item 58

모던 자바스크립트로 작성하기

채희수 / 23.08.13

타입스크립트는 자바스크립트의 상위집합이기 때문에,
코드를 최신 버전으로 바꾸다 보면
타입스크립트의 일부를 저절로 익힐 수 있다.

옛날 버전의 자바스크립트 코드를 최신 버전의 자바스크립트로 바꾸는 작업은 타입스크립트로 변환하는 작업의 일부로 볼 수 있다.

ECMAScript 모듈 사용하기

- ES2015이전에는 코드를 개별 모듈로 분할하는 표준 방법이 없었지만, 지금은 많다.

: <script> 태그, 직접 갖다 붙이기, Makefile 기법, Nodejs 스타일의 require 구문, AMD 스타일의 define 콜백까지 매우 다양하다.

- 그리고 ES2015부터는 임포트와 익스포트를 사용하는 ECMAScript 모듈이 표준이 되었다.

```
// ESCMAScript
// a.ts
import * as b from './b';
console.log(b.name);
// b.ts
export const name = 'Module B';
```

프로토타입 대신 클래스 사용하기

- ES2015에 class 키워드를 사용하는 클래스 기반 모델이 도입

- 프로토타입을 사용하는 코드를 클래스로 바꾸자.

```
function Person(first, last) {
  this.first = first;
  this.last = last;
}

Person.prototype.getName = function() {
  return this.first + ' ' + this.last;
}

const marie = new Person('Marie', 'Curie');
const personName = marie.getName();
console.log(personName);
```

```
Marie Curie
```

```
// tsConfig: {"noImplicitAny":false}

class Person {
  first: string;
  last: string;

  constructor(first: string, last: string) {
    this.first = first;
    this.last = last;
  }

  getName() {
    return this.first + ' ' + this.last;
  }
}

const marie = new Person('Marie', 'Curie');
const personName = marie.getName();
```

var 대신 let/const 사용하기

- var 대신 let/const 사용하면 스코프 문제를 피할 수 있다.
- let/const 는 제대로 된 블록 스코프를 가진다.

```
function foo() {
  bar();
  function bar() {
    console.log('hello');
  }
}
```

```
function foo() {
  const bar = () => {
    console.log('hello');
  }
  bar();
}
```

```
function foo() {
  bar();
}

const bar = () => {
    console.log('hello');
}
```

for(;;) 대신 for-of 또는 배열 매서드 사용하기

- for-of 루프

```
declare let array: number[];
for (const el of array) {
  // ...
}
```

- forEach 메서드 - 인덱스 변수가 필요한 경우

```
declare let array: number[];
array.forEach((el, i) => {
  // ...
});
```

함수 표현식보다 화살표 함수 사용하기

- this 키워드는 일반적인 변수들과 다른 스코프 규칙을 갖는다.
- 클래스 안에서 this를 사용하면 클래스 인스턴스를 참조하는 것으로 기대하지만, 결과가 다르기도 하다.
- 화살표 함수를 사용하면 상위 스코프의 this를 유지할 수 있다.

```
1    // tsConfig: {"noImplicitThis":false}
2
3    class Foo {
4      method() {
5        console.log(this);
6        [1, 2].forEach(function(i) {
7          console.log(this);
8        });
9      }
10   }
11   const f = new Foo();
12   f.method();
13   // Prints Foo, undefined, undefined in strict mode
14   // Prints Foo, window, window (!) in non-strict mode
```

```
1    // tsConfig: {"noImplicitThis":false}
2
3    class Foo {
4      method() {
5        console.log(this);
6        [1, 2].forEach(i => {
7          console.log(this);
8        });
9      }
10   }
11   const f = new Foo();
12   f.method();
13   // Always prints Foo, Foo, Foo
```

단축 객체 표현과 구조 분해 할당 사용하기

```
// tsConfig: {"noImplicitThis":false}

const x = 1, y = 2, z = 3;
const pt = {
  x: x,
  y: y,
  z: z
};
```

```
const pt = { x, y, z };
```

// 변수와 객체 속성의 이름이 같다면

```
// tsConfig: {"noImplicitThis":false}

['A', 'B', 'C'].map((char, idx) => ({char, idx}));
// [ { char: 'A', idx: 0 },  { char: 'B', idx: 1 }, { char: 'C', idx: 2 } ]
```

// 화살표 함수 내에서 객체를 반환해야한다면, 소괄호로 감싸서 한다.

```
// tsConfig: {"noImplicitThis":false,"noImplicitAny":false}

const obj = {
  onClickLong: function(e) {
    // ...
  },
  onClickCompact(e) {
    // ...
  }
};
```

// 객체 속성 중 함수를 축약해서 표현하는 법

단축 객체 표현과 구조 분해 할당 사용하기

```
declare let obj: {props: {a: string; b: number; }; };
const props = obj.props;
const a = props.a;
const b = props.b;
```

```
const {props} = obj;
const {a, b} = props;
```

```
const {props: {a, b}} = obj;
```

// 이렇게 축약하면 a와 b는 변수로 선언되었지만, props는 변수 선언된 것이 아니다.

```
const {a = 'default'} = obj.props;
```

// a 에 기본값 지정이 가능하다.

