

编号: 001

# Maven使用介绍

2019.6

Automatic  
Driving  
System

- **第一章 Maven关键知识点**
- 第二章 Maven的安装与配置**
- 第三章 用Maven构建项目**
- 第四章 搭建Nexus私服**

# 概念介绍

## 1.概念

- 是一系列构建工具的集合

## 2.约定大于配置

- Maven项目约定目录结构如右图

## 3.pom.xml文件

- 项目对象模型 (Project Object Model,POM)

## 4.Maven坐标

- groupId:artifactId:version,唯一确定一个项目。例如

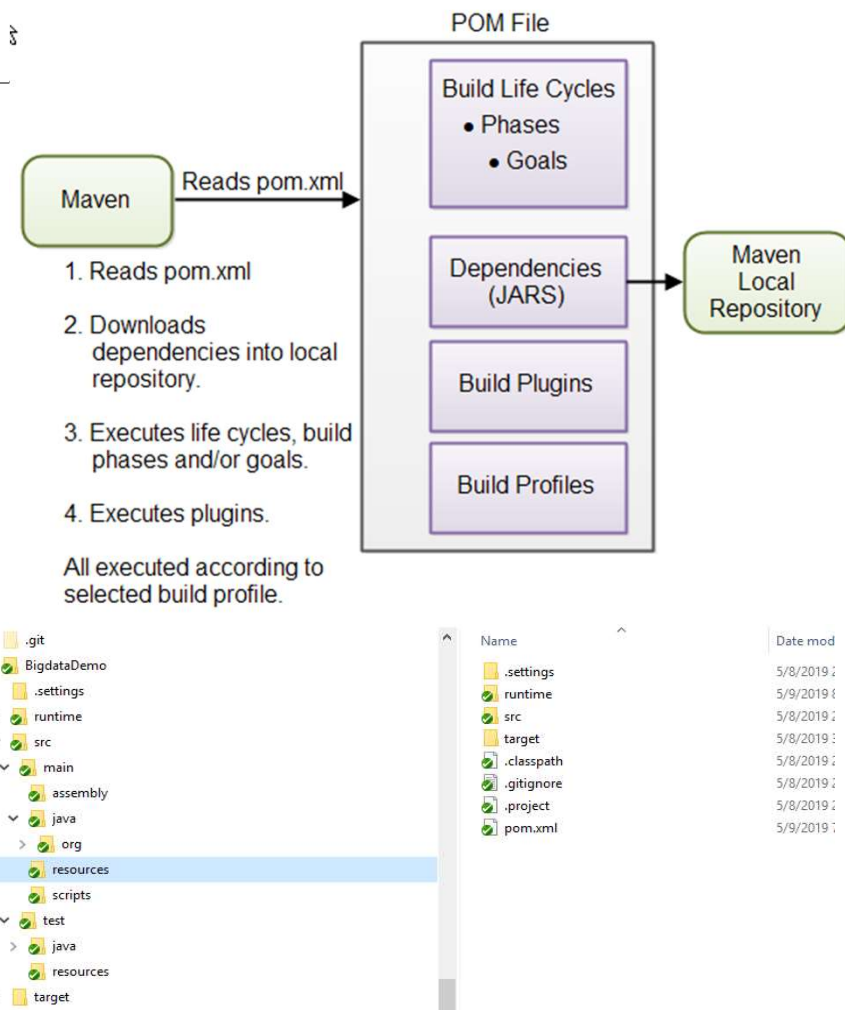
<dependency>

<groupId>com.jcraft</groupId>

<artifactId>jsch</artifactId>

<version>0.1.55</version>

</dependency>



- 本地仓库
  - 存放从中央仓库下载的依赖组件
  - Windows 默认地址:c:\user\%username%\m2\repository
  - linux默认地址:~/m2/repository
- 中央仓库
  - central repository:Apache官一样维护的组件库
  - 组件可升级

## 生命周期与对应插件

生命周期和插件是Maven的两个核心概念，命令行的输入往往就对应了生命周期，如mvn package就表示执行默认生命周期阶段package。Maven的生命周期是抽象的，其实际行为都由插件来完成。Maven的生命周期是为了对所有的构建过程进行抽象和统一。

Maven拥有三套相互独立的生命周期，分别为clean、default和site。clean生命周期的目的是清理项目，default生命周期的目的是构建项目，site目的是建立项目站点。

### clean:

1. pre-clean 执行一些清理前需要完成的工作。
2. clean 清理上一次构建生成的文件。
3. post-clean 执行一些清理后需要完成的工作。

