<https://blog.csdn.net/eric_jo/article/details/6818552>

**一．OO(面向对象)的设计基础**

**面向对象(OO)**：就是基于对象概念，以对象为中心，以类和继承为构造机制，充分利用接口和多态提供灵活性，

                             来认识、理解、刻划客观世界和设计、构建相应的软件系统。

**面向对象的特征**：虽然各种面向对象编程语言相互有别，但都能看到它们对面向对象基本特征的支持，

                                  即 “抽象、封装、继承、多态” ：

                                  – 抽象，先不考虑细节

                                  – 封装，隐藏内部实现

                                  – 继承，复用现有代码

                                  – 多态，改写对象行为

**面向对象设计模式**： 是“好的面向对象设计”，所谓“好的面向对象设计”是那些可以满足“应对变化，

提高复用”的设计。面向对象设计模式描述的是软件设计，因此它是独立于编程 语言的，但是面向

对象设计模式的最终实现仍然要使用面向对象编程语言来表达。面向对象设计模式不像算法技巧，

可以照搬照用，它是建立在对“面向对象”纯 熟、深入的理解的基础上的经验性认识。

上边就见大的描述一下面向对象和设计模式的概念和关系。我们进行设计的时候，就是充 分的理解和

利用OO的四个基本的特征来展开设计，所以大家必须在进行设计前，要熟悉和掌握面向对象的技术，

在这就不详细介绍了，而对于设计模式是给我们提供了设计时的参考模型，而掌握面向对象设计模式的

前提是首先掌握“面向对象”技术。

**二．OO(面向对象)的设计目标**

※**可扩展性**Extensibility：有了新的需求，新的性能可以容易添加到系统中，不影响现有的性能，也不会带来新的缺陷。

※**可修改性**Flexibility：系统一部分的代码要修改时不会破坏系统的现有结构，也不会影响到其它的部分。

※**可替换性**Pluggability：可以将系统中的某些代码替换为相同接口的其它类，不会影响到系统。

**三．OO设计的5大原则及其之间的关系**

**3.1 OO设计原则的总结**

关于OO设计的5个原则，在前面的文章里，每个都有一篇详细的说明，

在这里就不详细解释了，以下简单总结一下。

**※单一职责原则:就一个类而言，应该仅有一个引起它变化的原因。**

单一是一个类的优良设计。交杂不清的职责将使得代码看起来特别别扭牵一发而动全身，

有失美感和必然导致丑陋的系统错误风险。

**※开放封闭原则:是说软件实体（类、模块、函数等等）应该可以扩展但不可修改。**

实现开开放封闭原则的核心思想就是对抽象编程，而不对具体编程，因为抽象相对稳定。

让类依赖于固定的抽象，所以修改就是封闭的；而通过面向对象的继承和多态机制，

又可以实现对抽象类的继承，通过覆写其方法来改变固有行为，实现新的拓展方法，所以就是开放的。

“需求总是变化”没有不变的软件，所以就需要用封闭开放原则来封闭变化满足需求，

同时还能保持软件内部的封装体系稳定，不被需求的变化影响。

**※依赖倒置原则:依赖抽象，不要依赖具体。**

抽 象的稳定性决定了系统的稳定性，因为抽象是不变的，依赖于抽象是面向对象设计的精髓，

也是依赖倒置原则的核心。依赖于抽象是一个通用的原则，而某些时候依 赖于细节则是在所难免的，

必须权衡在抽象和具体之间的取舍，方法不是一层不变的。依赖于抽象，就是对接口编程，不要对实现编程。

**※里氏代换原则：子类型必须能够替换到他们的父类型。**

Liskov 替换原则，主要着眼于对抽象和多态建立在继承的基础上，因此只有遵循了Liskov替换原则，

才能保证继承复用是可靠地。实现的方法是面向接口编程：将公 共部分抽象为基类接口或抽象类，

通过ExtractAbstractClass，在子类中通过覆写父类的方法实现新的方式支持同样的职责。Liskov替 换原则能够

保证系统具有良好的拓展性，同时实现基于多态的抽象机制，能够减少代码冗余，避免运行期的类型判别。

**※接口隔离原则: 多个和客户相关的接口要好于一个通用接口。**

分离的手段主要有以下两种：1、委托分离，通过增加一个新的类型来委托客户的请求，隔离客户和接口的直接依赖，

但是会增加系统的开销。2、多重继承分离，通过接口多继承来实现客户的需求，这种方式是较好的。

下边是前面没有提到过的两个原则，也是设计时要考虑的重要原则。

**※迪米特法则：不相互直接通信的类之间，不要直接发生相互作用。**

如果两个类不必彼此直接通信，那么这两个类就不应当发生直接的相互作用。如果一个类需要调用领一个类的

某个方法话，可以通过第三者转发这个调用。迪米特法则首先强调的前提是在类的设计上，每一类都应当尽量

