

자료구조와 자료형

맵

- 지금까지 살펴본 자료구조의 특징
 - 객체 - 키가 있는 컬렉션을 저장함
 - 배열 - 순서가 있는 컬렉션을 저장함
- 키가 있는 데이터를 저장한다는 점에서 객체와 유사하지만, **맵은 키로 다양한 자료형의 값을 쓸 수 있도록 허용**
 - 객체의 경우 키로 문자열, 심볼 자료형 값만 사용 가능
- 맵에는 다음과 같은 주요 메서드와 속성이 존재
 - 맵 객체에는 객체처럼 대괄호 연산자를 사용하지 않고 **set, get 메소드를 사용하여 키, 값 설정 및 접근**함을 유의

```
new Map() - 맵을 만듭니다.  
map.set(key, value) - key를 이용해 value를 저장합니다.  
map.get(key) - key에 해당하는 값을 반환합니다. key가 존재하지 않으면 undefined를 반환합니다.  
map.has(key) - key가 존재하면 true, 존재하지 않으면 false를 반환합니다.  
map.delete(key) - key에 해당하는 값을 삭제합니다.  
map.clear() - 맵 안의 모든 요소를 제거합니다.  
map.size - 요소의 개수를 반환합니다.
```

맵 객체 사용 코드

```
let map = new Map();  
  
// 다양한 타입의 값을 키로 사용 가능  
map.set('1', 'str1'); // 문자형 키  
map.set(1, 'num1');   // 숫자형 키  
map.set(true, 'bool1'); // 불린형 키  
  
// 객체는 키를 문자형으로 변환하지만 맵은 키의 타입을 변환시키지 않고 그대로 유지  
// 따라서 아래의 코드는 출력되는 값이 다름  
// 숫자형 키 사용하여 접근  
alert( map.get(1) ); // 'num1'  
// 문자형 키 사용하여 접근  
alert( map.get('1') ); // 'str1'  
  
alert( map.size ); // 3
```

- 맵 객체의 키로 객체도 사용 가능

```
let john = { name: "John" };

// 고객의 가게 방문 횟수를 세본다고 가정
let visitsCountMap = new Map();

// john 객체를 맵의 키로 사용
visitsCountMap.set(john, 123);

alert( visitsCountMap.get(john) ); // 123
```

- map.set을 호출할 때마다 맵 자신(this)이 반환되므로 이를 이용하면 map.set 메소드를 체이닝(chaining)할 수 있음
 - [메소드 체이닝 패턴](#)

```
map.set('1', 'str1').set(1, 'num1').set(true, 'bool1');
```

맵의 요소에 반복 작업하기

- 세 가지 메서드(keys, values, entries)를 사용해 맵의 각 요소에 반복 작업을 할 수 있음

map.keys() - 각 요소의 "키"를 모은 반복 가능한(iterable, 이터러블) 객체를 반환
 map.values() - 각 요소의 "값"을 모은 이터러블 객체를 반환
 map.entries() - 요소의 [키, 값]을 한 쌍으로 하는 이터러블 객체를 반환합니다. 이 이터러블 객체는 for ... of반복문의 기초로 사용됨

```
let recipeMap = new Map([
  ['cucumber', 500], ['tomatoes', 350], ['onion', 50]
]);

// 키(vegetable)를 대상으로 순회
for (let vegetable of recipeMap.keys()) {
  alert(vegetable); // cucumber, tomatoes, onion (삽입 순서 기억)
}

// 값(amount)을 대상으로 순회
for (let amount of recipeMap.values()) {
  alert(amount); // 500, 350, 50
}

// [키, 값] 쌍을 대상으로 순회
// recipeMap.entries()와 동일
for (let entry of recipeMap) {
  // 여기서 entry는 배열 (배열 첫 번째 요소 => 키, 두 번째 요소 => 값)
  alert(entry); // cucumber,500 ...
}

// 각 (키, 값) 쌍을 대상으로 함수를 실행
recipeMap.forEach( (value, key, map) => {
  alert(`${key}: ${value}`); // cucumber: 500 ...
});
```

- 삽입된 속성의 순서를 기억 못하는 객체와는 달리 맵은 삽입 순서를 기억함

- 배열과 유사하게 내장 메서드 `forEach`도 지원

구조 분해 할당 (*)