### site

pre-site

site 生成项目站点文档。

post-site

site-deploy 将生成的项目站点发布到服务器上。

**default** 包含的操作很多，以下列出主要的：

process-sources 处理项目主资源文件。一般来说，是对src/main/resources目录的内容进行变量替换等工作后，复制到项目输出主classpath目录中。

compile 编译项目的主代码。一般来说，是编译src/main/java目录下文件至项目输出的主classpath目录中。

process-test-sources 处理项目测试资源文件 src/main/resources。

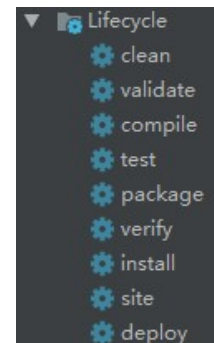
test-compile 编译项目的测试代码。src/test/java。

test 使用单元测试框架运行测试，测试代码不会被打包或部署。

package 接受编译好的代码，打包成可发布的格式，如jar。

install 将包安装到Maven本地仓库，供本地其他Maven项目使用。

deploy 将最终的包复制到远程仓库。



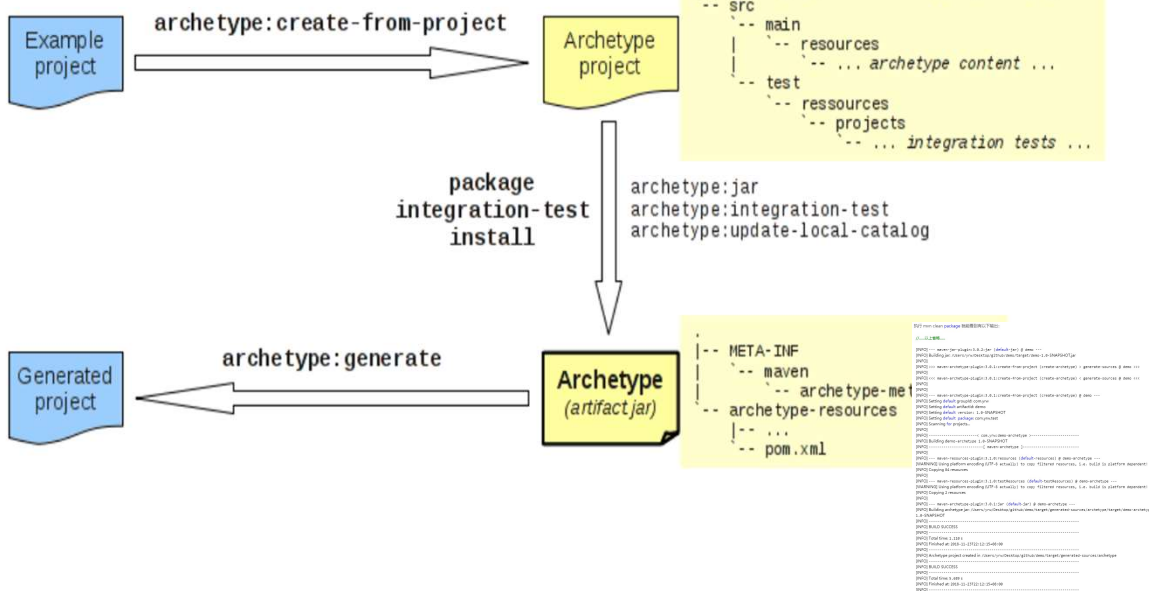
# 生命周期与对应插件

## 自定义绑定:

除了内置绑定以外，用户还能够自己选择将某个插件目标绑定到生命周期的某个阶段上。例如一个常用的插件 Maven Archetype Plugin，用户通过这个插件可以生成一个Maven项目的骨架，也可以从一个现成的项目中生成模板。

通常的用法是使用 `mvn archetype:create-from-project` 指令生成模板。如果我正在编写一个模板项目，我希望在打包的时候自动生成模板，那么我可以把这个目标绑定到default的package生命周期上，如下配置：

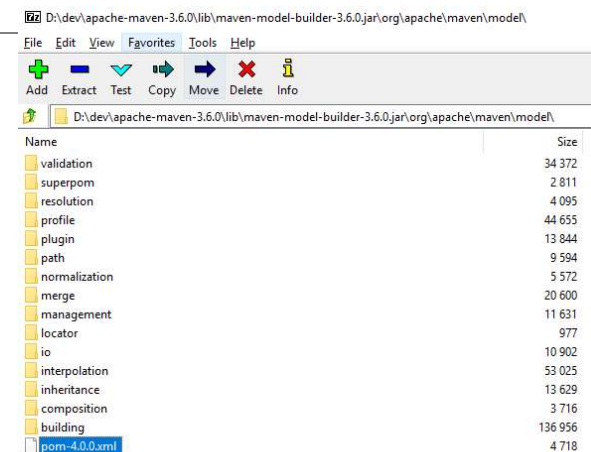
```
</plugins>
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-archetype-plugin</artifactId>
  <version>3.0.1</version>
  <executions>
    <execution>
      <id>create-archetype</id>
      <phase>package</phase>
      <goals>
        <goal>create-from-project</goal>
      </goals>
    </execution>
  </executions>
</plugin>
</plugins>
```



