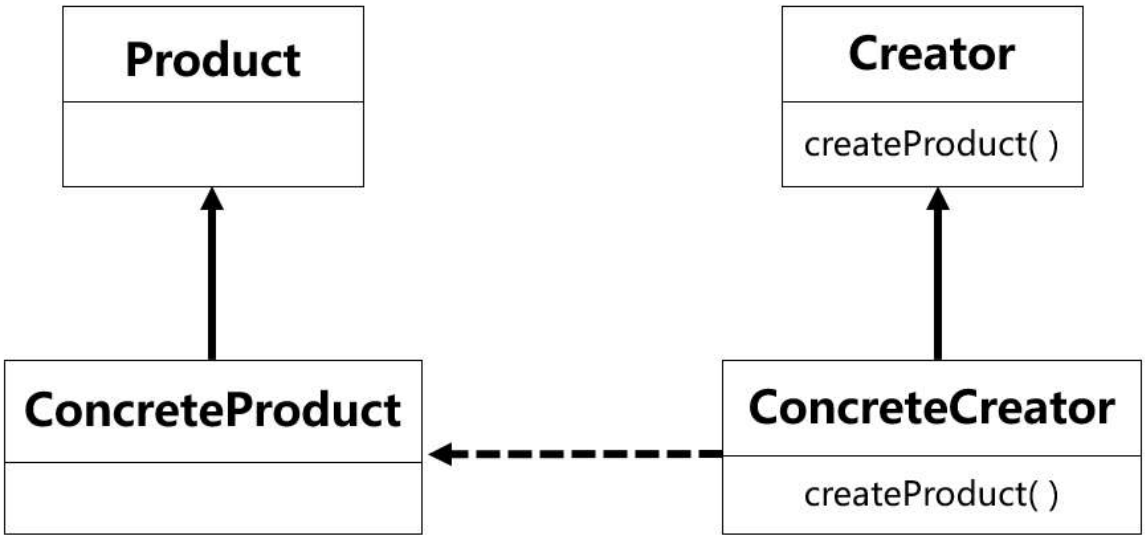
大家好，本小节中我们继续交流学习常见的设计模式。在上一小节中，我们介绍了设计模式的六大原则，并且重点阐述了单例模式的多种写法。单例模式也是为数不多的可以在面试中直接“手撕”的设计模式。除了单例模式外，在面试中考察的其余常见设计模式一般都不会要求我们手写代码。

所以，我们可以重点掌握常见设计模式的思想，在面试中可以较为清晰的阐述该种设计模式主要是在干什么，需要怎么干即可。好了，让我们来看下面试中常见的设计模式吧~

（1）工厂方法模式：

工厂方法模式是一种常见的设计模式。工厂方法模式定义了一个用于创建对象的接口，让子类决定实例化哪一个类。工厂方法模式使一个类的实例化延迟到其子类。

工厂方法模式的类图：



工厂方法模式通用类图

其中，Product和ConcreteProduct分别表示抽象产品类和具体产品类。Creator和ConcreteCreator则分别表示抽象创建类和具体创建类。抽象创建类Creator中定义了创建产品的方法createProduct。

这里我们给出一个简单的工厂方法模式的Demo：

```
package niuke.facatory;

public class FactoryMethodTest {
    public static void main(String[] args) {
        // 创建具体的创建类对象
        Creator creator = new ConcreteCreator();
        // 通过传入指定的产品类对象，来创建对应的产品
        Product product = creator.createProduct(ConcreteProduct1.class);
        // 创建对象之后，可以进行业务逻辑处理
        product.method1();
        product.method2();
    }
```

```
}
// 定义抽象产品类
abstract class Product {
    // 产品类的公共方法
    public void method1(){
        // 公共的业务逻辑
    }
    // 抽象方法
    public abstract void method2();
}

// 定义具体产品类
class ConcreteProduct1 extends Product {
    public void method2() {
        // 具体产品类1的业务逻辑处理
    }
}
class ConcreteProduct2 extends Product {
    public void method2() {
        // 具体产品类2的业务逻辑处理
    }
}

// 定义抽象创建类
abstract class Creator {
    // 创建对象的抽象方法
    public abstract <T extends Product> T createProduct(Class<T> c);
}
// 定义具体的创建类，真正来创建所需的对象
class ConcreteCreator extends Creator {
    public <T extends Product> T createProduct(Class<T> c){
        Product product=null;
        try {
            // 通过反射技术来创建对象
            product = (Product)Class.forName(c.getName()).newInstance();
        } catch (Exception e) {
            //异常处理
        }
        return (T)product;
    }
}
```

4

那么，工厂方法模式的优点有哪些呢?

