第七课 代理模式

学习目录

- **使用场景**
- **使用方式**
- **优缺点**

一．使用场景

有时候我们准备买二手房的时候，我们可以选择到房产中介先来看好房源，通过中介找到房东，在看房购房。也就是说我们可能不方便直接找到房东，房产中介作为代理层，帮助我们寻找好房子，我们只需要专心购房即可，中间看房与购房的逻辑由房产中介来处理。

在日常开发过程中也是一样的道理，有时候我们不方便直接使用某些对象，或者这个对象不能满足逻辑需求的时候，我们可以设计一个代理对象来完成这些逻辑需求。同时这个代理对象也可以控制、管理、协助我们对于某些目标对象的访问，有些请求可能我们的目标对象不想处理，也可能有些请求开销较大，目标对象只有在合适的时候才能处理。

代理模式的核心思想是构建一个代理对象，这个代理对象会对访问目标对象的请求进行处理，在把处理结果交给本体对象。

二．使用方式

在不同语言中代理模式有着不同的提现，比如远程代理、保护代理等等代理模式，在 js 中最常用的是虚拟代理。

在虚拟代理中首先要声明代理对象与目标对象一样的访问接口，在代理对象内部要包含

有目标对象，代理对象大多数情况下要在真正调用目标对象之前和之后做一些逻辑操作，保证代理对象可以对目标对象进行控制或者替换等。

文本内容验证

```
var createDiv = (function(){
    var $div = document.createElement('div');
    document.body.appendChild($div);
    return {
        createFn:function(str){
            $div.innerHTML = str;
        }
    }
})();
var proxyStrFn = (function(){
    var that = this;
    setTimeout(function(){
        createDiv.createFn(that.str);
    },2000);
    return {
        createFn:function(str){
            createDiv.createFn('请等待文本规则验证');
            if(str.indexOf('星星课堂') != -1){
```

```
                that.str = str;
            }else{
                that.str = '文本不包含星星课堂';
            }
        }
    }
})();
proxyStrFn.createFn('星星课堂');
```

图片预加载

```
var loadImgFn = (function(){
    var imgNode = document.createElement('img');
    document.body.appendChild(imgNode);
    return {
        setSrc:function(_src){
            imgNode.src = _src;
        }
    }
})();

var proxyImgFn = (function () {
    var img = new Image();
```

```
        img.onload = function () {
            loadImgFn.setSrc(img.src);
        }
        return function (lodingUrl, src) {
            loadImgFn.setSrc(lodingUrl);
            img.src = src;
        }
    })()

    proxyImgFn('./loadingIcon.gif',
'https://ss2.bdstatic.com/70cFvnSh_Q1YnxGkpoWK1HF6hhy/it/u=2953654911,2387843907&fm=26&gp=0.jpg');
```

缓存代理计算斐波那契数列

```
    var computedFn = function(num){
        if(num <= 2){
            return 1;
        }else{
            return arguments.callee(num -1) + arguments.callee(num - 2)
        }
    }
    var proxyFn = function(fn){
```

```
        var cache = {};
        return function(){
            var argsStr = Array.prototype.join.call(arguments,',');
            if(argsStr in cache){
                return cache[argsStr];
            }
            return cache[argsStr] = fn.apply(this,arguments);
        }
    }
    var fn1 = proxyFn(computedFn);
    console.log(fn1(40));
    console.log(fn1(40));
```

这些都是代理模式在 ES5 中的主要体现，在 ES6 中内置了 Proxy 对象，我们不必在用闭包来编写代理模式，但是 Proxy 对象的兼容性会有点问题，低版本的浏览器可能会报错，或者需要进行 ES6 转 ES5 在使用也可以，这里只是简单的介绍一下，在 ES6 课程中会详细介绍 Proxy 对象更多的用法与知识。

```
    var proxyNewFn = (fn, cache = {}) => {
        return new Proxy(fn, {
            apply:function(target, context, args) {
                var argsString = args.join(',');
                if (argsString in cache) {
```

```
                return cache[argsString];
            }
            return cache[argsString] = fn.apply(this,args);
        }
    })
}
var fn2 = proxyNewFn(computedFn);
```

三．优缺点

1.代理模式也是比较成熟的设计模式，很多其他语言也会用到，代理模式在 js 中可以利用闭包缓存变量的特点对较大运算进行缓存，有利于增进应用性能。

2.代理模式中的虚拟代理，可以由代理对象负责管理对目标对象的请求，让目标对象专注于单一职责原则，不需要处理其他代理逻辑。

3.代理模式虽然让目标对象可以专注于自身业务逻辑，但是会在业务逻辑中多出一个代理对象要维护，这也会增加一定的难度。

4.代理模式主要使用了闭包来编写，在使用的时候要注意闭包缓存的大小，避免引用类型数据量快速增量变化带来的内存泄露等问题。

谢谢观看！

我是星星课堂老师：周小周