目　录

致谢

当前文档 《Maven学习笔记》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-04-22。

Maven学习笔记

Maven学习笔记

Maven学习笔记，请点击下面的链接阅读或者下载电子版本：

- 在线阅读
  - 国内服务器：腾讯云加速，国内网速极快(推荐)
  - 国外服务器：gitbook提供的托管，服务器在国外，速度比较慢，经常被墙
- 下载pdf格式
- 下载mobi格式
- 下载epub格式

来源(书栈小编注)

> *https://github.com/skyao/learning-maven*

介绍

Maven介绍

==========

TBD

资料收集整理

资料收集整理

TBD...

安装

安装Maven

linux 安装

ubuntu apt-get安装

参考地址，执行命令依次如下：

```
1. sudo apt-get remove maven2
2. sudo add-apt-repository "deb
   http://ppa.launchpad.net/natecarlson/maven3/ubuntu precise main"
3. sudo apt-get update
4. sudo apt-get install maven3
5. sudo ln -s /usr/share/maven3/bin/mvn /usr/bin/mvn
```

执行命令 `mvn --version` 验证安装：

```
1. Apache Maven 3.2.1 (ea8b2b07643dbb1b84b6d16e1f08391b666bc1e9; 2014-
   02-14T17:37:52+00:00)
2. Maven home: /usr/share/maven3
3. Java version: 1.8.0_151, vendor: Oracle Corporation
4. Java home: /usr/lib/jvm/java-8-oracle/jre
5. Default locale: en_US, platform encoding: UTF-8
6. OS name: "linux", version: "2.6.32-042stab120.16", arch: "amd64",
   family: "unix"
```

安装artifactory

安装artifactory

准备JDK

安装前需要确保JAVA_HOME有正确设置，可以修改/etc/environment，加入JAVA_HOME:

```
1.  JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

最新版本的artifactory会要求至少jdk1.8版本，否则无法启动.如果遇到在安装好jdk8之后再安装jdk7一起安装的情况，需要额外执行一次命令:

```
1.  sudo apt-get install oracle-java8-set-default
```

下载安装

从https://www.jfrog.com/open-source/下载到最新的artifactory，将zip包解压。

> 注： 在*linux*上如果用*wget*命令直接下载会报错*404*，需要用浏览器先下载，得到具体下载地址之后再用*wget*命令下载。

将目录复制到/usr/lib，执行安装：

```
1. sudo mv artifactory*** /usr/lib/
2. cd /usr/lib/
3. sudo mv artifactory*** artifactory
4. cd artifactory/bin
5. sudo ./installService.sh
6.
7. sudo systemctl start artifactory.service
```

安装成功后就可以通过http://localhost:8081访问artifactory的页面了，默认管理员账号和密码为admin/password。

- 参考地址

安装设置

安全起见，登录后先修改admin密码，点击Admin -> Security -> Users -> User List -> admin，修改密码即可。

另外，Admin -> Security -> `General Security Configuration`，取消”Allow Anonymous Access”。

nginx反代

为了方便访问，避免输入8081这样的端口号，考虑使用nginx做反代。

1. 在 `/etc/nginx/sites-available` 目录下创建文件 `maven.dreamfly.io`

```
1.   server {
2.           listen 80;
```

```
3.
4.          server_name maven.dreamfly.io;
5.
6.          location /
7.          {
8.                  proxy_pass http://127.0.0.1:8081;
9.                  proxy_redirect      http://127.0.0.1:8081
    $scheme://maven.dreamfly.io;
10.         }
11.  }
```

2. 在 `/etc/nginx/sites-enabled` 目录下创建文件 `maven.dreamfly.io`

```
1.   sudo ln -s ../sites-available/maven.dreamfly.io
    maven.dreamfly.io
2.   sudo service nginx restart
```

3. 设置 `maven.dreamfly.io` 的域名解析到目标IP地址

设置maven

需要在maven的setttings文件中加入如下的 profile 设置，`sudo vi /usr/share/maven3/conf/settings.xml`：

```
1.  <profile>
2.      <id>dreamfly</id>
3.      <activation>
4.          <activeByDefault>false</activeByDefault>
5.      </activation>
6.      <repositories>
7.          <repository>
8.              <id>dreamfly-release</id>
9.              <name>Dreamfly Release Repository</name>
10.             <releases>
11.                 <enabled>true</enabled>
12.                 <updatePolicy>never</updatePolicy>
```