- 工厂方法模式具有很好的封装性。客户端不需要知道创建对象的过程，只需要知道要创建的是哪个具体的产品即可。
- 工厂方法模式对扩展开放。当新增一个产品种类的时候，我们只需要传入新增产品类对象给具体工厂，即可返回新增的产品对象。

工厂方法模式的使用场景:

- 工厂方法模式的作用就是创建指定的对象，可以做为new一个对象的替代方式。但是需要考虑是否有必要使用工厂方法模式来创建对象。
- 当需要灵活，可扩展的创建多个对象的场景时，可以使用工厂方法模式。

工厂方法模式总结:

工厂方法模式是一种常见，并且比较简单的设计模式。在面试中，我们一般需要说出工厂方法模式的思想和具体实现步骤，当然可以熟练的画出工厂方法模式的类图会更棒。

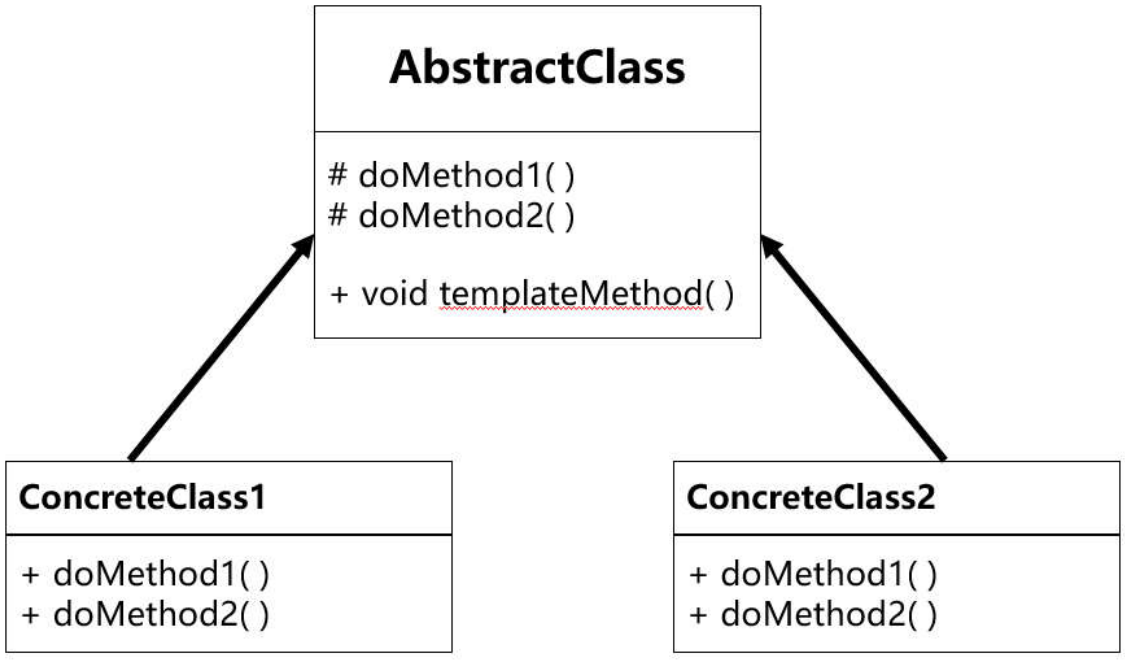
工厂方法模式的本质就是指定一个要创建对象的类，然后传给具体的工厂类，由具体工厂类通过反射技术来创建并且返回一个对象。工厂方法模式可以扩展成为简单工厂模式和多工厂模式等，限于文章篇幅，希望大家可以自行学习并且在评论区互动交流。

4

（2）模板方法模式：

模板方法模式也是一个常见的模式。模板方法模式定义了一个框架，将一些步骤延迟到其子类中实现，子类可以在不改变框架的前提下重新定义某些特定的执行步骤。

模板方法模式的类图：



模板方法模式通用类图

AbstractClass是一个抽象模板，它的方法分为模板方法和基本方法。

- 基本方法：是抽象方法，由子类实现，并且在模板方法中被调用。
- 模板方法：可以有一个或者几个，一般是具体的方法，实现对基本方法的调度，完成确定的业务逻辑，确定一个框架。

ConcreteClass1和ConcreteClass2属于具体模板类，实现抽象模板所定义的抽象方法，并且拥有父类模板中的模板方法。

接下来，我们给出一个模板方法模式的Demo：

```
package niuke.template;

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.start();

        Cat cat = new Cat();
        cat.start();
    }
}
```

```
// 抽象的父类
abstract class Animal{
    // 定义抽象方法
    abstract void eat();
    abstract void run();

    // 定义具体的模板方法
    public void start(){
        // 定义一个框架，确定方法的执行步骤
        eat();
        run();
    }
}
// 定于子类
class Dog extends Animal{
    @Override
    void eat() {
        System.out.println("我是小狗，我在吃东西...");
    }

    @Override
    void run() {
        System.out.println("我是小狗，我在跑步...");
    }
}

class Cat extends Animal{
    @Override
    void eat() {
        washHand();
        System.out.println("我是小猫，我在吃东西...");
    }

    @Override
    void run() {
        System.out.println("我是小猫，我在跑步...");
    }

    // 其余业务逻辑
    private void washHand(){
        System.out.println("我是小猫，我在洗手...");
    }
}
```

4

由案例可以看出，我们在不改变eat和run方法的执行顺序的前提下，Cat类通过在实现eat方法的加入其他的业务逻辑，来改变了eat这个特定的步骤。

那么，模板方法模式的优点有哪些呢?

