第2章 讨论

模式之间的关系

创建型模式之间存在竞争和互补关系。例如 抽象工厂 和 原型模式 在一些场景下能相互替代。另一些场景下抽象工厂持有一些 原型 用于生产产品。 建造者 能够采用其他创建型模式来生产产品的组件。 抽象工厂 、 建造者 和 原型 可以采用 单例 来实现。

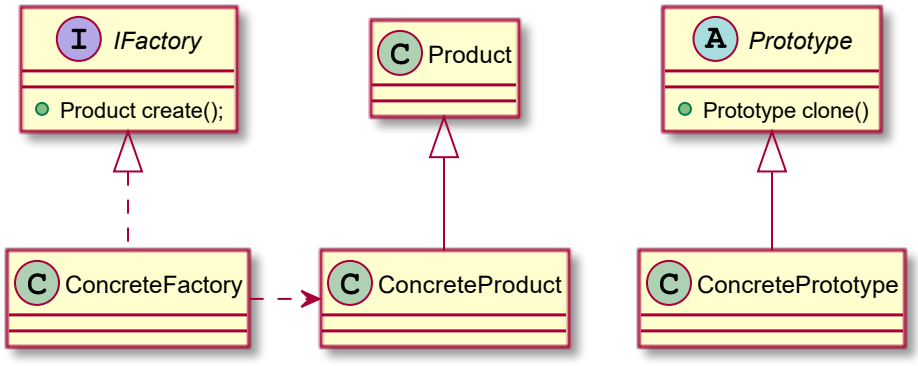
抽象工厂与工厂方法

结构上，抽象工厂定义了多个工厂方法，用于生产一个产品簇的产品，每个产品生产可以采用工厂方法模式实现。

> Abstract Factory classes are often implemented with factory methods.

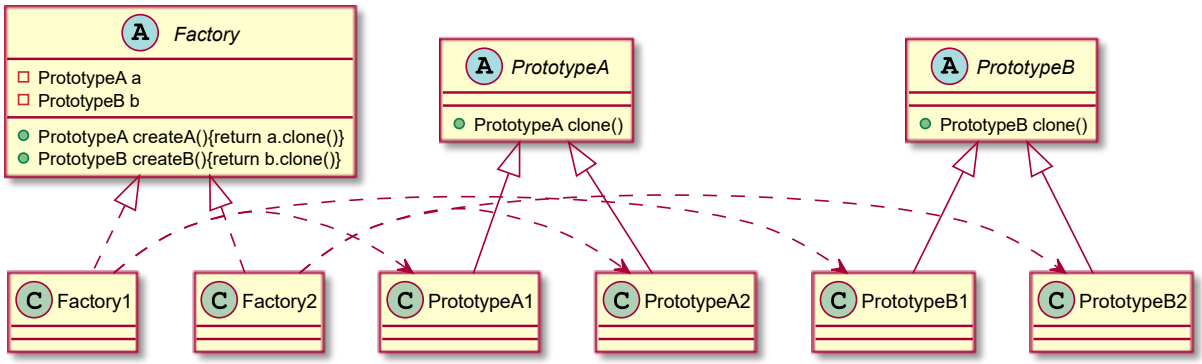
工厂方法与原型模式

结构上，原型模式可以看作工厂方法模式中产品类与工厂类合并后的特殊形式。 `IFactory` 和 `Product` 对应 `Prototype` 类， `ConcreteFactory` 类与 `ConcreteProduct` 对应 `ConcretePrototype` 类， `create()` 方法对应 `clone()` 方法。原型模式下产品类型的扩展不需要定义子类化工厂，但在创建对象时需要先初始化一个原型对象。



抽象工厂与原型模式

一般情况下抽象工厂采用工厂方法模式来实现，但也能用原型模式来实现。与采用工厂方法来实现的主要区别为，抽象工厂类中需要引用产品类，实例化产品类作为原型实现其他产品的生产。

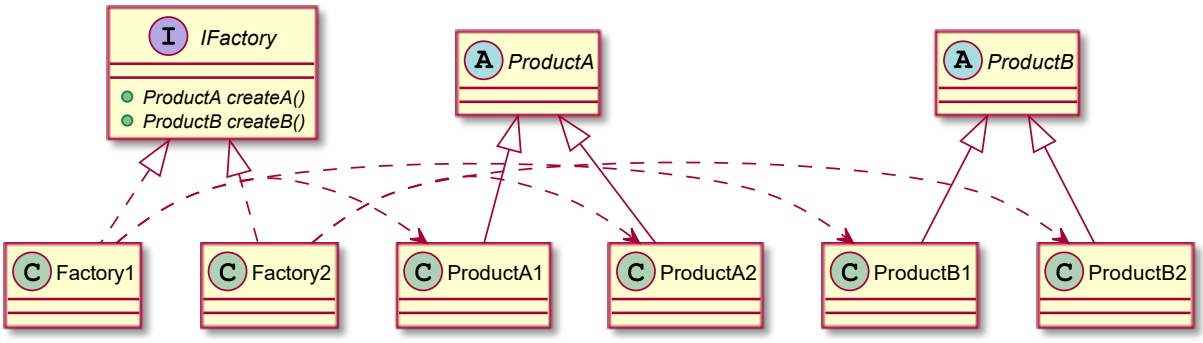


> If many product families are possible, the concrete factory can be implemented using the Prototype pattern. The concrete factory is initialized with a prototypical instance of each product in the family, and it creates a new product by cloning its prototype. The Prototype-based approach eliminates the need for a new concrete factory class for each new product family.

基于单例模式的工厂

一般情况一个产品簇只需要一个具体工厂来生产，因此可以将具体工厂类设计为单例模式。

> Factories as singletons. An application typically needs only one instance of a ConcreteFactory per product family. So it's usually best implemented as a Singleton.



实现代码：

```
// 工厂接口
public interface IFactory {
    public abstract ProductA createA();
    public abstract ProductB createB();
}
// 其中一个具体工厂实现，另一个工厂实现方式相同
public class Factory1 implements IFactory{
    private static final IFactory instance = new Factory1();
    public static IFactory getInstance() {
        return instance;
    }
    @Override
    public ProductA createA() {
        return new ProductA1();
    }
    @Override
    public ProductB createB() {
        return new ProductB1();
    }
}
// 客户端代码
public class Test {
    public static void main(String[] args) {
        IFactory f1 = Factory1.getInstance();
        IFactory f2 = Factory2.getInstance();
        System.out.println(f1.createA());
        System.out.println(f1.createB());
        System.out.println(f2.createA());
        System.out.println(f2.createB());
    }
}
```

抽象工厂与建造者模式

抽象工厂与建造者类似都用于创建复杂对象。主要区别在于建造者将复杂对象的建造流程分解，一步一步创建对象。抽象工厂注重产品簇的统一创建。