




Item 50, 51

Edited 2023. 03. 16. (THU) 신현호(@SWARVY)

Item. 50

오버로딩 타임보다는 조건부 타임 사용하기



원형 함수

```
function double(x) {  
  return x + x;  
}
```

```
function double(x: number|string): number|string;  
function double(x: any) { return x + x; }
```

function double에는 number 혹은 string이 들어갈 수 있어요
(함수 왜 저렇게 선언하는지 헷갈린다면 item 3 다시보고오기)

```
const num = double(12); // string | number  
const str = double('x'); // string | number
```

선언이 틀린건 아닌데 모호해요

이상합니다..!

```
const num = double(12);    // string | number  
const str = double('x');  // string | number
```

1번의 경우에는 number타입을 반환하는건 맞는데 string을 반환하는 경우도 포함되어있고
2번의 경우에는 string타입을 반환하는건 맞지만 number를 반환하는 경우도 포함되어있어요

제네릭을 사용하면 이러한 모델링이 가능해요

```
function double<T extends number|string>(x: T): T;  
function double(x: any) { return x + x; }
```


```
const num = double(12);    // 타입이 12  
const str = double('x');  // 타입이 "x"
```

근데 이건 너무 과하게 구체적이에요.

또 다른 방법으로는 여러가지 타입 선언으로 분리하는 것도 있어요

```
function double(x: number): number;  
function double(x: string): string;  
function double(x: any) { return x + x; }  
  
const num = double(12); // 타입이 number  
const str = double('x'); // 타입이 string
```


```
function f(x: number|string) {  
    return double(x);  
    // ~ 'string | number' 형식의 인수는  
    // 'string' 형식의 매개변수에 할당될 수 없습니다.  
}
```



조금 명확해지긴 했는데 여전히 버그는 남아있습니다.
string이나 number로는 잘 동작하는데 유니온 타입 관련해서 문제가 생깁니다

제일 좋은 방법은 조건부 타입을 사용하는거예요

```
function double<T extends number | string>(
  x: T
): T extends string ? string : number;
function double(x: any) { return x + x; }
```



이건 제네릭 사용한거랑 비슷해보이지만 반환타입이 더 정교해요
조건부타입은 자바스크립트의 삼항연산자처럼 사용하면 됩니다.

Item. 51

의존성 분리를 위해 미리 타임 사용하기

CSV 파일을 파싱하는 라이브러리를 만든다고 가정해볼게요.

```
function parseCSV(contents: string | Buffer): {[column: string]: string}[] {  
  if (typeof contents === 'object') {  
    // 버퍼인 경우  
    return parseCSV(contents.toString('utf8'));  
  }  
  // ...  
}
```

\$ npm install --save-dev @types/node

내용이 중요하다기 보단 Buffer 타입을 허용했다는거에 주목하시면 됩니다.
Buffer의 타입 정의는 NodeJS 타입 선언을 설치해서 얻을 수 있어요.

```
function parseCSV(contents: string | Buffer): {[column: string]: string}[] {  
  if (typeof contents === 'object') {  
    // 버퍼인 경우  
    return parseCSV(contents.toString('utf8'));  
  }  
  // ...  
}
```

해당 라이브러리를 공개하면 타입 선언도 포함하게 되는데,
이 타입선언이 @types/node에 의존하기때문에 @types/node는 devDependencies로 포함해야해요

근데, 이러면 문제가 생깁니다



- @types와 무관한 자바스크립트 개발자
- NodeJS와 무관한 타입스크립트 웹 개발자

두 그룹의 사용자들은 각자가 사용하지 않은 모듈이 포함되어있어서 혼란스러워해요.

Buffer는 NodeJS 개발자만 필요하고, @types/node는 NodeJS와 타입스크립트를 동시에 쓰는 사람만 필요하거든요

이런 경우에는 필요한 메서드와 속성만 별도로 작성하는걸 추천해요

```
interface CsvBuffer {  
  toString(encoding: string): string;  
}  
function parseCSV(contents: string | CsvBuffer): {[column: string]: string}[] {  
  // ...  
}
```

위의 예제는 인코딩 정보를 매개변수로 받는 toString 메서드를 가지는 인터페이스를 별도로 만들어 사용하면 됩니다

작성 중인 라이브러리가 의존하는 라이브러리의 구현과 무관하게 타입에만 의존한다면,
필요한 선언부만 추출하여 작성 중인 라이브러리에 넣는 것을 고려해보는게 좋아요

필수가 아닌 의존성을 분리할 때는 구조적 타이핑을 사용하기

공개한 라이브러리를 사용하는 자바스크립트 사용자가 @types 의존성을 가지지 않게 하기,
웹 개발자가 NodeJS 관련된 의존성을 가지지 않게 하기