```
13.                 <checksumPolicy>warn</checksumPolicy>
14.             </releases>
15.             <snapshots>
16.                 <enabled>false</enabled>
17.             </snapshots>
18.             <url>http://maven.dreamfly.io/artifactory/libs-
    release</url>
19.             <layout>default</layout>
20.         </repository>
21.
22.         <repository>
23.             <id>dreamfly-snapshot</id>
24.             <name>Dreamfly Snapshot Repository</name>
25.             <releases>
26.                 <enabled>false</enabled>
27.             </releases>
28.             <snapshots>
29.                 <enabled>true</enabled>
30.                 <updatePolicy>always</updatePolicy>
31.                 <checksumPolicy>warn</checksumPolicy>
32.             </snapshots>
33.             <url>http://maven.dreamfly.io/artifactory/libs-
    snapshot</url>
34.             <layout>default</layout>
35.         </repository>
36.     </repositories>
37.     <pluginRepositories>
38.         <pluginRepository>
39.             <id>dreamfly-plugin-release</id>
40.             <name>Dreamfly Plugin Release Repository</name>
41.             <releases>
42.                 <enabled>true</enabled>
43.                 <updatePolicy>never</updatePolicy>
44.                 <checksumPolicy>warn</checksumPolicy>
45.             </releases>
46.             <snapshots>
47.                 <enabled>false</enabled>
48.             </snapshots>
```

```
            <url>http://maven.dreamfly.io/artifactory/plugins-release</url>
            <layout>default</layout>
        </pluginRepository>

        <pluginRepository>
            <id>dreamfly-snapshot</id>
            <name>Dreamfly Plugin Snapshot Repository</name>
            <releases>
                <enabled>false</enabled>
            </releases>
            <snapshots>
                <enabled>true</enabled>
                <updatePolicy>always</updatePolicy>
                <checksumPolicy>warn</checksumPolicy>
            </snapshots>
            <url>http://maven.dreamfly.io/artifactory/plugins-snapshot</url>
            <layout>default</layout>
        </pluginRepository>
    </pluginRepositories>
</profile>
```

然后在activeProfiles中激活:

```
<activeProfiles>
    <activeProfile>dreamfly</activeProfile>
</activeProfiles>
```

deploy设置

再在 `<servers>` 中加入账号信息，注意id保持一致:

```
<servers>
    ......
    <server>
```

```
        <id>dreamfly</id>
        <username>admin</username>
        <password>dreamfly.io</password>
    </server>
</servers>
```

这样，在deploy时，就可以通过 `-DaltDeploymentRepository` 参数指定发布的目标地址，然后配合设置文件中的账号信息，完成deploy。

```
mvn deploy -DaltDeploymentRepository=dreamfly::default::http://maven.dreamfly.io/snapshot
```

maven 设置文件

maven 设置文件

参考资料

- setting.xml 配置详解

设置中央仓库的镜像

设置中央仓库的镜像

tags:镜像

maven默认的 central 中央仓库，指向的地址是 `repo.maven.apache.org` 。

这个地址经常出现问题，比如不能访问，或者速度超慢（低于1k/s）。因此在国内最好找一个国内的镜像站点来避免直接访问central 中央仓库。

阿里云的镜像

阿里云提供的maven中央仓库。

配置方式：修改maven根目录下的conf文件夹中的setting.xml文件，增加mirror，内容如下：

```
<mirrors>
    ......
    <mirror>
        <id>alimaven</id>
        <name>aliyun maven</name>
        <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
        <mirrorOf>central</mirrorOf>
    </mirror>
```

```
9.  </mirrors>
```

~~开源中国的镜像~~

注： 已经不能再使用.....

插件

插件

基础插件

基础插件

操作

部署

部署

部署artifact到maven仓库。

参考资料

- Maven使用deploy命令部署构建

deploy plugin(翻译)

maven deploy plugin

> 注： 内容翻译自 *Apache Maven Deploy Plugin*

deploy 插件主要用在 deploy 阶段，添加 artifact 到远程仓库来和其他开发者和项目分享。这通常在集成或者发布环境中完成。它也可以用于部署一个特殊的 artifact （例如第三方jar）。

因为仓库包含比 artifact 更多的内容(poms，原数据，MD5和SHA1散列文件…)，部署意味着不仅要复制 artifact，而且要保证所有这些信息正确更新。这是 deploy 插件的责任。

为了工作，部署要求：

- 仓库的信息： 它的地址，访问它的传输方法(FTP，SCP，SFTP…) 和可选的用户特定要求的账户信息
- artifact的信息： group，artifact，version，packaing，classifier…
- 部署者：实际执行部署的方法。可以实现为 wagon transport (使之跨平台)，或者使用系统特定方法

这些信息将来自隐含的pom文件和命令行。settings.xml 文件也可能解析来获取用户认证。

Goal概述

deploy 插件有两个 goal:

- delpoy:deploy 用于自动安装 artifact， 它的 pom 和特别项目产生的附带的 artifact 。即使不是全部也是大多数的和部署相关的信息存储在项目的pom文件中。
- deploy:deploy-file 用于安装单个 artifact 和它的pom。在这种场景下 artifact 信息可以来自可选的指定pom文件，也可以通过使用命令行来完成/覆盖。

使用

如何使用 deploy 插件的一般注意事项可以在 使用页面中找到。更多特别的使用案例在下面给出的例子中有描述。最后但不是不重要，用户偶尔在 插件的wiki页面贡献额外的例子，tips或者勘误。

例子

为了提供对 deploy 插件使用的更好理解， 可以看一下下面的例子:

项目部署:

- 使用FTP部署
- 使用外部SSH部署

Goals

Goals

这个插件的 goal有:

Goal	描述
deploy:deploy	部署artifact到远程仓库
deploy:deploy-file	在远程仓库安装artifact
deploy:help	显示帮助信息

使用

在project的插件配置中指定版本

