**By Brother.Lee 20180508 Shanghai**

mavne

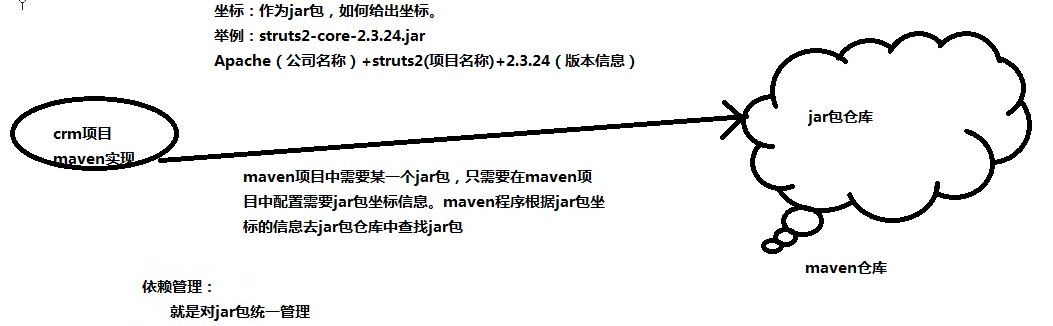
# maven基础

maven是项目构建工具;用来依赖管理!

## 1:maven的好处

使用传统的项目开发crm项目,项目大小40M;

同样的代码,实现功能都一样,maven项目如何做到的?可以初步推断maven项目中一定没有jar包,没有jar包的maven项目如何运行?



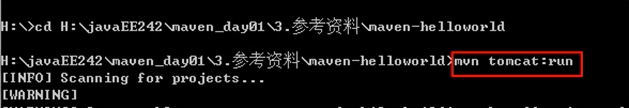
## 2:maven的好处如何实现

maven的两大核心:

\*\*依赖管理:对jar包管理过程

\*\*项目构建:项目在编码完成后,对项目进行编译,测试,打包,部署一系列的操作都通过命令来实现.

通过maven命令将web项目发布到tomcat



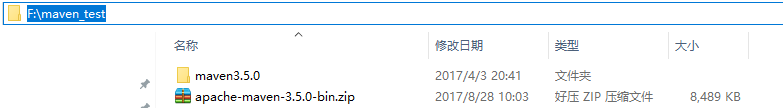
## 3:maven安装,配置本地仓库

maven程序安装前提:maven程序java开发,它的运行依赖jdk;

