factory模式【不主动去 new 来创建实例（否则如果实例变更，所有调用的地方都要修改）】

讲一个我平时面试的一个段子，我一般面试设计模式是必问，这决定了会不会写出屎一样的代码

100个人，90个人，90%比例，会告诉你，他说，我就会工厂模式和单例模式，真是一个笑话，身为一个工作几年的程序员来说

public **interface Product** {

   void operation();

}

public class **ProductImpl implements Product** {

   public void operation() {

       System.out.println("简单的接口实现");

  }

}

public class **Factory** {

   public static Product createProduct() {

       return new ProductImpl();

  }

}

public class Client {

   public static void main(String[] args) {

       Product product = Factory.createProduct();

       product.operation();

  }

}

工厂模式的核心思想，其实就是不要自己在代码里手动new一个实现类对象出来，因为那样的话，调用方就不是面向接口编程了，你还得自己去care实现了。

我们设想一下，假设有N个client都new了一个ProductImpl出来，结果，到了后面某一天，你要把ProductImpl换成ProductNewImpl了，完全换一套实现逻辑，这个时候就完蛋了。你需要在N个client里，都修改new ProductImpl()这个方法，简直是一场灾难啊。尤其如果调用你的类的是别人呢？别人还得来care这个事情？

所以一般都是用工厂的思想来提供所有实现类实例，然后**调用方面向接口来编程即可，接口不变，调用方代码不用变**

而在真正的企业开发中，现在我们都很少自己做工厂实现了，因为有spring，spring的容器其实本质上就是个大工厂的概念，这个工厂里什么对象都有，你需要什么，自己去找spring要就好了，非常方便。所以我们对工厂模式的体验，就在这个阶段，通过spring的容器使用来给大家实践。

spring会基于xml或者是注解，根据一些配置，去创建出来所有我们需要的这个类的实例，此时**spring就相当于是个工厂【用声明取代new创建】**

@Service

public class UserServiceImpl implemens UserService {

}

public class UserController {

**@Autowired**

private UserService userService;

}

我们在项目中，大量的用spring ioc这块功能，其实本身就是在实践工厂模式

如果一定要自己做工厂实现，那么一般就是在**封装底层框架或者中间件客户端的时候，可能需要自己用到工厂模式**

在电商项目中，我们另外初步定的是，会跟构造器模式结合起来给大家演示如何使用，还有一些别的模式，结合业务去给大家演示，自己做工厂模式怎么去做

853769620