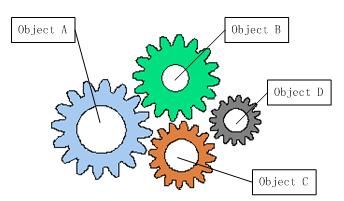
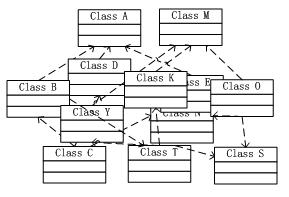
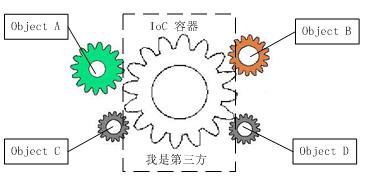
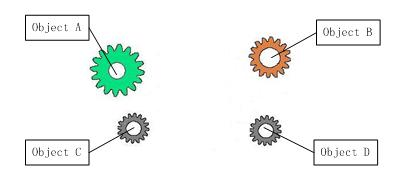
1. IoC理论的背景   
我们都知道，在采用面向对象方法设计的软件系统中，它的底层实现都是由N个对象组成的，所有的对象通过彼此的合作，最终实现系统的业务逻辑。   
  
   
  
  
图1：软件系统中耦合的对象   
如果我们打开机械式手表的后盖，就会看到与上面类似的情形，各个齿轮分别带动时针、分针和秒针顺时针旋转，从而在表盘上产生正确的时间。图1中描述的就是这样的一个齿轮组，它拥有多个独立的齿轮，这些齿轮相互啮合在一起，协同工作，共同完成某项任务。我们可以看到，在这样的齿轮组中，如果有一个齿轮出了问题，就可能会影响到整个齿轮组的正常运转。   
齿轮组中齿轮之间的啮合关系,与软件系统中对象之间的耦合关系非常相似。对象之间的耦合关系是无法避免的，也是必要的，这是协同工作的基础。现在，伴随着工业级应用的规模越来越庞大，对象之间的依赖关系也越来越复杂，经常会出现对象之间的多重依赖性关系，因此，架构师和设计师对于系统的分析和设计，将面临更大的挑战。对象之间耦合度过高的系统，必然会出现牵一发而动全身的情形。   
  
   
  
  
图2：对象之间复杂的依赖关系   
耦合关系不仅会出现在对象与对象之间，也会出现在软件系统的各模块之间，以及软件系统和硬件系统之间。如何降低系统之间、模块之间和对象之间的耦合度，是软件工程永远追求的目标之一。为了解决对象之间的耦合度过高的问题，软件专家Michael Mattson提出了IOC理论，用来实现对象之间的“解耦”，目前这个理论已经被成功地应用到实践当中，很多的J2EE项目均采用了IOC框架产品Spring。   
2. 什么是控制反转(IoC)   
IOC是Inversion of Control的缩写，多数书籍翻译成“控制反转”，还有些书籍翻译成为“控制反向”或者“控制倒置”。   
1996年，Michael Mattson在一篇有关探讨面向对象框架的文章中，首先提出了IOC 这个概念。对于面向对象设计及编程的基本思想，前面我们已经讲了很多了，不再赘述，简单来说就是把复杂系统分解成相互合作的对象，这些对象类通过封装以后，内部实现对外部是透明的，从而降低了解决问题的复杂度，而且可以灵活地被重用和扩展。IOC理论提出的观点大体是这样的：借助于“第三方”实现具有依赖关系的对象之间的解耦，如下图：   
  
  
   
  
  
图3：IOC解耦过程   
大家看到了吧，由于引进了中间位置的“第三方”，也就是IOC容器，使得A、B、C、D这4个对象没有了耦合关系，齿轮之间的传动全部依靠“第三方”了，全部对象的控制权全部上缴给“第三方”IOC容器，所以，IOC容器成了整个系统的关键核心，它起到了一种类似“粘合剂”的作用，把系统中的所有对象粘合在一起发挥作用，如果没有这个“粘合剂”，对象与对象之间会彼此失去联系，这就是有人把IOC容器比喻成“粘合剂”的由来。   
我们再来做个试验：把上图中间的IOC容器拿掉，然后再来看看这套系统：   
  
  
   
  
  
