TypeScript never Type

Summary: in this tutorial, you will learn about the TypeScript `never` type to represent a value that never occurs.

Introduction to the TypeScript never type

In TypeScript, a type is like a set of values. For example, the `number` type holds the numbers 1, 2, 3, etc. The `string` type holds the strings like `'Hi'`, `'Hello'`, etc. The `null` type holds a single value, which is `null`.

The `never` type is a type that holds **no value**. It is like an **empty set**.

Since a `never` type does not hold any value, you cannot assign a value to a variable with the `never` type.

For example, the following will result in an error:

```
let empty: never = 'hello';
```

The TypeScript compiler issues the following error:

```
Type 'string' is not assignable to type 'never'
```

So why do we need the `never` type in the first place?

Since the `never` type has zero value, you can use it to denote an **impossibility** in the type system.

For example, you may have an intersection type that can be both a string and a number at the same time, which is impossible:

```
type Alphanumeric = string & number; // never
```

Therefore, the TypeScript compiler infers the type of `Alphanumeric` as `never`.

This is because `string` and `number` are mutually exclusive. In other words, a value cannot be both a `string` and a `number` simultaneously.

Typically, you use the `never` type to represent the return type of a function that never returns the control to the caller. For example, a function that always throws an error:

```
function raiseError(message: string): never {
    throw new Error(message);
}
```

> Please do not confuse with functions that return void but still return the control to the caller.

If you have a function that contains an indefinite loop, its return type should be `never`. For example:

```
function forever(): never {
  while (true) {}
}
```

In this example, the type of the return type of the `forever()` function is `never`.

The TypeScript never example

Let's take an example of using the `never` type:

```
type Role = 'admin' | 'user';

const authorize = (role: Role): string => {
  switch (role) {
```

```
    case 'admin':
      return 'You can do anything';
    case 'user':
      return 'You can do something';
    default:
      // never reach here util we add a new role
      const _unreachable: never = role;
      throw new Error(`Invalid role: ${_unreachable}`);
  }
};

console.log(authorize('admin'));
```

How it works.

Step 1. Define a type `Role` that can be either a string `'admin'` or `'user'`:

```
type Role = 'admin' | 'user';
```

Step 2. Create the `authorize()` function that accepts a value of the `Role` type and returns a string:

```
const authorize = (role: Role): string => {
  switch (role) {
    case 'admin':
      return 'You can do anything';
    case 'user':
      return 'You can do something';
    default:
      // never reach here util we add a new role
      const _unreachable: never = role;
      throw new Error(`Invalid role: ${_unreachable}`);
  }
};
```

How it works.

First, use the switch statement to return a corresponding string if the role is `admin` or `user`.

Second, define a variable called `_unreachable` with the type `never` and assign the `role` to it. Also, throw an error in the `default` branch because the execution will never reach the `default` branch.

Why do we handle the `default` case?

The reason is that if we add a new value to the `Role` type and forget to add a logic to handle the new role, TypeScript will issue an error:

```
type Role = 'admin' | 'user' | 'guest';
```

In this case, we add the `'guest'` to the `Role` type.

And TypeScript issues the following error:

```
Type 'string' is not assignable to type 'never'.ts(2322)
```

This is because the value of the role in the `default` branch now becomes the string `'guest'` and you cannot assign a string value to a variable with the type `never`.

To fix this, you need to create a new `case` branch to handle the new role:

```
const authorize = (role: Role): string => {
  switch (role) {
    case 'admin':
      return 'You can do anything';
    case 'user':
      return 'You can do something';
    case 'guest':
      return 'You can do nothing';
    default:
      // never reach here util we add a new role
      const _unreachable: never = role;
      throw new Error(`Invalid role: ${_unreachable}`);
  }
};
```

To make it more concise, we can define a function with the return type never and use it in the `default` branch:

```ts
type Role = 'admin' | 'user' | 'guest';

const unknownRole = (role: never): never => {
  throw new Error(`Invalid role: ${role}`);
};

const authorize = (role: Role): string => {
  switch (role) {
    case 'admin':
      return 'You can do anything';
    case 'user':
      return 'You can do something';
    case 'guest':
      return 'You can do nothing';
    default:
      // never reach here util we add a new role
      return unknownRole(role);
  }
};

console.log(authorize('admin'));
```

Summary

- Use the `never` type that holds no value, denoting an impossibility in the type system.