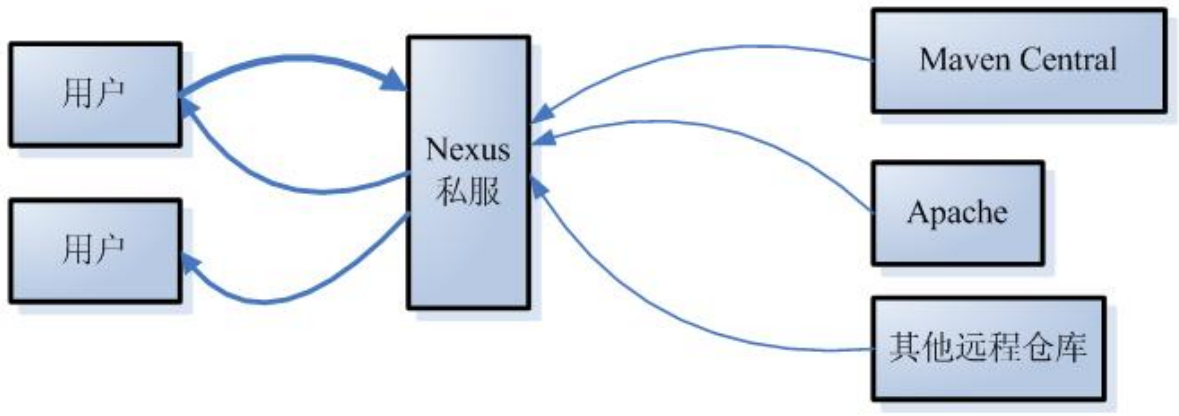
Maven学习总结(九)——使用Nexus搭建Maven私服 - 孤傲苍狼

一、搭建nexus私服的目的

为什么要搭建nexus私服，原因很简单，有些公司都不提供外网给项目组人员，因此就不能使用maven访问远程的仓库地址，所以很有必要在局域网里找一台有外网权限的机器，搭建nexus私服，然后开发人员连到这台私服上，这样的话就可以通过这台搭建了nexus私服的电脑访问maven的远程仓库。

