2024-04-03 5.6.3 union & narrowing & Object

현행 문서: 5.6.3 union & narrowing & Object

1 union & narrowing

타입스크립트가 해당 값을 바탕으로 추론을 수행하는 두 가지 핵심 개념을 소개하겠습니다.

- **유니언(union)**: 값에 허용된 타입을 두개 이상의 가능한 타입으로 확장 하는 것
- **내로잉(narrowing)**: 값에 허용된 타입이 하나 이상의 가능한 타입이 되지 않도록 좁히는 것

1.1 union 선언

```
let mathematician = Math.random() > 0.5
? undefined
: "Mark Goldberg";
```

mathematician은 undefined이거나 string일 수 있습니다.

'이거 혹은 저거'와 같은 타입을 유니언이라고 합니다.

타입스크립트는 가능한 값 또는 구성 요소 사이에 |(수직선) 연산자를 사용해 유니언 타입을 나타냅니다.

`let mathmatician: string | undefined;`

> *유니언 타입 선언의 순서는 중요하지 않습니다.*

1.2 union 속성

값이 유니언 타입일 때 타입스크립트는 유니언으로 선언한 모든 가능한 타입에 존재하는 멤버 속성에만 접근할 수 있습니다.

유니언 외의 타입에 접근하려고 하면 타입 검사 오류가 발생합니다.

```
let physicist = Math.random() > 0.5
? "Marie Curie"
: 84;

physicist.toString();
physicist.toUpperCase();
physicist.toFixed();
```

blocked URL

1.3 narrowing

내로잉은 값이 정의, 선언 혹은 이전에 유추된 것보다 더 구체적인 타입임을 코드에서 유추하는 것입니다.

타입스크립트가 값의 타입이 이전에 알려진 것보다 더 좁혀졌다는 것을 알게되 면 값을 더 구체적인 타입으로 취급합니다.

타입을 좁히는 데 사용할 수 있는 논리적 검사를 타입 가드(type guard)라고 합니다.

1.3.1 값 할당을 통한 내로잉

변수에 값을 직접 할당하면 타입스크립트는 변수의 타입을 할당된 값의 타입으로 좁힙니다.

```
    let admiral: number | string;
admiral = "Grace";

admiral.toUpperCase();
admiral.toFixed();
```

blocked URL

1.3.2 조건 검사를 통한 내로잉

일반적으로 타입스크립트에서는 변수가 알려진 값과 같은지 확인하는 if 문을 통해 변수의 값을 좁히는 방법을 사용합니다.

```
    let scientist = Math.random() > 0.5
    ? "Rosalind"
    : 51;
if (scientist === "Rosalind") {
    scientist.toUpperCase();
}
scientist.toUpperCase();
```

blocked URL

1.3.3 typeof 검사를 통한 내로잉

타입스크립트는 직접 값을 확인해 타입을 좁히기도 하지만, typeof 연산자를 사용할 수도 있습니다.

```
    let scientist = Math.random() > 0.5
    ? "Rosalind"
    : 51;
if (typeof scientist === "string") {
    scientist.toUpperCase();
}
scientist.toUpperCase();
```

blocked URL

!를 사용한 논리적 부정과 else 문도 잘 작동합니다.

```
    let scientist = Math.random() > 0.5
    ? "Rosalind"
    : 51;
if (!(typeof scientist === "string")) {
    scientist.toFixed();
} else {
    scientist.toUpperCase();
}
```

blocked URL

삼항 연산자도 지원합니다.

```
    let scientist = Math.random() > 0.5 ? "Rosalind" : 51;
typeof scientist === "string" ?
    scientist.toUpperCase() : scientist.toFixed();
```

blocked URL

1.4 엄격한 null 검사 (strict null checking)

리터럴로 좁혀진 유니언의 힘은 타입스크립트에서 엄격한 Null 검사라고 부르는 타입 시스템 영역인

'잠재적으로 정의되지 않은 undefined 값'으로 작업할 때 특히 두드러집니다.

1.4.1 십억 달러의 실수

> *저는 이를 십억 달러의 실수라고 부릅니다.*
>
> *1965년, null 참조의 발명으로 수많은 오류, 취약성 및 시스템 충돌이 발생했으며 지난 40년동안 십억 달러의 고통과 피해를 입었을 것입니다.*
>
> - *토니 호어(Tony Hoare), 2009*

'십억 달러의 실수'는 다른 타입이 필요한 위치에서 null 값을 사용하도록 허용하는 많은 타입 시스템을 가리키는 업계 용어입니다.

엄격한 null 검사가 없는 언어에서는 다음 예제 코드처럼 string 타입 변수에 null을 할당하는 것이 허용됩니다.

```
    const firstName: string = null;
```

타입 스크립트 컴파일러는 실행 방식을 변경할 수 있는 다양한 옵션을 제공합니다.

가장 유용한 옵션 하나인 strictNullChecks는 엄격한 null 검사를 활성화할지 여부를 결정합니다.

타입스크립트에서는 null 할당을 에러로 볼 수 있습니다.

blocked URL

1.5 type 별칭

타입스크립트에는 재사용하는 타입에 더 쉬운 이름을 할당하는 타입 별칭이 있습니다.

타입 별칭은 type 새로운 이름 = 타입 형태를 갖습니다.

편의상 타입 별칭은 파스칼케이스(PascalCase)로 이름을 지정합니다.

타입 별칭은 타입 애너테이션처럼 자바스크립트로 컴파일되지 않습니다.

```
    type RawData = boolean | number | string | null | undefined;
let rawDataFirst: RawData;
let rawDataSecond: RawData;
let rawDataThird: RawData;
```

blocked URL

1.6 type 결합

타입 별칭은 다른 타입 별칭을 참조할 수 있습니다.

```
    type ID = number | string;
type IDMaybe = ID | undefined | null;
```

blocked URL

1.7 Object

타입스크립트는 복잡한 객체 형태를 설명할 수 있고 객체의 할당 가능성을 확인할 수 있습니다.

1.8 Object type 선언

명시적으로 타입이 선언된 객체와는 별도로 객체의 형태를 설명하는 방법이 필요합니다.

```
    let poetLater: {
    born: number;
    name: string;
}
poetLater = {
    born: 1935,
    name: "Mary",
}
poetLater = "sat";
```

blocked URL

1.9 별칭 객체 타입

Object Type을 선언해 재사용 할 수 있습니다.

```
    type Poet = {
    born: number;
    name: string;
}

let poetLater: Poet;
poetLater = {
    born: 1935,
    name: "sara",
}

poetLater = "Emily";
```

blocked URL

1.10 구조적 타이핑

타입스크립트의 타입 시스템은 구조적으로 타입화(structually typed) 되어 있습니다.

즉, 타입을 충족하는 모든 값을 해당 타입의 값으로 사용할 수 있습니다.

```
    type WithFirstName = {
    firstName: string;
}
type WithLastName = {
    lastName: string;
}

const hasBoth = {
    firstName: "Luci",
    lastName: "Clifton",
}

let withFirstName: WithFirstName = hasBoth;
let withLastName: WithLastName = hasBoth;
```

blocked URL

1.11 교차 타입

타입스크립트에서는 **&** 교차 타입(intersection type)을 사용해 여러 타입을 동시에 나타냅니다.

```
    type Artwork = {
    genre: string;
    name: string;
}
type Writing = {
    pages: number;
    name: string;
}
type WrittenArt = Artwork & Writing;

let pa: WrittenArt = {
    genre: 'A',
    name: 'hello',
    pages: 123,
}
```

blocked URL