# Maven

1. 本地存储位置修改：

<localRepository>D:\maven\_repository</localRepository>

2、maven国内仓库配置：

编辑/conf/setting.xml文件，在mirrors节点下添加一下配置信息：

<mirror>

<id>alimaven</id>

<name>aliyun maven</name>

<url>http://maven.aliyun.com/nexus/content/groups/public/</url>

<mirrorOf>central</mirrorOf>

</mirror>

1. pom.xml配置信息：

-| “<modelVersion>4.0.0</modelVersion>” : pom模块的版本信息。

-| “<groupId>cn.mldn</groupId>” ：项目所属的组织编号，通常都是域名

-| “<artifactId>demo</artifactId>” ：项目名称，主要是生成日后的jar或war

-| “<version>0.0.1</version>” : 项目编译的版本号

-| “<packaging>jar</packaging>” : 项目的打包形式 ,一般是jar或是war

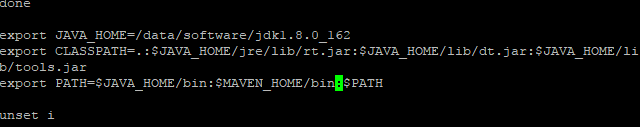
1. 执行生成的jar:java -cp target/demo-0.0.1-SNAPSHOT.jar cn.mldn.demo.App
2. Linux系统maven配置：

使用mvn编译java文件，需要jdk环境。

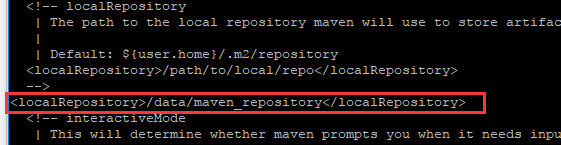
1. 解压maven压缩包
   1. tar zxvf apache-maven-3.5.3-bin.tar.gz
   2. 编辑文件vim/etc/profile，追加一下配置：

export MAVEN\_HOME=/data/software/apache-maven-3.5.3

在PATH后面追加$MAVEN\_HOME/bin，如：



* 1. 重新加载profile文件:source /etc/profile,执行mvn -version验证。
  2. 修改maven配置文件：
     1. 修改settings.xml中下载资源存放位置：



* + 1. 修改中央仓库镜像：

<mirror>

  <id>nexus-aliyun</id>

  <name>nexus-aliyun</name>

  <url>http://maven.aliyun.com/nexus/content/groups/public</url>

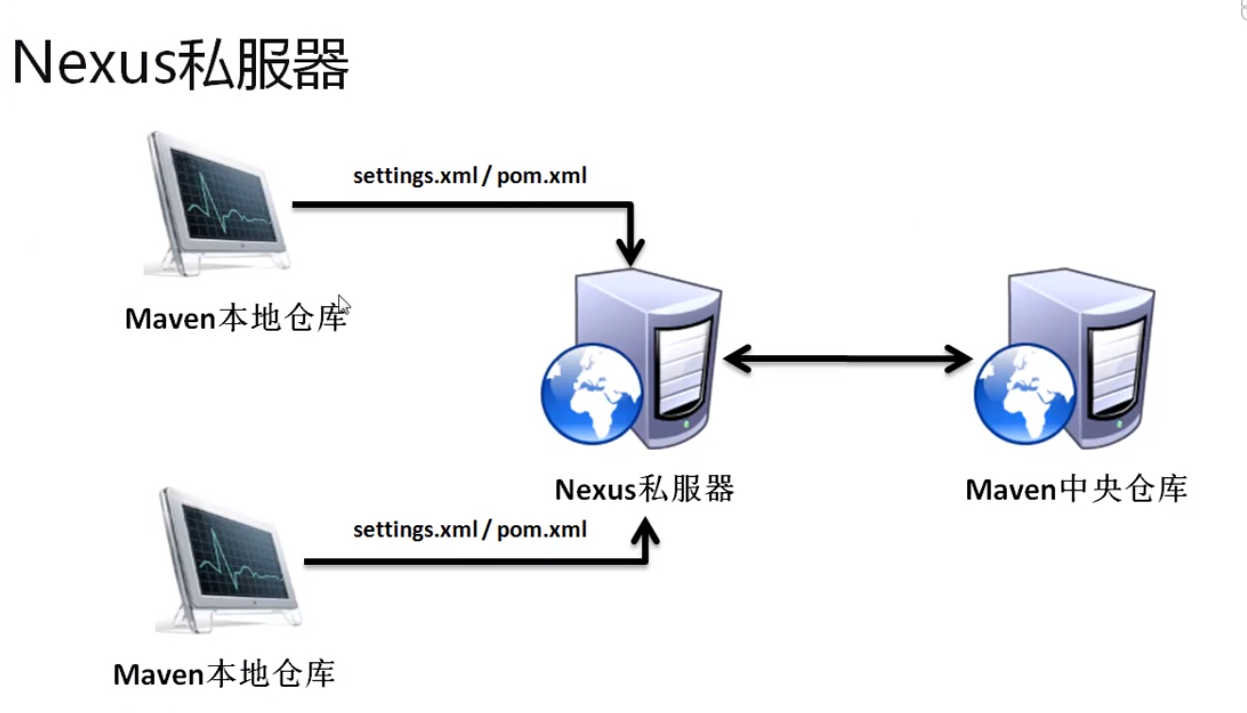
  <mirrorOf>central</mirrorOf>

</mirror>

1. 搭建Nexus私服

多个开发者间存在重复下载相同的开发包，且网络也会受到影响。所有开发者都使用同一个nexus服务器上的开发包，如果某一个开发包不存在，那么maven中央仓库会自动下载该开发包到nexus服务器上，其他开发者不用再通过网络去下载，直接使用nexus上的即可。

