# 建造者模式概述

## 概述

建造者模式关注如何一步一步地创建一个对象，不同的具体建造者可以定义不同的创建过程，各个具体建造者相互独立，且更换建造者或增加新的建造者都非常方便，系统具有较好的可扩展性。

在建造者模式中，要创建的对象也叫的复杂对象，其实就是指包含多个属性的对象。

## 结构

建造者模式包含以下4个角色：



1. Builder（抽象建造者）：通常是一个抽象类或接口，绝大多情况下是将其定义为抽象类，它为创建一个对象的各个部件（属性）定义抽象方法；在该角色中，一般定义了两类方法，一类方法是build属性名( )，它们是抽象方法，让具体建造者覆写，在具体建造者的这些方法中，创建对象的各个部件（属性）；另一类方法是getResult( )，用于返回要创建的对象。
2. ConcreteBuilder（具体建造者）：是抽象建造者的子类，覆写抽象建造者中定义的抽象方法后，在这些方法中，用于构建对象各个部件（属性），还可以提供一个方法返回创建好的复杂产品（对象）。
3. Product（产品）：一个普通类，它是被构建的复杂对象，也就是建造者模式最终要创建的对象，包含多个组成部件（属性），具体建造者会定义产品的创建过程。
4. Director（指挥者）：一个普通类，指挥者负责安排复杂对象的各个部件（属性）的建造次序，指挥会者和抽象建造者之间存在关联关系，在指挥者类中可以注入一个抽象建造者类型的对象，它提供了一个建造方法construct( )，在该方法中调用建造者对象中构建部件（属性）的方法，完成复杂对象的建造，最后返回一个产品对象。

# 实现

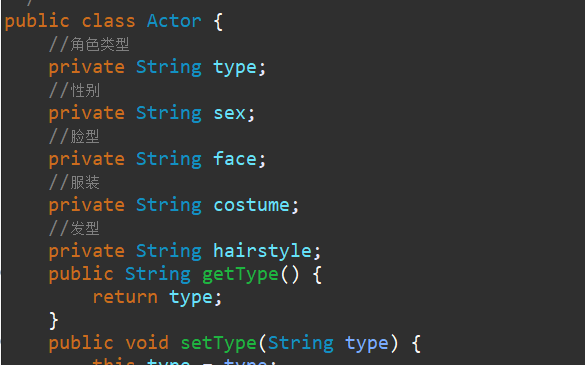
## 实例

实现说明：某游戏软件公司决定开发一款基于角色扮演的多人在线网络游戏，玩家可以在游戏中扮演虚拟世界中的一个特定角色，角色根据不同的游戏情节和统计数据（例如力量、魔法、技能等）具有不同的能力，角色也会随着不断升级而拥有更加强大的能力。作为一个该游戏的一个重要组成部分，需要对游戏角色进行设计，而且随着该游戏的升级将不断增加新的角色。通过分析发现，游戏角色是一个复杂对象（有属性的对象），它包含性别、脸型等多个组成部分（属性），不同类型的游戏角色，其性别、脸型、服装、发现等外部特征都有所差异，例如“天使”拥有美丽的面容和披肩的长发，并身穿一袭白裙；而“恶魔”极其丑陋，留着光头并穿一件刺眼的黑衣。无论是何种造型的游戏角色，它的创建步骤都大同小异，都需要逐步创建其组成部分，再将各组成部分装配成一个完整的游戏角色。