```
<project>
  ...
  <build>
    <!-- To define the plugin version in your parent POM -->
    <pluginManagement>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-deploy-plugin</artifactId>
          <version>2.8.2</version>
        </plugin>
        ...
      </plugins>
    </pluginManagement>
    <!-- To use the plugin goals in your POM or parent POM -->
    <plugins>
      <plugin>
```

```
18.         <groupId>org.apache.maven.plugins</groupId>
19.         <artifactId>maven-deploy-plugin</artifactId>
20.         <version>2.8.2</version>
21.       </plugin>
22.       ...
23.     </plugins>
24.   </build>
25.   ...
26. </project>
```

使用

使用

TBD:

https://maven.apache.org/plugins/maven-deploy-plugin/usage.html

使用FTP部署

使用FTP部署

为了使用 FTP 部署 artifact，必须首先在POM文件的 distributionManagement 元素中指定使用的 FTP 服务器，并在 build 元素中指定 extension:

```
<project>
  ...
  <distributionManagement>
    <repository>
      <id>ftp-repository</id>
      <url>ftp://repository.mycompany.com/repository</url>
    </repository>
  </distributionManagement>

  <build>
    <extensions>
      <!-- Enabling the use of FTP -->
      <extension>
        <groupId>org.apache.maven.wagon</groupId>
         <artifactId>wagon-ftp</artifactId>
         <version>1.0-beta-6</version>
      </extension>
    </extensions>
  </build>
  ...
</project>
```

settings.xml 文件需要包含一个 server 元素，这个元素的 id 要和上面的POM中指定的 FTP 仓库的id匹配:

```
<settings>
  ...
  <servers>
    <server>
      <id>ftp-repository</id>
      <username>user</username>
      <password>pass</password>
    </server>
  </servers>
  ...
</settings>
```

在使用maven部署之前确保可以手工登录给定的 FTP 服务器。一旦确认所有的事情都正确搭建，可以使用maven来部署artifact:

```
mvn deploy
```

外部SSH部署

在外部SSH命令中部署

为了使用SSH部署artifact，必须首先在pom的 distributionManagement 元素中指定使用的 SSH 服务器，并在 build 元素中指定 extension:

```
<project>
  ...
  <distributionManagement>
    <repository>
      <id>ssh-repository</id>
      <url>scpexe://repository.mycompany.com/repository</url>
    </repository>
  </distributionManagement>

  <build>
    <extensions>
      <!-- Enabling the use of SSH -->
      <extension>
        <groupId>org.apache.maven.wagon</groupId>
         <artifactId>wagon-ssh-external</artifactId>
         <version>1.0-beta-6</version>
      </extension>
    </extensions>
  </build>
  ..
</project>
```

如果从Unix或者有已安装的 Cygwin 中部署，不需要在 settings.xml 中有任何额外的配置，因为所有东西都将从环境中获

取。但是如果在windows上并使用类似plink，将需要下面的设置：

```
<settings>
  ...
  <servers>
    <server>
      <id>ssh-repository</id>
      <username>your username in the remote system if different from local</username>
      <privateKey>/path/to/your/private/key</privateKey> <!-- not needed if using pageant -->
      <configuration>
        <sshExecutable>plink</sshExecutable>
        <scpExecutable>pscp</scpExecutable>
        <sshArgs>other arguments you may need</sshArgs>
      </configuration>
    </server>
  </servers>
  ...
</settings>
```

在使用maven部署之前确保可以手工登录给定的 SSH 服务器。一旦确认所有的事情都正确搭建，可以使用maven来部署artifact：

```
mvn deploy
```

有时可能在部署时有遇到权限问题，如果是这样，可以像这样设置文件和目录权限：

```
<settings>
...
<servers>
 <server>
   <id>ssh-repository</id>
   <!--
    |
```