### 3.1:maven的下载安装

#### 3.1.1:找到资料;

#### 3.1.2:解压到本地磁盘(解压目录不要有中文,不要有空格)



#### 3.1.3:配置环境变量

jdk环境变量:JAVA\_HOME

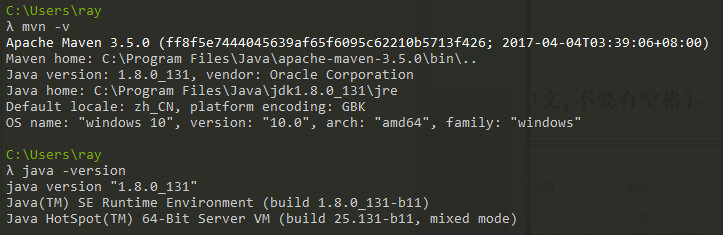
MAVEN\_HOME

F:\maven\_test\maven3.5.0

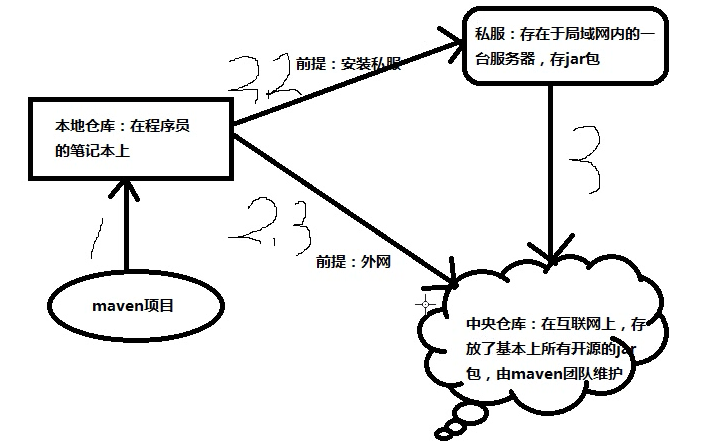
Path

环境变量中添加%MAVEN\_HOME%/bin

查询mavne版本信息



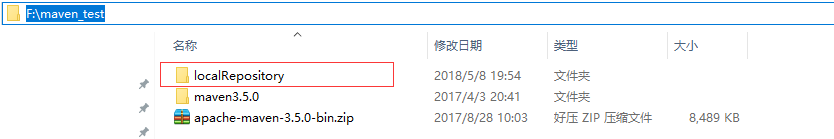
### 3.2配置本地仓库



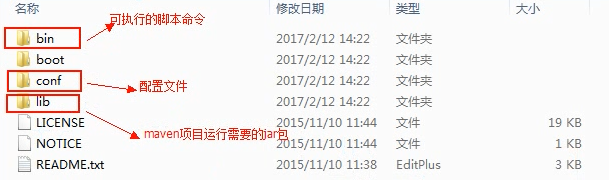
配置本地仓库:让maven程序知道仓库在哪

<localRepository>F:\maven\_test\localRepository</localRepository>

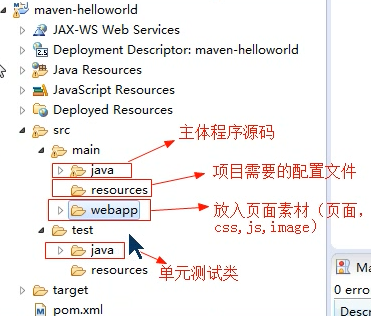




## 4:maven项目标准目录结构



对项目文件进行细分



## 5:maven的常用命令

### 5.1:clean:清理

将项目根目录下target目录清理掉;

### 5.2:compile:编译

将项目中.java文件编译为.class文件

### 5.3:test:单元测试

单元测试类名有要求:XxxxTest.java

将项目根目录下src/test/java目录下的单元测试类都会执行;

### 5.4:package:打包

web project ---war包

java project ---jar包

将项目打包,打包项目根目录下target目录

### 5.5:install:安装

解决本地多个项目公用一个jar包;

打包到本地仓库

### 5.6:maven项目的生命周期(了解)

在maven中存在三套生命周期,每一套生命周期相互独立,互不影响;

**在一套生命周期内,执行后面的命令,前面的操作会自动执行;**

cleanLifeCycle:清理生命周期

clean

defaultLifeCycle:默认生命周期

compile,test,package,install,deploy

siteLifeCycle:站点生命周期

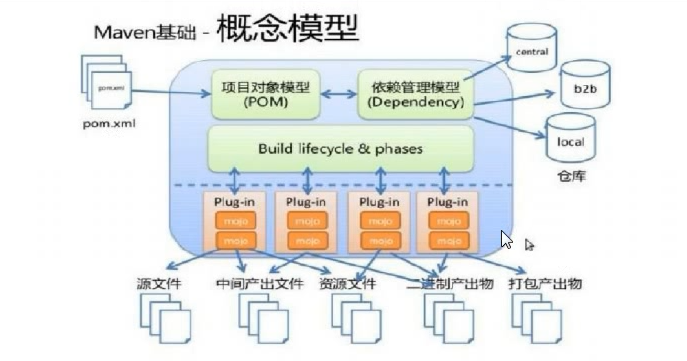
site

### 5.7:概念的模型

两个核心:

\*\*依赖管理:对jar的管理;

