

## 테스팅 타입의 함정에 주의하기

Item 52

여러분들이 프로젝트를 공개하려고 할 때  
오류없이 의도하는대로 동작하는지 확인하기 위해

테스트 코드를 작성하는 것은 필수이며  
타입 선언도 테스트를 거쳐야 합니다

하지만 타입 선언을 테스트하기는 매우 어렵습니다

# 타입 선언을 테스트하기 어려운 이유

## 1. 타입스크립트가 타입을 추론하는 과정을 정확하게 테스트하는 것은 어렵습니다.

- 타입스크립트는 타입을 추론하는 기능을 갖추고 있습니다. 때문에 타입 선언이 일부 경우에는 명시적으로 작성하지 않아도 자동으로 추론될 수 있습니다.

## 2. 복잡한 타입이나 제너릭 타입을 다룰 때는 타입 선언을 테스트하기가 어려울 수 있습니다.

- 특히 제너릭 타입은 다양한 유형의 입력을 수용하는 유틸리티 함수 등에서 많이 사용되며, 모든 케이스를 테스트하는 것은 번거로울 수 있습니다.

## 3. 외부 라이브러리나 타입 정의 파일을 사용하는 경우에도 타입 선언을 테스트하기 어려울 수 있습니다.

- 외부 의존성과의 상호작용을 테스트하려면 해당 의존성에 대한 테스트 환경을 구성해야 합니다.

## 4. 타입스크립트는 컴파일 타입에 타입 오류를 체크해주지만, 타입 선언 자체를 테스트하는 데에는 특별한 도구의 지원이 부족할 수 있습니다.

- 이로 인해 타입 선언에 대한 유닛 테스트를 작성하거나 자동화하기 어려울 수 있습니다.

# 해결 방법

- dtslint 와 같은 타입을 검사하는 도구 사용 추천

microsoft / DefinitelyTyped-tools

Q Type ↗ to search

>\_

+ ▾

🔄

🔗

📧

👤

<> Code

🕒 Issues 45

🔗 Pull requests 40

🎮 Actions

📁 Projects

🛡 Security

📊 Insights

📁 master ▾

DefinitelyTyped-tools / packages / dtslint / 📄

Q Go to file

t

Add file ▾

⋮

🤖 typescript-bot v0.0.166-next.1

00562c6 · yesterday 🕒 History

Name	Last commit message	Last commit date
📁 ..		
📁 docs	Add dtslint/dts-critic to the monorepo (#353)	2 years ago
📁 src	Format	2 days ago
📁 test	fix no-import-of-default-export (#659)	5 months ago

README.md

✎ ⋮

`dtslint` tests a TypeScript declaration file for style and correctness. It will install `typescript` and `tslint` for you, so this is the only tool you need to test a type definition.

Lint rules new to dtslint are documented in the [docs](#) directory.



```
1 declare function map<U, V>(array: U[], fn: (u: U) => V): V[];
```

유틸리티 라이브러리(Lodash...)가 제공하는 map함수에 대한 타입 선언을 작성해보았다고 가정해 봅시다.

이 타입 선언이 예상한 타입으로 잘 작동하는지 어떻게 확인 할 수 있을까요?

-> 해당 함수를 호출하는 테스트 파일을 작성하는것




```
1 map(['2017', '2018', '2019'], v => Number(v));
```

만약 map의 타입 선언에 단 하나의 매개변수만 작성되어 있다면 이를 잡아낼 수 있을 것입니다.

그러나 반환값에대한 체크가 누락되어 있기 때문에 완전한 테스트라고 할 수 없습니다.

## 예시 2

square라는 함수의 런타임 동작을 테스트한다면 다음과 같은 테스트 코드가 됩니다.



```
1  const square = (x: number) => x * x;
2  test('square a number', () => {
3    square(1);
4    square(2);
5  });
6
```

이 테스트는 square 함수가 에러를 발생시키지 않는지 확인합니다.  
그러나 반환값에 대한 검사가 없으므로 실제 동작에 대한 테스트는 이루어지지 않습니다.  
square 함수의 잘못된 구현이라도 이 테스트를 통과할 수 있습니다.

타입 선언 파일을 테스트할 때는 이 테스트 코드처럼 단순히 함수를 실행만 하는 방식을 일반적으로 적용하게 됩니다.

그 이유는 라이브러리 구현체의 기존 테스트 코드를 복사하면 간단히 만들 수 있기 때문입니다.

