스프레드 문법 (8월 6일)

배정	
상태	8월
속성	@2021년 8월 6일 오후 8:19
언어	

스프레드 문법(spread syntax)이란?

```
... <- 요걸 사용한다
```

> 하나로 뭉쳐 있는 여러 값들의 집합을 펼쳐서(spread) 개별적인 값들의 목록을 만드는 문법.

사용대상

> Array, String, Map, Set, DOM, arguments 등 for..of 문으로 순회할 수 있는 이터러블

```
// ...[1, 2, 3]은 [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
```

▼ 혼동하지 마세요 ! Rest parameter 와 Spread syntax

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

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

Rest 파라미터 : 배열로 묶는다

스프레드 문법 : 배열을 푼다

함수호출문의 인수목록에서 사용하는 경우

예전에는 배열을 펼처서 요소들의 목록을 함수인수로 전달하고 싶은 경우, function.prototype.apply을 사용 했다고 한다. 하지만 지금은 spread syntax를 사용한다.

```
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 max = Math.max.apply(null, arr); // -> 3

-----------------------------------------------------------------------------------

const arr = [1, 2, 3];

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

배열 리터럴 내부에서 사용하는 경우

▼ 2개의 배열을 1개의 배열로 결합하고 싶은 경우

concat → 스프레드 문법

▼ 중간에 다른 배열의 요소를 추가하거나 제거하고 싶은 경우

splice를 사용하는데

splice + apply메서드 → splice 파라미터 안에 스프레드 문법사용

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

/*
apply 메서드의 2번째 인수(배열)는 apply 메서드가 호출한 splice 메서드의 인수 목록이다.
apply 메서드의 2번째 인수 [1, 0].concat(arr2)는 [1, 0, 2, 3]으로 평가된다.
따라서 splice 메서드에 apply 메서드의 2번째 인수 [1, 0, 2, 3]이 해체되어 전달된다.
즉, arr1[1]부터 0개의 요소를 제거하고 그 자리(arr1[1])에 새로운 요소(2, 3)를 삽입한다.
*/
Array.prototype.splice.apply(arr1, [1, 0].concat(arr2));
console.log(arr1); // [1, 2, 3, 4]

-------------------------------------------------------------------------

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

arr1.splice(1, 0, ...arr2);
console.log(arr1); // [1, 2, 3, 4]
```

▼ 배열 복사(slice)

slice → 스프레드 문법

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

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

-------------------------------------------------------------------------

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

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

▼ 이터러블을 배열로 변환(apply 또는 call)

apply 혹은 call 메서드

```
// ES5
function sum() {
  // 이터러블이면서 유사 배열 객체인 arguments를 배열로 변환
  // apply도 같은 결과를 보여줌. call, apply 안쓸 경우 reduce를 사용 할 수 없음.
  var args = Array.prototype.slice.call(arguments);
```

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

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

-------------------------------------------------------------------------------

function sum() {
  // 이터러블이면서 유사 배열 객체인 arguments를 배열로 변환
  return [...arguments].reduce((pre, cur) => pre + cur, 0);
}

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

▼ 이터러블이면서 유사배열 객체인것

```
// ES6
function sum() {
  // 이터러블이면서 유사 배열 객체인 arguments를 배열로 변환
  return [...arguments].reduce((pre, cur) => pre + cur, 0);
}

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

-------------------------------------------------------------------------------

// 이터러블이 아닌 유사 배열 객체
const arrayLike = {
  0: 1,
  1: 2,
  2: 3,
  length: 3
};

const arr = [...arrayLike];
// TypeError: object is not iterable (cannot read property Symbol(Symbol.iterator))

-------------------------------------------------------------------------

// 이터러블이 아닌 유사 배열 객체를 배열로 바꿈.
const arrayLike = {
  0: 1,
  1: 2,
  2: 3,
  length: 3
};

const arr = Array.from(arrayLike); // -> [1, 2, 3]
arr.reduce((pre, cur) => pre + cur, 0); // 6
```

💡 `Array.from` 메서드는 유사배열객체를 배열로 바꿔준다.

객체 리터럴 내부에서 사용하는 경우

여러 개의 객체 병합 혹은 특정 프로퍼티를 변경 또는 추가시

Object.assign → 스프레드 문법

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

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

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

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

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

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