* 예습
1. 5/20

35. 스프레드 문법

TABLE OF CONTENTS

ES6에서 새롭게 도입된 스프레드 문법(Spread syntax, 전개 문법) ...은 하나로 뭉쳐 있는 여러 값들의 집합을 펼쳐서(전개, 분산하여, spread) 개별적인 값들의 목록으로 만든다.

스프레드 문법을 사용할 수 있는 대상은 Array, String, Map, Set, DOM 컬렉션(NodeList, HTMLCollection), Arguments와 같이 for...of 문으로 순회할 수 있는 이터러블에 한정된다.

JAVASCRIPT

```
// ...[1, 2, 3]는 [1, 2, 3]을 개별 요소로 분리한다(→ 1, 2, 3)
console.log( ...[1, 2, 3]) // 1 2 3

// 문자열은 이터러블이다.
console.log( ...'Hello'); // H e l l o

// Map과 Set은 이터러블이다.
```

```
console.log( ...new Map([['a', '1'], ['b', '2']])); // [ 'a', '1' ] [ 'b',
 '2' ]
console.log( ...new Set([1, 2, 3])); // 1 2 3

// 이터러블이 아닌 일반 객체는 스프레드 문법의 대상이 될 수 없다.
console.log( ...{ a: 1, b: 2 });
// TypeError: Found non-callable @@iterator
```

위 예제에서 ...[1, 2, 3]은 이터러블인 배열을 펼쳐서 요소값을 개별적인 값들의 목록 1 2 3으로 만든다. 이때 1 2 3은 값이 아니라 값들의 목록이다. 즉, 스프레드 문법의 결과는 값이 아니다. 이는 스프레드 문법 ...이 피연산자를 연산하여 값을 생성하는 연산자가 아님을 의미한다. 따라서 스프레드 문법의 결과는 변수에 할당할 수 없다.

JAVASCRIPT

```
const list = ...arr; // SyntaxError: Unexpected token ...
```

이처럼 스프레드 문법의 결과물은 단독으로 사용할 수 없고, 아래와 같이 쉼표로 구분한 값의 목록을 사용하는 문맥에서만 사용할 수 있다.

- 함수 호출문의 인수 목록
- 배열 리터럴의 요소 목록
- 객체 리터럴의 프로퍼티 목록 (2020년 3월 현재 Stage 4 제안)

#1. 함수 호출문의 인수 목록에서 사용하는 경우

요소값들의 집합인 배열을 펼쳐서 개별적인 값들의 목록으로 만든 후, 이를 함수의 인수 목록으로 전달해야 하는 경우가 있다. 아래의 예제를 살펴보자.

JAVASCRIPT

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

// 배열 arr의 요소 중에서 최대값을 구하기 위해 Math.max를 사용한다.
const maxValue = Math.max(arr);
```

```
console.log(maxValue); // NaN
```

Math.max 메소드는 매개변수 개수를 확정할 수 없는 가변 인자 함수이다. 아래와 같이 개수가 정해져 있지 않은 여러 개의 숫자를 인수로 전달받아 인수 중에서 최대값을 반환한다.

JAVASCRIPT

```
Math.max(1, 2); // → 2
Math.max(1, 2, 3); // → 3
Math.max(1, 2, 3, 4); // → 4
```

만약 Math.max 메소드에 숫자가 아닌 배열을 인수로 전달하면 최대값을 구할 수 없으므로 NaN을 반환한다.

JAVASCRIPT

```
Math.max([1, 2, 3]); // → NaN
```

이와 같은 문제를 해결하기 위해 배열을 펼쳐서 요소값들을 개별적인 값들의 목록으로 만든 후 Math.max 메소드의 인수로 전달해야 한다. 즉, [1, 2, 3]을 1, 2, 3으로 펼쳐서 Math.max 메소드의 인수로 전달해야 한다.

스프레드 문법이 제공되기 이전에는 배열을 펼쳐서 요소값의 목록을 함수의 인수로 전달하고 싶은 경우, Function.prototype.apply를 사용하였다.

JAVASCRIPT

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

// apply 함수의 2번째 인수(배열)는 apply 함수가 호출하는 함수의 인수 목록이다.
// 따라서 배열이 펼쳐져서 인수로 전달되는 효과가 있다.
var maxValue = Math.max.apply(null, arr);

console.log(maxValue); // 3
```

스프레드 문법을 사용하면 보다 간결하고 가독성이 좋다.