\*\*项目构建:通过命令进行项目构建

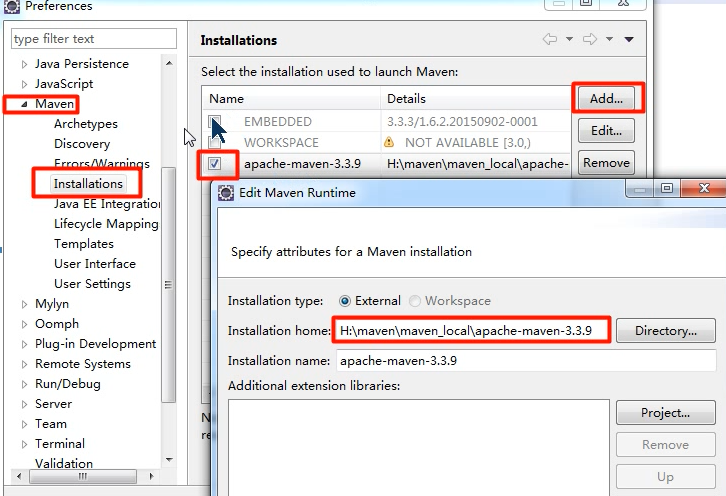


## 6:maven整合web项目案例

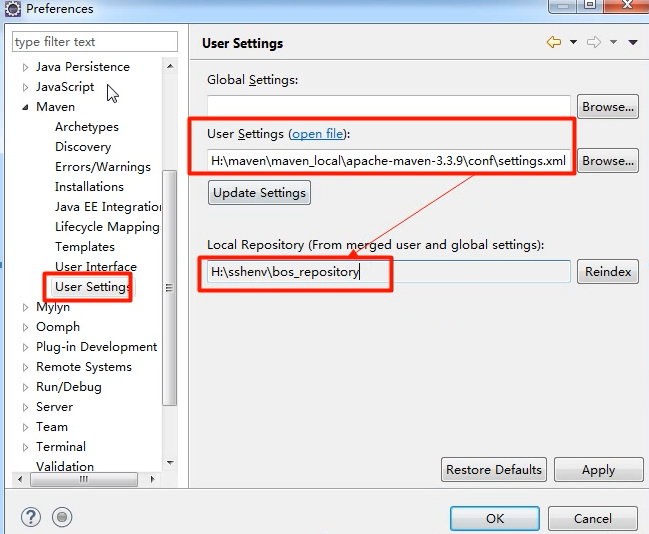
### 6.1:配置eclipse中maven环境

1:配置m2e插件,Mars2版本自带maven插件;

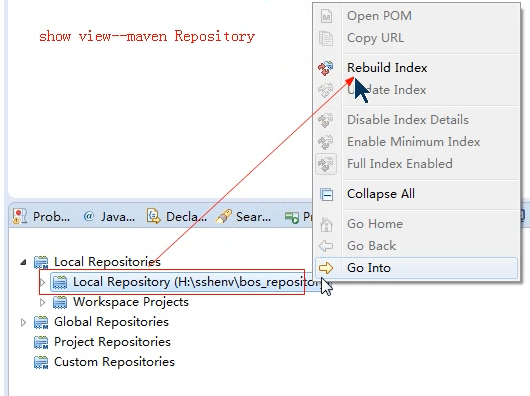
2:需要配置maven程序



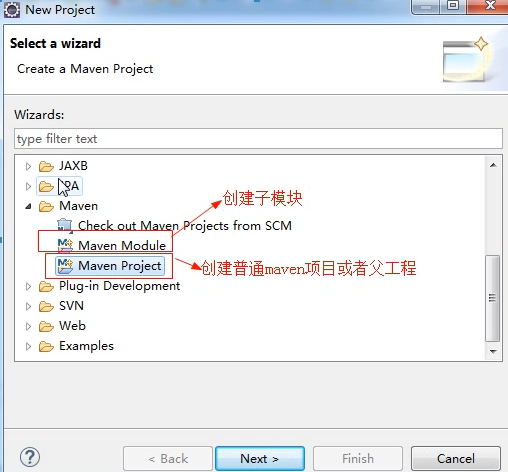
3:配置userSetting:让eclipse知道maven仓库位置.