- 자바스크립트 언어에서는 객체나 배열을 변수로 분해할 수 있게 해주는 특별한 문법인 구조 분해 할당(`destructuring assignment`)을 사용 가능

배열 분해하기

```
// 이름과 성을 요소로 가진 배열
let arr = ["Bora", "Lee"];

// 구조 분해 할당을 이용해 firstName엔 arr[0]을 surname엔 arr[1]을 할당하였습니다.
let [firstName, surname] = arr;
// 상수로 대입 받기도 가능
const [f, s] = arr;

// 위 코드와 똑같은 역할을 하는 코드
/*
let firstName = arr[0];
let surname = arr[1];
*/

alert(firstName); // Bora
alert(surname); // Lee
```

응용 코드

```
// split 메서드 호출 결과로 배열 반환, 이후 배열된 반환을 바로 변수로 대입
let [firstName, surname] = "Bora Lee".split(' ');
```

- 구조 분해 할당 적용할 때 대입문의 왼쪽에 모든 종류의 대입이 가능한 무언가(변수, 상수, 속성 등)가 올 수 있음

```
let user = {};
[user.name, user.surname] = "Bora Lee".split(' ');
alert(user.name); // Bora
```

- 침표를 사용하여 특정 요소를 무시할 수 있음

```
// 두 번째, 네 번째 요소는 필요하지 않음
// 두 번째 요소는 침표만 있고 대입될 대상이 없으므로 무시
// 네 번째 요소는 구조 분해 할당 대상으로 아예 포함되지 않음
let [firstName, , title] = ["Julius", "Caesar", "Consul", "of the Roman Republic"];

alert( firstName ); // Julius
alert( title ); // Consul
```

Spread syntax(...)로 나머지 요소 가져오기

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax
- 배열 앞쪽에 위치한 값 몇 개만 필요하고 그 이후 이어지는 나머지 값들은 한데 모아서 저장하고 싶을 때 점 세개(...) 연산자를 사용

```
let [name1, name2, ...rest] = ["Julius", "Caesar", "Consul", "of the Roman Republic"];

alert(name1); // Julius
alert(name2); // Caesar

// rest는 배열
alert(rest[0]); // Consul
alert(rest[1]); // of the Roman Republic
alert(rest.length); // 2
```

기본값

- 할당하고자 하는 변수의 개수가 분해하고자 하는 배열의 길이보다 크더라도 에러가 발생하지 않음

```
// 대입할 값이 없으므로 모두 undefined 값이 대입됨
let [firstName, surname] = [];

alert(firstName); // undefined
alert(surname); // undefined
```

- 대입 기호(=)을 이용하면 할당할 값이 없을 때 기본으로 할당해 줄 값인 기본값(default value)을 설정 가능

```
// surname은 기본값을 대입
let [name = "Guest", surname = "Anonymous"] = ["Julius"];

alert(name); // Julius (배열에서 받아온 값)
alert(surname); // Anonymous (기본값)
```

객체 분해하기

- 구조 분해 할당 문법을 이용하여 객체도 분해 가능

```
let {var1, var2} = {var1:..., var2:...}
```

- 할당 연산자 우측엔 분해하고자 하는 객체를, 좌측엔 상응하는 객체 프로퍼티의 "패턴"을 입력

```
let options = { title: "Menu", width: 100, height: 200 };

let { title, width, height } = options;

alert(title); // Menu
alert(width); // 100
alert(height); // 200
```

- let 괄호 안의 변수 순서가 바뀌어도 동일하게 동작함

```
let { height, width, title } = { title: "Menu", height: 200, width: 100 }
```

- 좌측 패턴에 콜론(:)을 사용하여, 객체 프로퍼티 값을 프로퍼티 키와 다른 이름을 가진 변수에 저장 가능

```
let options = { title: "Menu", width: 100, height: 200 };

// { 객체 프로퍼티 : 목표 변수 }
// width 값은 w 변수에, height 값은 h 변수에 저장하고, title 키의 속성값은 그대로 title 변수에 저장
let { width: w, height: h, title } = options;

alert(title); // Menu
alert(w); // 100
alert(h); // 200
```

- 프로퍼티가 없는 경우를 대비하여 대입 기호(=)을 사용해 기본값을 설정하는 것도 가능

```
let options = { title: "Menu" };
// width, height 값의 기본값을 지정
let { width = 100, height = 200, title } = options;

alert(title); // Menu
alert(width); // 100
alert(height); // 200
```