- 封装不变部分，扩展可变部分
- 提取公共部分代码，便于维护
- 行为由父类控制，子类实现

模板模式总结:

模板方法模式就是在模板方法中调用基本方法来确定整个算法的执行框架。希望大家可以有效理解模板方法和基本方法的含义，在面试中做到清晰的阐述。

限于文章篇幅，我们接下来的两个设计模式就进行一个简单的阐述，大家有想要交流的可以在评论区互动留言哦~

抽象工厂模式：

抽象工厂模式为创建一组相关或相互依赖的对象提供一个接口，而且无须指定它们的具体类。抽象工厂模式是工厂方法模式的升级版本，在有多个业务品种、业务分类时，通过抽象工厂模式产生需要的对象是一种非常好的解决方式。

抽象工厂模式和工厂方法模式的区别：

4

如果产品单一，适合使用工厂模式。但是如果有多个业务品种、业务分类时，需要使用抽象工厂模式。也就是说，工厂模式针对的是一个产品等级结构 ， 抽象工厂模式针对的是面向多个产品等级结构的。

代（dai）理模式：

代（dai）理模式也是常用的一种设计模式，在前面我们介绍过JDK的动态代（dai）理。代（dai）理模式为其他对象提供一种代（dai）理以控制对这个对象的访问。将原类进行封装，客户端不能直接找到原类，必须通过代（dai）理角色。即代（dai）理是原类的一个替身，客户端要找原类，必须找代（dai）理才可以搞定。明星和经纪人的关系就是一种代（dai）理模式。

代（dai）理模式又分为静态代（dai）理和动态代（dai）理。动态代（dai）理在实现阶段不用关心代（dai）理谁，而在运行阶段才指定代（dai）理哪一个对象。相对来说，自己写代（dai）理类的方式就是静态代（dai）理。

在实际的开发中，我们**常见的设计模式还有策略模式，责任链模式，观察者模式，装饰模式，适配器模式以及建造者模式等**。限于文章篇幅，我希望大家可以自行学习，欢迎大家在评论区讨论交流相关设计模式。

总结：

我们通过两个小节来对常见的设计模式进行了一个简单的阐述。在面试中，对单例模式的考察是常见的，我们必须熟练掌握，并且可以“手撕”成功。对于其余的设计模式，我们主要是掌握其基本思想和实现步骤，也就是明白这个设计模式的作用以及其实现步骤，设计模式的主要考察的还是一种思想。下一小节开始将进入面试分享章节，希望大家能够有所收获。

限于作者水平，文章中难免会有不妥之处。大家在学习过程中遇到我没有表达清楚或者表述有误的地方，欢迎随时在文章下边指出，我会及时关注，随时改正。另外，大家有任何话题都可以在下边留言，我们一起交流探讨。

讨论 评论

星如月勿忘初心 1#

动手实现一下这些设计模式才能更好的理解作者总结的内容

发表于 2020-03-02 22:43:59 赞(1) 回复(0)

牧水s 2#

是不是还有发布订阅模式呀....

发表于 2020-03-08 22:11:48 赞(0) 回复(3)

Offer快到碗儿里来： 是的是的，23种，面试说不完的

2020-03-11 15:55:18 赞(0) 回复(0)

我是祖国的花朵 作者： 策略模式在工作中也常用，可以替代if else

2020-03-15 17:37:13 赞(0) 回复(0)

中都： 发布订阅是观察者模式的一种

2020-04-21 13:41:20 赞(1) 回复(0)

请输入你的观点

4

回复

大姚Seven 3#

常见的还有

观察者模式(Zookeeper中的服务端的变动通知客户端),

适配器模式

策略模式:

希望大佬有时间能再讲讲

发表于 2020-04-14 00:47:31 赞(0) 回复(0)

百折不挠的隋靠谱 4#

面试中最常考的是哪几种设计模式

发表于 2020-05-19 23:40:49 赞(0) 回复(1)

我是祖国的花朵 作者： 手写的是单例模式，工厂，模版，策略，看你会啥吧，不是说要考啥。不熟某一个，你可以说说你熟悉的即可。

2020-05-22 13:20:31 赞(0) 回复(0)

请输入你的观点

回复