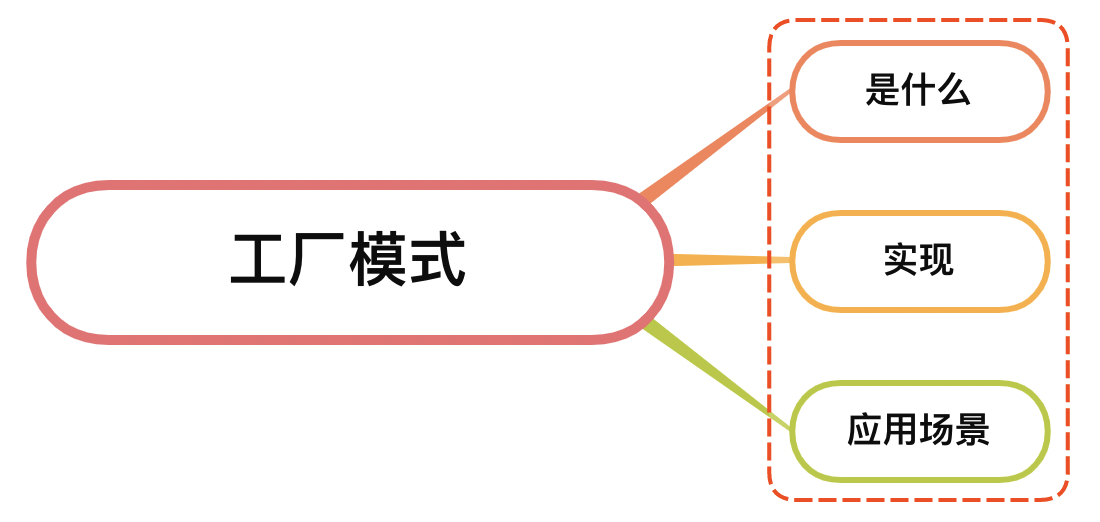
# 面试官：说说你对工厂模式的理解？应用场景？



## 一、是什么

工厂模式是用来创建对象的一种最常用的设计模式，不暴露创建对象的具体逻辑，而是将将逻辑封装在一个函数中，那么这个函数就可以被视为一个工厂

其就像工厂一样重复的产生类似的产品，工厂模式只需要我们传入正确的参数，就能生产类似的产品

举个例子：

* 编程中，在一个 A 类中通过 new 的方式实例化了类 B，那么 A 类和 B 类之间就存在关联（耦合）
* 后期因为需要修改了 B 类的代码和使用方式，比如构造函数中传入参数，那么 A 类也要跟着修改，一个类的依赖可能影响不大，但若有多个类依赖了 B 类，那么这个工作量将会相当的大，容易出现修改错误，也会产生很多的重复代码，这无疑是件非常痛苦的事；
* 这种情况下，就需要将创建实例的工作从调用方（A类）中分离，与调用方**解耦**，也就是使用工厂方法创建实例的工作封装起来（**减少代码重复**），由工厂管理对象的创建逻辑，调用方不需要知道具体的创建过程，只管使用，**而降低调用者因为创建逻辑导致的错误**；

## 二、实现

工厂模式根据抽象程度的不同可以分为：

* 简单工厂模式（Simple Factory）
* 工厂方法模式（Factory Method）
* 抽象工厂模式（Abstract Factory）

### 简单工厂模式

简单工厂模式也叫静态工厂模式，用一个工厂对象创建同一类对象类的实例

假设我们要开发一个公司岗位及其工作内容的录入信息，不同岗位的工作内容不一致

代码如下：

function Factory(career) {  
 function User(career, work) {  
 this.career = career   
 this.work = work  
 }  
 let work  
 switch(career) {  
 case 'coder':  
 work = ['写代码', '修Bug']   
 return new User(career, work)  
 break  
 case 'hr':  
 work = ['招聘', '员工信息管理']  
 return new User(career, work)  
 break  
 case 'driver':  
 work = ['开车']  
 return new User(career, work)  
 break  
 case 'boss':  
 work = ['喝茶', '开会', '审批文件']  
 return new User(career, work)  
 break  
 }  
}  
let coder = new Factory('coder')  
console.log(coder)  
let boss = new Factory('boss')  
console.log(boss)

