Item 21 타입 넓히기

강철원

1. 타입스크립트가 넓히기를 통해 상수의 타입을 추론하는 법을 이해해보자

2. 동작에 영향을 줄 수 있는 방법인 const, 타입 구문, as const에 익숙해지자

타입 넓히기란?

TS가 작성된 코드를 체크하는 정적 분석 시점에,
변수는 '가능한' 값들의 집합을 가진다.

상수를 사용해서 변수를 초기화 할 때 타입을 명시하지 않으면
타입 체커는 타입을 결정해야 한다.

이 말은 지정된 단일 값을 가지고 할당 가능한 집합을 유추해야하는데
이를 '넓히기' 라고 한다.

```ts
interface Vector3 {
  x: number
  y: number
  z: number
}

const getComponent = (vector: Vector3, axis: 'x' | 'y' | 'z') => {
  return vector[axis]
}

let x = 'x' // 타입은 string
let vec = {x:10, y:20, z: 30};

getComponent(vec, x) // 'string' 형식의 인수는 '"x" | "y" | "z"' 형식의 매개 변수에 할당될 수 없습니다
```

실행은 되지만 편집기에서 오류 발생

```
const mixed = ['x', 1]
```

mixed 타입의 후보

- ('x' | 1)[]
- ['x', 1]
- [string, number]
- readonly [string, number]
- (string | number)[]
- readonly (string | number)[]
- [any, any]
- any[]

타입스크립트가 아무리 영리하더라도 사람의 마음까지 읽을 수는 없다.

↓

넓히기의 과정을 제어할 수 있도록 몇가지 방법 제공

- cosnt
- as const

const

```ts
interface Vector3 {
  x: number;
  y: number;
  z: number;
}
function getComponent(vector: Vector3, axis: 'x' | 'y' | 'z') {
  return vector[axis];
}

const x = 'x';
let vec = { x: 10, y: 20, z: 30 };
getComponent(vec, x);
```

const 로 변수를 선언하면 더 좁은 타입이 된다.

하지만 const가 만능은 아니다.

객체에서 문제가 발생한다.

```ts
const v = {
  x: 1,
};

v.x = 3;
v.x = '3';
v.y = 4;
v.name = 'Pythagoras';
```

```
any
Property 'y' does not exist on type '{ x: number; }'. ts(2339)
View Problem (⌥F8)    No quick fixes available
```

```
(property) x: number
Type 'string' is not assignable to type 'number'. ts(2322)
View Problem (⌥F8)    No quick fixes available
```

```
any
Property 'name' does not exist on type '{ x: number; }'. ts(2339)
View Problem (⌥F8)    No quick fixes available
```

- string 할당 불가
- 다른 속성 추가 불가

as const

```ts
interface Vector3 {
  x: number;
  y: number;
  z: number;
}
function getComponent(vector: Vector3, axis: 'x' | 'y' | 'z') {
  return vector[axis];
}
const v1 = {
  x: 1,
  y: 2,
}; // 타입은 { x: number; y: number; }

const v2 = {
  x: 1 as const,
  y: 2,
}; // 타입은 { x: 1; y: number; }

const v3 = {
  x: 1,
  y: 2,
} as const; // 타입은 { readonly x: 1; readonly y: 2; }
```

as const 를 작성하면 타입스크립트는 최대한 좁은 타입으로 추론한다.

배열을 튜플 타입으로 추론할 때에도 as const를 사용

```ts
const a1 = [1, 2, 3]; // 타입이 number[]
const a2 = [1, 2, 3] as const; // 타입이 readonly [1, 2, 3]
```