Ioc（Inverse of Control），按照英文全拼解读意思是控制反转，它是Spring的内核，AOP、声明式事务等功能都是基于它的。它涉及很多问题的考量，包括代码解耦、设计模式、代码优化等等。

如何理解Ioc概念？

我这里就借花献佛了，套用一个实例来解读。

在电影《墨攻》中，饰演革离的刘德华说话：

代码清单 4-1　MoAttack：直接使用演员编排剧本

```
public class MoAttack {
  public void cityGateAsk(){

      //①演员直接侵入剧本
      LiuDeHua ldh = new LiuDeHua();
      ldh.responseAsk("墨者革离！");
  }
}
```

我们会发现，以上剧本在①处，作为具体角色饰演者的刘德华直接侵入剧本，使剧本和演员直接耦合在一起，如图 4-1 所示。

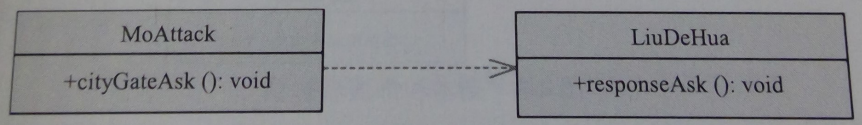


图 4-1　剧本和演员直接耦合

就如图中设计，直接创建了刘德华的类，由该对象说话，可以看出刘德华这个演员和电影完全的耦合在一起，就得引人深思，这个角色本身是革离还是刘德华呢。

一个优秀的导演应该考虑的是角色本身而不是演员本身，换句话说，我要考虑的是革离，然后演员是谁其实不是唯一，那么上面的设计显然是不符合要求的。有人说，为什么不考虑下Java的面向对象编程呢。对，这样考虑完全没问题，具体对象的实现放在具体考虑时决定，针对这种观点可以这样设计：

代码清单 4-2　MoAttack：通过角色编排剧本

```
public class MoAttack {
  public void cityGateAsk(){

      //①引入革离角色接口
      GeLi geli = new LiuDeHua();

      //②通过接口展开剧情
      geli.responseAsk("墨者革离！");
  }
}
```

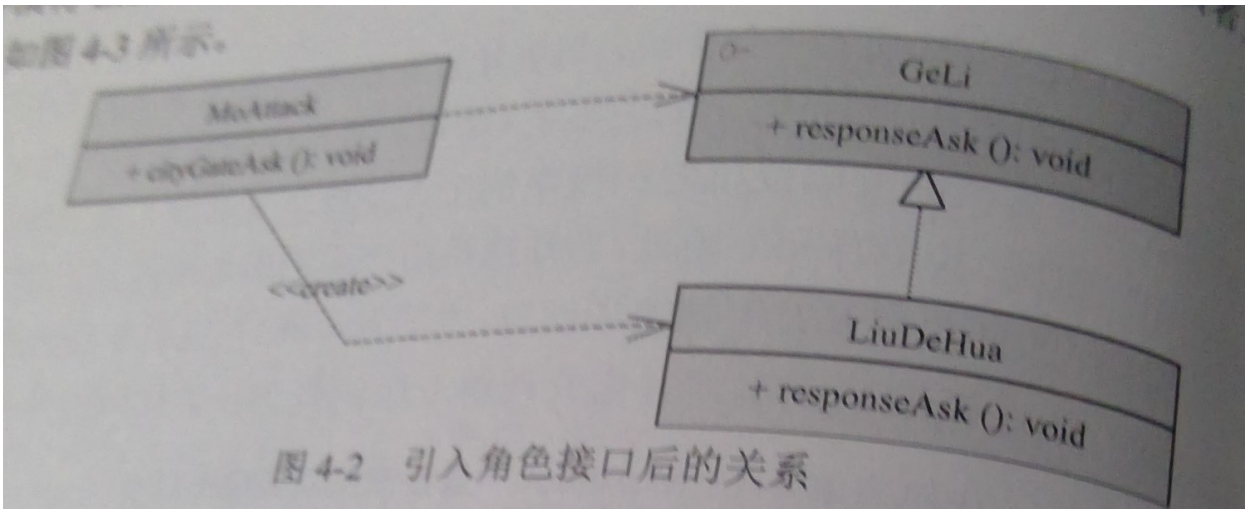


图 4-2　引入角色接口后的关系

但其实你发现了吗，我们电影确实关注角色了，但是刘德华还是绑定在电影里面了，现在的情况时电影不仅仅要关注刘德华了，要同时关注革离和刘德华，这看起来并不是一个优秀的设计。

那么，我们到底应该怎么设计呢。提供一个灵感，革离时电影的主角，刘德华时他的演员，其实无论刘德华，周润发，黄晓明还是其他谁，他们都是演员，谁来演革离都可以，还不是导演选的。对了，导演，导演就是其中的关键，演员可以换，导演却不会换，导演侵入到电影中是无所谓的，因为不会改动。那么，有思路了吗，其实思路很简单，由导演创建具体的角色演员，然后电影通过导演获取演员，这样电影是直接和导演挂钩而不需要和演员挂钩了：

