# 第5章 工厂品类要丰富——抽象工厂模式

## 5.1 供应商不靠谱？直接换掉

兔小白：熊小猫，一起去吃晚饭？回来给我讲讲抽象工厂模式。

熊小猫：好呀，今天去便利店买点关东煮对付一下吧。晚上我还得解决一个Bug。

兔小白：没问题，今天我请客！

图：俩人去买关东煮，关东煮的图片，两人流口水。为了说明什么是关东煮。

兔小白：我感觉今天的关东煮比往常好啊！更有嚼劲，味道也更鲜美！

熊小猫：是呀，确实进步不少，我也吃出来了。可能便利店最近换了关东煮供应商。

兔小白：产品质量不好，被换掉是迟早的事。生产关东煮的厂家太多了。

熊小猫：和软件开发一样，质量永远是重中之重！你想过为什么换掉供应商这么容易吗？

兔小白：如果不同品类的关东煮从多家供应商采购，换供应商会比较麻烦。好在所有供应商加工的关东煮品类基本都一样。便利店应该都是从一家供应商采购，只需要把这一家供应商换掉就可以了。便利店卖的还是原来的关东煮品类，无缝对接！

熊小猫：没错，咱们吃的这些鱼肉丸、魔芋丝、鸡肉串、豆腐包之类，任何一家供应商都能生产。所以想替换掉供应商就太容易了！

兔小白：**当变化产生时，如果很容易就可以应对，那么其中大概率蕴含了某种设计模式。**我猜这就是今天要讲的抽象工厂模式吧？

熊小猫：没错！这就是抽象工厂模式。关东煮的供应商可以看做工厂。和前两天谈到的汉堡工厂不同，关东煮工厂可以生产各式品类产品，而不是仅限于一种品类。

图：关东煮工厂A和B都是生产各种品类，只是质量不一样。

在关东煮行业，一定会有长期沉淀下来的关东煮品类表。不同供应商都会生产这份品类表上的产品。便利店只需要对接一家供应商。当便利店想订购某种关东煮品类，就给这家供应商打电话订购。换掉这家供应商很简单，便利店只需要打电话给另外一家供应商订购。

兔小白：换供应商，其实只需要换个电话号码？

熊小猫：没错！电话号码可以认为是供应商的引用。我们看下面的图来理解这个过程。

图：便利店定货，供应商送货。品类表相当于工厂接口。两个供应商，其中一个说：最近怎么没有订货啊？

使用抽象工厂模式，能让你轻松面对全系列产品替换的问题。仅需要换一家工厂就搞定了。这就是抽象工厂的优势。

兔小白：假如去四川开便利店卖关东煮，就换一家麻辣口味的供应商！

熊小猫：没错，除了增加一家新的供应商，便利店不需要做出任何变化！！这就叫做……

兔小白：**对扩展开放，对修改关闭！**

熊小猫：呀！把我的话抢了！

兔小白：哈哈，这么重要的原则，天天听你说。我早就背下来了。

熊小猫：理论倒是说的头头是道，但还得看你代码写的怎么样。回公司我考考你！

兔小白：你不是晚上还要改Bug？

熊小猫：没关系，Bug……我再熬熬夜！

## 5.2 实现不同风格的显示组件

熊小猫：请听题，假如购物APP的商品详情页包括以下两个组件模块：

1. 商品图片
2. 商品介绍

你来写段简单的程序，示意商品详情页展示过程。

十分钟后。

兔小白：代码写好了。首先定义商品详情页中用到的组件“图片”和“文本”。

public class Image {  
    private String address;  
​  
    public void display() {  
        System.out.println("商品图片：" + address);  
   }  
​  
    public void setAddress(String address) {  
        this.address = address;  
   }  
}

public class Text {  
    private String content;  
​  
    public void display() {  
        System.out.println("商品介绍：" + content);  
   }  
​  
    public void setContent(String content) {  
        this.content = content;  
   }  
}