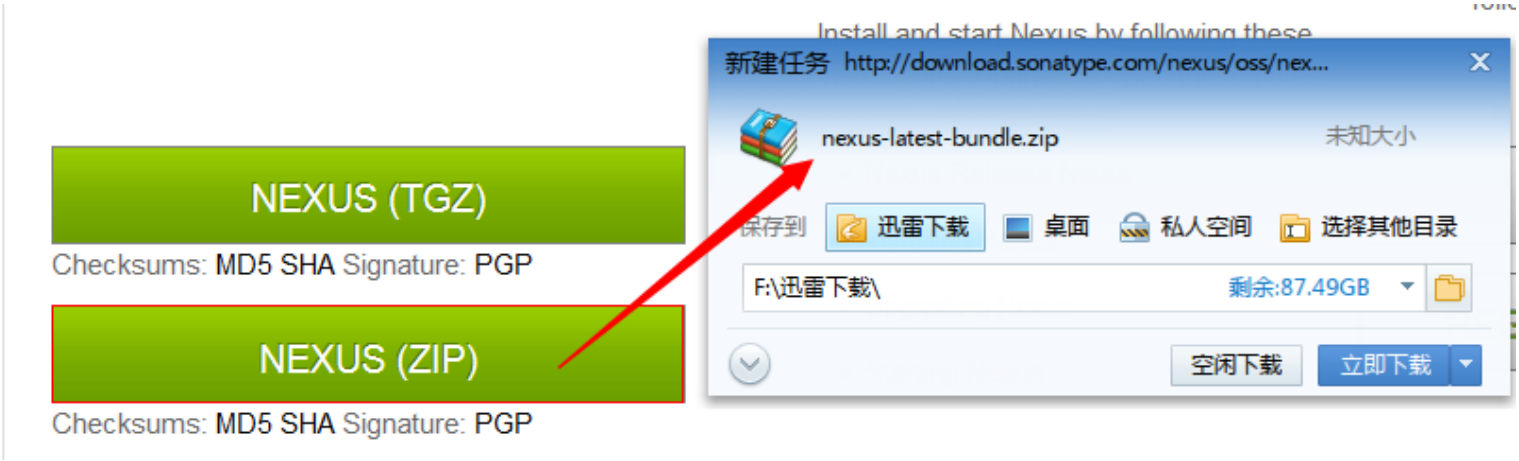
1.1、Nexus架构



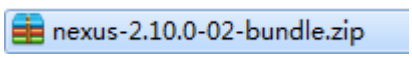
二、nexus私服的下载和安装

2.1、nexus下载

去下载最新版本的nexus，下载地址：http://www.sonatype.org/nexus/go ，如下图所示：



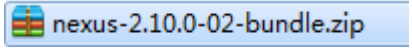
下载完成之后得到



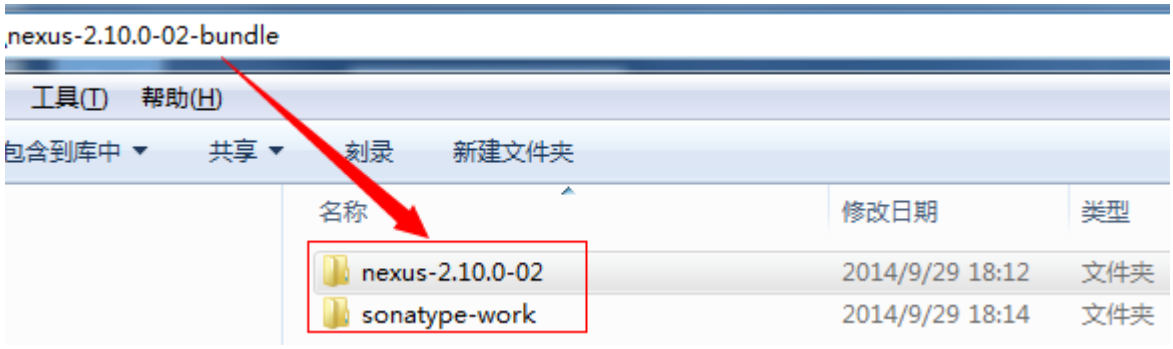
的压缩包。

2.2、nexus安装

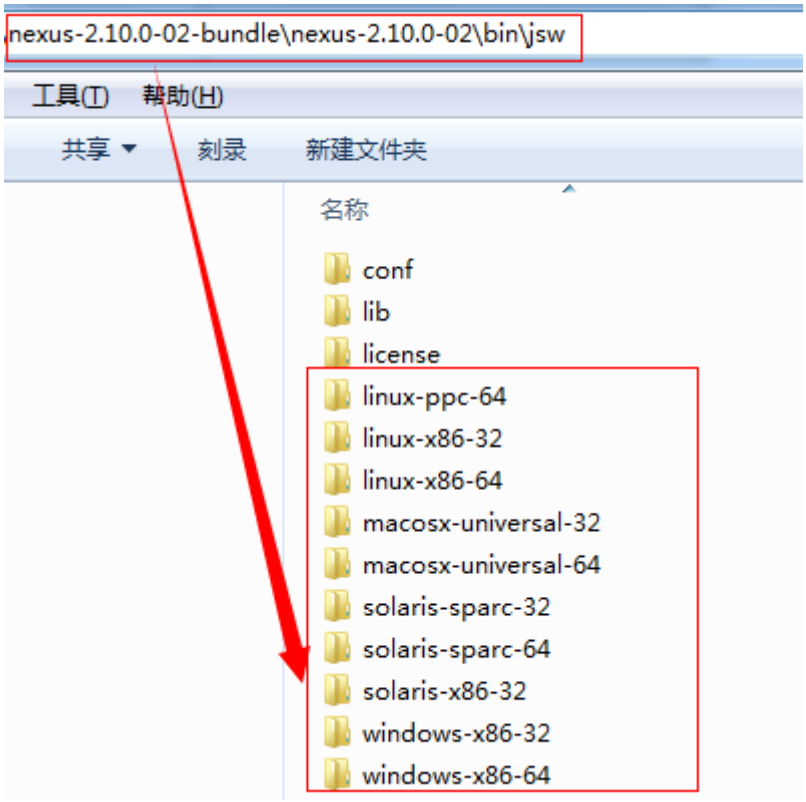
