三种工厂模式

- 工厂模式（Factory Pattern）是一种创建型设计模式，它提供了一种创建对象的接口，而不是通过具体类来实例化对象；工厂模式可以将对象的创建过程封装起来，使代码更具有灵活性和可扩展性。

简单工厂模式

概念

- 简单工厂模式（Simple Factory Pattern）： 通过一个工厂类来决定创建哪种具体类的实例；这个工厂类通常是静态类，提供一个静态方法，根据传入的参数创建相应的对象。
- 构成
  - 产品接口（Product Interface）：包含产品的生产的行为。
  - 具体产品类（Concrete Product Class）：实现产品类接口，具体产品对象。
  - 工厂类（Factory Class）：用于生成不同产品对象。

实例

- 产品接口

```
1 public interface IProduct
2 {
3     void DoSomething();
4 }
```

- 具体产品类

```
1 public class ProductA : IProduct
2 {
3     public void DoSomething()
4     {
5         Console.WriteLine("ProductA is doing something.");
6     }
7 }
8
9 public class ProductB : IProduct
```

```
{
    public void DoSomething()
    {
        Console.WriteLine("ProductB is doing something.");
    }
}
```

- 静态工厂类

```
public static class Factory
{
    public static IProduct CreateProduct(string type)
    {
    switch (type)
    {
        case "A":
            return new ProductA();
        case "B":
            return new ProductB();
        default:
            throw new ArgumentException("Invalid type");
        }
    }
}
```

- 使用示例

```
class Program
{
    static void Main(string[] args)
    {
        IProduct productA = Factory.CreateProduct("A");
        productA.DoSomething();

        IProduct productB = Factory.CreateProduct("B");
        productB.DoSomething();
    }
}
```

工厂方法模式

概念

- 工厂方法模式（Factory Method Pattern）：定义了一个创建对象的接口，但由子类决定实例化哪一个类。工厂方法模式使得一个类的实例化延迟到其子类。
- 构成
  - 产品接口（Product Interface）：定义产品的接口。
  - 具体产品类（Concrete Product Class）：实现产品接口的具体类。
  - 工厂接口（Factory Interface）：定义创建产品对象的接口。
  - 具体工厂类（Concrete Factory Class）：实现工厂接口的具体类。

实例

- 产品接口

```
1 public interface IProduct
2 {
3     void DoSomething();
4 }
```

- 具体产品类

```
1  public class ProductA : IProduct
2  {
3      public void DoSomething()
4      {
5          Console.WriteLine("ProductA is doing something.");
6      }
7  }
8
9  public class ProductB : IProduct
10 {
11     public void DoSomething()
12     {
13         Console.WriteLine("ProductB is doing something.");
14     }
15 }
```

- 工厂接口

```
public interface IFactory
{
    IProduct CreateProduct();
}
```

- 具体工厂类

```
public class FactoryA : IFactory
{
    public IProduct CreateProduct()
    {
        return new ProductA();
    }
}
```

- 使用示例

```
class Program
{
    static void Main(string[] args)
    {
        // 创建具体工厂类"A"专门生成产品"A"
        IFactory factoryA = new FactoryA();
        IProduct productA = factoryA.CreateProduct();
        productA.DoSomething();

        // 创建具体工厂类"B"专门生成产品"B"
        IFactory factoryB = new FactoryB();
        IProduct productB = factoryB.CreateProduct();
        productB.DoSomething();
    }
}
```

抽象工厂模式

概念

- 抽象工厂模式（Abstract Factory Pattern）： 提供一个接口，用于创建相关或依赖对象的家族，而无需明确指定具体类。通过使用抽象工厂模式，一个类可以实例化一组相关对象，而不需要知道它们的具体类。

- 构成
  - 抽象产品接口（Abstract Product Interface）：为每种产品声明一个接口。
  - 具体产品（Concrete Product）：实现抽象产品接口的具体类。
  - 抽象工厂接口（Abstract Factory Interface）：声明了一组用于创建一族产品的方法。
  - 具体工厂（Concrete Factory）：实现抽象工厂的接口，负责创建具体产品对象。

实例

- 产品接口

```
public interface IProductA
{
    void DoSomethingA();
}

public interface IProductB
{
    void DoSomethingB();
}
```

- 具体产品

```
// 标准版的产品A
public class StdProductA: IProductA
{
    public void DoSomethingA()
    {
        Console.WriteLine("StdProductA is doing something.");
    }
}

// 升级版的产品A
public class ProProductA: IProductA
{
    public void DoSomethingA()
    {
        Console.WriteLine("ProProductA is doing something.");
    }
}

// 标准版的产品B
public class StdProductB: IProductB
```

```
{
    public void DoSomethingB()
    {
        Console.WriteLine("StdProductB is doing something.");
    }
}

// 升级版的产品B
public class ProProductB: IProductB
{
    public void DoSomethingB()
    {
        Console.WriteLine("ProProductB is doing something.");
    }
}
```

