item 50

# 오버로딩 타입보다는 조건부 타입을 사용하기

채희수 / 23.07.30

## * 함수 오버로딩 : 동일한 이름에 매개변수만 다른 여러 버전의 함수를 허용하는 것

**타입스크립트가 함수 오버로딩 기능을 지원하기는 하지만, 온전히 타입 수준에서만 동작한다.**

```
function add(a: number, b: number) { return a + b; }
    // ~~~ Duplicate function implementation
function add(a: string, b: string) { return a + b; }
    // ~~~ Duplicate function implementation
```

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

// 하나의 함수에 대해 여러 개의 선언문 작성
function add(a: number, b: number): number;
function add(a: string, b: string): string;

// 구현체는 오직 한개
function add(a, b) {
  return a + b;
}

const three = add(1, 2);  // Type is number
const twelve = add('1', '2');  // Type is string
```

```ts
// 구현체는 오직 한개
function add(a, b) {
  return a + b;
}

// 오늘의 예제
function double(x) {
  return x + x;
}
```

저 빨간줄을 없애기 위해 조건부 타입을 배워볼까요?

## double 함수의 타입구현목표 ! 두둥 !

1. 매개변수의 타입이 반환타입이 되도록
2. number와 string을 모두 사용할 수 있도록

```
function double(x: number|string): number|string;
function double(x: any) { return x + x; }
```

유니온 타입으로 정의해볼게요. 선언하고 보니 모호한 점이 있네요.

```
const num = double(12);  // string | number
const str = double('x');  // string | number
```

매개변수타입이 반환타입이 되진 않죠.

목표와 다르게 반환타입을 모두 포함합니다.

그렇다면, 이번엔 제네릭을 사용해볼게요.

```
function double<T extends number|string>(x: T): T;
function double(x: any) { return x + x; }
```

```
const num = double(12);  // Type is 12
const str = double('x');  // Type is "x"
```

이번에도 매개변수타입이 반환타입이 되지 않네요.

그리고 매개변수 자체를 타입으로 선언하네요. 너무 구체적입니다.

아, 아까 맨 처음 함수 오버로딩에서 봤던 코드처럼

여러 가지 타입 선언으로 분리해봐야겠습니다.

```
function double(x: number): number;
function double(x: string): string;
function double(x: any) { return x + x; }

const num = double(12);  // Type is number
const str = double('x');  // Type is string
```

오, 매개변수 타입을 반환타입으로 반환합니다!

그런데,

```
function f(x: number|string) {
  return double(x);
           // ~ Argument of type 'string | number' is not assignable
           //   to parameter of type 'string'
}
```

다른 함수에서 double함수를 호출하려고하니

유니온 타입 stirng | number 를  string 또는 number에 할당할 수 없다는 오류가 납니다.

세 번째 오버로딩(string|number)을 추가할수도 있지만, 가장 좋은 해결책인 조건부타입을 사용하겠습니다.

```
function double<T extends number | string>(
  x: T
): T extends string ? string : number;
function double(x: any) { return x + x; }
```

조건부타입은 삼항연산자(?:)처럼 사용합니다.

- T가 string의 부분집합(string, 문자열 리터럴, 문자열 리터럴의 유니온)이면, 반환 타입은 string입니다.
- 그 외의 경우는 반환 타입은 number입니다.

아까 오류가 발생했던 유니온 타입을 어떻게 해석하는지 살펴보면!

```ts
function double<T extends number | string>(
  x: T
): T extends string ? string : number;
function double(x: any) { return x + x; }
```

+

```ts
// function f(x: string | number): string | number
function f(x: number|string) {
  return double(x);
}
```

조건부 타입의 유니온으로 분리되어 해석하기 때문에 오류가 사라집니다.

```ts
(number|string) extends string ? string : number;
-> (number extends string ? string : number) | (string extends string ? string : number)
-> number | string
```

## 요약

- 오버로딩 타입보다 **조건부 타입을 사용하는 것이 좋다.**
  조건부 타입은 추가적인 오버로딩 없이 **유니온 타입을 지원**할 수 있다.