# 简单工厂模式

## 一、基本概念 **1、定义：**

##### 定义一个工厂类，他可以根据参数的不同返回不同类的实例，被创建的实例通常都具有共同的父类。

##### 简单工厂模式中用于被创建实例的方法通常为静态(static)方法,因此简单工厂模式又被成为静态工厂方法(Static Factory Method)

### 2、组成环节：

##### 1、Factory(工厂):核心部分，负责实现创建所有产品的内部逻辑，工厂类可以被外界直接调用，创建所需对象

##### 工厂类是整个模式的关键.包含了必要的逻辑判断,根据外界给定的信息,决定究竟应该创建哪个具体类的对象.通过使用工厂类,外界可以从直接创建具体产品对象的尴尬局面摆脱出来,仅仅需要负责“消费”对象就可以了

##### 2、Product(抽象类产品)：工厂类所创建的所有对象的父类，封装了产品对象的公共方法，所有的具体产品为其子类对象

##### 3、ProductA、ProductB…(具体产品)：简单工厂模式的创建目标，所有被创建的对象都是某个具体类的实例。它要实现抽象产品中声明的抽象方法

### 3、特征

##### 1、简单工厂模式的核心思想就是：有一个专门的类（工厂类）来负责创建实例的过程。

##### 把产品看着是一系列的类的集合，这些类是由某个抽象类或者接口派生出来的一个对象树。而工厂类用来产生一个合适的对象来满足客户的要求。

##### 2、如果简单工厂模式所涉及到的具体产品之间没有共同的逻辑，那么我们就可以使用接口来扮演抽象产品的角色；

##### 由于对象的创建过程是我们不需要去关心的，而我们注重的是对象的实际操作，所以，我们需要分离对象的创建和操作两部分，如此，方便后期的程序扩展和维护。

##### 3、如果具体产品之间有功能的逻辑或，我们就必须把这些共同的东西提取出来，放在一个抽象类中，然后让具体产品继承抽象类。

##### 4.在程序中，需要创建的对象很多，导致对象的new操作多且杂时，需要使用简单工厂模式；

##### 缺点：

##### 5、工厂类集中了所有实例的创建逻辑，违反了高内聚责任分配原则，将全部创建逻辑集中到了一个工厂类中；它所能创建的类只能是事先考虑到的，如果需要添加新的类，则就需要改变工厂类

##### 6、当系统中的具体产品类不断增多时候，可能会出现要求工厂类根据不同条件创建不同实例的需求．这种对条件的判断和对具体产品类型的判断交错在一起，很难避免模块功能的蔓延

##### 场景：

##### 1、工厂类负责创建的对象比较少；

##### 2、客户只知道传入工厂类的参数，对于如何创建对象（逻辑）不关心；

## 二、go语言中实现

##### 1、首先定义一个factory类型

type factory struct{}

##### 2、在定义两个具体类型的factory

type T1 struct{}  
type T2 struct{}

##### 3、定义一个统一product的接口，并定义其实现的方法。

type prodcut interface {  
 Do()  
}

##### 4、对具体的工厂方法进行do的实现。

func (t T1) Do() {  
 fmt.Printf("T1 do.\n")  
}  
  
func (t T2) Do() {  
 fmt.Printf("T2 do.\n")  
}

##### 5、通过抽象的工厂查找具体方法。

func (t factory) Check(name string) prodcut {  
 switch name {  
 case *T1NAME*:  
 return T1{}  
 case *T2NAME*:  
 return T2{}  
 }  
 return nil  
}

##### 6、开始进行方法的调用。

func main() {  
 var f factory  
 f.Check(*T1NAME*).Do()  
 f.Check(*T2NAME*).Do()  
}

##### 7、结果的输出。

T1 do.

T2 do.