TypeScript Generic Interfaces

Summary: in this tutorial, you will learn how to develop TypeScript generic interfaces to create an interface that can work with different types while maintaining type safety.

Introduction to TypeScript generic interfaces

Like classes, interfaces can also be generic. A generic interface allows you to create an interface that can work with different types while maintaining type safety.

A generic interface has a generic type parameter list in angle brackets `<>` following the name of the interface:

```
interface interfaceName<T> {
    // ...
}
```

This makes the type parameter `T` visible to all members of the interface.

The type parameter list can have one or multiple types. For example:

```
interface interfaceName<U,V> {
    // ...
}
```

TypeScript generic interface examples

Let's take some examples of declaring generic interfaces.

1) Generic interfaces that describe object properties

The following shows how to declare a generic interface that consists of two members key and value with the corresponding types `K` and `V` :

```
interface Pair<K, V> {
    key: K;
    value: V;
}
```

Now, you can use the `Pair` interface to define any key/value pair with any type. For example:

```
let month: Pair<string, number> = {
    key: 'Jan',
    value: 1
};

console.log(month);
```

In this example, we declare a month key-value pair whose key is a string and the value is a number.

2) Generic interfaces that describe methods

The following declares a generic interface with two methods add() and remove():

```
interface Collection<T> {
    add(o: T): void;
    remove(o: T): void;
}
```

And this `List<T>` generic class implements the `Collection<T>` generic interface:

```
class List<T> implements Collection<T>{
    private items: T[] = [];

    add(o: T): void {
        this.items.push(o);
    }
```

```
    remove(o: T): void {
        let index = this.items.indexOf(o);
        if (index > -1) {
            this.items.splice(index, 1);
        }
    }
}
```

From the `List<T>` class, you can create a list of values of various types e.g., numbers, or strings.

For example, the following shows how to use the `List<T>` generic class to create a list of numbers:

```
let list = new List<number>();

for (let i = 0; i < 10; i++) {
    list.add(i);
}
```

3) Generic interfaces that describe index types

The following declares an interface that describes an index type:

```
interface Options<T> {
    [name: string]: T
}

let inputOptions: Options<boolean> = {
    'disabled': false,
    'visible': true
};
```

In this tutorial, you have learned about the TypeScript generic interfaces.