

Item 8.

타입 공간과 값 공간의 심벌 구분하기

Effective TypeScript

목 차

1. 타입스크립트 코드 구분의 필요성
2. 타입 공간과 값 공간
3. 타입 공간과 값 공간에서 다른 의미를 가지는 코드 패턴
4. 결론

O'REILLY®

Effective TypeScript

62 Specific Ways to Improve Your TypeScript



Dan Vanderkam

1. 타입스크립트 코드 구분의 필요성

타입스크립트의 ***심벌(symbol)**은 타입 공간이나 값 공간 중 한 곳에 존재
⇒ 같은 이름의 심벌이라도 어느 공간에 속하느냐 따라 다른 것을 나타낼 수 있다.

```
1 interface Cylinder {  
2   radius: number;  
3   height: number;  
4 }  
5  
6 const Cylinder = (radius: number, height: number) => ({  
7   radius, height});
```

⇒ "Cylinder" 라는 이름은 같지만 서로 아무런 관련이 없다.

```
1 function calculateVolume(shape: unknown) {  
2   if (shape instanceof Cylinder) {  
3     shape.radius  
4     // ~~~~~ Property 'radius' does not exist on type '{}'  
5   }  
6 }
```

타입스크립트 코드를 읽을 때 **타입인지 값인지 구분**하는 방법을 터득해야 한다.

*심벌 : 본 내용에서 식별자를 의미

2. 타입 공간과 값 공간

[타입과 값의 구분]

`type`, `interface` 다음에 오는 심벌 ⇒ **타입**

`const`, `let` 으로 선언된 심벌 ⇒ **값**

변수나 함수를 선언하면 "값 공간"에 선언되고, 타입을 선언하면 "타입 공간"에 선언됩니다.

2. 타입 공간과 값 공간

타입 공간(type space)

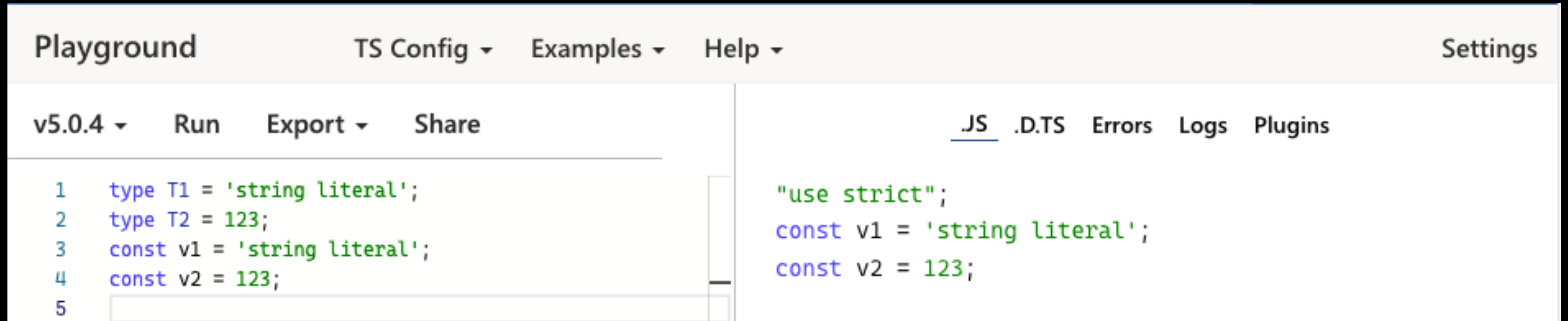
- 타입이 들어가는 공간
- 타입스크립트가 자바스크립트로 변환되면 사라짐

값 공간(value space)