降低成员的访问权限。它的根本思想是强调类之间的松耦合。

**※合成/聚合复用原则：尽量使用合成/聚合，尽量不要使用继承。**

合 成(Composition)和聚合(Aggregation)都是关联的特殊种类，聚合表示一种弱的拥有关系；

合成这是一种强的拥有关系，体现了严格的部分和整体的关系，部分和整体的生命周期一样。

优先使用合成或聚合原则将有助于保持每个类被封装，并被集中在单个任务上。这样类和类继承

层次会保持较小规 模，并且不太可能增长为不可控制的庞然大物

**3.2 OO设计原则之间的关系**

1. 实现“开-闭”原则的关键步骤是抽象化。基类与子类之间的继承关系就是抽象化的体现。

      因此里氏代换原则是对实现抽象化的具体步骤的规范。

      违反里氏代换原则意味着违反了“开-闭”原则，反之未必。

2. “开-闭”原则与依赖倒转原则是目标和手段的关系。如果说开闭原则是目标,依赖倒转原则

     是到达"开闭"原则的手段。如果要达到最好的"开闭"原则,就要尽量的遵守依赖倒转原则，

     依赖倒转原则是对"抽象化"的最好规范。

3. 里氏代换原则是依赖倒转原则的基础，依赖倒转原则是里氏代换原则的重要补充。

4. 接口分离原则也是确保“开-闭”原则的一个重要手段。

5. 对于单一职责原则，个人认为尽量做到为好，职责越单一，“开-闭”和里氏代换越容易实现。

**四．OO设计原则和目标的关系**

1.可扩展性Extensibility :允许一个具有同样接口的新类替代旧类，是对抽象接口的复用。

客户端依赖于抽象接口，而不是一个具体实现类，使得这个具体类可以被别的具体类替换，

而不影响客户端。以下原则实现可扩展性。

※开/闭原则

※里氏替换原则

※依赖倒转原则

※合成/聚合复用原则

2.可修改性Flexibility:模块相对独立，通信尽可能少。这样当一个模块修改时，对别的模块的影响很小。

以下原则实现可修改性。

※开/闭原则

※迪米特法则

※接口隔离原则

3、可替换性Pluggability：当一部分不再满足需要时，可以将旧的部分拔出，新的部分插入。

以下原则实现可替换性。

※开/闭原则

※里氏代换原则

※依赖倒转原则

※合成/聚合复用原则

**五．OO(面向对象)的设计过程**

**1. 分析式样，进行机能分类。**

**2. 根据机能进行类的抽象。**

※ 类的抽象 - 在这里步里，我们可以根据 “单一职责原则”，进行类的具体抽象。

     尽量做到，类的功能单一和清晰化。

※ 封装变化点– 使用封装来创建对象之间的分界层，让设计者可以在分界层的一侧进行修改，

     而不会对另一侧产生不良的影响，从而实现层次间的松耦合。

**3. 设计抽象基类和接口类。**

※ 在进行基本的基类的抽象和接口定义时，要遵照“接口分离原则”进行接口的抽象。

※ 在设计接口和基类时，不要总是关注细节，要记住针对接口编程，而不是针对实现编程。

※ 对于抽象的基类和派生类之间要做到“里氏替换原则”的要求。

**4.确定类间的耦合关系。**

**4.1 决定耦合的程度的依据何在呢？**

※ 简单的说，就是根据需求的稳定性，来决定耦合的程度。

※ 对于稳定性高的需求，不容易发生变化的需求，我们完全可以把各类设计成紧耦合的，

     因为这样可以提高效率，而且我们还可以使用一些更好的技术来提高效率或简化代码。

※ 如 果需求极有可能变化，我们就需要充分的考虑类之间的耦合问题，我们可以想出各种

     各样的办法来降低耦合程度，但是归纳起来，不外乎增加抽象的层次来隔离不同的类，

     这个抽象层次可以是抽象的类、具体的类，也可以是接口，或是一组的类。

     我们可以用一句话来概括降低耦合度的思想："针对接 口编程，而不是针对实现编程。

※ 在决定类的耦合关系时，尽量考虑“迪米特法则”和“合成/聚合复用原则”。

**4.2 怎样做到依赖倒转？**

※ 以抽象方式耦合是依赖倒转原则的关键。抽象耦合关系总要涉及具体类从抽象类继承，

     并且需要保证在任何引用到基类的地方都可以改换成其子类，因此，里氏代换原则是依赖倒转原则的基础。

※ 依赖于抽象：建议不依赖于具体类，即程序中所有的依赖关系都应该终止于抽象类或者接口。尽量做到：

    （1）任何变量都不应该持有一个指向具体类的指针或者引用。

    （2）任何类都不应该从具体类派生。

    （3）任何方法都不应该覆写它的任何基类中的已经实现的方法。

**5．运用OO设计的5大原则来对设计进行进一步的优化**

※ 对于类的抽象和职能，是否满足“单一职责原则”

※ 对于继承关系和引用基类的地方，是否满足“里氏代换原则”和“依赖倒置原则”

※ 对于接口和基类，是否“接口隔离原则”

※ 总体上是否满足“开-闭原则”

总体上说，在面向对象设计时，要充分考虑设计的5大原则，但不是强求的，一味的追求满足原则也可能会

导致设计出的系统在性能和资源上的消耗，可以根据具体的情况来具体的分析和设计。