IoC是什么？Inversion of Control，即反转控制，或许说为依赖注入更为合适。IoC就是IoC，不是什么技术，与GoF一样，是一种设计模式。   
Interface Driven Design接口驱动，接口驱动有很多好处，可以提供不同灵活的子类实现，增加代码稳定和健壮性等等，但是接口一定是需要实现的，也就是如下语句迟早要执 行：AInterface a = new AInterfaceImp(); 这样一来，耦合关系就产生了，如：

class A{

AInterface a;

A(){

a = new AInterfaceImp();

}

}

Class A与AInterfaceImp就是依赖关系，如果想使用AInterface的另外一个实现就需要更改代码了。当然我们可以建立一个Factory来根据条件生成想要的AInterface的具体实现，即：

class InterfaceImplFactory{

AInterface create(Object condition){

if(condition = condA){

return new AInterfaceImpA();

}elseif(condition = condB){

return new AInterfaceImpB();

}else{

return new AInterfaceImp();

}

}

}

表面上是在一定程度上缓解了以上问题，但实质上这种代码耦合并没有改变。通过IoC模式可以彻底解决这种耦合，它把耦合从代码中移出去，放到统一的XML文 件中，通过一个容器在需要的时候把这个依赖关系形成，即把需要的接口实现注入到需要它的类中，这可能就是“依赖注入”说法的来源了。

IOC模式，系统中通过引入实现了IOC模式的IOC容器，即可由IOC容器来管理对象的生命周期、依赖关系等，从而使得应用程序的配置和依赖性规范与实 际的应用程序代码分开。其中一个特点就是通过文本的配件文件进行应用程序组件间相互关系的配置，而不用重新修改并编译具体的代码。

当前比较知名的IOC容器有：Pico Container、Avalon 、Spring、JBoss、HiveMind、EJB等。   
在上面的几个IOC容器中，轻量级的有Pico Container、Avalon、Spring、HiveMind等，超重量级的有EJB，而半轻半重的有容器有JBoss，Jdon等。   
可以把IoC模式看做是工厂模式的升华，可以把IoC看作是一个大工厂，只不过这个大工厂里要生成的对象都是在XML文件中给出定义的，然后利用Java 的“反射”编程，根据XML中给出的类名生成相应的对象。从实现来看，IoC是把以前在工厂方法里写死的对象生成代码，改变为由XML文件来定义，也就是 把工厂和对象生成这两者独立分隔开来，目的就是提高灵活性和可维护性。   
IoC中最基本的Java技术就是“反射”编程。反射又是一个生涩的名词，通俗的说反射就是根据给出的类名（字符串）来生成对象。这种编程方式可以让对象 在生成时才决定要生成哪一种对象。反射的应用是很广泛的，象Hibernate、String中都是用“反射”做为最基本的技术手段。   
在过去，反射编程方式相对于正常的对象生成方式要慢10几倍，这也许也是当时为什么反射技术没有普通应用开来的原因。但经SUN改良优化后，反射方式生成对象和通常对象生成方式，速度已经相差不大了（但依然有一倍以上的差距）。

## IoC最大的好处是什么？

因为把对象生成放在了XML里定义，所以当我们需要换一个实现子类将会变成很简单（一般这样的对象都是现实于某种接口的），只要修改XML就可以了，这样我们甚至可以实现对象的热插拨（有点象USB接口和SCIS硬盘了）。

IoC最大的缺点是什么？   
（1）生成一个对象的步骤变复杂了（其实上操作上还是挺简单的），对于不习惯这种方式的人，会觉得有些别扭和不直观。   
（2）对象 生成因为是使用反射编程，在效率上有些损耗。但相对于IoC提高的维护性和灵活性来说，这点损耗是微不足道的，除非某对象的生成对效率要求特别高。   
（3） 缺少IDE重构操作的支持，如果在Eclipse要对类改名，那么你还需要去XML文件里手工去改了，这似乎是所有XML方式的缺憾所在。