Type Assertions

Summary: in this tutorial, you will learn about type assertions in TypeScript, which allows you to assign a new type to a value.

Type assertions instruct the TypeScript compiler to treat a value as a specified type. In TypeScript, you can use the `as` keyword or `<>` operator for type assertions.

Type assertions using the as keyword

The following selects the first input element on an HTML document using the `querySelector()` method:

```
let el = document.querySelector('input["type="text"]');
```

Since the returned type of the `document.querySelector()` method is the `Element` type, the following code causes a compile-time error:

```
console.log(el.value);
```

Error:

```
Property 'value' does not exist on type 'Element'.
```

The reason is that the value property doesn't exist in the `Element` type. It only exists on the `HTMLInputElement` type.

To resolve this, you can instruct the TypeScript compiler to treat the type of the `el` element as `HTMLInputElement` by using the `as` keyword like this:

```
const el = document.querySelector('input[type="text"]');
const input = el as HTMLInputElement;
```

Now, the `input` variable has the type `HTMLInputElement`. So accessing its value property won't cause any error. The following code works:

```
console.log(input.value);
```

Another way to assign the type `HTMLInputElement` to an `Element` object is when you access the property as follows:

```
let enteredText = (el as HTMLInputElement).value;
```

Note that the `HTMLInputElement` type extends the `HTMLElement` type that extends to the `Element` type.

The syntax for type assertion of a variable from `typeA` to `typeB` is as follows:

```
let a: typeA;
let b = a as typeB;
```

Type assertion using the <> operator

Besides the `as` keyword, you can use the `<>` operator to perform a type assertion. For example:

```
let input = <HTMLInputElement>document.querySelector('input[type="text"]');

console.log(input.value);
```

The syntax for type assertion using the `<>` operator is as follows:

```
let a: typeA;
let b = <typeB>a
```

Type assertion result

If a type assertion fails, different kinds of errors will occur depending on how you use type assertion and actual runtime types.

1) Compile-time errors

When you try to perform a type assertion between incompatible types, the TypeScript compiler may give you an error or warning. For example:

```
let price = '9.99';
let netPrice = price as number; // error
```

In this example, we attempt to assign the number type to a string, the TypeScript compiler issues the following compile-time error:

```
Conversion of type 'string' to type 'number' may be a mistake because neither type suffic
```

2) Runtime errors

When you perform a type assertion of an object to a type that doesn't match its structure and attempt to access a property that doesn't exist, you'll get a runtime error. For example:

```
let el = document.querySelector('#name');
let input = el as HTMLInputElement;
console.log(input.value.length);
```

In this example, if the element with id `#name` is not an input element, the `input.value` will be `undefined` at runtime. Hence, accessing the `length` property of the value will cause a runtime error:

```
TypeError: Cannot read properties of undefined (reading 'length')
```

3) Unexpected behaviors

If a type assertion is incorrect, you may not get a compile-time or runtime error but might experience unexpected behaviors later in your code. This can make debugging challenging because the error might not occur at the point of the type assertion.

Summary

- Type assertion allows you to assign a new type to a value.
- Use the `as` keyword or `<>` operator to perform a type assertion.