


아이템 27

함수형 기법과 라이브러리로 타입 흐름 유지하기

채희수 / 23.06.03

자바스크립트의 다양한 라이브러리

- 제이쿼리(jQuery) jquery 
- 언더스코어 underscore 
- 로대시(Lodash) @types/lodash 
- 램다(Ramda) @types/ramda 

=> 자바스크립트에서 유용하게 사용하는 라이브러리들은
타입스크립트와 조합하여 사용하면 더 좋다.

왜? 타입 정보가 그대로 유지되면서 타입 흐름이 계속 전달되도록 하기 때문에
데이터를 가공할 때 편리하다.

만약 직접 루프를 구현하면 타입 체크에 대한 관리도 직접해야 한다.

좋은 메서드나 라이브러리도 알아야 쓸 수 있으니깐

책에 나온 메서드 설명할게요

Array.prototype.flat 메서드

배열을 지정된 깊이(depth)만큼 평탄화(flat)하여 새로운 배열을 생성

- 평탄화 : 중첩된 배열을 단일 차원의 배열로 변환

```
arr.flat([depth]);
```

- depth : 기본값 1 / 양의 정수 / 0이하의 값이 주어지면 원래 배열 그대로 반환

```
const deeplyNestedArray = [1, [2, [3, [4, [5]]]]];
```

```
const flattenedArray = deeplyNestedArray.flat(3);  
console.log(flattenedArray); // [1, 2, 3, 4, 5]
```

* 사용하려고하면 flat 메서드에 대한 타입이 없다고 오류메세지가 발생하는데, tsconfig.json 파일에서 “lib”: [“es2019”] 추가하면 된다.
출처: <https://bobbyhadz.com/blog/typescript-property-flatmap-does-not-exist-on-type>

```
Property 'flat' does not exist on type '(number | (number | (number | (number | number[])[])[])[[]]'. Do you need to change your target library? Try changing the 'lib' compiler option to 'es2019' or later. (2550)
```

TS Config * typescript playground 에서도 설정 가능하다.

Lang

Which language should be used in the editor

Target:

Set the JavaScript language version for emitted JavaScript and include compatible library declarations.

JSX:

Specify what JSX code is generated.

Module:

Specify what module code is generated.

concat + 루프 사용으로 배열을 붙이기 보다는 “ flat으로 한번에 ”

```
declare const rosters: {[team: string]: BasketballPlayer[]};  
let allPlayers: BasketballPlayer[] = [];  
for (const players of Object.values(rosters)) {  
  allPlayers = allPlayers.concat(players); // OK  
}
```



```
declare const rosters: {[team: string]: BasketballPlayer[]};  
const allPlayers = Object.values(rosters).flat();  
// OK, type is BasketballPlayer[]
```

로대시의 zipObject

주어진 키(key) 배열과 값(value) 배열을 기반으로 객체를 만들어준다.

```
_.zipObject([keys], [values])
```

```
import * as _ from "lodash"

const keys = ['name', 'age', 'city'];
const values = ['hesu', 42, 'GM'];

const obj = _.zipObject(keys, values);

console.log(obj);
// 출력: { name: 'hesu', age: 42, city: 'GM' }
```

key-value 형태의 객체 생성은 “ zipObject로 코드를 더 짧게 ”

```
const rows = rawRows.slice(1).map(rowStr => {  
  const row = {};  
  rowStr.split(',').forEach((val, j) => {  
    row[headers[j]] = val;  
  });  
  return row;  
});
```



```
import _ from 'lodash';  
const rows = rawRows.slice(1)  
  .map(rowStr => _.zipObject(headers, rowStr.split(',')));  
// Type is _.Dictionary<string>[]
```

로대시와 언더스코어의 ‘체인’으로 타입 흐름 지속 & 코드의 가독성을 높인다.

```
declare const rosters: {[team: string]: BasketballPlayer[]};
const allPlayers = Object.values(rosters).flat();
// OK, type is BasketballPlayer[]
const teamToPlayers: {[team: string]: BasketballPlayer[]} = {};
for (const player of allPlayers) {
  const {team} = player;
  teamToPlayers[team] = teamToPlayers[team] || [];
  teamToPlayers[team].push(player);
}

for (const players of Object.values(teamToPlayers)) {
  players.sort((a, b) => b.salary - a.salary);
}

const bestPaid = Object.values(teamToPlayers).map(players => players[0]);
bestPaid.sort((playerA, playerB) => playerB.salary - playerA.salary);
console.log(bestPaid);
```



```
declare const rosters: {[team: string]: BasketballPlayer[]};
const allPlayers = Object.values(rosters).flat();
// OK, type is BasketballPlayer[]
const bestPaid = _(allPlayers)
  .groupBy(player => player.team)
  .mapValues(players => _.maxBy(players, p => p.salary)!)
  .values()
  .sortBy(p => -p.salary)
  .value() // Type is BasketballPlayer[]
```


요약

- 타입 흐름을 개선하고,
가독성을 높이고,
명시적인 타입 구문의 필요성을 줄이기 위해
- > 직접 구현보다는 내장된 함수형 기법과 로대시 같은 유틸리티 라이브러리를 사용하자.

주의. 내장 메서드나 유틸 라이브러리를 모르면 오히려 코드를 이해하거나 작성하기 어렵다.
유틸리티 라이브러리 도입은 같이 코드를 작성하는 팀원 간의 합의도 필요해 보인다.