解压



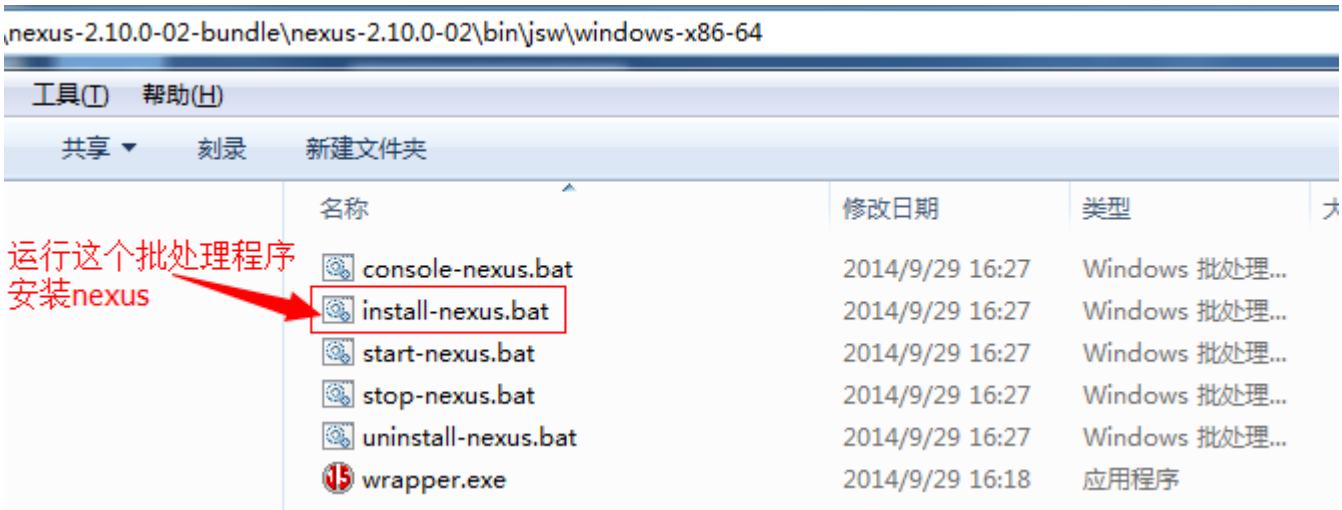
压缩包，如下图所示：



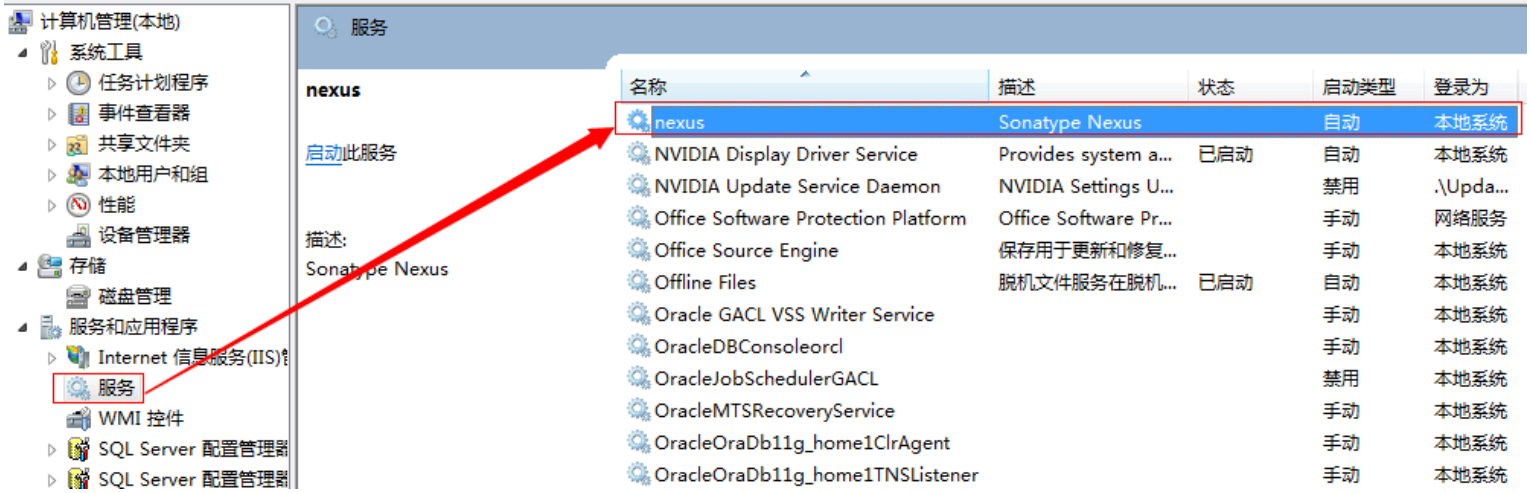
打开目录nexus-2.10.0-02-bundle\nexus-2.10.0-02\bin\jsw ，这个目录下面你会发现有很多系统版本的nexus环境，如下图所示：



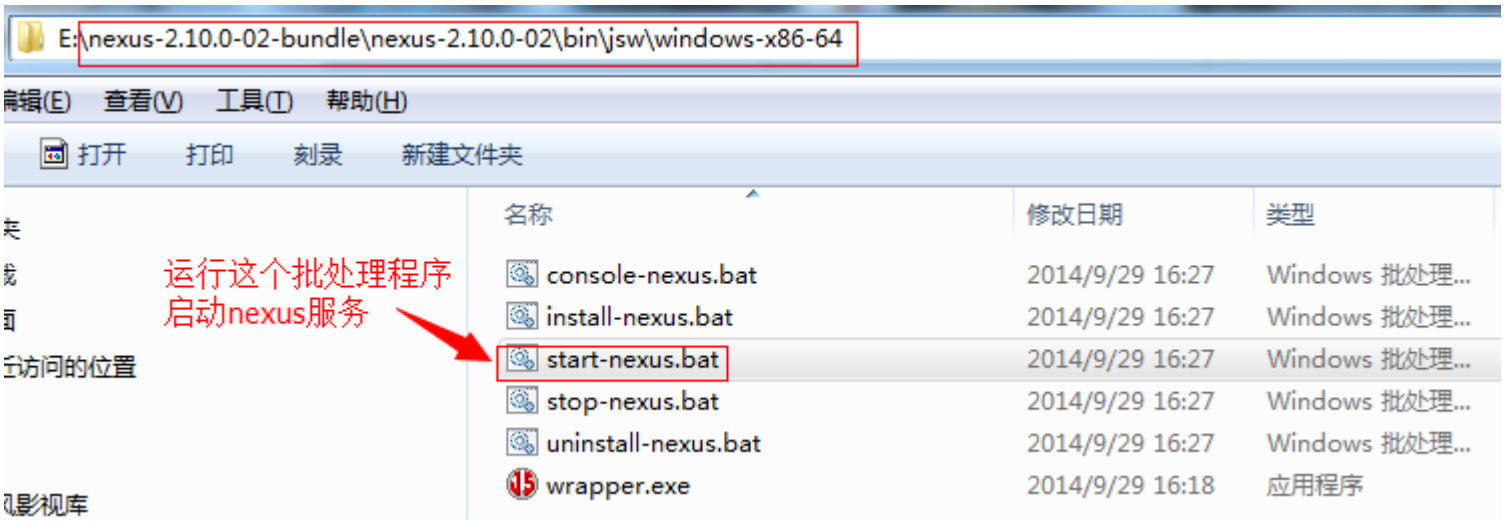
我的电脑是win7 (64)为的系统，所以我选择的是windows-x86-64这个版本，当然可以根据个人的电脑系统选择对应的版本，打开windows-x86-64文件夹，可以看到里面有如下图所示的文件：



