

# Item 11. 잉여속성 체크의 한계 인지하기

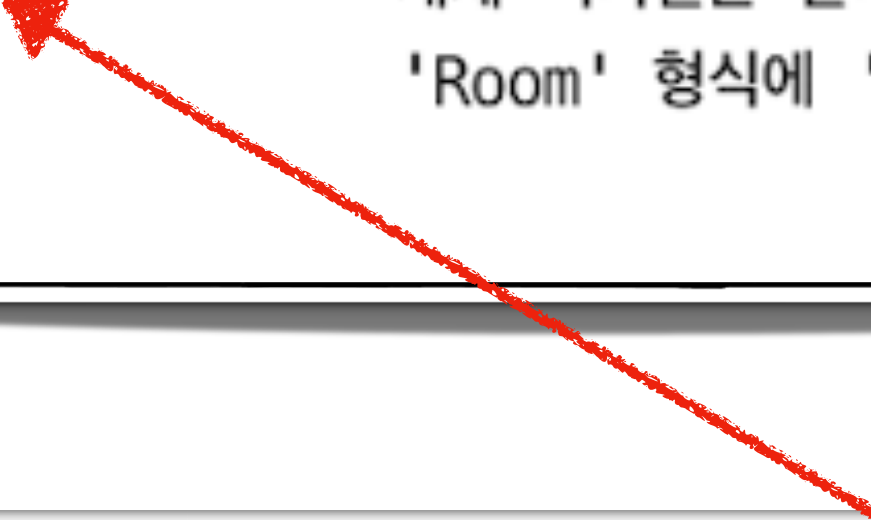
'23. 4. 30. (SUN) 신현호(@SWARVY)



```
interface Room {  
  numDoors: number;  
  ceilingHeightFt: number;  
}  
const r: Room = {  
  numDoors: 1,  
  ceilingHeightFt: 10,  
  elephant: 'present',  
// ~~~~~ 개체 리터럴은 알려진 속성만 지정할 수 있으며  
// 'Room' 형식에 'elephant'이(가) 없습니다.  
};
```

타입이 명시된 변수에 객체 리터럴을 할당할 때  
타입스크립트는 해당 타입이 속성이 있는지, 그리고 **'그 외의 속성은 없는지'** 확인해요

```
interface Room {  
  numDoors: number;  
  ceilingHeightFt: number;  
}  
const r: Room = {  
  numDoors: 1,  
  ceilingHeightFt: 10,  
  elephant: 'present',  
// ~~~~~ 개체 리터럴은 알려진 속성만 지정할 수 있으며  
// 'Room' 형식에 'elephant'이(가) 없습니다.  
};
```



해당 객체는 Room 이라는 타입에 elephant가 있는게 어색하지만,  
구조적 타이핑 관점으로 생각해보면 오류가 발생하지 않아야해요

```
const obj = {  
  numDoors: 1,  
  ceilingHeightFt: 10,  
  elephant: 'present',  
};  
const r: Room = obj; // 정상
```

임시변수를 통해 확인해본 결과, obj는 Room 타입에 할당이 가능해요  
왜냐? obj타입은 Room 타입의 부분 집합을 포함하기 때문이에요

```
interface Room {  
  numDoors: number;  
  ceilingHeightFt: number;  
}  
const r: Room = {  
  numDoors: 1,  
  ceilingHeightFt: 10,  
  elephant: 'present',  
// ~~~~~ 개체 리터럴은 알려진 속성만 지정할 수 있으며  
//           'Room' 형식에 'elephant'이(가) 없습니다.  
};
```

```
const obj = {  
  numDoors: 1,  
  ceilingHeightFt: 10,  
  elephant: 'present',  
};  
const r: Room = obj; // 정상
```

앞 두 예제의 차이점은 뭡까요?

```
interface Room {
  numDoors: number;
  ceilingHeightFt: number;
}
const r: Room = {
  numDoors: 1,
  ceilingHeightFt: 10,
  elephant: 'present',
// ~~~~~ 개체 리터럴은 알려진 속성만 지정할 수 있으며
//           'Room' 형식에 'elephant'이(가) 없습니다.
};
```

```
const obj = {
  numDoors: 1,
  ceilingHeightFt: 10,
  elephant: 'present',
};
const r: Room = obj; // 정상
```

