아이템 17

변경 관련된 오류 방지를 위해 readonly 사용하기

채희수 / 23.05.14

readonly?

타입 선언문 앞에 `readonly` 키워드 추가 시 변수나 속성을 읽기 전용으로 만들어준다.

이는 변수 또는 속성을 선언하고 초기화한 후에는 수정할 수 없음을 의미한다.

```
1   class Person {
2     readonly name: string;
3
4     constructor(name: string) {
5       this.name = name;
6     }
7   }
8
9   const person = new Person("John");
10  console.log(person.name); // "John"
11
12  person.name = "Jane"; // Cannot assign to 'name' because it is a read-only property.
```

readonly 언제 사용해?

함수가 매개변수를 변경하지 않는다면, readonly로 선언하자

```
function arraySum(arr: readonly number[]) {
  let sum = 0;
  for(const num of arr) {
    sum += num;
  }
  return sum;
}
```

```
function processTuple(tuple: readonly [string, number, boolean]) {
  // 튜플 요소를 처리하는 로직
  console.log(tuple[0]); // 첫 번째 요소 출력
  console.log(tuple[1]); // 두 번째 요소 출력
  console.log(tuple[2]); // 세 번째 요소 출력

  tuple[0] = "banana"; // Cannot assign to '0' because it is a read-only property.
}

const myTuple: [string, number, boolean] = ["apple", 10, true];
processTuple(myTuple);
```

readonly는 배열 및 튜플 리터럴 타입에 한정적으로 사용한다.

```
// 'readonly' type modifier is only permitted on array and tuple literal types.
function cl (message : readonly string) {
  console.log(message)
}
```

함수의 매개변수를 readonly로 선언하면

```
1  function arraySum(arr: readonly number[]) {
2    let sum = 0;
3    for(const num of arr) {
4      sum += num;
5    }
6    return sum;
7  }
```

readonly number[]

- 배열의 요소를 읽을 수는 있지만, 쓸 수는 없다.
- length를 읽을 수는 있지만, 바꿀 수는 없다.
- 배열을 변경하는 pop을 비롯한 다른 메서드를 호출 할 수 없다.

readonly의 장점!

타입스크립트는 매개변수가 함수 내에서 변경이 일어나는지 체크한다.
-> 의도치 않은 변경은 방지할 수 있다.

어떤 함수를 readonly로 만들면,
그 함수를 호출하는 다른 함수도 모두 readonly로 만들어야 한다.
-> 인터페이스를 명확하게 하고 타입 안정성을 높을 수 있다.
* 다른 라이브러리 함수를 호출하는 경우라면, 타입 단언문(as number[]) 사용

-> 지역 변수와 관련된 모든 종류의 변경 오류를 방지할 수 있고,
변경이 발생하는 코드도 쉽게 찾을 수 있다.

그런데,, 값을 변경하지 못하게 하는 거면, const도 있잖아. const와는 어떻게 달라?

	언제 사용해?	객체의 수정은 가능해?
Const	상수 선언	가능해. const로 선언된 객체의 속성은 수정 가능하며, 중첩된 객체 내부의 속성도 수정할 수 있어.
Readonly	클래스 속성, 타입이나 인터페이스 속성, 함수의 매개변수	얕게 동작해서 수정이 가능할 수도 있어.

readonly 얕게 동작한다.

```ts
interface Person {
    name: string;
    address: {
        street: string;
        city: string;
    };
}

const person: Readonly<Person> = {
    name: "John",
    address: {
        street: "123 Main St",
        city: "New York",
    },
};

person.name = "hesu" // Cannot assign to 'name' because it is a read-only property.
person.address.city = "GM"
```

= 중첩된 객체의 속성을 변환이 가능하다.

readonly가 얕게 동작한다면, 깊게 동작하게 할 수 있는 방법은 없을까?

'DeepReadonly'를 사용하면
중첩된 객체의 모든 속성을 읽기 전용으로 만들 수 있다.
(TypeScript 4.1 버전부터 도입된 타입 변환 기능)

```
type DeepReadonly<T> =
    T extends (infer R)[]
        ? DeepReadonlyArray<R>
        : T extends object
            ? DeepReadonlyObject<T>
            : T;

interface DeepReadonlyArray<T> extends ReadonlyArray<DeepReadonly<T>> {}

type DeepReadonlyObject<T> = {
    readonly [P in keyof T]: DeepReadonly<T[P]>;
};
```

중첩된 객체에 대해서도 재귀적으로 동작하기 때문에 객체 내부의 객체까지 모두 읽기 전용으로 변환된다.

DeepReadonly를 사용하면 중첩된 객체의 속성까지 모두 읽기 전용으로 변환돼.

```
const person: DeepReadonly<Person> = {
    name: "John",
    address: {
        street: "123 Main St",
        city: "New York",
    },
};

person.name = "hesu" // Cannot assign to 'name' because it is a read-only property.
person.address.city = "GM" // Cannot assign to 'city' because it is a read-only property.
```

요약

-> 만약 함수가 매개변수를 수정하지 않는다면 readonly로 선언하자.

-> readonly를 사용하면 변경하면서 발생하는 오류를 방지하고, 변경이 발생하는 코드도 쉽게 찾을 수 있다.

-> const와 readonly는 다르다.

-> readonly는 얕게 동작하니, 중첩배열의 타입은 DeepReadonly를 사용하자.