JAVASCRIPT

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

// 스프레드 문법을 사용하여 배열 arr을 1, 2, 3으로 펼쳐서 Math.max에 전달한다.
// Math.max( ...[1, 2, 3])는 Math.max(1, 2, 3)과 같다.
const maxValue = Math.max( ...arr);

console.log(maxValue); // 3
```

스프레드 문법은 앞에서 살펴본 Rest 파라미터와 형태가 동일하여 혼동할 수 있으므로 주의가 필요하다.

Rest 파라미터는 함수에 전달된 인수들의 목록을 배열로 전달받기 위해 매개변수 이름 앞에 ...을 붙이는 것이다. 스프레드 문법은 여러 개의 값이 하나로 뭉쳐 있는 배열과 같은 이터러블을 펼쳐서 개별적인 값들의 목록을 만드는 것이다. 따라서 Rest 파라미터와 스프레드 문법은 서로 반대의 개념이다.

JAVASCRIPT

```
// Rest 파라미터는 인수들의 목록을 배열로 전달받는다.
function foo(param, ...rest) {
  console.log(param); // 1
  console.log(rest);  // [ 2, 3 ]
}

// 스프레드 문법은 배열과 같은 이터러블을 펼쳐서 개별적인 값들의 목록을 만든다.
// [1, 2, 3] → 1, 2, 3
foo( ...[1, 2, 3]);
```

2. 배열 리터럴 내부에서 사용하는 경우

스프레드 문법을 배열 리터럴에서 사용하면 보다 간결하고 가독성이 좋게 표현할 수 있다. ES5에서 사용하던 방식과 비교하여 살펴보도록 하자.

2.1. concat

ES5에서 기존의 배열 요소들을 새로운 배열의 일부로 만들고 싶은 경우, 배열 리터럴 만으로 해결할 수 없고 concat 메소드를 사용해야 한다.

JAVASCRIPT

```
// ES5
var arr = [1, 2].concat([3, 4]);
console.log(arr); // [1, 2, 3, 4]
```

스프레드 문법을 사용하면 별도의 메소드를 사용하지 않고 배열 리터럴 만으로 기존의 배열 요소들을 새로운 배열의 일부로 만들 수 있다.

JAVASCRIPT

```
// ES6
const arr =
console.log(arr); // [1, 2, 3, 4]
```

2.2. push

ES5에서 기존의 배열에 다른 배열의 요소들을 push하려면 아래와 같은 방법을 사용한다.

JAVASCRIPT

```
// ES5
var arr1 = [1, 2];
var arr2 = [3, 4];

Array.prototype.push.apply(arr1, arr2);

console.log(arr1); // [1, 2, 3, 4]
```


스프레드 문법을 사용하면 아래와 같이 보다 간편하게 표현할 수 있다.

JAVASCRIPT

```
// ES6
const arr1 = [1, 2];
const arr2 = [3, 4];

// arr1.push(3, 4)와 같다.
arr1.push( ... arr2);

console.log(arr1); // [1, 2, 3, 4]
```

원본 배열을 직접 수정하는 push 메소드를 사용하는 것보다 스프레드 문법을 사용하는 것이 바람직하다.

JAVASCRIPT

```
// ES6
const arr1 = [1, 2];
const arr2 = [3, 4];

console.log(                    ); // [1, 2, 3, 4]
```

2.3. splice

ES5에서 기존의 배열에 다른 배열의 요소들을 삽입하려면 splice 메소드를 사용한다.

JAVASCRIPT

```
// ES5
var arr1 = [1, 4];
var arr2 = [2, 3];

// apply 메소드의 2번째 인수는 배열이다. 이것은 인수 목록으로 splice 메소드에 전달된다.
// [1, 0].concat(arr2) → [1, 0, 2, 3]
// arr1.splice(1, 0, 2, 3) → arr1[1]부터 0개의 요소를 제거하고
// 그자리(arr1[1])에 새로운 요소(2, 3)를 삽입한다.
Array.prototype.splice.apply(arr1, [1, 0].concat(arr2));

console.log(arr1); //
```

스프레드 문법을 사용하면 아래와 같이 보다 간편하게 표현할 수 있다.

JAVASCRIPT

```
// ES6
const arr1 = [1, 4];
const arr2 = [2, 3];

arr1.splice(1, 0, ...arr2);

console.log(arr1); // [1, 2, 3, 4]
```

2.4. 배열 복사

ES5에서 기존의 배열을 복사하기 위해서는 slice 메소드를 사용한다.

JAVASCRIPT