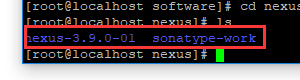
* 1. Nexus服务器的基本结构



* 1. 从https://www.sonatype.com/download-oss-sonatype上下载nexus。
     1. 在Linux系统上配置好jdk
     2. 上传压缩包nexus-3.9.0-01-unix.tar.gz到Linux
     3. 解压压缩包到/data/software/nexus目录下，

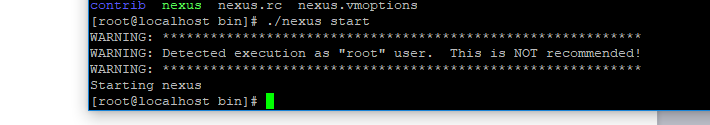
tar xzvf nexus-3.9.0-01-unix.tar.gz -C ./nexus

进入nexus目录后会发现生成了两个目录，分别是

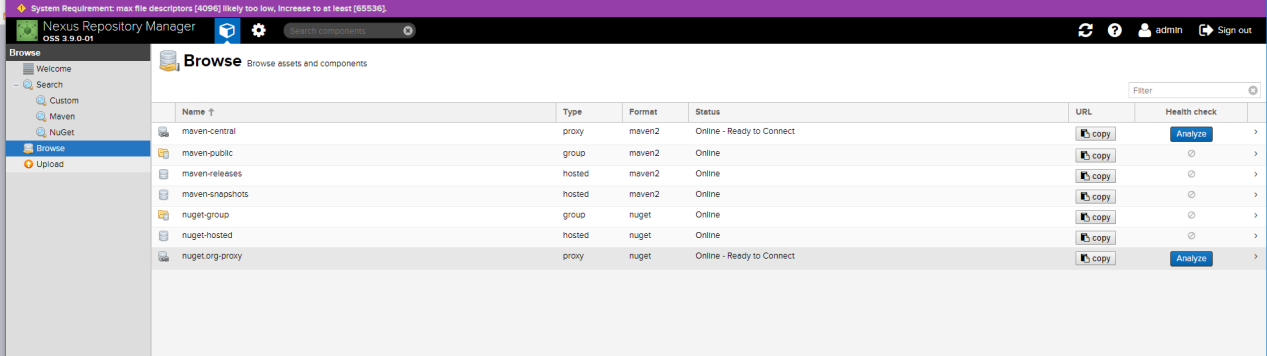


主要用的是nexus-3.9.0-01,sonatype-work是一些配置文件。

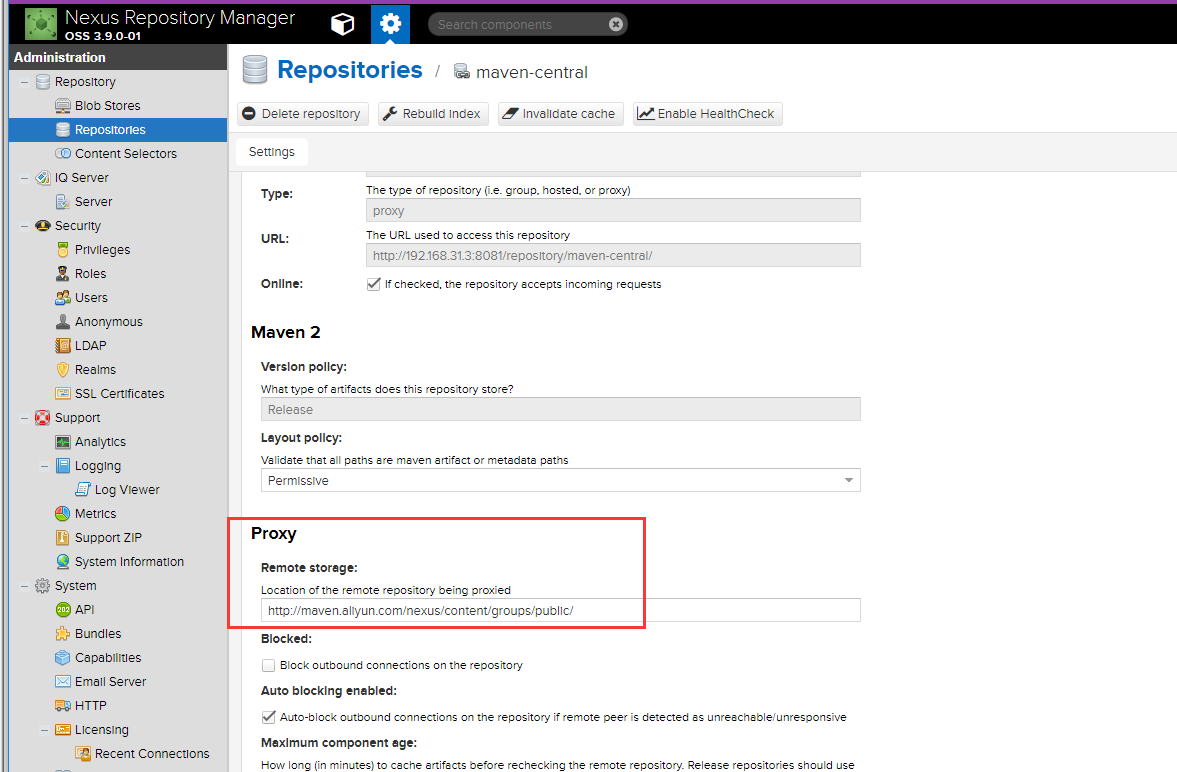
* + 1. 进入/data/software/nexus/nexus-3.9.0-01，启动nexus服务。
       1. Nexus自动集成了jetty容器。
       2. 进入bin目录中，编辑文件“nexus”,run\_as\_user=root(其实没用)



