2025.01.12.월

생성일	@2026년 1월 12일 오전 9:13
태그	JavaScript

목차

▼ 1) JS의 이해

- 더하기 빼기 차이
  - 1 + “2” = “12” (문자열)
  - “3” - 1 = 2 (숫자)
  - “3” + 1 = “31” (문자열)
- 단항 플러스 연산자
  - Number(value) 와 같은 역할
  - +true = 1
  - +false = 0
  - +”123” = 123
  - +”abc” = NaN
  - +null = 0
  - +undeinfed = NaN
- typeof의 배신
  - typeof null = “object” (js 초창기 버그)
  - typeof NaN = “number”
  - null 체크는 value === null (권장)
- == vs ===
  - 1 == “1” ⇒ true (값만 비교)
  - 1 === “1” ⇒ false (값과 자료형 비교)
  - null == undefined ⇒ true
  - null === undefined ⇒ false
- NaN의 정체성 위기
  - NaN === NaN ⇒ false
  - typeof NaN = “number”
  - Number.isNaN(value), value !== value 권장
- 빈 배열 비교

> ⚠️ boolean은 == 연산자에서 false→0, true→1로 형변환
> “” 빈 문자열은 0
> object → truthy, ![ ] → false

  - [ ] == false ⇒ true
  - [ ] == ![ ] ⇒ true

- 연속 비교의 함정
  - 1 < 2 < 3 ⇒ true
  - 3 > 2 > 1 ⇒ false
- 부동소수점 문제
  - 0.1 + 0.2 === 0.3 ⇒ false (2bit의 부동소수점 문제)
- Math.max(), Math.min()
  - Math.max() = -Infinity
  - Math.min() = Infinity
  - Math.max() < Math.min() ⇒ true
- parseInt와 map
  - [”1”, “2”, “3”].map(parseInt) ⇒ [1, NaN, NaN]
  - map은 콜백에 (value, index, array) 3개 인자 전달
  - 해결: .map(x ⇒ parseInt(x)) , .map(Number)
- sort()
  - [1, 2, 10, 20].sort() ⇒ [1, 10, 2, 20] (.sort는 기본적으로 문자열 비교)
  - 해결: .sort((a, b) ⇒ a - b)
- 빈 배열
  - [ ] + [ ] = “”
  - [ ] + { } = “[object Object]”
  - [1, 2] + [3, 4] = “1,23,4” (+ 연산자는 문자열 연결)
- 희소 배열
  - [1, , , 4].length = 4
  - [1, , , 4][1] = undefined
  - 빈 슬롯과 undefined는 다름
- 바나나 문제
  - “b” + “a” + +”a” + “a” = “baNaNa”
  - +”a” ⇒ NaN
  - “b” + “a” + NaN + “a” = “baNaNa”
- 문자열 비교
  - “10” > “9” ⇒ false (유니코드 순에서 9가 더 크다)
  - “10” > 9 ⇒ true (형변환)
- 객체 키의 비밀
  - 객체의 키는 항상 문자열
  - obj = { 1: “a”, “1”: “b”, 1.0: “c” }
  - Object.keys(obj).length = 1
  - obj[1] = obj[”1”] = obj[1.0] = “c”

▼ 2) closure

- 자바스크립트는 렉시컬 스코프를 사용하기 때문에 함수 선언시에 함수가 변수 환경을 기억하는 것

```
function createCounter(initialValue = 0) {
  const _initValue = initialValue;
  let value = _initValue;

  return {
```

```
    increment: () => {
      value = value + 1;
    },
    decrement: () => {
      value = value - 1;
    },
    getValue: () => {
      return value;
    },
    reset: () => {
      return (value = initialValue);
    },
  };
}

// 테스트
const counter = createCounter(10);
const counter2 = createCounter(100);

console.log(counter.increment()); // 11
console.log(counter.increment()); // 12
console.log(counter.decrement()); // 11
console.log(counter.getValue()); // 11
counter.reset();
console.log(counter.getValue()); // 10

console.log(counter2.increment());
console.log(counter2.getValue());
```