图4：拿掉IoC容器后的系统   
我们现在看到的画面，就是我们要实现整个系统所需要完成的全部内容。这时候，A、B、C、D这4个对象之间已经没有了耦合关系，彼此毫无联系，这样的话，当你在实现A的时候，根本无须再去考虑B、C和D了，对象之间的依赖关系已经降低到了最低程度。所以，如果真能实现IOC容器，对于系统开发而言，这将是一件多么美好的事情，参与开发的每一成员只要实现自己的类就可以了，跟别人没有任何关系！   
我们再来看看，控制反转(IOC)到底为什么要起这么个名字？我们来对比一下：   
软件系统在没有引入IOC容器之前，如图1所示，对象A依赖于对象B，那么对象A在初始化或者运行到某一点的时候，自己必须主动去创建对象B或者使用已经创建的对象B。无论是创建还是使用对象B，控制权都在自己手上。   
软件系统在引入IOC容器之后，这种情形就完全改变了，如图3所示，由于IOC容器的加入，对象A与对象B之间失去了直接联系，所以，当对象A运行到需要对象B的时候，IOC容器会主动创建一个对象B注入到对象A需要的地方。   
通过前后的对比，我们不难看出来：对象A获得依赖对象B的过程,由主动行为变为了被动行为，控制权颠倒过来了，这就是“控制反转”这个名称的由来。   
3. IOC的别名：依赖注入(DI)   
2004年，Martin Fowler探讨了同一个问题，既然IOC是控制反转，那么到底是“哪些方面的控制被反转了呢？”，经过详细地分析和论证后，他得出了答案：“获得依赖对象的过程被反转了”。控制被反转之后，获得依赖对象的过程由自身管理变为了由IOC容器主动注入。于是，他给“控制反转”取了一个更合适的名字叫做“依赖注入（Dependency Injection）”。他的这个答案，实际上给出了实现IOC的方法：注入。所谓依赖注入，就是由IOC容器在运行期间，动态地将某种依赖关系注入到对象之中。   
所以，依赖注入(DI)和控制反转(IOC)是从不同的角度的描述的同一件事情，就是指通过引入IOC容器，利用依赖关系注入的方式，实现对象之间的解耦。   
4. IOC为我们带来了什么好处   
我们还是从USB的例子说起，使用USB外部设备比使用内置硬盘，到底带来什么好处？   
第一、USB设备作为电脑主机的外部设备，在插入主机之前，与电脑主机没有任何的关系，只有被我们连接在一起之后，两者才发生联系，具有相关性。所以，无论两者中的任何一方出现什么的问题，都不会影响另一方的运行。这种特性体现在软件工程中，就是可维护性比较好，非常便于进行单元测试，便于调试程序和诊断故障。代码中的每一个Class都可以单独测试，彼此之间互不影响，只要保证自身的功能无误即可，这就是组件之间低耦合或者无耦合带来的好处。   
第二、USB设备和电脑主机的之间无关性，还带来了另外一个好处，生产USB设备的厂商和生产电脑主机的厂商完全可以是互不相干的人，各干各事，他们之间唯一需要遵守的就是USB接口标准。这种特性体现在软件开发过程中，好处可是太大了。每个开发团队的成员都只需要关心实现自身的业务逻辑，完全不用去关心其它的人工作进展，因为你的任务跟别人没有任何关系，你的任务可以单独测试，你的任务也不用依赖于别人的组件，再也不用扯不清责任了。所以，在一个大中型项目中，团队成员分工明确、责任明晰，很容易将一个大的任务划分为细小的任务，开发效率和产品质量必将得到大幅度的提高。   
第三、同一个USB外部设备可以插接到任何支持USB的设备，可以插接到电脑主机，也可以插接到DV机，USB外部设备可以被反复利用。在软件工程中，这种特性就是可复用性好，我们可以把具有普遍性的常用组件独立出来，反复利用到项目中的其它部分，或者是其它项目，当然这也是面向对象的基本特征。显然，IOC不仅更好地贯彻了这个原则，提高了模块的可复用性。符合接口标准的实现，都可以插接到支持此标准的模块中。   
第四、同USB外部设备一样，模块具有热插拔特性。IOC生成对象的方式转为外置方式，也就是把对象生成放在配置文件里进行定义，这样，当我们更换一个实现子类将会变得很简单，只要修改配置文件就可以了，完全具有热插拨的特性。