别人知道的设计模式

1

什么是设计模式

开发中特定问题的解决方案——思维

每一个模式描述了不断重复发生的问题以及该问题的解决方案的核心

理解方法

理解思想

理解观念

2

工厂模式

安全模式下的工厂模式

不同条件下创建不同实例

工厂模式合适用：

1:对象的构建十分复杂
2:需要依赖具体环境创建不同实例
3:处理大量具有相同属性的小对象

```
const factory=function(arr){
    if(!(this instanceof factory))
    {
        return new factory();
    }
}

factory.prototype={
    a:()=>{
        console.log('aaa');
    },
    b:()=>{
        console.log('bbb');
    }
}

const c=new factory();
c.a();

const d=factory();
d.a();
```

优点：
1、一个调用者想创建一个对象，只要知道其名称就可以了。
2、扩展性高，如果想增加一个产品，只要扩展一个工厂类就可以。
3、屏蔽产品的具体实现，调用者只关心产品的接口。

缺点：
每次增加一个产品时，都需要增加一个具体类和对象实现工厂，使得系统中类的个数成倍增加，在一定程度上增加了系统的复杂度，同时也增加了系统具体类的依赖。这并不是什么好事。

3

抽象工厂模式

抽象工厂模式

系统的产品有多于一个的产品族，而系统只消费其中某一族的产品。

在一个产品族里面，定义多个产品

```
<script>
    const factory = function(subType, superType) {
        if(typeof factory[superType] === 'function') {
            function F() {};
            F.prototype = new factory[superType]();
            subType.prototype = new F() ;
        }
    }
    factory.milk = function() {
        this.type = '牛奶'
    }
    factory.milk.prototype = {
        name: ()=> {console.log('蒙牛')},
        brand: ()=> {console.log('早餐奶')}
    }
    factory.milk1 = function() {this.type = '酸奶'}
    factory.milk1.prototype = {
        name: ()=> {console.log('伊利')},
        brand: ()=>{console.log('安慕希')}
    }
    const yili = function(mtype,num) {
        this.mtype=mtype;
        this.num=num;
    }
    factory(yili, 'milk')
    yili.prototype.name = ()=> {console.log('伊利')}
    yili.prototype.brand =()=> {console.log('冠军奶')}
    const new_milk = new yili('袋装', 2)
    new_milk.name(new_milk.name)
    console.log('类别：'+new_milk.type)
    console.log('类型：'+new_milk.mtype)
    console.log('数量：'+new_milk.num)
    console.log(new_milk instanceof factory.milk)
```

优点：当一个产品族中的多个对象被设计成一起工作时，它能保证客户端始终只使用同一个产品族中的对象。

缺点：产品族扩展非常困难，要增加一个系列的某一产品，既要在抽象的方法内里加代码，又要在具体的里面加代码。

4

代理模式

代理模式

想在访问一个类时做一些控制 增加中间层

例如: 买火车票不一定在火车站买，也可以去代售点

```
<script>

    const girl = function (name) {

        this.name = name;

    };

    const boy = function (girl) {

        this.girl = girl;

        this.sendGift =  (gift)=>{
            console.log("Hi " + girl.name + ", boy送你一个礼物：" + gift);
        }

    };

    const dl = function (girl) {

        this.girl = girl;

        this.sendGift =(gift)=>{
           var d=new boy(girl);
           d.sendGift(gift);
        }

    };

    const proxy = new dl(new girl("妹子"));

    proxy.sendGift("999朵玫瑰");
```

优点：

1、职责清晰。
2、高扩展性。
3、智能化。

缺点：

1、由于在客户端和真实主题之间增加了代理对象，因此有些类型的代理模式可能会造成请求的处理速度变慢。
2、实现代理模式需要额外的工作，有些代理模式的实现非常复杂。

虚拟代理模式

虚拟代理（根据需要创建开销很大的对象如渲染网页暂时用占位代替真图）

```
var MyImage = (function() {
    var imgNode = document.createElement('img');
    document.body.appendChild(imgNode);
    var img = new Image;
    img.onload = function() {
        imgNode.src = img.src;
    };
    return {
        setSrc: function(src) {
            imgNode.src = 'img/cat.jpg';
            img.src = src;
        }
    }
})();
MyImage.setSrc('http://e.hiphotos.baidu.com/image/pic/item/2f738bd4b31c8701ad467c1a2b7f9e2f0608ff5e.jpg');
MyImage.setSrc('');
```

待续……