* + - 1. 启动服务：./nexus start
    1. 现在启动了nexus服务器
       1. 后台管理链接地址：<http://192.168.31.3:8081/nexus> (控制台里的配置信息，开发人员千万不能改，否则代码就废了)，若是访问不了，可以使用http://192.168.31.3:8081/
       2. 点击网站右上角的log in , admin/admin123登录。



* 1. 修改poxy：



* 1. 本地maven配置，配置仓库地址有两种模式：
     1. 配置项目pom.xml文件，添加一下内容：

<repositories>

<repository>

<id>nexus</id>

<name>nexus</name>

<url>http://192.168.1.103:8081/nexus/content/groups/public/</url>

<releases>

<enabled>true</enabled>

</releases>

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

</repositories>

 <pluginRepositories>  
        <pluginRepository>  
            <id>localmaven</id>  
            <name>localmaven</name>  
            <url>http://192.168.31.3:8081/repository/maven-public/</url>  
            <releases>  
                <enabled>true</enabled>  
            </releases>  
            <snapshots>  
                <enabled>true</enabled>  
            </snapshots>  
        </pluginRepository>  
    </pluginRepositories>

* + 1. 在setting.xml中设置：

<mirror>

<id>maven</id>

<name>maven</name>

<url>http://192.168.31.3:8081/repository/maven-public/</url>

<mirrorOf>central</mirrorOf>

</mirror>

* 1. 添加私服认证信息：

<!-- releases仓库的用户认证信息 --><server>

<id>releases</id>

<username>admin</username>

<password>admin123</password></server>

<!-- snapshots 快照认证信息 --><server>

<id>snapshots</id>

<username>admin</username>

<password>admin123</password></server>

* 1. 设置核心配置信息：

<!--Nexus配置仓库-->

<profiles>

<profile>

<id>nexus</id>

<repositories>

<repository>

<id>central</id>

<url>http://192.168.31.3:8081/repository/maven-public/</url>

<releases><enabled>true</enabled></releases>

<snapshots><enabled>true</enabled></snapshots>

</repository>

<repository>

<id>nexus</id>

<name>local private nexus</name>

<url>http://192.168.31.3:8081/repository/maven-public/</url>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>central</id>