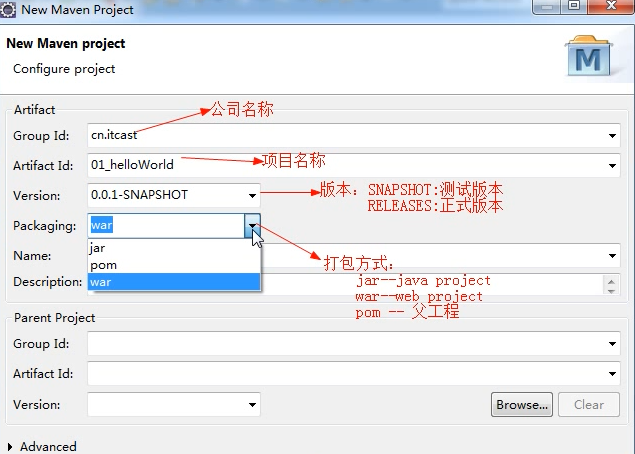


4:构建索引



### 6.2:maven整合servlet





#### 6.2.1:设置jdk编译版本为1.7/1.8的插件配置

添加项目jdk编译插件

##### 方式一：settings.xml 配置

打开 %maven%/conf/settings.xml 文件并编辑它（%maven% 表示 maven 的根目录） ：

<**profiles**>  
 <**profile**>  
 <**id**>development</**id**>  
 <**activation**>  
 <**jdk**>1.7</**jdk**>  
 <**activeByDefault**>true</**activeByDefault**>  
 </**activation**>  
 <**properties**>  
 <**maven.compiler.source**>1.7</**maven.compiler.source**>  
 <**maven.compiler.target**>1.7</**maven.compiler.target**>  
 <**maven.compiler.compilerVersion**>1.7</**maven.compiler.compilerVersion**>  
 </**properties**>  
 </**profile**>  
</**profiles**>

找到 <profiles> 节点，并添加如上配置（本机 jdk 1.7.0\_79 版本，配置时修改成你本机的 jdk 版本）。保存并关闭。重启 eclipse 即可。  
重启后配置生效。

##### 方式二：pom.xml 配置

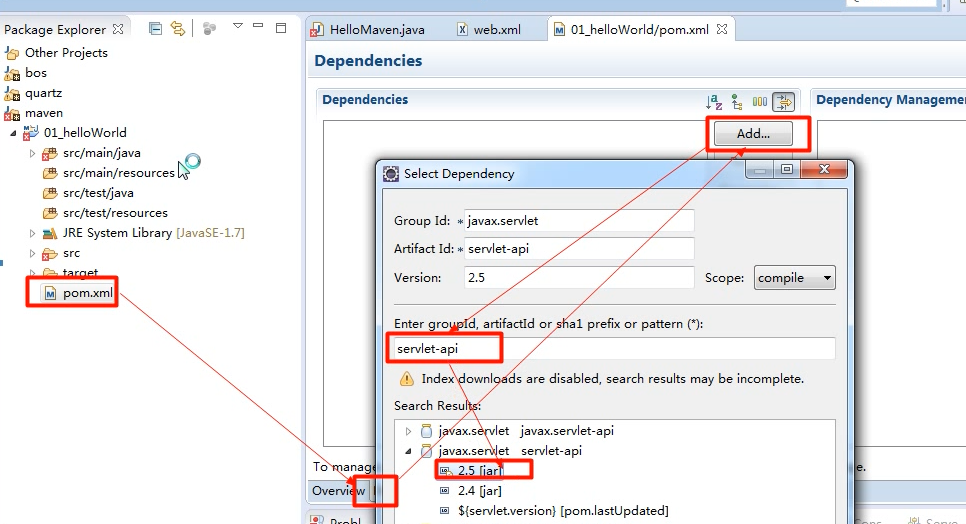
在 <build> 节点添加如下配置（本机 jdk 1.7.0\_79 版本，配置时修改成你本机的 jdk 版本）：

