

Item 12.

함수 표현식에 타입 적용하기

Effective TypeScript

목 차

1. 타입스크립트에서 함수 정의하기
2. 함수 타입 선언하기
3. 다른 함수 시그니처 참조하기
4. 결론

O'REILLY®

Effective TypeScript

62 Specific Ways to Improve Your TypeScript



Dan Vanderkam

1. 타입스크립트에서 함수 정의하기

함수를 정의하는 방법

- 함수 선언문

```
1 // 함수 선언문
2 function funcStatement(params: number): number { /* ... */ };
```

- 함수 표현식

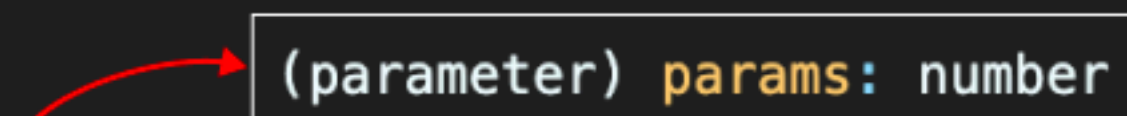
```
1 // 함수 표현식
2 const funcExpression1 = function(params: number): number { /* ... */ };
3 const funcExpression2 = (params: number): number => { /* ... */ };
```

함수의 매개변수부터 반환값까지 전체를 함수 타입으로 선언하여

함수 표현식에 **재사용** 가능

1. 타입스크립트에서 함수 정의하기

```
1 // 함수 타입 선언
2 type FuncType = (params: number) => number;
3
4 // 함수 표현식: 함수 타입 재사용 가능
5 const funcExpression: FuncType = params => { return 0; };
6
7 // 함수 선언식: 함수 타입 재사용 불가
8 function funcStatement(params: number): number { return 0; }
```



선언된 함수 타입을 “함수 표현식”에서는 **재사용 가능**

⇒ 에디터에서 params에 마우스를 올려 보면, params의 타입을 number로 인식

반면, “함수 선언문”에는 함수 타입 **재사용 불가**

2. 함수 타입 선언하기

함수 타입은 **재사용 가능**

⇒ 함수 타입의 선언은 불필요한 코드의 반복을 줄임



```
1 // 함수 선언문으로 정의된 사칙연산을 수행하는 함수
2 function add(a: number, b: number) { return a + b; }
3 function sub(a: number, b: number) { return a - b; }
4 function mul(a: number, b: number) { return a * b; }
5 function div(a: number, b: number) { return a / b; }
```




```
1 // 함수 타입 선언
2 type BinaryFn = (a: number, b: number) => number;
3
4 // 함수 표현식으로 정의된 사칙연산을 수행하는 함수
5 const add: BinaryFn = (a, b) => a + b;
6 const sub: BinaryFn = (a, b) => a - b;
7 const mul: BinaryFn = (a, b) => a * b;
8 const div: BinaryFn = (a, b) => a / b;
```

- 반복되는 함수 시그니처를 하나의 함수 타입으로 통합
- 매개변수의 타입과 반환 타입의 반복된 코드를 줄임
- 함수 구현부와 분리되어 로직이 보다 분명해짐

3. 다른 함수 시그니처 참조하기

fetch 함수가 특정 리소스에 HTTP 요청을 보내고, response.json() 사용해 응답 데이터를 추출



```
1  async function getData() {  
2    const response = await fetch('/data');  
3    const result = await response.json();  
4    return result; // result는 JSON 형식의 데이터  
5  }
```

정상적으로 응답한 경우, JSON 형태의 데이터 반환

그렇지 않은 경우, JSON 형식이 아닌 오류 메시지를 담은 rejected 프로미스 반환

3. 다른 함수 시그니처 참조하기

응답 상태 체크를 수행할 함수에 fetch의 타입을 적용하여 다음과 같이 코드를 작성



```
1 // lib.dom.d.ts
2 declare function fetch(input: RequestInfo, init?: RequestInit): Promise<Response>;
3
4 const checkedFetch: typeof fetch = async (input, init) => {
5   const response = await fetch(input, init);
6   if (!response.ok) {
7     // 비동기 함수 내에서 거절된 프로미스 반환
8     throw new Error('Request failed: ' + response.status);
9   }
10  return response;
11 }
12
```

- 함수 표현식으로 작성
- 함수 전체에 타입(typeof fetch) 적용

타입스크립트가 input과 init의 타입을 추론할 수 있고, checkedFetch의 반환 타입을 보장

3. 다른 함수 시그니처 참조하기

만약 throw 대신 return을 사용한다면?

⇒ 타입스크립트는 오류를 잡아냄

```
1 const checkedFetch: typeof fetch = async (input, init) => {  
2   const response = await fetch(input, init);  
3   if (!response.ok) {  
4     return new Error('Request failed: ' + response.status);  
5   }  
6   return response;  
7 }
```

```
const checkedFetch: {  
  (input: RequestInfo | URL, init?: RequestInit | undefined): Promise<Response>;  
  (input: RequestInfo, init?: RequestInit | undefined): Promise<...>;  
}  
  
Type '(input: any, init: any) => Promise<Response | Error>' is not assignable to type '{ (input: RequestInfo | URL, init?: RequestInit | undefined): Promise<Response>; (input: RequestInfo, init?: RequestInit | undefined): Promise<...>; }'.  
  Type 'Promise<Response | Error>' is not assignable to type 'Promise<Response>'.  
    Type 'Response | Error' is not assignable to type 'Response'.  
      Type 'Error' is missing the following properties from type 'Response': headers, ok, redirected, status, and 11 more. ts(2322)
```

함수 표현식 전체 타입을 정의하는 것은 코드를 간결하게 할 뿐만 아니라 **안전하게** 함

4. 결론

타입스크립트에서 함수 정의는 **함수 표현식을 사용하는** 것이 좋습니다.

함수를 작성할 때 매개변수의 타입과 반환 타입을 반복해서 작성하지 않고 **함수 전체의 타입을 선언하여 적용**해야 합니다.

이를 통해 불필요한 코드를 줄이고, 보다 안전하게 함수를 사용할 수 있습니다.

감사합니다 :))

Effective TypeScript

@Bori-github(이보리)