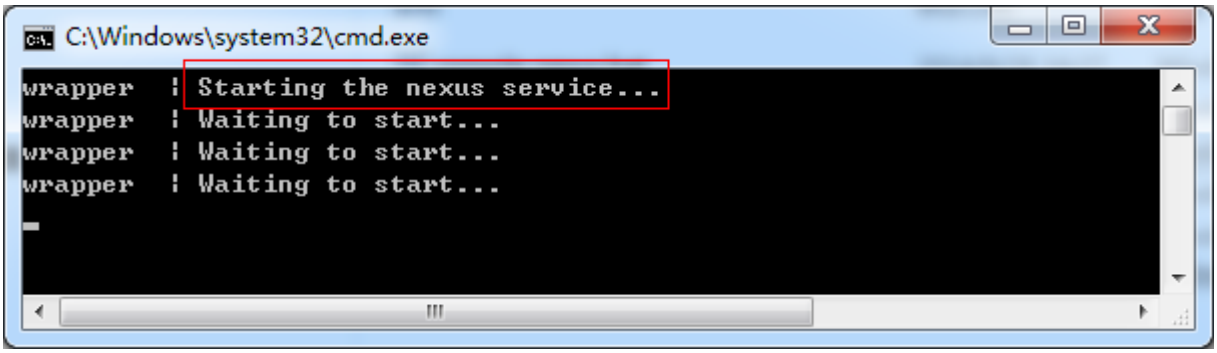
一般都是将nexus安装成windows服务，所以点击install-nexus.bat这个批处理程序将nexus安装成window服务，开机时自动启动。如下图所示：



从服务列表中可以看到，目前nexus服务的还没有启动，所以我们要启动nexus服务，运行nexus-2.10.0-02-bundle\nexus-2.10.0-02\bin\jsw\windows-x86-64下的【start-nexus.bat】批处理程序启动服务，如下图所示：