<**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-compiler-plugin</**artifactId**>  
 <**configuration**>  
 <**source**>1.7</**source**>  
 <**target**>1.7</**target**>  
 <**enconding**>UTF-8</**enconding**>  
 </**configuration**>  
 </**plugin**>  
 </**plugins**>  
</**build**>

配置完成后，需要执行一次更新项目配置的动作。选中项目 --> 右键 --> Maven --> Update Project（eclipse luna 版本，helios 或indigo 版本选择 Update Project Configuration）。  
pom.xml 仅作用于当前项目（每个项目都需要配一次），而 settings.xml 的配置是全局的（仅需要配一次）。以上两种配置方式中选其一即可。

#### 6.2.2:创建servlet,servlet编译报错,原因:缺失servlet-api-xxx.jar

#### 6.2.3:查找依赖



### 6.3:依赖的范围(了解)

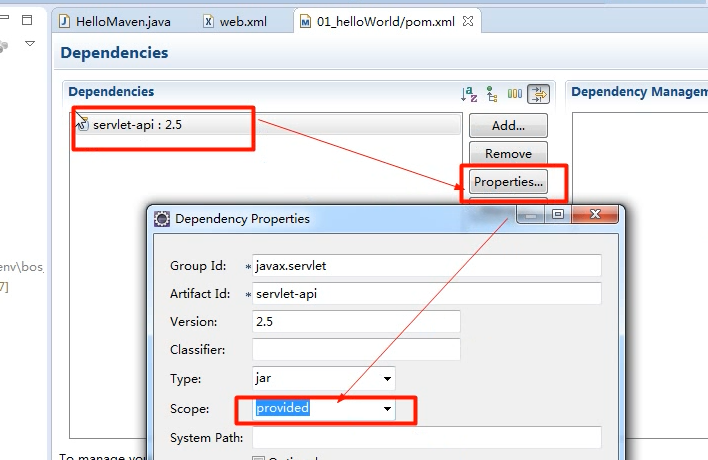


添加依赖范围:默认是compile

Provided:运行部署到tomcat不再需要

如果将servlet-api.jar设置为compile,打包后包含servlet-api.jar;war包部署到tomcat中存在servlet-api.jar包冲突.导致运行失败;

jar包冲突的解决办法:



**总结:如果使用到tomcat自带jar包,将项目中雨来作用范围设置为<scope>provided</scope>**

# Maven实战

## 1:传递依赖冲突解决(了解)

传递依赖:A依赖B,B依赖C;C就是A的传递依赖;B是A的直接依赖;

### 1.1:Maven自己调节原则

#### 1.1.1:第一声明者优先原则

谁先定义的就用谁的传递依赖.

#### 1.1.2:路径近者优先原则

直接依赖级别高于传递依赖;

### 1.2:排除依赖

<**dependencies**>  
 <**dependency**>  
 <**groupId**>com.cc.maven</**groupId**>  
 <**artifactId**>project-b</**artifactId**>  
 <**version**>1.0.0</**version**>  
 <**exclusions**>  
 <**exclusion**>  
 <**groupId**>com.cc.maven</**groupId**>  
 <**artifactId**>project-c</**artifactId**>  
 </**exclusion**>  
 </**exclusions**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.cc.maven</**groupId**>  
 <**artifactId**>project-b</**artifactId**>  
 <**version**>1.1.0</**version**>  
 </**dependency**>  
</**dependencies**>

### 1.3:版本的锁定(推荐使用)

版本锁定:指定项目中依赖的版本

//只是对版本进行管理，不会实际引入jar  
<**dependencyManagement**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-core</**artifactId**>  
 <**version**>3.2.7</**version**>  
 </**dependency**>  
 </**dependencies**>  
</**dependencyManagement**>  
  
//会实际下载jar包  
<**dependencies**>  
<**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-core</**artifactId**>  
</**dependency**>  
</**dependencies**>

在pom.xml定义properties标签  
<**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**spring.version**>1.2.6</**spring.version**>   
</**properties**>

