Effective Typescript

테스팅 타입의 함정에 주의하기

Item 52

ryan-dia(강철원)

여러분들이 프로젝트를 공개하려고 할 때

오류없이 의도하는대로 동작하는지 확인하기 위해

테스트 코드를 작성하는 것은 필수이며
타입 선언도 테스트를 거쳐야 합니다

하지만 **타입 선언**을 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.

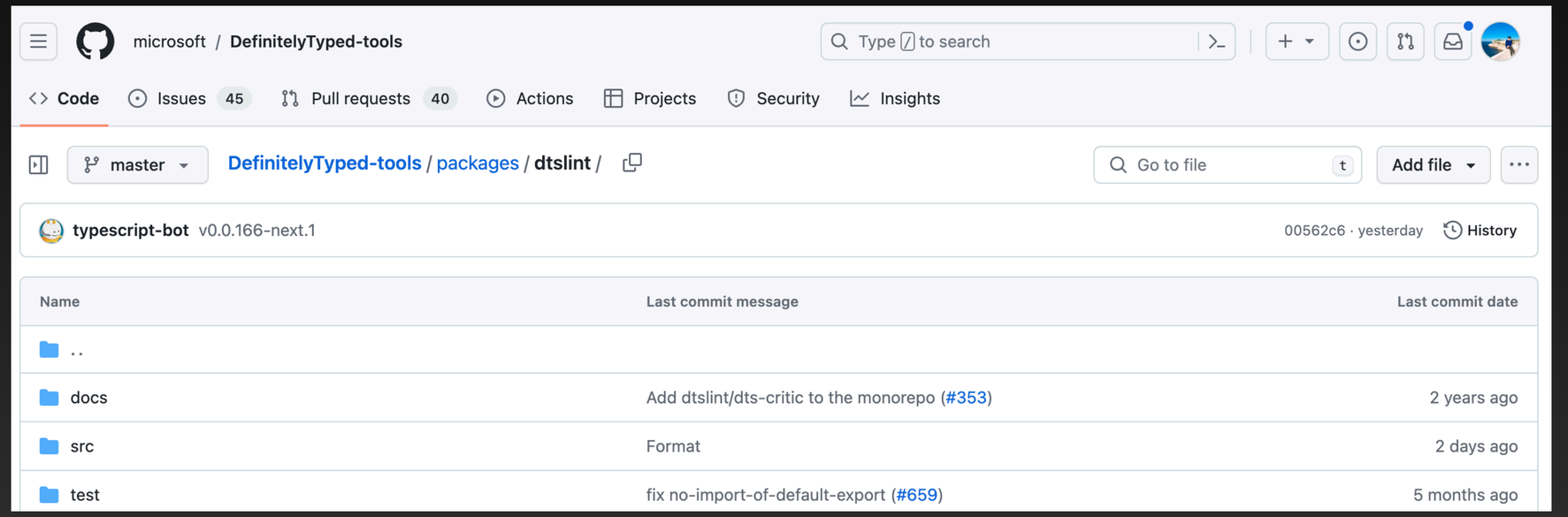
- 외부 의존성과의 상호작용을 테스트하려면 해당 의존성에 대한 테스트 환경을 구성해야 합니다.

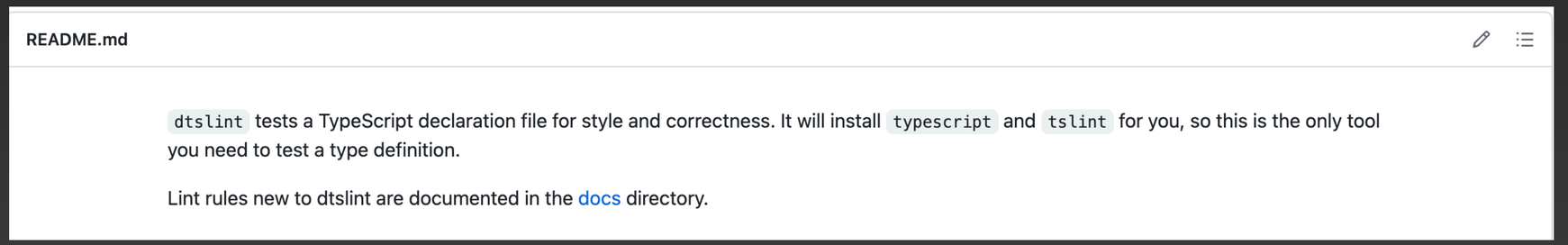
4. 타입스크립트는 컴파일 타입에 타입 오류를 체크해주지만, 타입 선언 자체를 테스트하는 데에는 특별한 도구의 지원이 부족할 수 있습니다.

