[KB] IT's Your Life

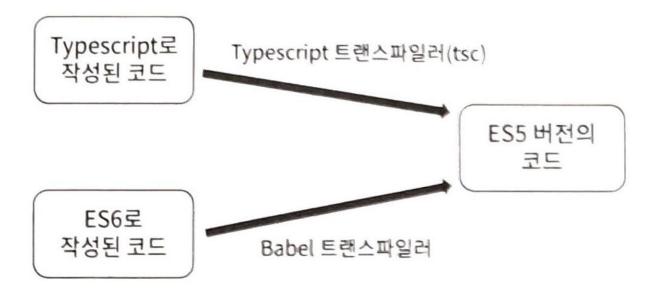


ES6(ECMAScript6)



ES6 미지원 브라우저에 대한 대처

- 트랜스파일러(Transpiler)를 이용해서 ES6 코드를 ES5 코드로 변환
- babel, tsc(typescript 트래스파일러) 등
- vue.js 개발 환경에서 자동으로 처리 해줌



▽ var 변수의 호이스팅

- 자바스크립트가 실행될 때 먼저 선언문을 먼저 해석 후 실행
- 변수의 선언문이 사용하는 문장 후에 있어도 해당 변수를 인식

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

○ 함수 단위로 호이스팅

- 블록 단위로 동작하지 않음
- 진정한 지역변수가 아님

○ 변수 선언을 여러 번 할 수 있음 (에러가 아님)

```
var V1 = 100;
console.log(V1);  // 100
var V1 = "hello";
console.log(V1);  // hello
```

2 let과 const

☑ let 변수

- ES6에서 도입한 블록 단위의 변수 선언
 - 유효 범위가 블록 단위임
- 호이스팅 지원하지 않음
 - 선언하지 않고 사용하면 에러 발생

⊀ KB국민은행

2 let과 const

☑ 02-02.js

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

```
OUTER
BLOCK
```

2 let과 const

const

- 값 변경 불가 → 상수 선언
- 범위(scope)는 let과 동일
- 참조 변수인 경우 참조는 변경 불가
 - 필드는 수정 가능

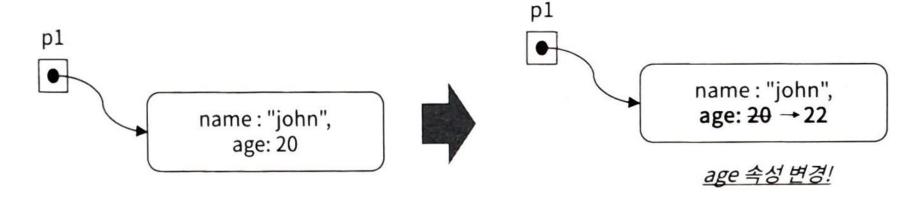
2 let과 const

☑ 02-03.js

```
const p1 = { name: 'john', age: 20 };
p1.age = 22;
console.log(p1);

p1 = { name: 'lee', age: 25 };
```

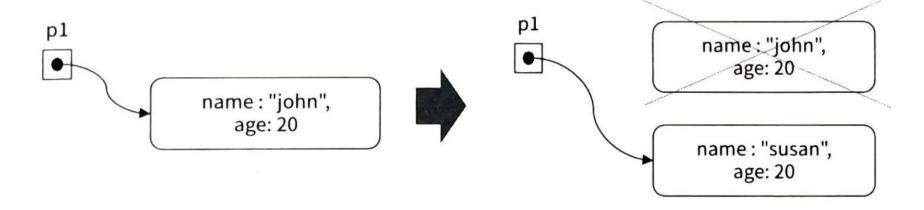
```
{ name: 'john', age: 22 }
```



p1이 참조하는 메모리 주소는 변화 없음

☑ 02-04.js

```
const p1 = { name : "john", age : 20 }
p1 = { name:"susan", age: 20 };
console.log(p1);
```



p1이 참조하는 메모리 주소가 바뀌는 것이므로 허용하지 않음.