# 라이브러리 기존 유닛 테스트 코드 예시



```
1 // utils.test.ts
2 import { map } from './utils';
3
4 test('map function converts string array to number array', () => {
5   // 입력 배열
6   const inputArray = ['1', '2', '3'];
7
8   // map 함수를 사용하여 string 배열을 number 배열로 변환
9   const resultArray = map(inputArray, (str) => Number(str));
10
11   // 변환된 결과와 기대하는 결과를 비교하여 테스트
12   expect(resultArray).toEqual([1, 2, 3]);
13 });
14
15 test('map function works correctly with an empty array', () => {
16   // 빈 배열
17   const inputArray: string[] = [];
18
19   // map 함수를 사용하여 빈 배열 변환
20   const resultArray = map(inputArray, (str) => Number(str));
21
22   // 빈 배열로 변환해야 함
23   expect(resultArray).toEqual([]);
24 });
```

utils.d.ts



```
1 declare function map<U, V>(array: U[], fn: (u: U) => V): V[];
2
```

test-utils.ts



```
1 // 타입 선언 파일을 가져옵니다.
2 import { map } from './utils';
3
4 // 테스트 함수를 작성합니다.
5 function testMap() {
6     const inputArray = ['1', '2', '3'];
7
8     // map 함수를 사용하여 string 배열을 number 배열로 변환합니다.
9     const resultArray = map(inputArray, (str) => Number(str));
10
11     // 변환된 결과를 출력합니다.
12     console.log(resultArray);
13 }
14
15 // 테스트 함수를 호출합니다.
16 testMap();
17
```

test-type-utils.ts



```
1 // 타입 선언 파일을 가져옵니다.
2 import { map } from './utils';
3
4 // 타입 선언에 대한 테스트 함수를 작성합니다.
5 function testTypeDeclaration() {
6     // map 함수의 반환값이 number[] 타입인지 확인합니다.
7     const resultArray: number[] = map(['1', '2', '3'], (str) => Number(str));
8
9     // 결과를 출력합니다.
10    console.log(resultArray);
11 }
12
13 // 타입 선언에 대한 테스트 함수를 호출합니다.
14 testTypeDeclaration();
```

**map** 함수의 반환값을 **number[]** 타입으로 명시적으로 지정하여 타입선언을 테스트하는 예시



## 이미 사용하고자하는 라이브러리에서 테스트코드를 만들어서 다 확인했는데 내가 다시 확인해야해?

때로는 라이브러리의 유닛 테스트가 타입 선언에 대해 **충분히 검증하지 못하는** 경우가 있을 수 있습니다.  
또한 라이브러리의 업데이트나 타입스크립트 버전 변경 등으로 인해 **타입 선언이 올바르게 작동하지 않을** 수 있습니다.

따라서, 만약 라이브러리의 유닛 테스트가 타입 선언의 정확성을 충분히 검증하고 있다고 확신할 수 있다면 따로 타입 선언에 대한 테스트를 작성할 필요는 없습니다.

**그러나 확신이 서지 않는다면, 타입 선언에 대한 추가적인 테스트를 작성하여 라이브러리의 안정성을 더욱 확보하는 것이 좋습니다.**

타입 선언 파일을 테스트할 때는 이 테스트 코드처럼 단순히 함수를 실행만 하는 방식을 일반적으로 적용하게 됩니다.

그 이유는 라이브러리 구현체의 기존 테스트 코드를 복사하면 간단히 만들 수 있기 때문입니다.

함수를 실행만 하는 테스트 코드가 의미 없는 것은 아니지만  
**실제로 반환 타입을 체크하는 것이 훨씬 좋은 테스트 코드입니다.**

## 반환값을 특정 타입의 변수에 할당하여 간단히 반환 타입을 체크할 수 있는 방법



```
1 declare function map<U, V>(array: U[], fn: (u: U) => V): V[];  
2 const lengths: number[] = map(['john', 'paul'], name => name.length);
```

이 코드는 일반적으로 불필요한 타입 선언(Item 19 : 추론 가능한 타입을 사용해 장황한 코드 방지)에 해당합니다.

그러나 테스트 코드관점에서는 중요한 역할을 하고 있습니다.

**map** 함수의 반환 타입이 **number[]**임을 보장합니다.

그러나 테스트를 위해 할당을 사용하는 방법에는 두 가지 근본적인 **문제**가 있습니다.

# 첫 째는 불필요한 변수를 만들어야 합니다.

반환값을 할당하는 변수는 샘플코드처럼 쓰일 수 있지만, 일부 린팅 규칙(미사용 변수 경고)을 비활성해야 합니다.

## 해결책

- 변수를 도입하는 대신 헬퍼 함수를 정의하는 것



```
1 declare function map<U, V>(array: U[], fn: (u: U) => V): V[];
2 function assertType<T>(x: T) {}
3
4 assertType<number[]>(map(['john', 'paul'], (name) => name.length));
5
```

이 코드는 불필요한 변수 문제를 해결하지만, 또 다른 문제점이 남아 있습니다.