Factory就是一个简单工厂。当我们调用工厂函数时，只需要传递name、age、career就可以获取到包含用户工作内容的实例对象

### 工厂方法模式

工厂方法模式跟简单工厂模式差不多，但是把具体的产品放到了工厂函数的prototype中

这样一来，扩展产品种类就不必修改工厂函数了，和心累就变成抽象类，也可以随时重写某种具体的产品

也就是相当于工厂总部不生产产品了，交给下辖分工厂进行生产；但是进入工厂之前，需要有个判断来验证你要生产的东西是否是属于我们工厂所生产范围，如果是，就丢给下辖工厂来进行生产

如下代码：

// 工厂方法  
function Factory(career){  
 if(this instanceof Factory){  
 var a = new this[career]();  
 return a;  
 }else{  
 return new Factory(career);  
 }  
}  
// 工厂方法函数的原型中设置所有对象的构造函数  
Factory.prototype={  
 'coder': function(){  
 this.careerName = '程序员'  
 this.work = ['写代码', '修Bug']   
 },  
 'hr': function(){  
 this.careerName = 'HR'  
 this.work = ['招聘', '员工信息管理']  
 },  
 'driver': function () {  
 this.careerName = '司机'  
 this.work = ['开车']  
 },  
 'boss': function(){  
 this.careerName = '老板'  
 this.work = ['喝茶', '开会', '审批文件']  
 }  
}  
let coder = new Factory('coder')  
console.log(coder)  
let hr = new Factory('hr')  
console.log(hr)

工厂方法关键核心代码是工厂里面的判断this是否属于工厂，也就是做了分支判断，这个工厂只做我能做的产品

### 抽象工厂模式

上述简单工厂模式和工厂方法模式都是直接生成实例，但是抽象工厂模式不同，抽象工厂模式并不直接生成实例， 而是用于对产品类簇的创建

通俗点来讲就是：简单工厂和工厂方法模式的工作是生产产品，那么抽象工厂模式的工作就是生产工厂的

由于JavaScript中并没有抽象类的概念，只能模拟，可以分成四部分：

* 用于创建抽象类的函数
* 抽象类
* 具体类
* 实例化具体类

上面的例子中有coder、hr、boss、driver四种岗位，其中coder可能使用不同的开发语言进行开发，比如JavaScript、Java等等。那么这两种语言就是对应的类簇

示例代码如下：

let CareerAbstractFactory = function(subType, superType) {  
 // 判断抽象工厂中是否有该抽象类  
 if (typeof CareerAbstractFactory[superType] === 'function') {  
 // 缓存类  
 function F() {}  
 // 继承父类属性和方法  
 F.prototype = new CareerAbstractFactory[superType]()  
 // 将子类的constructor指向父类  
 subType.constructor = subType;  
 // 子类原型继承父类  
 subType.prototype = new F()  
 } else {  
 throw new Error('抽象类不存在')  
 }  
}

上面代码中CareerAbstractFactory就是一个抽象工厂方法，该方法在参数中传递子类和父类，在方法体内部实现了子类对父类的继承

## 三、应用场景

从上面可看到，简单简单工厂的优点就是我们只要传递正确的参数，就能获得所需的对象，而不需要关心其创建的具体细节

应用场景也容易识别，有构造函数的地方，就应该考虑简单工厂，但是如果函数构建函数太多与复杂，会导致工厂函数变得复杂，所以不适合复杂的情况

抽象工厂模式一般用于严格要求以面向对象思想进行开发的超大型项目中，我们一般常规的开发的话一般就是简单工厂和工厂方法模式会用的比较多一些

综上，工厂模式适用场景如下：

* 如果你不想让某个子系统与较大的那个对象之间形成强耦合，而是想运行时从许多子系统中进行挑选的话，那么工厂模式是一个理想的选择
* 将new操作简单封装，遇到new的时候就应该考虑是否用工厂模式；
* 需要依赖具体环境创建不同实例，这些实例都有相同的行为,这时候我们可以使用工厂模式，简化实现的过程，同时也可以减少每种对象所需的代码量，有利于消除对象间的耦合，提供更大的灵活性

## 参考文献

* https://www.runoob.com/design-pattern/factory-pattern.html
* https://juejin.cn/post/6844903653774458888
* https://zhuanlan.zhihu.com/p/344119981