# Ioc和di

## Ioc（Inversion of Control，控制倒转）

在java中，一个java对象在完成它的业务功能时很多情况下可能需要多个对象来协作完成，这个对象在使用他的协作对象时，可能会使用new 关键字来完成协作对象的申请工作。这样有个坏处就是对象间的耦合度变高了。而IOC的思想是：Spring容器来完成这些相互依赖对象的创建、协调工作，对象只需要关系本身业务逻辑就可以。---对象在如何获得协作对象的责任上被反转了。----但我认为这不应该叫反转，叫转移更合适，反转我认为是彼此之间责任的互换，而ioc在获取申请协作对象的责任交给了第三方即ioc容器，而不是协作对象。

我记得lucene（java**全文检索引擎工具包**）在存储doc（文档）时使用到了一个倒排索引的技术，倒排索引我认为是一种类型的反转；和倒排索引相对的是正向索引，

正向索引的结构是：

“文档1”的ID > 单词1：出现次数，出现位置列表；单词2：出现次数，出现位置列表；…………。

“文档2”的ID > 此文档出现的关键词列表。

当用户在主页上搜索某关键词时，假设只存在正向索引（forward index），那么就需要扫描索引库中的所有文档。

即文档对应词。

倒排索引结构是：

“关键词1”：“文档1”的ID，“文档2”的ID，…………。

“关键词2”：带有此关键词的文档ID列表。

即词对应文档。

## Di（**Dependency Injection，依赖注入**）

**组件之间依赖关系**由容器在运行期决定，形象的说，即**由容器动态的将某个依赖关系注入到组件之中**。**依赖注入的目的并非为软件系统带来更多功能，而是为了提升组件重用的频率，并为系统搭建一个灵活、可扩展的平台。**通过依赖注入机制，我们只需要通过简单的配置，而无需任何代码就可指定目标需要的资源，完成自身的业务逻辑，而不需要关心具体的资源来自何处，由谁实现。

　　理解DI的关键是：“谁依赖谁，为什么需要依赖，谁注入谁，注入了什么”，那我们来深入分析一下：

　　●**谁依赖于谁：**当然是“**应用程序(n多java对象)” 依赖于“IoC容器”**；

　　●**为什么需要依赖：“应用程序”需要“IoC容器”来提供“各对象需要的外部资源”**；

　　●**谁注入谁：**很明显是“**IoC容器”注入“应用程序的某个对象”**；

**●注入了什么：**就是**注入“某个对象所需要的外部资源”（包括对象、资源、常量数据）**。

**IoC和DI**是什么**关系**呢？其实它们**是同一个概念的不同角度描述**，由于控制反转概念比较含糊（可能只是理解为容器控制对象这一个层面，很难让人想到谁来维护对象间关系）。

　 　看过很多对Spring的Ioc理解的文章，好多人对Ioc和DI的解释都晦涩难懂，反正就是一种说不清，道不明的感觉，读完之后依然是一头雾水，感觉 就是开涛这位技术牛人写得特别通俗易懂，他清楚地解释了IoC(控制反转) 和DI(依赖注入)中的每一个字，读完之后给人一种豁然开朗的感觉。我相信对于初学Spring框架的人对Ioc的理解应该是有很大帮助的。

Aop

OOP引入封装、继承和多态性等概念来建立一种对象层次结构，用以模拟公共行为的一个集合。

当我们需要为分散的对象引入公共行为的时候，OOP则显得无能为力。也就是说，OOP允许你定义从上到下的关系，但并不适合定义从左到右的关系。例如日志 功能。日志代码往往水平地散布在所有对象层次中，而与它所散布到的对象的核心功能毫无关系。对于其他类型的代码，如安全性、异常处理和透明的持续性也是如 此。这种散布在各处的无关的代码被称为横切（cross-cutting）代码，在OOP设计中，它导致了大量代码的重复，而不利于各个模块的重用。