**Decorators**

A Decorator is a special kind of declaration attached to a class declaration, method, accessor, property, or parameter. Decorators are defined in the @expression format, where expression evaluates a function that called at runtime.

For the decorator @frozen, we write the frozen function as follows:

function frozen(target) {

// do something with 'target' ...

}

In TypeScript, we need to enable experimental support for decorators. In the tsconfig.json file, we set the **experimentalDecorators** compiler option property to true, in order to use decorators in our code.

**Example:** tsconfig.json

{

"compilerOptions": {

"target": "ES6",

"experimentalDecorators": true

}

}

Note: By running tsc --init command, the Typescript compiler automatically creates a tsconfig.json file in our working directory.

**Types of Decorators**

There are 4 different types of decorators:

1. Class decorators
2. Method decorators
3. Property decorators
4. Parameter decorators

**Class Decorators**

A Class Decorator is declared before a class declaration that applied to the **constructor of the class** and is used to observe, modify, or replace a class definition.

We can define the @frozen decorator using the following function declaration:

function frozen(constructor: Function) {

console.log('-- decorator function invoked --');

Object.freeze(constructor);

}

The class definition that uses @frozen:

@frozen

class Greeter {

name: string;

constructor(name: string) {

console.log('-- this constructor invoked --');

this.name = name;

}

greet() {

console.log("Hello, " + this.name);

}

}

**Method Decorators**

A Method Decorator is declared before a method declaration, and is applied to the **Property Descriptor** for the method. They are used to observe, modify, or replace a method definition.

**Example:**

class Greeter {

name: string;

constructor(name: string) {

console.log('-- this constructor invoked --');

this.name = name;

}

@enumerable(false)

greet() {

console.log("Hello, " + this.name);

}

}

We can define the decorator @enumberable using the following function declaration:

function enumerable(value: boolean) {

return function (target: any, propertyKey: string, descriptor: PropertyDescriptor) {

descriptor.enumerable = value;

};

}

The @enumerable(false) decorator is a decorator factory. When the @enumerable(false) decorator called, it modifies the [enumerable property](https://javascript.info/property-descriptors) of the property descriptor.

**Property Decorator**

Property decorators are used to listen to state changes on a class.

**Example:**

class Employee {

@logProperty

public name: string;

}

//property decorator

function logProperty(target: any, key: string) {

// property value

var \_val = this[key];

// property getter

var getter = function () {

console.log(`Get: ${key} => ${\_val}`);

return \_val;

};

// property setter

var setter = function (newVal) {

console.log(`Set: ${key} => ${newVal}`);

\_val = newVal;

};

// Delete property.

if (delete this[key]) {

// Create new property with getter and setter

Object.defineProperty(target, key, {

get: getter,

set: setter,

enumerable: true,

configurable: true

});

}

}

**Parameter Decorators**

A Parameter Decorator is declared before a **parameter declaration** and is applied to the function for a class constructor or method declaration.

**Example:**

function greet(@required name: string) {

console.log( `hello, ${name}`);

}

function required(target: Object, propertyKey: string | symbol, parameterIndex: number) {

// parameter decorator definition

}

**Accessor Decorators**

An Accessor Decorator is applied to the **property descriptor for the accessor**.

**Example:**

class Student{

private \_id: number;

private \_name: string;

constructor(id: number, name: string) {

this.\_id = id;

this.\_name = name;

}

@configurable(false)

get id() { return this.\_id; }

@configurable(false)

get name() { return this.\_name; }

}

// Accessor Decorator

function configurable(value: boolean) {

return function (target: any, propertyKey: string, descriptor: PropertyDescriptor) {

descriptor.configurable = value;

};

}

The @configurable(false) decorator is a decorator factory. When the @configurable(false) decorator called, it modifies the [configurable property](https://javascript.info/property-descriptors) of the property descriptor.

**References**

* [TypeScript Docs - Decorators](https://www.typescriptlang.org/docs/handbook/decorators.html)