工厂方法模式

当我们使用 new 来实例化一个具体类时，我们需要知道创建该对象的一些信息，比如要传入什么参数，这样就导致代码绑定了具体类，这其实是在针对实现编程，而面对实现编程会导致代码更脆弱、缺乏弹性。
当我们有一组具体类，并且究竟实例化那个具体类要在运行时由一些条件来决定时，我们常常会写出如下代码：

```
Product product;
if (conf1)
    product = new ConcreteProduct1();
else if (conf2)
    product = new ConcreteProduct2();
else if (conf3)
    product = new ConcreteProduct3();
```

此时，一旦有变化或者扩展，就必须重新检查和修改这段代码，并且如果有多处位置利用这种方式构造对象，那我们必须检查所有位置，造成系统的维护和更新十分困难，而且极易出错，也违反了开闭原则。
但是，在 Java 中只提供了 new 关键字来创建对象，这是 Java 基础。其实，真正造成这种问题的原因是我们没有将变化的部分提取出来。实例化具体类的代码是变化的部分，我们应该将这部分从应用中抽出，并加以封装，使它们不会干扰应用的其他部分。

1、简单工厂模式

简单工厂模式又称为静态工厂模式，将实例化具体类的部分封装到一个工厂类的静态方法中，并通过参数（可以为约束字符串或者 Class 等）来判断究竟创建那个实例。此时，若需要变化或者扩展就只需要修改这一处位置即可。
简单工厂模式中的工厂类：

```
class Factory
{
    public static Product createProduct(String type)
    {
        Product product;
        if (type.equals("tyoe1"))
            product = new ConcreteProduct1();
        else if (type.equals("tyoe2"))
            product = new ConcreteProduct2();
        else if (type.equals("tyoe3"))
            product = new ConcreteProduct3();
        return product;
    }
```

```
}
```

这种创建方式比较简单，较为常用，其缺点是工厂类的不易扩展，不符合开闭原则，当有变化时还是需要修改这部分代码。

2、工厂方法模式

简单工厂模式实际上并不是一个设计模式，反而比较像是一种编程习惯，它是工厂方法模式的弱化。工厂方法模式克服了简单工厂不易扩展的缺点，它提供了一个抽象的工厂类，在增加产品类的情况下，只要适当的修改具体的工厂类或者扩展一个工厂类，就可以完成“拥抱变化”。并且，它同样屏蔽了产品类，调用者无需关心产品类的具体的创建过程，也无需知道产品类的实现是否发生变化，只关心产品的接口，只要接口不变，调用者就不用发生变化，降低了模块间的耦合。

工厂方法模式：定义了一个创建对象的接口，但由子类决定要实例化哪一个类。工厂方法让类把实例化推迟到子类。

工厂方法模式符合迪米特法则，只需要知道产品的抽象，而不关心实现；也符合依赖倒置原则，只依赖产品的抽象。

工厂方法模式的通用代码为：

```
//抽象产品类
abstract class Product
{
    public abstract void func();
}
//具体产品类
class ConcreteProduct1 extends Product
{
    public void func()
    {
        System.out.println("create concreteProduct1");
    }
}
class ConcreteProduct2 extends Product
{
    public void func()
    {
        System.out.println("create concreteProduct2");
    }
}
//抽象工厂类
abstract class Factory
{
    public abstract Product createProduct();
}
//具体工厂类
```

```
class ConcreteFactory1 extends Factory
{
    public Product createProduct()
    {
        return new ConcreteProduct1();
    }
}
class ConcreteFactory2 extends Factory
{
    public Product createProduct()
    {
        return new ConcreteProduct2();
    }
}
//场景类
public class Client
{
    public static void main(String[] args)
    {
        Product p1 = new ConcreteFactory1().createProduct();
        p1.func(); // create concreteProduct1
        Product p2 = new ConcreteFactory2().createProduct();
        p2.func(); // create concreteProduct2
    }
}
```

3、我的理解

工厂将调用者和产品分离，使调用者无需关心产品的具体实例化过程，使两个模块解耦，产品类发生变化或者扩展时不会对调用者造成影响，只需要在工厂这一个地方进行修改或扩展即可，使系统根据容易维护。

当一个产品有较多属性时，构造的时候会根据传入参数的不同而产生不同的产品，此时使用工厂方法模式，将产品的创建进行封装，根据参数的不同搭配建立不同的具体工厂，这样的话，调用者只需要按照所需产品种类调用对应的工厂创建对象即可，而无需再了解创建产品需要哪些参数。