<url>http://repo1.maven.org/maven2</url>

<releases><enabled>true</enabled></releases>

<snapshots><enabled>true</enabled></snapshots>

</pluginRepository>

<pluginRepository>

<id>nexus</id>

<name>Maven China Mirror</name>

<url>http://192.168.31.3:8081/repository/maven-public/</url>

<releases>

<enabled>true</enabled>

</releases>

<snapshots>

<enabled>false</enabled>

</snapshots>

</pluginRepository>

</pluginRepositories>

<properties>

<releases.url>http://192.168.31.3:8081/repository/maven-public/</releases.url>

</properties>

</profile>

</profiles>

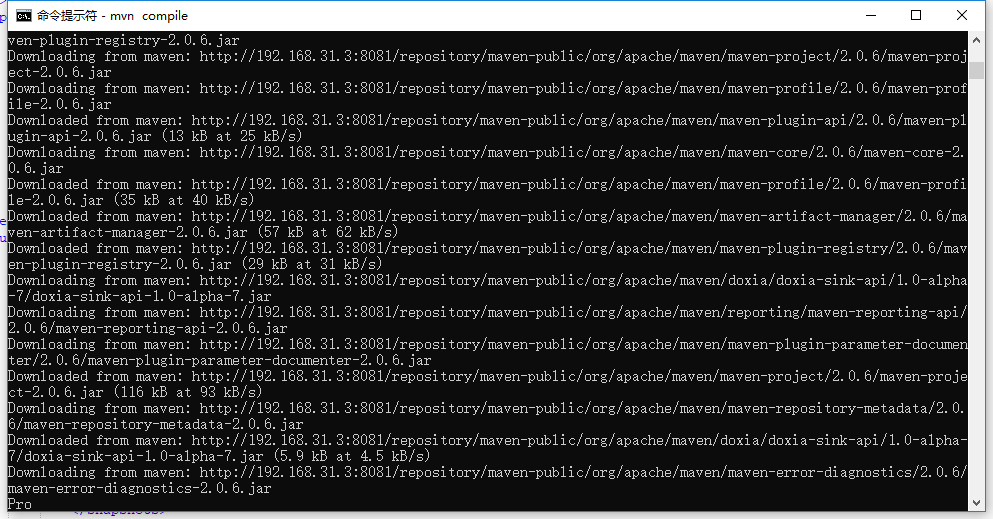
<!--激活Nexus仓库配置 -->

<activeProfiles>

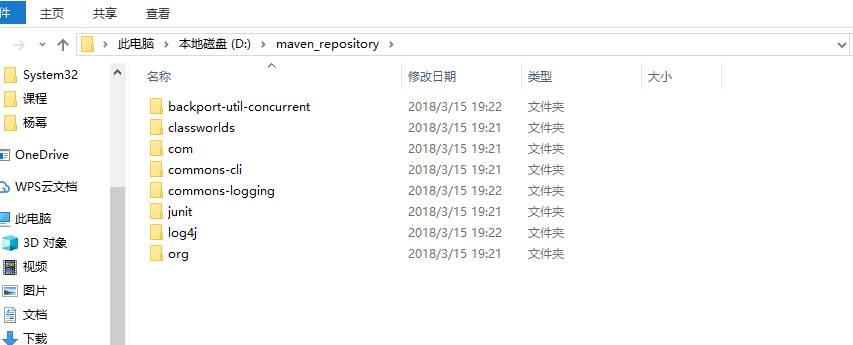
<activeProfile>nexus</activeProfile>

</activeProfiles>

编译meavn :

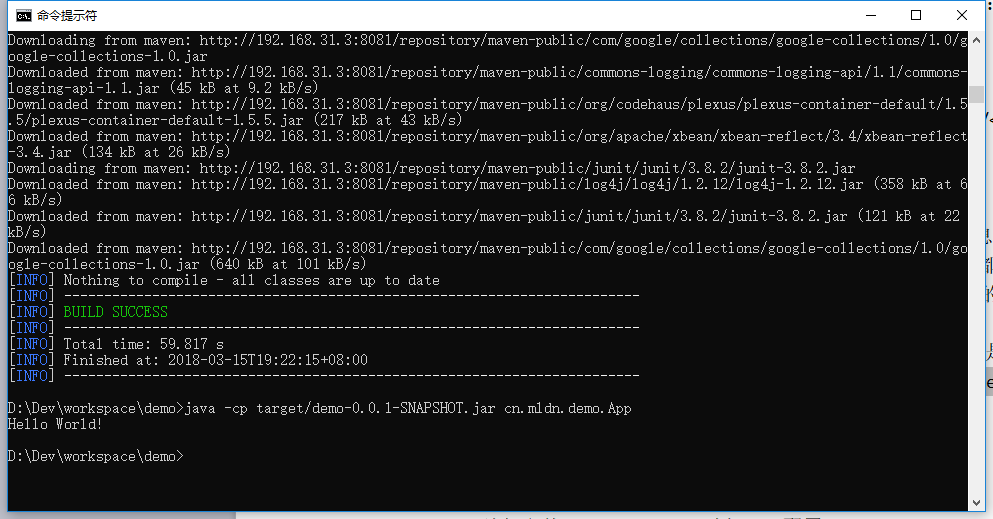


本地仓库



私服仓库





1. maven pom.xml解析
   1. 定义环境属性
      1. 定义全局属性

对于版本信息及目录信息使用全局定义，方便与维护。

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<junit.version>4.12</junit.version>

<spring.version>5.0.0.RELEASE</spring.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>2.9.4</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

* + 1. 插件配置

例如编译插件的配置

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId><!-- 定义插件组织信息 -->

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>${jdk.version}</source>

<target>${jdk.version}</target>

<encode>${project.build.sourceEncoding}</encode>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId><!-- 定义插件组织信息 -->