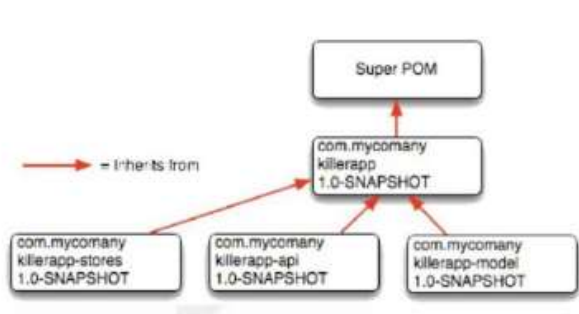


## 超级pom.xml文件

- 位于maven-model-builder-xxx.jar中  
示例apache-maven-3.6.0\lib\maven-model-builder-3.6.0.jar包中的  
\org\apache\maven\model\pom-4.0.0.xml



Name	Size
validation	34 372
superpom	2 811
resolution	4 095
profile	44 655
plugin	13 844
path	9 594
normalization	5 572
merge	20 600
management	11 631
locator	977
io	10 902
interpolation	53 025
inheritance	13 629
composition	3 716
building	136 956
pom-4.0.0.xml	4 718



- 新建的Maven项目pom.xml文件默认继承它

## Maven项目的版本号：

一个Maven项目发布版本号用version编码，用来分组和排序发布。Maven中的版本包含了以下部分：主版本，次版本，增量版本，和限定版本号。一个版本中这些部分对应如下格式。

`<major version>.<minor version>.<incremental version> - <qualifier>`

例如：版本1.3.5由一个主版本1，一个次版本3，和一个增量版本5.而一个版本5只有主版本，没有次版本和增量版本。限定版本用标识里程碑构建：alpha和beta发布，限定版本通过连字符与主版本，次版本或增量版本隔离。

例如，版本“1.3-beat-01”有一个主版本，一个次版本，和一个限定版本。



- **依赖范围：**用 `【】` , `()` , `【)` , `(】` 区间表示
- 可选依赖：避免某些组件的依赖传递
- 传递依赖：

假如有Maven项目A，项目B依赖A，项目C依赖B。那么我们可以说 C依赖A。也就是说，依赖的关系为：C—>B—>A。那么我们执行项目C时，会自动把B、A都下载导入到C项目的jar包文件夹中。这就是依赖的传递性。

注意：正常的版本号(如3.8.2),表示 “3.8.2” 最佳，其它版本也可以。而 **【3.8.2】** 表示必须3.8.2版本，其它版本不行

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>3.8.1</version>
  <scope>test</scope>
</dependency>
```

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>[3.8,4.0)</version>
  <scope>test</scope>
</dependency>
```

# 传递范围对依赖关系的影响

## 基本概念

A依赖B, 需要在A的pom.xml文件中添加B的坐标, 添加坐标时需要指定依赖范围, 依赖范围包括:

1. compile: 编译范围, 指A在编译时依赖B, 此范围为默认依赖范围。编译范围的依赖会用在编译、测试、运行, 由于运行时需要所以编译范围的依赖会被打包。
2. provided: provided依赖只有在当JDK或者一个容器已提供该依赖之后才使用, provided依赖在编译和测试时需要, 在运行时不需要, 比如: servlet api被tomcat容器提供。
3. runtime: runtime依赖在运行和测试系统的时候需要, 但在编译的时候不需要。比如: jdbc的驱动包。由于运行时需要所以runtime范围的依赖会被打包。
4. test: test范围依赖在编译和运行时都不需要, 它们只有在测试编译和测试运行阶段可用, 比如: junit。由于运行时不需要所以test范围依赖不会被打包。
5. system: system范围依赖与provided类似, 但是你必须显式的提供一个对于本地系统中JAR文件的路径, 需要指定systemPath磁盘路径, system依赖不推荐使用。

直接依赖 \ 传递依赖	compile	provided	runtime	test
compile	compile	-	runtime	-
provided	provided	provided	provided	-
runtime	runtime	-	runtime	-
test	test	-	test	-

例如有A、B、C, A依赖B、B依赖C, C可能是A的传递依赖, 如上图: 最左边一列为直接依赖, 理解为A依赖B的范围, 最顶层一行为传递依赖, 理解为B依赖C的范围, 行与列的交叉即为A传递依赖C的范围。

依赖范围	对于编译classpath有效	对于测试classpath有效	对于运行时classpath有效	例子
compile	Y	Y	Y	spring-core
test	-	Y	-	Junit
provided	Y	Y	-	servlet-api
runtime	-	Y	Y	JDBC驱动
system	Y	Y	Y	本地的, Maven仓库之外的类库

依赖范围由强到弱的顺序是:  
compile > provided > runtime > test

示例: 比如A对B有compile依赖, B对C有runtime依赖, 根据表格所示A对C有runtime依赖

直接依赖 \ 传递依赖	compile	provided	runtime	test
compile	compile	-	runtime	-
provided	provided	provided	provided	-
runtime	runtime	-	runtime	-
test	test	-	test	-

