Effective TypeScript

string 타입보다 더 구체적인 타입 사용하기

Item 33

강철원 Ryan-Dia

아래 코드의 interface를 만들어본다면?

```
const kindOfBlue = {
  가수: 'Miles Davis',
  제목: 'Kind of Blue',
  발매일: 'August 17th, 1959',
  녹음유형: 'Studio',
};
```

혹시 이렇게?

```
interface 앨범 {
    가수 : string;
    제목 : string;
    발매일 : string;
    녹음유형 : string;
}
```

조금 더 세분화

```
type 녹음유형 = 'studio' | 'live';

interface 앨범 {
  가수 : string;
  제목 : string;
  발매일 : Date;
  녹음_유형 : 녹음유형;
}
```

이런식으로 더 구체적으로 타입을 정의해주면 어떠한 점들이 좋을까?

1) 타입을 명시적으로 정의함으로써 다른 곳으로 값이 전달되어도 **타입 정보가 유지**된다.

2) 타입을 명시적으로 정의하고 해당 타입의 의미를 설명하는 **주석**을 붙여넣을 수 있다.

3) keyof 연산자로 더욱 **세밀하게 객체의 속성 체크**가 가능해진다.

다른 곳으로 값이 전달되어도 **타입 정보가 유지** 및 타입의 의미를 설명하는 **주석**을 붙여넣을 수 있다.

```ts
/**이 녹음은 어떤 환경에서 이루어졌는지? */
type 녹음유형 = 'Studio' | 'live';
```

```ts
function 녹음_유형에_따라_앨범_분류(recordingType: string): 앨범[] {
  // ...........
  // ..........
  return []
}
```

```ts
function 녹음_유형에_따라_앨범_분류(recordingType: 녹음유형): 앨범[] {
  // ...........
  // ..........
  return []
}
```

```ts
type 녹음유형 = "live" | "studio"
```

이 녹음은 어떤 환경에서 이루어졌는지?

```ts
ecordingType: 녹음유형): 앨범[] {
```

1) 타입을 명시적으로 정의함으로써 다른 곳으로 값이 전달되어도 **타입 정보가 유지**된다.

2) 타입을 명시적으로 정의하고 해당 타입의 의미를 설명하는 **주석**을 붙여넣을 수 있다.

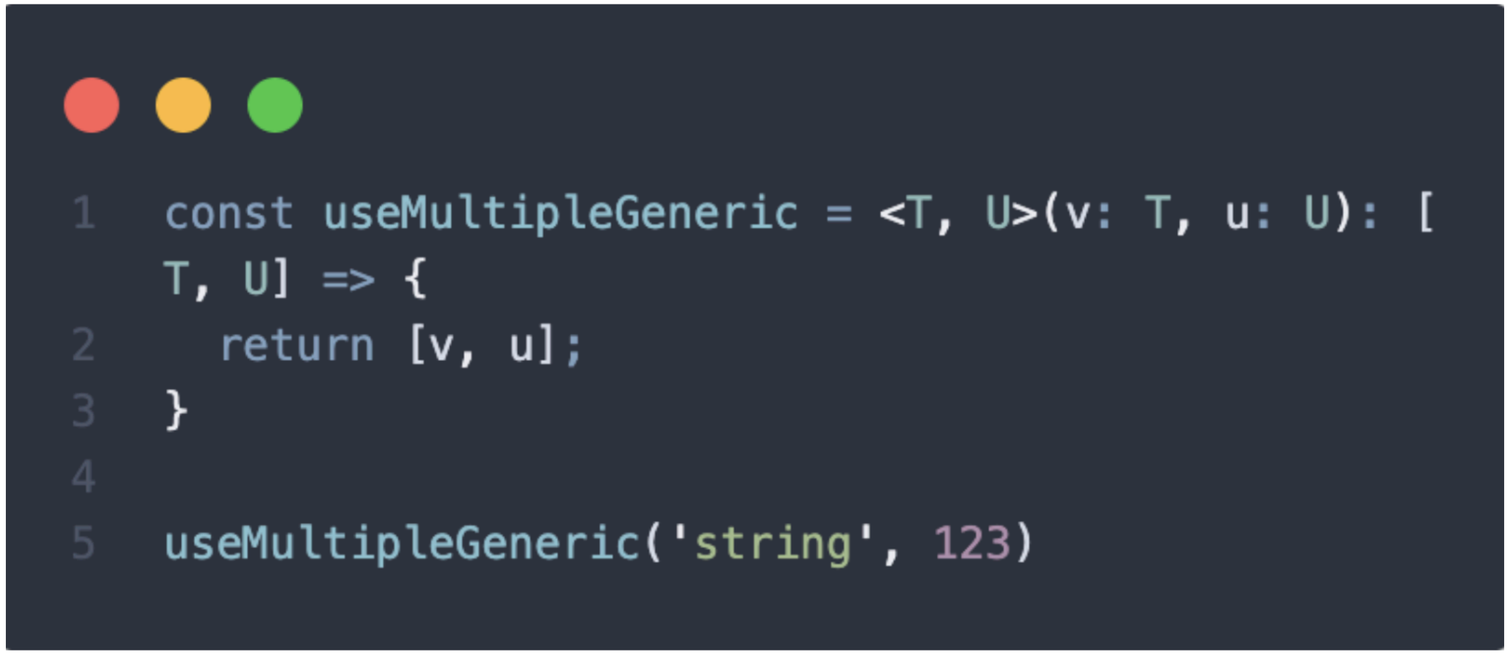
3) keyof 연산자로 더욱 **세밀하게 객체의 속성 체크**가 가능해진다.