- 콜론과 할당 연산자를 동시에 사용 가능

```
let options = { title: "Menu" };

let { width: w = 100, height: h = 200, title } = options;

alert(title); // Menu
alert(w); // 100
alert(h); // 200
```

- 전개 구문을 사용하면 배열에서 했던 것처럼 나머지 프로퍼티를 객체에 할당할 수 있음

```
let options = { title: "Menu", width: 100, height: 200 };

// title = 이름이 title인 프로퍼티, rest = 나머지 프로퍼티들
// rest는 나머지 프로퍼티들이 포함될 객체
let { title, ...rest } = options;

// title엔 "Menu", rest엔 { height: 200, width: 100 } 내용을 가진 객체가 할당됨
alert(rest.height); // 200
alert(rest.width); // 100
```


중첩 구조 분해 (@)

- 객체나 배열이 다른 객체나 배열을 포함하는 경우, 좀 더 "복잡한 패턴"을 사용하면 중첩 배열이나 객체의 정보를 추출 가능 => 중첩 구조 분해(nested destructuring)

```
let options = {
  size: { width: 100, height: 200 },
  items: ["Cake", "Donut"],
  extra: true
};

// 실제로 값을 대입받을 변수는 width, height, item1, item2, title
// (size, items는 중첩된 객체나 배열임을 알리기 위해서 사용됨)
let {
  // size는 여기
  size: {
    width,
    height
  },
  // items는 여기에 할당
  items: [ item1, item2 ],
  // 분해하려는 객체에 title 프로퍼티가 없으므로 기본값을 사용함
  title = "Menu"
} = options;

alert(title); // Menu
alert(width); // 100
alert(height); // 200
alert(item1); // Cake
alert(item2); // Donut
```

<pre>let { size: { width, height }, items: [item1, item2], title = "Menu" }</pre>		<pre>let options = { size: { width: 100, height: 200 }, items: ["Cake", "Donut"], extra: true }</pre>
---	---	---

- 함수를 호출하며 객체를 전달하고 함수의 **파라미터 목록(소괄호)**에서 **구조 분해**를 수행 가능

```
// 함수에 전달할 객체
let options = {
  title: "My menu",
  items: ["Item1", "Item2"]
};

// 전달받은 객체를 분해해 함수 지역 변수에 즉시 할당함
function showMenu({ title = "Untitled", width = 200, height = 100, items = [] }) {
  // title, items - 객체 options에서 가져옴
  // width, height - 기본값
  alert( `${title} ${width} ${height}` ); // My Menu 200 100
  alert( items ); // Item1, Item2
}

showMenu(options);
```

- 함수 파라미터 목록 내부에서 중첩 객체와 콜론을 조합하여 좀 더 복잡한 구조 분해 가능

```
function showMenu({
  title = "Untitled",
  width: w = 100, // width는 w에,
  height: h = 200, // height는 h에,
  items: [item1, item2] // items의 첫 번째 요소는 item1에, 두 번째 요소는 item2에
  할당함
}) {
  alert( `${title} ${w} ${h}` ); // My Menu 100 200
  alert( item1 ); // Item1
  alert( item2 ); // Item2
}
```

- 모든 인수에 기본값을 할당해 주려면 빈 객체를 명시적으로 전달
 - 혹은 빈 객체를 인수 전체의 기본값으로 설정해도 됨

```
// 구조 분해 패턴 오른쪽에 대입 연산자와 기본값으로 사용할 객체를 제공
// (인수 객체의 기본값을 빈 객체 {}로 설정하면 어떤 경우든 분해할 것이 생겨서 함수에 인수를
  하나도 전달하지 않아도 에러가 발생하지 않게 됨)
function showMenu({ title = "Menu", width = 100, height = 200 } = {}) {
  alert( `${title} ${width} ${height}` );
}

// 아무런 값을 전달하지 않아 기본값(빈 객체)이 대입되는 과정에서, 구조 분해 할당을 통해 기본
  대입 값을 전달받게 됨
showMenu(); // Menu 100 200
```

리퍼런스

- 위크맵과 위크셋

- <https://betterprogramming.pub/map-weakmap-set-weakset-in-javascript-77ecb5161e3>