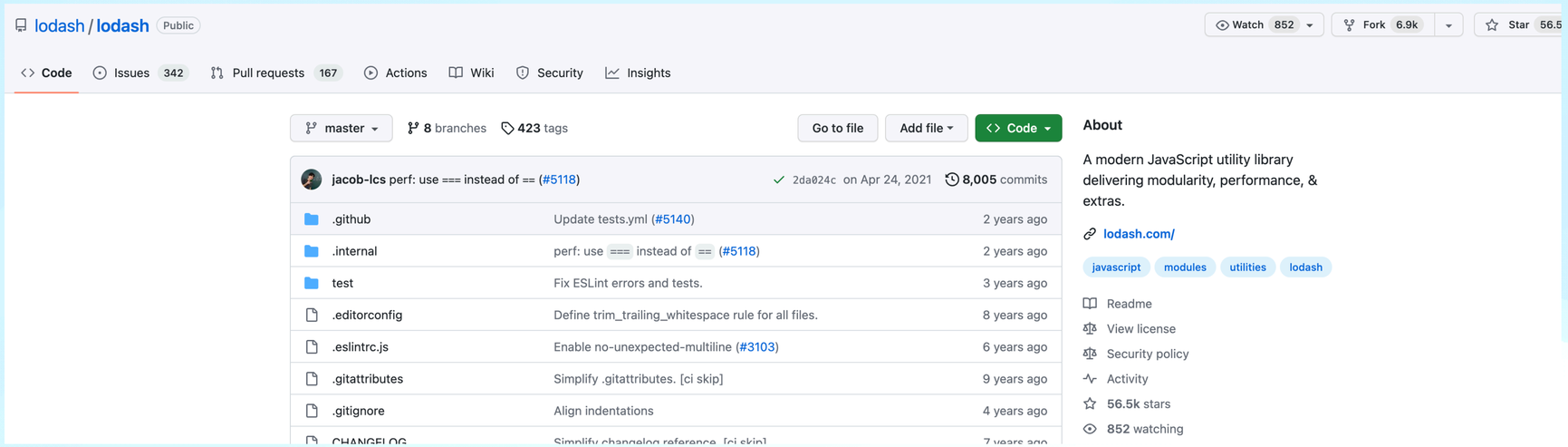
Item 27 함수형 기법과 라이브러리로 타입 흐름 유지하기

ryan-dia(강철원)

Lodash



함수형 프로그래밍과 데이터 조작을 위한 다양한 기능을 제공하는 라이브러리

절차형 프로그래밍 형태로 구현

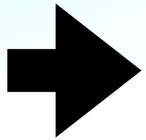
```js
const csvData = require('fs').readFileSync('testCsv.csv').toString();
const rawRows = csvData.split('\n');
const headers = rawRows[0].split(',');

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

console.log(rows);
```

testCsv.csv

```
이름,생년,월,일,국어 점수,영어 점수,수학 점수
홍길동,1992,7,17,100,90,70
희동이,1992,4,3,90,100,100
```



```
[
  {
    '이름': '홍길동',
    '생년': '1992',
    '월': '7',
    '일': '17',
    '국어 점수': '100',
    '영어 점수': '90',
    '수학 점수': '70'
  },
  {
    '이름': '희동이',
    '생년': '1992',
    '월': '4',
    '일': '3',
    '국어 점수': '90',
    '영어 점수': '100',
    '수학 점수': '100'
  }
]
```

📌 reduce를 사용

```
const csvData = require('fs').readFileSync('testCsv.csv').toString();
const rawRows = csvData.split('\n');
const headers = rawRows[0].split(',');

const rows = rawRows
  .slice(1)
  .map((rowStr) =>
    rowStr.split(',').reduce((row, val, i) => ((row[headers[i]] = val), row), {})
  );

console.log(rows);
```

📌 lodash 라이브러리의 zipObject 함수 사용

```
import _ from 'lodash';
import fs from 'fs';
const csvData = fs.readFileSync('testCsv.csv').toString();
const rawRows = csvData.split('\n');
const headers = rawRows[0].split(',');

const rows = rawRows.slice(1).map((rowStr) => _.zipObject(headers, rowStr.split(',')));
console.log(rows);
```

📌 타입스크립트로 변경했을 때

```
import fs from 'fs';
const csvData = fs.readFileSync('testCsv.csv').toString();
const rawRows = csvData.split('\n');
const headers = rawRows[0].split(',');

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

절차형 프로그래밍

```
const row: Record<string, string> = {};
```

```
import fs from 'fs';
const csvData = fs.readFileSync('testCsv.csv').toString();
const rawRows = csvData.split('\n');
const headers = rawRows[0].split(',');

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

reduce 사용

```
import _ from 'lodash';
import fs from 'fs';
const csvData = fs.readFileSync('testCsv.csv').toString();
const rawRows = csvData.split('\n');
const headers = rawRows[0].split(',');

const rows = rawRows.slice(1).map((rowStr) => _.zipObject(headers, rowStr.split(',')));
console.log(rows);
```

라이브러리 사용

📌 이처럼 같은 코드를 타입스크립트로 작성하면 서드파티 라이브러리를 사용하는 것이 무조건 유리하다.
타입 정보를 참고하며 작업할 수 있기 때문에 서드파티 라이브러리 기반으로 바꾸는데 시간이 훨씬 단축된다.

```ts
interface BasketballPlayer {
  name: string;
  team: string;
  salary: number;
}
declare const rosters: { [team: string]: BasketballPlayer[] };

let allPlayers = [];

for (const players of Object.values(rosters)) {
  allPlayers = allPlayers.concat(players);
}
```

```ts
let allPlayers: BasketballPlayer[] = [];

for (const players of Object.values(rosters)) {
  allPlayers = allPlayers.concat(players);
}
```

allPlayers에 타입 구문을 추가

```ts
interface BasketballPlayer {
  name: string;
  team: string;
  salary: number;
}
declare const rosters: { [team: string]: BasketballPlayer[] };

const allPlayers = Object.values(rosters).flat();
```

다차원 배열을 평탄화 해주는 Array.prototype.flat 사용

```ts
interface BasketballPlayer {
  name: string;
  team: string;
  salary: number;
}
declare const rosters: { [team: string]: BasketballPlayer[] };
const allPlayers = Object.values(rosters).flat();

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);
```

1번 코드

```ts
import _ from 'lodash';
interface BasketballPlayer {
  name: string;
  team: string;
  salary: number;
}
declare const rosters: { [team: string]: BasketballPlayer[] };
const allPlayers = Object.values(rosters).flat();

const bestPaid = _(allPlayers)
  .groupBy((player) => player.team)
  .mapValues((players) => _.maxBy(players, (p) => p.salary)!)
  .values()
  .sortBy((p) => -p.salary)
  .value(); //
console.log(bestPaid);
```

2번 코드

bestPaid

```
[
  { team: 'GSW', salary: 37457154, name: 'Stephen Curry' },
  { team: 'HOU', salary: 35654150, name: 'Chris Paul' },
  { team: 'LAL', salary: 35654150, name: 'LeBron James' },
  { team: 'OKC', salary: 35654150, name: 'Russell Westbrook' },
  { team: 'DET', salary: 32088932, name: 'Blake Griffin' },
  ...
]
```

📌 정리

타입 흐름을 개선하고 가독성을 높이고,
명시적인 타입 구문의 필요성을 줄이기 위해 직접 구현하기 보다는
내장된 함수형 기법과 로대시 같은 유틸리티 라이브러리를 사용하는 것이 좋다.