# 2024-04-03 5.6.5 Interface

현행 문서: 5.6.5 Interface



# 1 Interface

## 1.1 Type vs Interface

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

- 두 구문은 거의 같습니다. 하지만 인터페이스와 타입 별칭 사이에는 몇 가지 주요 차이점이 있습니다.
- 인터페이스는 속성 증가를 위해 병합할 수 있습니다. 이 기능은 내장된 전역 인터페이스 또는 npm 패키지와 같은 외부 코드를 사용할 때 특히 유용합니다.
- 인터페이스는 클래스가 선언된 구조의 타입을 확인하는 데 사용할 수 있지만 타입 별칭은 사용할 수 없습니다.
- 일반적으로 인터페이스에서 타입스크립트 타입 검사기가 더 빨리 작동합니다. 인터페이스는 타입 별칭 이 하는 것처럼 새로운 객체 리터럴의 동적인 복사 붙여넣기보다 내부적으로 더 쉽게 캐시할 수 있는 명명된 타입을 선언합니다.
- 인터페이스는 이름 없는 객체 리터럴의 별칭이 아닌 이름 있는(명명된) 객체로 간주되므로 어려운 특이 케이스에서 나타나는 오류 메시지를 좀 더 쉽게 읽을 수 있습니다.

## 1.2 속성 타입

### 1.2.1 선택적 속성

- ?를 사용해 선택적 속성을 표현 할 수 있습니다.

```
      interface Book{
    author?: string;
    pages: number;
}
```

### 1.2.2 읽기 전용 속성

- readonly 제한자는 타입 시스템에만 존재하며, 인터페이스에서만 사용할 수 있습니다.
- readonly 제한자는 객체의 인터페이스를 선언하는 위치에서만 사용되고 실제 객체에는 적용 되지 않습니다.

```
      interface Page {
    readonly text:string;
}

function read(page: Page) {
    console.log(page.text);

    page.text += "!";
}
```

blocked URL

## 1.3 함수와 메서드

- 메서드 구문: 인터페이스의 멤버를 member(): void 와 같이 객체의 멤버로 호출되는 함수로 선언
- 속성 구문: 인터페이스의 멤버를 member: () => void 와 같이 독립 함수와 동일하게 선언

```
      interface HasBothFunctionTypes {
    property: () =>  string;
    method(): string;
}

const hasBoth: HasBothFunctionTypes = {
    property: () => "",
    method() {
        return "";
    }
}

hasBoth.property();
hasBoth.method();
```

- 
```
      interface OptionalFunctionTypes {
    property?: () =>  string;
    method(): string;
}

const functionTypeSample: OptionalFunctionTypes = {
    property: () => "hello",
    method(): string {
        return "";
    }
}

functionTypeSample.property?.();
functionTypeSample.method();
```

- 메서드는 readonly로 선언 할 수 없지만 속성은 가능합니다.
- 기본 함수가 this를 참조할 수 있다는 것을 알고 있다면 메서드 함수를 사용하세요.
- 반대의 경우는 속성 함수를 사용하세요.

### 1.3.1 인터페이스 확장

- 인터페이스는 다른 인터페이스의 모든 멤버를 복사해서 선언할 수 있는 확장 인터페이스를 허용합니다.

```
      interface Writing {
    title: string;
}
interface Novella extends Writing {
    pages: number;
}

let myNovella: Novella = {
    pages: 195,
    title: "Ethan Frome",
};

let missingPages: Novella = {
    title: "The Awakening",
}

let extraProperty: Novella = {
    pages: 300,
    strategy: "baseline",
    style: "Naturalism",
}
```

blocked URL

## 1.4 인터페이스 병합

- 인터페이스의 중요한 특징 중 하나는 서로 병합하는 능력입니다.

```
      interface Merged {
    fromFirst: string;
}
interface Merged {
    fromSecond: string;
}

const sample: Merged = {
    fromFirst: "first",
    fromSecond: "second",
}
```

blocked URL

### 1.4.1 이름이 충돌되는 멤버

- 병합된 인터페이스는 타입이 다른 동일한 이름의 속성을 여러 번 선언할 수 없습니다.
- 속성 구문은 에러가 발생하지만, 메서드 구문은 두가지 오버로드 있는 different 메서드를 생성합니다.

```
      interface MergedProperties {
    same: (input: boolean) => string;
    different: (input: string) => string;
}

interface MergedProperties {
    same: (input: boolean) => string;
    different: (input: number) => string;
}

interface MergedMethods {
    different(input: string): string;
}

interface MergedMethods {
    different(input: number): string;
}
```