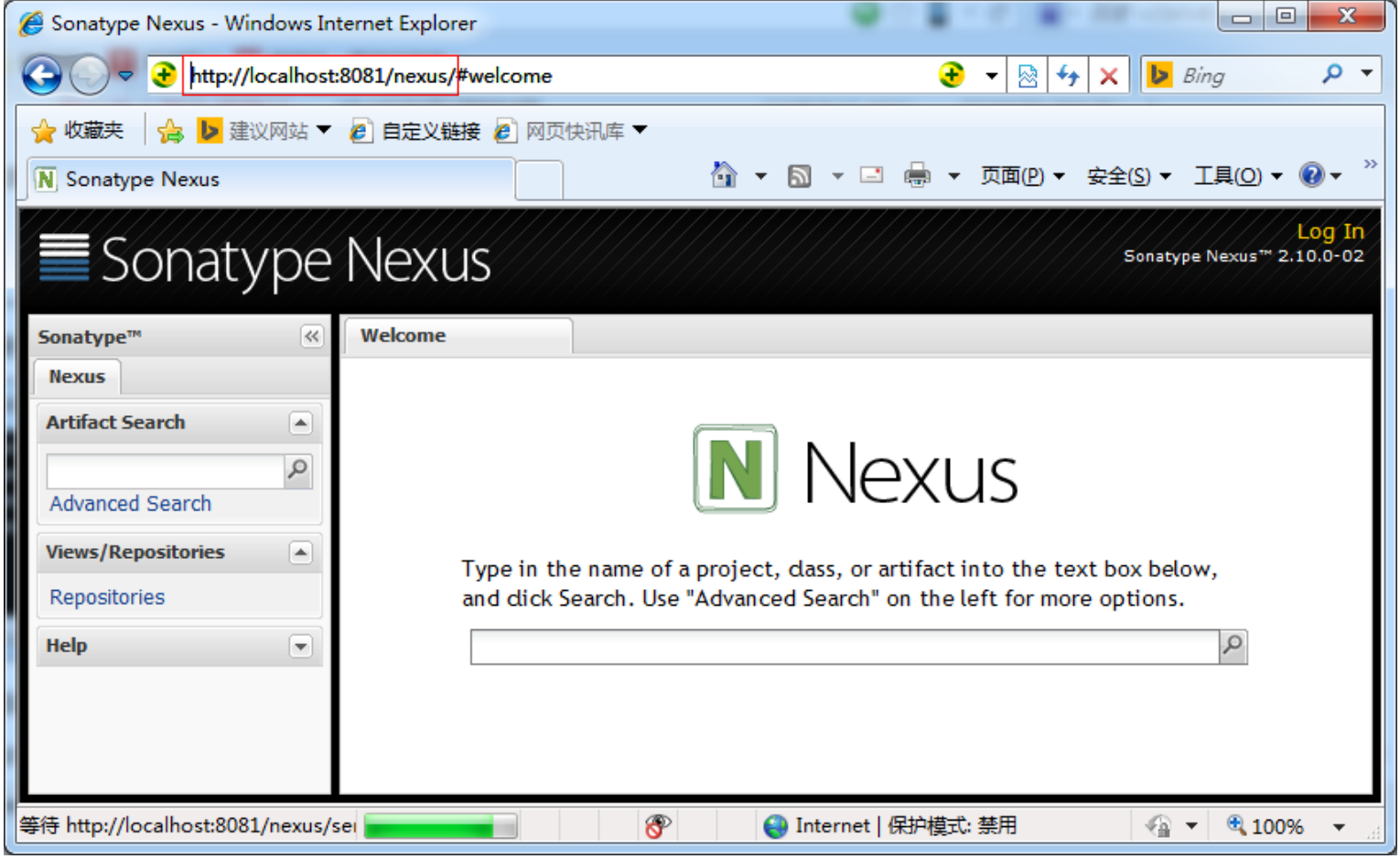
启动服务过程如下图所示：



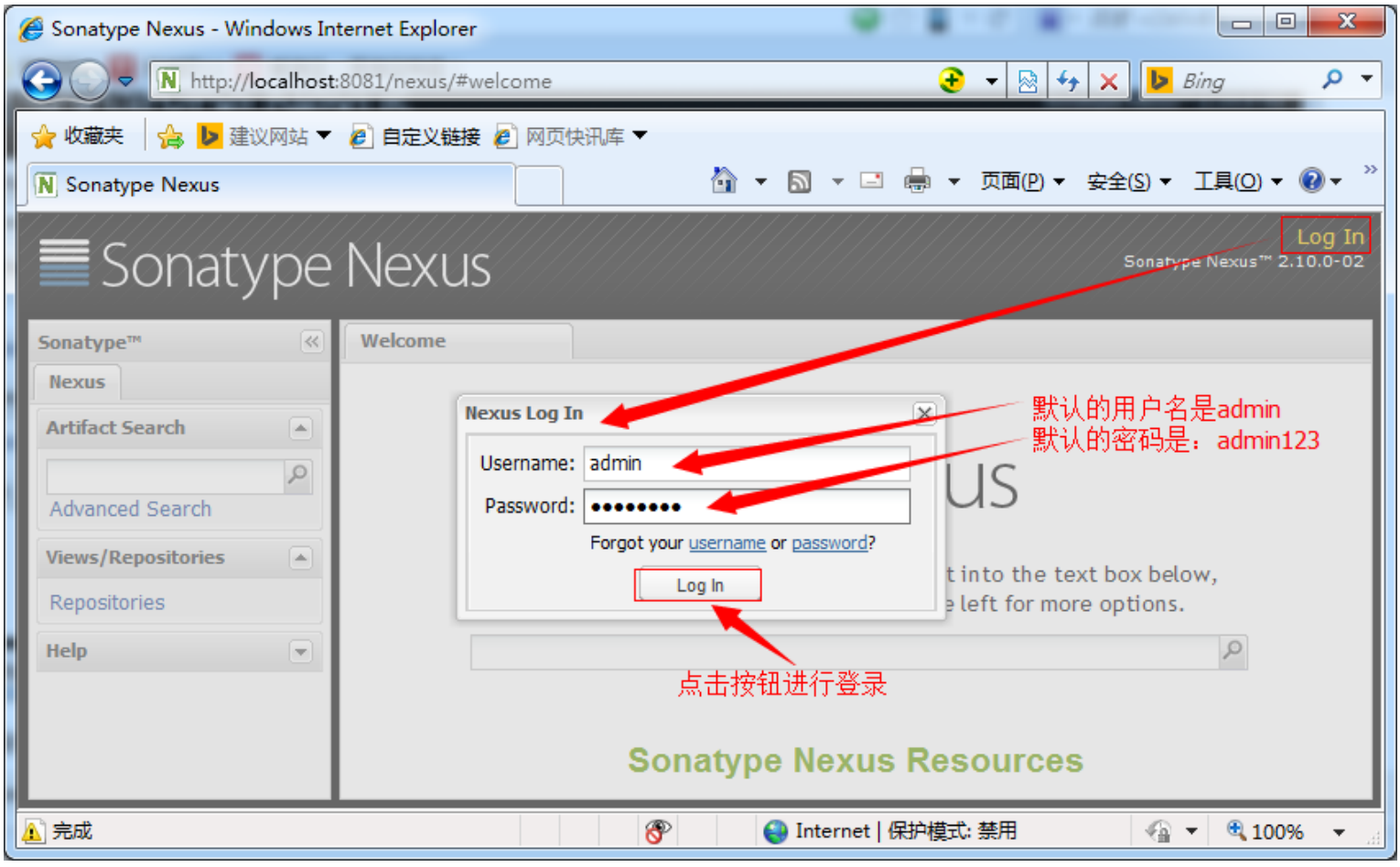
注意，启动服务时必须要保证8081端口没有被其他程序占用，否则是无法启动服务的。