以上内容就改成了  
<**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-core</**artifactId**>  
 <**version**>${spring.version}</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-aop</**artifactId**>  
 <**version**>${spring.version}</**version**>  
</**dependency**>

确实很清晰明了。

## 3:通过maven对项目进行拆分,聚合(重点)

### 3.1:创建父工程:

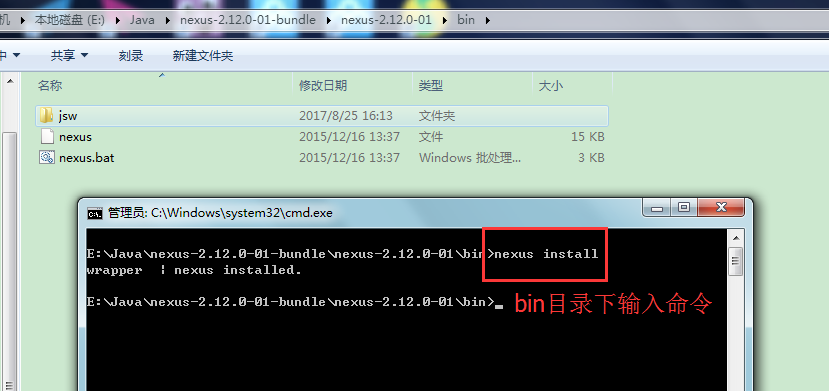
创建好父工程目录结构:**只有pom.xml,可以推断父工程不进行编码.**

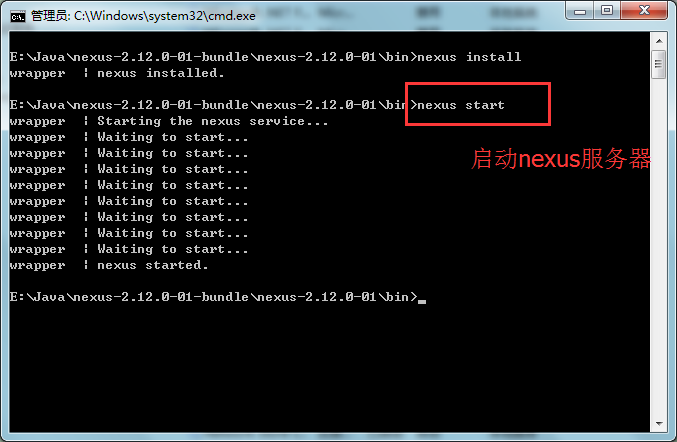
1:项目需要的依赖的信息,在父工程中定义,子模块继承过程;

2:将各个子模块聚合到一起;