<artifactId>maven-source-plugin</artifactId>

<configuration>

<encode>${project.build.sourceEncoding}</encode>

</configuration>

<executions> <!-- 进行执行的配置 -->

<execution>

<id>sources</id>

<goals>

<goal>jar</goal>

</goals>

</execution>

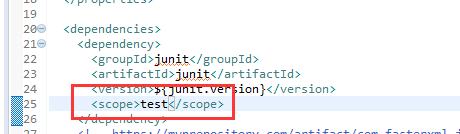
</executions>

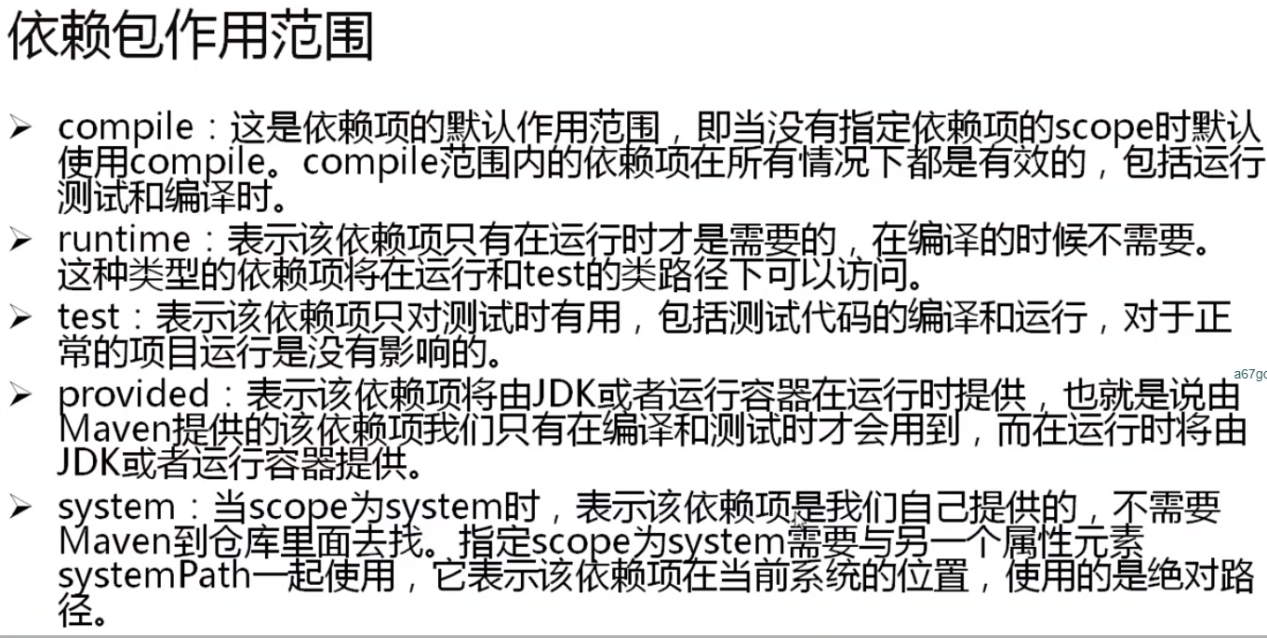
</plugin>

</plugins>

</build>

* + 1. 依赖包管理
       1. 包作用域：scope





* + - 1. 依赖包（传递）排除：

项目间传递

    如果我的当前项目是project1，project1要依赖project2，project1依赖project2的配置中加上 <optional>true</optional>，表示依赖可选，

<dependency>

<groupId>com.projecct</groupId>

<artifactId>project2</artifactId>

<version>1.0</version>

<scope>compile</scope>

<optional>true</optional></dependency>

 那么以后所有声明依赖project1的项目如果也依赖project2，就必须写手动声明。比如project3依赖project1和project2，如果project3只声明了对project1的依赖，那么project2不会自动加入依赖，需要重新声明对project2的依赖。

这种方式排除不了我项目中对第三方jar包所依赖的其他依赖，因为我不可能去修改第三方jar包的pom文件，所以只适合在项目组内部使用。

2. 依赖过滤

（1）单依赖过滤

       同依赖过滤直接处理：可以过滤一个或者多个，如果过滤多个要写多个<exclusion>。这个也解决不了我的问题，或者说解决太麻烦，我那里知道hbase要依赖那些包，记不住。