## 用Maven构建项目可以带来的好处

- 可自动解决组件依赖问题
  - 类似Centos的yum和Ubuntu的apt-get工具
- 其它原因
  - 本地/中央仓库实现多个项目的jar包集中管理
  - 编译、测试、打包、部署、分发自动化工具
  - 软件轻量级发布

## 第一章 Maven关键知识点

## ● 第二章 Maven的安装与配置

## 第三章 用Maven构建项目

## 第四章 搭建Nexus私服

### 前置条件:

- 安装并配置好java环境, 推荐oracle jdk 1.8+
- 能够访问互联网, 至少要能访问国内的maven 源

### 安装:

- 下载apache-maven-3.6.1-bin.tar.gz

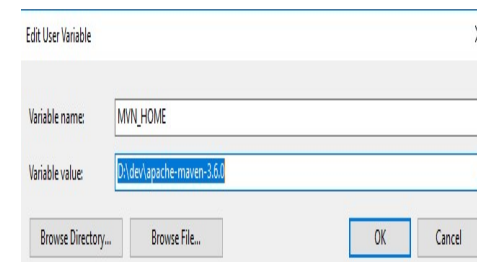
<http://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.6.1/binaries/apache-maven-3.6.1-bin.tar.gz>

- 解压, 例如: d:/dev/
- 配置path

windows 添加MVN\_HOME,指向安装目录, 并把%MVN\_HOME%/bin 添加到path 变量中, 变量间用半角“;” 分隔

linux中 在用户的profile文件中加入export MVN\_HOME=.... (安装目录) 并把\$ MVN\_HOME/bin添加到PATH变量中, 变量间用半角“:” 分隔

- 测试
- 启动命令行窗口, 录入mvn -version,如果显示版本信息表示成功。



```
Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5fffb20918da4f719f3, 2018-10-25T02:41:47+08:00)
Maven home: D:\dev\apache-maven-3.6.0\bin\
Java version: 1.8.0_201, vendor: Oracle Corporation, runtime: D:\Java\jdk1.8.0_201\jre
Default locale: en_US, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

## 配置:

- 修改Maven主配置文件，即%MVN\_HOME%下的config/setting.xml

### 指定本地仓库localRepository