- 기본 파라미터 Default Parameter
 - o 함수 호출 시 인수를 생략했을 때 가지는 기본값을 지정

```
function func(arg1, arg2 = 기본값, arg3 = 기본값, ...) { ... }
```

- 반드시 뒤부분에 지정
 - 중간에만 기본값 지정하는 것은 불가능

기본 파라미터와 가변 파라미터

2 02-05.js

```
function addContact(name, mobile, home="없음", address="없음", email="없음") {
    let 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");
```

```
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
```

3

- 가변 파라미터 Rest Paramter
 - 전달하는 파라미터의 개수를 가변적으로 적용
 - ㅇ 매개변수 앞에 ... 연산자 지정
 - 전달된 인수를 매개변수로 매칭한 후,
 - 매칭되지 않은 인수를 모아 배열로 지정 → 가변 파라미터로 전달

```
function func(arg1, arg2, ...argv) { ... }
```

○ 1개만 지정가능, 마지막 매개변수에만 지정가능

기본 파라미터와 가변 파라미터

2 02-06.js

```
function foodReport(name, age, ...favoriteFoods) {
   console.log(name + ", " + age);
   console.log(favoriteFoods);
}

foodReport("이몽룡", 20, "짜장면", "냉면", "불고기");
foodReport("홍길동", 16, "초밥");
```

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

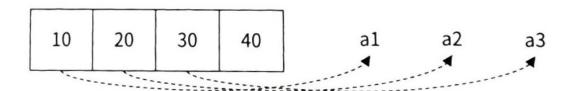
4 구조분해 할당

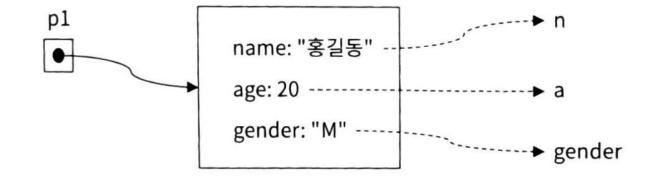
- 구조분해 할당 Destructing Assignment
 - 배열, 객체의 값들을 추출하여 한 번에 여러 변수에 할당할 수 있는 기능

2 02-07.js

```
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);
```

10 20 30 홍길동 20 M





2 02-08.js

```
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")
                                               이몽룡 010-3434-8989 이메일없음 0
                                               이몽룡 010-3434-8989 이메일없음 0
                                               이몽룡 010-3434-8989 이메일없음 0
```

🗸 화살표 함수의 형식

- 기존 함수 표현식 간결화
- 함수의 본체가 한 줄 문장인 경우
 - {}생략
 - 한 줄을 표현식으로 해석하여 자동 리턴(return 생략)

⊀ KB 국민은행

2 02-09.js

```
const test1 = function(a,b) {
    return a+b;
const test2 = (a,b) \Rightarrow {}
    return a+b;
};
const test3 = (a,b) \Rightarrow a+b;
console.log(test1(3,4));
console.log(test2(3,4));
console.log(test3(3,4));
```

☑ 자바스크립트의 this

- 메서드, 함수가 호출될 때마다 현재 호출중인 메서드를 보유한 객체가 this로 연결
- 현재 호출 중인 메서드를 보유한 객체 없다면 전역 객체가 연결
 - 전역 객체
 - 브라우저에서는 window 객체

2 02-10.js

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

{ result: 7, add: [Function (anonymous)] }

☑ 02-11.js

```
let obj = { result: 0 };

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

let add2 = obj.add;

console.log(add2 === obj.add) //true, 동일한 함수

add2(3,4) //전역변수 result에 7이 할당됨.
console.log(obj); //{ result: 0 }
console.log(result); //7
```

```
true
{ result: 0, add: [Function (anonymous)] }
7
```

☑ 자바스크립트의 this 지정

- 함수나 메서드를 호출할 때 this를 변경할 수 있음
- o bind()
 - 지정한 객체를 this로 미리 연결한(binding) 새로운 함수를 리턴
- o apply(), call()
 - 지정한 객체를 this로 연결한 후 함수를 직접 호출