keyof 복습

```ts
/** 이 녹음은 어떤 환경에서 이루어졌는지? */
type 녹음유형 = 'studio' | 'live';

interface 앨범 {
  가수 : string;
  제목 : string;
  발매일 : Date;
  녹음_유형 : 녹음유형;
}

type test = keyof 앨범    // 타입은  "가수" | "제목" | "발매일" | "녹음_유형"
```

제네릭 복습

- Generic이란 데이터의 타입을 일반화하다(generalize)한다는 것을 뜻한다.
- Generic은 자료형을 정하지 않고 여러 타입을 사용할 수 있게 해준다.
- 즉, 선언 시점이 아니라 생성 시점에 타입을 명시하여 하나의 타입만이 아닌 다양한 타입을 사용할 수 있도록 하는 기법이다.
- 한번의 선언으로 다양한 타입에 "**재사용**"이 가능하다는 장점이 있다.



```ts
type 녹음유형 = 'Studio' | 'live';
interface 앨범 {
  가수 : string;
  제목 : string;
  발매년도 : number;
  녹음_유형 : 녹음유형;
}

const kindOfBlue = {
  가수: 'Miles Davis',
  제목: 'Kind of Blue',
  발매년도: 1999,
  녹음_유형: 'Studio',
};

const E = {
  가수 : "BIGBANG",
  제목 : "우리 사랑하지 말아요",
  발매년도: 2015,
  녹음_유형: "live"
}

const 조각집 = {
  가수 : "아이유",
  제목 : "드라마",
  발매년도 : 2021,
  녹음_유형: "Studio"
}

const albums = [kindOfBlue, E, 조각집]

function 원하는_key에_대한_value추출(albums, key) {
  return albums.map(album => album[key])
}

console.log(원하는_key에_대한_value추출(albums, "제목"))
```

```
[ 'Kind of Blue', '우리 사랑하지 말아요', '드라마' ]
```

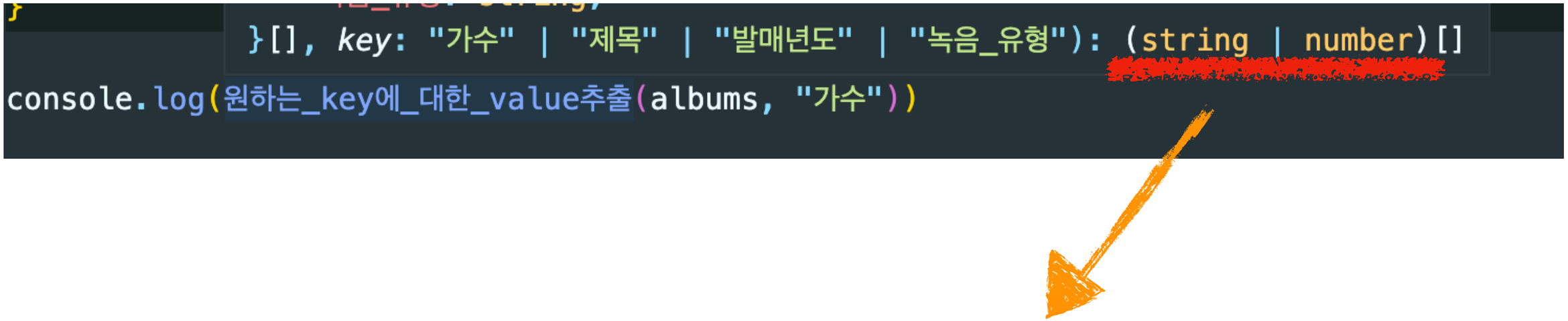
```ts
const kindOfBlue : 앨범 = {
  가수: 'Miles Davis',
  제목: 'Kind of Blue',
  발매년도: 1999,
  녹음_유형: 'Studio',
};

const E : 앨범 = {
  가수 : "BIGBANG",
  제목 : "우리 사랑하지 말아요",
  발매년도: 2015,
  녹음_유형: "live"
}

const 조각집 : 앨범 = {
  가수 : "아이유",
  제목 : "드라마",
  발매년도 : 2021,
  녹음_유형: "Studio"
}

const albums: 앨범[] = [kindOfBlue, E, 조각집]
```