<dependency>

<groupId>org.apache.hbase</groupId>

<artifactId>hbase</artifactId>

<version>0.94.17</version>

**<exclusions>**

**<exclusion>**

**<groupId>commons-logging</groupId>**

**<artifactId>commons-logging</artifactId>**

**</exclusion>**

**</exclusions>**  </dependency>

（2）多依赖过滤

     把所以依赖都过滤了。手起刀落~啊，世界都安静了。

<dependency>

<groupId>org.apache.hbase</groupId>

<artifactId>hbase</artifactId>

<version>0.94.17</version>

**<exclusions>**

**<exclusion>**

**<groupId>\*</groupId>**

**<artifactId>\*</artifactId>**

**</exclusion>**

**</exclusions>**</dependency>

## **使用Eclipse-maven去除依赖问题的技巧**

强大的maven管理工具，去除重复依赖问题非常简单。

### **POM文件中去除依赖**

<dependencies>

<dependency>

<groupId> org.apache.cassandra</groupId>

<artifactId>cassandra-all</artifactId>

<version>0.8.1</version>

<exclusions>

<exclusion>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</exclusion>

<exclusion>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

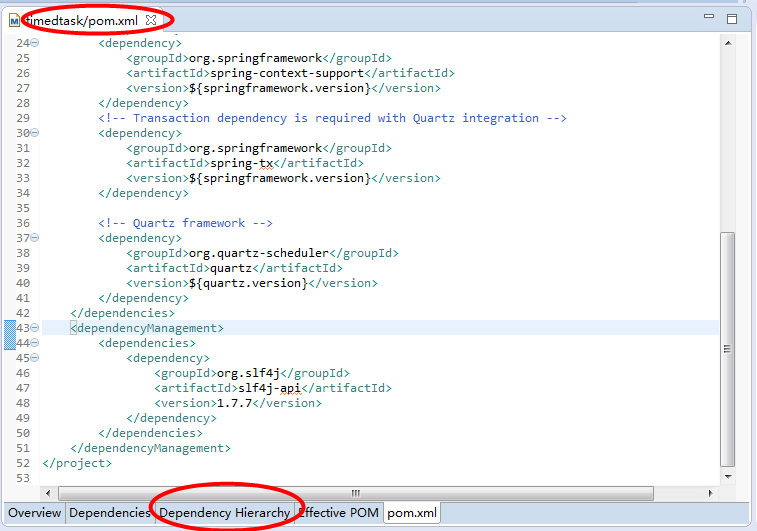
</exclusion>

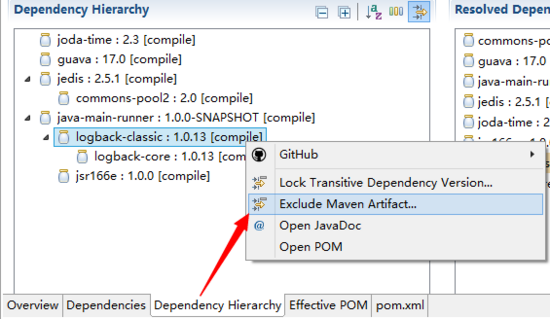
</exclusions>

</dependency></dependencies>

使用 exclusion去除依赖。

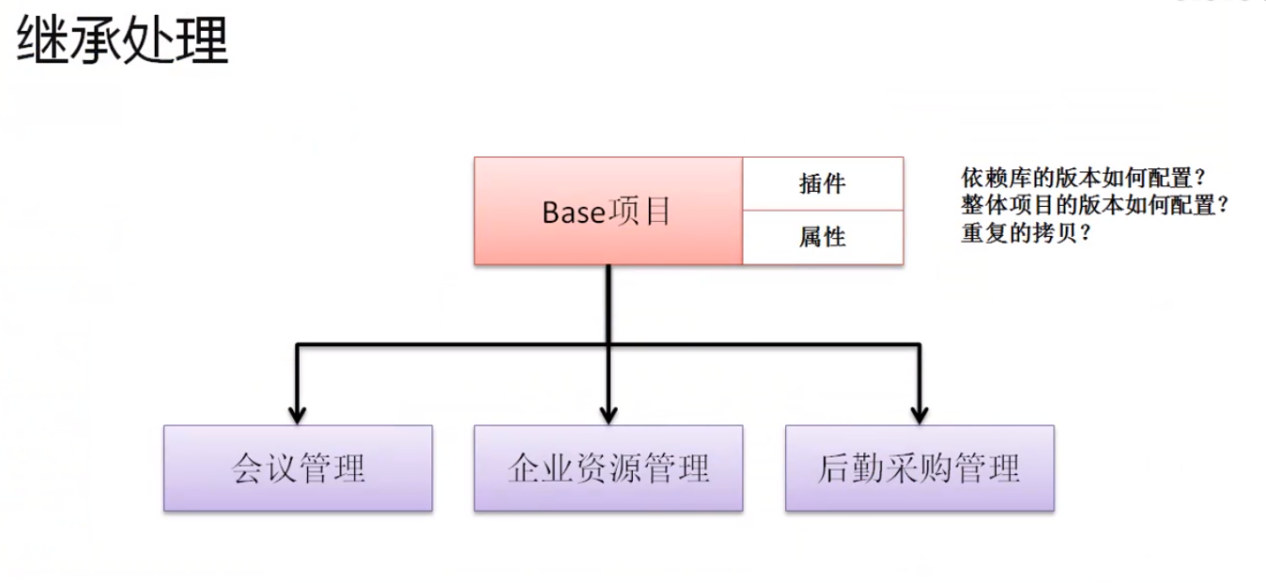
### **Eclipse图形界面去除依赖**





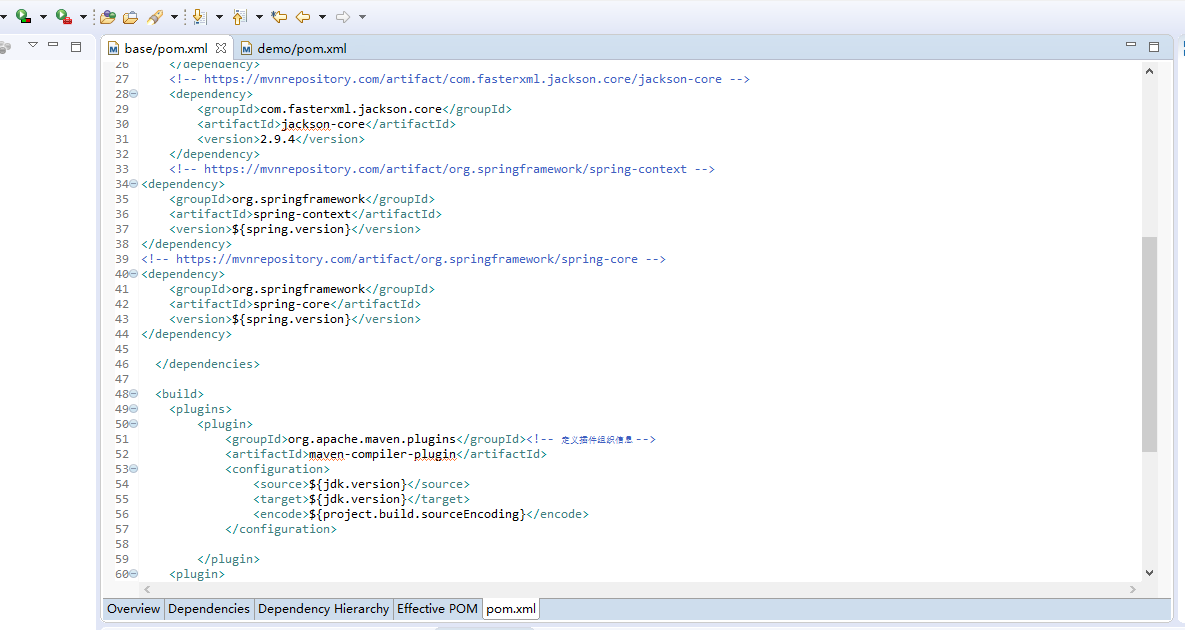
直接删除就OK了。

1. 继承关系



1. Base为最大的父项目，其类型应该为pom。将公用的配置放在pom.xml中，而后子项目引用父项目相关的定义。





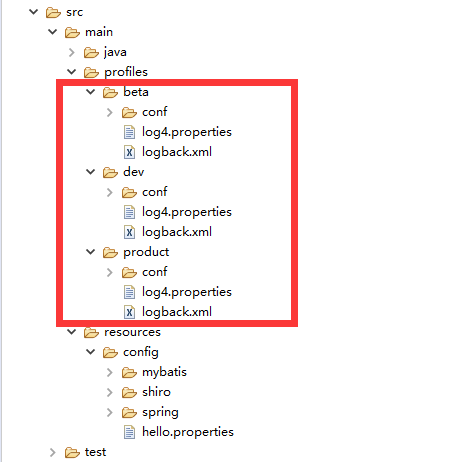
1. 子项目继承父项目的配置：



相关的属性及配置的插件都会通过父项目继承过来。

1. 构建设置

配置不同环境



<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>cn.mldn</groupId>

<artifactId>base</artifactId>

<version>0.0.1-SNAPSHOT</version>

<relativePath>../base/pom.xml</relativePath>

</parent>

<artifactId>oa</artifactId>

<packaging>war</packaging>

<name>oa Maven Webapp</name>

<url>http://maven.apache.org</url>