- 커링 (curring)
  - 여러개의 인자를 받는 함수를 인자를 하나씩 혹은 나눠서 호출할 수 있는 기법
- 이미터 (emitter)
  - 특정 이벤트가 발생했음을 시스템에 알리는 역할
  - 결합도가 낮은 loose coupling, 수신자 자유롭게 추가/삭제

▼ 3) 비동기 프로그래밍

- 동시성: 여러 작업을 번갈아 처리
- 병렬성: 여러 작업을 같은 시간에 처리 (멀티 코어/멀티 스레드)

```
console.log('1');

setTimeout(() => console.log('2'), 0);

Promise.resolve()
  .then(() => console.log('3'))
  .then(() => console.log('4'));

setTimeout(() => {
  console.log('5');
  Promise.resolve().then(() => console.log('6'));
}, 0);

Promise.resolve().then(() => {
  console.log('7');
  setTimeout(() => console.log('8'), 0);
});
```

```
console.log('9');

// 출력 순서: 1, 9, 3, 7, 4, 2, 5, 6, 8
```

- 이유: 동기 코드 → 마이크로 태스크 → 매크로 태스크 순으로 실행
  1. 동기 (call stack)
  2. 마이크로 태스크
     - promise.them
     - queueMicrotask
  3. 매크로 태스트
     - setTimeout
     - setInterval
  - 마이크로 태스크 안에 등록된 매크로 태스크는 뒤로 밀린다.
- async / await
  - 내부적으로 promise로 구현이 되어있음.
  - async: 비동기 실행
  - await: async 안에서 동기 실행

```
async function async1() {
  console.log('async1 start');
  await async2();
  console.log('async1 end');
}

async function async2() {
  console.log('async2');
}

console.log('script start');

setTimeout(() => console.log('setTimeout'), 0);

async1();

new Promise((resolve) => {
  console.log('promise1');
  resolve();
}).then(() => {
  console.log('promise2');
});

console.log('script end');
```

- Promise

```
const promise = new Promise((resolve, reject) => {
  // todo: 50%
  //  "Success! Chain1"
  //  "Failed" 즉시
  if (Math.random() < 0.5) {
    setTimeout(() => resolve('Success!'), 3000);
  } else {
    reject(new Error('Failed!'));
  }
```

```
});

// 테스트
promise
  .then((value) => {
    console.log(value); // "Success!"
    return value + ' Chain1';
  })
  .then((value) => {
    console.log(value); // "Success! Chain1"
  })
  .catch((e) => console.log(e.message)); // "Failed"

//-------------------------------------------------------
// 재사용 테스트
function makePromise() {
  return new Promise((resolve, reject) => {
    if (Math.random() < 0.5) {
      setTimeout(() => resolve('Success!'), 3000);
    } else {
      reject(new Error('Failed!'));
    }
  });
}

for (let i = 0; i < 5; i++) {
  makePromise()
    .then((value) => {
      console.log(value);
    })
    .catch((e) => console.log(e.message));
}
```

- Promis.all, Promise.allSettled

```
const p1 = fetch('https://jsonplaceholder.typicode.com/todos/1').then((e) =>
  e.json()
);
const p2 = fetch('https://jsonplaceholder.typicode.com/todos/2').then((e) =>
  e.json()
);
const p3 = fetch('https://jsonplaceholder.typicode.com/todos/3').then((e) =>
  e.json()
);

// 동기 프로그램
console.time('sync');
// [p1, p2, p3].map((p) => p.then(console.log));
console.time('p1');
await p1;
console.timeEnd('p1');
console.time('p2');
await p2;
console.timeEnd('p2');
console.time('p3');
await p3;
console.timeEnd('p3');
console.timeEnd('sync');
```

```
// todo: Promise.all 활용 구현
console.time('async');
Promise.allSettled([p1, p2, p3])
  .then((values) => {
    console.log(values);
  })
  .catch((e) => {
    console.log(e.message);
  });
console.timeEnd('async');
```