# 定义

依赖倒置原则（Dependence Inversion Principle，DIP）是Object Mentor公司总裁罗伯特·马丁（Robert C.Martin）于1996年在C++ Report上发表的文章。

依赖倒置原则的原始定义：高层模块不应该依赖底层模块，两者都应该依赖抽象；抽象不应该依赖细节，细节应该依赖抽象（High level modules should not depend upon low level modules.Both should depend upon abstractions.Details should not depend upon details.Details should depend upon abstractions）。

# 核心思想

依赖倒置原则的核心思想是：要面向接口编程，不要面向实现编程。

# 重要性

依赖倒置原则是实现开闭原则的重要途径之一，它降低了功能使用模块与功能提供模块之间的耦合度。

在软件设计中，细节具有多变性，抽象则相对稳定，因此以抽象为基础搭建起来的架构要比以细节为基础搭建起来的架构要稳定得多。这里的抽象指的是接口或者抽象类，而细节是指具体的实现类。

# 作用

依赖倒置的主要作用如下：

1. 降低类间的耦合度

2. 提高系统的稳定性

3. 降低并行开发的风险

4. 提高代码可读性和可维护性

# 实现方法

依赖倒置原则的目的是通过面向接口编程来降低类间的耦合度，所以我们在实际编程中要遵循以下4点：

1. 每个类尽量提供接口或抽象类，或者两者都具备。
2. 变量的声明类型尽量是接口或者抽象类。
3. 任何类都不应该从具体类派生。
4. 使用继承时尽量遵循里氏替换原则。

# 如何理解

下面以“顾客购物程序”为例来说明依赖倒置原则的应用。  
  
【例1】依赖倒置原则在“顾客购物程序”中的应用。

分析：本程序反映了“顾客类”与“商店类”的关系。商店类中有sell()方法，顾客类通过该方法购物，以下代码定义了顾客类通过韶关网店 ShaoguanShop 购物：

|  |
| --- |
| class Customer *{* public void shopping*(*ShaoguanShop shop*)  {* //购物  System.*out*.println（shop.sell*()*);  *} }* |

但是，这种设计存在缺点，如果该顾客想从另外一家商店（如婺源网店 WuyuanShop）购物，就要将该顾客的代码修改如下：

|  |
| --- |
| class Customer *{* public void shopping*(*WuyuanShop shop*)  {* //购物  System.*out*.println*(*shop.sell*())*;  *} }* |

顾客每更换一家商店，都要修改一次代码，这明显违背了开闭原则。存在以上缺点的原因是：顾客类设计时同具体的商店类绑定了，这违背了依赖倒置原则。解决方法是：定义“婺源网店”和“韶关网店”的共同接口 Shop，顾客类面向该接口编程，其代码修改如下：

|  |
| --- |
| class Customer *{* public void shopping*(*Shop shop*)  {* //购物  System.*out*.println*(*shop.sell*())*;  *} }* |

这样，不管顾客类Customer访问什么商店，或者增加新的商店，都不需要修改原有代码了，其类图如图 1 所示。



图1 顾客购物程序的类图

# 其他问题

## 如何理解依赖倒置原则是实现开闭原则的途径？

回答问题前不妨先回想一下，什么是开闭原则——“软件实体应该对扩展开放，对修改关闭”，意思是，在软件开发的过程中，不通过修改模块、类、接口、方法等实体的方式来满足新的需求，而是通过扩展的方式来满足新的需求。

遵循依赖倒置原则开发的软件系统整体结构比较稳定，高低层模块间耦合度较低，因此能够非常简单的将新的实现类引入系统，并且不影响系统中的其他实现类。

**7.2遵循依赖倒置原则开发的软件系统整体结构比较稳定**



生活中很多需要被大范围使用的商品都会被制定一套生产标准，比如说USB接口，各厂商都按照标准设计和生产产品。为什么要有这套标准呢？不妨想象如果没有这套标准会怎样？