함수 안으로 타입 단언문 감추기

Item 40

강철원(ryan-dia)

함수로 작성하다보면,

외부로 드러난 타입 정의는 간단하지만

내부 로직이 복잡해서 안전한 타입으로 구현하기 어려운 경우가 많습니다.

함수의 모든 부분을 안전한 타입으로 구현하는 것이 이상적이지만,

불필요한 예외 상황가지 고려해 가며 타입 정보를 힘들게 구성할 필요는 없습니다.

함수 내부에는 타입단언을 사용하고
함수 외부로 드러나는 타입 정의를 정확히 명시하는 정도로 끝내는 게 낫습니다.

프로젝트 전반에 위험한 타입 단언문이 드러나 있는 것보다
제대로 타입이 정의된 함수 안을 타입 단언문을 감추는 것이 더 좋은 설계 입니다.

```ts
declare function shallowEqual(a: any, b: any): boolean;
function cacheLast<T extends Function>(fn: T): T {
  let lastArgs: any[] | null = null;
  let lastResult: any;
  return function (...args: any[]) {
    // ~~~~~~~~~~~~~~~~~~~~~~~~~~
    //          Type '(...args: any[]) => any' is not assignable to type 'T'
    if (!lastArgs || !shallowEqual(lastArgs, args)) {
      lastResult = fn(...args);
      lastArgs = args;
    }
    return lastResult;
  };
}
```

```ts
function add(a: number, b: number): number {
    return a + b;
  }

  const cachedAdd = cacheLast(add);
  const result = cachedAdd(1, 2);
```

```ts
declare function shallowEqual(a: any, b: any): boolean;
function cacheLast<T extends Function>(fn: T): T {
  let lastArgs: any[]|null = null;
  let lastResult: any;
  return function(...args: any[]) {
    if (!lastArgs || !shallowEqual(lastArgs, args)) {
      lastResult = fn(...args);
      lastArgs = args;
    }
    return lastResult;
  } as unknown as T;
}
```

예시 1

왜 바로 as T로 하지 않았을 까?

Double assertion

```
let 나문자임: string = 'hi';

function 숫자클럽(숫자만오삼: number) {
  console.log(숫자만오삼);
}
숫자클럽(나문자임 as number)
숫자클럽(나문자임 as unknown as number);
```


```
let 나문자임: string = 'hi';

function 숫자클럽(숫자만오삼: number) {
  console.log(숫자만오삼);
}
숫자클럽(나문자임 as number)
숫자클럽(나문자임 as unknown as number);
```

예시 2

```
function shallowObjectEqual(objectA, objectB){
  for (const [key, value] of Object.entries(objectA)) {
      const isA객체의key가B객체의key에없어 =  !(key in objectB)
      const isA객체의key에해당하는value가똑같이B에없어 = value !== objectB[key]
    if (isA객체의key가B객체의key에없어 || isA객체의key에해당하는value가똑같이B에없어) return false;
  }
  const is두객체의길이가같다 = Object.keys(a).length === Object.keys(b).length;
  return is두객체의길이가같다
}
```

```
function shallowObjectEqual<T extends object>(objectA: T, objectB: T): boolean {
  for (const [key, value] of Object.entries(objectA)) {
    if (!(key in objectB) || value !== objectB[key]) {
      return false;
    }
  }
  return Object.keys(objectA).length === Object.keys(objectB).length;
}

```

```
function shallowObjectEqual<T extends object>(objectA: T, objectB: T): boolean {
  for (const [key, value] of Object.entries(objectA)) {
    if (!(key in objectB) || value !== objectB[key]) {
      // ~~~~ Element implicitly has an 'any' type
      //      because type '{}' has no index signature
      return false;
    }
  }
  return Object.keys(objectA).length === Object.keys(objectB).length;
}
```

```ts
function shallowObjectEqual<T extends object>(objectA: T, objectB: T): boolean {
  for (const [key, value] of Object.entries(objectA)) {
    if (!(key in objectB) || value !== objectB[key]) return false;
  }
  return Object.keys(objectA).length === Object.keys(objectB).length;
}
```

'{}' 형식에 인덱스 시그니처가 없으므로
요소에 암시적으로 'any'형식이 있습니다.

타입스크립트의 문맥 활용 능력이 부족

```ts
function shallowObjectEqual<T extends object>(objectA: T, objectB: T): boolean {
  for (const [key, value] of Object.entries(objectA)) {
    if (!(key in objectB) || value !== (objectB as any)[key]) return false;
  }
  return Object.keys(objectA).length === Object.keys(objectB).length;
}

```

실제 오류가 아니라는 것을 알고 있기 때문에 any로 단언하기

```ts
function shallowObjectEqual<T extends object>(objectA: T, objectB: T): boolean {
  for (const [key, value] of Object.entries(objectA)) {
    if (!(key in objectB) || value !== objectB[key as keyof typeof objectB]) return false;
  }
  return Object.keys(objectA).length === Object.keys(objectB).length;
}
```

마무리

📌 타입 선언문은 일반적으로 타입을 위험하게 만들지만
상황에 따라 필요하기도 하고 현실적인 해결책이 되기도 합니다.

📌 불가피하게 사용해야 한다면, 정확한 정의를 가지는 함수 안으로 숨기는 것이 좋습니다.