첫 번째 예제에서는 구조적 타입 시스템에서 발생할 수 있는 중요한 종류의 오류를 잡을 수 있도록 '잉여 속성 체크'라는 과정이 수행되었어요.

```
interface Room {
  numDoors: number;
  ceilingHeightFt: number;
}
const r: Room = {
  numDoors: 1,
  ceilingHeightFt: 10,
  elephant: 'present',
// ~~~~~ 개체 리터럴은 알려진 속성만 지정할 수 있으며
// 'Room' 형식에 'elephant'이(가) 없습니다.
};
```

```
const obj = {
  numDoors: 1,
  ceilingHeightFt: 10,
  elephant: 'present',
};
const r: Room = obj; // 정상
```

그러나 잉여 속성 체크 역시 조건에 따라 동작하지 않을 수도 있고,  
통상적인 할당 검사와 함께 쓰이면 구조적 타이핑이 무엇인지 혼란스러워 질 수 있어요



잉여 속성 체크



할당 가능 검사


잉여 속성 체크가 할당 검사와는 별도의 과정이라는 것을 알아두세요!



```
interface Options {  
  title: string;  
  darkMode?: boolean;  
}  
function createWindow(options: Options) {  
  if (options.darkMode) {  
    setDarkMode();  
  }  
  // ...  
}  
createWindow({  
  title: 'Spider Solitaire',  
  darkmode: true  
// ~~~~~ 개체 리터럴은 알려진 속성만 지정할 수 있지만  
//           'Options' 형식에 'darkmode'이(가) 없습니다.  
//           'darkMode'을(를) 쓰려고 했습니까?  
});
```

타입스크립트는 코드의 오류를 표시하는 것 뿐만 아니라,  
의도와 다르게 작성된 코드까지 찾아줘요

```
interface Options {  
  title: string;  
  darkMode?: boolean;  
}  
  
function createWindow(options: Options) {  
  if (options.darkMode) {  
    setDarkMode();  
  }  
  // ...  
}  
  
createWindow({  
  title: 'Spider Solitaire',  
  darkmode: true  
// ~~~~~ 개체 리터럴은 알려진 속성만 지정할 수 있지만  
//           'Options' 형식에 'darkmode'이(가) 없습니다.  
//           'darkMode'을(를) 쓰려고 했습니까?  
});
```



앞의 코드를 실행하면 런타임에 어떠한 종류의 오류도 발생하지 않아요.  
하지만 타입스크립트가 말해주는 것 처럼 의도한 대로 동작하지 않을 수 있어요!

```
interface Options {
  title: string;
  darkMode?: boolean;
}
function createWindow(options: Options) {
  if (options.darkMode) {
    setDarkMode();
  }
  // ...
}
createWindow({
  title: 'Spider Solitaire',
  darkmode: true
// ~~~~~ 개체 리터럴은 알려진 속성만 지정할 수 있지만
//           'Options' 형식에 'darkmode'이(가) 없습니다.
//           'darkMode'을(를) 쓰려고 했습니까?
});
```

```
const o1: Options = document;           // 정상
const o2: Options = new HTMLAnchorElement; // 정상
```

Option에 할당 가능한 값이 여러개 더 존재하는데  
document와 HTMLAnchorElement의 인스턴스 모두 string 타입의 title을 가져서 할당이 정상적으로 이루어져요

## 타입 구문 없는 임시변수 예제

```
const o: Options = { darkmode: true, title: 'Ski Free' };  
// ~~~~~ 'Options' 형식에 'darkmode'이(가) 없습니다.
```

```
const intermediate = { darkmode: true, title: 'Ski Free' };  
const o: Options = intermediate; // 정상
```

```
const o = { darkmode: true, title: 'Ski Free' } as Options; // 정상
```



첫 번째 줄은 객체 리터럴, 두 번째 줄은 객체 리터럴이 아님,  
따라서 잉여 속성 체크가 적용되지 않아서 오류가 사라져요



잉여 속성 체크는 타입 단언문을 사용할 때도 적용되지 않아요

잉여 속성 체크는...

타입 시스템의 구조적 본질을 해치지 않으면서 객체 리터럴에 알 수 없는 속성을 허용하지 않아요  
(앞에 나왔던 document, new HTMLAnchorElement 는 객체 리터럴이 아니에요)

그래서 잉여 속성 체크를 '엄격한 객체 리터럴 체크' 라고도 부릅니다