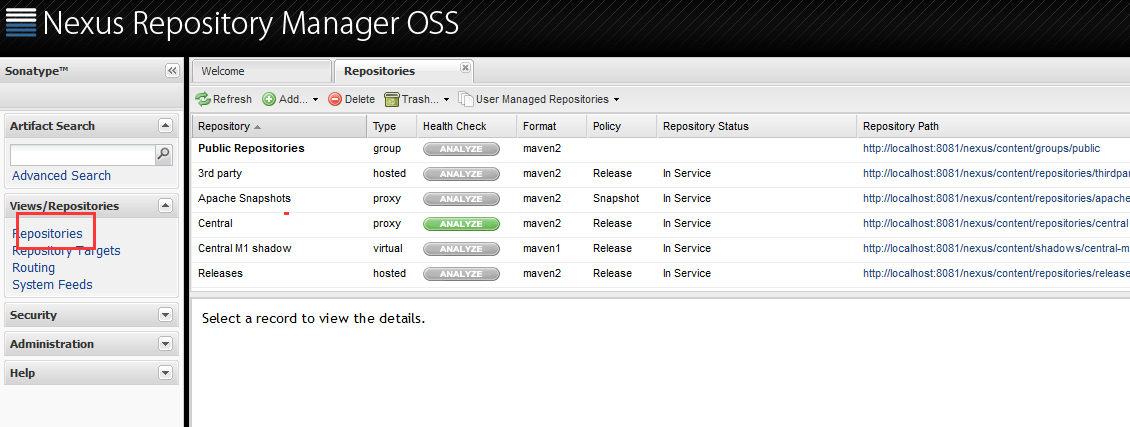
### 3.2:将创建的父工程发布到本地仓库

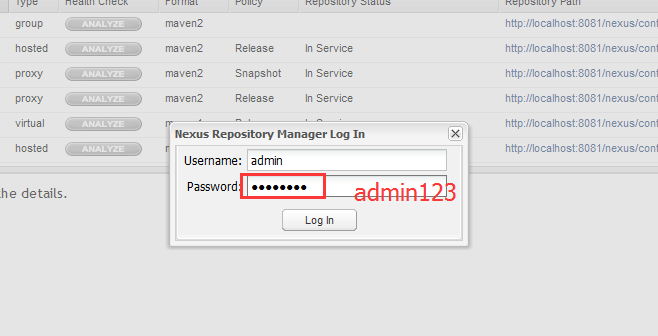
## 10.私服安装





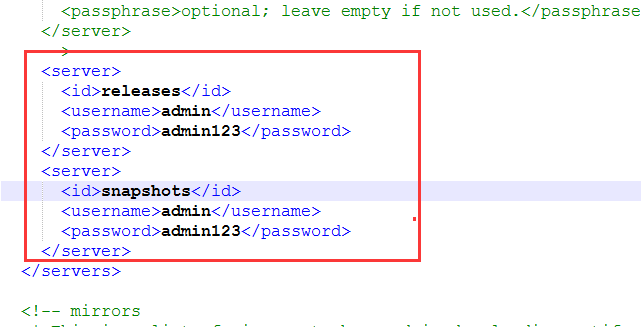
浏览器访问：<http://localhost:8081/nexus/>





## 11.上传jar包到私服

maven的配置文件中



然后在将要上传的项目的pom.xml中配置jar包上传路径url

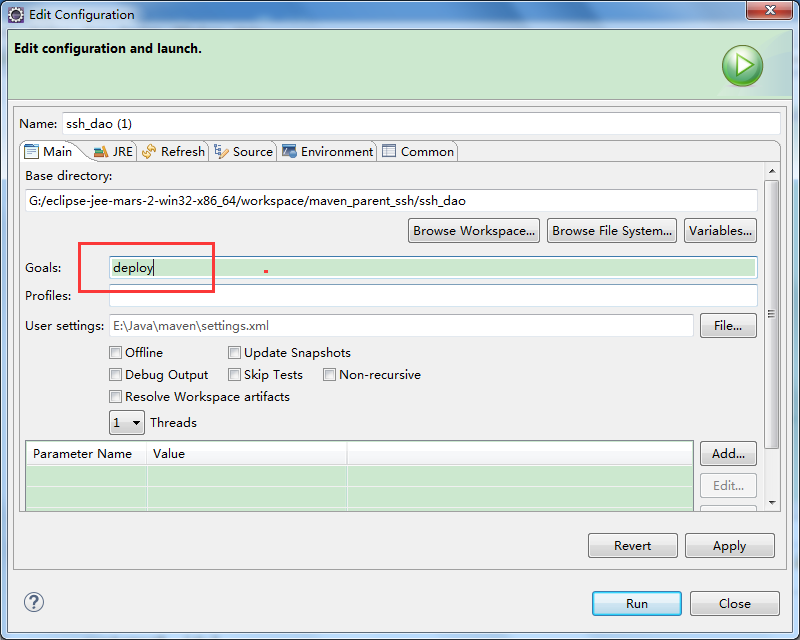
以下代码复制到要上传私服的项目pom.xml中

<**distributionManagement**>  
 <**repository**>  
 <**id**>releases</**id**>  
 <**url**>http://localhost:8081/nexus/content/repositories/releases/</**url**>  
 </**repository**>  
 <**snapshotRepository**>  
 <**id**>snapshots</**id**>  
 <**url**>http://localhost:8081/nexus/content/repositories/snapshots/</**url**>  
 </**snapshotRepository**>  
</**distributionManagement**>