```
    | Change default file/dir permissions
    |
    -->
   <filePermissions>664</filePermissions>
   <directoryPermissions>775</directoryPermissions>
   <configuration>
     <sshExecutable>plink</sshExecutable>
     <scpExecutable>pscp</scpExecutable>
   </configuration>
 </server>
</servers>
...
</settings>
```

Tips

TIPs

批量更新版本

批量更新版本

执行下面的命令可以批量更新当前项目所有子项目的版本,非常的方便:

```
1.  mvn versions:set -DnewVersion=0.5.0
```

显示更新

显示更新

TBD: 后续再完善，先记录下来。

显示依赖版本更新信息

```
1.  mvn versions:display-dependency-updates
```

显示插件版本更新信息

```
1.  mvn versions:display-plugin-updates
```

发布

发布

发布到中央仓库

发布到中央仓库

注册Sonatype 用户

注册地址:

https://issues.sonatype.org/secure/Signup!default.jspa

切记要记住用户名密码，后面都要用到。

```
email:aoxiaojian@gmail.com
Fullname: Sky Ao
Username: skyao
password: $$$$$$$$$
```

创建Issue

为了发布我们的构件，需要以提交issue的方式创建ticket,地址为:

https://issues.sonatype.org/secure/CreateIssue.jspa?issuetype=21&pid=10134

内容参考:

https://issues.sonatype.org/browse/OSSRH-37470

备注:

1. group id 可以是`io.dreamfly`这种，然后所有的子域名就都可以用
2. 需要提供信息，说明域名是自己所有的

提交之后等待答复，因为时差原因，一般是隔天。

如果审批通过，会有回复，同时issue状态修改为 resolved。

发布准备

生成gpg秘钥

使用 `gpg --gen-key` 命令生成gpg秘钥:

```
gpg --gen-key

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
```

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
       <n>  = key expires in n days
       <n>w = key expires in n weeks
       <n>m = key expires in n months
       <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y
```

然后会要求输入信息：

```
Real name: Sky Ao
Email address:aoxiaojian@gmail.com
Comment:aoxiaojian
```

再要求输入密码 `You need a Passphrase to protect your secret key.` 为了简单起见，设置为和sonatype登录账号一致。

之后有个奇怪的环节：

```
gpg: gpg-agent is not available in this session
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

Not enough random bytes available.  Please do some other work to give
the OS a chance to collect more entropy! (Need 242 more bytes)
```

只好乱敲一顿键盘，哔哔哔。

```
gpg: /home/sky/.gnupg/trustdb.gpg: trustdb created
gpg: key 732796B4 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   2048R/732796B4 2017-11-16
      Key fingerprint = DD69 ###
uid                  Sky Ao$$$
sub   2048R/400AA044 2017-11-16
```

总算生成了，验证一下：

```
gpg --list-keys
/home/sky/.gnupg/pubring.gpg
----------------------------
pub   2048R/732796B4 2017-11-16
uid                  Sky Ao (aoxiaojian) <aoxiaojian@gmail.com>
sub   2048R/400AA044 2017-11-16
```

注意这里的 `732796B4` 后面会用到。

发布gpg公钥

```
gpg --keyserver hkp://pool.sks-keyservers.net --send-keys 732796B4
gpg: sending key 732796B4 to hkp server pool.sks-keyservers.net
```

验证公钥是否发布成功：

```
gpg --keyserver hkp://pool.sks-keyservers.net --recv-keys 732796B4
gpg: requesting key 732796B4 from hkp server pool.sks-keyservers.net
gpgkeys: key 732796B4 not found on keyserver
```

```
gpg: no valid OpenPGP data found.
gpg: Total number processed: 0
gpg: keyserver communications error: key not found
gpg: keyserver communications error: bad public key
gpg: keyserver receive failed: bad public key
```

这里很无耻的失败了，看错误提示是没有找到资料，但是提交时没有报错。反复几次也是如此，最后灵机一动，是不是要翻墙？

```
hp gpg --keyserver hkp://pool.sks-keyservers.net --send-keys 732796B4
gpg: sending key 732796B4 to hkp server pool.sks-keyservers.net

hp gpg --keyserver hkp://pool.sks-keyservers.net --recv-keys 732796B4
gpg: requesting key 732796B4 from hkp server pool.sks-keyservers.net
gpg: key 732796B4: "Sky Ao (aoxiaojian) <aoxiaojian@gmail.com>" not changed
gpg: Total number processed: 1
gpg:              unchanged: 1
```

这次终于对了，走http proxy就可以发布成功，验证也通过了。

修改maven设置

修改maven的全局配置文件settings.xml，增加下面的内容，用户名密码为Sonatype上面注册的用户名和密码：

```
<servers>
    <server>
        <id>oss</id>
        <username>skyao</username>
        <password><![CDATA[ao$$$$$]]></password>
    </server>
</servers>
```