**<properties> <!-- 将所有资源文件的配置路径作为属性出现 -->**

**<profiles.dir>src/main/profiles</profiles.dir>**

**<resources.dir>src/main/resources</resources.dir>**

**</properties>**

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<!-- <version>${junit.version}</version> -->

<scope>test</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<!-- <version>${servlet.version}</version> -->

<scope>provided</scope>

</dependency>

</dependencies>

**<profiles> <!-- 明确的描述出要使用的不同环境的配置文件 -->**

**<profile>**

**<id>dev</id> <!-- 明确的描述出本配置文件的编号，一般会与文件目录重名 -->**

**<properties> <!-- 明确的描述出此配置项对应的路径 -->**

**<profile.dir>${profiles.dir}/dev</profile.dir>**

**</properties>**

**<activation> <!-- 如果没有指派profile的编号，默认使用此配置 -->**

**<activeByDefault>true</activeByDefault>**

**</activation>**

**</profile>**

**<profile>**

**<id>beta</id> <!-- 明确的描述出本配置文件的编号，一般会与文件目录重名 -->**

**<properties> <!-- 明确的描述出此配置项对应的路径 -->**

**<profile.dir>${profiles.dir}/beta</profile.dir>**

**</properties>**

**<activation>**

**<activeByDefault>false</activeByDefault>**

**</activation>**

**</profile>**

**<profile>**

**<id>product</id> <!-- 明确的描述出本配置文件的编号，一般会与文件目录重名 -->**

**<properties> <!-- 明确的描述出此配置项对应的路径 -->**

**<profile.dir>${profiles.dir}/product</profile.dir>**

**</properties>**

**<activation>**

**<activeByDefault>false</activeByDefault>**

**</activation>**

**</profile>**

**</profiles>**

**<build>**

**<finalName>oa</finalName>**

**<defaultGoal>package</defaultGoal> <!-- 在打包的时候进行的使用 -->**

**<resources> <!-- 可以理解为资源文件的是否输出的配置 -->**

**<resource>**

**<directory>${resources.dir}</directory>**

**<filtering>false</filtering>**

**<includes> <!-- 明确的描述出需要包含的文件名称 -->**

**<include>\*\*/spring-common.xml</include>**

**<include>\*\*/spring-shiro.xml</include>**

**</includes>**

**<excludes> <!-- 在进行资源选择的时候不包含任何目录下的\*.properties文件 -->**

**<exclude>\*\*/\*.properties</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>${profile.dir}</directory> <!-- 定义profile的输出路径 -->**

