抽象工厂

2020年2月24日 12:09

1. 概念

1.1 定义

抽象工厂模式，即Abstract Factory Pattern，提供一个创建一系列相关或相互依赖对象的接口（同一个产品族中的产品等级对象），而无须指定它们具体的类；具体的工厂负责实现具体的产品实例。

抽象工厂模式与工厂方法模式最大的区别：抽象工厂中每个工厂可以创建多种类（多个产品等级）的产品；而工厂方法每个工厂只能创建一类

1.2 主要作用

允许使用抽象的接口来创建一组相关产品，而不需要知道或关心实际生产出的具体产品是什么，这样就可以从具体产品中被解耦。

1.3 解决的问题

每个工厂只能创建一类产品，即工厂方法模式的缺点

2. 模式原理

2.1 UML类图

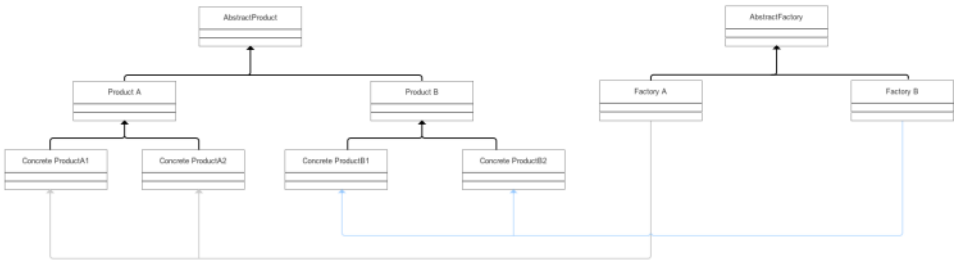

图2.1 抽象工厂UML结构图

2.2 模式组成

组成（角色）	关系	作用
抽象产品（Product）	具体产品的父类	描述具体产品的公共接口
具体产品（Concrete Product）	抽象产品的子类；工厂类创建的目标类	描述生产的具体产品
抽象工厂（Creator）	具体工厂的父类	描述具体工厂的公共接口
具体工厂（Concrete Creator）	抽象工厂的子类；被外界调用	描述具体工厂；实现FactoryMethod工厂方法创建产品的实例