```
41 | The sections in this sample file are intended to give you a running start at
42 | getting the most out of your Maven installation. Where appropriate, the default
43 | values (values used when the setting is not specified) are provided.
44 |
45 | -->
46 | <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
47 |   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
48 |   xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
49 |   <!-- localRepository
50 |    | The path to the local repository maven will use to store artifacts.
51 |    |
52 |    | Default: ${user.home}/.m2/repository
53 |    |<localRepository>path/to/local/repo</localRepository>
54 |    -->
55 |    <localRepository>d:/repo</localRepository>
56 |
57 |   <!-- interactiveMode
58 |    | This will determine whether maven prompts you when it needs input. If set to false,
59 |    | maven will use a sensible default value, perhaps based on some other setting, for
60 |    | the parameter in question.
61 |    |
62 |    | Default: true
63 |    |<interactiveMode>true</interactiveMode>
64 |
```

### 配置http代理 (如需代理上网)

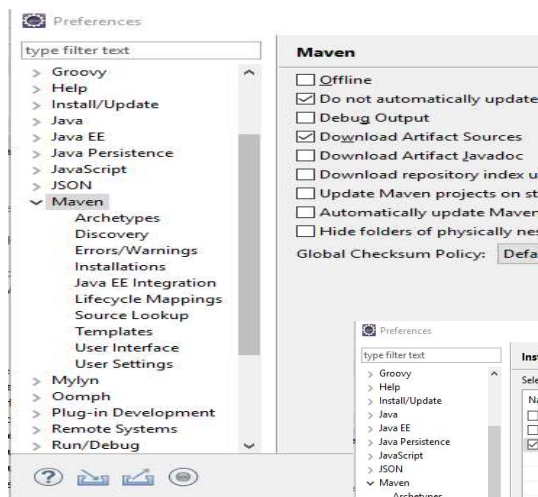
```
90 | |-->
91 | <proxies>
92 |   <!-- proxy
93 |    | Specification for one proxy, to be used in connecting to the network.
94 |    |
95 |    |<proxy>
96 |    |   <id>optional</id>
97 |    |   <active>true</active>
98 |    |   <protocol>http</protocol>
99 |    |   <username>proxyuser</username>
100 |    |   <password>proxypass</password>
101 |    |   <host>proxy.host.net</host>
102 |    |   <port>80</port>
103 |    |   <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
104 |    </proxy>
105 |   -->
106 | </proxies>
107 |
```

### 配置国内镜像

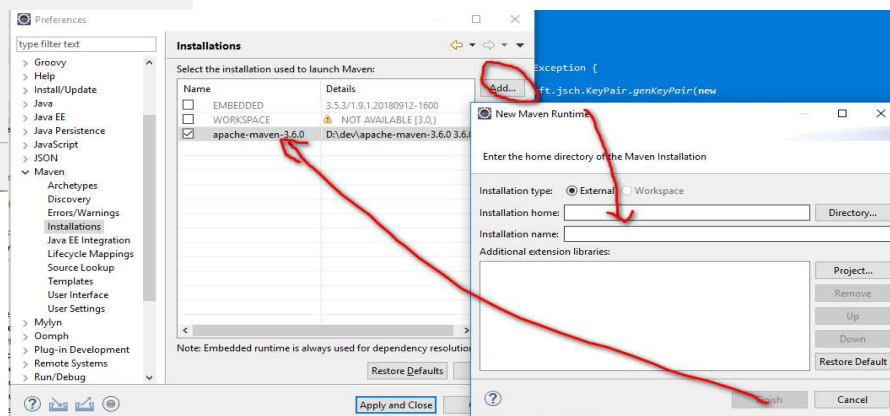
```
67 | <mirrors>
68 |   <!-- mirror
69 |    | Specifies a repository mirror site to use instead of a given repository. The repository
70 |    | this mirror serves has an ID that matches the mirrorOf element of this mirror. IDs are
71 |    | for inheritance and direct lookup purposes, and must be unique across the set of mirrors.
72 |    |
73 |    |<mirror>
74 |    |   <id>mirrorId</id>
75 |    |   <mirrorOf>repositoryId</mirrorOf>
76 |    |   <name>Human Readable Name for this Mirror.</name>
77 |    |   <url>http://my.repository.com/repo/path</url>
78 |    </mirror>
79 |   -->
80 |
81 |   <!-- 阿里云仓库 -->
82 |   <mirror>
83 |     <id>alimaven</id>
84 |     <mirrorOf>central</mirrorOf>
85 |     <name>aliyun_maven</name>
86 |     <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>
87 |   </mirror>
88 |
```

# Eclipse配置maven信息

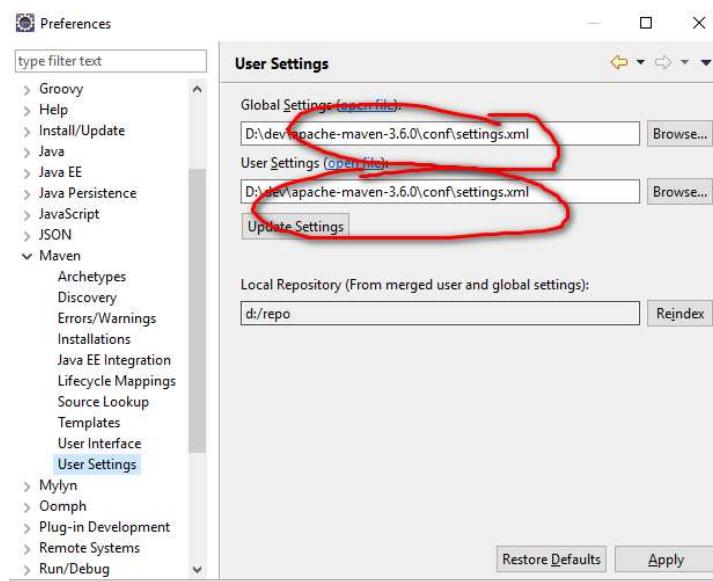
## 1.Menu: windows->Prefereces->Maven :



## 2. Installations 添加maven工具



## 3.修改确认setting配置





**第一章 Maven关键知识点**

**第二章 Maven的安装与配置**

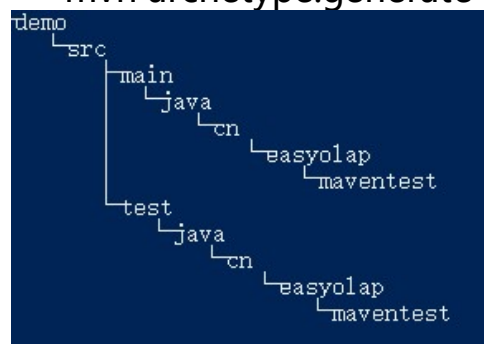
● **第三章 用Maven构建项目**

**第四章 搭建Nexus私服**

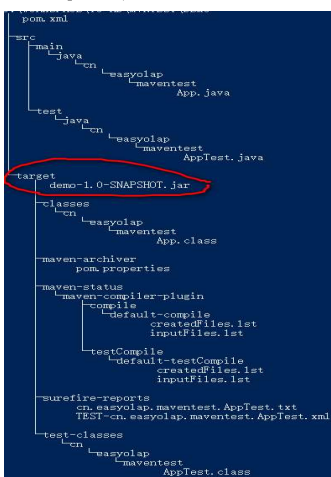
# 构建Java SE项目

## 1.创建Maven工程

`mvn archetype:generate -DgroupId=cn.easyolap.maventest -DartifactId=demo -DpackageName=cn.easyolap.maven`



## 2.构建打包 mvn install



## 3.查看有效的pom文件:

`mvn help:effective-pom` Maven会将有效pom输出到命令行。

## 4.发布站点

Mvn site 生成项目相关信息的网站

报错: `mvn site`:

`java.lang.ClassNotFoundException: org.apache.maven.doxia.siterenderer.DocumentContent`

需要把>maven-site-plugin升级到最新

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-site-plugin</artifactId>
      <version>3.7.1</version>
    </plugin>
  </plugins>
</build>
```