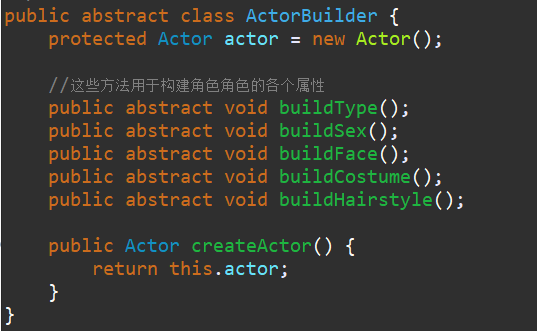


使用建造者模式完成：

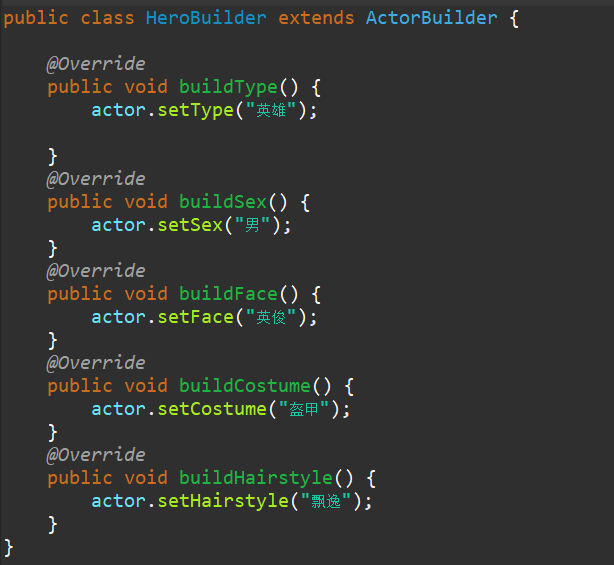
1. Actor（角色类）：充当产品，也就是建造者模式最终要创建的对象



1. ActorBuilder（角色创建者）：充当抽象建造者



1. HeroBuilder（英雄角色创建者）：抽象建造者的派生类，充当具体建造者，用于创建英雄这种类型的角色



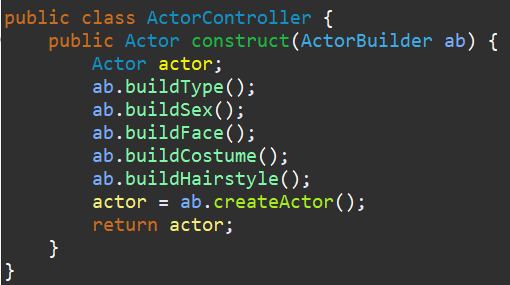
1. AngelBuilder（天使角色创建者）：抽象建造者的派生类，充当具体建造者，用于创建天使这种类型的角色



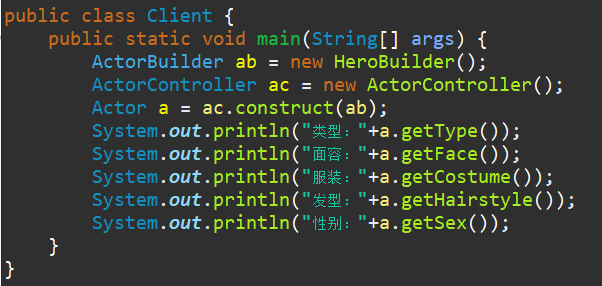
1. DevilBuilder（恶魔角色创建者）：抽象建造者的派生类，充当具体建造者，用于创建恶魔类型的角色