下面是商品详情页ProductDetailPage 类。构造函数中初始化“图片”和“文本”对象。display()方法负责显示两个组件内容。

public class ProductDetailPage {  
    private Image image;  
    private Text introduction;  
​  
    public ProductDetailPage(String imageAddress, String introductionContent) {  
        image = new Image();  
        image.setAddress(imageAddress);  
​  
        this.introduction = new Text();  
        this.introduction.setContent(introductionContent);  
   }  
​  
    public void display(){  
        image.display();  
        introduction.display();  
   }  
}

熊小猫：代码虽然简单，但完整实现了我的需求。

现在新的需求来了！页面默认显示的白底黑字，在夜间看起来会比较刺眼。为了提升用户夜间阅读体验，需要提供黑底色的夜间主题。

兔小白：这好办，把Image和Text类抽象一层，再分别写两种不同主题的的实现。

熊小猫：没错，增加一层抽象会带来相应的灵活度，客户端不再依赖页面组件的具体实现。赶紧去改吧！

二十分钟后，兔小白完成了代码修改。

兔小白：我把Image和Text类抽象为父类。增加了主题theme属性，通过构造函数设置theme。下面代码以Text为例。

public class Text {  
    private String content;  
    private String theme;  
​  
    public Text(String theme){  
        this.theme = theme;  
   }  
​  
    public void setContent(String content){  
        this.content=content;  
   }  
​  
    public void display() {  
        System.out.println("主题" + theme + "显示商品介绍：" + content);  
   }  
}

Image和Text各有两种子类。还是以Text为例，LightText和DarkText两个子类重写Text的无参构造方法，指定theme。代码如下：

public class LightText extends Text {  
    public LightText(){  
        super("Light");  
   }  
}

public class DarkText extends Text {  
    public DarkText(){  
        super("Dark");  
   }  
}

当使用夜间主题时，ProductDetialPage在构造方法中使用DarkImage和DarkText。

public class ProductDetailPage {  
    private Image image;  
    private Text introduction;  
​  
    public ProductDetailPage(String imageAddress, String introductionContent) {  
        image = new DarkImage();  
        image.setAddress(imageAddress);  
​  
        introduction = new DarkText();  
        introduction.setContent(introductionContent);  
   }  
​  
    public void display() {  
        image.display();  
        introduction.display();  
   }  
}

切换到日间主题，只需要修改下面两行代码，更换Image和Text实现。

image = new LightImage();  
introduction = new LightText();

别说两种主题，十种主题都不在话下！

熊小猫：修改之后确实可以应付多种主题。但是假如一种主题有10种组件呢？商品详情页除了图片、商品介绍，还会有评价、优惠显示、店铺介绍等组件。现在我们实现的只是最简单的商品详情页，未来增加需求的可能性非常大。

兔小白：如果10种组件切换主题， 那么ProductDetialPage类的构造方法需要改10行实例化组件的代码！这个成本可是太高了，而且一不小心还可能改错。本来要使用夜间主题，一旦改漏某个组件，页面中就会出现一片白色背景的区域。

熊小猫：没错，还得在程序设计上再下点功夫。今天咱们不是在学抽象工厂吗？你的工厂类呢？实例化Image和Text的工作可以交给工厂去做呀！想一想关东煮的的例子。所有品类的关东煮都是从一家供应商拿货。换一家供应商，就可以拿到同品类但质量更好的关东煮。

兔小白：可以把Image、Text类看成两种品类的关东煮——Image丸子和Text肉串。现在我的程序还缺两家关东煮的供应商！一家是黑暗系关东煮供应商，另外一家是光明系关东煮供应商。稍等，马上改好！

图：光明系和黑暗系供应商生产不同的Image丸子和Text肉串

熊小猫：哈哈！你这个类比虽然夸张，但还挺恰当！

十分钟过后，兔小白完成了代码修改。

兔小白：你说过关东煮行业有一份沉淀下来的品类列表，每家工厂都会生产列表上的产品。这其实就是工厂的接口，定义了工厂的标准。

public interface IFactory {  
    Text createText();  
    Image createImage();  
}