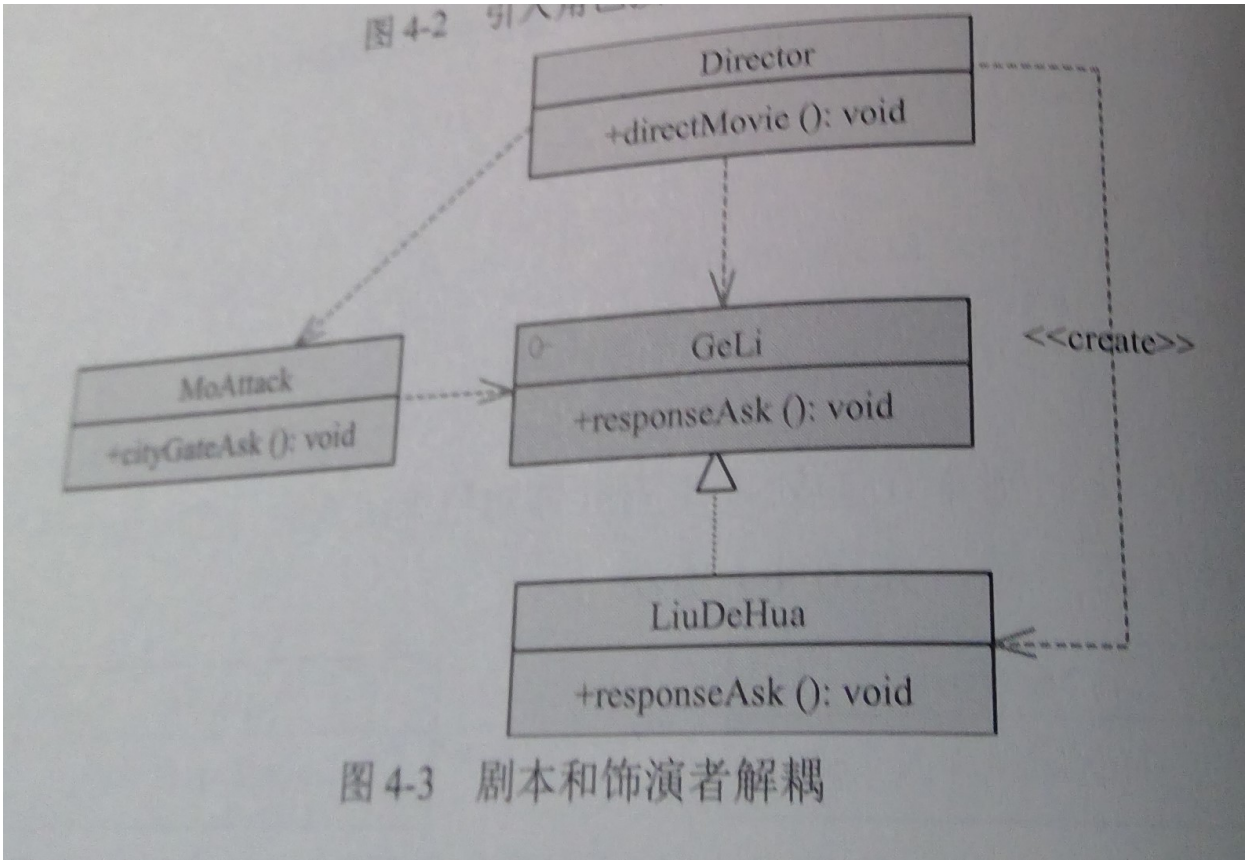


图 4-3　剧本和饰演者解耦

那么，有人要问了，我们不是在讲Ioc吗，和这个有什么关系。哈哈，到现在你还没有发现吗，可以这么说，我们的IoC就是这个案例里面的导演。我们来理解理解控制反转的含义，其分为控制和反转两部分，控制是指对对象的创建以及生命周期维护的控制权，反转是指将这样的控制权从具体类里移除转交第三方管理，Spring就是这样的第三方，通过bean配置来控制对象。

后来又有人提出了一种比IoC更直观的说法，DI（dependency injection），依赖注入，什么意思呢，我们的具体类创建一个类的对象时是不是依赖了这个类对象，这时我们通过第三方管理这个类对象，并让具体类使用，是不是相当于注入进去，故名为依赖注入。仔细一想，是不是就是和IoC描述的一个意思呢，所以可不要傻傻的还以为IoC和DI是Spring中的两个概念。

IoC的两种注入方式

依旧拿导演，角色，电影，演员这四者来举例，上面也说过了，一个优秀的电影应该是电影只关注角色，导演统管全局的，那么在具体的应用中又是如何的体现的呢？这里就要开始正式运用IoC的概念了，控制反转，导演如何做到不侵入电影就把演员和角色对应上呢？

1.构造器注入

顾名思义，这里得用到构造函数，如图：

```
public class MoAttack {
  private GeLi geli;

  //①注入革离的具体饰演者
  public MoAttack(GeLi geli){
      this.geli = geli;
  }
  public void cityGateAsk(){
      geli.responseAsk("墨者革离！");
  }
}
```