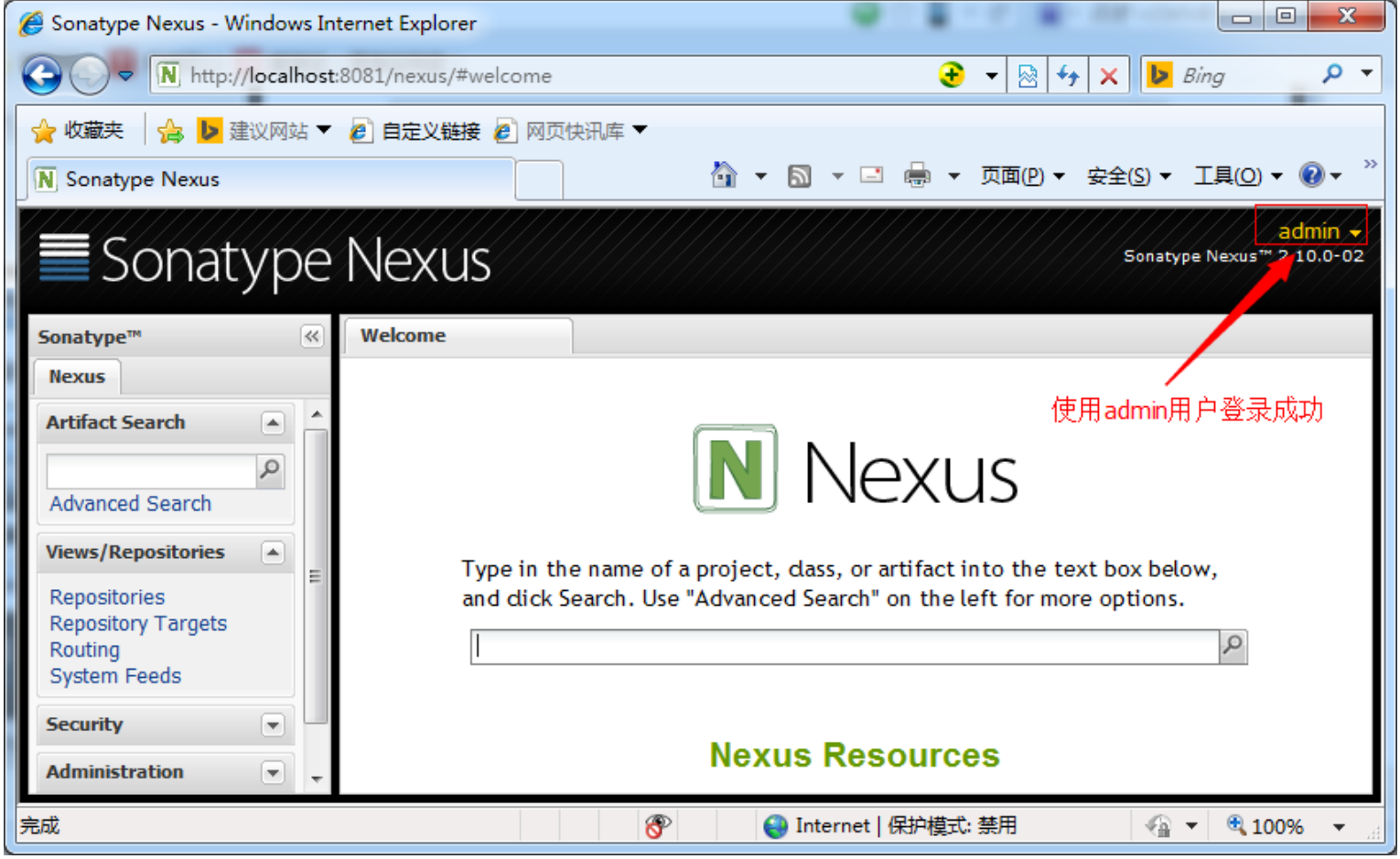
2.3、测试nexus是否安装成功

nexus服务启动之后，在浏览器中输入"http://localhost:8081/nexus/" 进行访问，如下图所示：



在右上角有个【Log in】 的超链接，点击登录，默认的用户名是 admin 密码是 admin123





可以看到，nexus可以正常访问和登录，这就说明nexus已经安装成功。

学习过Spring框架的人一定都会听过Spring的IoC(控制反转) 、DI(依赖注入)这两个概念，对于初学Spring的人来说，总觉得IoC 、DI这两个概念是模糊不清的，是很难理解的，今天和大家分享网上的一些技术大牛们对Spring框架的IOC的理解以及谈谈我对Spring Ioc的理解。

一、分享Iteye的开涛对Ioc的精彩讲解

首先要分享的是Iteye的开涛这位技术牛人对Spring框架的IOC的理解，写得非常通俗易懂，以下内容全部来自原文，原文地址：http://jinnianshilongnian.iteye.com/blog/1413846

1.1、IoC是什么

Ioc—Inversion of Control，即“控制反转”，不是什么技术，而是一种设计思想。在Java开发中，Ioc意味着将你设计好的对象交给容器控制，而不是传统的在你的对象内部直接控制。如何理解好Ioc呢？理解好Ioc的关键是要明确“谁控制谁，控制什么，为何是反转（有反转就应该有正转了），哪些方面反转了”，那我们来深入分析一下：

●谁控制谁，控制什么：传统Java SE程序设计，我们直接在对象内部通过new进行创建对象，是程序主动去创建依赖对象；而IoC是有专门一个容器来创建这些对象，即由Ioc容器来控制对 象的创建；谁控制谁？当然是IoC 容器控制了对象；控制什么？那就是主要控制了外部资源获取（不只是对象包括比如文件等）。

