<http://blog.csdn.net/coolingcoding/article/details/8043265>

[【设计原则】面向对象编程的六大原则](http://blog.csdn.net/coolingcoding/article/details/8043265)

**参考文章：**

[**http://blog.csdn.net/wangjunkg/article/details/3762132**](http://blog.csdn.net/wangjunkg/article/details/3762132)

# **一、单一职责原则：**

**全称：**“Single-Responsibility Principle”

**说明：**就一个类而言，应该只专注于做一件事和仅有一个引起它变化的原因。所谓职责，我们可以理解他为功能，就是设计的这个类功能应该只有一个，而不是两个或更多。也可以理解为引用变化的原因，当你发现有两个变化会要求我们修改这个类，那么你就要考虑撤分这个类了。因为职责是变化的一个轴线，当需求变化时，该变化会反映类的职责的变化。

使用SRP**注意点**：

1、一个合理的类，应该仅有一个引起它变化的原因，即单一职责；

2、在没有变化征兆的情况下应用SRP或其他原则是不明智的；

3、在需求实际发生变化时就应该应用SRP等原则来重构代码；

4、使用测试驱动开发会迫使我们在设计出现臭味之前分离不合理代码；

5、如果测试不能迫使职责分离，僵化性和脆弱性的臭味会变得很强烈，那就应该用Facade或Proxy模式对代码重构；SRP优点：消除耦合，减小因需求变化引起代码僵化。

# **二、里氏代换原则**

**全称**：“Liskov Substitution Principle”

**说明**：子类型必须能够替换它们的基类型。一个软件实体如果使用的是一个基类，那么当把这个基类替换成继承该基类的子类，程序的行为不会发生任何变化。软件实体察觉不出基类对象和子类对象的区别。

**优点**：可以很容易的实现同一父类下各个子类的互换，而客户端可以毫不察觉。

# **三、依赖倒置原则**

**全称**：“Dependence Inversion Principle”

**说明**：要依赖于抽象，不要依赖于具体。客户端依赖于抽象耦合。

抽象不应当依赖于细节；细节应当依赖于抽象；

要针对接口编程，不针对实现编程。

**优点**：使用传统过程化程序设计所创建的依赖关系，策略依赖于细节，这是糟糕的，因为策略受到细节改变的影响。依赖倒置原则使细节和策略都依赖于抽象，抽象的稳定性决定了系统的稳定性。

怎样做到依赖倒置？

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。

在抽象层次上的耦合虽然有灵活性，但也带来了额外的复杂性，如果一个具体类发生变化的可能性非常小，那么抽象耦合能发挥的好处便十分有限，这时可以用具体耦合反而会更好。

层次化：所有结构良好的面向对象构架都具有清晰的层次定义，每个层次通过一个定义良好的、受控的接口向外提供一组内聚的服务。

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。

# **四、接口隔离原则**

**全称**：“Interface Segregation Principle”

**说明**：使用多个专一功能的接口比使用一个的总接口总要好。从一个客户类的角度来讲：一个类对另外一个类的依赖性应当是建立在最小接口上的。过于臃肿的接口是对接口的污染，不应该强迫客户依赖于它们不用的方法。

优点：会使一个软件系统功能扩展时，修改的压力不会传到别的对象那里。

如何实现接口隔离原则

不应该强迫用户依赖于他们不用的方法。

1、利用委托分离接口。

2、利用多继承分离接口。

# **五、迪米特原则**

**全称：**“Law of Demeter”

**说明**：对象与对象之间应该使用尽可能少的方法来关联，避免千丝万缕的关系。

如何实现迪米特法则？

迪米特法则的主要用意是控制信息的过载，在将其运用到系统设计中应注意以下几点：

1) 在类的划分上，应当创建有弱耦合的类。类之间的耦合越弱，就越有利于复用。

2) 在类的结构设计上，每一个类都应当尽量降低成员的访问权限。一个类不应当public自己的属性，而应当提供取值和赋值的方法让外界间接访问自己的属性。

3) 在类的设计上，只要有可能，一个类应当设计成不变类。

4) 在对其它对象的引用上，一个类对其它对象的引用应该降到最低。

# **六、开放－封闭原则**

**全称**：“Open-Closed Principle”

**说明**：对扩展开放，对修改关闭。

**优点**：按照OCP原则设计出来的系统，降低了程序各部分之间的耦合性，其适应性、灵活性、稳定性都比较好。当已有软件系统需要增加新的功能时，不需要对作为系统基础的抽象层进行修改，只需要在原有基础上附加新的模块就能实现所需要添加的功能。增加的新模块对原有的模块完全没有影响或影响很小，这样就无须为原有模块进行重新测试。

如何实现“开-闭”原则？

在面向对象设计中，不允许更改的是系统的抽象层，而允许扩展的是系统的实现层。换言之，定义一个一劳永逸的抽象设计层，允许尽可能多的行为在实现层被实现。

解决问题关键在于抽象化，抽象化是面向对象设计的第一个核心本质。

对一个事物抽象化，实质上是在概括归纳总结它的本质。抽象让我们抓住最最重要的东西，从更高一层去思考。这降低了思考的复杂度，我们不用同时考虑那么多的东西。换言之，我们封装了事物的本质，看不到任何细节。

在面向对象编程中，通过抽象类及接口，规定了具体类的特征作为抽象层，相对稳定，不需更改，从而满足“对修改关闭”；而从抽象类导出的具体类可以改变系统的行为，从而满足“对扩展开放”。

对实体进行扩展时，不必改动软件的源代码或者二进制代码。关键在于抽象。