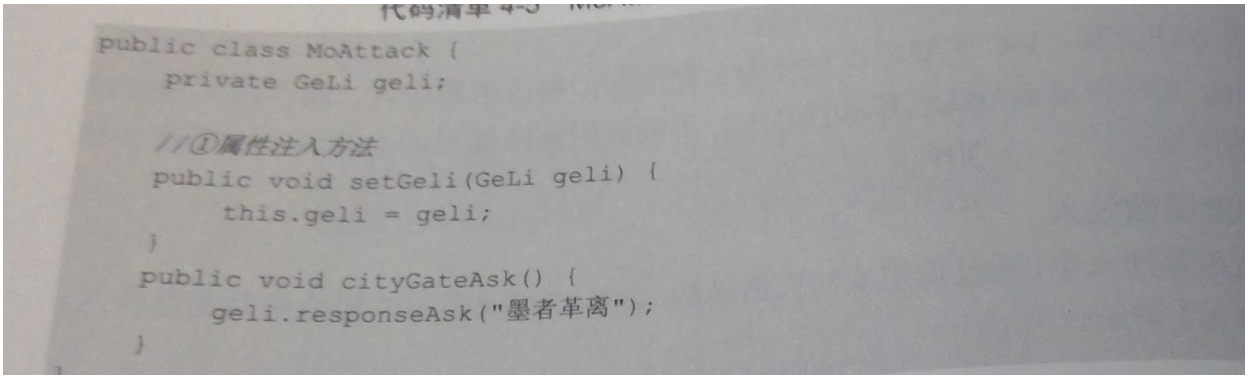
在电影中定义构造函数，形参就是角色，而导演中就这样设计，如图：

```
public class Director {
  public void direct(){

    //①指定角色的饰演者
    GeLi geli = new LiuDeHua();

    //②注入具体饰演者到剧本中
    MoAttack moAttack = new MoAttack(geli);
```

导演负责指定了角色的饰演者者是刘德华，并通过的电影的带参构造方法将实例化为刘德华的角色传入，这样电影就没有和刘德华演员直接挂钩就将刘德华带入了。

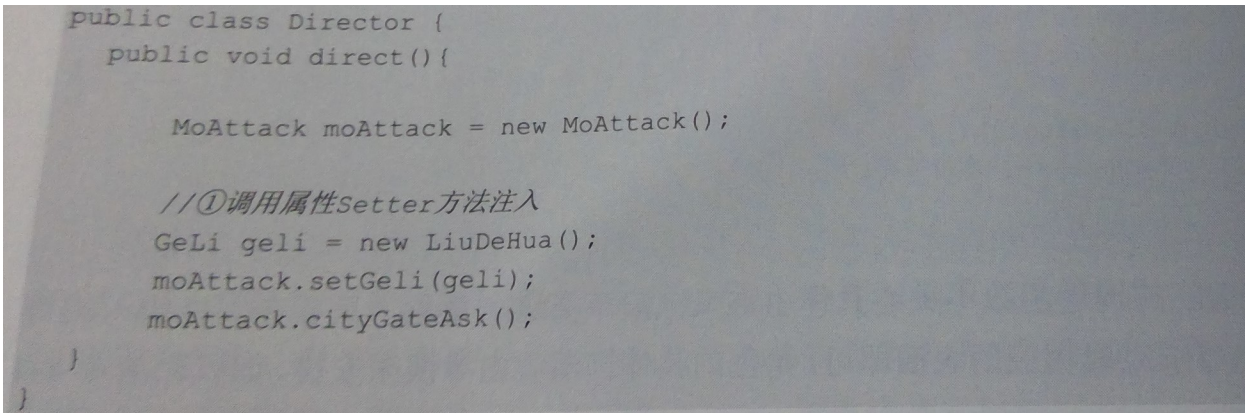
2.属性注入

其实更好理解，也更常用，为什么这么说呢？作为一个优秀的Java Web开发工程师，我们必不可少要写好多的实体类，那些实体类里什么最多，setter方法呀。这里的属性注入就是通过setter方法注入，好处就是定义实体类为啥不同构造函数而选用setter方法的好处，灵活呗。具体实现如图：



```
public class MoAttack {
    private GeLi geli;

    //①属性注入方法
    public void setGeli(GeLi geli) {
        this.geli = geli;
    }
    public void cityGateAsk() {
        geli.responseAsk("墨者革离");
    }
}
```

自然导演里面的设计也很容易联想了，如图：



```
public class Director {
  public void direct(){

      MoAttack moAttack = new MoAttack();

      //①调用属性Setter方法注入
      GeLi geli = new LiuDeHua();
      moAttack.setGeli(geli);
      moAttack.cityGateAsk();
  }
}
```

3.总结

总体来说，IoC就是这么个概念，Spring无非就是取代了导演这个位置，而且通过配置文件实现了更小的对代码的侵入。