●为何是反转，哪些方面反转了：有反转就有正转，传统应用程序是由我们自己在对象中主动控制去直接获取依赖对象，也就是正转；而反转则是由容器来帮忙创建及注入依赖对象；为何是反转？因为由容器帮我们查找及注入依赖对象，对象只是被动的接受依赖对象，所以是反转；哪些方面反转了？依赖对象的获取被反转了。

用图例说明一下，传统程序设计如图2-1，都是主动去创建相关对象然后再组合起来：

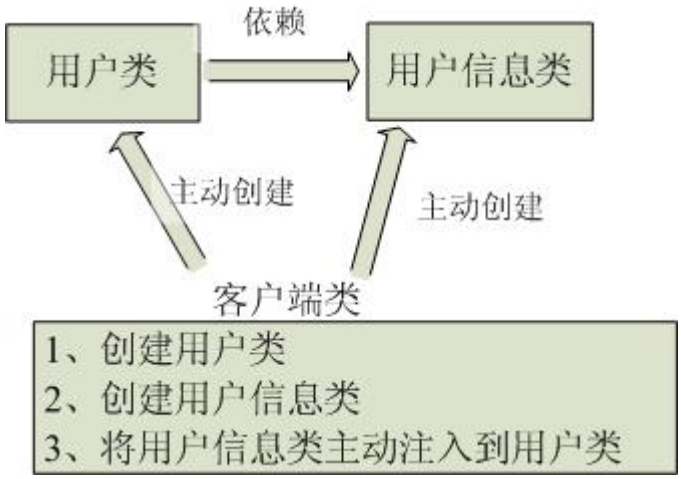


图1-1 传统应用程序示意图

当有了IoC/DI的容器后，在客户端类中不再主动去创建这些对象了，如图2-2所示:

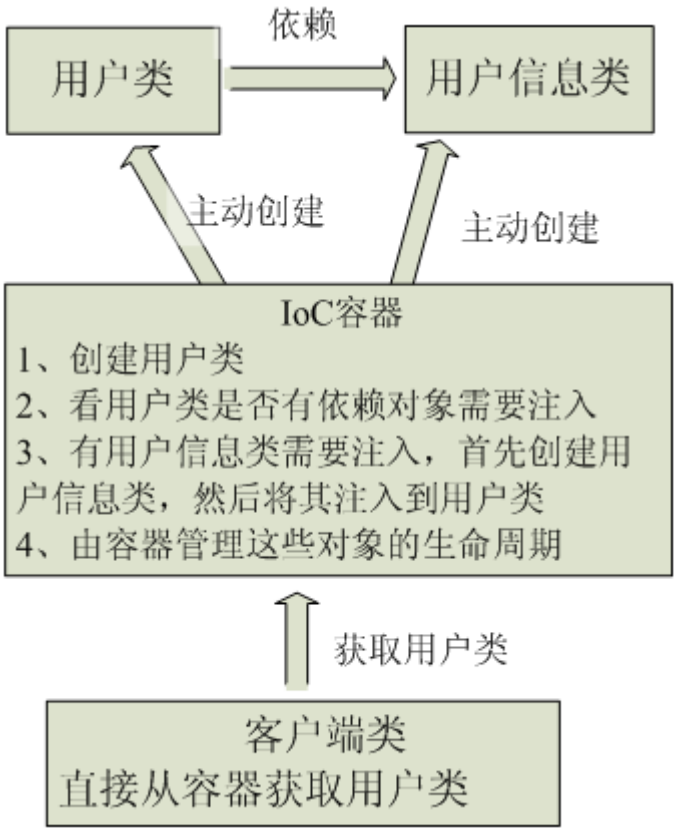


图1-2有IoC/DI容器后程序结构示意图

1.2、IoC能做什么

IoC 不是一种技术，只是一种思想，一个重要的面向对象编程的法则，它能指导我们如何设计出松耦合、更优良的程序。传统应用程序都是由我们在类内部主动创建依赖对象，从而导致类与类之间高耦合，难于测试；有了IoC容器后，把创建和查找依赖对象的控制权交给了容器，由容器进行注入组合对象，所以对象与对象之间是 松散耦合，这样也方便测试，利于功能复用，更重要的是使得程序的整个体系结构变得非常灵活。

其实IoC对编程带来的最大改变不是从代码上，而是从思想上，发生了“主从换位”的变化。应用程序原本是老大，要获取什么资源都是主动出击，但是在IoC/DI思想中，应用程序就变成被动的了，被动的等待IoC容器来创建并注入它所需要的资源了。

IoC很好的体现了面向对象设计法则之一—— 好莱坞法则：“别找我们，我们找你”；即由IoC容器帮对象找相应的依赖对象并注入，而不是由对象主动去找。

1.3、IoC和DI

DI—Dependency Injection，即“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容器注入应用程序某个对象，应用程序依赖的对象；

●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由什么关系呢？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关系），所以2004年大师级人物Martin Fowler又给出了一个新的名字：“依赖注入”，相对IoC 而言，“依赖注入”明确描述了“被注入对象依赖IoC容器配置依赖对象”。

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。

二、分享Bromon的blog上对IoC与DI浅显易懂的讲解

2.1、IoC(控制反转)