2 02-12.js

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

```
{ result: 7 }
```

- ◎ 화살표 함수에서의 this
 - 함수를 정의하는 영역의 this를 그대로 전달

⊀ KB 국민은행

화살표 함수

☑ 02-13.js

```
let 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
```

☑ 02-14.js

```
let 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 }
```

☑ 02-15.js

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

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

💟 객체 리터럴

○ 기존 객체 리터럴

```
let name = "홍길동";
let age = 20;
let obj = {
name: name,
age: age
};
```

○ ES6 객체 리터럴

■ 변수명이 객체의 속성명과 일치하는 경우 변수명 제시 가능 let name = "홍길동"; let age = 20; let obj = { name, age };

o 메서드인 경우 function 키워드 없이 바로 선언

2 02-16.js

```
let name = "홍길동";
let age = 20;
let email = "gdhong@test.com";
//let obj = { name: name, age: age, email: email };
let obj = { name, age, email };
console.log(obj);
```

{ name: '홍길동', age: 20, email: 'gdhong@test.com' }

☑ 02-17.js

```
let p1 = {
   name : "아이패드",
   price : 200000,
   quantity: 2,
   order : function() { // 기존 방법
       if (!this.amount) {
           this.amount = this.quantity * this.price;
       console.log("주문금액 : " + this.amount);
   discount(rate) { // ES6의 메서드 선언
       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();
```

20% 할인된 금액으로 구매합니다.

주문금액 : 320000

템플릿 리터럴

템플릿 리터럴(Template Literal)

- ``로 묶여진 문자열에서 템플릿 대입문(\${표현식})을 이용해 동적으로 문자열을 끼워 넣어 구성하는 법
- 개행 문자를 포함하여 여러 줄로 작성 가능

★ KB 국민은행

☑ 02-18.js

```
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.`;
                                            홍길동 님에게 Mon Jan 15 2024에 연락했다.
console.log(str);
                                            갤럭시S7의 가격은
```

199000원 입니다.

ES6 모듈

- ㅇ 모듈
 - JS 코드를 포함하고 있는 파일
- o import, export 구문을 이용해서 모듈을 가져오거나, 내보낼 수 있음
- 모듈 내부에서 선언된 모든 변수, 함수, 객체, 클래스는 지역적인(local) 것으로 간주
- 재사용 가능한 모듈을 만들려면 반드시 외부에 공개하는 것을 export 해야 함 export let a = 1000;
 export function f1(a) { ... }
 export { n1, n2 as othername, ...}

 // as를 사용해서 기존 이름(n2) 대신 다른 이름(othername)으로 사용
- o export된 모듈은 다른 모듈에서 import 구문으로 참조하여 사용

✓ npm init -y

package.json

```
""main": "index.js",
   "type": "module",
   "scripts": {
      "test": "echo \"Error: no test specified\" && exit 1"
   },
   ...
}
```

modules/02-19-module.js

```
let base = 100;
const add = (x) => base+x;
const multiply = (x) => base*x;

export { add, multiply };
```

```
let base = 100;
export const add = (x) => base+x;
export const multiply = (x) => base*x;
```

☑ 02-20-main.js

```
import { add } from './modules/02-19-module.js';
console.log(add(4));
```

☑ 02-19-module.js

```
let base = 100;
const add = (x) => base+x;
const multiply = (x) => base*x;
const getBase = ()=>base;

export { add, multiply };
export default getBase;
```

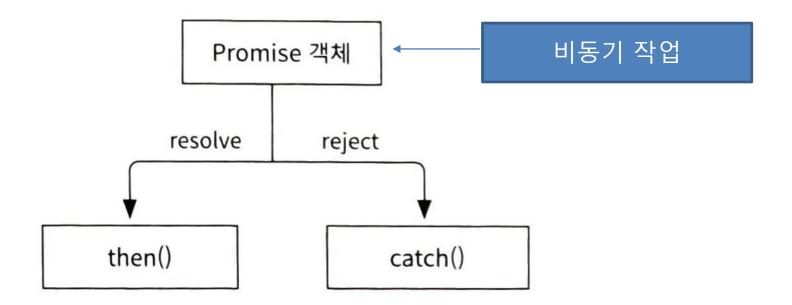
```
let base = 100;
export const add = (x) => base+x;
export const multiply = (x) => base*x;
export default () => base;
```