or

```ts
const albums = [kindOfBlue, E, 조각집] as 앨범[]
```

혹시 이렇게 하신분???

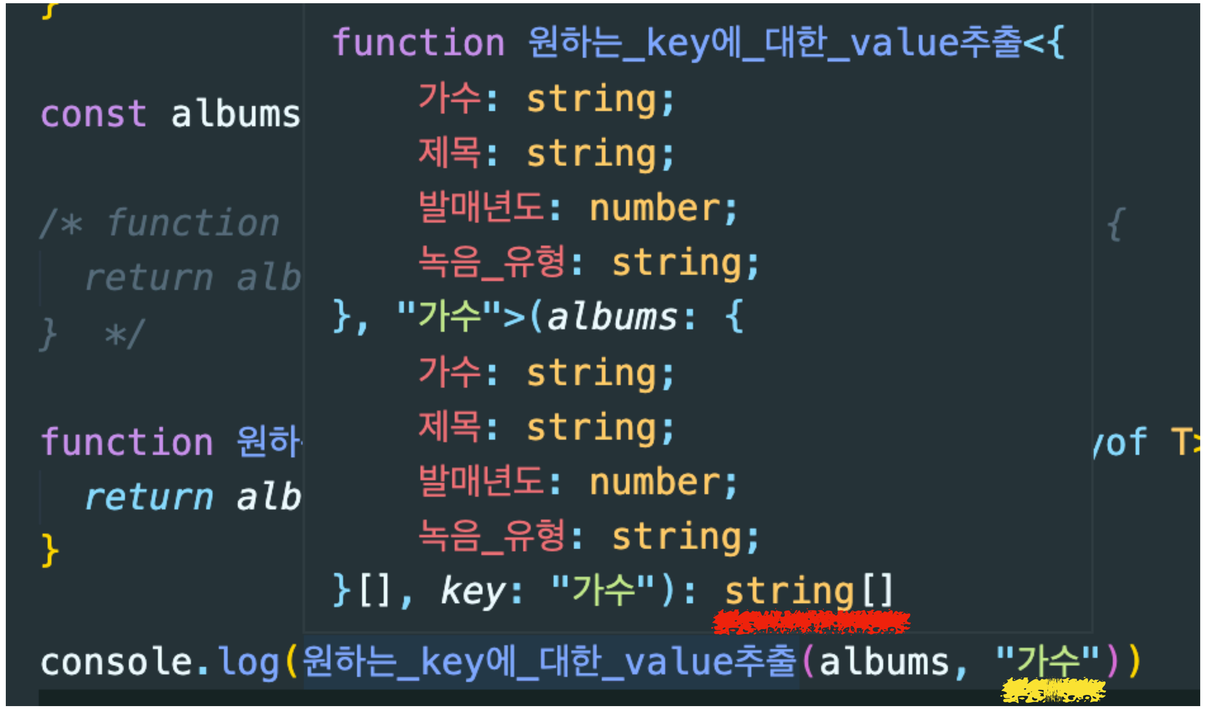
```
function 원하는_key에_대한_value추출<T>(albums : T[] , key: keyof T ): T[keyof T][] {
  return albums.map(album => album[key])
}

console.log(원하는_key에_대한_value추출(albums, "제목"))
```