- 工厂接口

```
public interface IFactory
{
    IProductA CreateProductA();
    IProductB CreateProductB();
}
```

- 具体工厂

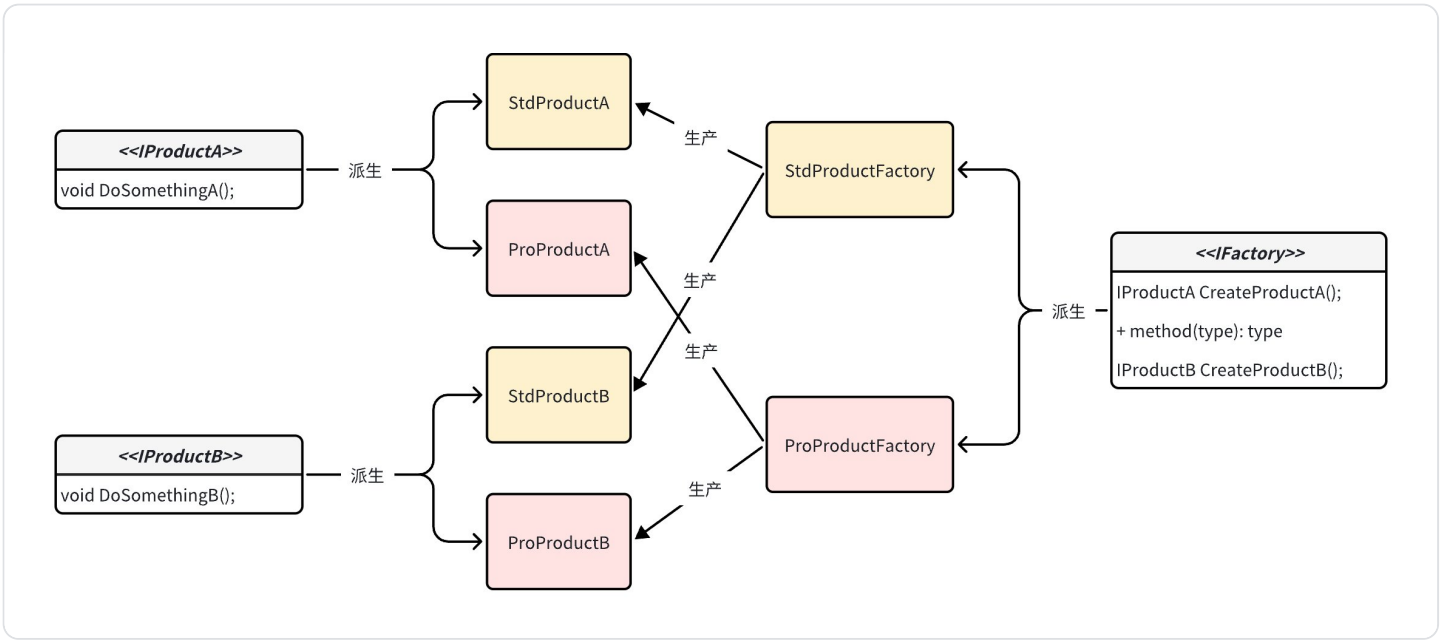
```
// 标准版产品生产工厂
public class StdProductFactory : IFactory
{
    public IProductA CreateProductA()
    {
        return new StdProductA();
    }

    public IProductB CreateProductB()
    {
        return new StdProductB();
    }
}

// 升级版产品生产工厂
public class ProProductFactory: IFactory
{
```

```
    public IProductA CreateProductA()
    {
        return new ProProductA();
    }

    public IProductB CreateProductB()
    {
        return new ProProductB();
    }
}
```

- 使用示例

```
class Program
{
    static void Main(string[] args)
    {
        // 创建一个用于生产标准版产品的工厂
        IFactory stdFactory = new StdProductFactory();
        IProductA stdProductA = stdFactory.CreateProductA();
        IProductB stdProductB = stdFactory.CreateProductB();
        stdProductA.DoSomethingA();
        stdProductB.DoSomethingB();

        // 创建一个用于生产升级版产品的工厂
        IFactory proFactory = new ProProductFactory();
        IProductA proProductA = proFactory.CreateProductA();
        IProductB proProductB = proFactory.CreateProductB();
        proProductA.DoSomethingA();
        proProductB.DoSomethingB();
    }
}
```

- UML类图



总结

- 这三种工厂模式适用于不同的场景：当对象的创建逻辑简单且不需要大量的子类时，工厂模式是一个不错的选择；当系统需要通过引入新的子类来扩展对象的创建过程时，工厂方法模式非常适用；当需要创建一组相关或相互依赖的对象时，抽象工厂模式是一个理想的选择。