1. ActorBuilderController（角色控制器）：充当指挥者



1. 编写客户端类，测试建造者模式

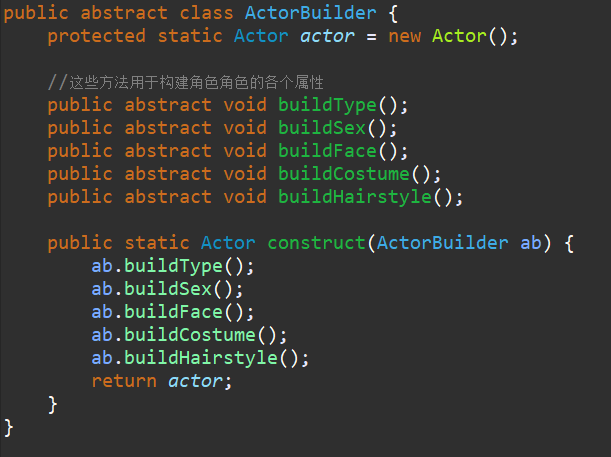


## 关于指挥者

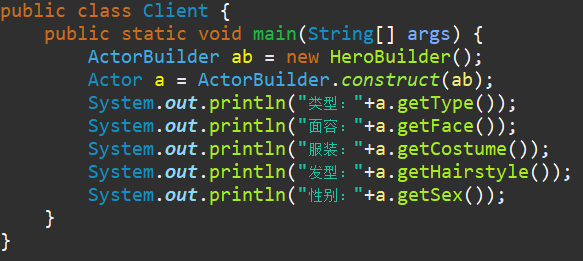
指挥者类Director是建造者模式的重要组成部分，简单的Director类用于指导具体建造者如何构建产品，它按一定次序调用具体建造者中build属性名( )方法，控制调用的先后次序，并向客户端返回一个完整的产品对象。下面讨论几种Director的变化形式。

1. 省略Director

在有些情况下，为了简化系统结构，可以将Director和抽象建造者进行合并，在抽象建造者中提供逐步构建复杂产品对象的construct( )方法。由于抽象建造者通常定义为抽象类，因此可以将construct( )方法定义为静态方法，以便客户端能够直接调用。如果将上面游戏角色实现例子中的指挥者类ActorController省略，在抽象建造者中的代码修改如下：



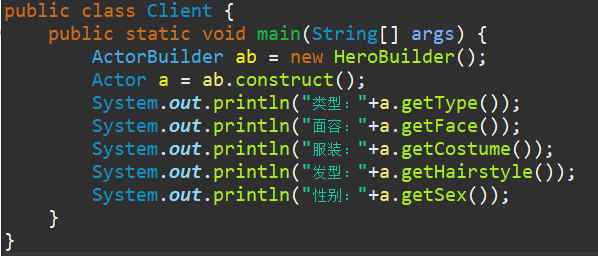
客户端测试类代码修改如下：



除此之外，还有一种更简单的处理方法，可以将construct( )方法中的参数去掉，直接在construct( )方法中通过当前对象调用build属性名( )方法，如下：



客户端测试类代码如下：



以上两种对Director类的省略方法都不影响系统的灵活性和可扩展性，同时还简化了系统结构，但加重了抽象建造者类的职责，如果construct( )方法较为复杂，待构建产品的组成部分很多，建议还是将construct( )方法单独封装在Director中，这样更好。

# 优缺点及适用环境

## 优点

建造者模式的核心在于如何一步步构建一个包含多个属性的对象，适用相同的构建过程构建不同的产品。建造者模式的优点主要如下：

1. 在建造者模式中，客户端不必知道产品（即对象）内部组成的细节，将产品本身与产品的创建过程解耦，使得相同的创建过程可以创建不同的产品对象。
2. 每一个具体建造者都相对独立，而与其他的具体建造者无关，因此可以很方便地替换具体建造者模式或增加新的具体建造者，用户使用不同的具体建造者即可得到不同的产品对象。由于指挥者类针对抽象建造者编程，增加新的具体建造者无需修改原有类库的代码，系统扩展方便。
3. 可以更加精细地控制产品的创建过程，将复杂产品的创建步骤分解在不同的方法中，使得创建过程更加清晰，也更方便使用程序来控制创建过程。

## 缺点

建造者模式的缺点主要如下：

1. 建造者模式所创建的产品一般具有较多的共同点，其组成部分相似，如果产品之间的差异性很大，例如很多组成部分都不相同，不适合使用建造者，因此其使用范围受到一定限制。
2. 如果产品的内部变化复杂，可能会导致需要定义很多具体建造类来实现这种变化，导致系统变得很庞大，增加系统的理解难度和运行成本。

## 适用环境

在以下情况下可以考虑适用建造者模式：

1. 需要生成的产品对象有复杂的内部结构，包含多个属性，并且属性间存在依赖，或需要指定各属性的生成顺序；
2. 对象的创建过程独立于创建该对象的类，在建造者模式中通过引入指挥类将创建过程封装在指挥者类中，而不在建造者类和客户端（即需要使用该对象产品的类）中。
3. 隔离复杂对象的创建和使用，并使得相同的创建过程可以创建不同的产品。