所有符合该标准的工厂，都需要有能力生产Text和Image对象。DarkThemeWidgetFactory负责夜间主题组件生产。LightThemeWidgetFactory负责日间主题组件生产。

public class DarkThemeWidgetFactory implements IFactory{  
    public Text createText(){  
        return new DarkText();  
   }  
    public Image createImage(){  
        return new DarkImage();  
   }  
}

public class LightThemeWidgetFactory implements IFactory{  
    public Text createText(){  
        return new LightText();  
   }  
    public Image createImage(){  
        return new LightImage();  
   }  
}

这两家工厂就如两家关东煮供应商，一家普通口味，一家麻辣口味！

现在，ProductDetialPage中用到的Text和Image对象改为从工厂获取。

public class ProductDetailPage {  
    private Image image;  
    private Text introduction;  
​  
    public ProductDetailPage(String imageAddress, String introductionContent) {  
        IFactory widgetFactory = new LightThemeWidgetFactory();  
          
        image = widgetFactory.createImage();  
        image.setAddress(imageAddress);  
​  
        introduction = widgetFactory.createText();  
        introduction.setContent(introductionContent);  
   }   
    public void display(){  
        image.display();  
        introduction.display();  
   }  
}

想换为日间主题？简直太方便了！只需要改一行代码，使用日间主题组件工厂即可。

IFactory widgetFactory = new LightThemeWidgetFactory();

刚才你给我出难题，更换10种组件风格。这一行代码就搞定啦！

## 5.3 抽象工厂模式适用场景

熊小猫：哈哈，不是我给你出难题。现实就是这样，唯一不变的就是变化。想要在面对变化时泰然自若，良好的程序设计是必不可少的。

这次改造完的代码，已经完全用上了抽象工厂模式！你先把这版代码的结构图画出来，咱们来仔细分析一下。

图：这版代码的类图

熊小猫：这版代码中，你对工厂进行了抽象。对工厂的产品——各种组件，也做了抽象。因此，客户端代ProductDetialPage类摆脱了对具体风格组件的依赖，同时也摆脱了对具体工厂的依赖。

使用抽象工厂模式可以带来如下好处：

1. 方便切换产品主题

这是抽象工厂模式最为显著的优势。客户端仅需修改创建工厂的一行代码，即可将所有涉及到的组件切换到另外一组主题实现。

1. 保证产品使用的一致性

抽象工厂模式对产品的使用进行了约束。客户端用到的组件全部出自同一工厂，必须是同一主题。这样在切换组件主题时不会出现漏网之鱼。

兔小白：客户端对工厂和产品实现了了依赖反转。全主题的产品替换也通过抽象工厂实现了开闭原则。真的是完美的模式。

熊小猫：其实谈不上完美。你想一想，如果需要增加新的组件，比如商品评价组件，这会造成每个工厂实现都需要添加商品评价组件。如果程序已经支持10种主题显示，那么就得改10个工厂！

兔小白：哇，这可是个灾难！

熊小猫：产品有两个维度的属性，一个维度是产品的品类，另一个是产品的主题。这是两个横纵交叉的维度。抽象工厂模式解决的是产品主题变更的灵活度。并不能解决产品品类变更的问题。