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

해결 방법

- dtslint 와 같은 타입을 검사하는 도구 사용 추천





```
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

```
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 테스팅할 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

```
// utils.test.ts
import { map } from './utils';

test('map function converts string array to number array', () => {
  // 입력 배열
  const inputArray = ['1', '2', '3'];

  // map 함수를 사용하여 string 배열을 number 배열로 변환
  const resultArray = map(inputArray, (str) => Number(str));

  // 변환된 결과와 기대하는 결과를 비교하여 테스트
  expect(resultArray).toEqual([1, 2, 3]);
});

test('map function works correctly with an empty array', () => {
  // 빈 배열
  const inputArray: string[] = [];

  // map 함수를 사용하여 빈 배열 변환
  const resultArray = map(inputArray, (str) => Number(str));

  // 빈 배열로 변환해야 함
  expect(resultArray).toEqual([]);
});
```

utils.d.ts

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

```

test-utils.ts

```ts
// 타입 선언 파일을 가져옵니다.
import { map } from './utils';

// 테스트 함수를 작성합니다.
function testMap() {
  const inputArray = ['1', '2', '3'];

  // map 함수를 사용하여 string 배열을 number 배열로 변환합니다.
  const resultArray = map(inputArray, (str) => Number(str));

  // 변환된 결과를 출력합니다.
  console.log(resultArray);
}

// 테스트 함수를 호출합니다.
testMap();

```

test-type-utils.ts

```ts
// 타입 선언 파일을 가져옵니다.
import { map } from './utils';

// 타입 선언에 대한 테스트 함수를 작성합니다.
function testTypeDeclaration() {
  // map 함수의 반환값이 number[] 타입인지 확인합니다.
  const resultArray: number[] = map(['1', '2', '3'], (str) => Number(str));

  // 결과를 출력합니다.
  console.log(resultArray);
}

// 타입 선언에 대한 테스트 함수를 호출합니다.
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 테스팅할 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

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

📌 해결책

- 변수를 도입하는 대신 헬퍼 함수를 정의하는 것

```
declare function map<U, V>(array: U[], fn: (u: U) => V): V[];
function assertType<T>(x: T) {}

assertType<number[]>(map(['john', 'paul'], (name) => name.length));

```

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

두 번째는 두 타입이 동일한지 체크하는 것이 아닌 할당 가능성을 체크하고 있습니다.

```ts
declare function map<U, V>(array: U[], fn: (u: U) => V): V[];
function assertType<T>(x: T) {}

const n = 12;
assertType<number>(n);  // OK
```

- 12는 number의 서브타입이고 할당 가능성 체크를 통과합니다.

단순히 헬퍼함수를 이런식으로 사용한다면 객체의 타입을 체크하는 경우 **문제가 발생**합니다.

객체의 타입을 체크

```
declare function map<U, V>(array: U[], fn: (u: U) => V): V[];
function assertType<T>(x: T) {}

const beatles = ['john', 'paul', 'george', 'ringo'];
assertType<{ name: string }[]>(
  map(beatles, (name) => ({
    name,
    inYellowSubmarine: name === 'ringo',
  }))
); // OK

```

map은 {name: string, inYellowSubmarine: boolean} 객체의 배열을 반환합니다.
반환된 배열은 {name: string}[]에 할당 가능하지만, **inYellowSubmarine** 속성에 대한 부분이 체크되지 않았습니다.

여기서 이상한 점이 한가지 더 있습니다.

이상한 점

```ts
declare function map<U, V>(array: U[], fn: (u: U) => V): V[];
function assertType<T>(x: T) {}

const add = (a: number, b: number) => a + b;
assertType<(a: number, b: number) => number>(add);  // OK

const double = (x: number) => 2 * x;
assertType<(a: number, b: number) => number>(double);  // OK!?
```

📌 double 함수의 체크가 성공하는 이유

타입스크립트의 함수는 매개변수가 더 적은 함수 타입에 할당 가능하기 때문입니다.

타입스크립트에서는 선언된 것보다 적은 매개변수를 가진 함수를 할당하는 것이 가능합니다.

예제 1

```
const g: (x: string) => any = () => 12;  // OK
```

예제 2

```
map(array, (name, index, array) => { /* ... */ });
```

예를 들어 로대시의 map 함수의 콜백은 세 가지 매개변수를 받습니다.

콜백 함수는 세 가지 매개변수 name , index, array 중에서 한두 개만 보통 사용합니다.
만약 매개변수의 개수가 맞지 않는 경우를 타입 체크에서 허용하지 않으면,
매우 많은 곳의 자바스크립트 코드에서 콜백 함수의 타입과 관련된 오류들을 발생하게 될 겁니다.