然后修改项目的pom.xml文件，加入需要的信息：

```
<name>Dreamfly Dependencies</name>
<description>Dependency definitions for all dreamfly projects</description>
<url>https://github.com/dreamfly-io/dreamfly-dependencies/</url>
<organization>
    <name>Dreamfly</name>
    <url>http://dreamfly.io</url>
</organization>
<licenses>
    <license>
        <name>Apache License, Version 2.0</name>
        <url>http://www.apache.org/licenses/LICENSE-2.0</url>
    </license>
</licenses>
<developers>
    <developer>
        <name>Sky Ao</name>
        <email>aoxiaojian@gmail.com</email>
    </developer>
</developers>
<scm>
    <connection>scm:git:git@github.com:dreamfly-io/dreamfly-dependencies.git</connection>
    <developerConnection>scm:git:git@github.com:dreamfly-io/dreamfly-dependencies.git</developerConnection>
    <url>git@github.com:dreamfly-io/dreamfly-dependencies.git</url>
</scm>
```

再增加一个profile，名为 `oss` ：

```
<profiles>
    <profile>
        <id>oss</id>
        <build>
            <plugins>
```

```
6.                  <!-- Source -->
7.                  <plugin>
8.                      <groupId>org.apache.maven.plugins</groupId>
9.                      <artifactId>maven-source-plugin</artifactId>
10.                     <version>${maven-source-plugin.version}
    </version>
11.                     <executions>
12.                         <execution>
13.                             <phase>package</phase>
14.                             <goals>
15.                                 <goal>jar-no-fork</goal>
16.                             </goals>
17.                         </execution>
18.                     </executions>
19.                 </plugin>
20.                 <!-- Javadoc -->
21.                 <plugin>
22.                     <groupId>org.apache.maven.plugins</groupId>
23.                     <artifactId>maven-javadoc-plugin</artifactId>
24.                     <version>${maven-javadoc-plugin.version}
    </version>
25.                     <executions>
26.                         <execution>
27.                             <phase>package</phase>
28.                             <goals>
29.                                 <goal>jar</goal>
30.                             </goals>
31.                         </execution>
32.                     </executions>
33.                 </plugin>
34.                 <!-- GPG -->
35.                 <plugin>
36.                     <groupId>org.apache.maven.plugins</groupId>
37.                     <artifactId>maven-gpg-plugin</artifactId>
38.                     <version>${maven-gpg-plugin.version}</version>
39.                     <executions>
40.                         <execution>
41.                             <phase>verify</phase>
```

```
                            <goals>
                                <goal>sign</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </build>
        <distributionManagement>
            <snapshotRepository>
                <id>oss</id>
<url>https://oss.sonatype.org/content/repositories/snapshots/</url>
            </snapshotRepository>
            <repository>
                <id>oss</id>
<url>https://oss.sonatype.org/service/local/staging/deploy/maven2/</u
            </repository>
        </distributionManagement>
    </profile>
</profiles>
```

特别注意：snapshotRepository 与 repository 中的 id 一定要与 setting.xml 中 server 的 id 保持一致。这里我们都设置为oss。

上传构件到OSS

执行下面maven发布命令，注意指定profile为pom.xml里面对应的 `oss` ：

```
mvn clean deploy -P oss
mvn clean deploy -P oss -Darguments="gpg.passphrase=密钥密码"
```

mvn命令执行中，在每个构件上传时，都会要求输入密码，这个密码就是前面gpg加密用的密码：

```
1. Enter passphrase: gpg: gpg-agent is not available in this session
```

看到一下提示信息，说明deploy成功：

```
1. Uploading:
   https://oss.sonatype.org/content/repositories/snapshots/io/openfounda
   dependencies-standard/0.1.0-SNAPSHOT/maven-metadata.xml
2. Uploaded:
   https://oss.sonatype.org/content/repositories/snapshots/io/openfounda
   dependencies-standard/0.1.0-SNAPSHOT/maven-metadata.xml (816 B at
   0.6 KB/sec)
```

此时构件已经部署在oss仓库中，但还不是中央仓库。

发布

在下里面地址登录，用户名和密码就是在sonatype注册的：

https://oss.sonatype.org/

在Staging Repositories找到自己的仓库，可以用模糊搜索。

> 备注：切记是要发布*release*版本的构件，不能是*snapshot*，不然不会出现在*Staging Repositories*里面。

状态为 Open，需要勾选然后点击 Close 按钮。当系统自动验证完成后，状态会变为 Closed。最后，点击 Release 按钮来发布。

第一次发布时，需要回到issue页面回复。然后等待审批。

验证中央仓库

同样上述步骤完成10分钟之后就可以同步到maven中央仓库，可以通过直接浏览仓库的方式做验证，比如访问下列地址：

http://repo1.maven.org/maven2/io/openfoundation/

gpg错误处理

mvn命令执行中，在每个构件上传时，都会要求输入密码，注意错误信息：

```
1. Enter passphrase: gpg: gpg-agent is not available in this session
```