图：品类、主题

每种设计模式只能解决特定问题，并不是万能的。没有完美的设计模式，只有最适合当前场景的设计模式。

**当不同产品之间存在使用约束，并且同类产品又分为各种主题时，为了实现切换主题的灵活性和正确性，可以考虑使用抽象工厂模式。**

兔小白：例子中商品图片和介绍的显示主题需要一致。夜间主题和日间主题则是产品的不同主题。这个场景用抽象工厂模式太合适了！

今天终于把工厂模式全部学完了！简单工厂、工厂方法、抽象工厂，说实话，我现在有点晕。

## 5.4 简单工厂、工厂方法、抽象工厂比较

熊小猫：我来做个比较，帮助你理解记忆。我以家具厂为例子，从模式实现、适用场景来比较这三种工厂模式。

首先说说三种工厂的实现：

1. 简单工厂模式

简单工厂模式只对某一类产品进行抽象。工厂没有抽象。工厂根据参数来创建同一类产品的不同实现（实现同一接口的产品）。

例如生产椅子的家具厂，可以生产电脑椅、餐椅、摇椅等椅子类产品。但是它只能生产椅子类产品

1. 工厂方法模式

工厂方法模式对某一类产品和工厂都进行了抽象。抽象工厂定义生产方法。多个工厂实现生产方法，可以生产部分产品，也可以只生产一种产品。

例如椅子家具厂，定义为仅生产椅子。有三家符合该定义的家具厂。一家生产电脑椅，一家生产餐椅，另外一家生产摇椅。此时家具厂分工更专一，专业性更强。

1. 抽象工厂模式

抽象工厂模式对几类产品都进行抽象。在抽象工厂中定义了每个品类产品的生产行为。

家具厂被定义为全能家具厂，可以生产椅子、桌子、柜子等多品类家具，但是每家全能家具厂生产的家具风格一致。例如有两家全能家具厂，一家为中式家具厂，生产中式的椅子、桌子、柜子。另一家为欧式家具厂，生产欧式的椅子、桌子、柜子。

怕你记不住，我还编了个顺口溜来记忆：

图片：

简单工厂，同类全套一家全

工厂方法，职责精专不慌乱

抽象工厂，主题确定多类产

简单工厂负责所有同类产品的生产。工厂方法模式中的工厂，职责更为单一，仅负责同类的部分产品生产，几家工厂加一块才能实现所有同类产品的生产。抽象工厂模式中的工厂，负责一个主题下全类型产品的生产。

兔小白：学习三个晚上，被你最后浓缩成三句话，厉害！

熊小猫：哈哈，功不唐捐！没有这几晚的努力，你也听不懂这三句顺口溜呀！

咱们再来看看三种工厂模式的适用场景。

1. 简单工厂模式

简单工厂的优点是简单，但也能让程序符合开闭原则、单一职责、依赖倒转。如果是解决简单的同类对象创建问题，可以优先考虑。

缺点是工厂职责不够单一，并且没有抽象，造成工厂难以扩展。

1. 工厂方法模式

优点是通过抽象工厂的不同实现，将工厂的职责进一步单一化。当引入新的产品时，对已有工厂的影响很小，甚至没有。工厂方法适用的场景有：

（1）创建产品对象的逻辑比较复杂时，使用简单工厂会导致简单工厂非常复杂。可以用工厂方法模式，让多家工厂来分担产品创建的工作。每家工厂职责更为单一。

（2）客户端代码中，使用工厂大量创建产品对象，并且未来可能替换产品为另外的实现时，可以考虑使用工厂方法模式。

工厂方法不足之处是产品的增加会造成工厂同步增长。

1. 抽象工厂模式

抽象工厂模式的最大优点是同主题产品更换非常方便。只需要扩展新的工厂实现，然后在客户端代码中替换掉工厂即可。抽象工厂模式可以保证更换系列后，产品的一致性。不会出现漏更换，导致风格不统一，甚至相关对象无法使用的问题。

如果你的代码里，涉及到不同风格、不同系列、不同平台等类似同一主题的对象实现，这正是抽象工厂的用武之地。

抽象工厂模式的缺点是在增加产品时，每个工厂都需要进行修改。

兔小白：本想着今天早点结束，没想学习量这么大！我回去好好消化消化。

熊小猫：时间不早，我也该休息了，咱们明天再见！

兔小白：不过……你不是还有Bug要改？

熊小猫：啊，我完全忘了！你先走吧，我继续熬夜改Bug……