

\* 예습

1. 5/21

2. 5/22

# 36. 디스트럭처링 할당

## TABLE OF CONTENTS

1. 배열 디스트럭처링 할당

2. 객체 디스트럭처링 할당

디스트럭처링 할당(구조 분해 할당, Destructuring assignment)은 구조화된 배열 또는 객체를 Destructuring(비구조화, 구조파괴)하여 1개 이상의 변수에 개별적으로 할당하는 것을 말한다. 배열 또는 객체 리터럴에서 필요한 값만을 추출하여 변수에 할당할 때 유용하다.

## #1. 배열 디스트럭처링 할당

ES5에서 구조화된 배열을 디스트럭처링하여 1개 이상의 변수에 할당하기 위한 방법은 아래와 같다.

### JAVASCRIPT

```
// ES5
var arr = [1, 2, 3];

var one   = arr[0];
var two   = arr[1];
var three = arr[2];

console.log(one, two, three); // 1 2 3
```

ES6의 배열 디스트럭처링 할당은 배열의 각 요소를 배열로부터 추출하여 1개 이상의 변수에 할당한다. 이때 할당 기준은 배열의 인덱스이다. 즉, 순서대로 할당된다.

## JAVASCRIPT

```
const arr = [1, 2, 3];

// ES6 배열 디스트럭처링 할당
// 변수 one, two, three를 선언하고 배열 arr을 디스트럭처링하여 할당한다.
// 이때 할당 기준은 배열의 인덱스이다.
const [one, two, three] = arr;

console.log(one, two, three); // 1 2 3
```

배열 디스트럭처링 할당을 위해서는 할당 연산자 왼쪽에 값을 할당 받을 변수를 선언해야 한다. 이때 여러 개의 변수를 배열 리터럴 형태로 선언한다.

## JAVASCRIPT

```
let x, y;
[x, y] = [1, 2];

// 위의 문과 아래의 문은 동치이다.
const [x, y] = [1, 2];
```

여러 개의 변수를 배열 형태로 선언하면 반드시 우변에 배열을 할당해야 한다.

## JAVASCRIPT

```
const [x, y];
// → SyntaxError: Missing initializer in destructuring declaration
```

배열 디스트럭처링 할당의 기준은 배열의 인덱스이다. 즉, 순서대로 할당된다. 이때 변수의 개수와 배열 요소의 개수가 반드시 일치할 필요는 없다.

## JAVASCRIPT

```
let x, y, z;
```

```
[x, y] = [1, 2];
console.log(x, y); // 1 2
```

```
[x, y] = [1];
console.log(x, y); // 
```

```
[x, y] = [1, 2, 3];
console.log(x, y); // 1 2
```

```
[x, , z] = [1, 2, 3];
console.log(x, z); // 
```

배열 디스트럭처링 할당을 위한 변수에 **기본값을 설정**할 수 있다.

## JAVASCRIPT

```
let x, y, z;
```

```
// 기본값
```

```
[x, y, z = 3] = [1, 2];
console.log(x, y, z); // 
```

```
//  보다  이 우선한다.
```

```
[x, y = 10, z = 3] = [1, 2];
console.log(x, y, z); // 
```

배열 디스트럭처링 할당은 배열에서 필요한 요소만 추출하여 변수에 할당하고 싶을 때 유용하다. 아래 예제는 Date 객체에서 년도, 월, 일을 추출하는 예제이다.

## JAVASCRIPT

```
const today = new Date();
console.log(today); // Sun Mar 22 2020 22:00:55 GMT+0900 (대한민국 표준시)
```

```
const formattedDate = today.toISOString().substring(0, 10);
console.log(formattedDate); // "2020-03-22"
```

```
// 문자열을 분리하여 배열로 변환한 후, 배열 디스트럭처링 할당을 통해 필요한 요소를 취득한다.
```

\* Date.prototype.toISOString  
ISO 8601 형식으로 날짜와 시간을 표현한 문자열을 반환  
ex. 2020/3/26/10:00 -> 2020-03-26T01:00:00.000Z  
\* substring 메소드  
첫번째 인수로 전달한 인덱스에 위치하는 문자부터 두번째 인수로 전달한 인덱스에 위치하는 문자의 바로 이전 문자까지 문자열의 부분 문자열을 반환

```
const [year, month, day] = formattedDate.split('-');
console.log([year, month, day]); //
```

배열 디스트럭처링 할당을 위한 변수에 Rest 파라미터와 유사하게 Rest 요소(Rest element) ...을 사용할 수 있다. Rest 요소는 Rest 파라미터와 마찬가지로 반드시 마지막에 위치해야 한다.

## JAVASCRIPT

```
// Rest 요소
const [x, ...y] = [1, 2, 3];
console.log(x, y); //
```

Rest 파라미터는 함수에 전달된 인수들의 목록을 배열로 전달받는다 ex.

```
function foo(param, ...rest) {
  console.log(param); // 1
  console.log(rest); // [ 2, 3, 4, 5 ]
}
```

```
foo(1, 2, 3, 4, 5);
```

## # 2. 객체 디스트럭처링 할당

ES5에서 객체의 각 프로퍼티를 객체로부터 디스트럭처링하여 변수에 할당하기 위해서는 프로퍼티 키를 사용해야 한다.

## JAVASCRIPT

```
// ES5
var user = { firstName: 'Ungmo', lastName: 'Lee' };

var firstName = user.firstName;
var lastName = user.lastName;

console.log(firstName, lastName); // Ungmo Lee
```

ES6의 객체 디스트럭처링 할당은 객체의 각 프로퍼티를 객체로부터 추출하여 1개 이상의 변수에 할당한다. 배열 디스트럭처링 할당과 마찬가지로 객체 디스트럭처링 할당을 위해서는 할당 연산자 왼쪽에 값을 할당 받을 변수를 선언해야 한다.