**<filtering>false</filtering>**

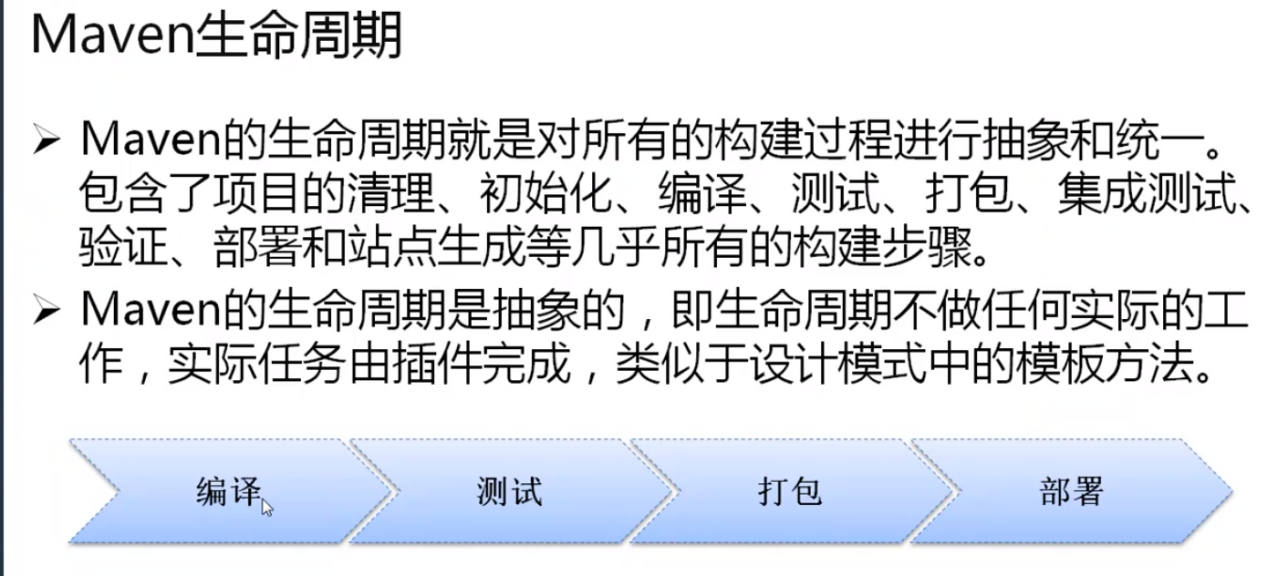
**</resource>**

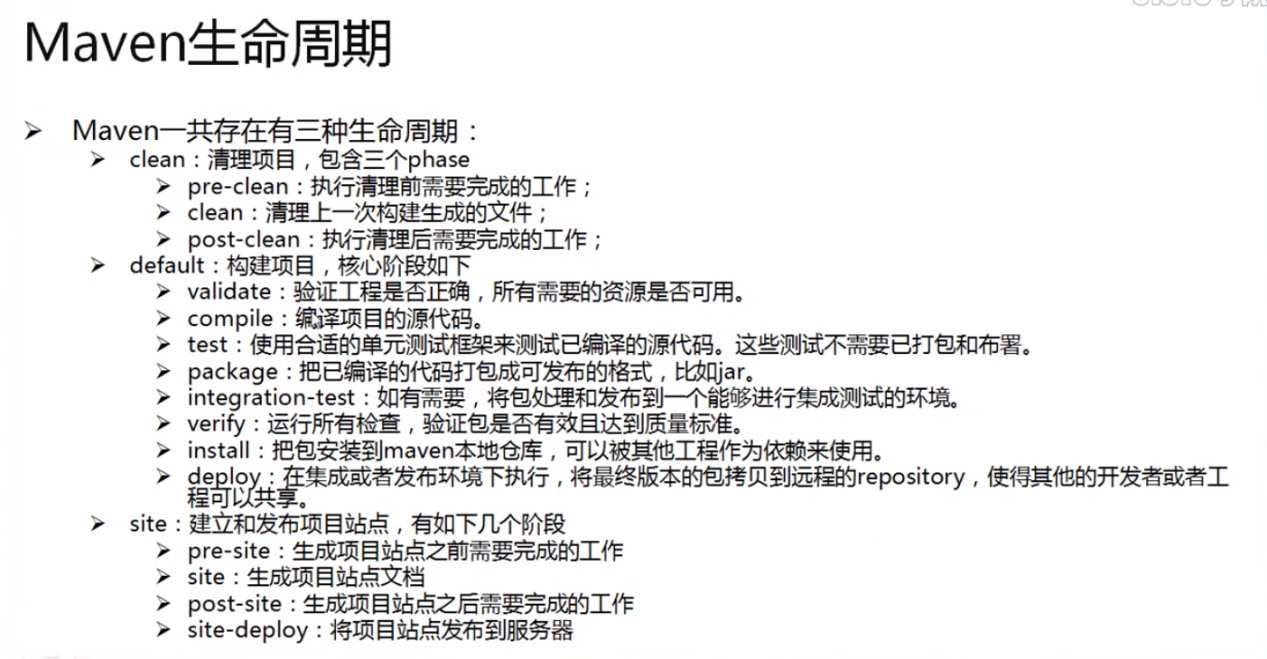
**</resources>**

</build>

</project>

1. Maven生命周期





补充：

对于官网没有的jar包，maven向本地仓库导入jar包用如下命令

Java代码

mvn install:install-file

-DgroupId=包名

-DartifactId=项目名

-Dversion=版本号

-Dpackaging=jar

-Dfile=jar文件所在路径

以pinyin4j-2.5.0为例：

1、将pinyin4j-2.5.0.jar文件放在“D:\JAR\_LIB”目录下（该目录任意）

2、执行命令：

mvn install:install-file -Dfile=D:\JAR\_LIB\pinyin4j-2.5.0.jar -DgroupId=net.sourceforge.pinyin4j -DartifactId=pinyin4j -Dversion=2.5.0 -Dpackaging=jar -DgeneratePom=true -DcreateChecksum=true

这样maven向本地仓库导入 pinyin4j-2.5.0.jar 包就成功了。

3.pom.xml文件配置：

Xml代码

<dependency>

<groupId>net.sourceforge.pinyin4j</groupId>

<artifactId>pinyin4j</artifactId>

<version>2.5.0</version>

</dependency>