1 显示依赖原则

***方法和类应该明确要求（通常通过方法参数或构造函数参数）它们需要的任何协作对象才能正常运行。***

当类中存在对于其它类的依赖 应作为构造函数参数或者方法参数暴露给外部 使外部明确该类的依赖类(与控制反转相关 把接口作为参数传递实现控制反转)

2 最小意外原则(适用与UI等交互模块)

https://en.wikipedia.org/wiki/Principle\_of\_least\_astonishment#Formulation

当界面的两个元素发生冲突或模棱两可时，行为应该是最不会让[用户](https://en.wikipedia.org/wiki/User_(computing))感到惊讶的行为；特别是，[程序员](https://en.wikipedia.org/wiki/Programmer)应该尝试考虑最不会让使用该程序的人感到惊讶的行为，而不是了解程序内部工作原理的自然行为

3 DIP 依赖倒置

当类之间发生依赖时 使用interfacer作为媒介 使类之间的依赖变成了类与接口的依赖

4 IOC（inversion of control） 控制反转

把接口的实现暴露给第三方(容器) 由容器决定依赖关系 这样服务将达到高度抽象 所有的服务中除特定业务外的所有实现都来自于外部的实现

4 DI 依赖注入

当通过IOC进行依赖的向外部暴露后 在服务初始化时需要传入对应的实例 通常我们是在构造函数中暴露这部分 所以通常是在new这个类的时候把依赖类作为参数传入

由此产生了一些容器类 这些类通常负责帮你管理实例的生命周期配置文件等等

例：Freamwork的Unity类

.core的IServiceCollection 和autoFac

AutoFac支持AOP