- string, number, object 등 구체적인 값이 들어가는 공간
- 타입스크립트가 자바스크립트로 변환되어도 유지

2. 타입 공간과 값 공간

타입스크립트 플레이그라운드에서 자바스크립트로 변환된 예제 코드



The screenshot shows the TypeScript Playground interface. The top navigation bar includes 'Playground', 'TS Config', 'Examples', 'Help', and 'Settings'. Below this, there are tabs for 'v5.0.4', 'Run', 'Export', and 'Share'. On the right side, there are tabs for '.JS', '.D.TS', 'Errors', 'Logs', and 'Plugins'. The main editor area is split into two panes. The left pane contains TypeScript code:

```
1 type T1 = 'string literal';  
2 type T2 = 123;  
3 const v1 = 'string literal';  
4 const v2 = 123;  
5
```

 The right pane shows the converted JavaScript code:

```
"use strict";  
const v1 = 'string literal';  
const v2 = 123;
```

T1, T2 심벌은 컴파일 과정에서 제거됨 ⇒ 두 심벌은 **타입**에 해당
v1, v2 심벌은 컴파일 과정을 거쳐도 유지 ⇒ 두 심벌은 **값**에 해당

2. 타입 공간과 값 공간

`class` 와 `enum`

- `class` 와 `enum` 은 상황에 따라 타입과 값 두 가지 모두 가능한 예약어
- 값 공간과 타입공간에 모두 속함
- 타입이기도 하면서 인스턴스를 만드는 생성자 함수이기도 함

3. 타입 공간과 값 공간에서 다른 의미를 가지는 코드 패턴

[`typeof` 연산자]

타입의 관점에서 `typeof`

- 값을 읽어서 타입스크립트 타입을 반환
- 이 때 `typeof` 는 보다 큰 타입의 일부분으로 사용할 수 있고, `typeof` 구문으로 이름을 붙이는 용도로 사용할 수 있다.

값의 관점에서 `typeof`

- 자바스크립트 런타임의 `typeof` 연산자로 동작
- 대상 심벌의 런타임 타입을 가리키는 문자열을 반환

3. 타입 공간과 값 공간에서 다른 의미를 가지는 코드 패턴



```
1 class Cylinder {  
2   radius=1;  
3   height=1;  
4 }  
5  
6 const v = typeof Cylinder; // Value is "function"  
7 type T = typeof Cylinder;  // Type is typeof Cylinder
```

`class` 는 자바스크립트에서 실제 함수로 구현되므로 `const v` 값 \Rightarrow "function"

`type T` 의 타입 \Rightarrow "typeof Cylinder"

3. 타입 공간과 값 공간에서 다른 의미를 가지는 코드 패턴

[this]

- 값에서는 자바스크립트 this 키워드
- 타입에서는 다형성(polymorphic) this라고 불리는 this의 타입스크립트 타입

[& 와 |]

- 값에서는 AND와 OR 비트 연산
- 타입에서는 인터섹션 타입과 유니온 타입

3. 타입 공간과 값 공간에서 다른 의미를 가지는 코드 패턴

[const]

- `const` 는 새 변수를 선언, `as const` 는 리터럴 또는 리터럴 표현식의 추론된 타입을 변경

[extends]

- 서브클래스(`class A extends B`)
- 서브타입(`interface A extends B`)
- 제네릭 타입의 한정자(`Generic<T extends number>`)

[in]

- 루프(`for(key in object)`) 또는 매핑된 타입(mapped type)에 등장

4. 결론

작성된 타입스크립트 코드가 개발자가 의도한대로 동작하기 위해서는 해당 코드가 **타입인지 값인지 구분**할 수 있어야 합니다.

이 구분이 필요한 이유는 타입인지 값인지에 따라 다른 것을 나타낼 수 있기 때문입니다.

타입스크립트 심벌이 타입 공간에만 존재하거나 타입 공간과 값 공간 모두 존재할 수 있음을 이해해야 하고, 우리가 일반적으로 사용하는 자바스크립트의 연산자와 키워드가 타입 공간에서는 다른 목적으로 사용될 수 있음을 인지해야 합니다.

감사합니다 :))

Effective TypeScript

@Bori-github(이보리)