### 1、设计模式7大原则

1. 开放-封闭原则
2. 单一职责原则
3. 依赖倒转原则
4. 最小知识原则
5. 接口隔离原则
6. 合成/聚合复用原则
7. 里氏代换原则，任何基类可以出现的地方，子类一定可以出现

### 2、依赖倒置

假设我们设计一辆汽车：先设计轮子，然后根据轮子大小设计底盘，接着根据底盘设计车身，最后根据车身设计好整个汽车。这里就出现了一个“依赖”关系：汽车依赖车身，车身依赖底盘，底盘依赖轮子。



维护性低。

现在换一种思路。我们先设计汽车的大概样子，然后根据汽车的样子来设计车身，根据车身来设计底盘，最后根据底盘来设计轮子。这时候，依赖关系就倒置过来了：轮子依赖底盘， 底盘依赖车身， 车身依赖汽车。



这就是依赖倒置原则——把原本的高层建筑依赖底层建筑“倒置”过来，变成底层建筑依赖高层建筑。高层建筑决定需要什么，底层去实现这样的需求，但是高层并不用管底层是怎么实现的。这样就不会出现前面的“牵一发动全身”的情况。

### 3、控制反转（Inversion of Control）

依赖倒置原则的一种代码设计的思路。具体采用的方法就是所谓的依赖注入（Dependency Injection）。说穿了，这几种概念的关系大概如下：





采用的构造函数传入的方式进行的依赖注入。其实还有另外两种方法：Setter传递和接口传递。这里就不多讲了，核心思路都是一样的，都是为了实现控制反转。



### 4、控制反转容器（IoC Container）

上述例子中，对车类进行初始化的那段代码发生的地方，就是控制反转容器。



因为采用了依赖注入，在初始化的过程中就不可避免的会写大量的new。这里IoC容器就解决了这个问题。这个①容器可以自动对你的代码进行初始化，你只需要维护一个Configuration（可以是xml可以是一段代码），而不用每次初始化一辆车都要亲手去写那一大段初始化的代码。

这是引入IoC Container的第一个好处。IoC Container的第二个好处是：我们在②创建实例的时候不需要了解其中的细节。在上面的例子中，我们自己手动创建一个车instance时候，是从底层往上层new的：



这个过程中，我们需要了解整个Car/Framework/Bottom/Tire类构造函数是怎么定义的，才能一步一步new/注入。而IoC Container在进行这个工作的时候是反过来的，它先从最上层开始往下找依赖关系，到达最底层之后再往上一步一步new（有点像深度优先遍历）：



IoC Container可以直接隐藏具体的创建实例的细节。