首先想说说IoC（Inversion of Control，控制反转）。这是spring的核心，贯穿始终。所谓IoC，对于spring框架来说，就是由spring来负责控制对象的生命周期和对象间的关系。这是什么意思呢，举个简单的例子，我们是如何找女朋友的？常见的情况是，我们到处去看哪里有长得漂亮身材又好的mm，然后打听她们的兴趣爱好、qq号、电话号、ip号、iq号……，想办法认识她们，投其所好送其所要，然后嘿嘿……这个过程是复杂深奥的，我们必须自己设计和面对每个环节。传统的程序开发也是如此，在一个对象中，如果要使用另外的对象，就必须得到它（自己new一个，或者从JNDI中查询一个），使用完之后还要将对象销毁（比如Connection等），对象始终会和其他的接口或类藕合起来。

那么IoC是如何做的呢？有点像通过婚介找女朋友，在我和女朋友之间引入了一个第三者：婚姻介绍所。婚介管理了很多男男女女的资料，我可以向婚介提出一个列表，告诉它我想找个什么样的女朋友，比如长得像李嘉欣，身材像林熙雷，唱歌像周杰伦，速度像卡洛斯，技术像齐达内之类的，然后婚介就会按照我们的要求，提供一个mm，我们只需要去和她谈恋爱、结婚就行了。简单明了，如果婚介给我们的人选不符合要求，我们就会抛出异常。整个过程不再由我自己控制，而是有婚介这样一个类似容器的机构来控制。Spring所倡导的开发方式就是如此，所有的类都会在spring容器中登记，告诉spring你是个什么东西，你需要什么东西，然后spring会在系统运行到适当的时候，把你要的东西主动给你，同时也把你交给其他需要你的东西。所有的类的创建、销毁都由 spring来控制，也就是说控制对象生存周期的不再是引用它的对象，而是spring。对于某个具体的对象而言，以前是它控制其他对象，现在是所有对象都被spring控制，所以这叫控制反转。

2.2、DI(依赖注入)

IoC的一个重点是在系统运行中，动态的向某个对象提供它所需要的其他对象。这一点是通过DI（Dependency Injection，依赖注入）来实现的。比如对象A需要操作数据库，以前我们总是要在A中自己编写代码来获得一个Connection对象，有了 spring我们就只需要告诉spring，A中需要一个Connection，至于这个Connection怎么构造，何时构造，A不需要知道。在系统运行时，spring会在适当的时候制造一个Connection，然后像打针一样，注射到A当中，这样就完成了对各个对象之间关系的控制。A需要依赖 Connection才能正常运行，而这个Connection是由spring注入到A中的，依赖注入的名

字就这么来的。那么DI是如何实现的呢？ Java 1.3之后一个重要特征是反射（reflection），它允许程序在运行的时候动态的生成对象、执行对象的方法、改变对象的属性，spring就是通过反射来实现注入的。 理解了IoC和DI的概念后，一切都将变得简单明了，剩下的工作只是在spring的框架中堆积木而已。三、我对IoC(控制反转)和DI(依赖注入)的理解 在平时的java应用开发中，我们要实现某一个功能或者说是完成某个业务逻辑时至少需要两个或以上的对象来协作完成，在没有使用Spring的时候，每个对象在需要使用他的合作对象时，自己均要使用像new object() 这样的语法来将合作对象创建出来，这个合作对象是由自己主动创建出来的，创建合作对象的主动权在自己手上，自己需要哪个合作对象，就主动去创建，创建合作对象的主动权和创建时机是由自己把控的，而这样就会使得对象间的耦合度高了，A对象需要使用合作对象B来共同完成一件事，A要使用B，那么A就对B产生了依赖，也就是A和B之间存在一种耦合关系，并且是紧密耦合在一起，而使用了Spring之后就不一样了，创建合作对象B的工作是由Spring来做的，Spring创建好B对象，然后存储到一个容器里面，当A对象需要使用B对象时，Spring就从存放对象的那个容器里面取出A要使用的那个B对象，然后交给A对象使用，至于Spring是如何创建那个对象，以及什么时候创建好对象的，A对象不需要关心这些细节问题(你是什么时候生的，怎么生出来的我可不关心，能帮我干活就行)，A得到Spring给我们的对象之后，两个人一起协作完成要完成的工作即可。 所以控制反转 IoC(Inversion of Control)是说创建对象的控制权进行转移，以前创建对象的主动权和创建时机是由自己把控的，而现在这种权力转移到第三方，比如转移交给了IoC容器，它就是一个专门用来创建对象的工厂，你要什么对象，它就给你什么对象，有了 IoC容器，依赖关系就变了，原先的依赖关系就没了，它们都依赖IoC容器了，通过IoC容器来建立它们之间的关系。

这是我对Spring的IoC(控制反转)的理解。DI(依赖注入)其实就是IOC的另外一种说法，DI是由Martin Fowler 在2004年初的一篇论文中首次提出的。他总结：控制的什么被反转了？就是：获得依赖对象的方式反转了。

四、小结

对于Spring Ioc这个核心概念，我相信每一个学习Spring的人都会有自己的理解。这种概念上的理解没有绝对的标准答案，仁者见仁智者见智。如果有理解不到位或者理解错的地方，欢迎广大园友指正！