# 第5章 工厂品类要丰富——抽象工厂模式

兔小白：熊小猫，今天晚上一块去吃汉堡？

熊小猫：为什么又吃汉堡，咱能不能吃点别的？

兔小白：今晚要和你学习抽象工厂模式，我琢磨又少不了拿汉堡举例。

熊小猫：你是不是在内涵我？不吃汉堡，一样给你讲设计模式。

兔小白：哈哈，我知道你厉害。但咱们也得先吃饭去。

熊小猫：今晚去便利店买点关东煮对付一下吧。我晚上得解决一个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。

## 4.1 供应商不靠谱？直接换掉

兔小白：今天的关东煮味道不错，比上次的感觉有嚼劲，味道也更鲜美！

熊小猫：是呀，确实进步不少，我也吃出来了！可能是最近换了关东煮的供应商。

兔小白：质量不好，被换掉是迟早的事。生产关东煮原料的厂商多的是。

熊小猫：和软件开发一样，质量永远是重中之重！

哎，你想过为什么换掉供应商如此容易吗？如果关东煮的每种品类是不同供应商供货，换掉供应商就没这么容易了。

兔小白：如你所说，如果有10种不同的关东煮，那么就得换掉10家供应商，这就太麻烦了！但据我所知，关东煮厂商绝不会只生产单一品类，而是生产一系列的品类。所以只需要换一家供应商就可以了。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，一家供应商都能生产。所以想替换掉供应商就太容易了！直接一锅端！

兔小白：嗯，这就是抽象工厂模式！

熊小猫：呀！学会抢答了？

兔小白：哈哈！**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。**今天要讲抽象工厂嘛，我盲猜一下！

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。

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。虽然品类一样，但是无论质量还是口味都会有区别。便利店作为供应商的客户，只需要对接一家供应商。比如说缺鱼丸，就给这家供应商打电话订鱼丸。如果想换掉这家供应商，只需要打电话给另外的供应商，要订的还是鱼丸。

兔小白：如此听来，想换供应商，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。两个便利店，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，说明我在你的指导下，进步还不小。

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！

## 4.2 实现不同风格的显示组件

回到公司后，熊小猫给兔小白出提出了一个编程问题。

熊小猫：请听题，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，示意介绍页的展示过程。

图：需求展示

兔小白：这个简单，我快点写，争取今天别让你熬夜。看你的黑眼圈多明显!

熊小猫：我这是天生的！快去写代码！

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 String getAddress() {  
        return address;  
   }  
​  
    public void setAddress(String address) {  
        this.address = address;  
   }  
}

public class Text {  
    private String content;  
​  
    public String getContent() {  
        return content;  
   }  
​  
    public void setContent(String content) {  
        this.content = content;  
   }  
}

ProductDetailPage 类中的display方法，用于始化化商品详情页的产品图片和介绍并显示。

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

熊小猫：代码虽然很简单，但是很好的展示了初始化商品详情页的过程。不过新的需求来了！现在页面显示效果是白底黑字，夜间看会比较刺眼。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：这个好办，把Image和Text类抽象一层，写两种不同主题的的实现就可以了。

熊小猫：没错，抽象一层会带来相应的灵活度，因为客户端不再依赖具体的实现。赶紧动改吧！

二十分钟后，兔小白完成了代码修改。

兔小白：Image和Text类抽象为父类，增加主题theme属性。并提供一个有参的构造函数来设置主题。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，有LightText和DarkText。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，客户端ProductDetialPage代码如下：

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

是不是非常灵活？别说两种主题，十种主题都不在话下！

熊小猫：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：10种组件切换主题， ProductDetialPage类得改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，还得在设计上再下点功夫。今天咱们不是要学抽象工厂吗？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，就可以拿到同类型但完全不一样的关东煮。

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！你这个类比虽然很夸张，但还挺恰当！

十分钟后。

兔小白：你说过关东煮行业有一个沉淀下来的品类列表，每家工厂都会生产列表上面的产品。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();  
   }  
}

好比组建了两家关东煮的供应商，一家普通口味，一家麻辣口味！

ProductDetialPage中的Text和Image对象改为从工厂生产。

public class ProductDetailPage {  
    private Image image;  
    private Text introduction;  
    public void display(String ImageAddress,String Introduction){  
        IFactory widgetFactory = new DarkThemeWidgetFactory();  
​  
        image = widgetFactory.createImage();  
        image.setAddress(ImageAddress);  
​  
        introduction = widgetFactory.createText();  
        introduction.setContent(Introduction);  
​  
        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，使用日间组件工厂即可。

