首先，工厂模式是干什么的？简单的说就是用来创建对象的。

已经有了非常直接让我new Xx();为何还需要再搞一个工厂模式去创建对象呢？我需要什么对象，我直接去new就可以了啊！对于小型的项目或者说今后不需要维护的、已经很明确就那么一个使用场景的项目，确实直接new就可以了，使用工厂模式反而使代码增多，增加工作量。

但是对于大型的项目，今后可能会不断的增加功能，增加代码，那么使用工厂模式是必不可少的。比如额esight的框架程序，它开始某个功能只是给服务器在使用的，那么就是需要服务器的某一个对象去做某种处理；但是后来又来了个存储，要做的动作其实和服务器差不多，只不过现在这里需要的是存储的对象。那么是不是又要重新去搞个框架代码呢？因为直接的代码是写死的，没法改了。

工厂模式就是解决这种问题的。我写的代码这里是要用某一个对象，但是我不需要具体的某个对象，只要是这个类型的对象就可以了。至于真正使用的时候用的时候是哪个对象，那么就要看你给我的是哪个对象了。

再一个假如某个对象的初始化比较复杂，需要传入各种参数，调用各种方法才能完成，那么在每次使用到这个对象时，都需要做这些动作。然后工厂就出现了，把组装初始化对象的动作都交给他，再使用这种对象时，直接从工厂拿就行。

总而言之，工厂模式有两种好处：1就是为了更好的对代码进行扩展；2将复杂的初始化对象的动作进行封装，只需要写一次就行。

再提一句，spring的ioc其实根本原理就是工厂模式。

1. 简单工厂模式：

这种工厂只生产一个类型的对象，且种类是固定的。代码层面其实就是用接口规定出对象类型，然后有固定的几个类去实现这个接口。简单工厂要做的就是利用switch case语句根据传来的参数确定生产哪个对象并返回该对象。

简单工厂模式是工厂模式的基础。但是可以看到它的缺点很明显，只能生产固定的几个对象，如果想生产新产品，那么久需要改造工厂（增加case）了。

1. 工厂模式：

工厂模式是在简单工厂模式基础上修改而来的。可以认为是设立了一个总厂（抽象类或者接口），然后可以将共同的一些方法写在抽象类里，各自生产具体对象的方法写成抽象方法。 具体生产对象的厂实现这个抽象类。然后只需要实现抽象方法，每个厂就能负责生产一种（或者比较类似的几种（工厂模式于简单模式的联合））对象，如果想要生产新类型的对象，需要建一个新的厂。

可以看到工厂模式是对简单工厂模式的的一种扩建。但其实本质还是没有改变的。只不过能生产的对象增多了而已

1. 抽象工厂模式

工厂模式中提供了一个生产的抽象方法，也就是说它的实现类工厂只能生产一个等级的产品结构的产品，比如他只能生产汽车，这个汽车可以是2.0的也可以是2.4的。

而抽象工厂模式提供了许多生产的抽象方法，也就是说的实现类工厂能够生产许多个等级的产品结构的产品。比如他既可以生产汽车，也可以生产摩托。抽象工厂是对工厂模式的一种扩展。

1. 工厂模式+反射

可以看到，其实在工厂模式中还是有许多缺点的。生产新的产品还是不可避免的要去对工厂进行修改等。但是加上反射机制，就会使代码的灵活性再次提高，可以更好的进行扩展。