// 배열에서도 특히 유용하다.

```
const point = [1, 2, 3];
const [x, y, z] = point;
const [, a, b] = point;  // Ignore the first one
```

```
const points = [
  [1, 2, 3],
  [4, 5, 6],
];
points.forEach(([x, y, z]) => console.log(x + y + z));
// Logs 6, 15
```

함수 매개변수 기본값 사용하기

- 자바스크립에서 함수의 모든 매개변수는 선택적(생략 가능)이며,
  매개변수를 지정하지 않으면 undefined로 간주된다.
- 매개변수에 기본값을 지정하면 코드가 간결해질 뿐만 아니라,
  base가 선택적 매개변수라는 것을 명확히 나타낼 수 있다.

```
1    // tsConfig: {"noImplicitAny":false}
2
3    function parseNum(str, base=10) {
4      return parseInt(str, base);
5    }
```

저수준 프로미스나 콜백 대신 async/await 사용하기

- async/await 를 사용하면 코드가 간결해져서 버그나 실수를 방지할 수 있고, 비동기 코드에 타입 정도가 전달되어 타입 추론이 가능하다.

```
function getJSON(url: string) {
  return fetch(url).then(response => response.json());
}
function getJSONCallback(url: string, cb: (result: unknown) => void) {
  // ...
}
```

```
async function getJSON(url: string) {
  const response = await fetch(url);
  return response.json();
}
```

연관 배열에 객체 대신 Map 과 Set 사용하기

```
1  function countWords(text:string) {
2    const counts: {[word:string]:number} = {};
3    for (const word of text.split(/[\s,.]+/)) {
4      counts[word] = 1 + (counts[word] || 0)
5    }
6    return counts;
7  }
8
9  console.log(countWords('Objects have a constructor'))
```

```
▼ {Objects: 1, have: 1, a: 1, constructor: '1function Object() { [native code] }'}
    Objects: 1
    a: 1
    constructor: "1function Object() { [native code] }"
    have: 1
  ▶ [[Prototype]]: Object
```

```
1  function countWords(text:string) {
2    const counts = new Map<string, number>();
3    for (const word of text.split(/[\s,.]+/)) {
4      counts.set(word, 1+ (counts.get(word) || 0));
5    }
6    return counts;
7  }
8
9  console.log(countWords('Objects have a constructor'))
```

```
▼ Map(4)
  ▼ [[Entries]]
    ▶ 0: {"Objects" => 1}
    ▶ 1: {"have" => 1}
    ▶ 2: {"a" => 1}
    ▶ 3: {"constructor" => 1}
      size: 4
    ▶ [[Prototype]]: Map
```

타입스크립크에 use strict 넣지 않기

- ES5에서 엄격 모드(strict mode)가 도입

: 버그가 될 수 있는 코드 페턴에 오류를 표시

- 타입스크립트에서 수행하는 안전성 검사(sanity check)가 엄격 모드보다 훨씬 더 엄격한 체크를 하기 때문에, 타입스크립트 코드에서는 'use strict'가 무의미하다.
- 타입스크립트 컴파일러가 생성하는 자바스크립트 코드에서 'use strict'가 추가된다.
  타입스크립트 코드에 'use strict'를 쓰지 않고, alwaysStrict 컴파일러 옵션을 설정한다.

```
{
  "compilerOptions": {
    "alwaysStrict": true
  }
}
```

- 자바스크립트 표준화 3단계 이상의 기능은 타입스크립트 내에서 사용할 수 있다.

요약

- 타입스크립트 개발 환경은 모던 자바스크립트도 실행할 수 있으므로
  모던 자바스크립트의 최신 기능을 적극적으로 사용하자.
- 타입스크립트 개발 환경에서는 컴파일러와 언어 서비스를 통해 클래스, 구조 분해,
  async/await 같은 기능을 쉽게 배울 수 있다.
- ‘use strict’ 는 타입스크립트 컴파일러 수준에서 사용되므로 코드에서 제거하자.
- TC39(자바스크립트를 관장하는 표준 기구)의 깃헙 저장소와
  타입스크립트의 릴리스 노트를 통해 최신 기능을 확인하자.