이를 위해 여러 개의 변수를 객체 리터럴 형태로 선언한다. 이때 할당 기준은 프로퍼티 키이다. 즉, 순서는 의미가 없으며 변수 이름과 프로퍼티 키가 일치하면 할당된다.

## JAVASCRIPT

```
const user = { firstName: 'Ungmo', lastName: 'Lee' };

// ES6 객체 디스트럭처링 할당
// 변수 lastName, firstName을 선언하고 객체 user를 디스트럭처링하여 할당한다.
// 이때 프로퍼티 키를 기준으로 디스트럭처링 할당이 이루어진다. 순서는 의미가 없다.
const { lastName, firstName } = user;

console.log(firstName, lastName); // Ungmo Lee
```

위 예제에서 객체 리터럴 형태로 선언한 변수는 lastName, firstName이다. 이는 프로퍼티 축약 표현을 통해 선언한 것이다.

\* 프로퍼티 축약 표현  
ES6에서는 프로퍼티 값으로 변수를 사용하는 경우, 변수 이름과 프로퍼티 키가 동일한 이름일 때, 프로퍼티 키를 생략할 수 있다. 이 때 프로퍼티 키는 변수 이름으로 자동 생성

## JAVASCRIPT

```
const { lastName, firstName } = user;
// 위와 아래는 동치이다.
const { lastName: lastName, firstName: firstName } = user;
```

따라서 객체의 프로퍼티 키와 다른 변수 이름으로 프로퍼티 값을 할당 받으려면 아래와 같이 변수를 선언한다.


## JAVASCRIPT

```
const user = { firstName: 'Ungmo', lastName: 'Lee' };

// ES6 객체 디스트럭처링 할당
// 프로퍼티 키를 기준으로 디스트럭처링 할당이 이루어진다.
// 프로퍼티 키가 lastName인 프로퍼티 값을 ln에 할당한다.
// 프로퍼티 키가 firstName인 프로퍼티 값을 fn에 할당한다.
const { lastName: ln, firstName: fn } = user;

console.log(fn, ln); // Ungmo Lee
```

-> 원래 할당이라는 게 우변의 것을 좌변에 할당하는 건데 이 설명은 지금 좌변의 것을 우변의 것으로 할당하는 뉘앙스



객체 디스트럭처링 할당을 위한 변수에 기본값을 설정할 수 있다.

## JAVASCRIPT

이 노란색, 빨간색 하이라이트 둘 중에 어떤 게 기본값?

```
const { firstName = 'Ungmo', lastName } = { lastName: 'Lee' };
console.log(firstName, lastName); // Ungmo Lee
```

```
const { firstName: fn = 'Ungmo', lastName: ln } = { lastName: 'Lee' };
console.log(fn, ln); // Ungmo Lee
```

객체 디스트럭처링 할당은 프로퍼티 키로 객체에서 필요한 프로퍼티 값만을 추출할 수 있다.

## JAVASCRIPT

```
const todo = { id: 1, content: 'HTML', completed: true };

// todo 객체로부터 id 프로퍼티만을 추출한다.
const { id } = todo;
console.log(id); // 1
```

객체 디스트럭처링 할당은 객체를 인수로 전달받는 함수의 매개변수에도 사용할 수 있다.

## JAVASCRIPT

```
function printTodo(todo) {
  console.log(`할일 ${todo.content}은 ${todo.completed ? '완료' : '비완료'} 상태입니다.`);
}

printTodo({ id: 1, content: 'HTML', completed: true });
// 할일 HTML은 완료 상태입니다.
```

위 예제에서 객체를 인수로 전달받는 매개변수 todo에 객체 디스트럭처링 할당을 사용하면 보다 간단하고 가독성 좋게 표현이 가능하다.

## JAVASCRIPT

```
function printTodo({ content, completed }) {
  console.log(`할일 ${content}은 ${completed ? '완료' : '비완료'} 상태입니다.`);
}
```

```
printTodo({ id: 1, content: 'HTML', completed: true });  
// 할일 HTML은 완료 상태입니다.
```

배열의 요소가 객체인 경우, 배열 디스트럭처링 할당과 객체 디스트럭처링 할당을 혼용할 수 있다.

#### JAVASCRIPT

```
const todos = [  
  { id: 1, content: 'HTML', completed: true },  
  { id: 2, content: 'CSS', completed: false },  
  { id: 3, content: 'JS', completed: false }  
];  
  
// todos 배열의 두번째 요소인 객체로부터 id 프로퍼티만을 추출한다.  
const [, { id }] = todos;  
console.log(id); // 2
```

중첩 객체의 경우는 아래와 같이 사용한다.

#### JAVASCRIPT

```
const user = {  
  name: 'Lee',  
  address: {  
    zipCode: '03068',  
    city: 'Seoul'  
  }  
};  
  
const { address: { city } } = user;  
console.log(city); // 'Seoul'
```

객체 디스트럭처링 할당을 위한 변수에 Rest 파라미터와 유사하게 **Rest 프로퍼티 ...**을 사용할 수 있다. Rest 프로퍼티는 Rest 파라미터와 마찬가지로 반드시 마지막에 위치해야 한다.

## JAVASCRIPT

```
// Rest 프로퍼티  
const { x, ...rest } = { x: 1, y: 2, z: 3 };  
console.log(x, rest); //
```

Rest 프로퍼티는 스프레드 프로퍼티와 함께 2019년 11월 현재 Rest/Spread 프로퍼티는 TC39 프로세스의 stage 4(Finished) 단계에 제안되어 있다.(<https://github.com/tc39/proposal-object-rest-spread>)