2.3 产品等级结构与产品族概念

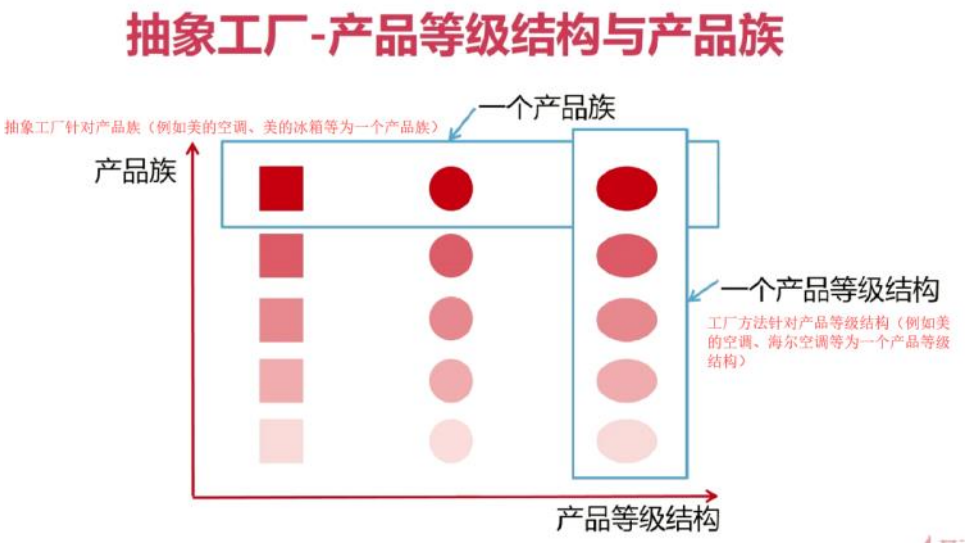

图2.1 产品等级结构与产品族概念图1

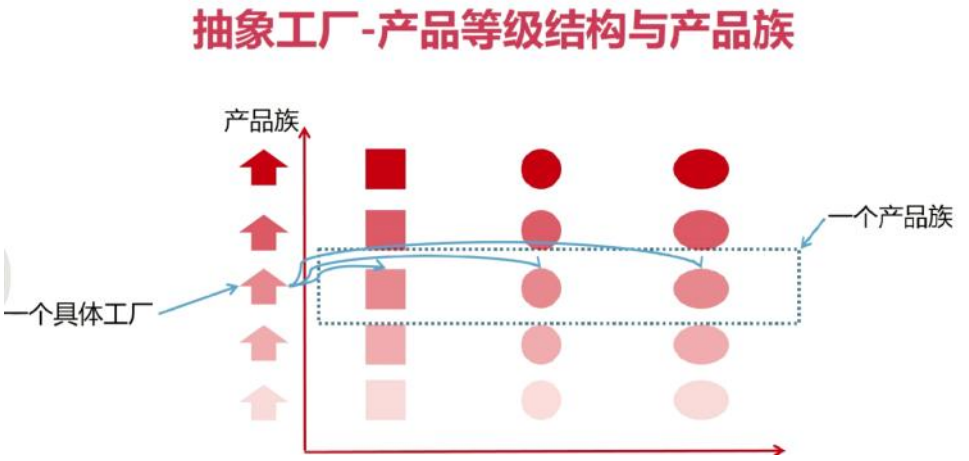

图2.3 产品等级结构与产品族概念图2

2.4 使用步骤

步骤1： 创建抽象工厂类，定义具体工厂的公共接口；

步骤2： 创建抽象产品类 （继承抽象产品族类），定义具体产品的公共接口；

步骤3： 创建具体产品类（继承抽象产品类） & 定义生产的具体产品；

步骤4：创建具体工厂类（继承抽象工厂类），定义创建对应具体产品实例的方法；

步骤5：客户端通过实例化具体的工厂类，并调用其创建不同目标产品的方法创建不同具体产品类的实例

3. 实例讲解

3.1 实例概况

背景：网站课程学习，每个课程都包含课程视频和课程手记，使用抽象工厂模式设计。例如JavaCourse是一个产品族，其中拥有两个产品等级JavaVideo、JavaArticle

3.2 使用步骤

步骤1： 创建抽象工厂类，定义具体工厂的公共接口

```
public interface CourseFactory {
    Video getVideo();
    Article getArticle();
}
```

步骤3： 创建抽象产品类 ， 定义具体产品的公共接口；

```
// Article抽象产品类
public abstract class Article {
    public abstract void produce();
}

// Video抽象产品类
public abstract class Video {
    public abstract void produce();
}
```

步骤4： 创建具体产品类（继承抽象产品类）， 定义生产的具体产品；

```
// JavaVideo具体产品类
public class JavaVideo extends Video {
    @Override
    public void produce() {
        System.out.println("录制Java视频");
    }
}
// JavaArticle具体产品类
public class JavaArticle extends Article {
    @Override
    public void produce() {
        System.out.println("编写Java手记");
    }
}
// PythonVideo具体产品类
public class PythonVideo extends Video {
    @Override
    public void produce() {
        System.out.println("录制Python视频");
    }
}
// PythonArticle具体产品类
public class PythonArticle  extends Article{
    @Override
    public void produce() {
        System.out.println("录制Python手记");
    }
}
```

步骤5：创建具体工厂类（继承抽象工厂类），定义创建对应具体产品实例的方法；

```
// Java产品产品族创建工厂 其中产品有video、article
public class JavaCourseFactory implements CourseFactory {
    @Override
    public Video getVideo() {
        return new JavaVideo();
    }
    @Override
    public Article getArticle() {
        return new JavaArticle();
    }
}

// Python产品产品族创建工厂 其中产品有video、article
public class PythonCourseFactory implements CourseFactory {
    @Override
    public Video getVideo() {
        return new PythonVideo();
    }
    @Override
    public Article getArticle() {
        return new PythonArticle();
    }
}
```

步骤6：客户端通过实例化具体的工厂类，并调用其创建不同目标产品的方法创建不同具体产品类的实例

```
public class AbstractFactoryPattern {
    public static void main(String[] args){
       CourseFactory courseFactory = new JavaCourseFactory();
         Video video = courseFactory.getVideo();
         Article article = courseFactory.getArticle();
         article.produce();
         video.produce();
}}
```

结果：

```
编写Java手记
```

录制Java视频

4. 优点

1. 降低耦合。抽象工厂模式将具体产品的创建延迟到具体工厂的子类中，这样将对象的创建封装起来，可以减少客户端与具体产品类之间的依赖，从而使系统耦合度低，这样更有利于后期的维护和扩展；
2. 更符合开-闭原则。新增一种产品族时，只需要增加相应的具体产品类和相应的工厂子类即可
3. 符合单一职责原则。每个具体工厂类只负责创建对应的产品
4. 不使用静态工厂方法，可以形成基于继承的等级结构。

5. 缺点

抽象工厂模式很难支持新种类产品（比如课程中新增了源码文件）的变化。这是因为抽象工厂接口中已经确定了可以被创建的产品集合，如果需要添加新产品，此时就必须去修改抽象工厂的接口，这样就涉及到抽象工厂类的以及所有子类的改变，这样也就违背了“开放—封闭”原则。对于新的产品族符合开-闭原则；对于新的产品种类不符合开-闭原则，这一特性称为开-闭原则的倾斜性。

6. 应用场景

1. 一个系统不要求依赖产品类实例如何被创建、组合和表达的表达，这点也是所有工厂模式应用的前提。
2. 这个系统有多个系列产品，而系统中只消费其中某一系列产品
3. 系统要求提供一个产品类的库，所有产品以同样的接口出现，客户端不需要依赖具体实现。

6. 参考资料

[1] https://www.jianshu.com/p/7deb64f902db