```
// ES5
var origin = [1, 2];
var copy = origin.slice();

console.log(copy); // [1, 2]
console.log(copy === origin); // false
```

스프레드 문법을 사용하면 보다 간편하게 배열을 복사할 수 있다.

JAVASCRIPT

```
// ES6
const origin = [1, 2];
const copy = [...origin];

console.log(copy); // [1, 2]
console.log(copy === origin); // false
```

이때 원본 배열의 각 요소를 얕은 복사(shallow copy)하여 새로운 복사본을 생성한다. 이는 slice 메소드도 마찬가지다.

2.5. 유사 배열 객체를 배열로 변환

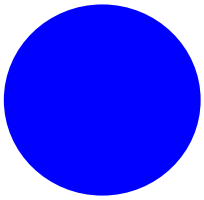
유사 배열 객체(Array-like object)를 배열로 변환하려면 slice 메소드를 apply 함수로 호출하는 것이 일반적이다.

JAVASCRIPT

```
// ES5
function sum() {
  // 유사 배열 객체인 arguments를 배열로 변환
  var args = Array.prototype.slice.apply(arguments);

  return args.reduce(function (pre, cur) {
    return pre + cur;
  }, 0);
}

console.log(sum(1, 2, 3)); // 6
```



스프레드 문법을 사용하면 보다 간편하게 유사 배열 객체를 배열로 변환할 수 있다. 유사 배열 객체인 arguments 객체는 이터러블이다. 따라서 스프레드 문법의 대상이 될 수 있다.

JAVASCRIPT

```
// ES6
function sum() {
  // 유사 배열 객체인 arguments를 배열로 변환
  const args = [...arguments];

  return args.reduce((pre, cur) => pre + cur, 0);
}

console.log(sum(1, 2, 3)); // 6
```

3. 객체 리터럴 내부에서 사용하는 경우

객체 리터럴의 프로퍼티 목록에서 스프레드 문법을 사용할 수 있는 스프레드 프로퍼티는 Rest 프로퍼티와 함께 2020년 3월 현재 TC39 프로세스의 stage 4(Finished) 단계에 제안되어 있다. (https://github.com/tc39/proposal-object-rest-spread)

스프레드 문법의 대상은 이터러블이어야 하지만 스프레드 프로퍼티 제안은 일반 객체를 대상으로도 스프레드 문법의 사용을 허용한다.

JAVASCRIPT

```
// 스프레드 프로퍼티
// 객체 복사(얕은 복사)
const obj = { x: 1, y: 2 };
const copy = { ...obj };
console.log(copy); // { x: 1, y: 2 }
console.log(obj === copy); // false

// 객체 병합
const merged = { x: 1, y: 2, ...{ a: 3, b: 4 } };
console.log(merged); // { x: 1, y: 2, a: 3, b: 4 }
```

스프레드 프로퍼티가 제안되기 이전에는 ES6에서 도입된 Object.assign 메소드를 사용하여 여러 개의 객체를 병합하거나 특정 프로퍼티를 변경 또는 추가하였다.

JAVASCRIPT

```
// 객체의 병합
// 프로퍼티가 중복되는 경우, 뒤에 위치한 프로퍼티가 우선권을 갖는다.
const merged = Object.assign({}, { x: 1, y: 2 }, { y: 10, z: 3 });
console.log(merged); //

// 특정 프로퍼티 변경
const changed = Object.assign({}, { x: 1, y: 2 }, { y: 100 });
console.log(changed); //

// 프로퍼티 추가
```

```
const added = Object.assign({}, { x: 1, y: 2 }, { z: 0 });
console.log(added); //
```

스프레드 프로퍼티는 Object.assign 메소드를 대체할 수 있는 간편한 문법이다.

JAVASCRIPT

```
// 객체의 병합
// 프로퍼티가 중복되는 경우, 프로퍼티가 우선권을 갖는다.
const merged = { ...{ x: 1, y: 2 }, ...{ y: 10, z: 3 } };
console.log(merged); //

// 특정 프로퍼티 변경
const changed = { ...{ x: 1, y: 2 }, y: 100 };
// changed = { ...{ x: 1, y: 2 }, ...{ y: 100 } }
console.log(changed); //

// 프로퍼티 추가
const added = { ...{ x: 1, y: 2 }, z: 0 };
// added = { ...{ x: 1, y: 2 }, ...{ z: 0 } }
console.log(added); //
```