IFactory widgetFactory = new LightThemeWidgetFactory();

刚才你给我出难题，要换10种组件的风格。也是这一行代码搞定！

## 4.3 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！这次改造完，你已经用上了抽象工厂模式！你把这版代码的结构图画一下，咱们来分析看看。

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 ，每种组件也做了抽象。这样客户端代ProductDetialPage 就摆脱了对具体风格组件的依赖，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. 方便切换产品系列

这是抽象工厂最为显著的优势。客户端代码中仅需要修改创建工厂的一行代码，就可以将所有涉及的组件切换到其他主题。

1. 确保产品的一致性

对产品使用进行了约束。用到的组件全部出自同一工厂，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依赖反转。对全系列的产品替换实现了开闭原则。不过总觉得哪里有瑕疵……

熊小猫：你想一想，如果增加商品评价组件，每个工厂的实现都需要添加评价组件。如果有10种主题，那么就得改10个工厂。

兔小白：这可是灾难了！

熊小猫：产品有两个维度的属性，一是产品的品类，二是产品的系列。这是两个横纵交叉的维度。抽象工厂模式解决的是产品系列变更的灵活度。并不能解决产品品类变更的问题。

每种设计模式只能解决特定问题，并不是万能的。

回到抽象工厂模式。**当不同产品之间存在使用约束，并且存在不同系列同类产品，可以考虑使用抽象产品工厂。**

兔小白：商品图片和介绍的主题需要一致，这是不同产品间的约束。夜间主题和日间主题则是产品的不同系列。这个例子非常符合抽象工厂模式的使用场景！

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。

## 4.3 简单工厂、工厂方法、抽象工厂比较

熊小猫：我来用生产家具的工厂作为例子，从模式实现、适用场景来比较这三种模式。

首先说说三种工厂的实现：

1. 简单工厂

只对一类产品进行抽象。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）。

例如一家只生产椅子的家具厂，可以生产电脑椅、餐椅、摇椅等椅子类产品。

1. 工厂方法

对一类产品和工厂都进行了抽象。抽象的工厂定义了生产方法。不同的工厂实现生产方法，仅负责生产部分产品，甚至只生产一种产品。

例如椅子家具厂的定义是仅生产椅子的工厂。有三家符合该定义的椅子家具厂，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

对几类产品分别进行抽象。工厂同样进行抽象，定义每类产品的生产行为。

此时的家具厂定义为全能家具厂，可以生产椅子、桌子、柜子等家具。全能家具厂生产的家具风格统一。例如两家全能家具厂。其中一家为中式家具厂，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，高能产**

**工厂方法，职责精专，不慌乱**

**抽象工厂，多类一物，系列换**

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。抽象工厂中，一家工厂负责一个系列全类型产品生产。

兔小白：给你点赞，大诗人！学习了三个晚上，最后被你浓缩成了三句话，厉害！

熊小猫：哈哈，功不唐捐！没有这几晚的努力，告诉你这三句顺口溜，你也听不懂呀！

咱们再来看看三种工厂方式的适用场景。

1. 简单工厂

优点是简单，但同样也能让你的程序做到符合开闭原则、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，造成工厂很难扩展。

1. 工厂方法

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. 当在客户端代码中，使用工厂大量创建产品对象，并且产品对象在未来有可能性被替换为另外的实现时，可以使用工厂方法模式。简单工厂无法扩展工厂，很难灵活实现这个需求。工厂方法可以轻松扩展出工厂接口的多种实现，而客户端代码切换自己用到的所有产品时，只需要修改初始化工厂的一行代码即可。

工厂方法不足之处是产品的增加会造成工厂的增加，给代码带来了一定的复杂度。此外有时需要配合简单工厂使用，也就是昨天提到的“工厂的工厂”。此举也会提高代码的复杂度。

1. 抽象工厂
2. 抽象工厂最大的优点就是同系列产品的更换非常方便，只需要扩展新的工厂实现，然后客户端代码换掉工厂即可。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。

如果你要实现的功能，涉及到不同风格、不同主题、不同平台等类似同一系列的对象实现。正是抽象工厂的用武之地。

抽象工厂的缺点是在增加产品时，每个工厂都需要进行修改。

想说的都说完了，不过想用好还需要你在实践中多思考。

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……

图，兔小白睡大觉，熊小猫熬夜改bug。