但是在命令行中已经有gpg-agent命令了。gogole之后发现是版本问题：

https://askubuntu.com/questions/860370/gpg-agent-cant-be-reached

首先安装gpg2：

```
1. sudo apt install gpgv2
```

然后在maven的settings.xml中加入两个属性，主要要在激活的profile里面：

```
1. <profile>
2.     <properties>
3.         <gpg.executable>gpg2</gpg.executable>
4.         <gpg.useagent>true</gpg.useagent>
5.     </properties>
6. </profile>
```

再次执行mvn命令：

```
mvn clean deploy -P oss
```

这是就换成gpg2了，但是依然不能直接免密码输入，还是会有一个弹窗要求输入密码，不过总算可以选择保存密码了。参考下文：

- Avoid gpg signing prompt when using Maven release plugin

TBD：还是需要找到更好的办法。

参考资料

参考：https://www.dexcoder.com/selfly/article/4352

Intellij Idea

Intellij Idea

Maven Helper

Idea 官方推荐的，被成为是 “使用 Maven 必备的插件”(A must have plugin for working with Maven)。

Maven Helper 官方地址

介绍

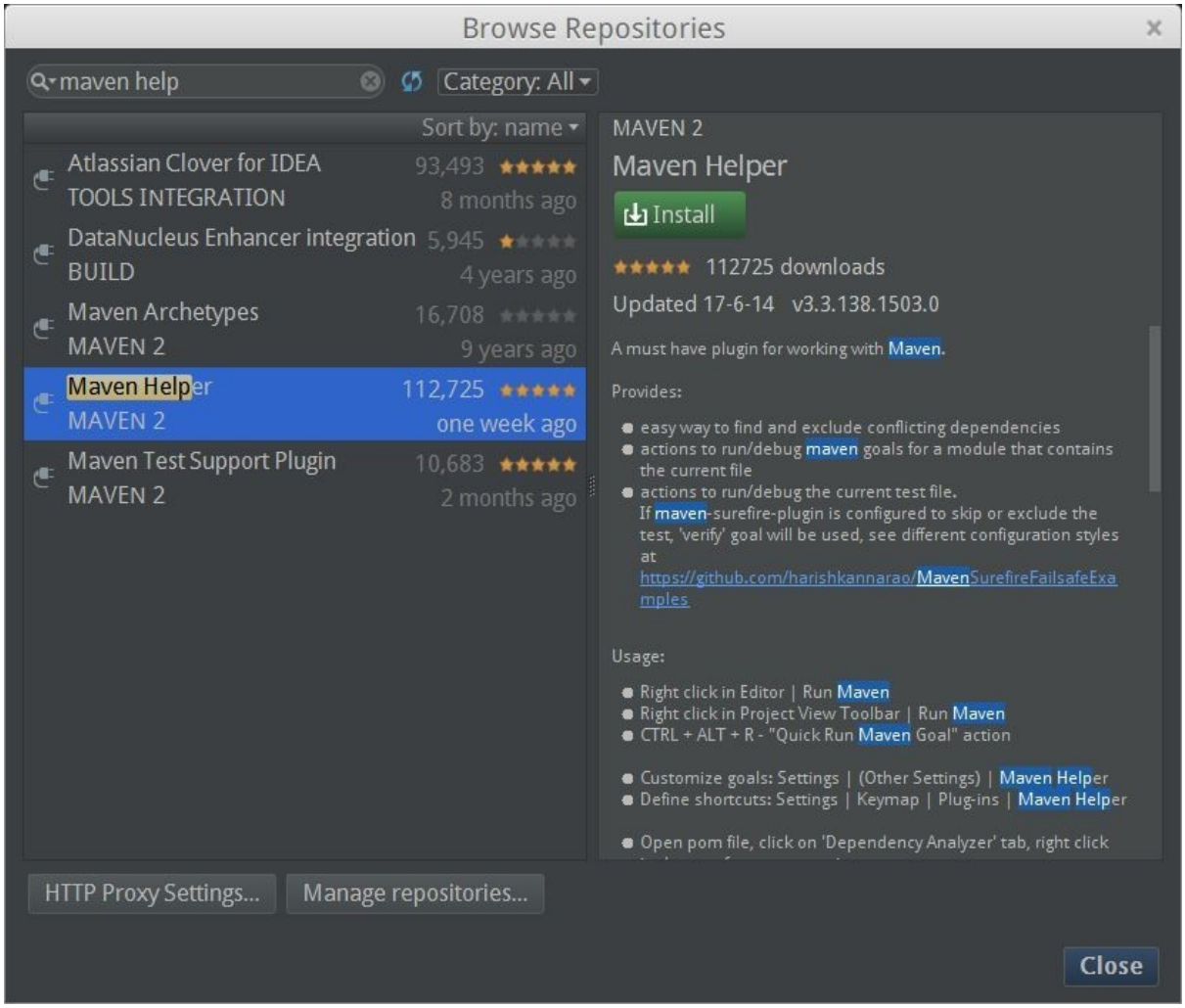
> 翻译自 *Maven Helper首页*

Maven Helper 提供:

- 简单方便的查找和排除依赖冲突
- 用来为包含当前文件的模块运行/调试 maven goal 的操作
- 运行/调试当前测试文件的操作
  - 如果 maven-surefire-plugin 被配置为 skip 或者 exclude 测试，`verify` goal 将被使用。在 https://github.com/harishkannarao/MavenSurefireFailsafeExamples 查看不同的配置风格。

安装

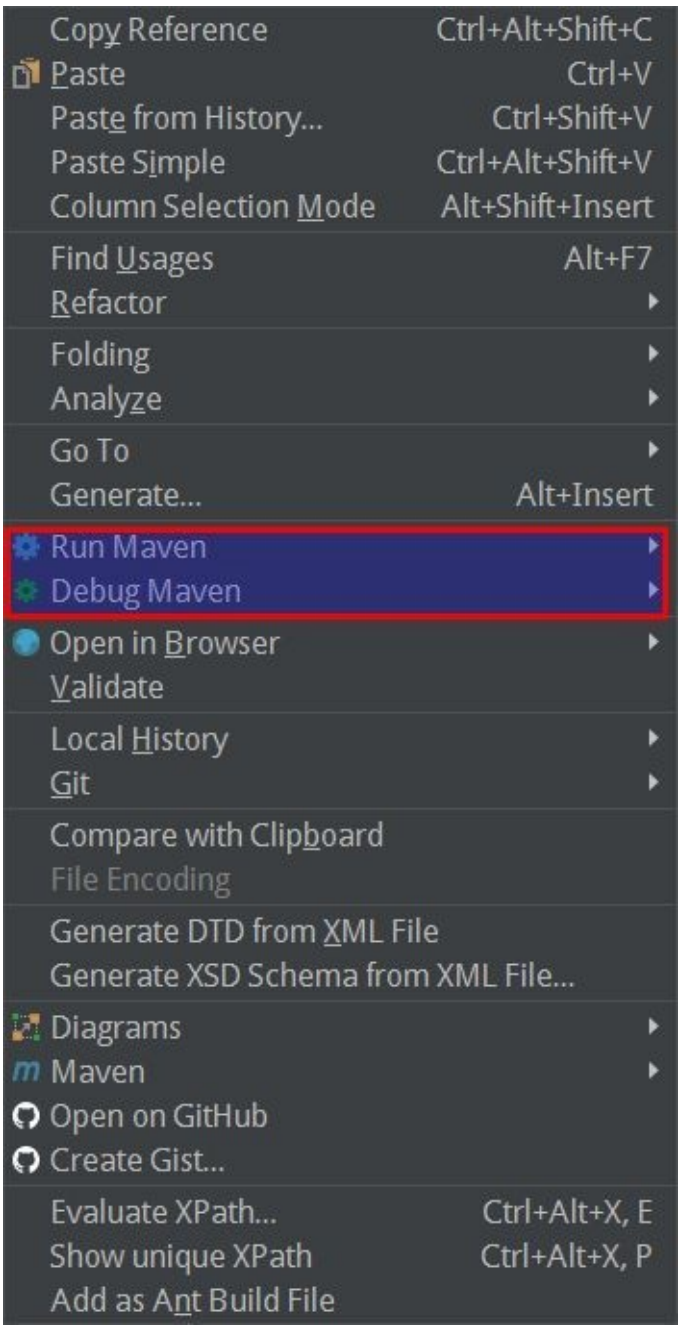
打开 “File” –> “Settings” –> “Browse Repository”, 搜索 “maven helper”:



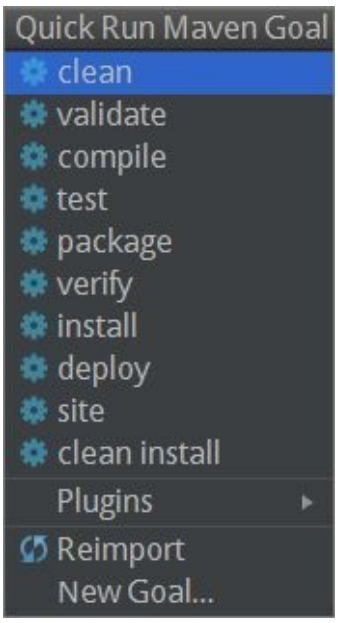
直接安装即可。

使用

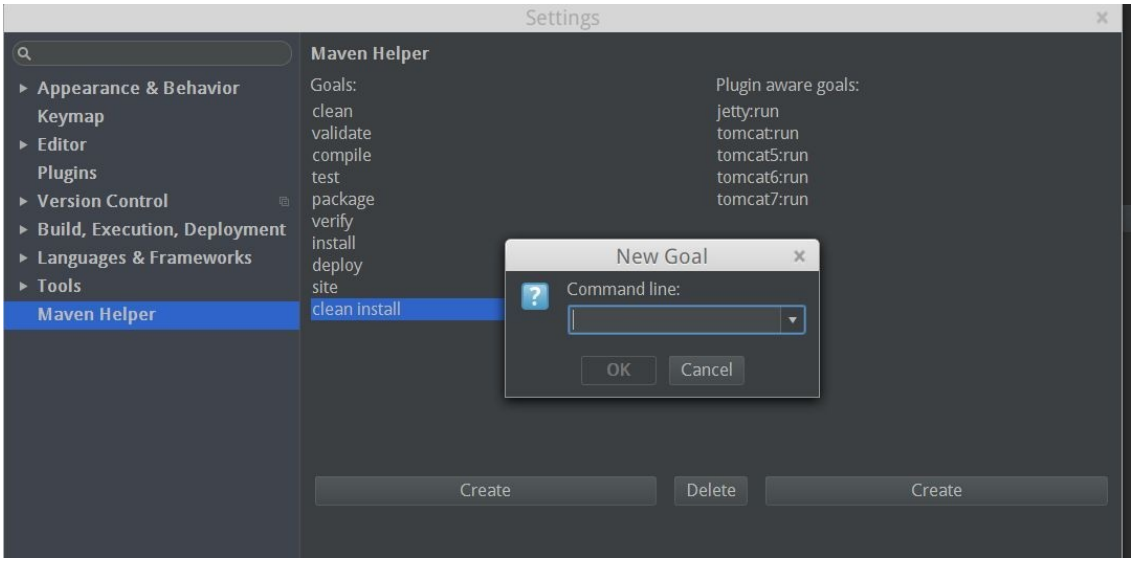
- 在Idea的编辑器中右键 | ``Run Maven
- 在项目（包括子项目）上右键 | ``Run Maven



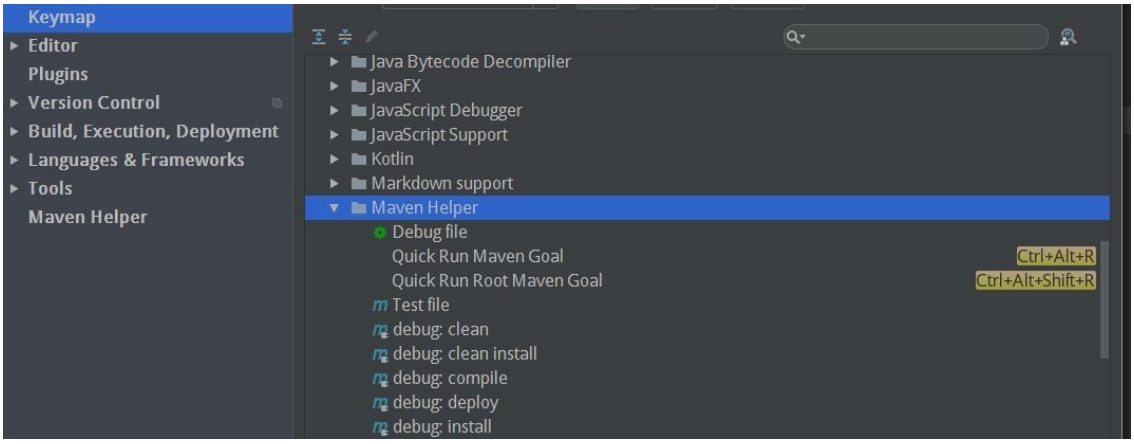
- 使用快捷键 `CTRL + ALT + R`



- 自定义 goals: “File” –> “Settings” –> “Maven Helper”, 点 “create”



- 自定义快捷键: “File” –> “Settings” –> “Keymap” –> “Plug-ins” –> “Maven Helper”



- 打开 pom 文件，点底下的 “Dependency Analyzer” tab，