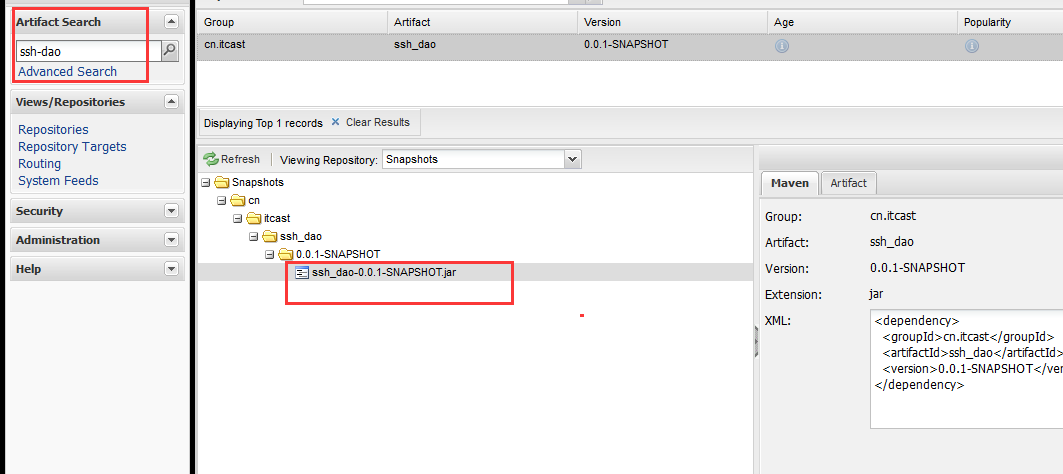
======================================================

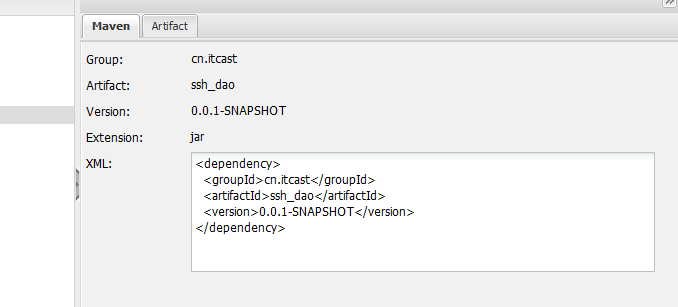
执行命令发布项目到私服（上传）

deploy









## 12.从私服上下载jar包到本地仓库

继续修改maven的配置文件

添加以下内容

<**profile**>  
 <!--profile的id-->  
 <**id**>dev</**id**>  
 <**repositories**>  
 <**repository**>  
 <!--仓库id，repositories可以配置多个仓库，保证id不重复-->  
 <**id**>nexus</**id**>  
 <!--仓库地址，即nexus仓库组的地址-->  
 <**url**>http://localhost:8081/nexus/content/groups/public/</**url**>  
 <!--是否下载releases构件-->  
 <**releases**>  
 <**enabled**>true</**enabled**>  
 </**releases**>  
 <!--是否下载snapshots构件-->  
 <**snapshots**>  
 <**enabled**>true</**enabled**>  
 </**snapshots**>  
 </**repository**>  
 </**repositories**>  
 <**pluginRepositories**>  
 <!-- 插件仓库，maven的运行依赖插件，也需要从私服下载插件 -->  
 <**pluginRepository**>  
 <!-- 插件仓库的id不允许重复，如果重复后边配置会覆盖前边 -->  
 <**id**>public</**id**>  
 <**name**>Public Repositories</**name**>  
 <**url**>http://localhost:8081/nexus/content/groups/public/</**url**>  
 </**pluginRepository**>  
 </**pluginRepositories**>  
</**profile**>  
 还需要激活一下模板  
<**activeProfiles**>  
<**activeProfile**>dev</**activeProfile**>  
</**activeProfiles**>

==============================================================

附加：可以切换到 maven 控制台 Maven Console

