依赖反转和修饰器

2021年2月20日

17:05

* 1. 高耦合

class Rate{

star(stars){

console.log('您对订单的评价为%s星',stars);

}

}

class Order{

constructor(){

this.rate = new Rate();

}

// 省去模块其余部分 ...

}

let order = new Order('新订单');

order.getInfo();

order.rate.star(5);

* 1. IOC

class Order {

// 用于维护依赖关系的Map

static modules = new Map();

constructor(){

for (let module of Order.modules.values()) {

// 调用模块init方法

module.init(this);

}

}

// 向依赖关系Map中注入模块

static inject(module) {

Order.modules.set(module.constructor.name, module);

}

/\*\* 其余部分略 \*/

}

class Rate{

init(order) {

order.rate = this;

}

star(stars){

console.log('您对订单的评价为%s星',stars);

}

}

const rate = new Rate();

// 注入依赖

Order.inject(rate);

const order = new Order();

order.rate.star(4);

* 1. 利用修饰器实现更高效简洁

class Aftermarket {

repair() {

console.log('已收到您的售后请求');

}

}

class Rate {

star(stars: string) {

console.log(`评分为${stars}星`);

}

}

class Share {

shareTo(platform: string) {

switch (platform) {

case 'wxfriend':

console.log('分享至微信好友');

break;

case 'wxposts':

console.log('分享至微信朋友圈');

break;

case 'weibo':

console.log('分享至微博');

break;

default:

console.error('分享失败，请检查platform');

break;

}

}

}

function Inject(modules: any) {

return function(target: any) {

modules.forEach((module:any) => {

target.prototype[module.name] = new module();

});

};

}

@Inject([Aftermarket,Share,Rate])

class Order {

constructor() {}

/\*\* 其它实现略 \*/

}

const order:any = new Order();

order.Share.shareTo('facebook');

**解释:** @修饰符后接函数; 如果需要传参, 可以传递一个返回新函数的高级函数, 此过程可以看做高级函数立即执行, 并把参数放入闭包中; 修饰器接受的函数有三个参数(target, name, descriptor), 分别代表修饰的对象/修饰器名称/对象