제대로된 assertType 사용방법

```ts
declare function map<U, V>(array: U[], fn: (u: U) => V): V[];
function assertType<T>(x: T) {}

const double = (x: number) => 2 * x;
let p: Parameters<typeof double> = null!;
assertType<[number, number]>(p);
//                           ~ Argument of type '[number]' is not
//                             assignable to parameter of type [number, number]
let r: ReturnType<typeof double> = null!;
assertType<number>(r);  // OK
```

```ts
let p: [x: number]
```

```ts
let r: number
```

```ts
let p: Parameters<typeof double> = null!;
```

this가 등장하는 콜백 함수의 경우는 또 다른 문제가 있습니다.

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

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

function assertType<T>(x: T) {}
const beatles = ['john', 'paul', 'george', 'ringo'];

assertType<number[]>(
  map(beatles, function (name, i, array) {
    // ~~~~~~~ Argument of type '(name: any, i: any, array: any) => any' is
    //         not assignable to parameter of type '(u: string) => any'
    assertType<string>(name);
    assertType<number>(i);
    assertType<string[]>(array);
    assertType<string[]>(this);
    // ~~~~ 'this' implicitly has type 'any'
    return name.length;
  })
);
```

```ts
declare function map<U, V>(
  array: U[],
  fn: (this: U[], u: U, i: number, array: U[]) => V
): V[];

function assertType<T>(x: T) {}
const beatles = ['john', 'paul', 'george', 'ringo'];

assertType<number[]>(
  map(beatles, function (name, i, array) {
    assertType<string>(name);
    assertType<number>(i);
    assertType<string[]>(array);
    assertType<string[]>(this);
    return name.length;
  })
);
```

세부 사항을 테스트하기 위해서 콜백 함수 내부에서 매개변수들의 타입과 this를 직접 체크해 보겠습니다.

권장하지 않는 모듈 선언

다음 모듈 선언을 하면 까다로운 테스트 통과할 수 있는 완전한 타입 선언 파일이 됩니다.

```
declare module 'overbar'
```

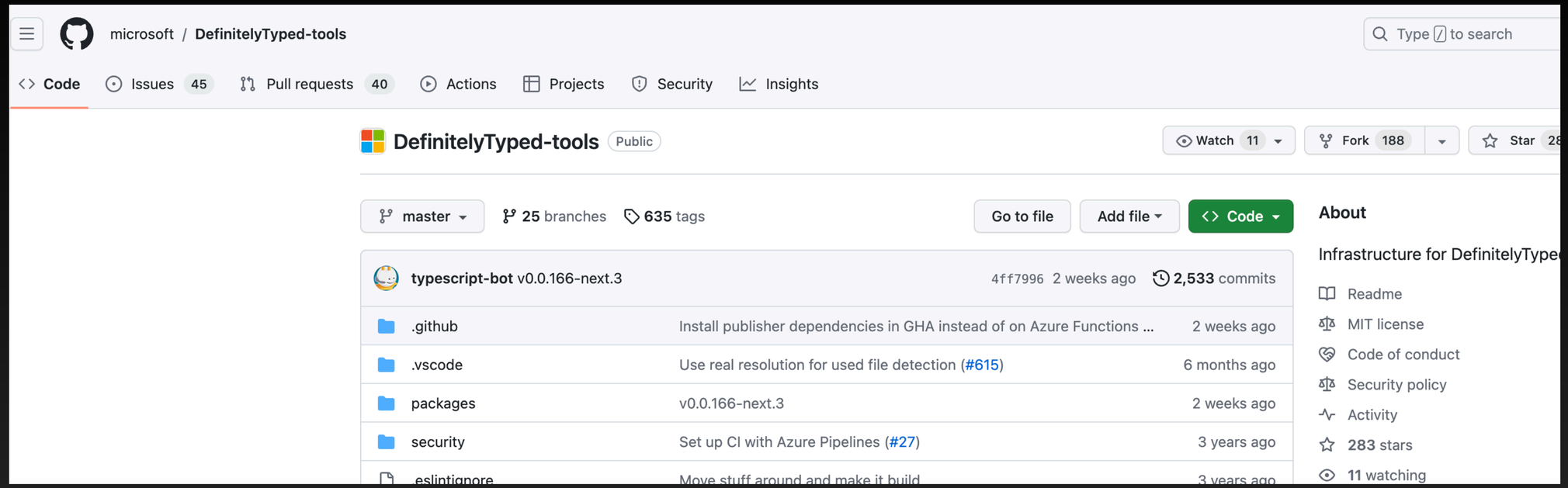
📌 why?

이 선언은 전체 모듈에 any 타입을 할당합니다.
따라서 테스트는 전부 통과하겠지만 **모든 타입 안전성을 포기**하게 됩니다.

더 나쁜 점은, 해당 모듈에 속하는 모든 함수의 호출마다 암시적으로 any 타입을 반환하기 때문에
코드 전반에 걸쳐 타입 안전성을 지속적으로 무너뜨리게 된다는 것 입니다.
noImplicitAny를 설정하더라도 타입 선언을 통해 여전히 any 타입이 생겨나게 됩니다.

타입 시스템 내에서 **암시적 any 타입을 발견해 내는 것은 매우 어렵습니다.**
이러한 어려움 때문에 타입 체커와 독립적으로 동작하는 **도구를 사용**해서 타입 선언을 테스트하는 방법이 권장됩니다.

DefinitelyTyped의 타입 선언을 위한 도구는 dtslint입니다.