2 02-20-main.js

```
import getBase, { add } from './modules/02-19-module.js';
console.log(add(4));
console.log(getBase());
```

Promise 객체

- 비동기 처리 지원
- 콜백 함수
 - 비동기 작업의 순차 처리 시 콜백 함수의 중첩 문제 발생
- o Promise 패턴



2 02-21.js Promise 객체

```
const p = new Promise((resolve, reject) => {
   setTimeout(()=> {
       let 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)
                                      ## Promise 객체 생성!
})
                                      처리 결과 : 0.32435778048671526
console.log("## Promise 객체 생성!");
                                      ## Promise 객체 생성!
```

오류 : 생성된 숫자가 0.8이상임 - 0.811963599046789

☑ 02-22.js

Promise 체인 - 비동기 작업의 순차 실행

```
let 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)
})
```

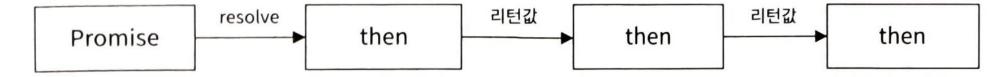
```
first!
second
third

first!

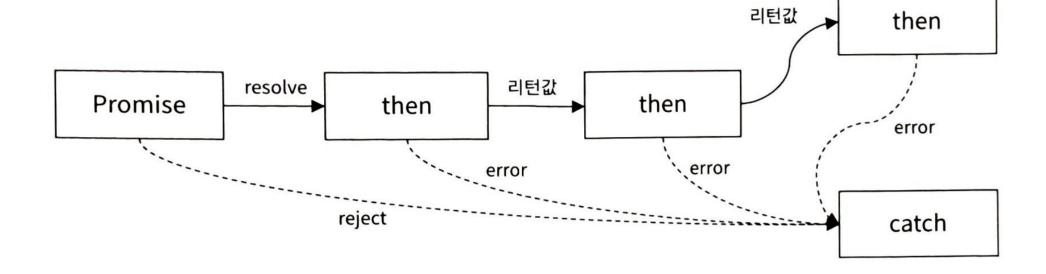
오류 발생 ==> Error: ## 에러!!
```

Promise 체인

ㅇ 정상 처리 시

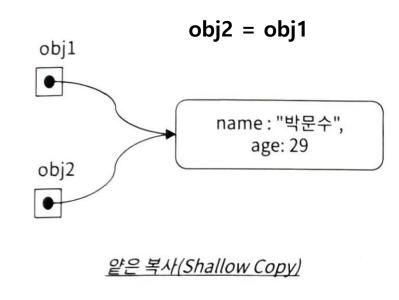


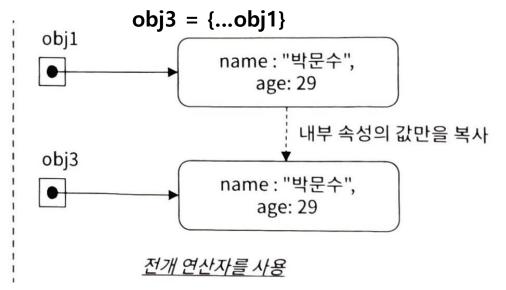
○ 오류 발생 시



전개 연산자 spread operator

- …연산자
- 배열이나 객체를 ...연산자와 함께 객체 또는 배열에서 사용
 - → 객체, 배열 내의 값을 분해된 값으로 전달
- 기존 값을 복사하거나, 기존 값을 복사하면서 새로운 항목 추가시 사용
 - let obj1 = { ... }
 - let obj2 = obj1
 - let obj3 = { ...obj1 }
 - let obj4 = { ...obj1, color }





