디스트럭처링

디스트럭처링(Destructuring)은 구조화된 배열 또는 객체를 Destructuring(비구조화, 파괴)하여 개별적인 변수에 할당하는 것이다. 배열 또는 객체 리터럴에서 필요한 값만을 추출하여 변수에 할당하거나 반환할 때 유용하다.

배열 디스트럭처링 (Array destructuring)

ES5에서 배열의 각 요소를 배열로부터 디스트럭처링하여 변수에 할당하기 위한 방법은 아래와 같다.

```
// 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 디스트럭처링은 배열의 각 요소를 배열로부터 추출하여 변수 리스트에 할당한다. 이때 추출/할당 기준은 **배열의 인덱스**이다.

```
// ES6 Destructuring
const arr = [1, 2, 3];

// 배열의 인덱스를 기준으로 배열로부터 요소를 추출하여 변수에 할당
// 변수 one, two, three가 선언되고 arr(initializer(초기화자))가 Destructuring(비구조화, 파괴)되어 할당된다.
const [one, two, three] = arr;
// 디스트럭처링을 사용할 때는 반드시 initializer(초기화자)를 할당해야 한다.
```

```
// const [one, two, three]; // SyntaxError: Missing initial
izer in destructuring declaration

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

배열 디스트럭처링을 위해서는 할당 연산자 왼쪽에 배열 형태의 변수 리스트가 필요하다.

```
let x, y, z;
[x, y, z] = [1, 2, 3];

// 위의 구문과 동치이다.
let [x, y, z] = [1, 2, 3];
```

왼쪽의 변수 리스트와 오른쪽의 배열은 배열의 인덱스를 기준으로 할당된다.

```
let x, y, z;

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

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

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

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

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

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

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

ES6의 배열 디스트럭처링은 배열에서 필요한 요소만 추출하여 변수에 할당하고 싶은 경우에 유용하다. 아래의 코드는 Date 객체에서 년도, 월, 일을 추출하는 예제이다.

```
const today = new Date(); // Tue May 21 2019 22:19:42 GMT+0900 (한국 표준시)
const formattedDate = today.toISOString().substring(0, 10);
// "2019-05-21"
const [year, month, day] = formattedDate.split('-');
console.log([year, month, day]); // [ '2019', '05', '21' ]
```

객체 디스트럭처링 (Object destructuring)

ES5에서 객체의 각 프로퍼티를 객체로부터 디스트럭처링하여 변수에 할당하기 위해서는 프로퍼티 이름(키)을 사용해야 한다.

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

var firstName = obj.firstName;
var lastName  = obj.lastName;

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

ES6의 객체 디스트럭처링은 객체의 각 프로퍼티를 객체로부터 추출하여 변수 리스트에 할당한다. 이때 할당 기준은 **프로퍼티 이름(키)**이다.

```
// ES6 Destructuring
const obj = { firstName: 'Ungmo', lastName: 'Lee' };

// 프로퍼티 키를 기준으로 디스트럭처링 할당이 이루어진다. 순서는 의미가
```

```
없다.
// 변수 lastName, firstName가 선언되고 obj(initializer(초기화
자))가 Destructuring(비구조화, 파괴)되어 할당된다.
const { lastName, firstName } = obj;

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

객체 디스트럭처링을 위해서는 할당 연산자 왼쪽에 객체 형태의 변수 리스트가 필요하다.

```
// 프로퍼티 키가 prop1인 프로퍼티의 값을 변수 p1에 할당
// 프로퍼티 키가 prop2인 프로퍼티의 값을 변수 p2에 할당
const { prop1: p1, prop2: p2 } = { prop1: 'a', prop2: 'b'
};
console.log(p1, p2); // 'a' 'b'
console.log({ prop1: p1, prop2: p2 }); // { prop1: 'a', pro
p2: 'b' }

// 아래는 위의 축약형이다
const { prop1, prop2 } = { prop1: 'a', prop2: 'b' };
console.log({ prop1, prop2 }); // { prop1: 'a', prop2: 'b'
}

// default value
const { prop1, prop2, prop3 = 'c' } = { prop1: 'a', prop2:
'b' };
console.log({ prop1, prop2, prop3 }); // { prop1: 'a', prop
2: 'b', prop3: 'c' }
```

객체 디스트럭처링은 객체에서 프로퍼티 이름(키)으로 필요한 프로퍼티 값만을 추출할 수 있다. 아래의 코드를 살펴보자.

```
const todos = [
  { id: 1, content: 'HTML', completed: true },
  { id: 2, content: 'CSS', completed: false },
  { id: 3, content: 'JS', completed: false }
];
```

```
// todos 배열의 요소인 객체로부터 completed 프로퍼티만을 추출한다.
const completedTodos = todos.filter(({ completed }) => comp
leted);
console.log(completedTodos); // [ { id: 1, content: 'HTML',
completed: true } ]
```

Array.prototype.filter 메소드의 콜백 함수는 대상 배열(todos)을 순회하며 첫 번째 인자로 대상 배열의 요소를 받는다. 이때 필요한 프로퍼티(completed 프로퍼티)만을 추출할 수 있다.

중첩 객체의 경우는 아래와 같이 사용한다.

```
const person = {
  name: 'Lee',
  address: {
    zipCode: '03068',
    city: 'Seoul'
  }
};

const { address: { city } } = person;
console.log(city); // 'Seoul'
```