# 软件架构设计的七大原则

# 开闭原则

开闭原则（Open-Closed Principle， OCP）是指一个软件实体如类、模块和函数应该对扩展开放，对修改关闭。所谓的开闭，也正是对扩展和修改两个行为的一个原则。强调的是用抽象构建框架，用实现扩展细节。可以提高软件系统的可复用性及可维护性。开闭原则，是面向对象设计中最基础的设计原则。它指导我们如何建立稳定灵活的系统，例如：我们版本更新，我尽可能不修改源代码，但是可以增加新功能。

# 依赖倒置原则

依赖倒置原则（Dependence Inversion Principle，DIP）是指设计代码结构时，高层模块不应该依赖底层模块，二者都应该依赖其抽象。抽象不应该依赖细节；细节应该依赖抽象。通过依赖倒置，可以减少类与类之间的耦合性，提高系统的稳定性，提高代码的可读性和可维护性，并能够降低修改程序所造成的风险。大家要切记：以抽象为基准比以细节为基准搭建起来的架构要稳定得多，因此大家在拿到需求之后，要面向接口编程，先顶层再细节来设计代码结构。

# 单一职责原则

单一职责（Simple Responsibility Pinciple，SRP）是指不要存在多于一个导致类变更的原因。假设我们有一个Class负责两个职责，一旦发生需求变更，修改其中一个职责的逻辑代码，有可能会导致另一个职责的功能发生故障。这样一来，这个Class存在两个导致类变更的原因。如何解决这个问题呢？我们就要给两个职责分别用两个Class来实现，进行解耦。后期需求变更维护互不影响。这样的设计，可以降低类的复杂度，提高类的可读性，提高系统的可维护性，降低变更引起的风险。总体来说就是一个Class/Interface/Method只负责一项职责。但是，我们在实际开发中会项目依赖，组合，聚合这些关系，还有项目的规模，周期，技术人员的水平，对进度的把控，很多类都不符合单一职责。但是，我们在编写代码的过程，尽可能地让接口和方法保持单一职责，对我们项目后期的维护是有很大帮助的。

# 接口隔离原则

接口隔离原则（InterfaceSegregationPrinciple，ISP）是指用多个专门的接口，而不使用单一的总接口，客户端不应该依赖它不需要的接口。这个原则指导我们在设计接口时应当注意一下几点：

1、一个类对一类的依赖应该建立在最小的接口之上。

2、建立单一接口，不要建立庞大臃肿的接口。

3、尽量细化接口，接口中的方法尽量少（不是越少越好，一定要适度）。

接口隔离原则符合我们常说的高内聚低耦合的设计思想，从而使得类具有很好的可读性、可扩展性和可维护性。我们在设计接口的时候，要多花时间去思考，要考虑业务模型，包括以后有可能发生变更的地方还要做一些预判。所以，对于抽象，对业务模型的理解是非常重要的。

# 迪米特法则

迪米特原则（LawofDemeterLoD）是指一个对象应该对其他对象保持最少的了解，又叫最少知道原则（Least Knowledge Principle，LKP），尽量降低类与类之间的耦合。迪米特原则主要强调只和朋友交流，不和陌生人说话。出现在成员变量、方法的输入、输出参数中的类都可以称之为成员朋友类，而出现在方法体内部的类不属于朋友类。现在来设计一个权限系统，Boss 需要查看目前发布到线上的课程数量。这时候，Boss要找到TeamLeader去进行统计，TeamLeader再把统计结果告诉Boss。学习软件设计原则，千万不能形成强迫症。碰到业务复杂的场景，我们需要随机应变。

# 里氏替换原则

里氏替换原则（Liskov Substitution Principle，LSP）是指如果对每一个类型为 T1 的对象o1，都有类型为T2的对象o2，使得以T1定义的所有程序P在所有的对象o1都替换成o2时，程序P的行为没有发生变化，那么类型T2是类型T1的子类型。

定义看上去还是比较抽象，我们重新理解一下，可以理解为一个软件实体如果适用一个父类的话，那一定是适用于其子类，所有引用父类的地方必须能透明地使用其子类的对象，子类对象能够替换父类对象，而程序逻辑不变。根据这个理解，我们总结一下：

引申含义：子类可以扩展父类的功能，但不能改变父类原有的功能。

1、子类可以实现父类的抽象方法，但不能覆盖父类的非抽象方法。

2、子类中可以增加自己特有的方法。

3、当子类的方法重载父类的方法时，方法的前置条件（即方法的输入/入参）要比父类方法的输入参数更宽松。

4、当子类的方法实现父类的方法时（重写/重载或实现抽象方法），方法的后置条件（即方法的输出/返回值）要比父类更严格或相等。

使用里氏替换原则有以下优点：

1、约束继承泛滥，开闭原则的一种体现。

2、加强程序的健壮性，同时变更时也可以做到非常好的兼容性，提高程序的维护性、扩展性。降低需求变更时引入的风险。

# 合成复用原则

合成复用原则（Composite/Aggregate Reuse Principle，CARP）是指尽量使用对象组合(has-a)/聚合(contanis-a)，而不是继承关系达到软件复用的目的。可以使系统更加灵活，降低类与类之间的耦合度，一个类的变化对其他类造成的影响相对较少。继承我们叫做白箱复用，相当于把所有的实现细节暴露给子类。组合/聚合也称之为黑箱复用，对类以外的对象是无法获取到实现细节的。要根据具体的业务场景来做代码设计，其实也都需要遵循OOP模型。

# 设计原则总结

学习设计原则，学习设计模式的基础。在实际开发过程中，并不是一定要求所有代码都遵循设计原则，我们要考虑人力、时间、成本、质量，不是刻意追求完美，要在适当的场景遵循设计原则，体现的是一种平衡取舍，帮助我们设计出更加优雅的代码结构。