



FPT POLYTECHNIC



THỰC HỌC – THỰC NGHIỆP



Conceive Design Implement Operate

LẬP TRÌNH TYPESCRIPT

DECORATORS

- ① 1.1 Tổng quan decorator
- ① 1.2 Class decorator
- ① 1.3 Decorator factory
- ① 1.4 Property decorator
- ① 1.5 Method decorator
- ① 1.6 Accessor decorator
- ① 1.7 Parameter decorator
- ① 1.8 Trả về giá trị trong decorator





PHẦN 1

- ❑ Decorator: là cú pháp khai báo đặc biệt đi kèm với khai báo class, method, accessor, property hoặc parameter
- ❑ Decorator: có nhiệm vụ thay đổi hoặc bổ sung cho đối tượng được decorator
- ❑ Cú pháp: **@*expression***
 - ❖ Expression: trỏ tới một function sẽ được gọi lúc run-time

Class decorator

@Theme

```
class Employee{
```

@Required

```
employeeID : number;
```

@Required

```
fullName : string;
```

@Track

```
showDetails() : void {
```

```
}
```

```
}
```

Property decorator

Method decorator

Accessors decorator

Parameter decorator

- ❑ Trong file tsconfig.json bỏ comment thuộc tính "experimentalDecorators": true

```
{  
  "compilerOptions": {  
    "target": "ES5",  
    "experimentalDecorators": true  
  }  
}
```

❑ Khai báo trên 1 dòng

```
@f() @g() method() {}
```

❑ Khai báo trên nhiều dòng

```
@f()  
@g()  
method() {}
```

- ☐ Class decorator — Ưu tiên 4
(Object Instance, Static)
- ☐ Method Decorator — Ưu tiên 2
(Object Instance, Static)
- ☐ Accessor or Property Decorator — Ưu tiên 3
(Object Instance, Static)
- ☐ Parameter Decorator — Ưu tiên 1
(Object Instance, Static)

- ❑ Nếu decorator áp dụng cho constructor class thì độ ưu tiên decorator như sau
 - ❖ 1. parameter
 - ❖ 2. method
 - ❖ 3. accessor hoặc property decorator
 - ❖ 4. class decorator

□ Ví dụ

```
class C {  
    @f()  
    @g()  
    method() {}  
}  
//f():evaluated  
//g():evaluated  
//g():called  
//f():called
```

- ❑ Class decorator được khai báo ngay trước khai báo class.
- ❑ Class decorator được áp dụng cho constructor của class và có thể được sử dụng để theo dõi, sửa đổi hoặc thay thế cho định nghĩa class

❏ Ví dụ

```
function Logger(constructor: Function) {  
  console.log('Logging...');  
  console.log(constructor);  
}
```

```
@Logger  
class Person {  
  name = 'Max';  
  
  constructor() {  
    console.log('Creating person object...');  
  }  
}
```

```
const pers = new Person();
```

```
console.log(pers);
```

❑ **Decorator factory:** là một hàm trả về chính hàm decorator và có thể thêm đối số cho hàm.

❑ Ví dụ:

```
function Logger(logString: string) {  
  return function(constructor: Function) {  
    console.log(logString);  
    console.log(constructor);  
  };  
}  
  
@Logger('LOGGING - PERSON')  
class Person {  
  name = 'Max';  
  
  constructor() {  
    console.log('Creating person object...');  
  }  
}
```

demo



PHẦN 2

- ❑ Property decorator: được áp dụng trên phần khai báo thuộc tính
- ❑ Hàm property decorator được gọi với hai đối số sau:
 - ❖ Hàm constructor của class cho static member HOẶC prototype của class cho instance member
 - ❖ Tên thuộc tính

❏ Ví dụ

```
function Log(target: any, propertyName: string | Symbol) {  
    console.log('Property decorator!');  
    console.log(target, propertyName);  
}  
  
class Product {  
    @Log  
    title: string;  
    private _price: number;  
  
    set price(val: number) { }  
  
    constructor(t: string, p: number) {  
        this.title = t;  
        this._price = p;  
    }  
  
    getPriceWithTax() {}  
}
```

- ❑ Method decorator: được áp dụng cho phần khai báo phương thức của class
- ❑ Hàm method decorator được gọi với 3 đối số

```
function MethodDecorator(target: any,  
                           propertyKey: string,  
                           descriptor: PropertyDescriptor)  
{  
    //code here  
}
```

- ❖ Target: Hàm constructor của class cho static member HOẶC prototype của class cho instance member
- ❖ propertyKey: tên của method (phương thức)
- ❖ Descriptor: Property Descriptor của method

□ Ví dụ

```
function Log3(target: any, name: string | Symbol, descriptor: PropertyDescriptor) {
    console.log('Method decorator!');
    console.log(target);
    console.log(name);
    console.log(descriptor);
}
```

```
class Product {

    title: string;
    private _price: number;

    set price(val: number) { }

    constructor(t: string, p: number) {
        this.title = t;
        this._price = p;
    }

    @Log3
    getPriceWithTax() { }
}
```

- ❑ Accessor decorator: giống method decorator nhưng được áp dụng cho ***setter hoặc getter***
- ❑ Typescript không cho phép tạo decorator cho cả getter và setter

□ Ví dụ

```
class Product {  
    title: string;  
    private _price: number;  
  
    @Log2  
    set price(val: number) {  
        if (val > 0) {  
            this._price = val;  
        } else {  
            throw new Error('Invalid price - should be positive!');  
        }  
    }  
  
    constructor(t: string, p: number) {  
        this.title = t;  
        this._price = p;  
    }  
  
    getPriceWithTax() { }  
}
```

- ❑ Parameter decorator được áp dụng cho khai báo cho tham số của phương thức hoặc tham số của constructor
- ❑ Parameter decorator được gọi với 3 đối số
 - ❖ Target: Hàm constructor của class cho static member HOẶC prototype của class cho instance member
 - ❖ Tên của parameter được decorator
 - ❖ Thứ tự của param trong list các params của function cha
- ❑ Parameter decorator chỉ được sử dụng để kiểm tra sự tồn tại của params trong function , và thường được dùng kết hợp với method decorator hoặc accessor decorator

□ Ví dụ

```
function Logger<T extends { new(...args: any[]): {} }>(constructor: T) {  
  return class extends constructor {  
    name_class = 'decorator_ex';  
  }  
}
```

❑ Ví dụ

```
function Autobind(_: any, _2: string, descriptor: PropertyDescriptor) {  
  const originalMethod = descriptor.value;  
  const adjDescriptor: PropertyDescriptor = {  
    configurable: true,  
    enumerable: false,  
    get() {  
      const boundFn = originalMethod.bind(this);  
      return boundFn;  
    }  
  };  
  return adjDescriptor;  
}
```



```
class Printer {  
  message = 'This works!';  
  
  @Autobind  
  showMessage() {  
    console.log(this.message);  
  }  
}
```

```
const p = new Printer();  
p.showMessage();
```

```
const button = document.querySelector('button');  
button.addEventListener('click', p.showMessage);
```

demo

- ☑ Decorator và factory decorator
- ☑ Class decorator
- ☑ Property decorator
- ☑ Method decorator
- ☑ Accessor decorator
- ☑ Parameter decorator



thank
you!