item 42

모르는 타입의 값에는 any 대신 unknown을 사용하기

채희수 / 23.07.08

“ 타입 뭔지 몰라 ? 그럼 unknown ! ”

1. 함수의 반환값과 관련된 unknown

```ts
function parseYAML(yaml: string): any {
  // ...
}
interface Book {
  name: string;
  author: string;
}
const book = parseYAML(`
  name: Jane Eyre
  author: Charlotte Brontë
`);
alert(book.title);
// No error, alerts "undefined" at runtime
book('read');
// No error, throws "TypeError: book is not a
// function" at runtime
```

return 타입을 any로 작성하자, 반환값의 없는 속성값에 대한 에러 표시를 하지 않는다.

1. 함수의 반환값과 관련된 unknown

```ts
function parseYAML(yaml: string): any {
  // ...
}
interface Book {
  name: string;
  author: string;
}
const book = parseYAML(`
  name: Jane Eyre
  author: Charlotte Brontë
`);
alert(book.title);
// No error, alerts "undefined" at runtime
book('read');
// No error, throws "TypeError: book is not a
// function" at runtime
```

```ts
function parseYAML(yaml: string): any {
  // ...
}
interface Book {
  name: string;
  author: string;
}
function safeParseYAML(yaml: string): unknown {
  return parseYAML(yaml);
}
const book = safeParseYAML(`
  name: The Tenant of Wildfell Hall
  author: Anne Brontë
`);
alert(book.title);
  // ~~~~ Object is of type 'unknown'.
book("read");
// ~~~~~~~~~~ Object is of type 'unknown'
```

any보다는 unknown을 쓰라고 하니, 반환타입을 unknown 으로 바꿔보자.

unknown 이라서 안된다고? 그럼 어떻게 해?

1. 함수의 반환값과 관련된 unknown

```ts
function parseYAML(yaml: string): any {
  // ...
}
interface Book {
  name: string;
  author: string;
}
const book = parseYAML(`
  name: Jane Eyre
  author: Charlotte Brontë
`);
alert(book.title);
// No error, alerts "undefined" at runtime
book('read');
// No error, throws "TypeError: book is not a
// function" at runtime
```

```ts
function parseYAML(yaml: string): any {
  // ...
}
interface Book {
  name: string;
  author: string;
}
function safeParseYAML(yaml: string): unknown {
  return parseYAML(yaml);
}
const book = safeParseYAML(`
  name: Villette
  author: Charlotte Brontë
`) as Book;
alert(book.title);
    // ~~~~~ Property 'title' does not exist on type 'Book'
book('read');
// ~~~~~~~~~~ this expression is not callable
```

unknown으로 선언된 값은 그 상태로 사용하려고 하면 오류가 나기 때문에,
적절한 타입으로 변환하도록 강제할 수 있다.

2 . 변수 선언과 관련된 unknown

어떠한 값이 있지만 그 타입을 모르는 경우 unknown을 사용한다.

```
interface Geometry {}
interface Feature {
  id?: string | number;
  geometry: Geometry;
  properties: unknown;
}
```

그리고 그 값을 사용하기 위해서 타입 변환을 한다.
타입 단언뿐만 아니라 타입 가드를 사용 할 수도 있다.

```
function processValue(val: unknown) {
  if (val instanceof Date) {
    val  // Type is Date
  }
}
```

```
type Book = {
  name: string
  author: string
}
function isBook(val: unknown): val is Book {
  return (
    typeof(val) === 'object' && val !== null &&
    'name' in val && 'author' in val
  );
}
function processValue(val: unknown) {
  if (isBook(val)) {
    val;  // Type is Book
  }
}
```

2 . 변수 선언과 관련된 unknown

unknown 대신 제너릭 매개변수를 사용하기도 하지만, 추천 ㄴㄴ
제너릭을 사용한 스타일은 타입 단언문과 기능적으로 동일하다.
제너릭보다는 unknown을 반환하고 사용자가 직접 단언문을 사용하거나 원하는 대로 타입을 좁히도록 강제하는 것이 좋다.

```
function parseYAML(yaml: string): any {
  // ...
}
function safeParseYAML<T>(yaml: string): T {
  return parseYAML(yaml);
}
```

3 . 단언문과 관련된 unknown

이중단언문에서 any 대신 unknown을 사용할 수 있다.
기능적으로론 똑같지만, 나중에 두 개의 단언문을 분리하는 리팩터링을 하면 unknown 형태가 더 안전하다.

```
function parseYAML(yaml: string): any {
  // ...
}
interface Foo { foo: string }
interface Bar { bar: string }
declare const foo: Foo;
let barAny = foo as any as Bar;
let barUnk = foo as unknown as Bar;
```

4 . unknown과 비슷하지만 다른 타입

{ } : null과 undefined를 제외한 모든 값을 포함
- 정말로 null과 undefined가 불가능하다고 판단되는 경우 사용

object : 모든 비기본형 타입(객체타입)으로 객체, 배열, 함수, 날짜, 정규 표현식을 포함

- 객체 : {[key: strung]:any}
- 배열 : any[ ]
- 함수 : Function / type Log = (message:string) => void
- 날짜 : Date
- 정규 표현식 : RegExp

{ } , object 모두 unknown 만큼 넓은 타입이지만, unknown보다는 범위가 약간 좁다.

요약

- **unknown은 any 대신 사용할 수 있는 안전한 타입이다.**
  어떠한 값이 있지만 그 타입을 알지 못한다면 unknown 사용하자.
- **사용자가 타입 단언문이나 타입 체크를 사용하도록 강제하려면 unknown을 사용하자.**
- { }, object, unknown의 차이점을 이해하자.