```
const beatles = ['john', 'paul', 'george', 'ringo'];
map(beatles, function(
  name,  // $ExpectType string
  i,     // $ExpectType number
  array  // $ExpectType string[]
) {
  this   // $ExpectType string[]
  return name.length;
});  // $ExpectType number[]
```

추천 도구

SamVerschueren / tsd

Type / to search

Code Issues 32 Pull requests 8 Actions Projects Security Insights

tsd Public

Watch 10 Fork 57 Star 2.1k

main 3 branches 40 tags Go to file Add file Code

sindresorhus 0.28.1 45b209c on Mar 28 156 commits

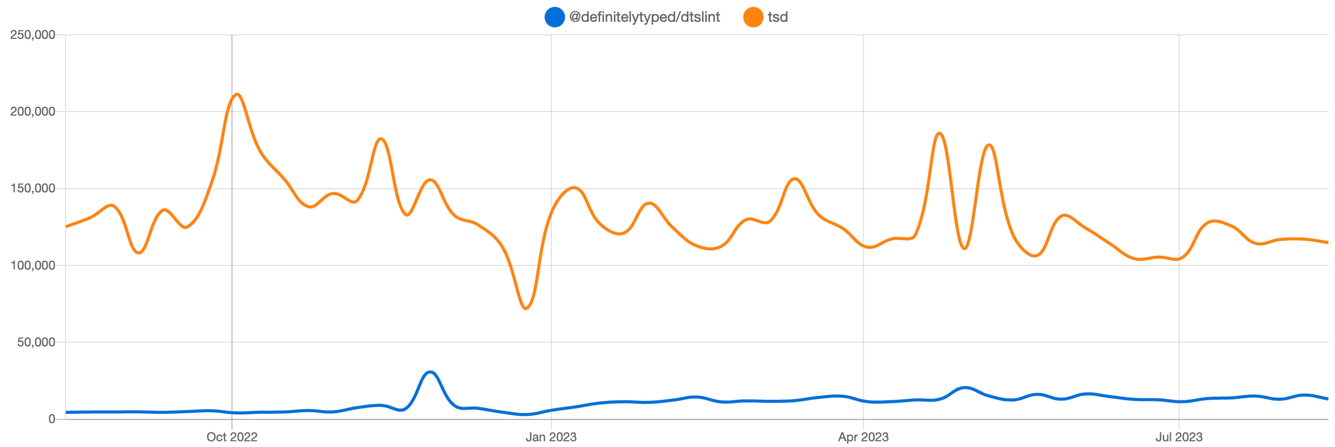
.github/workflows	Require Node.js 14	last year
media	Make expectType assertion strict	4 years ago
source	Fix error messages when `tsd` fails (#189)	5 months ago

About

Check TypeScript type definitions

Readme
MIT license
Activity
2.1k stars
10 watching

Downloads in past 1 Year



Just looking for ExpectType and ExpectError?

Use tsd instead.

```
index.test-d.ts:4:19
✖  4:19  Argument of type number is not assignable to parameter of type string.

1 error
```

📚 정리

- 타입을 테스트할 때는 특히 함수 타입의 동일성과 할당 가능성의 차이점을 알고 있어야 합니다.

- 콜백이 있는 함수를 테스트할 때, 콜백 매개변수의 추론된 타입을 체크해야 합니다.
  또한 this가 API의 일부분이라면 역시 테스트해야 합니다.

- 타입 관련된 테스트에서 any를 주의해야 합니다.
  더 엄격한 테스트를 위해 dtslint 같은 도구를 사용하는 것이 좋습니다.