## 5.运行

`java -cp target/demo-1.0-SNAPSHOT.jar cn.easyolap.maventest.App`

```
D:\workspace\tg-hd\mvntest\demo>java -cp target/demo-1.0-SNAPSHOT.jar cn.easyolap.maventest.App
Hello World!
```

## 构建Java SE项目

### 6.查看已解决的插件依赖

mvn dependency:resolve

```
Scanning for projects...
Building demo 1.0-SNAPSHOT
[ jar ]
--- mvn dependency:resolve:3.8.1:resolve (default-cli) @ demo ---
The following files have been resolved:
junit:junit:jar:3.8.1:test
BUILD SUCCESS
Total time: 1.200 s
Finished at: 2019-06-25T20:41:39+08:00
```

### 7.查看依赖树

mvn dependency:tree

```
cn.easyolap.maven:test:demo:jar:1.0-SNAPSHOT
\-- junit:junit:jar:3.8.1:test
BUILD SUCCESS
Total time: 16.366 s
Finished at: 2019-06-25T20:35:35+08:00
```

### 8.忽略测试失败

mvn test -Dmaven.test.failure.ignore=true

### 9.忽略测试

方法一：mvn install -DskipTests 或 mvn install -Dmaven.test.skip=true

方法二：使用maven插件的方法

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-
plugin</artifactId>
  <version>2.18.1</version>
  <configuration>
    <skipTests>true</skipTests>
  </configuration>
</plugin>
```

方法三：使用定义变量的方法

```
<properties>
  <skipTests>true</skipTests> </properties>
或者
<properties> <maven.test.skip>true</maven.test.skip>
</properties>
```

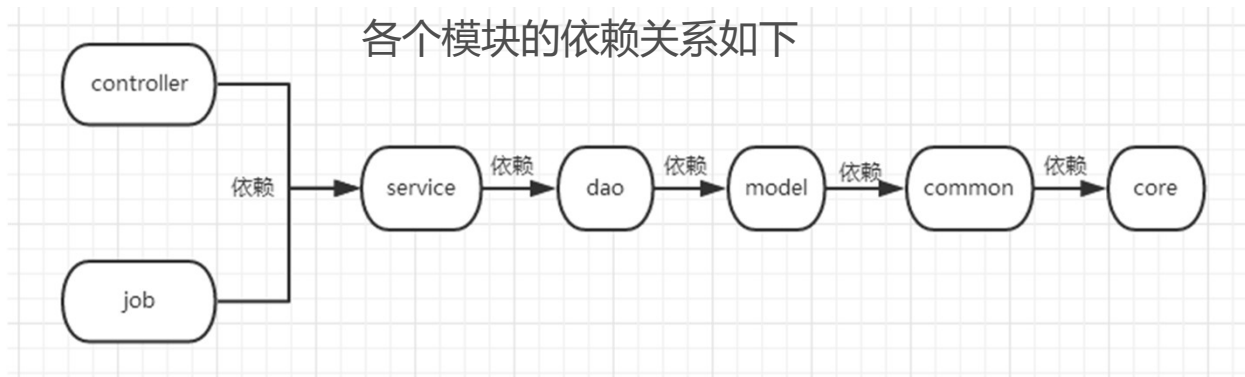
## 构建spring boot项目

练习1:

1.构建单工程的spring-boot项目

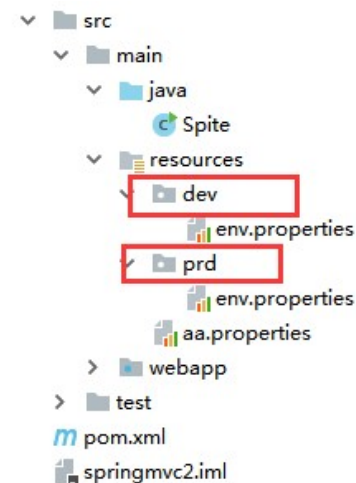
练习2:

1.构建多模块的spring-boot项目



## maven profile配置

```
<profiles>
  <profile>
    <id>dev</id>
    <properties>
      <env>dev</env>
    </properties>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
  </profile>
  <profile>
    <id>prd</id>
    <properties>
      <env>prd</env>
    </properties>
  </profile>
</profiles>
```



