


```
8         }
9         else if(condition = condB)
10        {
11            return new AInterfaceImpB();
12        }
13        else
14        {
15            return new AInterfaceImp();
16        }
17    }
18 }
```

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置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，像Hibernate、Spring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遍应用开来的原因。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）。

优缺点

编辑

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实现于某种接口的），只要修改XML就可以了，这样我们甚至可以实现对象的热插拔（有点像USB接口和SCSI硬盘了）。



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，那么你还需去XML文件里手工去改了，这似乎是所有XML方式的缺憾所在。

实现初探

编辑

IOC关注服务(或应用程序部件)是如何定义的以及他们应该如何定位他们依赖的其它服务。通常，通过一个容器或定位框架来获得定义和定位的分离，容器或定位框架负责：

保存可用服务的集合

提供一种方式将各种部件与它们依赖的服务绑定在一起

为应用程序代码提供一种方式来请求已配置的对象(例如，一个所有依赖都满足的对象)，这种方式可以确保该对象需要的所有相关的服务都可用。

类型

编辑

现有的框架实际上使用以下三种基本技术的框架执行服务和部件间的绑定：

类型1 (基于接口): 可服务的对象需要实现一个专门的接口，该接口提供了一个对象，可以重用这个对象查找依赖(其它服务)。早期的容器Excalibur使用这种模式。

类型2 (基于setter): 通过JavaBean的属性(setter方法)为可服务对象指定服务。HiveMind和Spring采用这种方式。

类型3 (基于构造函数): 通过构造函数的参数为可服务对象指定服务。PicoContainer只使用这种方式。HiveMind和Spring也使用这种方式。

实现策略

编辑

IoC是一个很大的概念,可以用不同的方式实现。其主要形式有两种：

分享

★

👁

💬

👤