2 02-23.js

console.log(arr1); // [100, 200, 300]

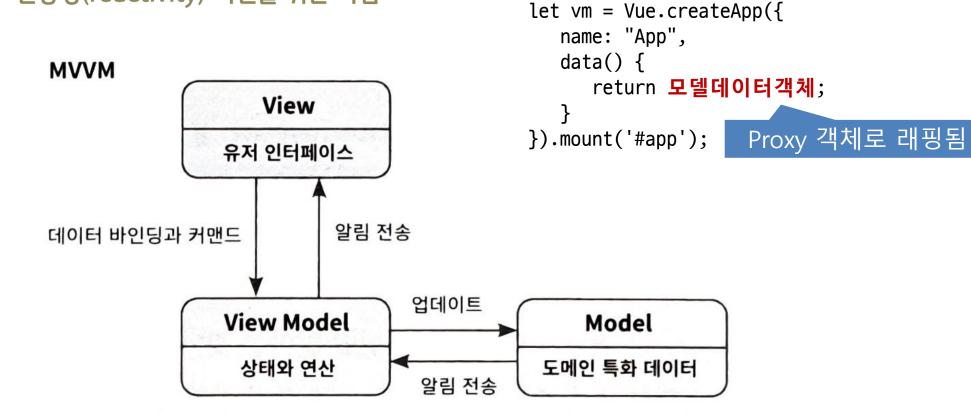
console.log(arr2); // ["hello", 100, 200, 300, "world"]

```
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
obi2.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(obi1 == obi3); //false
let arr1 = [ 100, 200, 300 ];
                                                             { name: '박문수', age: 19 }
let arr2 = [ "hello", ...arr1, "world"];
```

```
{ name: '박문수', age: 19 }
{ name: '박문수', age: 29 }
true
false
[ 100, 200, 300 ]
[ 'hello', 100, 200, 300, 'world' ]
```

Proxy

- 객체의 속성을 읽어오거나 설정하는 작업을 가로채기 위해 래핑할 수 있도록 하는 객체
- 객체의 속성에 접근할 때 개발자가 원하는 지정된 작업을 수행하도록 할 수 있음
- Vue 내부에서Proxy를 사용
 - 개발자가 직접 개발하지 않음
- o 반응성(reactivity) 지원을 위한 기법

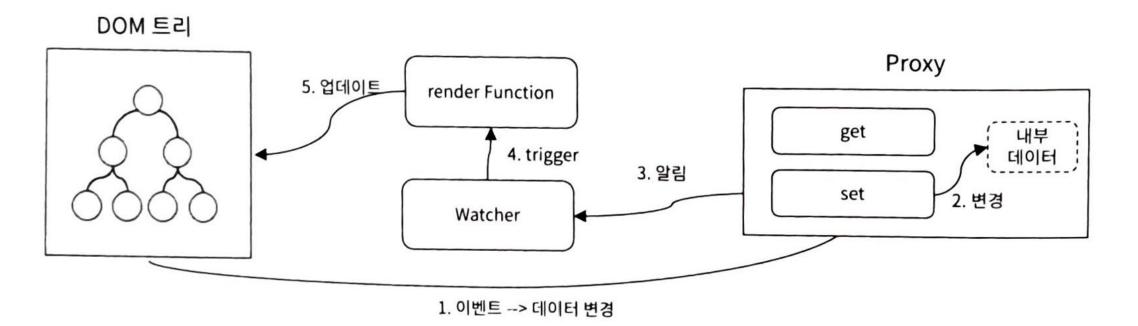


☑ 02-24.js Proxy로 객체 래핑하기

```
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 = "이몽룡";
                             //쓰기 작업 set 호출
                                                        ## get name
                                                        홍길동
                             //쓰기 작업 set 호출
proxy.age = 30;
                                                        ## set name
console.log(proxy);
                                                        ## set age
                                                        { name: '이몽룡', age: 30 }
```

▽ Vue3의 반응성

루트 Vue 인스턴스



☑ 02-25.js Proxy로 배열 래핑하기

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