```
<build>
  <finalName>springmvc2</finalName>
  <resources>
    <resource>
      <directory>src/main/resources/${env}</directory>
    </resource>
  </resources>
</build>
```

**第一章 Maven关键知识点**

**第二章 Maven的安装与配置**

**第三章 用Maven构建项目**

● **第四章 搭建Nexus私服**

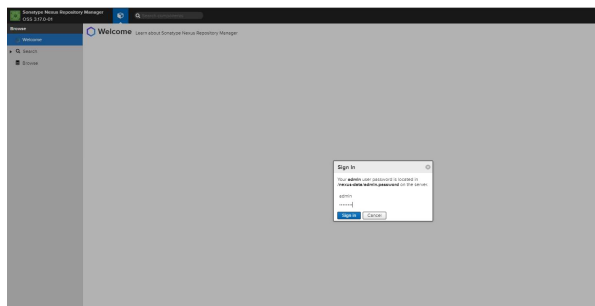
ATLASSIAN\_HOME=/data/devops/nexus/

```
docker run -d \  
--name nexus \  
--hostname nexus \  
--user root:root \  
--restart always \  
--link=mysql-server:mysql-server \  
--link=openldap:openldap \  
-v $ATLASSIAN_HOME/work:/nexus-data \  
-p 58081:8081 \  
-p 58082:8082 \  
-p 5100-5110:5100-5110 \  
-e NEXUS_CONTEXT=nexus \  
sonatype/nexus3
```

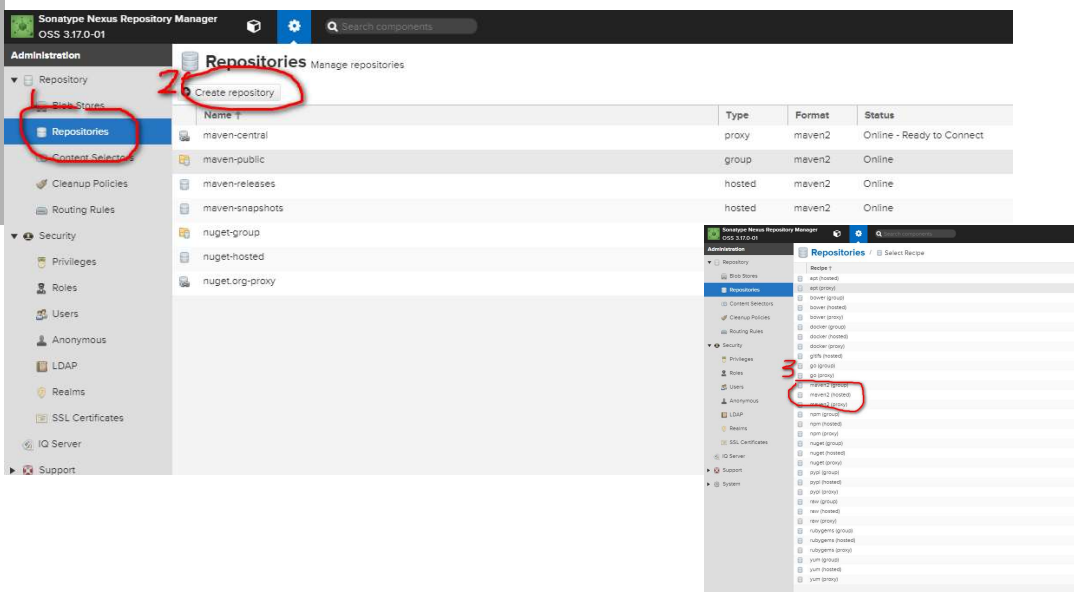


# Nexus配置

## 登录



## 创建私有仓库



# Nexus配置

## 创建私有仓库

**Repositories** / Select Recipe / Create Repository: maven2 (hosted)

**Name:**  A unique identifier for this repository

**Online:** ☒ If checked, the repository accepts incoming requests

**Maven 2**

**Version policy:**  
What type of artifacts does this repository store?  
Release

**Layout policy:**  
Validate that all paths are maven artifact or metadata paths  
Strict

**Storage**

**Blob store:**  
Blob store used to store asset contents  
☒ default

**Strict Content Type Validation:**  
☒ Validate that all content uploaded to this repository is of a MIME type appropriate for the repository format

**Hosted**

**Deployment policy:**  
Controls if deployments of and updates to artifacts are allowed  
Disable redeploy

**Cleanup Policy**

**Available cleanup policies:**  
Select a cleanup policy  
None

**Create repository** Cancel

## 把代理和私有仓库加入到分组中

**Sonatype Nexus Repository Manager**  
OSS 3.17.0-01

**Administration**

- Repository
- Blob Stores
- Repositories**
- Content Selectors
- Cleanup Policies
- Routing Rules
- Security
  - Privileges
  - Roles
  - Users
  - Anonymous
  - LDAP
  - Realms
  - SSL Certificates
- Support
  - System

**Repositories** / maven-public

Delete repository Invalidate cache

**Settings**

**Name:** maven-public  
**Format:** maven2  
**Type:** group  
**URL:** http://39.96.53.81:58081/nexus/repository/maven-public/  
**Online:** ☒ If checked, the repository accepts incoming requests

**Storage**

**Blob store:** default  
**Strict Content Type Validation:** ☒ Validate that all content uploaded to this repository is of a MIME type appropriate for the repository format

**Group**

**Member repositories:**  
Select and order the repositories that are part of this group

Available

Filter

test

Members

- maven-releases
- maven-snapshots
- maven-central
- aliyun-maven
- repotest

Save Discard

## 仓库分类

每个仓库的格式为maven2或者maven1。

此外，仓库还有一个属性为Policy（策略），表示该仓库为发布（Release）版本仓库还是快照（Snapshot）版本仓库。最后两列的值为仓库的状态和路径。

- Maven Central：该仓库代理Maven中央仓库，其策略为Release，因此只会下载和缓存中央仓库中的发布版本构件。
- Releases：这是一个策略为Release的宿主类型仓库，用来部署组织内部的发布版本构件。
- Snapshots：这是一个策略为Snapshot的宿主类型仓库，用来部署组织内部的快照版本构件。
- 3rd party：这是一个策略为Release的宿主类型仓库，用来部署无法从公共仓库获得的第三方发布版本构件。
- Apache Snapshots：这是一个策略为Snapshot的代理仓库，用来代理Apache Maven仓库的快照版本构件。
- Codehaus Snapshots：这是一个策略为Snapshot的代理仓库，用来代理Codehaus Maven仓库的快照版本构件。
- Google Code：这是一个策略为Release的代理仓库，用来代理Google Code Maven仓库的发布版本构件。
- java.net-Maven 2：这是一个策略为Release的代理仓库，用来代理java.net Maven仓库的发布版本构件。
- Public Repositories：该仓库组将上述所有策略为Release的仓库聚合并通过一致的地址提供服务。
- Public Snapshot Repositories：该仓库组将上述所有策略为Snapshot的仓库聚合并通过一致的地址提供服务。

## 使用Nexus私服

在项目中使用私服:

- 1.修改Maven安装目录settings.xml, 添加mirror 图1
- 2.在项目的pom.xml中添加, 图2

图1