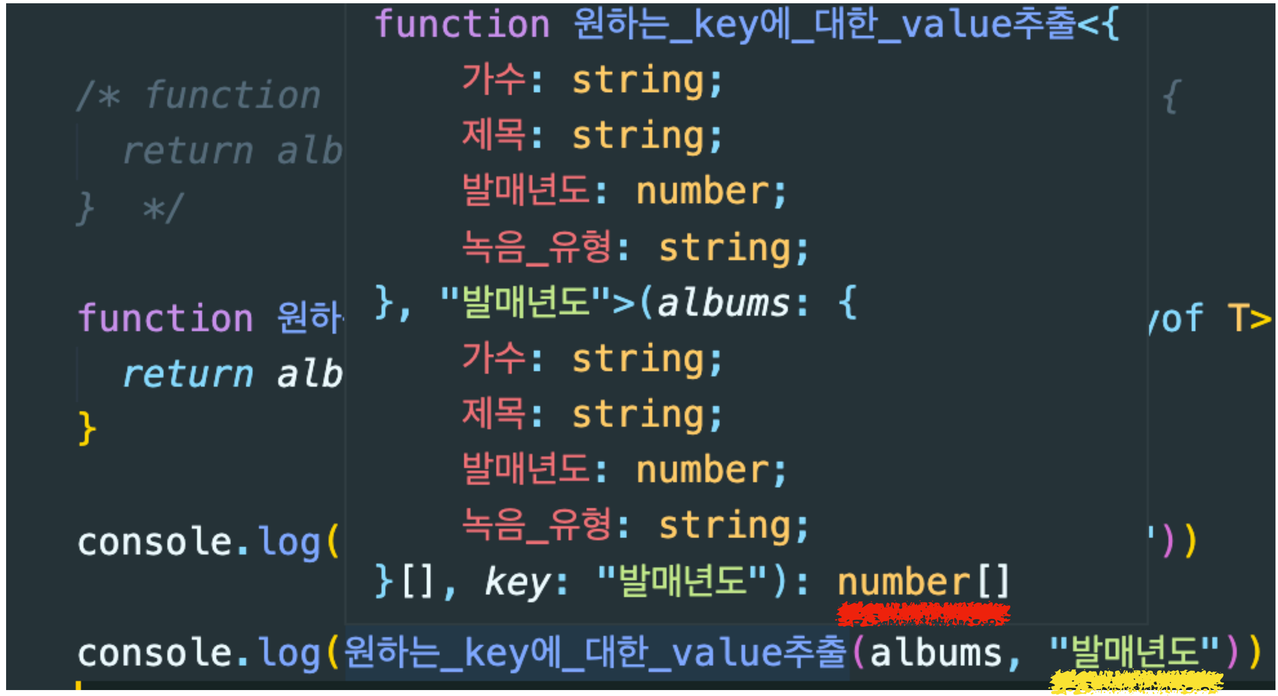


"가수"는 string 밖에없는데 (string | number)[ ] 으로 타입이 추론되고있다.

(string | number)[ ] 이 아니라 string[ ] 으로 추론되어야 한다.

```ts
function 원하는_key에_대한_value추출<T, K extends keyof T>(albums : T[] , key: K): T[K][] {
  return albums.map(album => album[key])
}
```





keyof T 를 하였을 때 장점

```
function 원하는_key에_대한_value추출<T, K extends keyof T>(albums : T[] , key: K): T[K][] {
  return albums.map(album => album[key])
}
```

```
console.log(원하는_key에_대한_value추출(albums, ""))
```

가수 가수
녹음_유형
발매년도
제목

타입을 string으로 했으면 이렇게 자동완성이 뜨지 않는다.

타입을 잘못 사용하게 되면 무효한 값을 허용하고 타입 간의 관계를 감추어 버린다.

이러한 문제점은 타입 체커를 방해하고 실제 버그를 찾지 못하게 만든다.

TS에서 string의 부분 집합을 정의할 수 있는 기능은

자바스크립트 코드에 타입 안정성을 크게 높인다

보다 정확한 타입을 사용하면 오류를 방지하고 코드의 가독성도 향상시킬 수 있다.

- 문자열을 남발하여 선언된 코드를 피하고자 모든 문자열을 할당할 수 있는 string 타입보다는 **더 구체적인 타입을 사용하는 것이 좋다.**

- 변수의 범위를 보다 정확하게 표현하고 싶다면 string타입보다는 **문자열 리터럴 타입의 유니온을 사용하면 된다.**

- 객체의 속성 이름을 함수 매개변수로 받을 때는 string 보다는 **keyof T를 사용하는 것이 좋다.**