**修饰器Decorator**

1、类Decorator

给类添加静态属性，或给实例添加属性

@addSkill

class Person { }

function addSkill(target) {

target.say = "hello world"; //直接添加到类中

target.prototype.eat = "apple"; //添加到类的原型对象中

}

var personOne = new Person()

console.log(Person['say']) // 'hello world'

console.log(personOne['eat']) // 'apple'

2、方法Decorator

作用于类方法或属性上

class Person {

constructor() {}

@myname //方法修饰器

name() {

console.log('霖呆呆')

}

}

function myname(target, key, descriptor) {

console.log(target); //类原型对象

console.log(key); //所要修饰的属性名

console.log(descriptor); //该属性的描述对象

descriptor.value = function() {

console.log('霖呆呆')

}

}

var personOne = new Person() //实例化

personOne.name() //调用name()方法

//打印结果：

Person {}

name

{ value: [Function: name],

writable: true,

enumerable: false,

configurable: true

}

霖呆呆

3、多修饰器执行顺序

class Person {

constructor() {}

@dec(1)

@dec(2)

name() {

console.log('霖呆呆')

}

}

function dec(id) {

console.log('out', id);

return function(target, key, descriptor) {

console.log(id);

}

}

var person = new Person()

person.name()

//结果

out 1

out 2

2

1

霖呆呆

4、不能作用于函数

这是因为函数和变量一样都会提升

var counter = 0;

var add = function () {

counter++;

};

@add

function foo() {

}

提升后效果

@add

function foo() {

}

var counter;

var add;

counter = 0;

add = function () {

counter++;

};

可以采用高阶函数的形式直接执行

function doSometing(name) {

console.log('Hello' + name)

}

function myDecorator(fn) {

return function() {

console.log('start')

const res = fn.apply(this, arguments)

console.log('end')

return res

}

}

const wrapped = myDecorator(doSometing)

doSometing('lindaidai')

//Hellowlindaidai

wrapped('lindaidai')

//start

//Hellowlindaidai

//end