```
<mirror>
  <id>self</id>
  <mirrorOf>central</mirrorOf>
  <name>my self maven images.</name>
  <url>[redacted]repository/maven-public/</url>
</mirror>
```

图2

```
<!-- nexus私服远程仓库配置 -->
<repositories>
  <repository>
    <id>tghd-nexus2</id>
    <name>Nexus Repository</name>
    <url>[redacted]y/maven-public/</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
</repositories>
```

项目成果特提交到私有仓库中, 负责管理构件的发布包和其他编译生成的支撑文件。

修改Maven项目的pom.xml文件

```
<distributionManagement>
  <repository>
    <id>tec-nexus-releases</id>
    <name>Nexus_Release_Repository</name>
    <url>[redacted]sitory/tec-releases/</url>
  </repository>
  <snapshotRepository>
    <id>tec-nexus-snapshots</id>
    <name>Nexus_Snapshot_Repository</name>
    <url>[redacted]sitory/tec-snapshots/</url>
  </snapshotRepository>
</distributionManagement>
```

修改Maven安装目录settings.xml

```

  <server>
    <id>tec-nexus-releases</id>
    <username>admin</username>
    <password>admin123</password>
  </server>
  <server>
    <id>tec-nexus-snapshots</id>
    <username>admin</username>
    <password>admin123</password>
  </server>
</servers>
```

## Maven将本地jar上传到本地maven库或私有仓库

有时一些jar通过在pom文件中追加dependency的方式无法更新到jar，但在mavenrepository中央库确可以下载到该jar。这时可以采用本方法把jar上传到本地maven库或上传到私有maven库。

Jar的maven配置

```
<dependency>
<groupId>org.apache.thrift</groupId>
<artifactId>libthrift</artifactId>
<version>0.9.2</version>
</dependency>
```

### 1、上传到本地Maven库

命令：

```
mvn install:install-file -Dfile=D:\thrift-0.9.2.jar
-DgroupId=org.apache.thrift -DartifactId=libthrift -Dversion=0.9.2
-Dpackaging=jar
```

其中：

- DgroupId和-DartifactId构成了该jar包在pom.xml的坐标，对应依赖的groupId和artifactId
- Dfile表示需要上传的jar包的绝对路径
- Dpackaging 为安装文件的种类

### 2、上传到私有仓库

```
mvn deploy:deploy-file -DgroupId=org.apache.thrift -DartifactId=libthrift -Dversion=1.12 -Dpackaging=jar
-Dfile=D:\thrift-0.9.2.jar -Durl=http://ip:port/nexus/content/repositories/thirdparty/ -
DrepositoryId=thirdparty
-- DgroupId和DartifactId构成了该jar包在pom.xml的坐标，对应依赖的groupId和artifactId
-- Dfile表示需要上传的jar包的绝对路径
-- Durl私服上仓库的url精确地址(打开nexus左侧repositories菜单，可以看到该路径)
-- DrepositoryId服务器的表示id，在nexus的configuration可以看到
```

# 谢谢



surpass\_li@aliyun.com



<https://www.easyolap.cn/>

作者：李在超

擅长：Java程序员, 了解Devops, 捣鼓 linux和Go

